Bridging the Human-Machine Gap in Applied Machine Learning with Visual Analytics

A dissertation

submitted by

Dylan Cashman, MS, Tufts; Sc. B, Brown

In partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Science

TUFTS UNIVERSITY

August 2020

ADVISOR: Prof. Remco Chang

To Erica, and Leo, and Bryan and Greg and Mom and Dad, and Nana.

Acknowledgments

I'm so grateful for the embarrassment of riches that I've had throughout my educational journey, including an excellent public school education, my Math and Computer Science professors at Brown and at Tufts, and support from my family and friends as I meandered towards this degree at the age of 31. I have to acknowledge just how many shoulders I've stood on to get here, and what a true gift it has been to have the opportunity to pursue my intellectual curiosity.

I've been lucky to receive love and support from my wife, Erica, all throughout graduate school, dating back to one fateful night when we both decided that we were going to go back to get our Master's degrees, many years ago. She's been there for me through rejection and disappointment, and without her words of encouragement, it would have been hard to sustain any idealism through some of the rough patches.

I've had a number of great mentors during internships, including Stephen Kelley and Jordan Crouser at MIT Lincoln Laboratory, Marzieh Nabi at PARC, and Hendrik Strobelt and Adam Perer at IBM Research. Working with them helped me believe that I belonged in this field.

I originally only intended to get a Masters degree, but within a few hours of talking research with my advisor, Remco Chang, I was hooked. Remco has been one of the most important intellectual influences in my life. Through his continual effort and dedication, he has helped me understand not only the visualization landscape, but the value and ingenuity of my own ideas. It's a lesson that will enrich me for the rest of my life and I can't thank him enough.

I also would like to thank the many members, both near and far, of the Visual Analytics Lab at Tufts (VALT), including past members and current. Our group has always felt like a family, and I thank you all for your continuing filial piety.

A highlight of my experience at Tufts has been the ability to cultivate relationships with professors to help understand teaching and to discuss research. In particular, I would like to thank Anselm Blumer and Liping Liu of the machine learning reading group, Rob and Kathy Jacob, and Noah Mendelsohn. In addition, the grad students from the computer science and math departments have always kept me practical and sane. Thank you.

Finally, I would like to thank the members of my committee for their time and effort in guiding me through this process, as well as their mentorship throughout my degree program: Professors Rob Jacob, Michael Hughes, and Xiaozhe Hu of Tufts University, and Professor Michael Gleicher of University of Wisconsin, Madison.

This work has been supported by DARPA through the data driven discovery of models program. The results and findings expressed here are not a representation of the views of DARPA.

Portions of this work have been previously published [CPM+18, CHH+19c, CPCS19]. I thank my co-authors for their permission to include our joint published work in this thesis.

Dylan Cashman

TUFTS UNIVERSITY

August 2020

Bridging the Human-Machine Gap in Applied Machine Learning with Visual Analytics

Dylan Cashman

ADVISOR: Prof. Remco Chang

Machine learning is becoming a ubiquitous toolset for analyzing and making use of large collections of data. Advanced learning algorithms are able to learn from complex data to build models that can tackle artificial intelligence tasks previously thought impossible. As a result, organizations from many domains are attempting to apply machine learning to their data analysis problems. In practice, such efforts can suffer from gaps between the goals of the human and the objective being optimized by the machine, resulting in models that perform poorly in deployment. In this dissertation, I present my thesis that visual analytics systems can improve the performance of deployed models in applied machine learning tasks by allowing the user to compensate for vulnerabilities in learning paradigms. First, I outline how learning paradigms used by machine learning algorithms can miss out on certain aspects of the end goal of the user. Then, I will describe four different visual analytics systems that allow the user to intervene in the learning process across many types of data and models. In these systems, visualizations help users understand how a model performs on different regions of the data. They can also help a user encode their domain expertise to the learning algorithm, to correct for misalignments between the goals of the machine and the needs of the application scenario. This work offers evidence that advanced machine learning algorithms are applied more effectively by involving a domain user in the learning process, using a visual analytics tool as a medium.

Contents

“He could physically feel, in his skin, how things were trending. Computers couldn’t do this. This was the kind of ability that couldn’t be quantified or systematized. . . Deciding which information to extract, and sifting through massive amounts of information to find what is useful, was something only a flesh-and-blood person could do.”

-Haruki Murakami, *1Q84*

Chapter 1

Introduction

In today's world, data is stored in every interaction with a computer. Snapshots are taken for every click or keystroke, for every website visited and every link followed. Cameras follow us in big cities, and our household appliances have computers that log most aspects of our daily lives. Our capabilities to process that data have become incredibly complex, with new data science programs popping up at universities around the world, and large companies bidding against each other for talent to fill new artificial intelligence divisions. New hardware is being developed simply to speed up data processing, and cloud services and supercomputing centers offer potential for massive scale solutions. Machine learning algorithms promise to mine information from datasets and apply it to solve complex artifical intelligence problems. But in the face of all this capability, we must remember that data is only so useful in how it can benefit the end goal of the user. Sometimes there is a broad gap between the end goal of the user and the capabilities of the machine.

I present my thesis that visual analytics systems can improve the performance of deployed models in applied machine learning tasks by allowing the user to compensate for vulnerabilities in machine learning paradigms. Data visualization can enable the viewer to discover insights into data, such as trends, outliers, or comparisons, that are difficult to find in other media such as spreadsheets or databases. These types of insights can prove to be valuable in the discovery, training, and debugging of machine learning models due to the value of ancillary data created by the learning algorithm. This ancillary data includes performance metrics, such as accuracy, as well as the predictions made by the model on a holdout set. By analyzing this data during and after training, a user is able to insert themselves into the modeling process and make use of their domain expertise to improve the model selection process.

Automated methods for data analysis have arisen as a potential solution to applied machine learning. As an example, consider a brief history of computer vision algorithms for image classification. Originally, convolutional filters were handcrafted to capture vertical and horizontal edges, or other local features like fuzzy textures. These handcrafted filters were used to featurize an image, which would then be fed into a machine learning algorithm [Hua96]. Now, these filters are automatically learned using deep neural networks, and no handcrafting is necessary. Many of the most accurate image recognition or language recognition models are made by statisticians and computer scientists, without domain experts like linguists [VDDP18].

There is a general trend towards automation that throws into question whether a human user is needed at all to apply machine learning to a domain problem. The thesis defended in this dissertation aims to offer some existential evidence for the value of the human in the loop. I argue that automated methods fail to meet the needs of many applied machine learning use cases because of basic assumptions embedded in learning algorithms.

In this introduction, I will outline how learning paradigms used by the machine can miss out on certain aspects of the end goal of the user. However, visualizations of training data, internal representations of the machine learning models during training, and the predictions made by those models have the potential to reveal to a user whether the learning algorithm has found an appropriate model. These visualizations can also serve as a medium for a domain expert to augment the learning algorithm's capabilities with their expertise. In the subsequent chapters, I will review the literature around visual analytics for applied machine learning and present four projects which tackle different scenarios in which an automated algorithm will be misaligned with a user's goal in applied machine learning.

1.1 Background on Machine Learning

I begin with some definitions of terms to motivate the discussion. I define a *machine learning model* as a function $f : \mathcal{X} \mapsto \mathcal{Y}$, mapping from the input space $\mathcal{X}$ to the prediction space $\mathcal{Y}$. Machine learning models are typically defined by a set of parameters θ. For example, a simple linear regression model is parameterized by the coefficients of the regression line, while a neural network is parameterized by the weights of the transformations within each layer. A *learning algorithm* is a process in which a machine learning model's parameters are iteratively changed in such a way that the resulting function f_θ performs well at some task. There are many different types of learning algorithms, but in this dissertation I focus only on supervised learning, although much of this treatment could be applied to other types of learning. In supervised learning, a training set $D_{train} = (X_{train}, Y_{train}); X_{train} \subset \mathcal{X}, Y_{train} \subset \mathcal{Y}$ consisting of multiple pairs $(x_i, y_i); x_i \in X_{train}, y_i \in Y_{train}$, is available. The learning algorithm takes in the training set and learns a set of parameters θ that define the resulting model. The learning algorithm chooses the parameters that optimize an objective function, thus finding the model f that performs the best, according to that objective function, on the data provided. The definition outlined above encapsulates a broad problem space, so I'll provide an illustrative example to provide a practical basis for the remainder of the discussion.

Example: Self-driving cars. At the time of the writing of this dissertation, there is a hotly contested race to develop artificial intelligence systems that are able to drive automotive vehicles in a safe and reliable manner. This race began with DARPA's Grand Challenge in the mid-2000s [TMD$^+$06], and continues today with robotics labs at some of the largest companies in the world, such as Uber, General Motors, and Tesla. Towards the goal of fully autonomous vehicles, there exist many individual tasks that can be expressed as function learning and solved using machine learning models. One example is scene recognition. Cars must be able to interpret scenes in front of them and determine whether they need to brake to avoid a collision or if they are able to continue going forward. Consider $\mathcal{X}$ to be the space of possible images taken out of the front-facing camera of a self-driving car, and $\mathcal{Y} = \{openroad, car, redlight, greenlight, \ldots\}$ to be the set of possible scene classifications. A learning algorithm could be used to learn a machine learning model f that maps from images to scene classifications. To train that model, machine learning scientists might gather a list of images by driving a car around manually to get X_{train} and then manually labeling each image to obtain Y_{train}.

1.2 Vulnerabilities of Learning Paradigms

The use of automated tools, such as machine learning algorithms, should be accompanied by assurances that they will behave in an acceptable way. We want to be assured that the self-driving car will brake when it sees a person in the street. But when we select and then train machine learning models, we only have information about how they perform on the data that we have gathered. To engender trust and mitigate risk in applied machine learning, we need to be able to understand the potential for error when a machine learning model is deployed in the wild.

Computational learning theorists attempt to find theoretically sound definitions of concepts found in machine learning, such as training data, validation accuracy, and modelling. These definitions are then used to prove bounds on various metrics like error and runtime. In the early 90s, Vapnik of Bell Laboratories modeled the mitigation of risk in function learning, which we reproduce below [Vap92]. At a high level, we would like to bound the risk of deploying our model, and Vapnik outlines a method for bounding that risk by the performance of the model on a collection of training data.

Assume that we have a *loss function* L that maps from the image under f, $f(X) \subseteq \mathcal{Y}$, to a scalar quantity representing the penalty of our model making mistakes. In the self-driving car example, the model might accrue loss when it incorrectly interprets a green light when the scene actually contains a red light. The risk functional $R(\theta)$ is defined as the expected amount of loss weighted by the joint distribution of $(x, y); x \in \mathcal{X}, y \in \mathcal{Y}$:

$$R(\theta) = \int L(y, f_\theta(x))dP(x, y) \tag{1.1}$$

Finding an optimal model is equivalent to finding the parameters θ that minimize the risk.

$$\theta^* = \operatorname*{argmin}_\theta R(\theta) \tag{1.2}$$

In practice, we can't calculate the true risk $R(\theta)$, because we don't have a closed-form expression for the joint distribution $P(x, y)$. This is particularly true for applied machine learning, where we are modeling complex real-world data. Instead, we try to learn the θ that minimizes the *empirical risk*, that is, the risk that we are able to calculate on a set of training examples.

$$R_{emp}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f_\theta(x_i)) \tag{1.3}$$

$$\hat{\theta} = \operatorname*{argmin}_\theta R_{emp}(\theta) \tag{1.4}$$

The empirical risk is simply the average loss on our training set. We assume that empirical risk and the true risk are *close enough*, so by minimizing the empirical risk, we find a set of parameters $\hat{\theta}$ that is a good approximation of the optimal θ^*. Under certain conditions about the data being trained on, the models being used, and the complexity of the problem at hand, empirical risk minimization can be shown to converge to an approximately correct model, within a reasonable epsilon of error, within a reasonable amount of training time [KVV94, SSBD14]. The logical leap, then, in trusting that our trained machine learning models will behave well in the wild, is believing that the empirical risk that we calculate on our gathered training data in the training process is a fair approximation of the true risk during deployment. However, there are many situations found in real data that break these assumptions. As a result, machine learning models are particularly vulnerable to failure in applied scenarios.

In particular, I highlight the following three vulnerabilities in applied machine learning using empirical risk minimization.

- **V1: Misalignment between Training Data and Real World Data.** Any guarantees on the performance of a machine learning model out in the wild rely on an assumption that the training set, which determines the model's learned parameters, comes from the same distribution of data as the data that model will see during deployment (i.e. the data that we want some guarantees of performance on). If the data seen in deployment is very different, the model could perform poorly or in unexpected ways during deployment. In our self-driving car case, suppose all images gathered for our training set were taken in the pacific northwestern united states, with lots of greenery, clouds, and rain defining the dataset. If the self-driving car were to be deployed in the same environment, we might be reasonably confident it would behave about as well as it did in training. But if the self-driving car were deployed in the middle of the desert, with few clouds, no greenery, and no rain, it would be hard to say whether the car would perform as well as it did during training. Assessing whether the training set is appropriate requires some understanding of the deployment scenario of the model. In the machine learning literature, this is called *domain adaptation* [PY09, SSW15, Mur12].

- **V2: Incorrect Loss Function.** In applied machine learning scenarios, the actual risk that we want to minimize can be hard to describe in a computable way. For example, the risk of deployment of our self-driving car is likely tied to not just the basic task of detecting objects, but the greater goal of getting all passengers to their destination safely and efficiently. It could even be expressed broadly to include the success of the company that produces the car and the potential impact of civil litigation in the event of car accidents. This is dependent on a number of unknown factors including the regulatory conditions and the economic climate. Learning algorithms typically use simple loss functions that calculate simple metrics like classification accuracy or regression mean squared error.

When a simplistic loss function is used for a case with complex risk, there can be negative externalities that result in much higher true risk. For example, transportation researchers have suggested that self-driving cars could actually increase congestion (and therefore emissions, pedestrian risk, and travel time) because there is no cost difference between parking versus “cruising”, or circling a block until they are flagged to pick up a passenger [MB19]. The actual risk of a model is something that is hard to explicitly encode in a learning algorithm, but it likely can be estimated by domain experts knowledgeable about the environment in which the model will be deployed.

- **V3: Non-learnable problems.** In some cases, it won’t be possible to find a good solution to the learning problem and guarantee its performance. As a result, the resulting deployed model will perform poorly at the applied task. As an example, consider a special case of a classic problem of computer science, the XOR problem (See Figure 1.1a). Four different points are sitting on a 2D plane. Our goal is to divide the 2D plane into two regions using a linear boundary such that all filled in points are found in one region, and all hollow points are found in the other. With this particular dataset, it is impossible. The XOR problem is usually brought up to demonstrate why different representations can be needed in machine learning, since it can be solved with a non-linear boundary between the two regions.

  However, the problem points to the difficulties in finding simple solutions to data that doesn’t behave well (in our case, isn’t linearly separable). The more complex boundary we have to learn, the more chance of overfitting or having

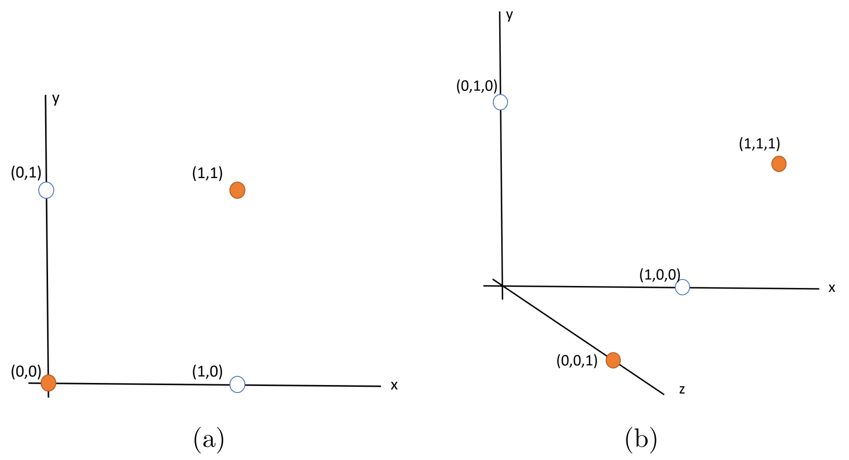


Figure 1.1: A special case of the XOR problem. In (a), a single line can't divide a 2D plane up so that each class is in its own region. With an additional dimension of data, as seen in (b), the XOR problem is now solvable. A single plane can now divide the space up so that each class is in its own region.

poor performance or long training times. And there may be cases where we have constraints on the type of model we allow, i.e. we may be required to use a linear model due to interpretability.

Rather than use a different representation, learnability can sometimes be addressed by finding additional attributes of data. If an additional dimension is added in which the points are more discriminable (See Figure 1.1b), the points can now be separated with a linear boundary (in this case, a plane rather than a line, because we have increased the dimensionality by 1). In a similar manner, some machine learning tasks are impossible if the phenomenon being modeled (such as the linear separability of classes) is not fully present in the training data. But with additional data, they might become solveable.

Each of these vulnerabilities have previously been identified by machine learning scientists, and there exist approaches to mitigate them. For example, for domain mismatch (**V1**), learning algorithms exist that detect and account for drift between training data and testing data [GMCR04]. New loss functions can be added into most machine learning software [PVG$^+$11], and issues with learnability are frequently addressed with novel learning representations such as very deep neural network architectures [HZRS16].

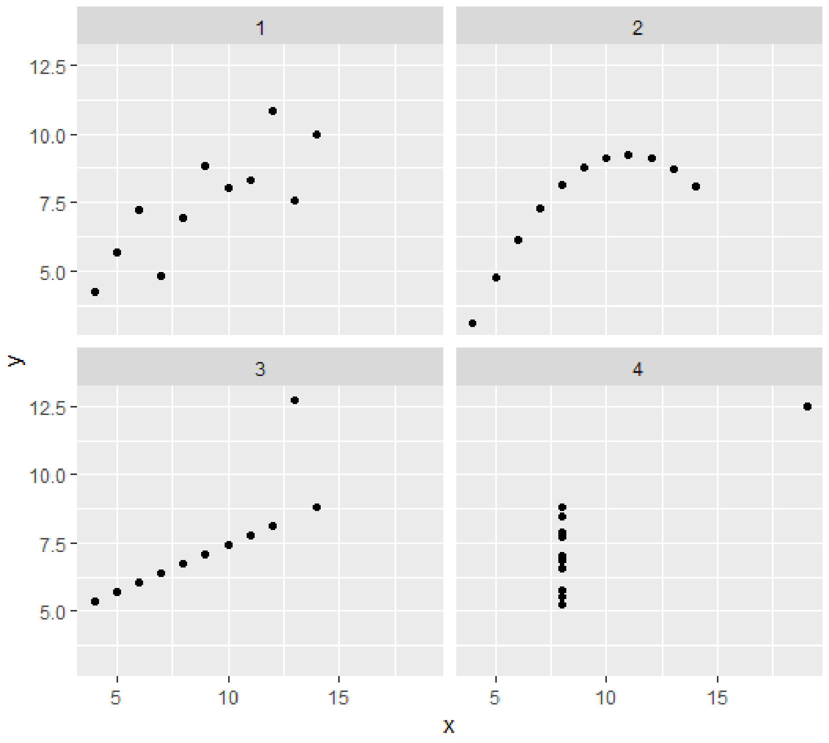
However, these approaches are insufficient in two ways. First, identifying the problem and applying the correct fix requires machine learning expertise, and in many applied machine learning scenarios, a domain expert is applying off-the-shelf learning algorithms and has no access to a machine learning expert. Visual analytics systems can provide a middle ground which have a low enough threshold for non-experts to use, but high enough ceiling of capability to meet the needs of application scenarios. Second, the actual vulnerability may be something that only lives in the head of the domain expert, and it may only emerge once that domain expert has had the opportunity to explore the data and engage with the model and its predictions. For example, a domain user may not realize that the loss function being used by a learning algorithm doesn't capture one of their requirements (**V2**) until they find some problematic errors made by that model. In addition, it has previously been shown that human and machine collaborations can often solve problems that are intractable by machine alone [KFM71]. In fact, we find evidence of this phenomena in applied machine learning in chapter 3.

Fully automated machine learning solutions have been proposed as promising tools for domain scientists to use in applied scenarios. However, they will run into these four problems when applied to real-world problems with messy, real-world datasets and usage scenarios. We posit that visual analytics systems can allow users to compensate for these vulnerabilities in learning algorithms. In this dissertation, I will describe visual analytics tools that address all four vulnerabilities.

1.3 Visualization as a Medium

The four problems outlined above might be addressed by inspection of data that is generated throughout the training process. For example, an expert in the applied domain might be able to identify vulnerability **V1** by looking at the training set in a spreadsheet and determine that it is not a representative sample of data. They may be able to work with a machine learning expert to craft a better learning algorithm that accounts for a more domain-specific loss function (**V2**). They could forage for additional data dimensions to craft a dataset in which the problem becomes learnable (**V3**). But these solutions can be time-consuming and require expertise that is outside of the expert's domain. Data is often too large to manually inspect, and collaborations between domain experts and machine learning experts are not always available.

Visual analytics tools can empower domain experts with augmented control over applied machine learning tasks. While there are many ways that visualization can help with data-centric tasks, we highlight two particular uses for visualization in applied machine learning.



X Mean	9.0
Y Mean	7.5
X Variance	11.0
Y Variance	4.12
Correlation	0.816
Linear Regression Line	y = 3.0 + 0.5x

Figure 1.2: Anscombe's quartet of four datasets with identical summary statistics but very different visual shapes demonstrates the value of visualizing a dataset.

First, visualization has been shown to enable insights that are easily missed by common statistical tools or static views into the data. The canonical example from the literature is Anscombe's quartet (Figure 1.2), a set of four datasets with identical summary statistics but very different shapes when visualized [Ans73]. Visualizations of the data make it clear that the four datasets model different phenomena. However, this fact might have been hidden if the datasets were only observed in a spreadsheet or through statistical metrics. Visualizations designed with gestalt principles in mind can allow for important, complex, and often abstract insights to be made just by viewing the shapes and textures of the sum of the data [Won10]. Visualizations can also deal with scale by aggregating information that is orthogonal to the task at hand to allow a visual overview. Using interactions, visual analytics systems can support multi-scale exploration of data.

Second, visualizations can be used as affordances to allow users to communicate their domain expertise to the underlying learning algorithm. By combining simple, semantically relevant interactions, such as clicking and dragging points together on a canvas or manipulating sliders, with dynamically updating data visualization, visual analytics tools can let a user explore their data, the parameter spaces of the models trained on that data, and the predictions of those models. And by direct manipulation of these controls, the user can implicitly tell the system to modify the learning algorithm. In this manner, they can imbue the learning algorithm with enough domain expertise to address the vulnerabilities outlined above.

Taken together, these two uses comprise a medium, including input and output, between a human and a machine. This medium enables the closing of the gap between what a user desires in applied machine learning (robust and predictable performance with high accuracy), and that which a fully-automated approach can deliver.

1.4 Visual Analytics Tools for Addressing Vulnerabilities

In this dissertation I describe four different visual analytics projects for applied machine learning, each demonstrating the role of visualization in discovering and accounting for vulnerabilities of learning paradigms.

We address misalignments between training data and real world data (**V1**) by letting users visually explore the training data as well as the machine learning model predictions. *Snowcat* was developed as part of the DARPA program *Data Driven Discovery of Models* (d3m), whose stated goal was to develop tools to enable subject matter experts to discover applied machine learning models on par with or better than the models that would be handcrafted by a machine learning expert. In *Snowcat*, users first explore the data to discover relationships between features. An automated machine learning backend trains a set of models on that data, and then users compare the models' performances by looking at individual predictions on a validation set. The user then compares the models and tries to select the one that they believe would perform best on a held out test dataset. Rather than simple comparison by high-level metrics, such as accuracy or mean square error, users visually compare the predictions of models in the context of the input data. In a blind evaluation by the National Institute of Standards and Technology (NIST), our system was the only human in the loop system in the d3m program in which the user chose a model that performed better on held out test data than the model chosen by a fully automated machine learning algorithm. This work was originally presented at the 2019 Eurovis conference and published in the *Computer Graphics Forum* journal in 2019 [CHH+19c].

Visual analytics can serve as a medium for users to implicitly drive the learning algorithm to account for their understanding of the risk of the machine learning model and compensate for the learning algorithm using an incorrect loss function (**V2**). In this dissertation, we show this in two ways.

First, we consider when the loss function involves possibly multiple objectives in which the trade off between objectives is part of the analysis itself. *REMAP* enables a user to drive the search for an architecture for a convolutional neural network. Users must trade off between many objectives, depending on the deployment scenario of the model. They may care about accuracy, or model size, or performance on particular classes, or even particular types of misclassifications. For example, it is particularly important for a self-driving car that a pedestrian is not misclassified as dust particles. By exploring the model space across several projections, and directing the search within regions of that space, users can end up finding models that better match their own notion of risk and loss. This work was presented at the IEEE Conference on Visualization in 2019 and published in the journal of *Transactions on Visualization and Computer Graphics* in 2019 [CPCS19].

Second, we consider the case when the loss function should include more heuristic or qualitative properties that are difficult to encode into a learning algorithm's optimization. For example, it is important that a machine translation model such as English to Chinese is able to remember context such as the gender of the subject of a sentence. These qualitative properties of a model can help a domain user understand the capabilities of their model and determine if it has trained sufficiently. *RNNbow* is a web tool that visualizes gradients used to update the parameters of a recurrent neural network. It can show a user whether an RNN is learning temporal dependencies, a key aspect of any sequential model. It helps a user decide if a model has trained enough and has acquired a long enough memory to be deployed. This work was published in the *Computer Graphics and Applications* journal in 2018, and presented at the IEEE Conference on Visualization in 2019 [CPM+18].

Lastly, visual analytics can be used to enable the discovery and manipulation of additional features by serving as a medium to external information repositories like knowledge graphs. By adding additional features to a training set, users can help a problem become more learnable (**V3**). *Auger* is a tool built for interactive columnar data augmentation. Users can forage for additional data columns by looking through knowledge graphs and executing complex queries over the graph topology. This lets the user shape the training set by adding additional columns that include information that is needed to model the phenomenon of interest. At the time of writing, this work has been conditionally accepted to be presented at the IEEE Conference on Visualization in 2020 and published in the journal of *Transactions on Visualization and Computer Graphics* in 2020 [CXD+20].

1.5 Outline of this Work

In chapter 2, I will review relevant work from the literature. I'll cover many different human-in-the-loop systems for applied machine learning, categorizing them by types of models, types of interactions, and goals of the system.

In chapter 3, I will describe my visual analytics system for general applied machine learning, titled *Snowcat.* I will also present a workflow for visual analytics systems to enable exploratory model analysis, a process in which users can explore the space of potential models that can be trained on a given dataset.

In chapter 4, I demonstrate the use of visualizations to communicate properties of models during training. I present a visual analytics tool for understanding and identifying poor training of recurrent neural network models.

In chapter 5, I present a visual analytics tool for semiautomated neural architecture search. This tool lets users find an architecture that is custom fit for their specific usage scenario.

In chapter 6, I'll describe my work on using visual analytics to facilitate data augmentation to improve the learnability of applied machine learning problems.

Chapter 7 will consist of discussion of several topics. First, I tie the four systems together in terms of risk minimization. Then, I'll compare these systems to efforts from the machine learning (ML) community to build more robust tools. Lastly, I'll discuss a theme for future work, to build abstractions so that the techniques shown in this thesis can be generally applied across all models.

The dissertation will then be concluded in chapter 8.

Chapter 2

Related Work

In this chapter, I review related work from the literature. First, I will give a brief survey of visualization's ability to facilitate a user's ability to gather insights during exploratory data analysis. Then, I give some background on the workflows[1] of visual analytics systems described in the literature to provide context on the role of visualization in the user's insight generation processes. This is relevant because the visual analytics tools described in this dissertation rely on the user's ability to derive insights and understanding by viewing visualizations of data generated during the training process. I also provide a brief survey on research on human-in-the-loop systems for applied machine learning, categorizing them by types of models, types of interactions, and goals of the system.

The systems in chapters 4 and 5 consider applied scenarios in which the user is trying to build recurrent neural networks and convolutional neural networks, respectively, so I offer some review of neural networks and the previous attempts made in visualizing their training and predictions. Chapter 6 describes an effort to allow a domain expert to augment a dataset with additional features to make an applied problem easier to solve, so I also review data augmentation in this chapter.

I review some fully automated techniques for applied machine learning prob-

[1]Frameworks, pipelines, models, and workflows are often used interchangeably in the visualization community to describe abstractions of sequences of task. In this dissertation, I use the word *workflow* to avoid confusion. Further, I use the word *model* to specifically refer to machine learning models and not visualization workflows.

lems next, paying special attention to fully automated techniques for neural architecture search, which is done in a semi-automated way in chapter 5, and data augmentation for model improvement, which is addressed with visual analytics in chapter 6.

2.1 Exploratory Data Analysis

The statistician Tukey developed the term *exploratory data analysis* (EDA) in his work from 1971 through 1977 [Tuk93] and his 1977 book of the same name [Tuk77]. EDA focuses on exploring the underlying data to isolate features and patterns within [HME00]. EDA was considered a departure from standard statistics in that it de-emphasized the methodical approach of posing hypotheses, testing them, and establishing confidence intervals [Chu79]. Tukey's approach tended to favor simple, interpretable conclusions that were frequently presented through visualizations.

A flourishing body of research grew out of the notion that visualization was a critical aspect of making and communicating discoveries during EDA [PS08]. This research resulted in many (static) statistical visualization libraries for use by domain scientists, (such as ggplot [WC$^+$08], plotly [SPH$^+$16], and matplotlib [Hun07]). In more recent years, the visualization community has produced additional open source software (such as D3 [BOH11], Voyager [SMWH17], InfoVis toolkit [Fek04]), commercial visualization systems (such as Tableau [tab], spotfire [spo], Power BI [pow]), and other visualization software designed for specific types of data or domain applications (for some examples, see surveys such as [DOL03, HBO$^+$10]).

2.2 Visual Analytics Workflows

Visual analytics workflows grew out of research into Information Visualization (Infovis). Chi and Riedl [CR98] proposed the *InfoVis reference model* (later refined by Card, Mackinlay and Shneiderman [CMS99]) that emphasizes the mapping of data elements to visual forms. The framework by van Wijk [VW05] extends this with interaction – a user can change the specification of the visualization to focus on a different aspect of the data.

The notion of effective design in Infovis has largely been summarized by Shneiderman's mantra; *Overview, zoom & filter, details-on-demand* [Shn96]. Keim et. al. noted that as data increases in size and complexity, it becomes difficult to follow such a mantra; an overview of a large dataset necessitates some sort of reduction of the data by sampling or, alternatively, an analytical model. The authors provide a framework of Visual Analytics that incorporates analytical models in the visualization pipeline [KKE10]. Wang et al. [WZM+16] extended the *models* phase in the framework by Keim et al. to include a model-building process with: *feature selection and generation*, *model building and selection*, and *model validation.* Chen and Golan [CG16] discuss prototypical workflows that include model building to aid data exploration with various degrees of model integration into the analysis workflows. Sacha et al. formalized the notion of user knowledge generation in visual analytics system, accounting for modeling in the feedback loop of a mixed-initiative system [SSZ+16]. While these frameworks have proven invaluable in guiding the design of countless visual analytics systems, they muddle the delineation between the different goals of including modeling in the visualization process, conflating model building with insight discovery. The *Auger* system described in chapter 6 of this dissertation ties these modern visual analytics workflows to earlier conceptions of *information foraging*, or the process of seeking and gathering information to apply towards a task [PC99, PC05].

The ontology for visual analytics assisted machine learning proposed by Sacha et al. [SKKC19] presents a fairly complete picture of common concepts in visual analytics systems that use machine learning, and offers suggestions of how popular systems in the literature map onto that encoding. In chapter 3, I present a workflow for machine learning model selection. While each step of this workflow can be mapped into the ontology, a key distinction in my workflow is in the *Prepare-Learning* process. Sacha et al. note that "in practice, quite often, the ML *Framework* was determined before the step *Prepare-Data* or even before the raw

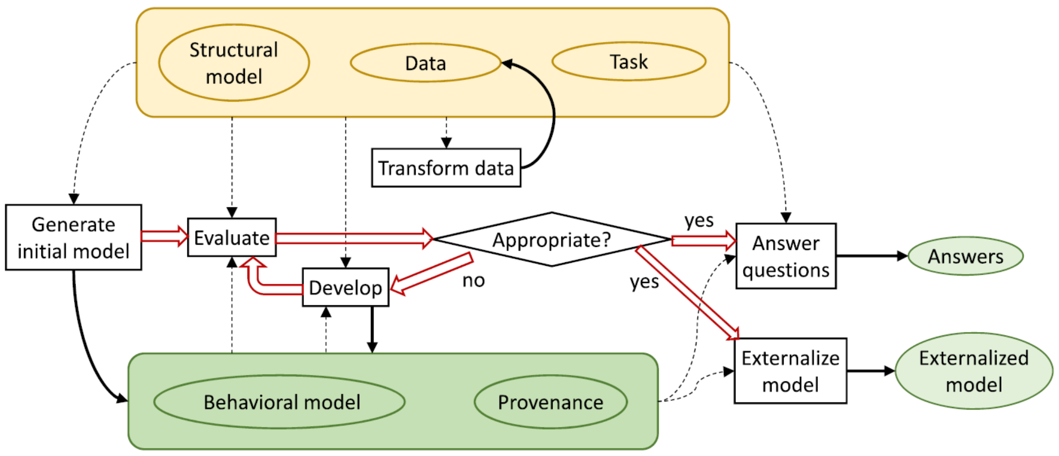


Figure 2.1: The model generation framework of visual analytics by Andrienko et al. [ALA$^+$].

data was captured". In my workflow, this is not the case - the framework, or machine learning modelling problem and its corresponding algorithms are not chosen a priori.

Most similar to the *Snowcat* workflow in chapter 3 of this dissertation is the one recently introduced by Andrienko et al. [ALA$^+$], shown in Figure 2.1, that posits that the outcome of the VA process can either be an "answer" (to a user's analysis question) or an "externalized model". Externalized models can be deployed for a multitude of reasons, including automating an analysis process at scale or for usage in recommender systems. While similar in concept, we propose that the spirit of the workflow by Andrienko et al. is still focused on data exploration (via model generation) which does not adequately distinguish between a data- from a model-focused use case such as the aforementioned financial broker and the quantitative analyst.

2.3 Modeling in Visual Analytics

I summarize several types of support for externalizing models using visual analytics with a similar categorization to that given by Liu et. al. [LWLZ17]. I summarize these efforts into four groups: visual analytics for model **construction and**

steering, **explanation**, **debugging**, and **comparison**.

Model Construction and Steering. A modeling expert frequently tries many different settings when building a model, modifying various hyperparameters in order to maximize some utility function, whether explicitly or implicitly defined. Visual analytics systems can assist domain experts to control the model fitting process by allowing the user to directly manipulate the model's hyperparameters or by inferring the model's hyperparameters through observing and analyzing the user's interactions with the visualization.

Sedlmair et al. [SHB$^+$14] provide a comprehensive survey of visual analytics tools for analyzing the parameter space of models. Example types of models used by these visual analytics tools include regression [MP13], clustering [NHM$^+$07, CD19, KEV$^+$18, SKB$^+$18], classification [VDEvW11, CLKP10], dimension reduction [CLL$^+$13, JZF$^+$09, NM13, AWD12, LWT$^+$15], and domain-specific modeling approaches including climate models [WLSL17]. In these examples, the user directly constructs or modifies the parameters of the model through the interaction of sliders or interactive visual elements within the visualization.

In contrast, other systems support model steering by inferring a user's interactions. Sometimes referred to as semantic interaction [EFN12], these systems allow the user to perform simple, semantically relevant interactions such as clicking and dragging and dynamically adjusts the parameters of the model accordingly. For example, ManiMatrix is an interactive system that allows users to express their preference for where to allot error in a classification task [KLTH10]. By specifying which parts of the confusion matrix they don't want error to appear in, they tell the system to search for a classification model that fits their preferences. Disfunction [BLBC12] allows the user to quickly define a distance metric on a dataset by clicking and dragging data points together or apart to represent similarity. Wekinator enables a user to implicitly specify and steer models for music performance [FTC09]. BEAMES [DCCE19] allows a user to steer multiple models simultaneously by expressing priorities on individual data instances or data features. Heimerl et. al. [HKBE12b] support the task of refining binary classifiers for document retrieval by letting users interactively modify the classifier's decision on any document.

Model Explanation. The ability to extract meaningful explanations of a model's decisionmaking is not only important to the model builder themselves, but to anyone else affected by that model, as required by ethical and legal guidelines such as the European Union's General Data Protection Regulation (GDPR) [Cou18]. This is sometimes called the *explainability* or *interpretability* of a model, and it has shown to be an ill-defined property that is difficult to measure [AGM+18, Lip16, DVK17]. Visual analytics systems attempt to explain models by visualizing their predictions as well as decisions made leading to those predictions. With Squares [RAL+17], analysts can view classification models based on an in-depth analysis of label distribution on a test data set. Krause et. al. allowed for instance-level explanations of triage models based on a patient's medication listed during intake in a hospital emergency room [KDS+17]. Gleicher noted that a simplified class of models could be used in a VA application to trade off some performance in exchange for a more explainable analysis [Gle13]. Many other systems and techniques purport to render various types of models interpretable, including deep learning models [LSC+18b, LSL+17, SGPR18, YCN+15, BJY+18], topic models [WLS+10], word embeddings [HG18], regression models [MP13], classification models [PBD+10, RSG16, ACD+15], and composite models for classification [LXL+18].

Model Debugging. While the calculations used in machine learning models can be excessively complicated, endemic properties of models that cause poor predictions can sometimes be diagnosed visually relatively easily. Seq2Seq-Vis visualizes the five different modules used in sequence-to-sequence neural networks, and provides examples of how errors in all five modules can be diagnosed [SGB+18]. Alsallakh et al. provide several visual analysis tools for debugging classification errors by visualizing the class probability distributions [AHH+14]. Kumpf et al. [KTB+18] provide an interactive analysis method to debug and analyze weather forecast models based on their confidence estimates. These tools allow a model builder to view *how* and *where* their model is breaking, on specified data instances.

Model Comparison. The choice of which model to use from a set of candidate models is highly dependent on the needs of the user and the deployment scenario of a model. Gleicher provides strategies for accommodating comparison with visualization, many of which could be used to compare model outputs [Gle18]. Interactivity can be helpful in comparing multiple models and their predictions on a holdout set of data. Zhang et. al. recently developed Manifold, a framework for interpreting machine learning models that allowed for pairwise comparisons of various models on the same validation data [ZWM$^+$18]. Mühlbacher and Piringer [MP13] support analyzing and comparing regression models based on visualization of feature dependencies and model residuals. TreePOD [MLMP18] helps users balance potentially conflicting objectives such as accuracy and interpretability of decision tree models by facilitating comparison of candidate tree models.

The *Snowcat* system described in chapter 3 of this dissertation is primarily a tool for model comparison, in this categorization. The *RNNbow* system described in chapter 4 can be considered a system for both model explanation, since it visualizes the model's training and inference processes, as well as model debugging. The *REMAP* system described in chapter 5 is a tool for model construction and steering. *Auger*, described in chapter 6, aids in model construction and steering by allowing the user to modify the training set on which a model learns its parameters.

2.4 Artifical Neural Networks

Artificial neural networks (ANNs) are a class of machine learning models that are inspired by the message passing mechanisms found between neurons in brains. ANNs typically learn a sequence of linear transformations that excel at learning good representations for modeling phenomena in data. In recent years, they have most famously been applied to artificial intelligence tasks by learning from large datasets of complex data such as images and text [LB$^+$95, KSH12, LXLZ15]. In this dissertation, chapters 4 and 5 both address issues with training neural networks because they are notoriously difficult to build and train [PMB13].

Prior to the recent explosion in big-data neural networks, artificial neural networks were generally small enough to allow for overview visualizations of all nodes and edges in their computation graph. Early work in visualizing neural network activity focused on "opening the black box" in this complicated computation graph. A good example can be found in Tzeng and Ma's work to display three-layer networks as node-link diagrams, using the size and color of the nodes and edges to encode activation magnitude and uncertainty [TM05]. As the number of layers increases, such visualizations did not scale, and visualizations began to focus on either aggregate views of activations on particular inputs, or by viewing inputs that maximize activation of particular nodes [YCN+15]. Some visualizations of the popular Convolutional Neural Network take advantage of the visual form of the input space, integrating images into overviews of the node activations [LSL+17]. Some visualizations treat neural networks like other similar high-dimensional classifiers, visualizing 2-D projections of their classifications to provide insight into their decision boundaries [KAKC18].

2.4.1 Recurrent Neural Networks

Recurrent Neural Networks are a class of neural networks that are designed to take sequential input and produce sequential output [HS97, Wer90]. Common use cases include financial forecasting, language-to-language machine translation, and clinical diagnoses [SPW98, POF01, LKEW15]. Because of their sequential nature, RNNs proffer an opportunity for more concrete temporal visualizations. In an influential blog post and accompanying publication [KJL15], Karpathy et al. used a variety of visualizations that overlayed some representation of node activation over subsets of the input space to show how different hidden nodes are responsible for different decision logic. There has also been work in interpreting the hidden state dynamics of a trained RNN [SGPR18, MCZ+17]. Their visualizations suggested that activations in the hidden layer contain information about the length of memory in a model.

Most of the tools listed are used after training to attempt to render interpretable the state of a trained network at test time. In contrast, the system described in chapter 4 is used to visualize how the network has learned. Thus, it could be useful to view gradients during training, to know whether hyperparameters need to evolve or if the experiment needs to be rerun with a different set of hyperparameters. Mid-training visualization is one of the features of *TensorBoard*, a visualization tool built on top of Google's *TensorFlow* [AAB+15]. *TensorBoard* allows users to write out values calculated during training to a log, and then generates basic visualizations, such as line graphs and bar charts, of those values throughout training. A typical use case is to plot the loss of a network as a function of the number of batches trained to confirm that the model is improving performance. While it would be possible to log gradients by patching the backpropagation calculation in a *TensorFlow* project, there is minimal support for visualizing those gradients beyond line graphs and bar charts at the time of writing. A recent work by Liu et. al. does visualize the training process for deep generative models [LSC+18a] by plotting the data flow of activations through layers accompanied by basic measures of performance such as accuracy over time.

2.4.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of neural networks that use convolutional layers consisting of small local filters to exploit locality in input data, first popularized by Lecun in the 90s and reinvigorated by the Imagenet competition in the most recent decade [LB+95, KSH12]. They have most famously been applied successfully to image data because of the relevance of spatial locality to semantic understanding of an image. In recent years, interest in neural networks has exploded as they have proven to be state of the art algorithms for image classification [KSH12], text classification [LXLZ15], video classification [KTS+14], image captioning [XBK+15], visual question answering [LYBP16], and a host of other classic artificial intelligence problems.

Visualization has been used in both the machine learning literature and the visual analytics literature for understanding and diagnosing neural networks. In particular, attempts have been made to explain the decision making process of trained networks. Saliency maps [SVZ13] and gradient-based methods [SDV+16] were an early attempt to understand which pixels were most salient to a network's predictions in image classification networks. However, recent work has shown that saliency maps may be dependent only on inherent aspects of the image and not the network's decision making, calling into doubt some of the truthfulness of such methods [AGM+18]. Methods also exist which inspect the effect of individual layers on the decisions of the network [ZF14, YCN+15]. *Lucid* is a library built on the *Tensorflow* machine learning library for generating various visualizations of networks [OSJ+18].

Visual analytics tools extend these techniques by offering interactive environments for users to explore their networks. Some tools allow users to inspect how various components of a trained network contribute to its predictions [LSL+17, SGPR18, WGSY19, KAKC18, WSW+18], while others allow the user to build and train toy models to understand the influence of various hyperparameter choices [SC, KTC+19]. Other tools focus on debugging a network to determine which changes must be made to improve its performance by viewing the activations, gradients, and failure cases of the network [SGB+18, LSC+18b, PHVG+17]. Hohman et al. provide a comprehensive overview of visual analytics for deep learning [HKPC18] .

All of these visual analytics tools presuppose that the user has selected an architecture and wants to inspect, explain, or diagnose it. In contrast, the semi-automated neural architecture search described in chapter 5 allows the user to discover a new architecture. A user of that system might take the discovered architecture and then feed it into a tool such as DeepEyes to more acutely fine tune it for maximal performance [PHVG+17].

2.5 Automated Machine Learning

Automated Machine Learning (AutoML) comprises a set of techniques designed to automate the end-to-end process of ML. It supposes that, with a well-defined task, a desired outcome (metric), and training data, autoML should be able to produce an optimal ML model without further human involvement. To accomplish this, autoML techniques automate a range of ML operations, including but not limited to, data cleaning, data pre-processing, feature engineering, feature selection, algorithm selection and hyperparameter optimization [GBC+15]. The goal of autoML is to hide the complex manipulation of these processes from the user, allowing them to benefit from the use of advanced ml without expertise in statistics or ml.

Since autoML is a relatively recent advancement in the machine learning community, the exact role and function of an autoML system are still undefined. For example, some autoML engines include parts of the machine learning pipeline while others don't (e.g., automated data cleaning [VDR17]) Similarly, the inputs and outputs of an autoML system have not been standardized. While all systems require a formal task specification (including metrics) and training data as input, the exact format of the specification is system-dependent. It is reasonable to expect an autoML VA system to either work closely with their autoML backend developers, as is our case, or to develop their own adapters for existing tools such as autoWeka or CloudML. Further, what autoML should produce as an output is also unclear. Beyond producing an "optimal" model and its performance metrics, some autoML systems may produce the top k models. It could also be valuable for an autoML system to allow further access into the modeling processes, such as the evaluations used to produce the performance metrics (e.g., the details of the internal cross-fold validations used during the model search). This information could be analyzed to provide insight into what parts of the data are hardest to model.

To automatically produce a "best" model, autoML needs to search through a large number of algorithms and their associated hyperparameters, potentially training each discovered model before sampling a new one. Posed as an optimization problem, the goal of autoML is therefore to maximize the user-specified outcome (or metric) quickly and efficiently.

Successful autoML systems attempt to use a better-than-random sampling strategy to iteratively sample from the learning algorithm and hyperparameter spaces. For example, Auto-WEKA [THHLB13, KTH+16] uses Bayesian optimization techniques to iteratively choose from a subset of the available learning algorithms and hyperparameters in the Java-based WEKA machine learning tool [HFH+09]. Hyperopt [BYC13, KBE14] extends a similar technique by parallelizing the search through model space, and interfacing with the Python-based `scikit-learn` [PVG+11] machine learning library. Other autoML tools develop sophisticated prior beliefs on which learning algorithms and hyperparameters work best on which datasets by precomputing their performance on open source datasets [KSS16, SDC+17]. Then, for a new dataset, they can sample from the model space according to where similar datasets performed well.

While the goal of autoML is to hide the technical details of machine learning from the user, in practice these systems are still out of the reach of most non-technical users because the use of autoML requires significant knowledge about machine learning or data science. For example, when performing a classification task, the user needs to specify performance metrics such as area-under-the-curve (AUC), F1 score, uncertainty coefficients, etc. as input to autoML. Similarly, it can be difficult for a user to decide if a model resulting from autoML is good enough and safe enough for their applied deployment because the autoML acts as a black box (**V2**).

2.5.1 Neural Architecture Search

Algorithms for the automated discovery of neural network architectures were proposed as early as the late 1980s using genetic algorithms [MTH89]. Algorithm designers were concerned that neural networks were excessively hard to implement due to their large parameter space and odd reaction to poor parameterizations. In recent years, the success of CNNs and RNNs and their application to many artificial intelligence problems has renewed interest in their use for applied machine learning problems. An increased interest in automated neural architecture searches has followed, resulting in a variety of algorithms using Bayesian optimization [SLA12], network morphisms [JSH18], or reinforcement learning [ZL16, BGNR16]. These algorithms typically define the architecture space so that it is easily searchable by classical parameter space exploration techniques, such as gradient-based optimization [KNS^{+}18, LSY18]. Elsken et al. provide a summary of new research in algorithmic methods in a recent survey [EMH18].

Such methods are driven by an attempt to compete with state of the art performant architectures such as ResNet [HZRS16] or VGGNet [SZ14] that were carefully handcrafted based on years of incremental research in the community. Because performance has been the primary motivator, automated neural architecture search algorithm designers have depended on expensive hardware setups using multiple expensive GPUs and very long search and training times [LSY18]. As a result, the use of these algorithms is out of reach for many potential users without expensive hardware purchases or large outlays to cloud machine learning services. They also are subject to overfitting the training set (**V1**), and can be difficult to fine tune to accommodate a domain expert's understanding of risk (**V2**).

2.6 Data Augmentation

In the machine learning community, the goal of data augmentation is to expand a training dataset so that as much of the phenomenon being modeled is present as possible to try to address problems with learnability (**V3**). This is done in two ways: adding objects (rows) to the training data, and adding new features (columns) for the objects. For example, in image datasets, adding new objects in the form of slightly modified versions of the training objects (i.e. rotating, adding noise, cropping, modifying color) can improve a machine learning model's ability to generalize [KSH12, SK19, PW17, JWC^{+}20, PC20].

Approaches to add columns to a dataset typically differ in where that data comes from, and whether that data is used in training the model or in model inference. Feature engineering [KV15] is a common method to derive new features from existing columns by applying operations to them, such as the difference of two columns. However, it is limited in that it can only express data that is in the scope of existing attributes.

Both automated and semi-automated tools for data augmentation aim to mine large information repositories for useful data that can be included in analysis. This applies to both row-wise and column-wise data augmentation.

2.6.1 Automated Approaches

Automated augmentation of datasets is of interest to the database and the data science communities. In the database community, the goal of data augmentation can manifest in many ways. For example, before augmentation can take place, the first challenge is to find datasets that are suitable for joining. In this scenario, sometimes referred to as a "data lake" [Mil18], the data joining system assumes that there is a finite number of candidate datasets, and the task is to identify which of them can be joined with a user's base dataset. Systems such as Google Goods [HKN+16], Infogather [YGCC12], Octopus [CHK09] Aurum [FAK+18], and work by Sarma et al. [DSFG+12] examine the attributes of the candidate datasets and learn the relationship between those attributes and the attributes of the input data.

Another type of data augmentation appears in the form of entity matching. Entity matching refers to finding the same entity in different datasets, often using machine learning techniques. Once identified, the entity can be removed (for deduplication) or merged (for augmentation). Examples of entity matching systems include Magellan [KDSG+16], Nadeef [DEE+13], Autojoin [ZHC17], and work by Mudgal et al. [MLR+18].

What the data joining and entity matching approaches have in common is that the systems assume little knowledge about the data. The challenge is therefore to identify the commonalities between the datasets (e.g. schema matching) to determine if the datasets or the entities within are related and therefore joinable.

Knowledge graphs have recently been used to incorporate world knowledge into machine learning models [SR17] for a range of models, including text processing [ACD18], image classification [MSG17], and machine translation [MNNBA19], but most work incorporates knowledge graphs into the last step of the machine learning pipeline, inference, to gather facts, rather than augmenting an entire dataset. The Python library RDFFrames [MAG$^+$20] helps extract data from knowledge graphs to improve machine learning training. It allows practitioners to effectively express queries to knowledge graphs and execute them efficiently. However, it requires extensive data science and programming experience to use.

While automated approaches can discover joinable attributes for a given dataset, they lack semantic knowledge of expert users. This may result in a large number of attributes that are irrelevant to the analysis problem being added to the dataset, hindering the users in further analyzing and gaining insights from the dataset. In addition, in common one-to-many relationships, joins require some user guidance for aggregating a collection into a single value. In chapter 6, we overcome these issues by involving a user in the process. Attributes are only added to the dataset if the user decides that they are helpful for their analysis.

2.6.2 Interactive Visual Data Curation

Curating, improving, and augmenting an existing dataset is typically done to aid interpretation and sensemaking [CNE17] during analysis. Sometimes referred to as "data blending", many commercial visualization systems, such as Tableau [var20e], Alteryx [var20a], and D:Swarm [var20b] support it by allowing users to join data tables. The Google Data Studio [var20c] and OpenRefine [var20d], both based on the Webtables project [CHW$^+$08], also support querying web APIs and integrating data from them into a table. In contrast to our approach, none of these tools supports users with discovering new data for augmentation in a systematic way. Users have to find additional data on their own before they can join it to the base dataset.

Other visual interactive approaches allow users to use and integrate data from heterogeneous web sources. This includes helping users extract information from textual web sources [HSV$^+$17], query knowledge graphs [HGVS14], or automatically create visual representations of information stored in knowledge graphs. VAiRoma helps users extract and combine information from Wikipedia articles to create visualizations that provide insight into historical events [CDW+15]. Vispedia [CWT+08] lets users interactively collect and integrate data from Wikipedia tables to create visualizations and answer analysis questions. All of these approaches help users access information stored in different forms, but they don't join to tables of data for the purpose of improving an applied machine learning task.

Tools for data wrangling, such as Wrangler, help users to interactively identify data quality problems, and fix them to improve their datasets [KPHH11]. Data wrangling is an important step towards integrating data from diverse sources [SBI+13]. In this dissertation, we largely leave issues of data wrangling to future research, and focus on the pressing need for the discovery of relevant attributes and their extraction from knowledge graphs. The resulting datasets could then be fed into a tool like Wrangler.

Data improvement and curation tasks are also supported as part of creating and analyzing machine learning models. Some approaches directly allow users to add and modify data by providing labels or corrections for train or test data [HKBE12a, LSD19, MGB+19]. Other systems support identifying mislabeled [RAL+17, BJY+17, SGB+18] or generally low quality instances [MP13, GBYH20] for a range of different model types. In chapter 6, we instead help users improve a dataset by adding relevant high-quality data columns to the entire dataset.

Chapter 3

Exploratory Model Analysis

In this chapter, I describe a visual analytics workflow that enables subject matter experts to discover machine learning models for applied machine learning problems. Users can upload a dataset, explore it through various visualizations, and generate and inspect models that can be trained on that dataset. The end of the workflow consists of selecting a model to export from a list of models generated and trained by a fully automated machine learning algorithm. Rather than letting the machine make the final decision and simply choosing the model that scored the highest on a validation set according to some metric, this workflow presents visualizations of the behaviors of the different models, and lets the user determine which trained model might generalize best.

By exploring both data and model predictions, users can get a sense of which models are learning about the portions of data that they believe to be important in the model's eventual deployment. They can also tell if models are overfitting particular aspects of the distribution of the training set by exploring and filtering the data. This can inform their choice of model, and allow them to implicitly address the domain mismatch vulnerability (**V1**).

While the workflow describes all steps a user would follow to export a model, including initial data exploration, exploration of the problem space, and problem specification, only the final steps in which the user explores and compares the trained models comprise the user's role in model selection that addresses domain mismatch.

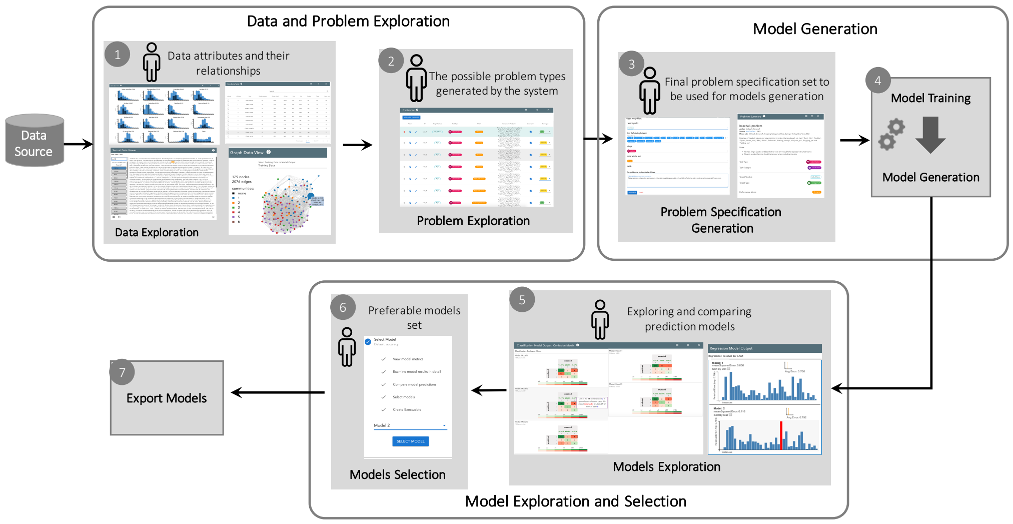

Data and Problem Exploration
Model Generation
Data Source
1
Data attributes and their relationships
Data Exploration
2
The possible problem types generated by the system
Problem Exploration
3
Final problem specification set to be used for models generation
Problem Specification Generation
4
Model Training
Model Generation
5
Exploring and comparing prediction models
Models Exploration
6
Preferable models set
Models Selection
Model Exploration and Selection
7
Export Models

Figure 3.1: The proposed EMA visual analytics workflow for discovery and generation of machine learning models. In **step 1**, the system uses interactive visualizations (such as histograms or graphs) to provide an initial data overview. The system then generates a number of possible modeling problems based on analyzing the data set (**step 2**) from which the user analyzes and selects one to try (**step 3**). Next, (**step 4**) an automated ML system trains and generates candidate models based on the data set and given problem. In **step 5**, the system shows comparisons of the generated prediction models through interactive visualizations of their predictions on a holdout set. Lastly, in **step 6**, users can select a number of preferable models, which are then exported by the system during **step 7** for predictions on unseen test data. At any time, users can return to **step 3** and try different modeling problems on the same dataset.

However, for posterity, we reproduce the entire workflow because the model selection portion was only one requirement of the research project.

To evaluate the workflow, I built a software tool, called *Snowcat*, for allowing subject matter experts to discover machine learning models to predict on a data source. This tool was planned and built with the help of many advisors and fellow researchers[1], and was part of a larger effort in DARPA's data driven discovery of models project. The concepts explored in this chapter were largely influenced by conversations with other performers in this project, and I am indebted to them for their work and discussions.

In section 3.3.3, I present findings from a user study which provide evidence that the final steps of a workflow enable a user to address the domain mismatch vulnerability. Given a dataset and a specific machine learning problem, participants are able to choose a machine learning model that performed better than a model chosen by an automated machine learning algorithm. This shows that exploratory modeling facilitated by the workflow described in this chapter can allow a user to find a better solution to an applied machine learning problem than a machine learning algorithm alone.

3.1 Exploratory Model Analysis

Exploratory data analysis (EDA) has long been recognized as one of the main components of visual analytics [CT05]. EDA is an analysis process through which a user "*searches and analyzes databases to find implicit but potentially useful information*" [KMSZ06], with the use of an interactive visual interface. As described by Tukey, the process of data exploration helps users to escape narrowly assumed properties about their data and allows them to discover patterns and characteristics that were not previously known [Tuk77]. In this sense, the goal of EDA and the use of traditional visual analytics systems is to help the user gain early insight into

[1] While I use first person pronouns to describe work in this chapter, there were many people who worked on this project. A complete list of collaborators in the project is given by the author list of the corresponding publication [CHH$^+$19c]. Significant engineering in addition to my own efforts was done in particular by Shah Rukh Humayoun, Subhajit Das, and Florian Heimerl.

their data [Nor06, CZGR09].

However, in the modern era of big data, machine learning, and AI, visual analytics systems have begun to take on a new role: to help the user in refining *machine learning models*. Systems such as TreePOD [MLMP18], BEAMES [DCCE19], and Seq2SeqVis [SGB+18] propose new visualization and interaction techniques not for a user to better understand their data, but to understand the characteristics of the machine learning models trained on their data and the effects of modifying their parameters and hyperparameters. The goal of these visual analytics systems is to produce a predictive model which will then be used on unseen data.

These systems help analyze and refine a particular type of model with a predefined modeling goal. This limits their ability to support an exploratory analysis process since the user cannot try multiple modeling problems in the same system, and instead are confined to decision trees, regressions, and sequence-to-sequence models, respectively. In this chapter, I consider a previously unsupported scenario in which the *type of model and the modeling task is not known* at the beginning of the analysis. I introduce the term *Exploratory Model Analysis* (EMA), and define it as the process of exploring the set of potential models that can be trained on a given set of data. EMA shares characteristics with EDA in that both describe an analysis process that is open-ended and whose results are not clearly defined a priori, and may change and adapt during the process.

The goal of EMA is twofold: discover variables in the dataset on which reliable predictions can be made, and find the most suitable and robust types of models to predict these variables. There may be multiple models discovered at the end of the process - an analyst may end up discovering regression models between variables a, b, and c, classification models where variables d and e predict the label of variable f, and neural networks that use all independent variables to predict the value of variable g.

Despite the parallels between the two, the analysis processes that EDA and EMA describe are applicable to different sets of analysis scenarios. To illustrate the difference, consider two users of visual analytics systems in a financial services company: a broker, who must be able to explain the current state of the market, in the context of its near present and past, and the quantitative analyst, who must be able to model the future behavior of the market. The broker may use machine learning models to support their exploration of the data, but their ultimate goal is to understand current patterns in the data, so that they can make decisions in the current market landscape. In contrast, the quantitative analyst might be interested in what types of predictions are possible given the data being collected, and beyond that, which types of predictions are robust. Exploratory data analysis might expose some information that is predictive, such as the correlation between features, but for large and complex datasets, complex modeling is needed to make sufficiently robust predictions. The use of my visual analytics workflow can help the quantitative analyst to try different types of models and explore the model space.

In this example, there are two distinctions between these two users: (1) their intended goals, and (2) how data is used in the process. For the broker, the intended outcome of using visual analytics is a decision, a data item (e.g. in an anomaly detection task), or an interesting pattern within the data. The data is therefore the focus of the investigation. On the other hand, for an analyst, the intended outcome is a model (or set of models), its hyperparameters, and properties about its predictions on held out data. The data is used to train and validate the model. It is not in itself the focus of attention.

While there is a plethora of tools and techniques in the visual analytics literature that support using machine learning models, most existing workflows (such as the visual data-exploration workflow by Keim et al. [KAF+08], the knowledge generation model by Sacha et al. [SSS+14], the economic model of visualization by van Wijk [VW05], and four out of the six workflows described by Chen and Golan [CG16]) focus on the exploration and analysis of data, rather than the discovery of the model itself. These workflows presuppose that the user knows what their modeling goal was (e.g. using a regression model to predict the number of hours a patient will use a hospital bed). Although these workflows (and the many visual analytics systems built following these workflows) are effective in helping a user in data exploration tasks, I note that there is often an earlier step of modeling where users do not yet know what types of models can be built from a data source. Model exploration is an important aspect of data analysis that is underrepresented in visual analytics workflows. By exploring many different models and their predictions concurrently, users are able to discover which learning algorithm best matches the properties of the data that they deem important.

The primary contribution of this work is a workflow for EMA that supports model exploration and selection. I first identified a set of functionality and design requirements needed for EMA through a pilot user study. These requirements are then synthesized into a step-by-step workflow (see Figure 3.1) that can be used to implement a system supporting EMA. To validate our proposed workflow for exploratory model analysis, I developed a prototype visual analytics system for EMA and ran a user study with nine data modelers. I report the outcomes of this study and also present two use cases of EMA to demonstrate its applicability and utility. I also report the results of a blind experiment conducted by the National Institute of Standards and Technology (NIST) in which a domain expert using our tool chose a better performing machine learning model than both a fully-automated tool and a machine learning expert.

To summarize, in this chapter we describe the following contributions to the visual analytics community:

- **Definition of exploratory model analysis**: We introduce the notion of exploratory model analysis and propose an initial definition.
- **Workflow for exploratory model analysis**: Based on a pilot study with users, we developed a workflow that supports exploratory model analysis.
- **User studies that validate the efficacy and feasibility of the workflow**: We developed a prototype visual analytics system based on our proposed workflow and evaluated its efficacy with domain expert users. We also present two use cases to illustrate the use of the system.

3.2 A Workflow for Exploratory Model Analysis

The four types of modeling described above all presuppose that the user's modeling task is well-defined: the user of the system already knows what their goal is in using a model. I contend that our workflow solves a problem that is underserved by previous research - Exploratory Model Analysis (EMA). In EMA, the user seeks to discover what modeling can be done on a data source, and hopes to export models that excel at the discovered modeling tasks. Some of the cited works do have some exploratory aspects, including allowing the user to specify which feature in the dataset is the target feature for the resulting predictive model. However, to the best of my knowledge, no existing system allows for multiple modeling types, such as regression and classification, within the same tool.

Beyond the types of modeling outlined above, there are two new requirements that must be accounted for. First, EMA requires an interface for modeling problem specification - the user must be able to explore data and come up with relevant and valid modeling problems. Second, since the type of modeling is not known *a priori*, a common workflow must be distilled from all supported modeling tasks. All of the works cited above are specifically designed towards a certain kind of model, and take advantage of qualities about that model type (i.e. visualizing pruning for decision trees). To support EMA, an application must support model discovery and selection in a general way.

In this section, I describe our method for developing a workflow for EMA. I adopt a user-centric approach that first gathers task requirements for EMA following similar design methodologies by Lloyd and Dykes [LD11] and Brehmer et al. [BISM14]. Specifically, this design methodology calls for first developing a prototype system based on best practices. Feedback by expert users are then gathered and distilled into a set of design or task requirements. The expert users in this feedback study were identified by the National Institute of Standards and Technology (NIST) and were trained in data analysis. Due to confidentiality reasons, I do not report the identities of these individuals.

3.2.1 Prototype System

My goal in this initial feedback study was to distill a common workflow between two different kinds of modeling tasks. Our initial prototype system for supporting exploratory model analysis allowed for only two types of models – classification, in which the model is asked to predict a label for each instance, and regression, in which the model predicts a scalar for each instance. The design of this web-based system consisted of two pages using tabs, where on the first page, a user sees the data overview summary through an interactive histogram view. Each histogram in this view represented an attribute/field in the data set, where the x-axis represented the range of values while the y-axis represented the number of items in each range of values. On the second tab of the application, the system showed a number of resulting predicted models based on the underlying data set. A screenshot of the second tab of this prototype system is shown in Figure 3.2.

Classification models were shown using scatter plots, where each scatter plot showed the model's performance on held out data, projected down to two dimensions. Regression models were visualized using bar charts, where each vertical bar represented the amount of residual and the shape of all the bars represents the model's distribution of error over the held out data.

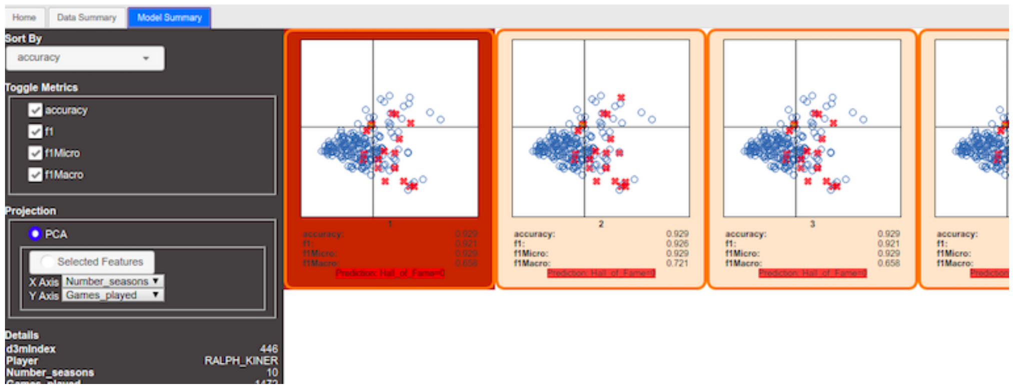


Figure 3.2: A prototype EMA visual analytics system used to determine task requirements. Classification is shown in this figure. During a feedback study with expert users, participants were asked to complete model selection tasks using this view. This process is repeated for regression models (not shown). Feedback from this prototype was used to distill common steps in model exploration and selection across different model types.

3.2.2 Task Requirements

I conducted a feedback study with four participants to gather information on how users discover and select models. The goal of the study was to distill down commonalities between two problem types, classification and regression, in which the task was to export the best predictive model. Each of the four participants used the prototype system to examine two datasets, one for a classification task and the other for regression. Participants were tasked with exporting the best possible predictive model in each case. The participants were instructed to ask questions during the pilot study. Questions as well as think-aloud was recorded for further analysis. After each participant completed their task, they were asked a set of seven open-ended questions relating to the system's workflow, including what system features they might use for more exploratory modeling. The participants' responses were analyzed after the study and distilled into a set of six requirements for exploratory model analysis:

- *G1: Use the data summary to generate prediction models:* Exploration of the dataset was useful for the participants to understand the underlying dataset. This understanding can then be transformed into a well-defined problem specification that can be used to generate the resulting prediction models. Visualization can be useful in providing easy exploration of the data, and cross-linking between different views into the dataset can help facillitate understanding and generate hypotheses about the data.

- *G2: Change and adjust the problem specification to get better prediction models:* Participants were interested in modifying the problem specifications to change the options (e.g., performance metrics such as accuracy, f1-macro, etc. or the target fields) so that they would get more relevant models. The insights generated by visual data exploration can drive the user's refinements of the problem specification.

- *G3: Initially rank the resulting prediction models:* Participants were interested to see the ranking of prediction models based on some criteria, e.g., a performance metric. The ranking should be meaningful to the user, and visualizations of models should help explain the rankings.

- *G4: Determine the most preferable model, beyond the initial rankings:* In many cases, ranking is not enough to make judgment of the superior model. For example, in a classification problem of cancer related data, two models may have the same ranking based on the given accuracy criteria. However, the model with fewer false negative predictions might be preferable. Visualizations can provide an efficient way to communicate the capabilities of different models; even simple visualizations like colored confusion matrices offer much more information than a static metric score.

- *G5: Compare model predictions on individual data points in the context of the input dataset:* Information about the model's predictions, such as their accuracies or their error, were difficult to extrapolate on without the context of individual data instances they predicted upon. Users suggested that having the data overview and the model results on separate tabs of the system made this difficult. Users want to judge model predictions in coordination with exploratory data analysis views. Model explanation techniques such as those linking confusion matrix cells to individual data instances offer a good example of tight linking between the data space and the model space [ZWM+18, ACD+15, RAL+17].

- *G6: Transition seamlessly from one step to another in the overall workflow:* Providing a seamless workflow in the resulting interface helps the user to perform the different tasks required in generating and selecting the relevant models. The system should guide the user in the current task as well as to transition it to the next task without any extra effort. Furthermore, useful default values (such as highly relevant problem specifications or the top ranked predictive model) should be provided for non-expert users so that they can finish at least the default steps in the EMA workflow. Accompanying visualizations that dynamically appear based on the current workflow step can provide easy-to-interpret snapshots of what the system is doing at each step.

It should be noted that the distilled set of requirements does not include participants' comments relating to data cleaning, data augmentation, or post-hoc manual parameters tuning of the selected models. While they are important to the users and relevant to their data analysis needs, these topics are familiar problems in visual analytics systems and are therefore omitted from consideration.

3.2.3 Workflow Design

Based on the six identified task requirements, I propose a workflow as shown in Figure 3.1. The workflow consists of seven steps that are then grouped into three high-level tasks: data and problem exploration, model generation, and model exploration and selection. Below, I detail each step of the workflow.

Step 1 – Data Exploration: In response to **G1**, I identify data exploration as a required first step. Before a user can explore the model space, they must understand the characteristics of the data. Sufficient information needs to be presented so that the user can make an informed decision as to which types of predictions are suitable for the data. Furthermore, the user needs to be able to identify relevant attributes or subsets of data that should be included (or avoided) in the subsequent modeling process.

Step 2 – Problem Exploration: In response to **G1** and **G2**, I also identify the need of generating automatically a valid set of problem specifications. These problem specifications give the user an idea of the space of potential models, and they can use their understanding of the data from Step 1 to choose which are most relevant to them.

Step 3 – Problem Specification Generation: In response to **G2** and **G3**, I identify the need of generating a valid, machine-readable final set of problem specifications after the user explores the dataset and the automated generated problem specifications set. A EMA visual analytic system needs to provide the option to user to refine and select a problem specification from the system generated set or to add a new problem specification. Furthermore, the user should also be able to provide or edit performance metrics (such as accuracy, F1-score, mean squared root, etc.) for each problem specification.

Step 4 – Model Training and Generation: The generated problem specifications will be used to generate a set of trained models. Ideally, the resulting set of models should be diverse. For example, for a classification problem, models should be generated using a variety of classification techniques (e.g. SVM, random forest, kNN, etc.). Since these techniques have different properties and characteristics, casting a wide net will allow the user to better explore the space of possible predictive models in the subsequent EMA process.

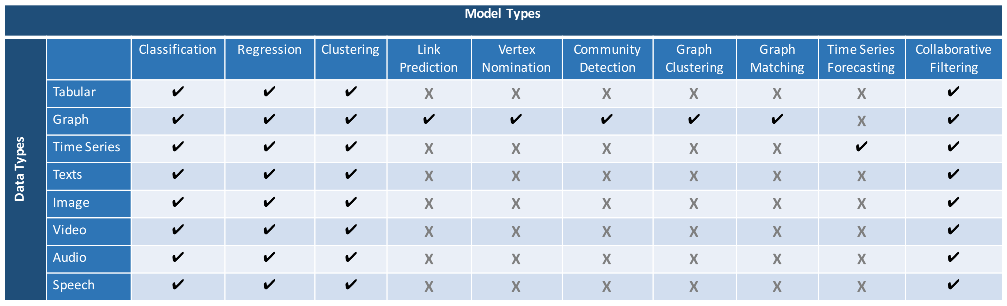
Step 5 – Model Exploration: In response to **G3**, I identify the need of presenting the resulting predictive models in some ranked form (e.g., based on either used performance metric or the time required in generating the model). An EMA visual analytics system needs to present the resulting models through some visualizations, e.g., a confusion matrix for a classification problem type or a residual bar chart for regression problem type (see Fig. 3.1(5)), so that the user can explore predictions of the models and facillitate comparisons between them. I also identify from **G5** that cross-linking between individual data points in a model and data exploration visualization would be useful for the user to better understand the model. It should be noted that a prerequisite for model exploration is to present models in an interpretable encoding, and the available encoding depends on the types of models being explored. Lipton posited that there are two types of model interpretability: transparency, in which a model's internal decision-making can be inspected, and post-hoc interpretability, in which a model is interpreted via its predictions on held out data [Lip16]. In our workflow, because I aim to allow for any type of model, it is difficult to compare wildly different parts of the model space (a kNN model vs. a deep learning model) based on their structure. Instead, I favor a post-hoc approach, where the models are explored via their predictions.

Step 6 – Model Selection: In response to **G4** and **G5**, I identify the need for selecting the user's preferred models based on the model and data exploration. An EMA visual analytics system needs to provide the option to the user to select one or more preferable models in order to export for later usage.

Step 7 – Export Models: In response to **G4**, I also identify that the user also requires to export the selected preferable models so that they can use them for future predictions.

Finally, I identify from the response of **G6** that an EMA visual analytic system needs to make sure that the transition from one workflow step to another one should be seamless. I assume that any implementation of our proposed EMA workflow in Figure 3.1 should supply such smooth transitions so that a non-expert user would also be able to finish the workflow from start to end.

3.3 Iterative System Design and Evaluation



		Model Types									
		Classification	Regression	Clustering	Link Prediction	Vertex Nomination	Community Detection	Graph Clustering	Graph Matching	Time Series Forecasting	Collaborative Filtering
Data Types	Tabular	✔	✔	✔	X	X	X	X	X	X	✔
	Graph	✔	✔	✔	✔	✔	✔	✔	✔	X	✔
	Time Series	✔	✔	✔	X	X	X	X	X	✔	✔
	Texts	✔	✔	✔	X	X	X	X	X	X	✔
	Image	✔	✔	✔	X	X	X	X	X	X	✔
	Video	✔	✔	✔	X	X	X	X	X	X	✔
	Audio	✔	✔	✔	X	X	X	X	X	X	✔
	Speech	✔	✔	✔	X	X	X	X	X	X	✔

Table 3.1: List of all model types and data types supported by our experimental system. A check mark indicates if a model type can be applied to a particular data type, while a cross mark is used to denote incompatible matching between data and model types.

To validate our visual analytics workflow for EMA, I performed two rounds of iterative design and evaluation of the initial prototype system. First, I describe the updated system used in the evaluation. Due to confidentiality concerns, the screenshots shown in this chapter use a set of publicly available datasets[2] that are different from the data examined by the subject matter experts during the two

[2]https://gitlab.com/datadrivendiscovery/tests-data

rounds of evaluations.

3.3.1 Redesigned EMA System

My redesigned system used for the study significantly differs from the original prototype in three important ways. First, it fully supports the workflow as described in the previous section, including Problem Exploration and Problem Specification Generation. Second, the new system supports 10 types of models (compared to the prototype that only supported two). Lastly, in order to accommodate the diverse subject matter experts' needs, our system was expanded to support a range of input data types. Table 3.1 lists all of the supported model and data types of our redesigned system.

From a visual interface design standpoint, the new system also appears differently from the prototype. The key reason for the interface redesign is to provide better guidance to users during the EMA process, and to support larger number of data and model types. We realized during the redesign process that the UI of the original prototype (which used tabs for different steps of the analysis) would not scale to meet the requirements of addressing the seven steps of the EMA workflow.

Figure 3.3 shows screenshots of components of the system that highlight the system's support for guiding the user through the steps of the EMA workflow. The visual interface consists of two parts (see Fig. 3.3, where I provide the overall system layout in the center). The *workflow-panel* (see Fig. 3.3(a)), positioned on the left side of the system layout, shows the current level and status of workflow execution. On the right side of the system layout, the *card-panel* consists of multiple *cards* where each card targets a particular step described in the EMA workflow.

Visualization Support for Data Exploration:

For step one of the workflow, **data exploration**, the system renders several cards providing an overview of the dataset. This includes both a dataset summary card containing any metadata available, such as dataset description and source, as well as cards with interactive visualizations for each data type in the dataset (a few examples are provided in Fig. 3.1(a) and in Fig. 3.3(b)). Currently, the system supports eight

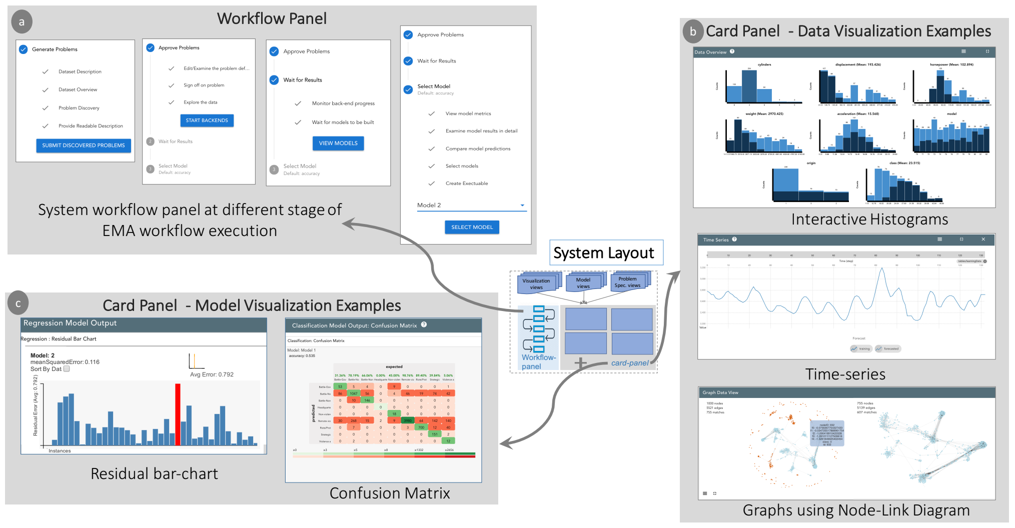

a
Workflow Panel
System workflow panel at different stage of EMA workflow execution
b
Card Panel - Data Visualization Examples
Interactive Histograms
Time-series
Graphs using Node-Link Diagram
System Layout
Visualization views
Model views
Problem Spec. views
Workflow-panel
card-panel
c
Card Panel - Model Visualization Examples
Regression Model Output
Residual bar-chart
Classification Model Output: Confusion Matrix
Confusion Matrix

Figure 3.3: Components of the experimental system. The box in the center shows the system layout, which consists of two parts, the left-side workflow panel and the right-side card panel. (a) shows EMA workflow at different stages in the experimental system, (b) shows three examples of data visualization cards, and (c) shows two examples of model visualization cards.

input data types: tabular, graph, time-series, text, image, video, audio, and speech. Datasets are taken in as CSVs containing tabular data that can point to audio or image files, and rows can contain references to other rows, signifying graph linkages. Data types (e.g., numeric, categorical, temporal, or external references) are explicitly provided - the system does no inference of data types. If a dataset contains multiple types of data, the system a specifically designed card for each type of data. In all cases, the user is also provided a searchable, sortable table showing the raw tabular data. All data views are cross-linked to facilitate insight generation. To limit the scope of the experimental system, our system is not responsible for data cleaning or wrangling, and it assumes that these steps have already been done before the system gets the data.

For example, in the case of only tabular data a set of cross-linked histograms are provided (see Fig. 3.3(b)), empowering the user to explore relationships between features and determine which features are most predictive. Furthermore, a searchable table with raw data fields is also provided. For graph data, node-link diagrams are provided (see Fig. 3.3(b)). Temporal data is displayed through one or more time-series line charts (see Fig. 3.3(b)), according to the number of input time-series. For textual data, the system shows a simple searchable collection of documents to allow the user to search for key terms. Image data is displayed in lists sorted by their labels. Audio and speech files are displayed in a grid-list format with amplitude plots, and each file can also be played in the browser through the interface. Video files are also displayed in a grid-list format, and can be played in the browser through the interface as well. In the case of an input dataset with multiple types of data, such as a social media networks where each row in the table references a single node in a graph, visualizations are provided for both types of data (e.g., histograms for table and node-link diagrams for graphs) and are cross-linked via the interaction mechanisms (i.e., brushing and linking). The exact choices for visual encodings for input dataset types are not contributions of this chapter, and so mostly standard visualizations and encodings were used.

Problem Specification Generation and Exploration:

After data exploration, the user is presented with a list of possible problem specifications depending on the input dataset (step 2, **problem exploration** in the EMA workflow). This set is auto-generated by first choosing each variable in the dataset as the target variable to be predicted, and then generating a problem specification for each machine learning model type that is valid for that target variable. For example, for a categorical target variable, a classification problem specification is generated. For a numeric target variable, specifications are generated for both regression and collaborative filtering. Table 3.1 shows the relationships between an input dataset and the possible corresponding model types supported in our system. The system also generates different problem specifications for each metric valid for the problem type and the type of predicted variable (e.g., accuracy, f1 score, precision). Together, the target prediction variable, the corresponding model type, metrics, and features to be used for predicting the target prediction variable make up a *problem specification.*

The user can select interesting problem specifications from the system-generated list of recommendations, and refine them by removing features as predictors. Users can also decide to generate their own problem descriptions from scratch, in case non of the system-generated suggestions fit the their goals. In either case, the next step of the EMA workflow is for the user to finalize the **problem specifications** (see Fig. 3.1(3)). The resulting set of problem specifications is then used by backend autoML systems to generate the corresponding machine learning models.

Visualization Support for Model Exploration and Selection:

Our system's support for **model generation** (step 4 of the EMA workflow) relies on the use of an automated machine learning (autoML) library, developed under the DARPA D3M program [She]. These autoML systems are accessible through an open source API[3] based on the gRPC protocol[4]. An autoML system requires the user to provide a well-defined problem specification (i.e., the target prediction variable, the model type, the list of features to be used for training the model, the performance metrics) and a set of training data. It then automatically searches through a collection of ML algorithms and their respective hyperparameters, returning the "best" models that fit the user's given problem specification and data. Different autoML libraries such as AutoWeka [THHLB13, KTH$^+$16], Hyperopt [BYC13, KBE14], and Google Cloud AutoML [LL] are in use either commercially or as open source tools. Our system is designed to be compatible with several autoML libraries under the D3M program, including [JSH18, SSW$^+$18]. Note that the sampling of models is entirely driven by the connected autoML systems, and our system does not encode any instructions to the autoML systems beyond the problem specification chosen by the user. However, the backends we connect to generate diverse, complex models, and automatically construct machine learning pipelines including feature extraction and dimensionality reduction steps.

Given the set of problem specifications identified by the user in the previous step, the autoML library automatically generates a list of candidate models. The candidate models are then visualized in an appropriate interpretable representation of their predictions, corresponding to the modeling problem currently being explored

[3]https://gitlab.com/datadrivendiscovery/ta3ta2-api
[4]https://grpc.io/

by the user (step 5, **model exploration**). All types of classification models, including multiclass, binary, and variants on other types of data such as community detection, are displayed to the user as interactive confusion matrices (see Fig. 3.3(c)). Regression models and collaborative filtering models are displayed using sortable interactive bar charts displaying residuals (see Fig. 3.3(c)). Time-series forecasting models are displayed using line charts with dotted lines for predicted points. Cross-linking has been provided between individual data points on these model visualizations and the corresponding attributes in the data exploration visualizations of the input dataset. Furthermore, cross-linking between the models has also been provided to help the user in comparing between the generated models.

Our system initially shows only the highest ranked models produced by the autoML library, as the generated models could be in the hundreds in some cases. This ranking of models is based on the user selected metric in the problem specification.

After a set of suggested models had been generated and returned by the autoML engine, the system provides views to inspect the model's predictions on holdout data. Using this information, they select one or more specific models and request the autoML library to export the selected model(s) (Steps 6 and 7, **model selection** and **export models**).

3.3.2 Evaluation

To evaluate the validity of our proposed EMA workflow and the efficacy of the prototype system, we conducted three rounds of evaluations. Similar to the feedback study, the participants of these rounds of evaluation were also recruited by NIST. In the first two rounds of evaluation, participants tried out the full EMA workflow to discover machine learning problems and export models. Afterward we were able to interview participants after they completed their tasks to get additional qualitative feedback. For the last round of evaluation, a blind evaluation was held explicitly to compare fully automated model selection with the human-in-the-loop approach of the last 3 steps of our workflow. Due to sensitivity of the research, NIST did not allow us access to the participants afterwards, or provide access to the data.

Five subject matter experts participated in the first round of evaluation, four participated in the second, and three in the third. One participant in the second round was unable to complete the task due to connectivity issues. None of the experts participated in both studies (and none of them participated in the previous feedback study). The three groups each used different datasets, in an aim to test out the workflow in differing scenarios.

Method: For the first two experiments, several days prior to each experiment, participants were part of a teleconference in which the functioning of the system was demonstrated on a different dataset than would be used in their evaluation. They were also provided a short training video [CHH+19b] and a user manual [CHH+19a] describing the workflow and individual components of the system they used. Participants of the third experiment were not included in the teleconference, but did get access to the training materials.

For the evaluation, participants were provided with a link to a web interface through which they would do their EMA. They were asked to complete their tasks without asking for help, but were able to consult the training materials at any point in the process. The modeling specifications discovered by users were recorded, as well as any exported models. After completing their tasks, participants were given an open-ended questionnaire about their experience. After the first round of evaluation, some user feedback was incorporated into the experimental system, and the same experiment was held with different users and a different dataset. All changes made to the experimental system were to solve usability issues, in order to more cleanly enable users to follow the workflow presented in this chapter.

Tasks: In all three evaluation studies, participants were provided with a dataset on which they were a subject matter expert. They were allowed to explore the data and complete their tasks at any time within a 24-hour period. The first two evaluation studies started with a task to explore the model space and generate model specifications. The first task was to explore the given dataset and come up with a set of modeling specifications that interested them. All three evaluations did the next

Dataset	Problem Type	Metric	autoML Score	EMA Score (Ours)
News&Events	Classification	Accuracy	0.89707928	**0.90472879**
Sensor Data	Regression	MSE	19.8853693	**9.440946857**

Table 3.2: Results from an experiment comparing a model found by a subject matter expert using the *Snowcat* system verse a model found by a fully automated tool (autoML). Across two datasets, each with a different modeling problem, our system allowed a subject matter expert to discover a better model than the autoML system. Scores reported were average across three participants.

task. The second task supplied them with a problem specification, and asked them to produce a set of candidate models using our system, explore the candidate models and their predictions, and finally choose one or more models to export with their preference ranking. Their ranking was based on which models they believed would perform the best on held out test data. The two tasks taken together encompass the workflow proposed in this chapter. The problem specifications discovered by participants were recorded, as well as the resulting models with rankings exported by the participants.

3.3.3 Findings

Human in the Loop Model Selection First, I report on the quantitative results from the third experiment, because it is most relevant to the domain mismatch vulnerability and discussion of this larger document. In this experiment, subject matter experts were asked to use our system to choose a machine learning model from a set of models produced from an automated machine learning algorithm that they believed would perform the best on held out data. The model chosen by our users was evaluated on held out data, and compared against both a baseline model handcrafted by a machine learning expert and the highest ranked model produced by the automated machine learning algorithm, according to metrics on a validation dataset. The automated machine learning algorithms were developed by other members of the DARPA d3m program. Results can be found in Table 3.2. Across two different datasets, models chosen by the automated tools performed better than the handcrafted baselines. But our subject matter experts were able to select models that performed even better than the models chosen by automated tools.

To interpret these findings, I first note that the baseline our human users were judged against was a high baseline - the automated machine learning backend discovered and selected a model that far outperformed a handcrafted baseline model. Our users selected a model that outperformed that baseline, which is impressive. But it's also important to understand the narrow scope of the choices available to the users to make this selection. The automated machine learning algorithm generates a number of models, and returns them to the user, ranked according to their performance on a validation set. The baseline model, then, is the one that performed the best on the validation set, and according to the treatment of empirical risk minimization given in the introduction of this dissertation, that model should probably be the model that performs best on held out data. But using our system, the participants of the study identified that one of the other models, with worse performance on the validation set, would be a better choice for held out data. Something that they were able to see in our visualizations helped them identify qualities about the model that were not visible to the automated machine learning algorithm.

Efficacy of workflow: Next, I report qualitative results from the first two experiments. All participants were able to develop valid modeling problems and export valid predictive models. Participants provided answers to a survey asking for their top takeaways from using the system to complete their tasks. They were also asked if there were additional features that were missing from the workflow of the system. I report common comments on the workflow, eliding comments pertaining to the specific visual encodings used in the system.

Participants felt that the workflow was successful in allowing them to generate models. One participant noted that the workflow "*...can create multiple models quickly if all (or most data set features are included... [the] overall process of generating to selecting model is generally easy*". Another participant agreed, stating that "*The default workflow containing comparisons of multiple models felt like a good* *conceptual structure to work in.*"

The value of individual stages of the workflow were seen as well: "*The problem discovery phase is well laid out. I can see all the datasets and can quickly scroll through the data*". During this phase, participants appreciated the ability to use the various visualizations in concert with tabular exploration of the data, with one participant stating that "*crosslinking visualizations [between data and model predictions] was a good concept*", and another commenting that the crosslinked tables and visualizations made it "*very easy to remove features, and also simple to select the problem.*"

Suggestions for implementations: Participants were asked what features they thought were most important for completing the task using our workflow. I highlight these so as to provide guidance on the most effective ways to implement our workflow, and also to highlight interesting research questions that grow out of tools supporting EMA.

Our experimental system allowed for participants to select which features to use as predictor features (or independent variables) when specifying modeling problems. This led several participants to desire more sophisticated capabilities for feature generation, to "*create new derivative fields to use as features*".

One participant noted that some of the choices in generating problem specifications were difficult to make without first seeing the resulting models, such as the loss function chosen to optimize the model. The participant suggested that, rather than asking the user to provide whether root mean square error or absolute error is used for a regression task, that the workflow "*have the system combinatorically build models for evaluation (for example, try out all combinations of "metric")*". This suggests that the workflow can be augmented by further automating some tasks. For example, some models could be trained before the user becomes involved, to give the user some idea of where to start in their modeling process.

The end goal of EMA is one or more exported models, and several participants noted that documentation of the exported models is often required. One participant suggested the system could "*export the data to include the visualizations* *in reports*". This suggests that an implementation of our workflow should consider which aspects of provenance it would be feasible to implement, such as those expounded on in work by Ragan et al. [RESC16], in order to meet the needs of data modelers. Another participant noted that further "*understanding of model flaws*" was a requirement, not only for the sake of provenance, but also to aid in the actual model selection. Model understandability is an open topic of research [Gle13], and instance-level tools such as those by Ribeiro et al. [RSG16] and Krause et al. [KDS+17] would seem to be steps in the right direction. Lastly, it was noted that information about how the data was split into training and testing is very relevant to the modeler. Exposing the training/testing split could be critical if there is some property in the data that makes the split important (i.e. if there are seasonal effects in the data).

Limitations of the Workflow: The participants noted that there were some dangers in developing a visual analytics system that enabled exploratory modeling, noting that "*simplified pipelines like those in the task could very easily lead to serious bias or error by an unwary user (e.g. putting together a causally nonsensical model)*". The ethics of building tools that can introduce an untrained audience to new technology is out of the scope of this work, but I do feel the topic is particularly salient in EMA, as the resulting models will likely get deployed in production. I also contend that visual tools, like those supported by our workflow, are preferable to non-visual tools in that the lay user can get a sense of the behavior of models and the training data visually. It could be that additional safeguards should be worked into the workflow to offer a sort of spell-check of the models, similar to how Kindlmann and Scheidegger recommend that visualizations are run through a suite of sanity checks before they are deployed [KS14].

The same participant also noted that streamlining can also limit the ability of the user if they are skilled: "*it doesn't provide sufficient control or introspection... I wanted to add features, customize model structure, etc., but I felt prisoner to a fixed menu of options, as if I was just a spectator*". While some of this can be ameliorated by building a system more angled at the expert user and including more customization options, ultimately the desired capabilities of a system by an expert user may be beyond the ceiling of the workflow I have presented.

3.4 Usage Scenarios

In this section, I offer two examples of how the system might be used for exploratory model analysis. Through these two scenarios I explain the role of the user during each step of the workflow. The first scenario involves the exploration of a sociological dataset of children's perceptions of popularity and the importance of various aspects of their lives. It is used to build predictive models which can then be incorporated into an e-learning tool.The second scenario requires building predictive models of automobile performance for use in prototyping and cost planning.

3.4.1 Analyzing the Popular Kids Dataset

The *Popular Kids*[5] dataset consists of 478 questionnaires of students in grades 4, 5, and 6 about how they perceive importance of various aspects of school in the popularity of their peers. The original study found that among boy respondents, athletic prowess is perceived as most important for popularity, while among girl respondents, appearance is perceived as most important for popularity [CD92].

John works for a large public school district that is trying to determine what data to collect for students on an e-learning platform. Project stakeholders believe that they have some ability to gather data from students in order to personalize their learning plan, but that gathering too much data could lead to disengagement from students. Therefore, John must find what sorts of predictive models can be effective on data that is easy to gather.

John downloads the Popular Kids dataset and loads it into the application. The system shows him linked histograms of the various features of the dataset, as well as a searchable table. He explores the data (**Step 1 in EMA workflow**), noting that prediction of a student's belief in the importance of grades would be

[5]http://tunedit.org/repo/DASL

a valuable prediction for the e-learning platform. He scans the list of generated problems (**Step 2**), selecting a regression problem predicting the belief in grades. He refines the problem (**Step 3**, removing variables in the training set of which school the student was from, since that data would not be relevant in his deployment scenario. He then sends this problem to the autoML backend, which returns a set of models (**Step 4**). The system returns to him a set of regression models (**Step 5**), displayed as bar charts showing residuals on held out data (see Figure 3.5. He notes that none of the regression models have particularly good performance, and in particular, by using cross linking between the regression models and the raw data visualizations, he notes that the resulting models have much more error on girls than on boys.

At this point, John determines that the dataset is not particularly predictive of belief in grades, and decides to search for another predictive modeling problem. He returns to **Step 3** and scans the set of possible problems. He notes that the dataset contains a categorical variable representing the student's goals, with each student marking either *Sports*, *Popular*, or *Grades* as their goal. He chooses a classification problem, predicting student goal, and removes the same variables as before. He submits this new problem and the backend returns a set of models (**Step 4**). The resulting classification models are visualized with a colored confusion matrix, seen in figure 3.4. John compares the different confusion matrices (**Step 5**), and notes that even though model 2 is the best performing model, it performs poorly on two out of the three classes. Instead, he chooses model 3, which performs farily well on all three classes (**Step 6**). He exports the model (**Step 7**), and is able to use it on data gathered by the e-learning platform.

3.4.2 Modeling Automobile Fuel Efficiency

Erica is a data scientist at an automobile company and she would like to develop predictive models that might anticipate the performance or behavior of a car based on potential configurations of independent variables. In particular, she wants to be able to predict how various designs and prototypes of vehicles might affect properties of the car that affect its sales and cost. She hopes to discover a model that can be used to assess new designs and prototypes of vehicles, before they are built.

Erica has access to a dataset containing information about 398 cars (available from OpenML [VvRBT13]), and she would like to build a set of predictive models using different sets of prediction features to determine which features may be most effective at predicting fuel efficiency. She begins by loading the **Data Source** and explores the relationship between attributes in the histogram view (**Step 1**), shown in the top histogram visualization in Figure 3.3(b). By hovering over the bars corresponding to mpg, she determines that the number of cylinders and the class may be good predictors. She then explores the system generated set of problem specifications (**Step 2**). She looked on all the generated problem specifications with "class" as predicting feature. She decides on predicting miles per gallon, and selects a regression task. She selects the provided default values for the rest of the problem specification (**Step 3**).

The ML backend trains on the given dataset and generates six models (**Step 4**). Erica starts to explore the generated regression models, visualized through residual bar charts (**Step 5**). The model visualization in this case gives Erica a sense of how the different models apportion residuals by displaying a bar chart of residuals by instance, sorted by the magnitude of residual (see Fig. 3.5).

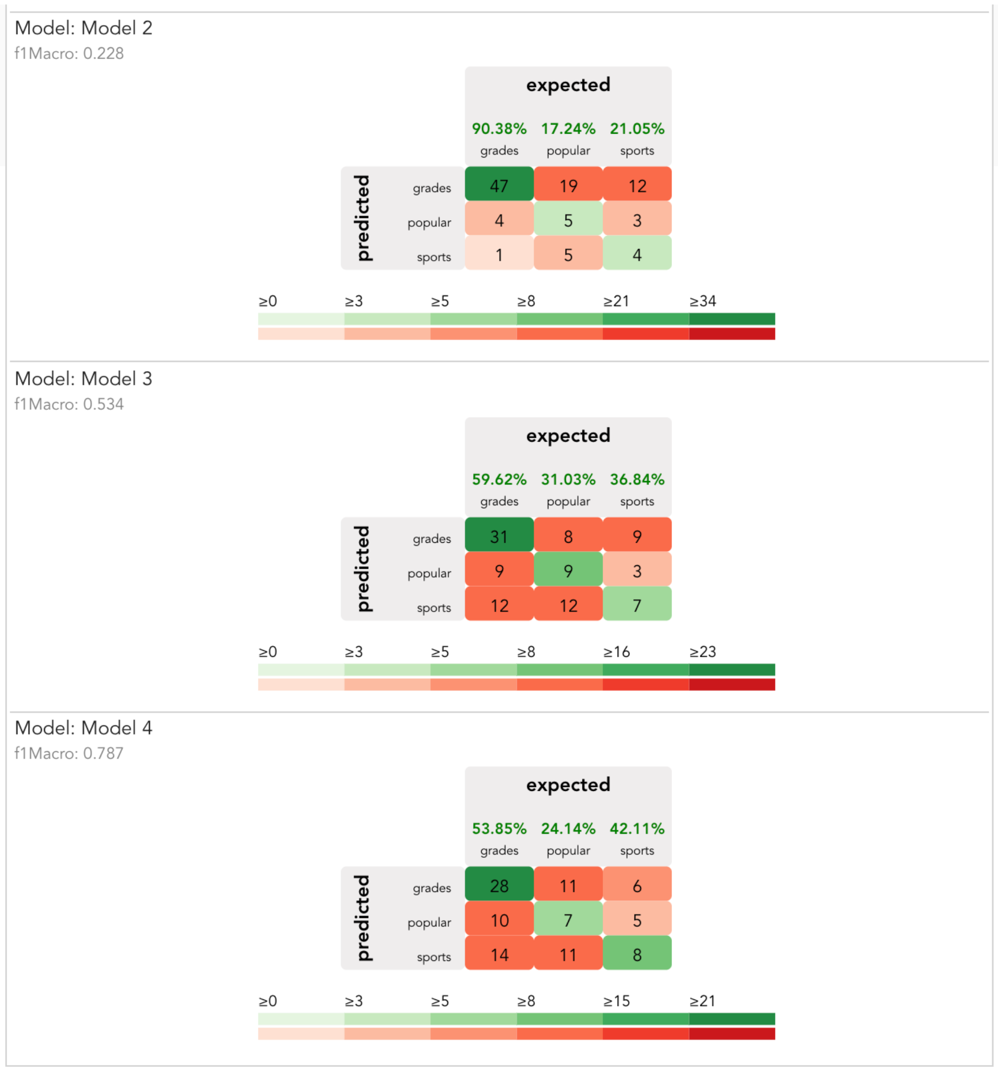
Erica notices that the two best models both have similar scores for mean squared error. She views the residual plots for the two best models, and notes that, while the mean squared error of model 4 is lowest, model 5 apportions residuals more evenly among its instances (see Fig. 3.5). Based on her requirements, it is more important to have a model that gives consistently close predictions, rather than a model that performs well for some examples and poorly for others. Therefore, she selects the model 5 (**Step 6**) to be exported by the system (**Step 7**). By following the proposed EMA workflow, Erica was able to get a better sense of her data, to define a problem, to generate a set of models, and to select the model that she believed would perform best for her task.

3.5 Conclusion

In this chapter, I define the process of exploratory model analysis (EMA), and contribute a visual analytics workflow that supports EMA. I define EMA as the process of discovering and selecting relevant models that can be used to make predictions on a data source. In contrast to many visual analytics tools in the literature, a tool supporting EMA must support problem exploration, problem specification, and model selection in sequence. Our workflow was derived from feedback from a pilot study where participants discovered models on both classification and regression tasks.

To validate our workflow, I built a prototype system and ran user studies where participants were tasked with exploring models on different datasets. Participants found that the steps of the workflow were clear and supported their ability to discover and export complex models on their dataset. Participants also noted distinct manners in which how visual analytics would be of value in implementations of the workflow. Subject matter experts were able to use our system to discover models that performed better on held out data than models chosen by an automated machine learning algorithm. I also present two use cases across two disparate modeling scenarios to demonstrate the steps of the workflow. By presenting a workflow and validating its efficacy, this work lays the groundwork for visual analytics for exploratory model analysis through visual analytics.

In three user studies run by NIST, users were able to explore the data and model spaces and discover models to solve their applied machine learning problems. In particular, users of our system were able to select models that performed better than those models that were selected by automated machine learning algorithms. Snowcat enabled them to do that by facilitating visual exploration of the training data and the strengths and weaknesses of potential models (**V1**).



Model: Model 2

f1Macro: 0.228

predicted \ expected	grades (90.38%)	popular (17.24%)	sports (21.05%)
grades	47	19	12
popular	4	5	3
sports	1	5	4

Model: Model 3

f1Macro: 0.534

predicted \ expected	grades (59.62%)	popular (31.03%)	sports (36.84%)
grades	31	8	9
popular	9	9	3
sports	12	12	7

Model: Model 4

f1Macro: 0.787

predicted \ expected	grades (53.85%)	popular (24.14%)	sports (42.11%)
grades	28	11	6
popular	10	7	5
sports	14	11	8

Figure 3.4: A set of confusion matrices showing the different classification models predicting *Goal* of students in the *Popular Kids* dataset [CD92]. The user is able to see visually that, while the middle model has the highest overall accuracy, it performs much better on students who have high grades as their goal. Instead, the user chooses the bottom model, because it performs more equally on all classes.

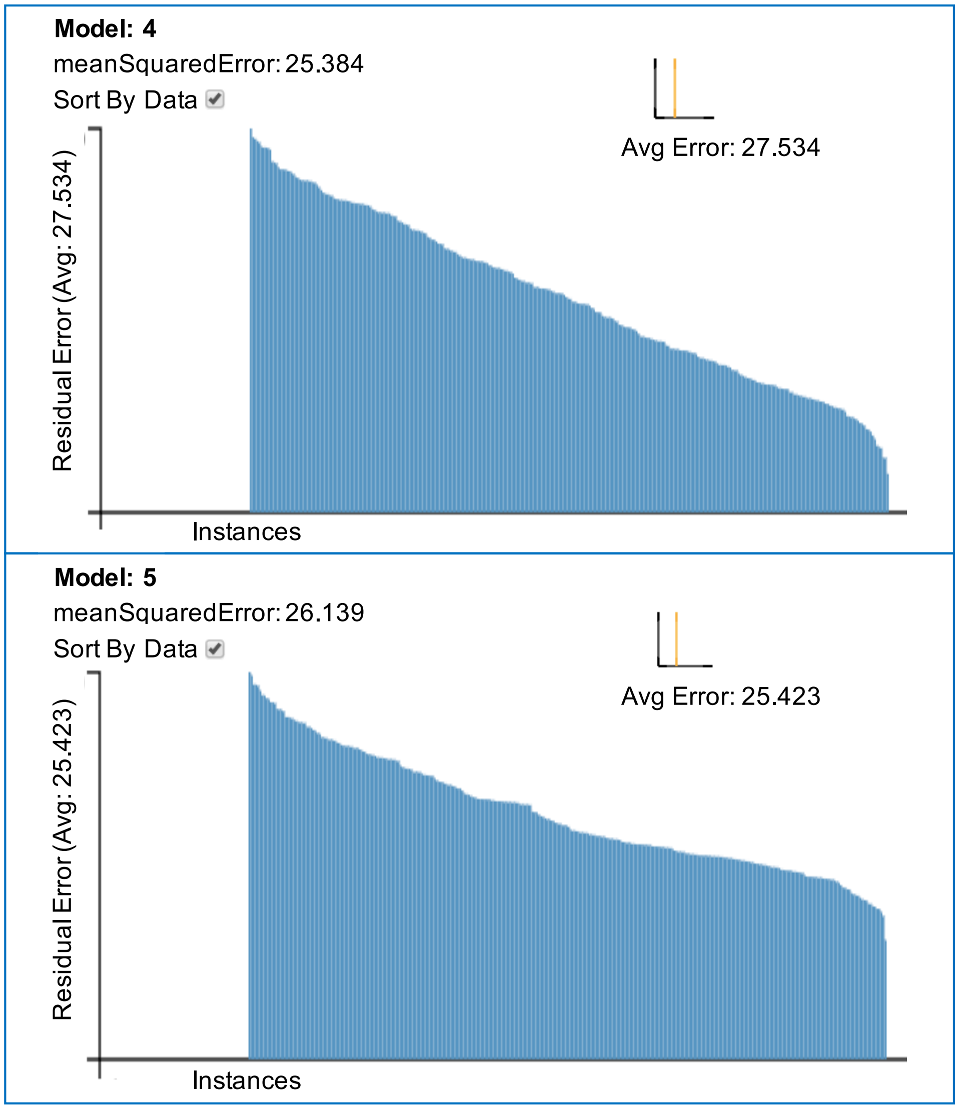


Figure 3.5: Regression plots of the two best models returned by a machine learning backend.

Chapter 4

Monitoring Training of Recurrent Neural Networks

In the previous chapter, I presented evidence that a human in the loop can make use of visualizations of training data and model predictions to choose a better model than a machine alone. In this chapter, I address a different pitfall in learning algorithms. As outlined in the introduction, even if an automated learning algorithm is able to find an optimal model, it may not be aware of all of the qualitative properties that are important for the model builder. In this chapter, I present a visual analytics tool that allows a user to monitor a recurrent neural network during training to assess whether it has learned enough context. Temporal context is a crucial quality of language models, but it may not be encapsulated in the objective function of the recurrent neural network. In addition, these types of models are notorious for their lengthy training times and unusual behavior. The tool described here helps the user make decisions earlier in training about whether to stop or restart the process based on the recurrent neural network's ability to learn temporal dependencies during training.

Artifical Neural Networks (ANNs) have made revolutionary improvements in classification in many domains, with particular attention given to their ability to classify images using convolutional filters [KSH12]. A commonly-cited issue with all ANNs is that they act as a black box, with large numbers of hidden layers each individually learning their own weights resulting in a massive parameter space. Problems of interpretability are compounded by non-linear transformations which obfuscate interactions between each layer. Visualizations of activations within convolutional neural networks have seen some success in illuminating the inner workings of networks to both help understanding and to assist in hyperparameter settings [YCN+15]. However, such activation visualizations are specific to the domain of image processing, and primarily offer insight into how a network is functioning after training. In this chapter, I present RNNbow, a tool for providing insight into the training of Recurrent Neural Networks (RNNs). RNNbow visualizes values calculated during training in order to show the user if their network is learning long-term time dependencies over sequences, a necessary feature in most sequential models. It can be used to uncover problems with poorly parameterized networks early in training. It helps a user know if a model has been trained or if it needs to be scrapped and reparameterized.

A key insight that differentiates this work from other visualizations for deep learning is that it visualizes the *gradients*, not the *activations*. Activations are the responses of the network during inference - when fed an input, what neurons are firing? While this is instructive in comprehending how the network *makes decisions*, it offers little insight into how the network *learns*. Learning in ANNs is typically accomplished via *gradient descent*, a method which minimizes loss over a training set by iteratively updating parameters in the direction dictated by the gradient of that loss. Thus, to analyze how the network is learning (or if it is learning at all), we must inspect the gradients.

RNNs are a particular class of ANNs that map input sequences to output sequences. As with all ANNs, their function depends on what they are fed in as inputs and what they are fed as desired outputs. They can learn to label each item in a sequence if their training data includes labels; a good example of this is training an RNN to do part-of-speech tagging. Alternatively, RNNs can be used to generate sequences that look like the training data. This is accomplished in a technique first

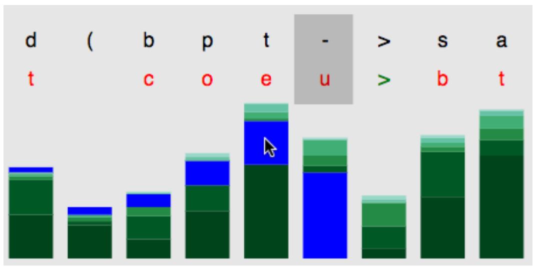


Figure 4.1: RNNbow helps the user see the flow of gradients due to an individual cell's loss during training of a Recurrent Neural Network. Here, we see highlighted in blue the gradient resulting from loss due to predicting the character "u" when the true character was "-".

proposed by Elman [Elm90] in which, for a given training set $s_1 \dots s_n$ the input sequence is set to $s_1 \dots s_{n-1}$ and the output sequence is set to $s_2 \dots s_n$. In this way, the RNN learns to predict the succeeding element of a sequence. RNNs are behind some of Deep Learning's most astonishing results, including language translation, generating image captions, and predicting medical outcomes.

RNNs have been called both "unreasonably effective" [Kar15] and "difficult to train" [PMB13]. One of the major issues with training RNNs is ensuring that the gradient descent updates propagate far enough back that long-term dependencies can be learned. Consider an RNN that tried to produce the following sentence.

i-1	**i**	i+1	i+2	i+3	i+4	**i+5**	i+6
The	**man**	bought	a	toy	for	**his**	dog.

In order for the RNN to be able to know the gender of the pronoun **his**, it must remember the gendered noun **man** 5 time steps earlier. Since RNNs learn via gradient descent, the only way to learn time dependencies of that distance is to have the gradient due to the loss incurred by an errant prediction propagate back to update the parameters that controlled how much the model learned at an earlier step. In other words, if the word **his** at $t = i + 5$ is a function of the word **man** at $t = i$, then the gradient at $t = i$ with respect to the loss at $t = i + 5$ must be greater than 0.

If an RNN is parameterized poorly, it may fall victim to the well-studied *vanishing gradient* problem [PMB13], in which gradient only flows a few cells back, at which point the network may be no more capable than using frequency counts over the training set. To try to address this problem, the user must not only select numerical parameters like the size of the hidden layer or the number of layers, but also must choose between different architectures (stacks or grids) and different RNN cell types such as Long Short-Term Memory cells (LSTMs) or Gated Recurrent Units (GRUs). The theoretical distinctions involved in making these choices can be mystifying to many users of RNNs, and it can be confusing to try to diagnose learning issues resulting from poor RNN design. Tools are needed to reveal endemic issues in gradient flow in RNNs so that the user has early evidence of whether their network architecture is able to learn or not.

RNNbow is a tool to visualize the gradient flow during training of an RNN. It provides an overview of the magnitude of the gradient updates thorughout training, showing the user how quickly a model is learning, and how the regime of parameter updates changes over the course of training. It also allows users to drill down into a particular batch and view the individual influence of each of the RNN's predictions on on parameter updates in the hidden layer. By breaking down the gradient update at each cell by each component's origin, it makes the vanishing gradient apparent. It helps users assess their parameterization of their network during training. It also provides an illustration of the change in gradient behavior as a network trains. In the case of the earlier example, RNNbow can help the user detect if the loss incurred during the update of the word **him** is successfully propagated 5 steps back to the word **man**. At the time of writing, it is the only neural network visualization that visualizes gradient flow in RNNs that the author is aware of.

Further, because RNNbow visualizes the gradient and not the input space, the use of the tool is agnostic to the domain of the problem. In contrast to many of the prevalent ANN visualizations that focus on convolutional neural networks that operate on images [LSL+17, YCN+15], RNNbow could be used to visualize the gradient of any RNN. In this chapter I use a character-level RNN as a demonstration, but RNNbow could be applied to show learning of other sequential data, including video frames and words.

For a use case, I repeat a well-known RNN experiment [KJL15] to learn and generate statements in the C programming language via a character-level RNN. I present some insights that can be gleaned via RNNbow. I explain how traditional implementations of backpropagation can be modified to collect the itemized gradients visualized by RNNbow, and discuss complexity implications. I discuss the advantages of visualizing gradient over activation, discuss the role of visual analytics in deep learning, and conclude by considering future work in using RNNbow to compare different architectures.

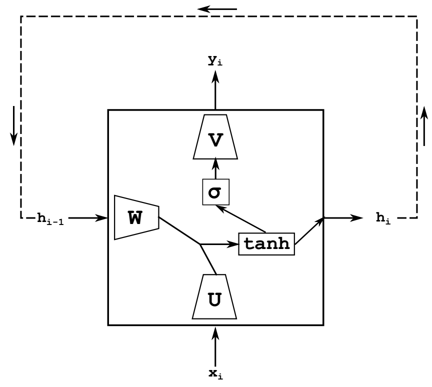


Figure 4.2: A simple one-cell recurrent neural network, seen as a cyclic computation graph. Trapezoids are linear transformations by a weight matrix. Rectangles are element-wise scalar functions. The RNN produces an output y_i for every input x_i, passing on the calculated hidden state h_i back to itself to use for the next input.

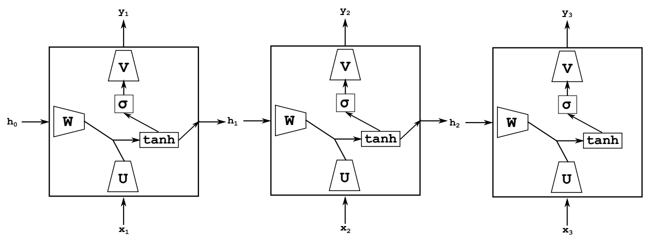


Figure 4.3: To make inference over a sequence of inputs, the RNN from Fig 4.2 is unrolled once for each element in the input sequence. This RNN is unrolled three times to make three cells. It takes three inputs, and produces three outputs. The weight matrices U, W, and V are shared in each cell.

4.1 Recurrent Neural Networks

The goal of an RNN is to produce output given sequence input. Their advantage over other sequential learners such as Markov chains or Maximum Entropy Classifiers is that they are able to learn long-term dependencies via nonlinear dynamics in their hidden layer. The basic RNN architecture can be viewed as a graph with cycles, as seen in Figure 4.2. At any given point in inference, the input x_i and the previous hidden state h_{i-1} are used to calculate the new hidden state h_i, which is then used to calculate the emission y_i. Mathematically, this can be described as:

$$h_i = \tanh\left(Wh_{i-1} + Ux_i\right) \tag{4.1}$$

$$y_i = \sigma(Vh_i) \tag{4.2}$$

Here, W, U, and V are weight matrices, and σ is the sigmoid function $\sigma(x) = \frac{1}{1-e^{-x}}$. Both tanh and σ are common *activation functions* in the deep learning literature. Intuitively, the weight matrices perform a linear transformation on the data, and then the activation functions squash the values back to an interpretable, normalized range, with tanh bounded by $(-1, 1)$, and σ bounded by $(0, 1)$. In addition, these activation functions add a nonlinearity into computation so that the RNN can fit more than polynomial functions.

During training, the training data set is partitioned into regular *batches*. An RNN trains on one batch at a time, in sequential order, by unrolling for a number of steps equal to the size of the batch. A batch size of 3 is seen in Figure 4.3; however, a typical batch size might range from a dozen elements to around a hundred. Within a batch, the RNN steps through input in order, taking in an input, calculating a hidden state, emitting an output, and then passing on the hidden state to be used for the next item in the sequence. The inputs are any data that can be sequenced (characters, words, frames, etc.), and the outputs can be classifications of or regression on those inputs, or distributions of potential classifications over the output range. The outputs are compared to the true labels, and a loss is calculated. The objective of training is to minimize the loss by choosing the optimal weight matrices W, U, and V. After total loss has been calculated for an entire batch, the gradient of the loss with respect to these weight matrices is calculated and they are then updated via gradient descent. These gradients are typically calculated using *backpropagation*, an efficient algorithm for calculating gradients in computational graphs. The newly updated weight matrices are used for the next batch. The batch size, the size of the hidden layer, and certain constants used in the updating of the weight matrices are all hyperparameters set by the user.

For example, in the use case described in section 4.3, I build an RNN to generate code for the C programming language. Before training, W, U, and V are initialized randomly. If we used our RNN with these randomly-initialized weight matrices to generate text, it would be the same as sampling from a uniform distribution over all characters, and thus it would not look like code. By feeding our RNN input data that looks like valid code, we gradually update our weight matrices so that our RNN generates sequences that better match not only the distribution of characters in the training set but the transitions between characters as well. Examples of code generated by this RNN before and after training can be seen in Figure 4.4. For more examples of character level RNNs, including much more in-depth analysis of the generation of C code, see [KJL15].

```
Batch 0:      0cu     |nv"M$R/m^u tt+^CeU@x>Uh
Batch 10000:     s ged->bat ag_Me_sertaket())
```

Figure 4.4: Text generated by a character level RNN as used in our use case. The first line was generated before any training, and seems like a random sampling of characters. The second line was generated after training on 250000 characters of the Linux Kernel, and seems to have captured some understanding of the syntactic rules of the C programming language, such as function calls, underscore separators in function names, and pointer accessing.

Our full training dataset is the Linux kernel, which I split into batches of 25 characters. This also corresponds to unrolling our RNN 25 steps. Referring to the equations defined in (4.1) and (4.2), the character input x_i could be encoded densely using a mapping such as ASCII, or it could use one-hot encoding. I use a hidden layer of 100 nodes, meaning that the weight matrix U transforms our input x_i into a 100-dimensional vector representation of the original input character. The hidden state vector is also 100 dimensions. Wh_{i-1} and Ux_i are added together and then squashed through the tanh activation function. That result is then multiplied by the weight matrix V, which projects back into the character-space encoding, providing a multinomial distribution over all possible characters.

To train on a given batch, the RNN starts with the first character as input, calculates a hidden state, and outputs a multinomial distribution for what it thinks the next character could be. In RNNbow, I show the most-likely character from that multinomial distribution, $\max(y_i)$, as shown in area 2 of Figure 4.5. The true label for the first character in the batch is the second character in the batch, since in this experiment, I want the RNN to predict succeeding characters based on the current character and its context. I use a softmax loss, which generates high loss if our RNN suggests there is a low probability of the true label and a high probability of other labels. That loss is then used to calculate the gradient update for the weight matrices. In RNNbow, I only visualize the gradients of W, since W is what controls the memory of the RNN.

4.2 RNNbow

RNNbow is a web application that visualizes the gradients used to update parameters during training of a recurrent neural network. In this section, I describe the interface, then I review how the gradient data is harvested during training via back-propagation through time (BPTT) [Hoc98].

4.2.1 Interface

The user interface of RNNbow is a coordinated multiple view implemented in *d3.js* that provides both an overview of the data as well as details of particular elements of the training set. The interface can be seen in Figure 4.5. It takes in data on gradients

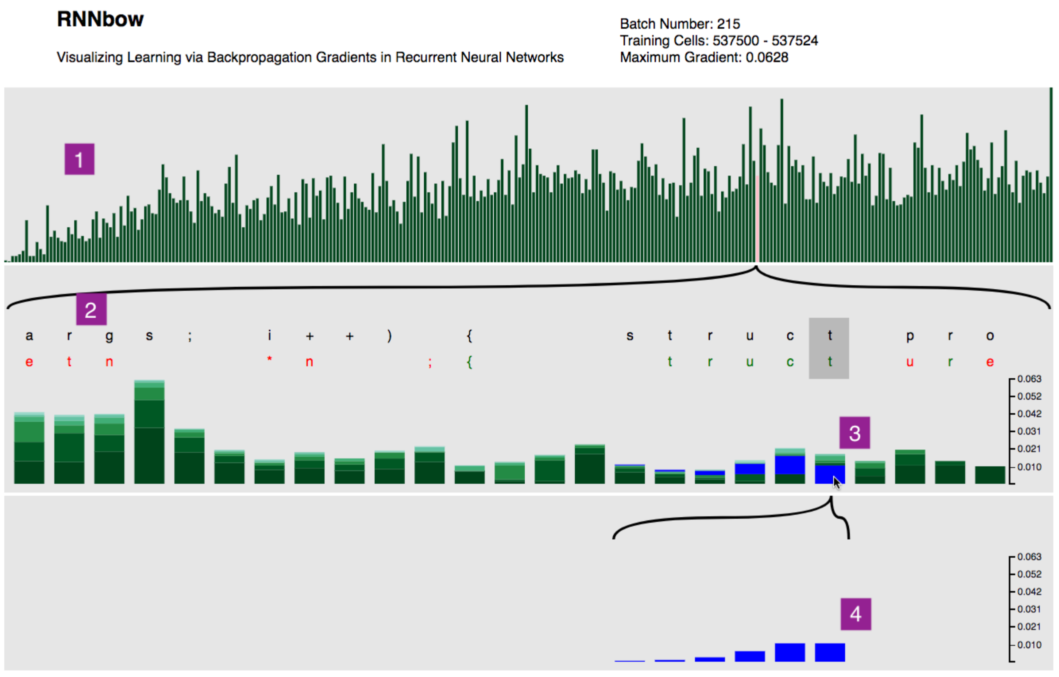

RNNbow
Visualizing Learning via Backpropagation Gradients in Recurrent Neural Networks
Batch Number: 215
Training Cells: 537500 - 537524
Maximum Gradient: 0.0628
1
2
a r g s ; i + +) { s t r u c t p r o
e t n * n ; { t r u c t u r e
3
4
0.063
0.052
0.042
0.031
0.021
0.010

Figure 4.5: The user interface to RNNbow. In (1), the user is shown a bar chart where each bar represents the maximum gradient per batch. By mousing over different batches, the user can drill down to view gradient data from each batch in the training set. The pink bar seen 3/4 of the way through (1) indicates the currently selected batch, and some information for that batch is seen printed above (1). In (2), the top row of characters holds the true labels from the training set, and the bottom row holds the prediction from the RNN at training time. The prediction is colored green if it is correct, and red if it is incorrect. (3) shows the magnitude of the gradients being used to update the weights at each point in time. Each bar is decomposed into the different sources of the loss that created that gradient. On mousing over a particular gradient, we see the gradient due to a single loss highlighted in blue, and projected down to (4) for easy inspection. Different batches can be previewed in (2), (3), and (4) by hovering over their respective bars in (1), and selected by clicking on those bars.

recorded during a pass over a training set. The specific data-visual mappings and how such data is generated during training of an RNN model are both described in section 4.2.2. RNNbow provides both an overview of gradients across all batches and the ability to drill down and visualize the gradients from a single batch at a time, so that the gradients at individual locations in the training set can be seen clearly. Descriptions of the interface below will make repeated use of the numeric labels from Figure 4.5.

4.2.1.1 Area 1: Overview of Max Gradient of All Batches

Area 1 of the interface is a bar chart overview of the maximum gradient within each batch. Our data comprised of 300 batches of gradient data during training. Each bar in area 1 represents the maximum gradient across all training iterations within that batch. Because RNNbow visualizes the maximum gradient in each batch as the height of a bar in a bar chart, the user is able to easily to navigate to individual instances of their training set where the most learning is happening. While visualizing the mean would also be valid in that it would show which batches were the most informative, visualizing the max instead shows which individual elements of the training set are the most informative, which I feel is a more interpretable value to drill down to.

The user is able to drill down into a particular batch by hovering over a particular bar, resulting in information about that batch being visualized in areas 2, 3, and 4. To fix that batch as the selected batch, the user may click on a bar. The currently selected batch is signified as a pink bar in area 4. Some basic information about the batch being visualized is displayed in text above area 1, including the batch number, which training cells it corresponded to, and the maximum gradient in that batch.

4.2.1.2 Area 2: Prediction and True Labels of a Single Batch

Area 2 provides details on demand for the batch selected in are 1. In the top row of characters in area 2, we can see the ground truth labels from the training set per element in the batch. Immediately underneath those labels, we see our RNN's prediction for the label of that element at the time this batch was passed through during training. These predictions are colored according to whether they are correct (green) or incorrect (red). In this figure, the training set batch begins with the five characters "`args; `", and our RNN predicts the five characters "`etn  `". Showing the true and predicted labels helps ground the user in their data.

4.2.1.3 Area 3: Per Batch Gradients

Underneath each label, in area 3, gradients at each time step of the selected batch of training data are visualized as a stacked bar chart. The height of each bar represents the magnitude of the gradient used to update the weights of the RNN *at that step in time*, relative to the gradients within that specific batch. The bars are partitioned according to how far in the future that portion of the gradient resulted from. The lowest, darkest portion of the bar is the gradient due to the loss of the current point in time (due to the loss of the label immediately above the bar). Stepping up in the stacked bar, each new partition is gradient due to the loss accrued due to the succeeding label. In RNNbow, each vertical bar can show the gradient contribution from up to 5 time steps in the future. The number of steps was chosen empirically based on this use case; for other datasets, a larger horizon may be necessary. For more discussion, see section 4.2.2.3.

Blue bars highlight the gradient flow from the loss on a single instance in the batch. They are described in greater detail below.

4.2.1.4 Area 4: Gradients Due to Individual Prediction

While seeing the breakdown of the sum of the gradient at each step may be informative, it is also interesting to see how gradient flows backwards from a particular time step - this would show how the network was learning long-term dependencies. By hovering over any portion of a bar in area 3, we highlight all portions in neighboring bars due to the same loss. In addition, the particular prediction and label that are responsible for that loss are highlighted with a darker gray background, as seen in the gray box behind the two “t” characters in Figure 4.5. For the sake of analysis, these portions are projected down into area 4 to highlight the rate at which the gradient decays.

As an example, in Figure 4.5, the cursor is hovering over the bottom component of the gradient bar below the true label ‘`t`‘. In area 4, we can see that the gradient due to this decision propagated back 5 time steps, albeit diminishing in magnitude. Each blue bar represents the amount the parameters are being updated at that point in time due to the prediction made at ‘t‘. The magnitude of these bars is a proxy for how large the influence is of a previously predicted character, such as the white space character instead of an ‘s‘ at the beginning of ‘struct‘, had on the prediction of the character ‘t‘ at the end of ‘struct‘. The faster this gradient decays, the shorter the time dependency is that the model is learning. I call this projected bar chart the *gradient horizon*, as it aims to show when the gradient contribution vanishes as it passes back. As the user sweeps the mouse up and down and across bars, area 4 changes which gradient horizon it displays, allowing the user to quickly navigate the decomposition of gradients.

4.2.2 Generating Gradient Data

Given a loss function, backpropagation passes that loss back to any parameters involved in the loss's calculation, via the chain rule. In a Convolutional Neural Network, where one prediction is made, there is generally a single loss calculation, which is then passed along via gradients. In an RNN, there are multiple losses; loss is calculated at each output y_i. Calculating the gradient of W is a difficult task since each hidden state and each output are compounded functions of W.

To account for the multiple losses, RNNs use a special form of an algorithm called backpropagation through time (BPTT) [Hoc98]. To use BPTT, RNNs are *unrolled* - that is, each cycle in the computation graph is represented as an additional copy of the RNN, to create a directed acyclic computation graph that backpropagation can then be used on, as seen in Figure 4.3.

Backpropagation is designed to be as computationally fast as possible, making extensive use of dynamic programming to memoize intermediate calculations so that the gradient can be calculated in a single pass backwards through time. However, fully utilizing dynamic programming will cause us to lose track of some of the intra-sequence effects that RNNbow aims to illuminate. Thus, I remove one level of dynamic programming, trading off increased computational complexity for the ability to record itemized gradients. To motivate this, in the following section I fully derive an expression for the gradient, pointing out what the mapping is between RNNbow and the terms in that expression. I also show how our implementation is equivalent to backpropagation.

4.2.2.1 Derivation of Itemized Gradients

We are concerned with $\frac{\partial L}{\partial W}$, the rate at which the loss (L) changes with respect to the weights of the hidden layer (W). In training, I use that quantity to update W via gradient descent. Because we are interested in the gradient contributions from each time step, I decompose the loss into loss contributed from the prediction made at each time step. Here, L_t is defined as the loss due to predicting y_t, and n is the size of the batch.

$$L = \sum_{t=1}^{n} L_t \tag{4.3}$$

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial W} \tag{4.4}$$

For a given time step $t = i$, we have the following decomposition, via the chain rule.

$$\begin{aligned}\frac{\partial L_i}{\partial W} &= \frac{\partial L_i}{\partial y_i} \cdot \frac{\partial y_i}{\partial W} \\ &= \frac{\partial L_i}{\partial y_i} \cdot \frac{\partial y_i}{\partial h_i} \cdot \frac{\partial h_i}{\partial W}\end{aligned} \tag{4.5}$$

We can further decompose the rightmost term, $\frac{\partial h_i}{\partial W}$, but we must be careful: h_i is a function of W, but it is also a function of h_{i-1} which is in turn a function of W, so we must use the product rule.

$$\frac{\partial h_i}{\partial W} = \tanh'(Ux_i + Wh_{i-1}) \left[h_{i-1} + W \frac{\partial h_{i-1}}{\partial W} \right] \tag{4.6}$$

The leftmost term is the derivative of $\tanh(x)$, evaluated at $Ux_i + Wh_{i-1}$. Notice that we still must further expand the rightmost term, just like we had to with $\frac{\partial h_i}{\partial W}$. In the following derivations, the term $\tanh'(Ux_i + Wh_{i-1})$ is truncated to $\tanh'_i$ for the sake of readability.

$$\frac{\partial h_i}{\partial W} = \tanh'_i \left[h_{i-1} + W \tanh'_{i-1} \left[h_{i-2} + W \frac{\partial h_{i-2}}{\partial W} \right] \right] \tag{4.7}$$

We would then have to expand $\frac{\partial h_{i-2}}{\partial W}$, and $\frac{\partial h_{i-3}}{\partial W}$, and on until we end up with the term $\frac{\partial h_1}{\partial W}$ which does not expand since h_0, our initialized hidden state, does not depend on W - it is a hyperparameter set by the user. In this way, the loss due to our prediction at $t = i$ ends up propagating all the way back through the input sequence to $t = 1$.

$$\frac{\partial h_i}{\partial W} = \tanh'_i \left[h_{i-1} + W \tanh'_{i-1} \left[h_{i-2} + \ldots + W \frac{\partial h_1}{\partial W} \right] \right] \tag{4.8}$$

If we were to expand out all of the products in (4.8), we would end up with summands that were only dependent on values available in ordered subsets of the sequence.

$$\frac{\partial h_i}{\partial W} = \tanh'_i h_{i-1} + \tanh'_i W \tanh'_{i-1} h_{i-2} + \ldots \tag{4.9}$$

Note that calculating the first summand only requires knowing h_{i-1} and the additional arguments to $\tanh'_i$, U, W, and x_i. Then, we can memoize the value of $\tanh'_i$, and when calculating the next summand, we only need that memoized value and U, W, h_{i-2}, and x_{i-1}. Let M_j be the $i - j$th summand of (4.9), so $M_i = \tanh'_i h_{i-1}$, $M_{i-1} = \tanh'_i W \tanh'_{i-1} h_{i-2}$. Note that calculating M_j depends only on the values U, W, x_j, h_{j-1}, h_j, and M_{j+1}.

$$\frac{\partial h_i}{\partial W} = M_i + M_{i-1} + \ldots + M_1 \tag{4.10}$$

$$M_j = \frac{M_{j+1}}{h_j} W \tanh'_j h_{j-1} \quad ; \quad 0 < j < i \tag{4.11}$$

$$M_i = \tanh'_i h_{i-1} \tag{4.12}$$

Then we can rewrite (4.5) in a way that clarifies our implementation of its calculation.

$$\frac{\partial L_i}{\partial W} = \sum_{j=1}^{i} \frac{\partial L_i}{\partial y_i} \cdot \frac{\partial y_i}{\partial h_i} \cdot M_j \tag{4.13}$$

In order to calculate the gradient for the entire batch, we would need to sum this over each time step, so we substitute (4.13) into (4.4).

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{n} \sum_{j=1}^{t} \frac{\partial L_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} \cdot M_j \tag{4.14}$$

In order to use RNNbow, I record the summand of (4.14) for each value of (t, j). I call these summands the *itemized gradients*, as they are itemized by the time step that was the source of their loss.

To calculate this in $O(n^2), \{M_j\}$ can be implemented as a one-dimensional dynamic programming table that is filled in from right to left. Then each can be calculated in a single backward pass of the batch, from $j = t$ down to $j = 1$. As an example, suppose that we were training a character-level RNN, and had a batch to train on that was the six characters `g u i t a r`, but our RNN instead predicted the six characters `b a n a n a`. To calculate our gradient, we start at the last character, $t = 6$, and see that we predicted a instead of r, and so we incorporate some loss. We record the gradient due to that prediction at $t = 6$, and then pass back that loss via M_6 to $t = 5, \cdots, t = 1$. Once we have calculated all itemized gradients due to predicting a instead of r at $t = 6$, we jump to $t = 5$, calculate the loss due to predicting n instead of a, and pass a different set of M_j back. This is based on an implementation of BPTT from [Bri15].

The full area of the batch view seen in area 3 of Figure 4.5 represents the full value of $\frac{\partial L}{\partial W}$. The full area of the detailed gradient horizon seen in area 4 of Figure 4.5 represents the full value of $\frac{\partial L_i}{\partial W}$ described in (4.13), and each bar within area 3 corresponds to an individual summand. A vanishing gradient would correspond to the summands decreasing as j decreases, which can be seen in Figure 4.8.

Traditional backpropagation only takes $O(n)$, but it doesn't expose the itemized gradients that we need to record for RNNbow. A further exploration of the relationship between our calculation and traditional backpropagation is given below.

4.2.2.2 Itemized Gradients vs. Backpropagation

The calculation of (4.14) takes $O(n^2)$, where n is the size of the batch. It is possible to utilize dynamic programming further to speed it up to $O(n)$; this is used in most implementations of backpropagation. First, we expand both sums in (4.14).

$$\begin{aligned}\frac{\partial L}{\partial W} =& \frac{\partial L_n}{\partial y_n} \cdot \frac{\partial y_n}{\partial h_n} \cdot [M_n + M_{n-1} + \cdots + M_1] \\ &+ \frac{\partial L_{n-1}}{\partial y_{n-1}} \cdot \frac{\partial y_{n-1}}{\partial h_{n-1}} \cdot [M_{n-1} + \cdots + M_1] \\ &\dots \\ &+ \frac{\partial L_1}{\partial y_1} \cdot \frac{\partial y_1}{\partial h_1} \cdot M_1 \end{aligned} \tag{4.15}$$

Next, we distribute, group the terms by M_j, and factor.

$$\begin{aligned}\frac{\partial L}{\partial W} =& M_n \left(\frac{\partial L_n}{\partial y_n} \cdot \frac{\partial y_n}{\partial h_n} \right) \\ &+ M_{n-1} \left(\frac{\partial L_n}{\partial y_n} \cdot \frac{\partial y_n}{\partial h_n} + \frac{\partial L_{n-1}}{\partial y_{n-1}} \cdot \frac{\partial y_{n-1}}{\partial h_{n-1}} \right) \\ &+ \cdots \\ &+ M_1 \left(\frac{\partial L_n}{\partial y_n} \cdot \frac{\partial y_n}{\partial h_n} + \ldots + \frac{\partial L_1}{\partial y_1} \cdot \frac{\partial y_1}{\partial h_1} \right) \end{aligned} \tag{4.16}$$

Let $N_j = \left(\frac{\partial L_n}{\partial y_n} \cdot \frac{\partial y_n}{\partial h_n} + \ldots + \frac{\partial L_j}{\partial y_j} \cdot \frac{\partial y_j}{\partial h_j} \right)$. Then $\{N_j\}$ can be implemented with a dynamic programming table as with $\{M_j\}$, and we can calculate the gradient in a single pass.

$$\frac{\partial L}{\partial W} = \sum_{i=1}^{n} M_i N_i \tag{4.17}$$

$$N_j = N_{j+1} + \frac{\partial L_j}{\partial y_j} \cdot \frac{\partial y_j}{\partial h_j} \quad ; \quad 0 < j < i \tag{4.18}$$

$$N_i = \frac{\partial L_i}{\partial y_i} \cdot \frac{\partial y_i}{\partial h_i} \tag{4.19}$$

In optimized versions of backpropagation through an RNN, I only make a single pass backwards through time, passing back our accumulation of the gradients N_j, and adding on the gradient of the current time step. For RNNbow, I can't use this method, however, because we lose track of the terms in the expanded product of (4.16) when we make use of the dynamic programming table for $\{N_j\}$. Thus, I need to use the $O(n^2)$ version described by (4.14), saving each summand as we accumulate the sum. It is possible that, depending on the implementation library, keeping track of the intermediate M_j and N_j, and then utilizing vector math, as is commonly used in the python library Numpy, could allow us to use traditional backpropagation.

4.2.2.3 Computational Concerns vs. Estimation

In practice, it may be impractical and unadvised to calculate the itemized gradients throughout all of training. To begin with, this algorithm generates an immense amount of data, storing $O(HNn)$ gradients in a single pass over the training set, where H is the number of nodes in the hidden layer, N is the size of the training set, and n is the batch size. In the use case, I used small batches and a small hidden layer ($n = 25, H = 100$) compared to many networks, and if we had used the entire training set, even using these small settings for n and H we would have created data that was 2500 times the size of our training data.

The problem of data size can be ameliorated by only calculating the itemized gradients periodically - in our use case, we only store the gradients every 100 batches, reverting to the optimized version of backpropagation for the other 99% of batches. Lastly, gradients are averaged between all nodes in the hidden layer, as our goal is to see the general rate of training, rather than drilling down into individual hidden nodes. Since this data is used for visualizations, visualizing data from all nodes would lead to occlusion problems, and the general trends of gradient can be viewed in the average.

It may also not be necessary to step all the way back through the batch when calculating itemized gradients. Equation (4.10) shows that the gradient due to a particular time step's loss is decomposable into a sum of sequence. We can use (4.11) to analyze the rate of decay of that sequence.

$$\frac{M_j}{M_{j+1}} = \frac{h_{j-1}}{h_j} W \tanh'_j \tag{4.20}$$

W is generally initialized close to 0, and regularization is used to keep it having small magnitude during gradient descent. $\tanh'$ has a range of $(0, 1]$, and $\frac{h_{j-1}}{h_j}$ should generally be close to one. Thus, the sequence $\{M_j\}$ should decay as j decrements. Then it would stand to reason that we might be able to choose a value k such that we only have to step back k steps to be close enough to the real gradient.

$$S_k = M_i + M_{i-1} + \ldots + M_{i-k} \tag{4.21}$$

$$S_k \approx \frac{\partial h_i}{\partial W} \tag{4.22}$$

In the use case, I empirically chose $k = 5$ based on manual inspection of the data. However, it is highly likely that acceptable k should vary with RNN architecture and cell choice. It might be possible to find a k such that $||S_k - \frac{\partial h_i}{\partial W}|| < \epsilon$ globally across a validation set. Since k is loosely a measure of how far the gradient horizon is, it would stand to reason that a more sophisticated architecture would demand a larger k.

4.3 Use Case

To demonstrate the use of RNNbow, I trained a character-level RNN on the Linux Kernel to try to get it to generate code that looks like the C programming language, replicating an experiment done in Karpathy et al.'s seminal RNN work [KJL15]. I used batches of 25 characters, and recorded gradients every 100 batches over the first 50000 batches of the training data. I use a hidden layer of 100 nodes, but I average the gradients across all nodes. In this section, I outline several insights that can be found via RNNbow.

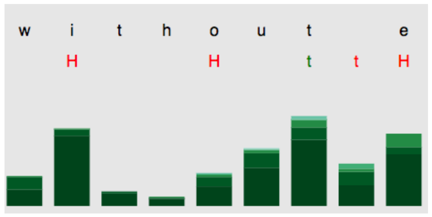


Figure 4.6: The gradients of a batch early in training. Note that the gradients are mostly composed of darker shades of color. This signifies that the gradient updates are primarily due to local loss, zero, one or two time steps away.

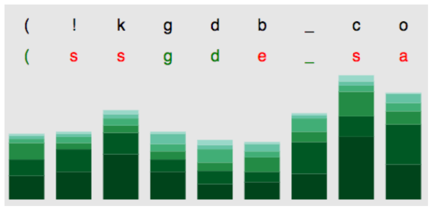


Figure 4.7: The gradients of a batch later in training. Here, the gradients are much more distributed across different shades, suggesting that longer-term dependencies are being learned.

4.3.1 Overview of Gradients Over Time

Figure 4.5 shows the result of training an RNN using our approach. Looking at the overview section, seen as area 1 in Figure 4.5, the first insight is that the magnitude of the gradient starts very small, and then appears to plateau, albeit with a fair amount of variance. This suggests that early in training, the weights update slowly - there may be some burn-in required before the parameters are updating efficiently. The overview also points the user to batches with maximal gradient. It makes it easy for the user to view the elements of the training set that the RNN learns the most from.

It can also be instructive to compare the batch visualizations (area 2 in Figure 4.5) as they change from early in training to late in training. Figure 4.6 shows the gradients of a batch early in training, and Figure 4.7 shows the gradients of a batch late in training. At a glance, the darker the batch visualization is, the shorter the gradient horizon is; a larger portion of the update at each step comes from local losses. In Figure 4.6, most of the bars in the visualized batch are primarily composed of dark green bars. Compare that pattern to a batch later in training in Figure 4.7, where the gradient is much lighter; this corresponds to longer gradient horizons for training in this batch. Exploring the training patterns over training reveals that this particular RNN seemed to lengthen its time dependencies as training went on.

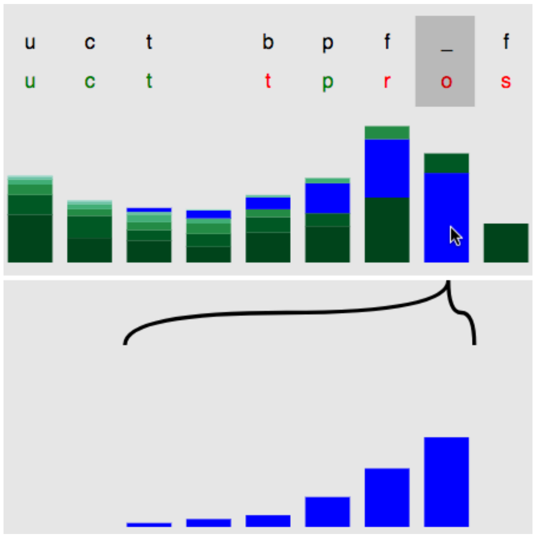


Figure 4.8: The detailed view of the gradients due to a single character's loss. Note that it makes the vanishing gradient effect very apparent.

4.3.2 Vanishing Gradient

A well-known consequence of the activation functions used in RNNs, tanh and σ, is that they result in a gradient that decreases as it is passed back in time. For large swaths of the hyperparameter space, the gradient may decay incredibly fast, restricting any long-term dependency learning [PMB13, Hoc98].

The primary function of the projection of gradients in the visualization, area 4 in Figure 4.5, is to illustrate the rate at which the gradient decays. By mousing over a gradient bar, the user can see the rate at which that particular gradient due to a particular character's loss vanishes. This can be seen in Figure 4.8.

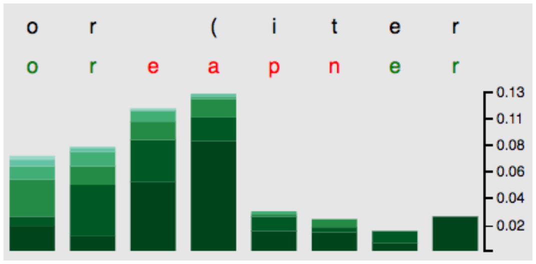


Figure 4.9: The maximal gradient in the training set. The RNN assumes a large gradient when predicting the character `a` instead of the character `(`. This may be due to a large loss being incurred by the model not learning the iterator grammar of the *C* programming language.

4.3.3 Batches With Maximal Gradient

The overview bar chart in area 1 of the interface shown in Figure 4.5 can be used to cue the user towards parts of the training set that the RNN learns the most from. For an example, I clicked on the rightmost bar of area 1 to change the focus of RNNbow to that batch, since that bar had the greatest height and thus the greatest maximal gradient. The stacked bar chart of the maximal gradient in that batch can be seen in Figure 4.9. The maximal gradient is due to predicting the character `a` instead of predicting the character `(`, in spite of the context of being in a `for` loop. Note that it also assumes some gradient from incorrectly predicting the subsequent characters as well. This suggests that our RNN has not learned the iterator grammar of the *C* programming language. It also confirms that our RNN is learning from reasonal mistakes rather than overfitting a non-generalizable error. As this maximal gradient comes late in our training set and is on a legitimate mistake, it cues the user that we have not trained enough and training must continue.

4.4 Discussion

4.4.1 Limitations

RNNbow may not be a good fit for industry-scale recurrent neural networks; such users would be better served with custom visualizations and custom analytics within their deep learning pipelines. RNNbow is most useful to the non-expert. The current implementation does have some scaling issues, both in the interface as well as in the implementation of backpropagation through time, described in section 4.2.2. However, it is more likely that a non-expert would use smaller networks that are executable on a personal computer; that is the scale I aim to currently support.

The design does have some visual layout issues with scale as well. This interface doesn't support more than a few hundred batches, although this should be solvable with some aggregation and drilldown. In practice, RNNs may train over hundreds of thousands of batches. A heuristic could be used to point the user to particularly interesting batches within the training data. Similarly, the stacked bar chart might not scale to batch sizes of 50 or more. In the use case given in this work, with a batch size of 25 and $k = 5$, the stacked bar chart was responsible for visualizing 125 pieces of data. A sophisticated RNN might have a batch size of 128 and would hopefully have a much larger memory; more iterations of design are needed to come up with an appropriate visualization of so many gradients. In addition, some form of aggregation would need to be defined for the predicted labels (in this case, characters) in large batch sizes. Perhaps the largest hurdle to supporting industry scale networks is the number of layers visualized. RNNbow currently supports a single layer; there are popular CNNs with more than a hundred layers, and RNNs are following suit [PGCB13]. It's unclear how the stacked bar chart would scale to even a dozen layers. It may be that a higher resolution visualization of gradients between layers is needed.

Prior to the design of the visualization, basic experiments were run to estimate the gradient horizon throughout the first 50000 batches of training, and it was found that it varied from 2 to 5 characters before the gradient dropped below a certain $\epsilon > 0$. Production-level RNNs, with sophisticated architectures, may need gradient horizons of tens or even hundreds of time steps. The current algorithm used may not be scalable to record the gradient that many steps back. This could be addressed by taking the gradients much less frequently than every 100 batches.

4.4.2 Visualizing Gradients

One of the key insights of RNNbow is that it visualizes the gradient, as opposed to the activation. Gradients are increasingly being accepted as a more salient representation of the learning surfaces and thus a more informative value to study than activation and loss [KL17]. Considering that the gradient is what dictates updates to the model, i.e. what is learned, its visualization is revealing of the training process. It is particularly salient in a visualization of RNNs, as the many-to-many relationship between losses and time steps can be difficult to intuit about. It proved useful in discovering endemic properties like vanishing gradients; there may be other endemic qualities in network training that cannot be noticed in visualizations of activations.

In this work, I focused on the gradient $\frac{\partial L_i}{\partial W}$, in part because this is a value that is already traditionally calculated during backpropagation, as it represents the quantity by which we update the tunable parameter W in the RNN. This in turn led to calculating intermediate gradients including $\frac{\partial L_i}{\partial y_i}$ and $\frac{\partial h_i}{\partial W}$. There are other gradients, however, that could have been calculated during training. In particular, $\frac{\partial h_i}{\partial h_j}$ could have revealed interesting interplay between nodes within the hidden layer and how they kept track of memory. As we seek to crack open the black box of neural networks, it is important to remember that each gradient is a different tangent surface to the manifold on which the full parameter space of the network lies. In this work, I analyze a very specific tangent surface with the goal of understanding time dependencies. In work on visualizing CNNs, it is very common to consider gradients of the activations as they correspond to particular outcomes with a goal of determining what parts of the image are most sensitive to a network's understanding of a label. In order to better understand the cross effects between parameters in a deep network, it may be necessary to explore several of these gradient surfaces, coupled with a visual analytics system that guides the user towards logical conclusions, as RNNbow does for the vanishing gradient. Each gradient surface may change drastically over different architectures and settings of hyperparameters. Comparing gradient surfaces generated by different networks may offer more insight into the sensitivity of network behavior to each hyperparameter.

4.4.3 Role of Visual Analytics in Deep Learning

It is not unreasonable to ask: *is there even a role for Visual Analytics in Deep Learning*? While it is clear that visualization has a role in helping a user build models that they trust, there are extenuating circumstances that make it hard to apply the same considerations to deep learning. A typical Support Vector Machine or Decision Tree has only a handful of hyperparemeters that need to be set by the user, and it learns only a dozen or so parameters during training. In contrast, the famous AlexNet CNN from 2012 [KSH12], a relative dinosaur in deep learning years, fits 60 million parameters. Users must choose architectures, cell types, learning rate, filter size, and a number of other hyperparameters. While this may sound like it offers a good opportunity for building an analytics system that abstracts away these decisions, in practice, it is impractical to infer these values. Generally, visual analytics systems that collaboratively build a model with a user do so by engendering an abstraction, or mental model, of the underlying parameters, and then enable the user to manipulate those parameters with intuitive interactions. In many cases, industry-size neural networks simply have too many underlying parameters, and these parameters tend to interact with one another in unintuitive ways. Once a user has trained enough to understand how those parameters work together, they likely have gained the ability to program their own scripts using one of the many fully-featured deep learning frameworks, at which point they may find a visual analytics sytem superfluous.

Perhaps because most builders of deep learning models were already able to write their own scripts, the most succesful visualizations related to deep learning were not incorporated into visual analytics systems, but were rather used to try to explain the inner workings of a deep learning model post-hoc, after it had been built, in order to gain insight into some of the structures that empirically were working. This led to static visualizations of cleverly calculated values, including gradients, typically overlaid on the input domain to ground the network in the data on which it had been trained. In their popularity, these visualizations suggested that it could be possible to apply some interactivity into model building if the *right* data was visualized.

The current usage of deep learning in the wild This landscape leads to very different recommendations for designing tools depending on the audience of the tool. Our tool is primarily targeted at nonexperts tuning simple one-layer RNNs, possibly in an educational setting. For that reason, I favored *simple, low-dimensional visualizations*, and used *calculated values with a clear, unambiguous meaning.* For an expert, the design decisions are drastically different. The designer can make an assumption that the user will be directly interacting with their model through a deep learning scripting framework and that this framework gives them the ability to calculate their own values of interest. Then these works can focus more on an overview of many different aspects of the network, almost like a monitoring system. Good examples of support for industry-scale model builders can be found in recent works by Google [WSW$^+$18] and Facebook [KAKC18]. For a more thorough treatment of the state of visual analytics in deep learning, I direct the reader to a recent survey [HKPC18].

4.4.4 Future Work

The design space of RNNs is rich and constantly evolving, and much of the progress has the goal of having a longer gradient horizon. As visualizing this gradient horizon is the key feature of RNNbow, it should prove invaluable in comparing different approaches. One direction of RNN research focuses on making more sophisticated cells in which the calculation of the hidden state and outputs differs from the vanilla RNN equations, seen in (4.1) and (4.2). I would like to adapt RNNbow to visualize the gradient flow of these alternative cells. It should be illustrative in comparing a vanilla RNN to a Long Short Term Memory (LSTM). LSTMs have two hidden layers, one of which is supposed to hold short term memory, and one of which holds long term memory. It would be interesting to see if this is supported in visualizations of the gradients of each hidden layer. Another type of cell called a Gated Recurrent Unit (GRU) only has one hidden layer, but in practice accomplishes long-term dependencies similarly to LSTMs. It isn't particularly clear why this should be. In both LSTMs and GRUs, the calculations of emissions require additional computation with each type of cell having several gates that supposedly lengthen the memory. Perhaps RNNbow would be able to be extended to reveal the gradient flow within a cell, not just between cells.

Beyond different cell types, different arrangements of cells have also proved helpful in practical RNNs. One type of cell architecture is a bidirectional RNN, in which two recurrent neural networks are run in parallel for each batch, with one running through the input sequence from left to right and the other in reverse. Their outputs are then combined through a learned linear combination. The bidirectional RNN allows learning former and future context, and viewing the gradient flow in both directions may aid in the understanding of its training. More advanced architectures results from adding layers of RNNs, either to match to multidimensional sequences, or to use multiple layers to capture different levels of abstraction in the input sequence as is typical in CNNs. These architectures are difficult to conceptualize; visualizing the gradient may help. However, their spatial complexity proves a challenge in the current layout of RNNbow; some cleverness will be necessary to determine a layout for such architectures.

In this work, I didn't consider the separate nodes in the hidden layer - I just averaged the gradients together. This is in contrast to many of the works on interpreting ANNs, in particular because the activation of a single node tends to be a binary decision maker. Previous work suggests that individual nodes have unique responsibilities; in the same experiment as used in this work, Karpathy et al. [KJL15] found that a single node was responsible for remembering the state of generated *C* code as being in a parenthesis or out of a parenthesis. However, there is no scalable way to visualize all hidden nodes in RNNbow; some heuristic will need to be developed to cue the user to interesting nodes, and some overview of node performance other than the mean will need to be added.

4.5 Conclusion

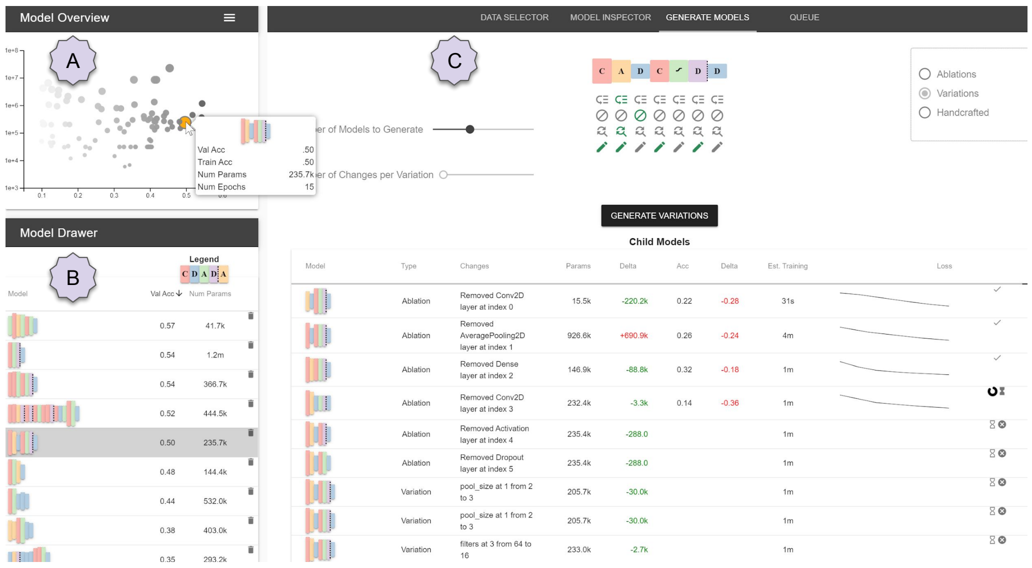
I present RNNbow, a web-based visualization tool for analyzing gradient flow during training of a Recurrent Neural Network. I demonstrate how it can be used to find endemic properties in a network that are important to the user but not necessarily encoded into the loss function used by the algorithm (**V2**). I show how it can provide insights into the learning process, and can be a useful educational tool for illuminating the vanishing gradient phenomenon. I review how to calculate the itemized gradients necessary for loading data into RNNbow. I discuss potential uses of the tool, especially the applications to other RNN architectures.

Chapter 5

Controlling the Loss Function in Neural Architecture Search

In the previous chapter, I described a system that helped a user monitor the qualitative properties of a machine learning model during training. This has value because these properties may not be guaranteed by high performance on the metrics optimized by the machine learning algorithm. In this chapter, I describe a similar tool which helps a user discover and train a convolutional neural network for image classification. Similar to recurrent neural networks, convolutional neural networks are expensive to train. But beyond that, this tool also gives several levels of control over the learning algorithm, implicitly making the loss function better match their estimation of risk for the deployment of the model (**V2**). For example, users can choose to find a model that has a small number of parameters or only uses certain layers.

Deep neural networks have been applied very successfully in recent advances in computer vision, natural language processing, machine translation and many other domains. However, in order to obtain good performance, model developers must configure many layers and parameters carefully. Issues with such manual configuration have been raised as early as 1989, where Miller et al. [MTH89] suggested automated neural architecture search should be useful in enabling a wider audience



Val Acc	.50
Train Acc	.50
Num Params	235.7k
Num Epochs	15

Model	Val Acc	Num Params
	0.57	41.7k
	0.54	1.2m
	0.54	366.7k
	0.52	444.5k
	0.50	235.7k
	0.48	144.4k
	0.44	532.0k
	0.38	403.0k
	0.35	293.2k

Child Models

Model	Type	Changes	Params	Delta	Acc	Delta	Est. Training	Loss
	Ablation	Removed Conv2D layer at index 0	15.5k	-220.2k	0.22	-0.28	31s	
	Ablation	Removed AveragePooling2D layer at index 1	926.6k	+690.9k	0.26	-0.24	4m	
	Ablation	Removed Dense layer at index 2	146.9k	-88.8k	0.32	-0.18	1m	
	Ablation	Removed Conv2D layer at index 3	232.4k	-3.3k	0.14	-0.36	1m	
	Ablation	Removed Activation layer at index 4	235.4k	-288.0			1m	
	Ablation	Removed Dropout layer at index 5	235.4k	-288.0			1m	
	Variation	pool_size at 1 from 2 to 3	205.7k	-30.0k			1m	
	Variation	pool_size at 1 from 2 to 3	205.7k	-30.0k			1m	
	Variation	filters at 3 from 64 to 16	233.0k	-2.7k			1m	

Figure 5.1: A screenshot of the REMAP system. In the Model Overview, section A, a visual overview of the set of sampled models is shown. Darkness of circles encodes performance of the models, and radius encodes the number of parameters. In the Model Drawer, section B, users can save models during their exploration for comparison or to return to later. In section C, four tabs help the user explore the model space and generate new models. The Generate Models tab, currently selected, allows for users to create new models via ablations, variations, or handcrafted templates.

to use neural networks:

> "Designing neural networks is hard for humans. Even small networks can behave in ways that defy comprehension; large, multi-layer, nonlinear networks can be downright mystifying." [MTH89]

Thirty years later, the authors' note is still a common refrain. While research has continued in automated neural architecture search, much of the progress in algorithms has focused on developing more performant models using prohibitively expensive resources. For example, state of the art algorithms in reinforcement learning taking 1800 GPU days [ZVSL17] and evolutionary algorithms taking 3150 GPU days [RAHL18] to discover their reported architectures. Those users that have access to the type of hardware necessary to use these algorithms likely would either have the expertise needed to manually construct their own network or would have access to a machine learning expert that would be able to do it for them.

Likewise, a number of visual analytics tools have been released that make neural networks more interpretable and customizable [HKPC18]. However, these tools presuppose that a sufficiently performant model architecture has been chosen *a priori* without the aid of a visual analytics tool. The initial choice of neural network architecture is still a significant barrier to access that limits the usability of neural networks. Tools are needed to provide a human-driven search for neural network architectures to provide a data scientist with an initial performant model. Once this model has been found, existing visual analytics tools could be used to fine tune it, if needed.

In this chapter, I present REMAP, a tool for human-in-the-loop neural architecture search. Compared to the manual discovery of neural architectures (which is tedious and time consuming), REMAP allows a model builder to discover a deep learning model quickly via exploration and rapid experimentation. In contrast to fully automated algorithms for architecture search (which are expensive and difficult to control), REMAP uses a semi-automated approach where users have fine-grained control over the types of models that are generated. This allows users to trade off between the size of the model, the performance on individual classes, and the overall performance of the resulting model.

Through a set of interviews with model builders, I establish a set of tasks used in the manual discovery of neural network architectures. After developing an initial version of REMAP, I held a validation study with the same experts and incorporated their feedback into the tool. In REMAP, users first explore an overview of a set of pre-trained small models to find interesting clusters of models. Then, users guide the discovery of new models via two operations on existing models: ablations, in which a new model is generated by removing a single layer of an existing model, and variations, in which several new models are generated by random atomic changes of an existing model, such as a reparameterization or the replacement of an existing layer. Users can also manually construct or modify any architecture via a simple drag-and-drop interface. By enabling global and local inspection of networks and allowing for user-directed exploration of the model space, REMAP supports model selection of neural network architectures for data scientists.

The model space for neural networks poses unique challenges for our tool. Whereas many of the parameter spaces explored in other types of models have a set number of choices of parameters, the parameter space for neural networks is potentially infinite - one can always choose to add more layers to a network. In order to aid in the interpretation of the model space, I propose 2-D projections based on two different distance metrics for embedding neural networks based on Lipton's two forms of model interpretability, *transparency* and *post-hoc interpretability* [Lip16].

The second significant hurdle for a visual model selection over neural networks is to find a visual encoding for neural networks that enabled comparison of many networks while still conveying shape and computation of those networks. In this chapter, I contribute a novel visual encoding, called Sequential Neural Architecture Chips (SNACs), which are a space-efficient, adaptable encoding for feed-forward neural networks. SNACs can be incorporated into both visual analytics systems and static documents such as academic papers and industry white papers.

The workflow of this system largely follows the conceptual framework for visual parameter space analysis from Sedlmair et. al. [SHB$^+$14]. A starting set of models is initially sampled from the space in a preprocessing stage, and projections of the models are calculated. Models are then explored in three derived spaces: two MDS projections corresponding to the two distance metrics as well as a third projection with interpretable axes. The system then uses the *global-to-local* strategy of navigating the parameter space, moving from an overview of models to an inspection of individual models in neighborhoods in the derived spaces. During exploration, users can instruct the system to spawn additional models in the neighborhood of already-sampled models, rendering more definition in their mental model of the parameter space on the regions they are most interested in.

Overall, the contributions outlined in this chapter include:

- REMAP, a visual analytics system for semi-automated neural architecture search that is more efficient than existing manual or fully-automated approaches

- A set of visual encodings and embedding techniques for visualizing and comparing a large number of sequential neural network architectures
- A set of design goals derived from a design study with four model builders
- A use case applying REMAP to discover convolutional neural networks for classification of sketches

5.1 Motivation

Neural networks are a class of machine learning models that are inspired by the message passing mechanisms found between neurons in brains. A neural network consists of an architecture and corresponding parameters[1] chosen by the model builder for each component of that architecture. The architecture defines the computational graph mapping from input to output, e.g. how the input space, such as an image, is transformed into the output space, such as a classification (the image is a *cat* or a *dog*). In sequential neural networks, which have simple computation graphs representable by linked lists, the nodes of the computations graphs are called *layers*.

Choosing an architecture that performs well can be difficult [MTH89]. Small changes in parameters chosen by model builders can result in large changes in performance, and many configurations will result in models that quickly plateau without gaining much predictive capacity through training. In addition, training neural networks is very slow relative to other machine learning methods. As a result, the process of manually discovering a performant model can be frustrating and costly in time and resources.

Automated algorithms for neural architecture search generate thousands of architectures in order to find performant architectures [ZL16] and can require tens of thousands of GPU hours of training [ZVSL17, RAHL18]. The best discovered

[1]Parameters chosen by the model builder are sometimes called hyperparameters to differentiate from the parameters of a model that are learned during training. In this chapter, we call both of these terms parameters, but refer to the latter as learned parameters for the sake of delineation.

models might be too large for a model builder if they aim to deploy their model on an edge device such as a tablet or an internet of things device. Ideally, a model builder would be able to handcraft each generated model and monitor its training to not waste time and resources discovering models that were not useful. However, handcrafting each model can be time consuming and repetitive.

In our tool, we seek a middle ground. We initially sample a small set of architectures, and then use visualizations to facilitate exploration of the model space. Model builders can find regions of the space that produce models they are interested in, and then they can execute a local, constrained, automated search near those models. As they get closer to finding an acceptable model, they can explicitly handcraft models through a graphical interface. Rather than training thousands of architectures, the model builder trains orders of magnitude less, and stops the architecture search when they have found an acceptable model. Our semi-automated approach lets the user search for neural architectures without the tedium of manually constructing each model and without the resources and time required by fully-automated algorithms for neural architecture search.

5.2 Design Study

In order to develop a set of task requirements, I interviewed a set of model architects about their practices in manually searching for neural network architectures. I also asked the experts what visualizations might be helpful for non-experts in a human-in-the-loop system for neural network architecture search.

Participants: To gather participants, I recruited individuals with experience in designing deep neural network architectures. Four experienced model builders agreed to participate in the interview study. Three of the participants are PhD students in machine learning, and the fourth participant has a Masters degree in Computational Data Science and works in industry. They had previously used neural networks for medical image classification, image segmentation, natural language processing, and graph inference. One participant contributed to an open source automated neural architecture search library. All four participants were from different universities or companies and had no role in this project. Participants were compensated with a twenty dollar gift card.

Method: Interviews were held with each participant to establish a set of tasks used to manually discover and tune neural networks. The interviews were held one-on-one using an online conferencing software with an author of this work and took one hour each. Audio was recorded and transcribed with the participants' consents so that quotes could be taken.

Interviews were semi-structured, with each participant being asked the same set of open-ended questions. They were first asked to describe their work with neural networks, including what types of data they had worked with. They were then asked about their typical workflow in choosing and fine tuning a model. Then, the benefits of human-in-the-loop systems for neural network model selection were discussed. Lastly, participants were prompted for what types of features might be useful in a visual analytics system for selecting a neural network.

Findings: The findings from the interview study resulted in the following set of design goals.

- **Goal G1: Find Baseline Model:** Three out of the four participants noted that when they are building an architecture for a new dataset, they start with a network that they know is performant. This network might be from a previous work in the literature or it might be a network they've used for a different dataset. This network typically provides a baseline, upon which they then do fine tuning experiments: "*The first step is just use a structure proposed in the paper. Second step I always do is to change hyperparameters. For example, I add another layer or use different dropouts.*" One participant noted that they prioritize using a small model as a baseline because they are more confident in the stability of small models, and it is easier to run fine tuning experiments on small models because they train faster.

- **Goal G2: Generate Ablations and Variations:** Three participants noted that in order to drive their fine tuning, they typically do two types of experiments on a performant network. First, they do ablation studies, a technical term referring to a set of controlled experiments in which one independent variable is turned off for each run of the experiment. Based on the results of the ablation studies, they then generate variations of the architecture by switching out or reparameterizing layers that were shown to be less useful by the ablations. Two participants noted that these studies can be onerous to run, since they need to write code for each version of the architecture they try.

- **Goal G3: Explain/Understand Architectures:** When asked about the types of information to visualize for data scientists, two participants noted that users might be able to glean a better understanding of how neural networks are constructed by viewing the generated architectures. While it may be obvious to the study participants that convolutional layers early in the network are good at extracting features but less helpful in later layers, that understanding comes from experience. By visually comparing models, non-experts might come to similar conclusions. One participant pointed out that the human-in-the-loop could interpret the resulting model more, helping "*two people, the person developing the results, and the person buying the algorithm.*"

- **Goal G4: Human-supplied Constrained Search:** Participants were asked what role a human-in-the-loop would have in selecting a neural network architecture, compared to a fully-automated model search. All four participants noted that if the data is clean and correctly labeled, and there are sufficient resources and time, that a human-in-the-loop would not improve upon an automated neural architecture search. But three participants noted that when resources are limited, the human user can compensate by offering constraints to an automated search, pointing an automated search to particular parts of the model space that are more interesting to the user. One participant noted that for fully-automated model search, "*some use reinforcement learning, [some] use Bayesian optimization. The human can also be the controller.*"

From these findings, I distill the following tasks that our system must support to enable data scientists to discover performant neural network architectures.

- **Task T1: Quickly search for baseline architectures through an overview of models.** Users must be able to start from an effective baseline architecture [G1]. Experts typically refer to the literature to find a starting architecture that has already been shown to work on a similar problem, such as VGGNet [SZ14] or ResNet [HZRS16]. These models, however, have hundreds of millions of parameters and cannot be easily and quickly experimented upon, so some other manner for finding compact, easily trainable baseline models is needed. Users should be able to find small, performant baseline models easily via visual exploration.

- **Task T2: Generate local, constrained searches in the neighborhood of baseline models.** Our tool needs to provide the ability to explore and experiment on baseline models using ablations and variations [G2]. These experiments should help the user in identifying superfluous layers in an architecture. The human user should be able to provide simple constraints to the search for new architectures [G4].

- **Task T3: Visually compare subsets of models, their hyperparameters, and their performances to understand small, local differences in architecture.** The tool should support visual comparisons of models to help the user understand what components make a successful neural network architecture. This helps the user interpret the discovered neural network models [G3] while also informing the user's strategies for generating variations and exploring the model space [G4].

Beyond these three tasks, I also note that compared to many fully automated neural architecture searches, we must be cognizant of limitations on resources. Much of the neural network literature assumes access to prohibitively expensive hardware and expects the user to wait hours or days for a model to train. In our tool, I focus instead on small models that are trainable on more typical hardware. While these models may not be state of the art, they are accessible to a much wider audience.

5.3 REMAP: Rapid Exploration of Model Architectures and Parameters

REMAP is a client-server application that enables users to interactively explore and discover neural network architectures.[2] A screenshot of the tool can be seen in Figure 5.1. The interface features three components: a Model Overview represented by a scatter plot (Fig. 5.1A), a Model Drawer for retaining a subset of interesting models during analysis (Fig. 5.1B), and a data/model inspection panel (Fig. 5.1C).

All screenshots in this section use the CIFAR-10 dataset, a collection of 50,000 training images and 10,000 testing images each labeled as one of ten mutually exclusive classes [KH09]. Model training including both preprocessing and in-situ model generation was done using a Dell XPS 15 laptop with a 2.2ghz i7-8750 processor, 32 GB of RAM, and a NVIDIA GeForce GTX 1050 Ti GPU with 4GB of VRAM.

5.3.1 General Workflow

The user workflow for REMAP is inspired by the common workflow identified in the interview study and encompasses tasks T1, T2, and T3 as defined in section 5.2. First, they find a baseline model by visually exploring a set of pre-trained models in the Model Overview [T1], seen in Figure 5.1A. They select models of interest by clicking on their respective circles, placing them into the Model Drawer, seen in Figure 5.1B. By mousing over models in the overview and scanning the Model Drawer, users can visually compare models of interest [T2]. Then, they use the ablation and variation tools [T3] to fine tune each model of interest, as seen in Figure 5.1C. These tools spawn new models with slightly modified architectures

[2]The source code for the tool along with installation instructions are publicly available at `https://github.com/dylancashman/remap_nas`.

that train in the background, which in turn get embedded in the Model Overview. Instructions for new models are sent back to the server. The server maintains a queue of models to train and communicates its status after each epoch of training.

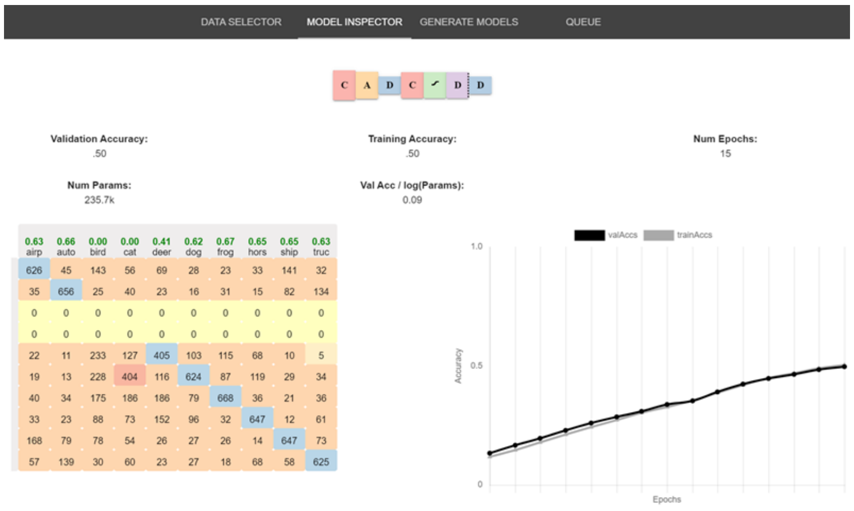
Users iterate between exploring the model space to find interesting baseline models and generating new architectures from those baseline models. For the types of small models explored in this tool, training can take 1-3 minutes for a single model. Users can view the current training progress of child models in the Generate Models tab, or can view the history of all training across all models in the Queue tab. In the Queue tab, they can also reorder or cancel models if they don't want to wait for all spawned models to train.

If users are particularly interested in performance on certain classes in the data, they can select a data class using the Data Selector seen in Figure 5.2b to modify the Model Overview. Users can also see a confusion matrix corresponding to each model in the Model Inspector tab, seen in Figure 5.2a. By interacting with both the model space and the data space, they are able to find models that match their understanding of the data and the importance of particular classes.

5.3.2 Preprocessing

In order to provide a set of model baselines, REMAP must generate a set of initial models. This set should be diverse in the model space, using many different combinations of layers in order to hopefully cover the space. That way, whether the user hopes to find a model that performs well on a particular class or that has a particularly small number of parameters, there will exist a reasonable starting point to their model search.

REMAP generates this initial model space by using a random scheme based on automated neural architecture searches in the literature [EMH18]. A Markov Chain is defined which dictates the potential transition probabilities from layer to layer in a newly sampled model. Starting from an initial state, the first layer is sampled, then its hyperparameters are sampled from a grid. Then, its succeeding layer is sampled based on what valid transitions are available. Transition probabilities and



Validation Accuracy: .50

Training Accuracy: .50

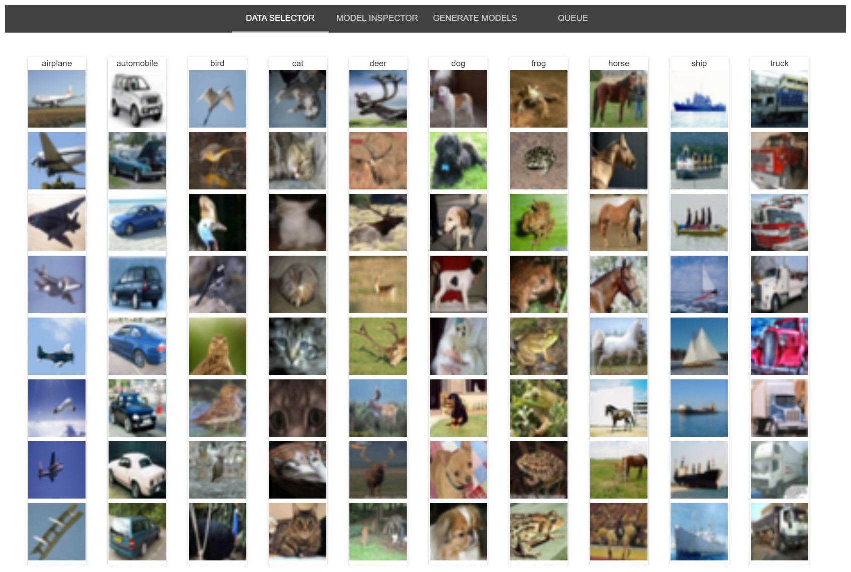
Num Epochs: 15

Num Params: 235.7k

Val Acc / log(Params): 0.09

0.63 airp	0.66 auto	0.00 bird	0.00 cat	0.41 deer	0.62 dog	0.67 frog	0.65 hors	0.65 ship	0.63 truc
626	45	143	56	69	28	23	33	141	32
35	656	25	40	23	16	31	15	82	134
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
22	11	233	127	405	103	115	68	10	5
19	13	228	404	116	624	87	119	29	34
40	34	175	186	186	79	668	36	21	36
33	23	88	73	152	96	32	647	12	61
168	79	78	54	26	27	26	14	647	73
57	139	30	60	23	27	18	68	58	625

(a)



(b)

Figure 5.2: (a) The model inspection tab lets users see more granular information about a highlighted model. This includes a confusion matrix showing which classes the model performs best on or misclassifies most frequently. Users can also view training curves to determine if an architecture might be able to continue to improve if trained further. (b) By selecting individual classes from the validation data, users can update the darkness of circles in the the Model Overview to see how all models perform on a given class.

layer hyperparameters were chosen based on similar schemes in the autoML literature [BGNR16], as well as conventional rules of thumb. For example, convolutional layers should not follow dense layers because the dense layers remove the locality that convolutional layers depend on. In essence, REMAP uses a small portion of a random automated neural architecture search to initialize the human-in-the-loop search. For models in this section and in screenshots, 100 initial models were generated and trained for 10 epochs each, taking approximately 4 hours. While that is a nontrivial amount of required preprocessing time, it compares favorably to the tens of thousands of GPU hours required by a fully automated search [ZVSL17, RAHL18], which might sample over 10,000 models [ZL16].

5.3.3 Model Overview

The top left of the interface features the Model Overview (Fig. 5.1A), a scatter plot which visualizes three different 2D projections of the set of models. The user is able to toggle between the different 2D projections. The visual overview of the model space serves two purposes. First, it can serve as the starting point for model search, where users can find small, performant baseline models to further analyze and improve. The default view plots models on interpretable axes of validation accuracy vs. a log scale of the number of parameters, visible in Figure 5.1. Each circle represents a trained neural network architecture. The darkness of the circle encodes the accuracy of the architecture on a held out dataset, with darker circles corresponding to better accuracy. The radius of the circle encodes the log of the number of parameters. This means that in the default projection, the validation accuracy and the number of parameters are double encoded - this is based on the finding from the interview study that finding a small, performant baseline model is the first step in model selection. The lower right edge of the scatter plot forms a Pareto front, where model builders can trade off between performance of a model and its size, similar to the complexity vs. accuracy plots found in Muhlbacher et al.'s TreePOD tool for decision trees [MLMP18].

Once baseline models have been selected, the Model Overview can also be used to facilitate comparisons with neighbors of the baseline. Users are able to view details of neighboring architectures by hovering over their corresponding points in the overview. By mousing around a neighborhood of an interesting baseline model, they might be able to see how small changes in architecture affect model performance. However, it is well known that neural networks are notoriously fickle to small changes in parameterization [MTH89]. Two points close together in that view could have wildly different architectures.

To address this, REMAP offers two additional projections based on two distance metrics between neural networks. The two metrics are based on the two types of model interpretability identified in Lipton's recent work [Lip16]: structural and post-hoc. Their respective projections are seen in Figure 5.4b, with the same model highlighted in orange in both projections. 2-D Projections are generated from distance metrics using `scikit-learn`'s implementation of Multidimensional Scaling [PVG+11].

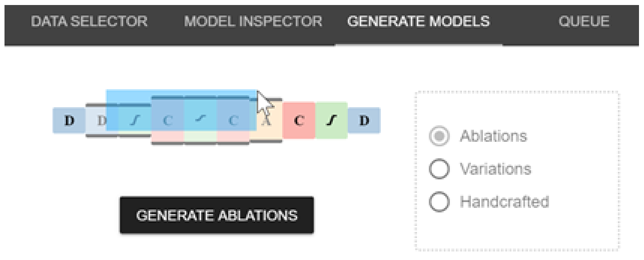
Structural interpretability refers to the interpretability of how the components of a model function. A distance metric based on structural interpretability would place models with similar computational components, or layers, close to each other in the projection. The system uses OTMANN distance, an Optimal Transport-based distance metric that measures how difficult it is to transform one network into another, similar to the Wasserstein distance between probability distributions [KNS+18]. The resulting projection is seen in section B of Figure 5.4b. Projecting by this metric allows users to see how similar architectures can result in large variances in validation accuracy and number of parameters.

Post-hoc interpretability refers to understanding a model based on its predictions. A distance metric based on post-hoc interpretability would place models close together in the projection if they have similar predictions on a held-out test set. Ideally, this notion of similarity should be more sophisticated than simply comparing their accuracy on the entire test set — it should capture if they usually predict the same even on examples that they classify incorrectly. We use the edit distance between the two architectures' predictions on the test set. The resulting projection is seen in section C of Figure 5.4b. It can be used to find alternative baseline architectures that have similar performance to models of interest.

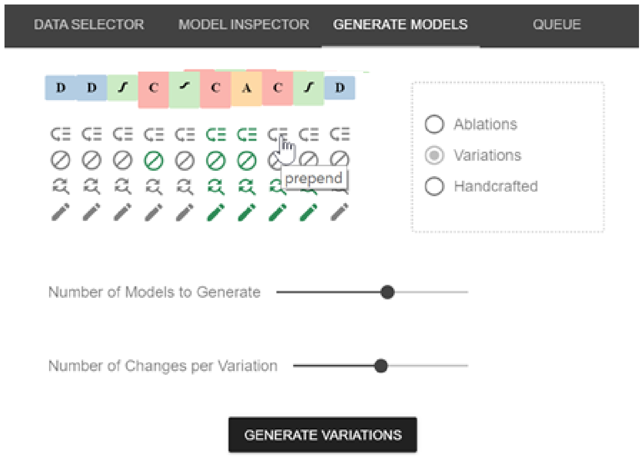
New models generated via ablations and variations are embedded in the

Model Overviews via an out-of-sample MDS algorithm [TP08]. Users can view how spawned models differ from their parent models in the different spaces and get a quick illustration of which qualities were inherited by the parent model.

5.3.4 Ablations and Variations



(a)



(b)

Figure 5.3: Controls for creating (a) Ablations and (b) Variations. Users toggle between the two types of model generation with a radio button. Ablations create a set of models, one for each layer with that layer removed, to communicate the importance of each layer. The Variations feature runs constrained searches in the neighborhood of a selected model. Users toggle which types of variations are allowed for each layer, as well as the number of variations allowed per model

According to our expert interviews, an integral task in finding a performant neural network architecture is to run various experiments on slightly modified versions of a baseline architecture. One type of modification that is done is an ablation study, in which the network is retrained with each feature of interested turned off, one at a time. The goal of ablations is to determine the effect of each feature of a network. This might then drive certain features to be pruned, or for those features to be duplicated.

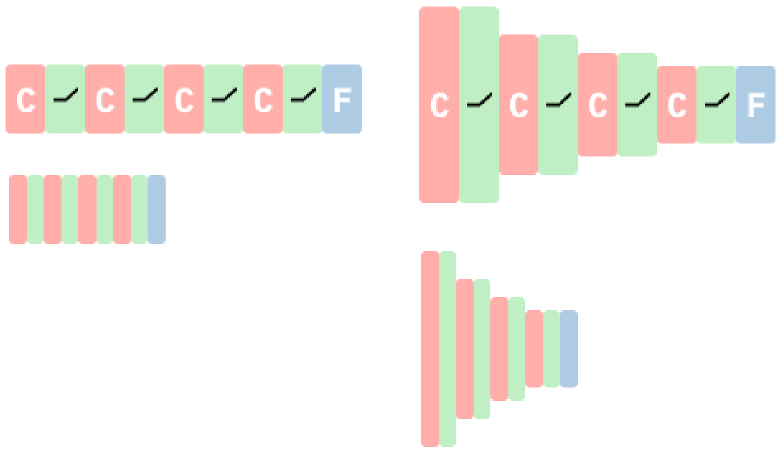
In our system, users can automatically run ablation studies that retrain a selected model without each of its layers. The system will then train those models for the same number of epochs as the parent model, and display to the user the change in validation accuracy. If the user wants to make a more fine-grained comparison between the models, the user can move the model resulting from an ablation into the Model Drawer, and then use the Model Inspector to compare their confusion matrices.

Using the Variations feature in REMAP, seen in Figure 5.3b, users can sample new models that are similar to the baseline model. By default, the variation command will randomly remove, add, replace, or reparameterize layers. Users can constrain the random generation of variations by specifying a subset of types of variations for a given layer. For example, a user might not want to remove or replace a layer that was very important according to the ablation studies, but could still allow it to be reparameterized. Valid variation types are *prepend* with a new layer, *remove* a layer, *replace* a layer, or *reparameterize* a layer.

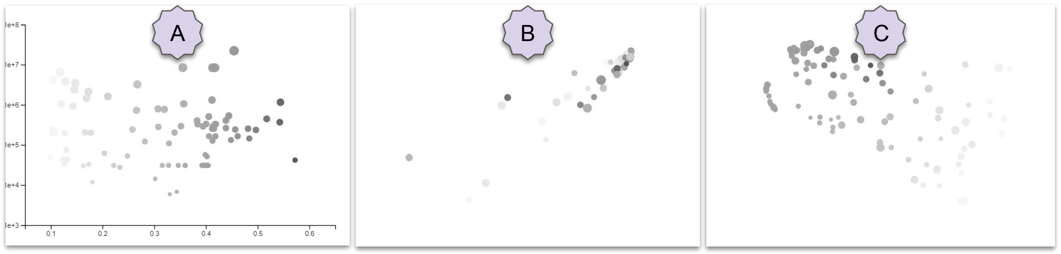
When generating ablations and variations, the user is shown each child model generated from the baseline model that is selected (Fig. 5.1C). Changes that were made to generate that model are shown as well. By viewing all children on the same table, the user may be able to see the effect of certain types of changes; e.g. adding a dense layer typically dramatically increases the number of parameters, while adding a convolutional layer early sometimes increases the validation accuracy. Spark lines communicate the loss curve of each child model as it trains. Each child model is embedded into the Model Overview, and can be moved to the Model Drawer to become a model baseline.

5.3.5 Sequential Neural Architecture Chips

We developed a visual encoding, SNAC (Sequential Neural Architecture Chip), for displaying sequential neural network architectures. Seen in Figure 5.4a, SNAC is



(a)



(b)

Figure 5.4: (a) The *SNAC* visual encoding of a neural network architectures, seen at four different resolutions. This architecture has a three convolutions, each followed by an activation, and concludes with a fully connected layer. (b) Three alternative visual overviews of the model space. Section A shows the set of models on a set of interpretable axes, validation accuracy vs. log of the number of parameters. Sections B and C use multidimensional scaling to lay out the same set of models based on structural similarity (B) and prediction similarity (C). The darkness of the circle encodes the model accuracy, and the radius of the circle encodes the log of the number of parameters.

designed to facilitate easy visual comparisons across several architectures via juxtaposition in a tabular format. Popular visual encodings used in the machine learning [ZF14, KSH12, LB+95, HZRS16, SLJ+15] and visual analytics literature [TM05, WSW+18, KFC16] take up too much space to fit multiple networks on the same page. In addition, the layout of different computational components and the edges between them makes comparison via juxtaposition difficult [Gle18].

The primary visual encoding in a SNAC is the sequence of types of layers. This is based on the assumption that the order of layers is displayed in most other visualizations of networks. Layer type is redundantly encoded with both color and symbol. Beyond the symbol, some layers have extra decoration. Activation layers have glyphs for three possible activation functions: hyperbolic tangent (*tanh*), rectified linear unit (ReLU), and sigmoid. Dropout layers feature a dotted border to signify that some activations are being dropped. The height of each block corresponds to the data size on a bounded log scale, to indicate to the user whether the layer is increasing or decreasing the dimensionality of the activations flowing through it. SNACs are available as an open source component for use in publications and visual analytics tools.[3].

5.4 Expert Validation Study

The initial version of REMAP was developed based on a design study described in section 5.2. Two months later, a validation study was held with the same four model builders that participated in the design study. The goal of the validation study was to assess whether the features of REMAP were appropriate and sufficient to enable a semi-automated model search, and to determine if the system aligned with the mental model of deep learning model builders. Users were asked to complete two tasks using REMAP, and then provide feedback on how individual features supported them in their tasks.

Participants: The same four individuals with experience in designing deep neural network architectures that participated in the first study agreed to participate in the validation study. Participants were compensated with a forty dollar gift card.

Method: Interviews were again held one-on-one using an online conferencing software and took approximately two hours each. Audio of the conversation as well as screen sharing were recorded.

At the start of the study, participants were first given a short demo of the system, with the interviewer sharing their screen and demonstrating all of the features

[3]The open source implementation of SNACs can be viewed at `http://www.eecs.tufts.edu/~dcashm01/snacs/`

of REMAP. Then, participants were given access to the application through their browser and were given two tasks to complete using the tool. The participant's screen was recorded during their completion of the two tasks. Participants were asked to evaluate the features of the tool through their usage in completing their tasks. One of the four participants was unable to access the application remotely, and as a result, directed the interviewer on what interactions to make in REMAP and followed along as the interviewer shared their screen.

Both tasks consisted of discovering a performant neural network architecture for image classification on the CIFAR-10 dataset, a collection of 50,000 training images and 10,000 testing images each labeled as one of ten mutually exclusive classes [KH09]. This dataset was chosen because all four experts had experience building neural network architectures for this dataset. This allowed the participants to quickly assess whether the system enabled them to do the types of operations they might have done manually searching for an architecture on CIFAR-10. In this evaluation, we report participants' feedback on whether the tool enabled them to navigate the model space in a similar manner to their manual model discovery process.

Tasks: The first task given to the participants was to simply find the neural network architecture that would attain the highest accuracy on the 10,000 testing images of CIFAR-10. For the second task, participants were given a scenario that dictated constraints on the architecture they had to find. Participants were asked to find a neural network architecture for use in a mobile application used by bird watchers in a certain park that had many birds and many cats. Birds and Cats are two of the ten possible labels in the CIFAR-10 dataset. The resulting architecture needed to prioritize high accuracy on those two labels, and also needed to have under 100,000 parameters so that it would be easily deployable on a mobile phone. The two tasks were chosen to emulate two types of usage for REMAP: unconstrained model search and constrained model search.

Participants were given up to an hour to complete the two tasks and were encouraged to ask questions and describe their thought process. Then, they were asked about the efficacy of each feature in the tool.

Findings: Participants were able to select models for both tasks. However, each participant expressed frustration at the lack of fine-grained control over the model building process. In general, participants found that the tool could be useful as an educational tool for non-experts because of the visual comparison of architectures. They also acknowledged that using the tool would save them time writing code to run fine tuning experiments. We describe participant feedback on individual features of the system and then outline two additional features added to address these concerns.

5.4.1 Participant Feedback

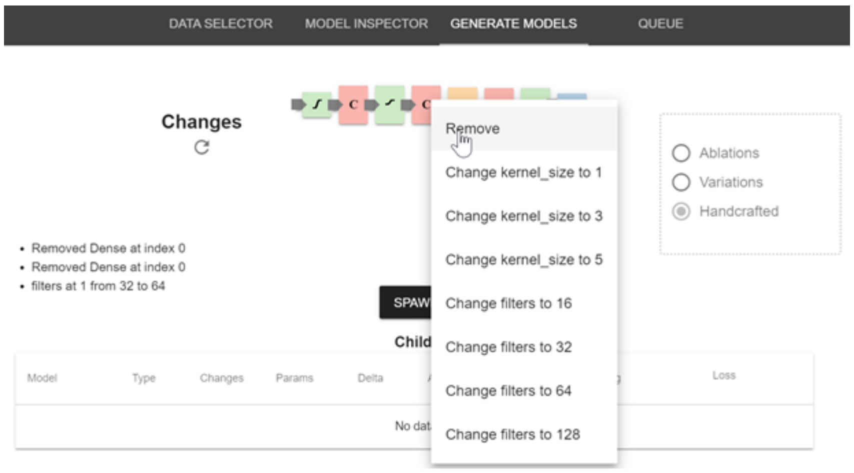
Model Overview: All participants made extensive use of the Model Overview with interpretable axes, seen in Figure 5.4b(A), to find baseline models. Two participants started by selecting the model with the highest accuracy irrespective of parameter size, while one participant selected smaller models first, noting that they start with smaller models when they manually select architectures: *"My intuition is to start with simple models, not try a bunch of random models, using your Model Overview."* Another participant noted that rather than start with the model with the highest accuracy, they *"thought it would make more sense to find a small model that is doing almost as well and then try to change it."*

Two participants appreciated using the Model Overview based on prediction similarity. One noted "*To me, exploring the models in that space seems like a very appealing thing to do. ... To be able to grab a subselection of them and be able to at a glance see how they are different, how do the architectures differ?*". Another participant used the model view in trying to find a small architecture for the second task that performed well on cats and birds: "*instead of looking at every model, I start with a model good at birds, then look at prediction similarity. Since it does good on birds, I'm assuming similar models do well on birds as well*". That participant explored in the neighborhood of their baseline model for a model that also performed well on cats.

Model drawer and inspection: Each participant moved multiple interesting models into the Model Drawer, and then inspected each model in the Model Inspector. They all used the confusion matrix to detect any poor qualities about models. Several participants ignored or discarded models that had all zeros in a single row which indicated that the model never predicted an instance to be that class across the entire testing dataset.

Generation of new models: While some participants found the ablation studies interesting, one participant noted that some ablations were a waste of resources: "*I basically don't want my system to waste time training models that I know will be worse... For example, removing the convolutional layer.*". Some participants used their own background and experience to inform which variations they did, while others used the Model Overview and Model Drawer to discover interesting directions to do variations in. When viewing two architectures with similar accuracy but very different sizes, a participant commented "*I can visually tell, the only difference I see is a pink color. It's a nice way to learn that dense layers add a lot of parameters.*"

All participants expressed a desire to have more control over the construction of new models. This would allow them to do more acute experimentation once they had explored in the neighborhood of an interesting baseline model. One participant described it as the need for more control over the model generation process: "*I think we need more customization on the architecture. Currently, everything is rough control ... Of course for exploring the search space, rough control would be more helpful. But for us to understand the relation [between architecture and performance], sometimes we need precise control.*" All participants noted that relying on rough control resulted in many models being spawned that were not of interest to them, especially once they had spent some time exploring the model space and knew what kind of model they wanted to generate.



Ablations Variations [✓ Handcrafted]

- Removed Dense at index 0
- Removed Dense at index 0
- filters at 1 from 32 to 64

Figure 5.5: The ability to handcraft models was added based on feedback from a validation study with model builders. Starting from a model baseline, users can remove, add, or modify any layer in the model by clicking on a layer or connections between layers. This provides fine-grained control over the models that are generated.

5.4.2 System Updates

The feedback from the expert validation study led to two changes to the system. Both changes allow for more fine-grained control over which models were generated, both to allow for more precise experimentation and to reduce the number of models that need to be trained.

- **Change C1: Creating Handcrafted Models:** While variations proved useful for seeing more models in a small neighborhood in the model space, participants expressed frustration at not being able to explicitly create particular architectures. To address this, we added the handcrafted model control, seen in Figure 5.5. Users see the same SNAC used in the Ablations and Variations controls, but with additional handles preceding each layer. By clicking on the layer itself, users can select to either remove a layer or reparameterize it. By clicking on the handles preceding each layer, the user can choose to add a layer of any type.

- **Change C2: Subselections of ablations:** Two participants found that the ablations tool wasted time by generating models that weren't particularly of interest to the user. We added a brushing selector, seen in Figure 5.3a to allow the user to select *which* layers were to be used in ablations, so that the user could quickly run ablations on only a subset of the model.

5.5 Use Case: Classifying Sketches

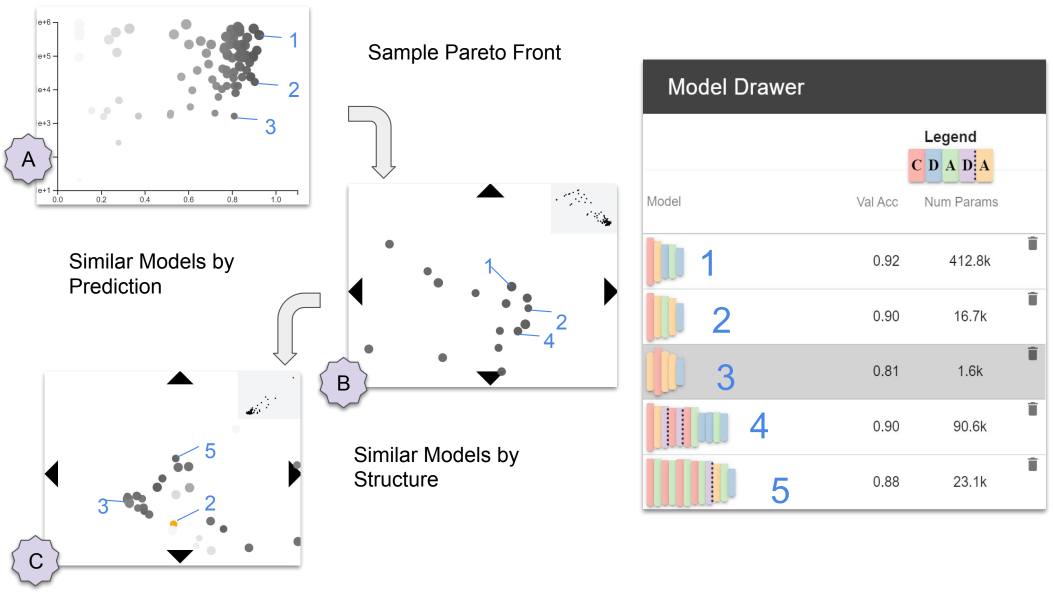
To validate the new features suggested by the study, we present a use case for generating a performant, small model for an image classification dataset. In this use case, we refer to tasks T1, T2, and T3 supported by our system as outlined in section 5.2.

Leon is a data scientist working for a non-governmental organization that researches civil unrest around the world. He is tasked with building a mobile app for collecting and categorizing graffiti, and would like to use a neural network for classifying sketched shapes. Because his organization would like to gather data from all over the world, the application must be performant on a wide swath of mobile devices. As a result, he needs to consider the tradeoff between model size and model accuracy.

Data: He downloads a portion of the *Quick Draw* dataset to use as training data for his image classifier. Quick Draw is a collection of millions of sketches of 50 different object classes gathered by Google [qui]. Rather than download the entire dataset, Leon downloads 16,000 training images and 4,000 training images from each of 10 classes that are commonly found in graffiti to serve as training data[4]. Overnight, he uses REMAP to auto-generate an initial set of 100 models, and the next day, he loads up REMAP to begin his model search.

Search for baselines in the Model Overview: To find a set of baseline models [T1], he starts with the default Model Overview, seen in Figure 5.6A. He sees that there are many models that achieve at or above 90% accuracy, but they appear to have many parameters. He samples three models from the pareto front, two which

[4]For this use case, we used the 10 most convergent classes in Quick Draw as identified by Strobelt et al. [SPM]

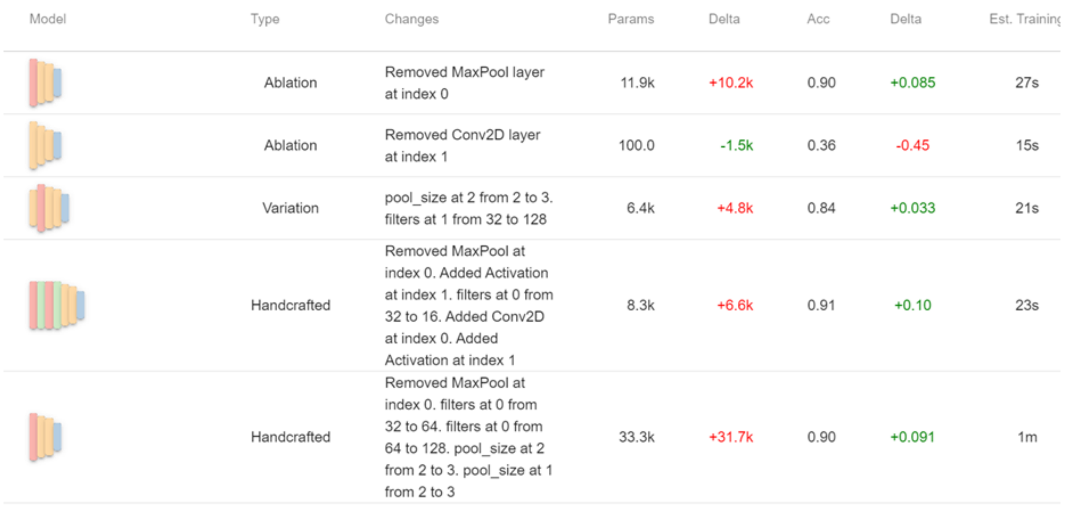


Model	Val Acc	Num Params
1	0.92	412.8k
2	0.90	16.7k
3	0.81	1.6k
4	0.90	90.6k
5	0.88	23.1k

Figure 5.6: In our use case, the model builder first samples models 1, 2, and 3 on the pareto front of accuracy vs. model size. He then selects models 4 and 5 from the two alternative Model Overviews provided.

have the high accuracy he desires and one which has an order of magnitude less parameters. He switches the model view to lay out models based on performance prediction similarity (Figure 5.6B) and hovers the mouse around the neighborhood of his selected models to see what alternative architectures could result in similarly good performance, and adds an additional model which has multiple convolutional and dense layers, as well as some dropout layers. Lastly, he switches the model view to lay out models based on structural similarity (Figure 5.6C) to see how small differences in architecture correspond to changes in either accuracy or parameters [T3]. He selects a fifth model which differs from his previously selected models in that it spreads its convolutional filters over multiple layers instead of concentrating them in a single initial layer.

Ablations: He decides to start with the smallest model, model 3, since it has reasonably high accuracy of 81% and a very small amount of parameters, approximately 1600. Having chosen a baseline, he moves on to generate local, constrained searches in the neighborhood of the baseline [T2]. After checking in the Model Inspector that the model performs reasonably well on all classes, he runs ablations on this model



Model	Type	Changes	Params	Delta	Acc	Delta	Est. Trainin
	Ablation	Removed MaxPool layer at index 0	11.9k	+10.2k	0.90	+0.085	27s
	Ablation	Removed Conv2D layer at index 1	100.0	-1.5k	0.36	-0.45	15s
	Variation	pool_size at 2 from 2 to 3. filters at 1 from 32 to 128	6.4k	+4.8k	0.84	+0.033	21s
	Handcrafted	Removed MaxPool at index 0. Added Activation at index 1. filters at 0 from 32 to 16. Added Conv2D at index 0. Added Activation at index 1	8.3k	+6.6k	0.91	+0.10	23s
	Handcrafted	Removed MaxPool at index 0. filters at 0 from 32 to 64. filters at 0 from 64 to 128. pool_size at 2 from 2 to 3. pool_size at 1 from 2 to 3	33.3k	+31.7k	0.90	+0.091	1m

Figure 5.7: After generating ablations, variations, and several handcrafted models, the model builder compares all discovered models and chooses the model in the fourth row, because of its high accuracy and low number of parameters.

and sees that removing the first and last max pool layers increased both accuracy and the number of parameters. He notes that, with an accuracy of 90% and 11.9k parameters, the model resulting from removing the first max pool layer is now on the pareto front between validation accuracy and number of parameters, so he adds it to his Model Drawer for further consideration.

Variations: While the ablations indicated that he may want to remove some of the pooling layers, he wants to see the effects of various other modifications to his baseline model. He decides to generate variations of all kinds (*prepend, remove, replace, reparameterize*) along the pooling layers, and also allow for reparameterization of the convolutional layer. He generates 10 new variations from those instructions, and by looking at their results, sees that increasing the number of convolutional filters results in too many parameters, but this can be compensated for by also increasing the pool size.

Handcrafting Models: After developing an understanding of the model space, he generates some handcrafted models. He removes the first max pooling layer because that helped in the ablation studies. He then creates three new models from this template. First, he splits the starting convolutional layer into three convolutional layers with fewer filters, to be more like model 5. He then tries adding dropout, to be more like model 4. Lastly, he creates a model with activations like model 2, and different options chosen for pooling layers and kernels inspired by the variations. The trained results can be seen in Figure 5.7.

Result: Leon eventually decides on using an architecture with 91% accuracy and only 8.3k parameters, seen in the fourth row of Figure 5.6. This model has comparable accuracy to models 1 and 2 that were initially chosen from the pareto front, seen in Figure 5.6, but drastically fewer parameters than model 1 (412.8k) and model 2 (16.7k). As a result, the architecture found by Leon can be deployed on older technology and classify images faster than any of the initially sampled models.

5.6 Discussion

5.6.1 Human-in-the-Loop Neural Architecture Search

This study suggested that the presence of a human-in-the-loop benefited the discovery of neural architectures. However, a common pattern in deep learning research is for applications to start with the neural network as an independent component in a set of semantic modules, only for subsequent research to point out that subsuming all components into the neural network and training it end-to-end results in superior performance. As an example, the *R-CNN* method for object recognition dramatically outperformed baselines for object detection using a CNN in concert with a softmax classifier and multiple bounding box regressors [GDDM14]; however, its performance was eclipsed only one year later by *Fast R-CNN*, which absorbed the classifiers and regressors into the neural network [Gir15, ZZXW19]. This suggests that the user processes in REMAP , such as selecting models on the pareto front and running certain ablations and variations, could be automated, and the whole process run end to end as a single optimization without a human-in-the-loop. Ultimately, this perspective ignores the tradeoffs that users are able to make; users can very quickly and efficiently narrow the search space to only a small subset of interesting baselines based on a number of criteria that are not available to the automated methods. These include fuzzy constraints on the number of parameters, a fuzzy cost function that differs per class and instance, and domain knowledge of the deployment scenario of the model. For this reason, I advocate that the human has a valuable role when searching for a neural architecture using REMAP .

5.6.2 Generalizability

The workflow of REMAP is generalizable to other types of automated machine learning and model searches beyond neural networks. The two primary components of REMAP are a set of projections of models and a local sampling method to generate models in a neighborhood of a baseline model. As long as these two components can be defined for a model space, the workflow of REMAP is applicable. Of the three projections used, both the semantically meaningful projection of accuracy vs. number of parameters and the prediction similarity distance metrics are generalizable to any machine learning model, while structural similarity distances can be easily chosen, such as the Euclidean distance between weights for a support vector machine. Similarly, random sampling in the neighborhood of a model can be done in any number of ways; if the model space is differentiable, gradient-based techniques can be used to sample in the direction of accurate or small models.

5.6.3 Scalability

In order to facilitate human-in-the-loop-neural architecture search, REMAP must make several constraints on its model space. It limits the size of the architectures it discovers so that they can be trained in a reasonable amount of time while the user is engaged with the application. In certain domains, however, the tradeoff between accuracy and size of the model is very different; stakeholders don't want to sacrifice any accuracy. In that case, the cap on model size in REMAP could be removed, and REMAP could be used to find large networks that take many hours to train. It isn't feasible to expect a user to stay *in situ* the entire time while REMAP trained the several dozen models needed to enable architecture discovery. Instead, a dashboard-like experience, easily viewable in a casual setting on a small screen such as a phone might be preferable. In general, the types of user experiences used in visual analytics tools for machine learning models may have to be adapted to the scale of time necessary for constructing and searching through industry-level neural networks.

The visual encoding used for neural network architectures, SNACs, can only display network architectures that are linked lists, which leaves out some newer types of architectures that have *skip connections*, which are additional linkages between layers. This problem could be solved by improving the encoding to communicate skip connections. Ultimately, supporting every possible network architecture amounts to supporting arbitrary graphs, and there is no space-efficient way to do so without losing information. For that reason, we limit the scope in this project to network architectures that are linked lists, because they are simpler to understand and are a common architecture that are more performant than non-neural network models for image classification problems.

5.7 Conclusion

Neural networks can be difficult to use because choosing an architecture requires tedious and time consuming manual experimentation. Alternatively, automated algorithms for neural architecture search can be used, but they require large computational resources and cannot accommodate soft constraints such as trading off accuracy for model size or trading off on performance between classes. I present REMAP, a visual analytics tool that allows a model builder to discover a deep learning model quickly via exploration and rapid experimentation of neural network architectures and their parameters. REMAP enables users to quickly search for baseline models through a visual overview, visually compare subsets of those models to understand small, local differences in architectures, and then generate local, constrained searches to fine tune architectures. Through a design study with four model builders, we derive a set of design goals. I provide a use case in building a small image classifier for identifying sketches in graffiti that is small enough to used on even very old mobile devices. I demonstrate that the semi-automated approach of REMAP allows users to discover architectures quicker and easier than through manual experimentation or fully automated search. It also gives the user greater control over the types of models that are found by the learning algorithm. This allows them to implicitly account for their own understanding of the risk of deploying the trained model.

Chapter 6

Data Augmentation to Improve Learnability

In chapters 3, 4, and 5, I presented three different ways in which visual analytics tools can help a user discover, assess, and train a model to predict on a data source. But in many cases, the applied machine learning goal may not be learnable with the data available. In that case, no matter the amount of computation and human control, it may be impossible to train a model that meets the needs of the user (**V3**). In this chapter, I offer a visual analytics solution to this issue. Experts in the domain of the applied machine learning problem may know additional information that is not included in the initial dataset. I present a visual analytics tool that lets a user explore external information repositories in order to craft additional attributes.[1] These attributes can make the applied machine learning problem more learnable.

Over 20 years ago, Pirolli and Card coined the term "information foraging" to denote the process of seeking and gathering information to apply towards a task [PC99, PC05]. While they focused on foraging for additional entities or rows

[1]As with the work described in chapter 3, much of the work described in this section was done as part of a team of researchers in DARPA's data driven discovery of models project. While I use first person pronouns in describing this work, there was significant effort in advisement, planning, and software engineering by many others. A complete list can be found in the author list provided in the references [CXD$^+$20]. In particular, Subhajit Das and Shenyu Xu helped considerably with design and implementation of the user interface, and were responsible for running the user study and analyzing it.

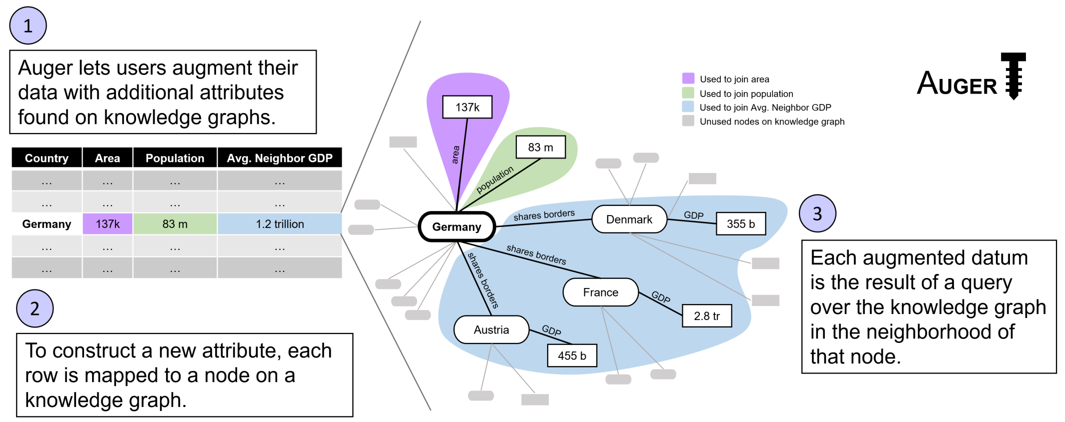

1
Auger lets users augment their data with additional attributes found on knowledge graphs.
Country | Area | Population | Avg. Neighbor GDP
Germany | 137k | 83 m | 1.2 trillion
2
To construct a new attribute, each row is mapped to a node on a knowledge graph.
137k
area
83 m
population
Germany
shares borders
Denmark
GDP
355 b
France
2.8 tr
Austria
455 b
Used to join area
Used to join population
Used to join Avg. Neighbor GDP
Unused nodes on knowledge graph
AUGER
3
Each augmented datum is the result of a query over the knowledge graph in the neighborhood of that node.

Figure 6.1: The Auger system allows a user to augment a tabular dataset with additional columns gathered from a knowledge graph. This figure illustrates the process of gathering three additional attributes corresponding for a single row of the data.

of a dataset, foraging for additional data attributes can unlock new analytical capabilities. For example, if a dataset contains a list of countries, adding the population of those countries as a column enables per-capita analyses.

Traditionally, crafting new attributes and augmenting a dataset with them is done prior to any visual analysis. Incorporating this process into visual analytics workflows can benefit both the augmentation process as well as the underlying tasks of the system in several ways. First, the need for an additional column may only arise based on insights generated during analysis; if data augmentation is embedded in the system, users can toggle back and forth between analysis and augmentation. Second, the process of discovering data and crafting a new attribute is an analytical task in its own right, and typically consists of complex querying that can be abstracted away by a visual interface. Lastly, user-driven curation of the columns of the dataset can serve as an additional medium of communication between user and system: the user can communicate domain knowledge by adding new columns, and can likewise learn from the attributes that are discovered by the system.

However, there are many complications in integrating data augmentation into visual analytics systems. Finding external data and matching it to the entites in the user's dataset is nontrivial. Once connected to external data, it can be difficult to determine which data is relevant or useful to the user's analysis. Some attributes may require the aggregation of multiple pieces of external data and can require joining through several intermediate entities. These hurdles should be abstracted away so that the user is not required to be a database expert.

Large scale information repositories provide the potential for interactive data augmentation because they put the potential data at the user's fingertips. However, in order to use such repositories a number of challenges must be addressed. If the repository is in the form of a data pool (a large collection of tables, e.g. WikiTables [BND13]), it can be difficult to determine which tables are relevant and joinable to the user's dataset. Tabular data can often be fragmented as well: for example, if a user is foraging for data on water usage across the United States, each municipality might be responsible for publishing its own data, and the formats may not align, resulting in sparse, error-prone joins. Joining together multiple tables can also result in ambiguities over the type of join (left, right, outer, inner) that are difficult to resolve, and limit the types of attributes that can be constructed without additional data munging. In addition, plenty of external data may be irrelevant or even harmful to the analysis tasks at hand. While there have been great strides in addressing some of the technical issues in using Data Lakes (see section 2.6.1), little work has been done to allow a user to explore potential attributes and construct new ones. Knowledge graphs offer an alternative to data lakes, because they simplify entity matching and joining ambiguities due to their graph format. They bring different challenges, however, in scalability and complexity, and there is still a dearth of tools available to facilitate foraging over knowledge graphs.

In this chapter, I present Auger, a visual analytics system for augmenting datasets with carefully crafted columns. These columns are retrieved by running queries over knowledge graphs, as illustrated in Figure 6.1. First, each row of a dataset is mapped to an entity in a knowledge graph. Then, to help the user identify potential information of interest, Auger explores the local graph neighborhood about entities in the dataset to determine commonly held attributes. Using visualizations of data quality, distribution, and the local topology of the knowledge graph, Auger guides the user through crafting additional columns of data without any programming or explicit querying. Users can express complicated operations over the knowledge graph by interacting with these visualizations. In sum, Auger abstracts away the complexity of searching, retrieving, and joining data so that the user can focus on the exploratory task of determining which attributes are relevant to their analysis.

Auger shows the promise of integrating column augmentation as a foraging process within a VA system. We claim that a resulting VA system can make the exploration and discovery of additional attributes interactive, and let users construct complex features without programming. Users are able to discover new attributes as well as create attributes they had in mind. The construction of the augmented dataset can improve the outcome of downstream analytical tasks, such as insight generation, predictive modeling, or any other number of tasks that rely on the presence of a robust dataset.

We offer evidence of these claims by describing two use cases for insight generation and predictive modeling. In the first use case, we demonstrate how a domain scientist's analysis can be enriched through cycles of in-situ column augmentation and analysis. And in the second use case, we show how a user can craft attributes that result in significantly more accurate machine learning models. We also conduct a preliminary user study on two datasets to assess the usability of Auger. The results confirm that users are able to both discover and craft relevant attributes easily and quickly.

In this chapter, I present the following contributions:

- I present a visual analytics system, Auger, for exploratory data augmentation using knowledge graphs.
- I provide use cases of Auger being applied to both insight generation and predictive modeling to demonstrate the generality of our approach.
- I conduct a user study to assess the ability of our system to join both semantically meaningful external data as well as data that improves a predictive model trained on the dataset.

6.1 Knowledge Graphs

Relational databases are one of the most popular formats to store data because they can be implemented and queried effectively and efficiently. However, recently knowledge graphs have gained popularity for storing very large datasets because their entity-based model is conducive to how humans think about data. Knowledge graphs represent data entities as nodes in a graph and the edges between those nodes are relationships between those entities. The term *knowledge graph* has been loosely used to describe collections of information. Several different definitions have been offered in recent years. In practice the term has been used interchangeably with *knowledge base* or *ontology* [EW16]. Structuring entities in the form of an ontology is also used to organize conceptual spaces, for example for the evaluation of visual analytics systems [CE19] and visualizations that support machine learning tasks [SKKC19].

The current popularity of knowledge graphs to store information can be traced back to 2012, when Google introduced its *Knowledge Graph*. Google's Knowledge Graph is a repository that retrieves facts about entities in search terms on their search results pages [Sin12, DGH$^+$14], and is used to power the Google Assistant [Lyn16] and Google Home voice queries [Boh16]. Recent work has demonstrated that the approach can generalize to other artificial intelligence tasks such as image captioning and conversational agents [HZLL19, ZYZC18, HCH15].

Since 2012, Wikipedia and other Wikimedia projects have collected information into the *Wikidata* knowledge graph. It contains broad information about tens of millions of entities. Other large public repositories exist, from DBPedia [ABK$^+$07], a collection of data resources, to the domain-specific universal protein knowledgebase *UniProt* [uni17] and the linguistic resource *WordNet* [Mil95]. Many of these graphs are part of the Linked Open Data Cloud [McC20], a knowledge graph that contains information about other knowledge graphs. All these graphs contain a tremendous wealth of structured data that can be accessed programmatically with little to no data munging.

A wealth information is available in machine-readable form in knowledge graphs. With Auger users can tap these knowledge repositories as a data source to augment a dataset. Knowledge graphs have the advantage of providing clean, curated data which means that the need for data cleaning and matching is greatly reduced. In addition, the entity-focused way in which knowledge graphs store information can make it easier to think about the relationship of data objects and thus helps users guide the augmentation process.

In this chapter, I describe how a visual analytics system can mine a knowledge graph to discover new attributes to join to a dataset. I assume that knowledge graphs contain entities as nodes and express attributes of those entities as edges, connecting the source entity to either another entity (to express a type of relationship) or to a literal (such as a string, number or datetime). The number of edges connected to a node can be a proxy for the number of attributes that are held by that entity. The flexibility of this structure allows for complex relationships to be expressed between entities, including one-to-one (i.e. a country has one head of state), one-to-many (a country is composed of multiple municipalities), and many-to-many (countries share borders with other countries non-exclusively).

Auger requires that the connected knowledge graph has a data retrieval endpoint that can respond to simple queries about entities and their neighbors. The connected graph must also have some sort of service to map from the values in the dataset uploaded to Auger to the entities on that knowledge graph. For example, if a dataset with U.S. states is uploaded, there must be an existing service that maps from "New York" or "NY" to the entity in the knowledge graph corresponding to that state. These two requirements allow Auger to connect a user's data to the knowledge graph and to retrieve relevant data for the user to forage from.

In the experiments in this chapter, I connect Auger to Wikidata. It meets both requirements listed above: it responds to the SPARQL query language so gathering neighborhood data about entities is simple, and it has a service `wbsearchentities` to map strings to entities on the graph. But it also has some extra features that I take advantage of. Wikidata contains a broad set of information making it easy to find related attributes for many different kinds of datasets. It also has additional metadata about the data in its graph including data-type information and human readable descriptions and labels for each node and edge in the graph. Lastly, due to its popularity, there is a large amount of documentation and guidance on constructing complex queries.

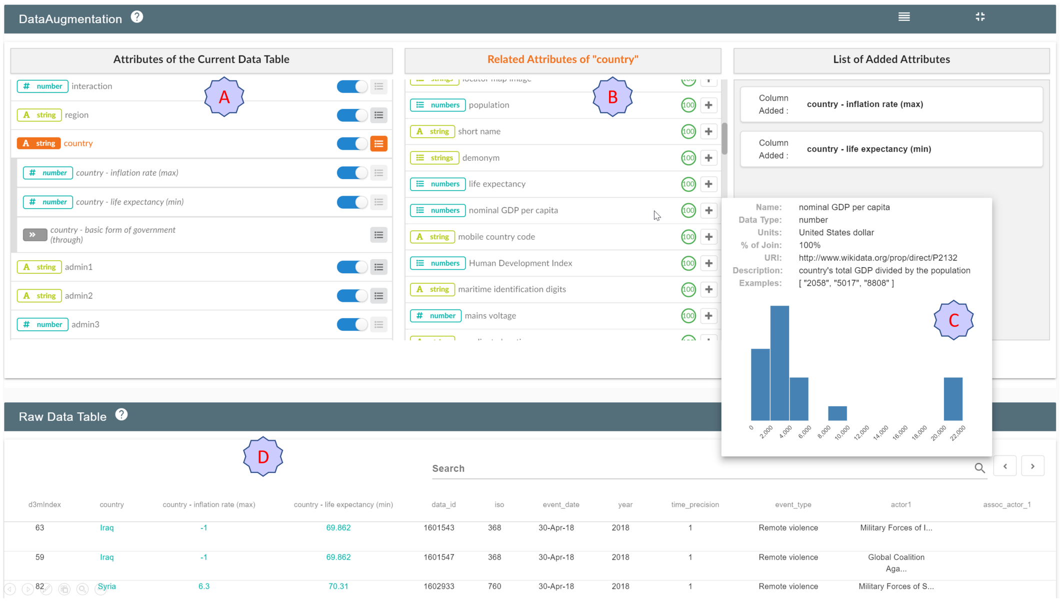
6.2 Tasks and Goals

Auger is developed as part of the DARPA Data-Driven Discovery and Modeling (D3M) program whose goal is to develop software infrastructure and algorithms to make automated machine learning accessible to general data scientists. Inspired by the observation that both the use of exploratory visualization and the use of advanced machine learning are fundamentally limited by the input data, Auger was developed to help a data scientist craft better predictive models by foraging for additional features to add to their dataset.

We conducted interview sessions with four teams within the DARPA D3M program developing applications for data exploration and predictive modeling. The goal of the interview sessions was to better understand how a tool like Auger can help the user to perform data augmentation for the purpose of improving machine learning model performance and accuracy. Each of the interviewed teams was shown an early implementation of Auger that was able to search for related attributes and return a list of them, but without any visualizations. The participants were then asked about what additional features and interactions would facilitate the discovery of the types of data that they were interested in for their applications.

I distilled a list of tasks from the interviews that would enable a user to meaningfully augment their dataset with additional attributes.

1. **View a list of joinable attributes for any existing attribute in the dataset.** Revealing the list of potential attributes that can be joined with a particular column in the dataset helps the user determine what external data is available for augmentation. It is a key aspect of the exploration process in



DataAugmentation

Attributes of the Current Data Table

# number	interaction
A string	region
A string	country
# number	*country - inflation rate (max)*
# number	*country - life expectancy (min)*
»	*country - basic form of government (through)*
A string	admin1
A string	admin2
# number	admin3

Related Attributes of "country"

numbers	population	100
A string	short name	100
strings	demonym	100
numbers	life expectancy	100
numbers	nominal GDP per capita	100
A string	mobile country code	100
numbers	Human Development Index	100
A string	maritime identification digits	100
# number	mains voltage	100

List of Added Attributes

Column Added : **country - inflation rate (max)**

Column Added : **country - life expectancy (min)**

Name:	nominal GDP per capita
Data Type:	number
Units:	United States dollar
% of Join:	100%
URI:	http://www.wikidata.org/prop/direct/P2132
Description:	country's total GDP divided by the population
Examples:	["2058", "5017", "8808"]

Raw Data Table

d3mIndex	country	country - inflation rate (max)	country - life expectancy (min)	data_id	iso	event_date	year	time_precision	event_type	actor1	assoc_actor_1
63	Iraq	-1	69.862	1601543	368	30-Apr-18	2018	1	Remote violence	Military Forces of I...	
59	Iraq	-1	69.862	1601547	368	30-Apr-18	2018	1	Remote violence	Global Coalition Aga...	
82	Syria	6.3	70.31	1602933	760	30-Apr-18	2018	1	Remote violence	Military Forces of S...	

Figure 6.2: The full system showing an analysis session on the ACLED dataset. **A** shows the initial list of attributes in the table as well as augmented attributes, like the maximum inflation rate recorded for each country. **B** shows the list of attributes related to the Country column as discovered from Wikidata. The attribute of the returned attribute is encoded in a symbol, a color, and a label on the left side of each row. The donut chart in each row shows the estimated amount of rows of the dataset that will have data available in the connected knowledge graph. When a row in either of the two columns is clicked, additional metadata about that attribute is shown in a popup, as seen in **C**. As attributes are added to the dataset, a preview of the table is updated in the raw data table as seen in **D**.

information foraging because it may reveal data that the user wasn't aware of.

2. **Analyze the properties of possibly related attributes *before* the join occurs:** Before joining a new attribute into a dataset, users will need to gather information about the new attribute to gauge the new attribute's impact on the quality of the dataset. For this reason, users should be provided with as much information as possible about the potentially joinable attributes before the join. This might include metadata, examples of the attribute, and any available information about the general distribution of the new data. In addition, it is important to communicate whether there will be any missing values in the joined attribute.

3. **Specify aggregations:** The relationships between entities in the user's dataset and entities in the external data source are varied - they could be one-to-one,

one-to-many, or many-to-many. In order to fit information encoded as plural relationships into a single row of data, aggregations must be specified. For example, a user might want to add the populations of a list of states into the their data. However, because there could be multiple measures of the state's population over the years, the user would need to perform an aggregation function over these results, such as MINIMUM, MAXIMUM, or AVERAGE depending on their analysis need.

4. **Connect through to additional data:** Attributes relevant to the user's analysis goals may not always be directly encoded as a property of an entity that exists on the initial dataset. For example, a user may want to augment a dataset of countries with the population of each country's capital. This information may not be directly accessible as a property of each country in the data repository. Instead, population size is linked to the entity that represents the capital, which in turn is linked to the country. To support these cases, users should be supported to join *through* intermediate attributes giving them fine-tuned control over how joins pass through multiple relationships.

6.3 Auger: A Visual Analytics System

In this section, I describe the function and design of Auger. I start by describing the design of each component of the interface and their interactions. Then, I give details on the implementation of the back-end connection to the Wikidata knowledge graph.

6.3.1 User Interface

Here, I describe the general look and feel of Auger and go into depth about how its design enables the tasks listed above. As shown in Figure 6.2, Auger has the following main interface components:

Column View: This view shows the attributes and their data types (represented using colored icons) that are present in the loaded data (see Figure 6.2-A). As the user discovers new attributes, they can remove any particular column if they find it

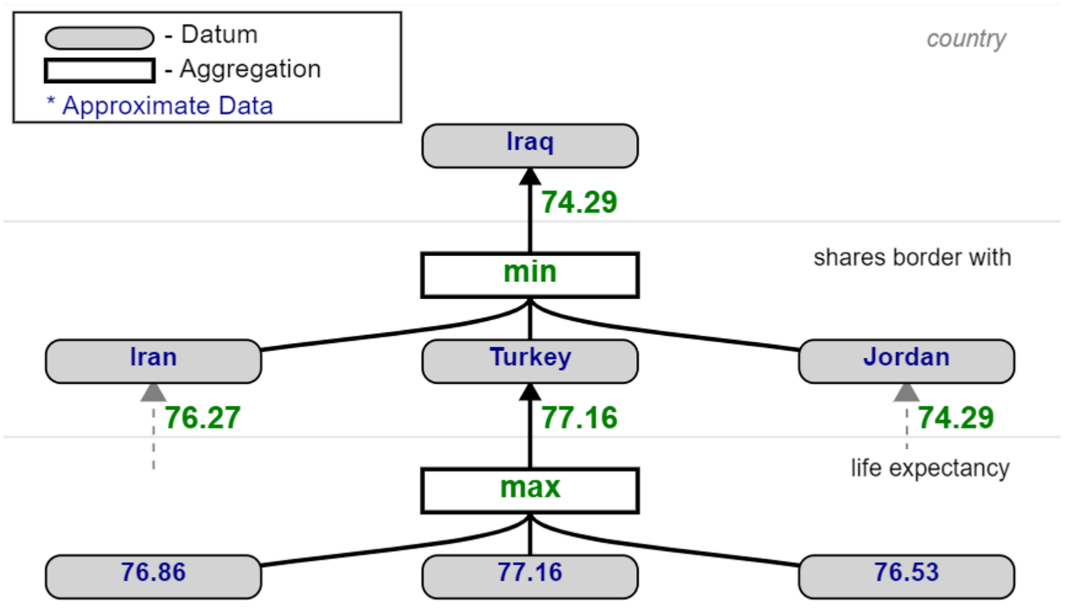


Figure 6.3: When a through join is specified, a dialog window pops up to allow the user to specify aggregations at each level of the join. To help them specify that query to the system, Auger shows them an example of the neighborhood in the knowledge graph that is queried for a single row of the table. In this image, the user would like to add the lowest life expectancy of any neighboring countries because they believe this will help their analysis of conflict. And for each country, several values are available for life expectancy, so the user selects the "max" operator. All six neighbors for Iraq, but only three are shown for simplicity of the illustration.

is not helpful to their analysis by a simple toggle button. Users can click on any row of this view to see the data distribution of that attribute shown as a histogram chart, along with any available meta-data describing that attribute (see Figure 6.2-C).

Related Attributes: A user can toggle a search for a collection of related attributes of any string-like attribute in the Column View (Task **T1**). Auger shows the retrieved attributes from the connected knowledge graph in a second column next to the list of attributes of the current table, as seen in Figure 6.2-B. Each row in this list represents a potential attribute that can be joined to the selected column. Similar to attributes in the column view, metadata and distributions of each attribute can be viewed in a tooltip, as seen in Figure 6.2-C. The numbers surrounded by colored circles, also known as donut charts, represent the estimated percent of rows of the dataset that have joinable data for that attribute in the knowledge graph. This information can help the user identify whether a related attribute has the potential to help their analysis *before* joining it to their data (Task **T2**). Both the estimated probability and the estimated attribute distribution are calculated from random sample of possible joinable data, since calculating the true values would require completing the full join for every row in the dataset. For more details on implementation techniques, see section 6.3.2.

Adding Attributes: To add any attribute, users click the plus symbol, and Auger will show a drop-down menu revealing various join-operations users can perform to join this attribute to the data. The available join operations correspond to the data type of the relationship. If the relationship is one-to-one, i.e. a country has a single head of government, then a single value will be retrieved for each row of the dataset. If the relationship is one-to-many or many-to-many, an aggregation must be specified. For example, if the attribute is a collection of numbers, such as set of populations recorded for a country, the user can select numerical aggregations such as *mean*, *max*, *min*, *sum*, or *variance* (Task **T3**).

It can be very costly to resolve all of the queries needed to join an additional attribute to an entire dataset; gathering the data from the knowledge graph requires the retrieval of many different parts of the knowledge graph rather than the retrieval of a single column from a relational database. Auger mitigates much of that time cost by only joining attributes for the top 10 rows of the dataset. Users are still able to glance at the dataset preview, as seen in Figure 6.2-D, and get an idea of the data that is being joined to the dataset. This lets the user explore the available attributes rapidly. When the exploration phase is completed, the full join can be executed.

Through Joins: In some cases, the user may want to join to data through an intermediate attribute. If the intermediate attribute has a one-to-one relationship with the rows of the original dataset, i.e. a country has one head of government, then the user can simply join the intermediate attribute first, and then use that as their starting point to seek more related attributes. If the intermediate attribute is a one-to-many relationship, then the user must specify multiple levels of aggregation. I call this type of aggregation a *"multi-hop aggregation,"* since it requires aggregating data across multiple hops on the knowledge graph.

As an example, suppose the user wants to determine if a country has a lower life expectancy than its neighbors. Consider how that would be calculated for a single country, Iraq. To gather the required data from the knowledge graph, the user must first join to all bordering countries, then connect through them to reach the life expectancies of those countries. Once all data is selected, aggregations must be specified to produce the desired value. In Wikidata, countries have multiple life expectancies recorded, so the user has to specify that they want the maximum life expectancy recorded. That value is then calculated for each bordering country (i.e. Iran, Turkey, Jordan, etc.). Then, they have to specify that they want the minimum among all bordering countries. Expressing this type of query is complicated enough to explain for a single value, let alone for an entire column. In Auger, the user is shown a simplified illustration of the topology of the knowledge graph to assist them in understanding the aggregations that the user must choose, as seen in Figure 6.3. By viewing the values that will get *passed through* the knowledge graph to ultimately calculate the attribute for one row of their data, users get reinforcement that the complex query they are building results in reasonable data.

Showing the augmented data: When additional attributes are successfully added to the data, the column view shows a card on the right showing the list of attributes that are added to the data. Added columns are also shown as a nested list in the first card below the parent attribute through which it was retrieved from the knowledge graph, as seen in Figure 6.2-A. If the added column is a string, it can then be used as a starting point to search for more related attributes. In that manner, the user can reach further and further away from the initial attribute set by iteratively adding attributes As attributes are added, the table view updates with the new list of columns and shows the user example values for the top 10 rows of the dataset (see Figure 6.2-D).

6.3.2 Backend Implementation

Here, I describe how our visual analytics system augments a dataset by translating the tasks supported by the user interface described above into valid queries over a knowledge graph. Auger gathers data from the connected knowledge graph in two different subroutines. In all examples in the chapter, Auger connects to Wikidata using the SPARQL query language.[2]

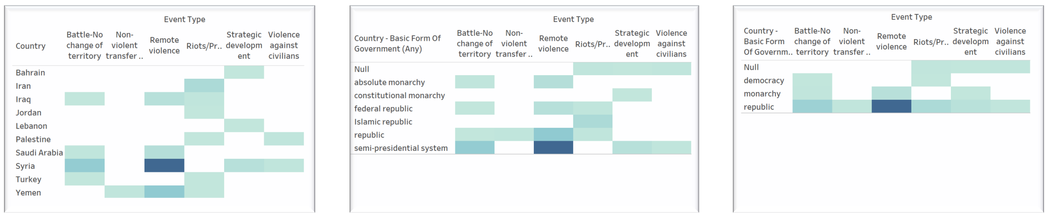
Finding Related Attributes: This subroutine receives a column of data as input (i.e. the Country column and all it's data, "Germany," "France," "Austria," etc.), and returns a list of attributes along with assorted metadata. In a knowledge graph, although these are all entities about a country, they might not share the same set of attributes (i.e. they might have different numbers of edges connected to the entities, and those edges might describe different types of relationships). In order to provide a user with a consistent set of attributes for these entities, we first need to find their commonalities. As such, the primary goal of this subroutine is to return the list of attributes that are available on as many of the entities as possible. Checking the related attributes of each entity to gather this list can take prohibitively long, especially if the input data contains thousands to millions of rows.

As an optimization step to make this subroutine support a user's interactive analysis, I employ a sampling-based technique to reduce computation time. With this optimization, some subset of the dataset is randomly sampled; empirically I have found that 20-30 rows are sufficient. Each sampled row is mapped to an entity on the knowledge graph, and a list of related attributes for that row is retrieved. Then, the lists for all the rows are compared, and the 50 attributes that appear on the most lists are returned. The percent of lists that the attribute appears on is passed on as metadata, and is visually encoded in Auger as a colored donut chart, seen in each attribute row in Figure 6.2-B.

Various metadata is returned, including three examples for each attribute. To get the data for the histograms, seen in Fig 6.2-C, an additional query is run for

[2]Example queries generated by the system for both subroutines are available in the supplemental material.

each of the 50 related attributes to get 20-30 examples through which to get a rough estimate of the distribution of the data. This subroutine runs queries in parallel and is only limited in speed by the concurrency limits on the knowledge graph API. For the types of exploration done in the user study and in examples given in this chapter, retrieving the list of related attributes from Wikidata typically takes 2-5 seconds.



(a) (b) (c)

Figure 6.4: Three steps of analysis of an armed conflicts dataset. Visualizations show the number of records of each event type by (a) country, (b) basic form of government, and (c) government type. Iniitally, only country is available. Auger is used to gather basic form of government first, then, after some analysis by the user, it is used to gather the government type to better understand patterns in the event types. Visualizations generated with Tableau.

Materializing Joins: This subroutine takes in join instructions and returns an augmented dataset with an additional column. The join instructions specify the path taken through the knowledge graph, along with any aggregation functions, such as "count" or "max." Auger builds a SPARQL query to crawl the knowledge graph to retrieve the desired data (see Fig 6.1). The time to retrieve data scales linearly with the number of rows and is limited by the concurrency limits on the knowledge graph API. While the user is interactively exploring attributes, Auger will only materialize the join for the top 10 rows of the data table to provide the user with a preview of the table in real time (see Fig 6.2-D). When the foraging is done, all joins are materialized for the entire dataset (rather than just a sample of rows) in the order in which they were constructed by the user, which takes around 10-15 seconds on the datasets explored in the user study in section 6.6.

Auger is implemented with a VueJS frontend web application and a NodeJS backend to handle all asynchronous calls. Queries are sent concurrently whenever possible. Source code is withheld to maintain anonymity and will be released in the final publication.

6.4 Use Case 1: Conflict Data

To demonstrate the value of column augmentation for insight generation, I present a use case in which Auger is integrated into a typical workflow for the popular visual analytics tool, Tableau. Suppose a political scientist wants to study the factors that lead to armed conflict. They download the Armed Conflict Location & Event Data Project (ACLED) dataset for April 2018, which contains hundreds of records of armed conflicts across many different countries and times [RLHK]. Initially, they load the data into Tableau to analyze relationships between countries and event types. They produce a heatmap, seen in Figure 6.4(a), to try to detect patterns, but find it difficult to visually group countries without more data.

They load their dataset into Auger to look for additional attributes that may be relevant to event types. When the system is first loaded, the user sees a list of the attributes in the uploaded table in Figure 6.2-A. They also can see the first five rows of the data below in Figure 6.2-D.

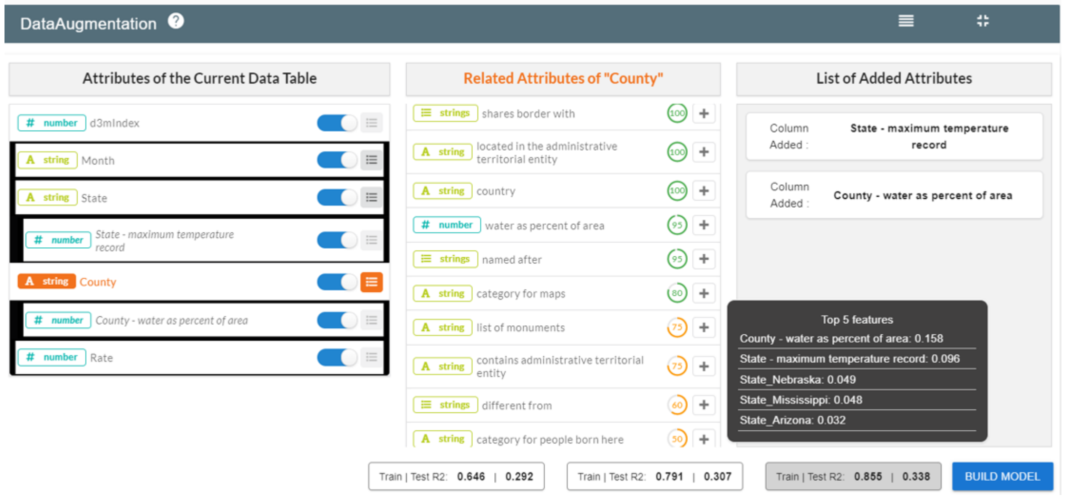
The political scientist believes the type of government could be related to the event type, so they click the *related attributes* button next to "Country" in Figure 6.2-A. The system returns a list of related attributes found by scraping Wikidata, as seen in Figure 6.2-B (**T1**). The user scans through the list of related attributes, looking for information about the type of government. They see that Wikidata holds the *basic form of government* for each country. The tooltip for that attribute, seen in Figure 6.2-C, provides more information to affirm that this holds the right data by showing the user examples of the attribute, like "unitary state," "federal republic," and "absolute monarchy" (**T2**). The user adds this attribute to the dataset. While looking for the form of government, the user also notices additional interesting columns for analysis and joins them for future analysis, including the max *inflation rate*, minimum *life expectancy*, mean *nominal GDP per capita*, and the mean *Human Development Index* (**T3**).

The user loads this next iteration of the dataset into Tableau, and generates the same visualization but with the basic form of government replacing the country name, seen in Figure 6.4(b). While some patterns begin to emerge, the user notes that the basic form of government is too fine-grained for the type of analysis they want to do, since most categories only have one or two countries in them. They return to Auger to see if there is any higher-level categorization available for that column by searching for attributes off of that attribute (**T4**). They find that Wikidata holds a *subclass of* attribute for basic forms of government with examples including "democracy," "monarchy," "republic," and add that attribute using a through join before returning to Tableau to generate the final visualization, seen in Figure 6.4(c). Now, they can see a pattern in that conflicts concentrate on republics in the region, with democracies and monarchies having interesting patterns in their types of conflicts as well. This use case demonstrates how the integration of column augmentation into a traditional insight generation pipeline can unlock new analyses in-situ.

6.5 Use Case 2: Modeling Unemployment

In the second use case, I demonstrate the value of Auger for a separate type of visual analytics task - the generation of a predictive model. Many visual analytics systems have been constructed to enable domain experts to interact with and steer the generation of machine learning models on their data. Augmenting a dataset with new columns is one way in which a user can imbue domain knowledge into the modeling process. They may know that certain attributes can help the prediction while others may mislead. While it may seem unusual to augment a training set with additional attributes, the additional attributes found on the knowledge graph can likewise be added to any data that the resulting model is asked to predict on at test time when the model is deployed.

I modify Auger to be able to train a predictive model at any point in the


DataAugmentation
Attributes of the Current Data Table
number d3mIndex
string Month
string State
number State - maximum temperature record
string County
number County - water as percent of area
number Rate
Related Attributes of "County"
strings shares border with 100
string located in the administrative territorial entity 100
string country 100
number water as percent of area 95
strings named after 95
string category for maps 80
string list of monuments 75
string contains administrative territorial entity 75
strings different from 60
string category for people born here 50
List of Added Attributes
Column Added : State - maximum temperature record
Column Added : County - water as percent of area
Train | Test R2: 0.646 | 0.292
Train | Test R2: 0.791 | 0.307
Train | Test R2: 0.855 | 0.338
BUILD MODEL

Top 5 features
County - water as percent of area: 0.158
State - maximum temperature record: 0.096
State_Nebraska: 0.049
State_Mississippi: 0.048
State_Arizona: 0.032

Figure 6.5: A modified version of Auger used for both Use Case 2 and Task 2 of the user study. Users can build predictive models as they augment their dataset to help them discover attributes that help predict a target variable. This figure portrays the construction of regression models to predict the rate of unemployment in counties of the United States. The history of trained models is shown at the bottom of the column view. By mousing over a model, the user can see the five highest feature importance scores for that model, and the list of attributes used to train that model are highlighted in black.

data augmentation process (see Figure 6.5). When the user instructs Auger to build a model, Auger splits the data into a training and test set in the ratio of 0.8:0.2 and trains a random forest regressor. For each model, Auger reports the *R Squared* score on the training and test sets, as well as the feature importance scores of the five most important features.

Consider Andy is a public policy analyst who seeks to build a regression model on the unemployment rate dataset [une20] with 1200 data samples from the year 2015. Every row in the data is a county and contains a sparse set of attributes such as *County*, *State*, *Month*, and the response variable *Unemployment Rate*. Andy seeks to augment this data by adding meaningful columns to accurately predict the *Unemployment Rate*. When Andy first loads the data in Auger, an initial model is trained with an *R Squared* score of 0.646 on training and 0.292 on test set respectively. Andy seeks to improve the regression model's performance further by augmenting the base data using REMAP 's workflow and visual interface. Andy searches for related attributes of the variable *State* from the Column View.

In response, Auger shows a list of related attributes that Andy may consider to add to the data. From these attributes they click on a set of interesting attributes such as *shares border with*, *head of government*, *inflation rate*, etc. to see the distribution of values, and its metadata as a text view.

From these set of attributes Andy thinks that the attributes *Inflation Rate* and *GDP per capita* are good attributes to predict the *unemployment rate* and thus decides to add them to the data using the join operation *Median*. They notice that the newly added column is displayed on the Table View. However looking at a few rows of the table Andy finds quite a few cells that are empty indicating that the joined data may have missing values. Nonetheless, Andy clicks a button to construct a new regression model using the augmented data. Andy notices that the *R Squared* score marginally improves (new score: 0.712 and 0.301 on training and test set respectively). They hover the mouse over the model metric card to see the list of "Top 5" attributes with their weights utilised in the regression model. While Andy expected to see a substantial improvement in the model performance, they infer that the marginal improvement is probably due to missing values in the data after the join operation.

Motivated to improve the model further, Andy searches for columns that may directly help to predict the unemployment rate per county. Based on prior knowledge, Andy understands that the population of a state may be directly proportional to the unemployment rate, and they add that attribute. In the process of searching for other relevant columns, Andy notices the column *Maximum temperature*. Inquisitive to see if a temperature of a state is correlated with its *unemployment rate* they add it to the table. After constructing a new regression model, Andy notices that the *R Squared* score improved substantially from 0.712 to 0.855 and from 0.301 to 0.338 for the training and test dataset respectively. Happy with the progress so far they decide to remove any column that may not be contributing to the prediction task. First they remove the column *Month* (by triggering the slider on the Column View) and then triggers Auger to construct a new regression model. As expected Andy notices that the *R Squared* score did not change. Next to investigate the contribution of the column *Maximum temperature*, they remove it from the data. Andy notices that the *R Squared* score of the new model dropped marginally meaning that the column *Maximum temperature* was helpful in improving the model. They add the column back to the data.

In a short time, Andy has constructed a regression model with substantially better R2 score than the model trained on the base data. They have also gained insight into which attributes are relevant to their modeling problem, which may help their understanding of the resulting machine learning model.

6.6 User Study

I evaluate Auger in a user study. The purpose of the evaluation is to validate Auger both in terms of its usability and its effectiveness in helping a user improve a machine learning model through data augmentation. Specifically, I hypothesize that:

- **H1**: Auger allows users to accurately join external data given a written description of that data.
- **H2**: Auger is able to help users discover additional data that can improve the predictive quality of machine learning models trained on the dataset.

We recruited 6 participants (3 Female, 3 Male), between the age of $23 - 36$. We required each participant to have at least an elementary knowledge of machine learning and data analysis. Due to COVID-19, we were not able to conduct an in-person study and were limited in the amount of participants we could recruit. We conducted an online study using Bluejeans[3]. The participants interacted with Auger on their own computer while sharing their screen. We provided the participants with a url to our system that was hosted on our local machine that is exposed using the Ngrok remote tunneling software[4]. The study took approximately $50 - 60$ minutes and we compensated the participants with a \$10 Amazon gift card.

[3]https://www.bluejeans.com/
[4]https://ngrok.com/

6.6.1 Study Design

Before the study we asked participants to fill out a background information questionnaire regarding their name, age, gender, machine learning expertise and various use cases in which they use machine learning. We began the study by showing the participants a tutorial video of Auger, explaining the workflow, interface GUI elements, and its interaction capabilities that support various join operations. Next we asked the participants to perform three tasks, the first of which was a practice task to ensure that they were sufficiently knowledgeable about Auger to perform the experimental trials. We proceeded to the experimental sessions only when we observed that the participants were confident and able to use Auger on their own. In the next two tasks we asked the participants: (1) To add a set of specified columns to a given dataset. These columns can be added to the data using various join operations supported by Auger. (2) To freely augment the data such that they can improve the performance metric of a machine learning model (metric being a R Squared Score of a regression model). We used the Scikit Learn machine learning library [PVG$^+$11] to construct *Random Forest Regression* models, in the same manner as described in Section 6.5. We collected the following data from each experimental trial: (1) *Task competition time*, (2) *Task Accuracy*, (3) *Model performance metric* e.g., R Squared Score, and (4) *User ratings* collected using a post-study Likert-scale questionnaire.

6.6.2 Datasets

For the tutorial video and practice task I used a U.S. Census Bureau dataset containing data about poverty in different counties around the U.S. [pov18]. The dataset contains 3136 rows, and 6 attributes such as *FIPS*, *State*, *County*, *RUCCode*, and the number of residents living in the county under the poverty line. For the first task in our experimental session we use the IMDB Movies dataset [imd18] containing 500 movies and 28 attributes such as *Director-name*, *Duration*, *Movie-Title*, *Movie Cast Facebook Likes*, etc. The second task in the experimental session which required users to augment data and construct regression models, used the unemployment rate dataset [une20]. This dataset contained 1200 rows where each row corresponded to a county's poverty rate measured in a given month. This dataset contained only 5 attributes such as *Month*, *State*, *County*, and the *Unemployment-Rate* (dependent variable).

6.6.3 Tasks and procedure

Participants were first asked to complete a practice task. During the practice task, their answers were not recorded and their performance was not included in the analysis described below. For the practice task, using the Poverty dataset, we asked the participants to add three columns: (1) the area of the county, (2) the population of the state, and (3) the earliest inception date of any county that each county shared borders with. While the first two columns could be retrieved using a straight-forward *join by value* operation, the third column required the *join through* operation to search for the required column that was one "hop" away in the knowledge graph. For the first experimental task using the IMDB dataset, participants were given descriptions of four attributes to add: (1) the director's country of citizenship, (2) the number of awards received by the director, (3) the number of cast members in each movie, (4) the sum of the number of awards received by all the producers of each movie. Note that attributes 3 and 4 required the participants to *join through* several attributes.

The final experimental task gave participants 10 minutes to augment the Unemployment rate dataset with as many attributes as they'd like that they believed would aid in building an accurate regression model to predict poverty rate. At any point in the 10 minutes, participants could instruct Auger to build a regression model with the current dataset to give them insight into whether they were improving the predictive modeling process. Building a model took about 10-20 seconds, and participants would be shown the change in R Squared Score for each new model, as well as the weights of top "5" features used in the model. Random Forest regression models were used because their flexibility, speed, and accuracy met the constraints of the experiment.

6.6.4 Data Collection

After all the tasks were completed, participants were asked to fill out a post-study questionnaire (using Google Forms) that included: (1) Likert-scale rating questions on their use of the system, (2) a NASA-TLX [HS88] questionnaire to measure task difficulty, and (3) Open-ended descriptive questions asking users about the interface design, the workflow, and other responses related to improve the usability of the system[5]. The Likert-scale questions asked the participants to rate (in a scale of $1 - 7$, 1 being "strongly disagree") if they found the system: (1) Easy to learn, (2) Intuitive, and (3) Expressive for data augmentation. We used the open-ended descriptive feedback to qualitatively evaluate the usability and the interaction design of Auger for their tasks. Open-ended prompts included (1) *Describe your thoughts about REMAP* , (2) *Describe things you disliked about the system or the workflow. Elaborate on how you think it could have been improved*, and (3) *Describe your strategy or your process to augment the data for Tasks 1 and 2* With the consent of the participants, each session was video and audio-recorded. We encouraged the participants to verbalize their thoughts following a think-aloud protocol. Furthermore, to assess if data augmentation using Auger led to any change in the regression models performance we saved: (1) model metric (i.e., R Squared Score), (2) feature weights, and (3) predicted values from the model. We also saved user mouse-clicks to analyze the set of columns the user explored to augment the data.

6.6.5 Result and Analysis

I report three types of results: task performance, user satisfaction, and qualitative feedback.

6.6.5.1 Task Performance

To assess **H1**, I reviewed the number of attributes that participants correctly joined in Task 1. Five out of six participants properly joined all four expected attributes.

[5]Pre-experiment and Post-experiment surveys are attached as supplemental material

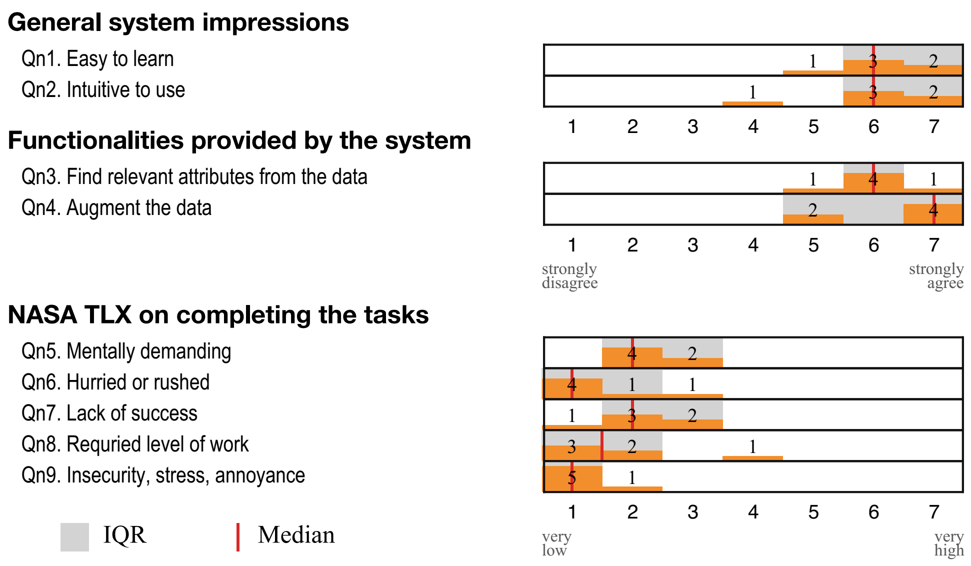


Figure 6.6: Participants' responses about the system (Qn1–Qn4, the higher the better) and the overall effort (Qn5–Qn9, the lower the better).

Only one participants missed one attribute when joining the number of awards received by the director. This indicates that in general participants were able to use Auger to accurately add new data attributes.

Next, to assess **H2**, I similarly reviewed the *R Squared* score improvement that participants achieved by augmenting additional data in Task 2. On average, participants improved the *R Squared* Score by 0.048 ($\sigma = 0.014$), starting with a baseline *R Squared* Score of 0.292. The area and population of the county are two most frequently joined attributes across all the participants, which is reasonable as these two attributes are usually the most influential for predicting unemployment rate.

As a result of the analysis, I **accept both H1 and H2** because all participants were able to successfully use Auger to explore and identify the correct attributes for data joining in Task 1, and to improve the predictive quality of a regression model in Task 1.

6.6.5.2 User satisfaction

I use the post-study questionnaire to assess user satisfaction of the data augmentation process in Auger. Figure 6.6 shows the results of the questionnaire responses.

Although the participant size is too small to infer statistical significance using quantitative analysis, I do observe that participants found Auger easy and intuitive to use based on the Likert scale user ratings (Figure 6.6, **Qn1,2**). Participants were also satisfied with the two main functionalities provided by Auger, finding relevant attributes from the data and joining additional data (Figure 6.6, **Qn3,4**). The mean ratings of **Qn1–4** were all 6 or above. Furthermore, in analyzing user satisfaction related to the overall process (Figure 6.6, **Qn5–9**), I found that participants were generally satisfied with the process of conducting the tasks, as the mean ratings of **Qn5–9** are all 2 or lower.

6.6.5.3 Qualitative Feedback

To assess **H1** and **H2** from a qualitative perspective, I analysed participants' descriptive feedback collected from post-study interviews. I also observed participants' workflow in using Auger from audio and video recordings of their computer. I report the following three main qualitative user responses from the study:

Intuitive workflow: All the participants fount Auger's workflow intuitive to find and augment relative data attributes. The primary justification for Auger's intuitiveness is that Auger provides a visual interface that is easy to infer and for users to search and add relevant attributes from the data. As *P6* noted, "*I liked the interface of this tool because it was easy to navigate and add/remove columns to the base data.*" The intuitiveness helped most participants to easily perform the requested tasks, especially Task 1. For example, *P1* described his experience in doing Task 1 as "*just looked at the question and implemented it*" and "*it was pretty straight forward.*"

Two major strategies in model construction: When conducting Task 2, I observed two common strategies used by the participants: (1) searching for relevant attributes based on existing or prior knowledge and (2) exploring all possible combinations of available attributes to find one that results in improvement in model performance. Some participants relied on their existing domain knowledge for augmenting the data with new attributes. This was summarized nicely by *P5*, "*I was actually using my domain knowledge. Thinking which factors can improve unemployment rate predictions, and then tried to select the variables based on the choices I had. This actually helped me improve the model's performance.*" However, a few participants who may lack prior knowledge about the data, instead explored the dataset and tried out all possible combinations of attributes to find a model that is a significant improvement over the model trained on the base data. "*I selected columns one by one to see the change of R-squared score. By doing so, I was able to filter the variables that decrease R-square. In the end, I was able to get a relatively high R-squared model*" (*P6*). The rest of the participants adopted a combination of both strategies. For example, *P2* described his strategies as follows, "*Thought a bit about what could boost performance, then I did some trial and error (manual feed-forward selection of attributes).*"

Need for more powerful system features for expert users: Participants with visual analytics backgrounds expected editable visualizations to represent or encode feature distribution differently. "*It would be cool to add additional ways of visualizing attributes*" (*P2*). What's more, participants with database backgrounds wanted more details about queries of fetching data to increase flexibility. "*Maybe consider providing the real query of fetching the dataset to the expert users to give them more clear sense and allow them to change the query*" (*P1*). Similarly, participants with more expertise in machine learning sought more control over the model building process. "*May be add hyperparameter tuning for the model building phase, such as L1, L2 norm or change learning algorithm of the model*" (*P2*). All these suggestions are valuable for guiding our further improvement in generalizing Auger to tackle various problem domains.

6.6.6 Limitations

While our user study results support both **H1** and **H2**, the results should be read in the light of the study limitations. The number of the participants for our study are limited, which might be a confounding point of the results. As we conducted the study online, the uncertainty existing in the online setup, such as internet connection, might also confound the results. To improve the performance of the system and the fluidity of user interaction, I limited the number of related attributes that participants could fetch for each parent attribute. Although it did help participants with a smoother user experience during the study, participants also complained about the restriction as it limited their performance of tasks, especially Task 2. Despite the limitations, the user study does help us to better identify the limitations in our system design and improve the system with new features.

6.7 Discussion

Through the user study and use cases, I have demonstrated the efficacy of information foraging using knowledge graphs within visual analytics systems. The generality of Auger suggests that data augmentation could added into traditional visual analytics system workflows to improve the outcome of any embedded task. Many fruitful avenues of research arise when considering how to apply information foraging to the full space of use cases which are served by visual analytics systems.

6.7.1 Mental Models of Data Augmentation

Knowledge graphs can be difficult to reason about, especially when they are used to find data corresponding to an entire column of a dataset. In Auger, I address this by showing the user previews of the joined dataset, as well as visualizations of the distribution of joined data and an example of the portion of the knowledge graph that is used to construct a single value (see Figure 6.3). This approach was based on our conversations with designers of visual analytics applications for building predictive models. A better understanding of the user's mental model of the knowledge graph and the joining process is needed to generalize this approach to other use cases. I generally aimed to hide the complexity of the underlying knowledge graph, but it may be the case that more direct exploration of the knowledge graph is helpful for some cases.

6.7.2 Design Space for Interactions with Knowledge Graphs

In Auger, the channel between the user and the knowledge graph is limited to the two subroutines shown in section 6.3.2, which allow the user to see lists of related attributes and then construct queries given that information. However, there is more potential to improve the user's control over the process and address edge cases by expanding the set of interactions between user and knowledge graph. Automated processes in Auger could be replaced by collaborations between user and system.

For example, ambiguities in the entity resolution used to retrieve a list of related attributes can be solved by user interaction. A list of countries could refer to the governmental entities they describe, or they could refer to the national soccer teams participating in the World Cup; there will always exist cases where disambiguating the type of a column will require domain expertise from a user.

The user could also benefit from more fine-grained control over the process of building aggregation queries. Users may want to join timestamped data, which would necessitate some specification from the user of how to parse and interpret temporal columns. Spatial data offers an additional potential, as the user might want to use geographical data in their dataset to search for the closest weather station or other geographically tagged entity. There are many types of queries that might necessitate different user interactions than have been used in previous visual analytics systems.

6.7.3 Scalability

In order for Auger to operate at the speed necessary for exploration, it makes use of only samples of the data, rather than the entire dataset . When gathering the list of related attributes for a column, it samples only a subset of values from that column. Those related attributes are then shown with estimations of their distribution as well as what percent of rows have a joinable value. While I use a sampling-based approach, there is plenty of opportunity to develop methods that provide better estimates or allow for joining to more data in less time. The database community is working on tools for query optimization over knowledge graphs [MAG+20], which will help. But specific indices, query optimizations, and data structures should be explored by the visual analytics community, the same way that many advancements in querying tabular data have been made to allow for faster data exploration [LKS13, WCL+18, PSSC16, LJH13, TLW+19].

6.8 Conclusion

In this chapter, I presented Auger, a visual analytics system for explorative information foraging using knowledge graphs. Knowledge graphs offer a wealth of information on a broad array of topics that could be used to improve the outcome of embedded tasks of visual analytics systems. In our use cases, I demonstrated that the data gathered through Auger could result in better predictive models and better insight generation on two datasets. And in our experiment, I showed that Auger is simple to learn and use, as all six of our participants were able to effectively explore and join data on multiple datasets within a 60 minute session. By crafting new attributes for their dataset, users can utilize their domain expertise to provide more information for the learning algorithm to use to find an appropriate model.

Chapter 7

Discussion

In chapter 1, I posited that visual analytics can help bridge the gap in applied machine learning problems because it allows a user to make up for vulnerabilities in learning algorithms. In chapters 3, 4, 5, and 6, I described four different systems that each showed one way in which visual analytics could be used in that way. In this chapter, I start by explaining how these four systems can be taken together as the foundations of a theory that is applicable to future visual analytics research. Then, I outline a set of guidelines for the visual analytics community based on this experience for transitioning from system building to theory building. Lastly, I list the ramifications of this work on the future of applied machine learning, and describe avenues for future work in that direction.

7.1 Bridging the Gap

In this dissertation, I have argued that visual analytics can improve outcomes of applied machine learning tasks. In the previous chapters, I presented four different systems that demonstrated examples. In this section, I aim to elaborate on the common thread between those systems as media for filling the gap in machine learning algorithms with human intervention. In particular, I claim that these systems taken together form a contribution to the theory of visual analytics for machine learning.

I described some visual analytics theory in the related work, and the workflow for exploratory model analysis described in chapter 3 is an incremental addition on that theory - specifically that the modeling process should be brought into the visual analytics tool as early as possible so that we can reap some benefit from visual exploration. But this has been done, piece meal, by previous systems. If we look at the related work section and see the set of visual analytics tools that help with model building and construction, they bring modeling into the visual analytics workflow. My work in chapter 3 differentiates itself by pointing out that those systems work only with a single model type, whereas our general approach lets a user search through learning algorithms as well. This is a small difference. But the work described in that chapter led me to investigate a broader hypothesis that led to the other three works described in this dissertation.

In the work leading to this dissertation, I looked into the ways that visual analytics was empirically used to improve machine learning outcomes. The value of the visual analytics systems found in the literature was large enough to justify the arduous and expensive task of building those systems. But there was a a dearth of theory that would explain why visual analytics was particularly helpful. It's often suggested that there's utility in having a user have a hand in building a model, feel its weight and timber. That way, the user will have a better trust and understanding when deploying that model. It could be that the only role of human in the loop systems is to provide a better explanation of the automated process to the decision-maker.

However, in this thesis, we posit that there are completely quantitative reasons why the user needs to be involved in building a model in applied machine learning. This isn't a specious statement; in internal discussions, some machine learning researchers involved in the DARPA Data Driven Discovery of Models project have suggested that, given sufficient resources and clean data, an automated model search will always outperform a human-in-the-loop model search. But in our early work in building a visual analytics system for model selection, described in chapter 3, we showed that wasn't the case. In a blind evaluation run by the National Institute of Standards and Technologies (NIST), participants were able to select a better model than machines in both regression and classification tasks. And interestingly enough, the *Snowcat* system does not feature complex visualizations or model steering capabilities. All that the participants had at their disposal were cross-linked data visualizations that facilitated explorations of the training data and comparisons of the predictions of multiple models. A very difficult problem was solved with seemingly easy means. This drove me to look for a reason why automated machine learning model searches might fail. And it demanded further isolated study on other ways in which a human could account for vulnerabilities in learning algorithms.

This led to the three systems described in chapters 4, 5, and 6. The applied machine learning problems they addressed were targeted because they each reflect a different assumption in the learning theory that must be present to guarantee performance. While the methods for validation varied from use cases to blind user studies, each work provided some new evidence for the value of visual analytics in its respective applied domain. In each case, the user's goals were met better than a strictly automated solution. The theory of involving the user to compensate for vulnerabilities in learning algorithms suggested that the human and the machine could work collaboratively towards solving those goals using visual analytics as a medium, and that was borne out in the validations of the four tools.

7.2 Guidelines for Transitioning from Systems to Theory

Ideally, when building a new theory for a nascent field, one would first build a mathematical model, and then create a set of controlled experiments that poke and prod to provide evidence for and address any skepticism of that model's assumptions. But for visual analytics, our methods for evaluation are still mostly shifting sands, as the primary research venue for visual analytics, IEEE Visual Analytics for Science and Technology (VAST) is only about a decade old. We haven't agreed on how to build theory yet, and what stands for convincing validation. That is at least partially because it is incredibly difficult. Evaluating visual analytics involves evaluating both the human experience of a tool or technique as well as its downstream effect on the outcome of the analytical session. And the computational tools of analytics are being reinvented literally every day as new machine learning research comes out on arXiv at a more rapid pace than any science in human history. No standard testbed exists because analytics is such a broad topic. Similarly, the contributions for the human side of visual analytics lack clear metrics of efficacy due to the inability to separate trust, learning, usability, background, or visual literacy from one another to conduct a controlled study.

As a result, most contributions to visual analytics found in conferences and journals are not theory or techniques, but are systems applied to a specific target audience and problem. If no other evaluation instrumentation can be agreed upon, we can still build a system, put a relevant user in front of that system, and observe and monitor their behavior or their competency at tasks. This is the bedrock of progress in our field. But each new piece of evidence requires the construction of a new system, which can be very expensive as visual analytics tools typically require complex engineering across many different technologies. In order to improve the pace of our research and the generalizability of our results, we need to find a different way. And we need to make sure that the systems that we do build are designed to isolate generalizable contributions as much as possible. This leads me to the following guidelines on transitioning from expensive system-building to rapid experimentation.

1. **G1: Start the design process off with a clearly stated hypothesis.** Many visual analytics projects begin with an interesting domain problem, and they proceed as design studies for applying visual analytics to that domain problem. I posit that the traditional design study methodology outlined by Sedlmair et. al. [SMM12] is not alone an appropriate guideline for visual analytics applications. Compared to information visualizations, the design space for visual analytics is not as well understood and there are many dimensions of that design space that can be very expensive to design in but offer little

generalizable value, such as the choice of progressive or synchronous updates between backend and frontend. To motivate the generalizability of results, then, I recommend that each design study begins with a narrow hypothesis of what will be tested.

2. **G2: Remove extraneous features in order to control for the independent variable.** Most visual analytics systems are designed to be as effective as possible for their target audience. They often contain many different elaborately designed features. These features are often linked together in coordinated multiple views. As a result, it becomes almost impossible to control for any independent variable being studied. If a user enjoys a system or is able to meet their goal using the system, which feature was responsible? Even if the user expresses which feature was helpful, with cross-linked components in the UI, it can be hard to know if there wasn't some unseen function that actually led them to complete their goal. Except in application studies where a tool is actually going to be deployed and used, every effort should be made to pare down features that confound the study of the hypothesis.

3. **G3: Design smaller experiments to study individual techniques.** When a complex visual analytics theory is being tested, we may not have the appropriate instrumentation to test it all at once. Multiple smaller-scale experiments with less expensive technological burden on the researcher can be a more prudent approach. They allow the researcher to analyze one variable at a time, and if the experimental hypotheses are strongly confirmed, they can provide more rationale for the design of a larger experiment.

4. **G4: Build integrative mathematical models of computation, visualization, perception, and cognition.** The analytical loop between human and machine is the composition of several smaller loops that have been studied in different fields. Tying these loops together can provide better starting points for our theory because it builds on the brilliant contributions of older fields. Early visualization research integrated cognitive psychology, such as power

laws, into theory about how visualization was perceived [VW05]. More recent work has tied computation theory [CFEC17] and information theory [CJ10] into building an understanding of the roles of human and machine in a collaboration. This work is hopefully a contribution integrating learning theory into this gap as well.

These guidelines address a common need in visual analytics research: to conduct more impactful research at lesser expense. I believe that this is a goal worth working towards, and hope that other efforts to reform the state of evaluation, such as the EVIVA-ML and BELIV workshops at IEEE VIS continue to see broad support and energetic discussion.

7.3 Future Work

My primary focus in continuing work in applied machine learning is to find a broader impact for visual analytics. Machine learning is growing at a breakneck pace in research output and user base. In both research and daily use, visualization is a frequent tool. Almost every poster presentation at any machine learning conference has an interesting, well-thought-out visualization communicating the insight of their research. And most consumer-facing analytics tools use basic visualizations as their medium of communication with the user. However, neither of these visualizations are typically built by visualization researchers. Our field tends to not be able to penetrate to the broader fields of data science and computer science.

I attribute this to the complexity, nuance, and possible obtuseness in the announced contributions of our work. Since novel visual analytics research often takes the form of an elaborate system applied to a single use case, those outside our discipline have little reason to believe that there is something of value to learn in exchange for the investment of learning that domain problem. If, instead, our contributions were more atomic, more clearly and succinctly stated, and more convincingly evaluated, they might be adopted by a broader audience.

I believe that the thesis defended in this dissertation points to some potentially impactful yet simple visual analytics techniques. For example, vulnerability **V1**, the domain mismatch between training and testing examples, is a well known issue in machine learning with a plethora of literature about how to mitigate it, such as concept drift [ŽPG16]. However, you still typically need to identify the type of mismatch you have, something that visualization should always be able to help with. I believe that this can be solved in a generic way by simply providing small, exploratory views of the training and testing set, overlaid with the model's predictions. This technique proved useful in chapter 3, but it needs to be isolated and tested in a more controlled environment. If the test is successful, it should be easily deployed as an addition to machine learning libraries or as a jupyter notebook widget.

I also plan on continuing to investigate the ways in which data augmentation can act as a medium for a domain expert to communicate their expertise to a learning algorithm. For example, by adding in a "weather" attribute on a dataset, the user is constraining the model to account for the relationships present between "weather" and the other attributes. However, there is little research into the types of interactions that are helpful in a user crafting these attributes. In chapter 6, we show just one way, embedded in a larger application. A sequence of smaller experiments comparing techniques would have broader applicability to other analytics environments.

Chapter 8

Conclusion

In this dissertation, I defend my thesis that visual analytics systems can improve the performance of deployed models in applied machine learning tasks by allowing the user to compensate for vulnerabilities in machine learning paradigms.

I identify four vulnerabilities based on the assumptions found in the empirical risk minimization paradigm. **V1** claims that a misalignment between training set and testing set can result in unexpected behavior in deployment. This can be addressed with visual exploration of the training set as well as visualizations of the model's predictions and apportionments of errors. **V2** claims that the actual risk is often much more complex than the loss function used in ERM. Instead, domain experts should be able to drive the parameter space search to find appropriate regions that perform well according to their understanding of risk in deployment. Visualizations of the model's behavior during training can also help a user judge whether a model is learning heuristic properties that may not be encoded into the machine learning algorithm's optimization. And lastly, **V3** states that some problems will not be solveable without more information. By adding additional attributes to a dataset and thus adding more domain-specific information, a user might be able to solve that vulnerability.

For each vulnerability, I design a system to demonstrate how visual analytics is an appropriate medium for bridging the gap between human goal and machine learning. Through these systems, I show existential evidence that visualization provides a unique opportunity to address difficult applied machine learning problems. By acting as a medium between human and machine, visual analytics can address some of the most significant issues in applying machine learning algorithms to real-world data.

Bibliography

[AAB$^+$15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[ABK$^+$07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[ACD$^+$15] Saleema Amershi, Max Chickering, Steven M Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 337–346. ACM, 2015.

[ACD18] KM Annervaz, Somnath Basu Roy Chowdhury, and Ambedkar Dukkipati. Learning beyond datasets: Knowledge graph augmented neural networks for natural language processing. *arXiv preprint arXiv:1802.05930*, 2018.

[AGM$^+$18] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian J. Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. *CoRR*, abs/1810.03292, 2018.

[AHH$^+$14] Bilal Alsallakh, Allan Hanbury, Helwig Hauser, Silvia Miksch, and Andreas Rauber. Visual methods for analyzing probabilistic classification data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1703–1712, 2014.

[ALA$^+$] N. Andrienko, T. Lammarsch, G. Andrienko, G. Fuchs, D. Keim, S. Miksch, and A. Rind. Viewing visual analytics as model building. *Computer Graphics Forum*, 0(0).

[Ans73] Francis J Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, 1973.

[AWD12] Anushka Anand, Leland Wilkinson, and Tuan Nhon Dang. Visual pattern discovery using random projections. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 43–52. IEEE, 2012.

[BGNR16] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016.

[BISM14] Matthew Brehmer, Stephen Ingram, Jonathan Stray, and Tamara Munzner. Overview: The design, adoption, and analysis of a visual document mining tool for investigative journalists. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2271–2280, 2014.

[BJY$^+$17] Alsallakh Bilal, Amin Jourabloo, Mao Ye, Xiaoming Liu, and Liu Ren. Do convolutional neural networks learn class hierarchy? *IEEE* *transactions on visualization and computer graphics*, 24(1):152–162, 2017.

[BJY+18] Alsallakh Bilal, Amin Jourabloo, Mao Ye, Xiaoming Liu, and Liu Ren. Do convolutional neural networks learn class hierarchy? *IEEE Transactions on Visualization and Computer Graphics*, 24(1):152–162, 2018.

[BLBC12] Eli T Brown, Jingjing Liu, Carla E Brodley, and Remco Chang. Dis-function: Learning distance functions interactively. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 83–92. IEEE, 2012.

[BND13] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. Methods for exploring and mining tables on wikipedia. In *Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics*, pages 18–26, 2013.

[BOH11] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, (12):2301–2309, 2011.

[Boh16] Dieter Bohn. Google home: a speaker to finally take on the amazon echo. `https://www.theverge.com/2016/5/18/11688376/google-home-speaker-announced-virtual-assistant-io-2016`, 2016. Retrieved April 30, 2020.

[Bri15] Denny Britz. Recurrent neural networks tutorial, part 3 - backpropagation through time and vanishing gradients. `http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/`, 2015. Accessed: 2017-07-15.

[BYC13] James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pages 13–20, 2013.

[CD92] MA Chase and GM Dummer. The role of sports as a social determinant for children. *Research Quarterly for Exercise and Sport*, 63:18–424, 1992.

[CD19] M. Cavallo and Ç. Demiralp. Clustrophile 2: Guided visual clustering analysis. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):267–276, 2019.

[CDW$^+$15] Isaac Cho, Wewnen Dou, Derek Xiaoyu Wang, Eric Sauda, and William Ribarsky. Vairoma: A visual analytics system for making sense of places, times, and events in roman history. *IEEE transactions on visualization and computer graphics*, 22(1):210–219, 2015.

[CE19] Min Chen and David S Ebert. An ontological framework for supporting the design and evaluation of visual analytics systems. In *Computer Graphics Forum*, volume 38, pages 131–144. Wiley Online Library, 2019.

[CFEC17] R Jordan Crouser, Lyndsey Franklin, Alex Endert, and Kris Cook. Toward theoretical techniques for measuring the use of human effort in visual analytic systems. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):121–130, 2017.

[CG16] Min Chen and Amos Golan. What may visualization processes optimize? *IEEE Transactions on Visualization and Computer Graphics*, 22(12):2619–2632, 2016.

[CHH$^+$19a] Dylan Cashman, Shah Rukh Humayoun, Florian Heimerl, Kendall Park, Subhajit Das, John Thompson, Bahador Saket, Abigail

Mosca, John Stasko, Alex Endert, et al. d3m Snowcat Training Manual. http://www.eecs.tufts.edu/~dcashm01/public_img/d3m_manual.pdf, 2019. Accessed: 2019-03-30.

[CHH+19b] Dylan Cashman, Shah Rukh Humayoun, Florian Heimerl, Kendall Park, Subhajit Das, John Thompson, Bahador Saket, Abigail Mosca, John Stasko, Alex Endert, et al. d3m Snowcat Training Video. https://youtu.be/_JC3XM8xcuE, 2019. Accessed: 2019-03-30.

[CHH+19c] Dylan Cashman, Shah Rukh Humayoun, Florian Heimerl, Kendall Park, Subhajit Das, John Thompson, Bahador Saket, Abigail Mosca, John Stasko, Alex Endert, et al. A user-based visual analytics workflow for exploratory model analysis. In *Computer Graphics Forum*, volume 38, pages 185–199. Wiley Online Library, 2019.

[CHK09] Michael J Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *Proceedings of the VLDB Endowment*, 2(1):1090–1101, 2009.

[Chu79] Russell M Church. How to look at data: A review of john w. tukey's exploratory data analysis 1. *Journal of the experimental analysis of behavior*, 31(3):433–440, 1979.

[CHW+08] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[CJ10] Min Chen and Heike Jaenicke. An information-theoretic framework for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1206–1215, 2010.

[CLKP10] Jaegul Choo, Hanseung Lee, Jaeyeon Kihm, and Haesun Park. ivisclassifier: An interactive visual analytics system for classification based on supervised dimension reduction. In *Visual Analytics Science and*

Technology (VAST), 2010 IEEE Symposium on, pages 27–34. IEEE, 2010.

[CLL$^+$13] Jaegul Choo, Hanseung Lee, Zhicheng Liu, John Stasko, and Haesun Park. An interactive visual testbed system for dimension reduction and clustering of large-scale high-dimensional data. In *Visualization and Data Analysis 2013*, volume 8654, page 865402. International Society for Optics and Photonics, 2013.

[CMS99] Stuart K Card, Jock D Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[CNE17] Nick Cramer, Grant Nakamura, and Alex Endert. The impact of streaming data on sensemaking with mixed-initiative visual analytics. In *International Conference on Augmented Cognition*, pages 478–498. Springer, 2017.

[Cou18] Council of European Union. 2018 reform of eu data protection rules, 2018. `https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en`.

[CPCS19] Dylan Cashman, Adam Perer, Remco Chang, and Hendrik Strobelt. Ablate, variate, and contemplate: Visual analytics for discovering neural architectures. *IEEE transactions on visualization and computer graphics*, 26(1):863–873, 2019.

[CPM$^+$18] Dylan Cashman, Geneviève Patterson, Abigail Mosca, Nathan Watts, Shannon Robinson, and Remco Chang. Rnnbow: Visualizing learning via backpropagation gradients in rnns. *IEEE Computer Graphics and Applications*, 38(6):39–50, 2018.

[CR98] Ed Huai-hsin Chi and John T Riedl. An operator interaction framework for visualization systems. In *Information Visualization, 1998. Proceedings. IEEE Symposium on*, pages 63–70. IEEE, 1998.

[CT05] Kristin A Cook and James J Thomas. Illuminating the path: The research and development agenda for visual analytics. 2005.

[CWT$^+$08] Bryan Chan, Leslie Wu, Justin Talbot, Mike Cammarano, and Pat Hanrahan. Vispedia: Interactive visual exploration of wikipedia data via search-based integration. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1213–1220, 2008.

[CXD$^+$20] Dylan Cashman, Shenyu Xu, Subhajit Das, Florian Heimerl, Cong Liu, Shah Rukh Humayoun, Michael Gleicher, Alex Endert, and Remco Chang. Auger: A visual analytics system for exploratory data augmentation using knowledge graphs. *In Revision*, 2020.

[CZGR09] Remco Chang, Caroline Ziemkiewicz, Tera Marie Green, and William Ribarsky. Defining insight for visual analytics. *IEEE Computer Graphics and Applications*, 29(2):14–17, 2009.

[DCCE19] Subhajit Das, Dylan Cashman, Remco Chang, and Alex Endert. Beames: Interactive multimodel steering, selection, and inspection for regression tasks. *IEEE computer graphics and applications*, 39(5):20–32, 2019.

[DEE$^+$13] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 541–552, 2013.

[DGH$^+$14] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

[DOL03] MC Ferreira De Oliveira and Haim Levkowitz. From visual data exploration to visual data mining: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):378–394, 2003.

[DSFG$^+$12] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 817–828, 2012.

[DVK17] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[EFN12] Alex Endert, Patrick Fiaux, and Chris North. Semantic interaction for sensemaking: inferring analytical reasoning for model steering. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2879–2888, 2012.

[Elm90] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[EMH18] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[EW16] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, 48, 2016.

[FAK$^+$18] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1001–1012. IEEE, 2018.

[Fek04] J-D Fekete. The infovis toolkit. In *Information Visualization, IEEE Symposium on*, pages 167–174. IEEE, 2004.

[FTC09] Rebecca Fiebrink, Dan Trueman, and Perry R Cook. A meta-instrument for interactive, on-the-fly machine learning. In *New Interfaces for Musical Expression*, pages 280–285, 2009.

[GBC$^+$15] Isabelle Guyon, Kristin Bennett, Gavin Cawley, Hugo Jair Escalante, Sergio Escalera, Tin Kam Ho, Núria Macia, Bisakha Ray, Mehreen Saeed, Alexander Statnikov, et al. Design of the 2015 chalearn automl challenge. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

[GBYH20] Michael Gleicher, Aditya Barve, Xinyi Yu, and Florian Heimerl. Boxer: Interactive comparison of classifier results, 2020.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[Gir15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[Gle13] Michael Gleicher. Explainers: Expert explorations with crafted projections. *IEEE Transactions on Visualization and Computer Graphics*, (12):2042–2051, 2013.

[Gle18] Michael Gleicher. Considerations for visualizing comparison. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):413–423, 2018.

[GMCR04] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian symposium on artificial intelligence*, pages 286–295. Springer, 2004.

[HBO$^+$10] Jeffrey Heer, Michael Bostock, Vadim Ogievetsky, et al. A tour through the visualization zoo. *Communications of ACM*, 53(6):59–67, 2010.

[HCH15] Ben Hixon, Peter Clark, and Hannaneh Hajishirzi. Learning knowledge graphs for question answering through conversational dialog. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 851–861, 2015.

[HFH$^+$09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[HG18] Florian Heimerl and Michael Gleicher. Interactive analysis of word vector embeddings. In *Computer Graphics Forum*, volume 37, pages 253–265. Wiley Online Library, 2018.

[HGVS14] Patrick Hoefler, Michael Granitzer, Eduardo E Veas, and Christin Seifert. Linked data query wizard: A novel interface for accessing sparql endpoints. In *LDOW*, 2014.

[HKBE12a] F. Heimerl, S. Koch, H. Bosch, and T. Ertl. Visual classifier training for text document retrieval. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2839–2848, 2012.

[HKBE12b] Florian Heimerl, Steffen Koch, Harald Bosch, and Thomas Ertl. Visual classifier training for text document retrieval. *IEEE Transactions on Visualization and Computer Graphics*, (12):2839–2848, 2012.

[HKN$^+$16] Alon Halevy, Flip Korn, Natalya F Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*, pages 795–806, 2016.

[HKPC18] Fred Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *arXiv preprint arXiv:1801.06889*, 2018.

[HME00] David C. Hoaglin, Frederick Mosteller, and John W. Tukey (Editor). *Understanding Robust and Exploratory Data Analysis.* Wiley-Interscience, 1 edition, 2000.

[Hoc98] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, April 1998.

[HS88] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier, 1988.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HSV$^+$17] Orland Hoeber, Anoop Sarkar, Andrei Vacariu, Max Whitney, Manali Gaikwad, and Gursimran Kaur. Evaluating the value of lensing wikipedia during the information seeking process. In *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval*, CHIIR '17, page 77–86, New York, NY, USA, 2017. Association for Computing Machinery.

[Hua96] Thomas Huang. Computer vision: Evolution and promise. 1996.

[Hun07] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[HZLL19] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 105–113, 2019.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[imd18] IMDB 5000 Movie Dataset. `https://www.kaggle.com/carolzhangdc/imdb-5000-movie-dataset`, 2018. Online; accessed 10-Mar-2020.

[JSH18] Haifeng Jin, Qingquan Song, and Xia Hu. Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 2018.

[JWC$^+$20] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug. `https://github.com/aleju/imgaug`, 2020. Online; accessed 19-Mar-2020.

[JZF$^+$09] Dong Hyun Jeong, Caroline Ziemkiewicz, Brian Fisher, William Ribarsky, and Remco Chang. ipca: An interactive system for pca-based visual analytics. In *Computer Graphics Forum*, volume 28, pages 767–774. Wiley Online Library, 2009.

[KAF$^+$08] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. Information visualization. chapter Visual Analytics: Definition, Process, and Challenges, pages 154–175. Springer-Verlag, Berlin, Heidelberg, 2008.

[KAKC18] Minsuk Kahng, Pierre Y Andrews, Aditya Kalro, and Duen Horng Polo Chau. Activis: Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, 24(1):88–97, 2018.

[Kar15] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`, 2015. Accessed: 2017-07-15.

[KBE14] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, 2014.

[KDS$^+$17] Josua Krause, Aritra Dasgupta, Jordan Swartz, Yindalon Aphinyanaphongs, and Enrico Bertini. A workflow for visual diagnostics of binary classifiers using instance-level explanations. *Visual Analytics Science and Technology (VAST), IEEE Conference on*, 2017.

[KDSG$^+$16] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment*, 9(12):1197–1208, 2016.

[KEV$^+$18] Bum Chul Kwon, Ben Eysenbach, Janu Verma, Kenney Ng, Christopher De Filippi, Walter F Stewart, and Adam Perer. Clustervision: Visual supervision of unsupervised clustering. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):142–151, 2018.

[KFC16] Minsuk Kahng, Dezhi Fang, and Duen Horng (Polo) Chau. Visual exploration of machine learning results using data cube analysis. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 1:1–1:6, New York, NY, USA, 2016. ACM.

[KFM71] Patrick Krolak, Wayne Felts, and George Marble. A man-machine approach toward solving the traveling salesman problem. *Communications of the ACM*, 14(5):327–334, 1971.

[KH09] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[KJL15] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015.

[KKE10] Daniel Keim, Jörn Kohlhammer, and Geoffrey Ellis. Mastering the information age: Solving problems with visual analytics, eurographics association, 2010.

[KL17] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730*, 2017.

[KLTH10] Ashish Kapoor, Bongshin Lee, Desney Tan, and Eric Horvitz. Interactive optimization for steering machine classification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1343–1352. ACM, 2010.

[KMSZ06] Daniel A Keim, Florian Mansmann, Jörn Schneidewind, and Hartmut Ziegler. Challenges in visual data analysis. In *Information Visualization, 2006. IV 2006. Tenth International Conference on*, pages 9–16. IEEE, 2006.

[KNS$^+$18] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2020–2029, 2018.

[KPHH11] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372, 2011.

[KS14] Gordon Kindlmann and Carlos Scheidegger. An algebraic process for visualization design. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2181–2190, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[KSS16] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 979–984. IEEE, 2016.

[KTB$^+$18] Alexander Kumpf, Bianca Tost, Marlene Baumgart, Michael Riemer, Rüdiger Westermann, and Marc Rautenhaus. Visualizing confidence in cluster-based ensemble weather forecast analyses. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):109–119, 2018.

[KTC$^+$19] Minsuk Kahng, Nikhil Thorat, Duen Horng Polo Chau, Fernanda B Viégas, and Martin Wattenberg. GAN lab: Understanding complex deep generative models using interactive visual experimentation. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):310–320, 2019.

[KTH$^+$16] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 17:1–5, 2016.

[KTS$^+$14] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[KV15] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015.

[KVV94] Michael J Kearns, Umesh Virkumar Vazirani, and Umesh Vazirani. *An introduction to computational learning theory.* MIT press, 1994.

[LB$^+$95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[LD11] David Lloyd and Jason Dykes. Human-centered approaches in geovisualization design: Investigating multiple methods through a long-term case study. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2498–2507, 2011.

[Lip16] Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

[LJH13] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[LKEW15] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

[LKS13] Lauro Lins, James T Klosowski, and Carlos Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans-*

actions on Visualization and Computer Graphics, 19(12):2456–2465, 2013.

[LL] Fei-Fei Li and Jia Li. Cloud automl: Making ai accessible to every business. https://www.blog.google/topics/google-cloud/cloud-automl-making-ai-accessible-every-business/. Accessed: 2018-03-29.

[LSC$^+$18a] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu. Analyzing the training processes of deep generative models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):77–87, Jan 2018.

[LSC$^+$18b] Mengchen Liu, Jiaxin Shi, Kelei Cao, Jun Zhu, and Shixia Liu. Analyzing the training processes of deep generative models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):77–87, 2018.

[LSD19] Phil Legg, Jim Smith, and Alexander Downing. Visual analytics for collaborative human-machine confidence in human-centric active learning tasks. *Human-centric Computing and Information Sciences*, 9(1):5, 2019.

[LSL$^+$17] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):91–100, 2017.

[LSY18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.

[LWLZ17] Shixia Liu, Xiting Wang, Mengchen Liu, and Jun Zhu. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, 1(1):48–56, 2017.

[LWT$^+$15] Shusen Liu, Bei Wang, Jayaraman J Thiagarajan, P-T Bremer, and Valerio Pascucci. Visual exploration of high-dimensional data through subspace analysis and dynamic projections. In *Computer Graphics Forum*, volume 34, pages 271–280. Wiley Online Library, 2015.

[LXL+18] Shixia Liu, Jiannan Xiao, Junlin Liu, Xiting Wang, Jing Wu, and Jun Zhu. Visual diagnosis of tree boosting methods. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):163–173, 2018.

[LXLZ15] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[LYBP16] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In *Advances In Neural Information Processing Systems*, pages 289–297, 2016.

[Lyn16] Matthew Lynley. Google unveils google assistant, a virtual assistant that's a big upgrade to google now. `https://techcrunch.com/2016/05/18/google-unveils-google-assistant-a-big-upgrade-to-google-now/`, 2016. Retrieved April 30, 2020.

[MAG+20] Aisha Mohamed, Ghadeer Abuoda, Abdurrahman Ghanem, Zoi Kaoudi, and Ashraf Aboulnaga. Rdfframes: Knowledge graph access for machine learning tools. *arXiv preprint arXiv:2002.03614*, 2020.

[MB19] Adam Millard-Ball. The autonomous vehicle parking problem. *Transport Policy*, 75:99–108, 2019.

[McC20] John P. McCrae. The linked open data cloud. `https://lod-cloud.net/`, 2020. Online; accessed 22-Apr-2020.

[MCZ+17] Yao Ming, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, and Huamin Qu. Understanding hidden memories of recurrent neural networks. *CoRR*, abs/1710.10777, 2017.

[MGB$^+$19] Hamid Mansoor, Walter Gerych, Luke Buquicchio, Kavin Chandrasekaran, Emmanuel Agu, and Elke Rundensteiner. Delfi: Mislabelled human context detection using multi-feature similarity linking. In *2019 IEEE Visualization in Data Science (VDS)*, pages 11–19. IEEE, 2019.

[Mil95] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[Mil18] Renée J Miller. Open data integration. *Proceedings of the VLDB Endowment*, 11(12):2130–2139, 2018.

[MLMP18] Thomas Mühlbacher, Lorenz Linhardt, Torsten Möller, and Harald Piringer. Treepod: Sensitivity-aware selection of pareto-optimal decision trees. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):174–183, 2018.

[MLR$^+$18] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34, 2018.

[MNNBA19] Diego Moussallem, Axel-Cyrille Ngonga Ngomo, Paul Buitelaar, and Mihael Arcan. Utilizing knowledge graphs for neural machine translation augmentation. In *Proceedings of the 10th International Conference on Knowledge Capture*, K-CAP '19, page 139–146, New York, NY, USA, 2019. Association for Computing Machinery.

[MP13] Thomas Mühlbacher and Harald Piringer. A partition-based framework for building and validating regression models. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):1962–1971, 2013.

[MSG17] K. Marino, R. Salakhutdinov, and A. Gupta. The more you know: Using knowledge graphs for image classification. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20–28, July 2017.

[MTH89] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[NHM+07] E. J. Nam, Y. Han, K. Mueller, A. Zelenyuk, and D. Imre. Clustersculptor: A visual analytics tool for high-dimensional data. In *2007 IEEE Symposium on Visual Analytics Science and Technology*, pages 75–82, 2007.

[NM13] Julia EunJu Nam and Klaus Mueller. Tripadvisorˆ{ND}: A tourism-inspired high-dimensional space exploration framework with overview and detail. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):291–305, 2013.

[Nor06] Chris North. Toward measuring visualization insight. *IEEE Computer Graphics and Applications*, 26(3):6–9, 2006.

[OSJ+18] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[PBD+10] Kayur Patel, Naomi Bancroft, Steven M Drucker, James Fogarty, Andrew J Ko, and James Landay. Gestalt: integrated support for implementation and analysis in machine learning. In *Proceedings of the*

23nd annual ACM symposium on User interface software and technology, pages 37–46. ACM, 2010.

[PC99] Peter Pirolli and Stuart Card. Information foraging. *Psychological review*, 106(4):643, 1999.

[PC05] Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of international conference on intelligence analysis*, volume 5, pages 2–4. McLean, VA, USA, 2005.

[PC20] Daniel S. Park and William Chan. SpecAugment. `https://ai.googleblog.com/2019/04/specaugment-new-data-augmentation.html`, 2020. Online; accessed 19-Mar-2020.

[PGCB13] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *CoRR*, abs/1312.6026, 2013.

[PHVG$^+$17] Nicola Pezzotti, Thomas Höllt, Jan Van Gemert, Boudewijn PF Lelieveldt, Elmar Eisemann, and Anna Vilanova. Deepeyes: Progressive visual analytics for designing deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):98–108, 2017.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org, 2013.

[POF01] Juan Antonio Perez-Ortiz and Mikel L Forcada. Part-of-speech tagging with recurrent neural networks. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pages 1588–1592. IEEE, 2001.

[pov18] Small Area Income And Poverty Estimates (SAIPE) Program state and county estimates. `https://www.census.gov/programs-surveys/saipe/data.html`, 2018. Online; accessed 10-Mar-2020.

[pow] Power BI. `https://powerbi.microsoft.com/`. Accessed: 2018-12-12.

[PS08] Adam Perer and Ben Shneiderman. Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 265–274. ACM, 2008.

[PSSC16] Cicero AL Pahins, Sean A Stephens, Carlos Scheidegger, and Joao LD Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE transactions on visualization and computer graphics*, 23(1):671–680, 2016.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PW17] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[PY09] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[qui] Quick, Draw! can a neural network learn to recognize doodling? `https://quickdraw.withgoogle.com/`. Accessed: 2019-03-30.

[RAHL18] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018.

[RAL+17] Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):61–70, 2017.

[RESC16] Eric D Ragan, Alex Endert, Jibonananda Sanyal, and Jian Chen. Characterizing provenance in visualization and data analysis: an organizational framework of provenance types and purposes. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):31–40, 2016.

[RLHK] Clionadh Raleigh, Andrew Linke, Havard Hegre, and Joakim Karlsen. Introducing acled: an armed conflict location and event dataset: special data feature. *Journal of peace research*.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[SBI+13] Michael Stonebraker, Daniel Bruckner, Ihab F Ilyas, George Beskales, Mitch Cherniack, Stanley B Zdonik, Alexander Pagan, and Shan Xu. Data curation at scale: the data tamer system. In *Cidr*, volume 2013, 2013.

[SC] Daniel Smilkov and Shan Carter. Tensorflow Playground. `https://playground.tensorflow.org/`. Accessed: 2019-03-31.

[SDC+17] Thomas Swearingen, Will Drevo, Bennett Cyphers, Alfredo Cuesta-Infante, Arun Ross, and Kalyan Veeramachaneni. Atm: A distributed, collaborative, scalable system for automated machine learning. In *Big*

Data (Big Data), 2017 IEEE International Conference on, pages 151–162. IEEE, 2017.

[SDV$^+$16] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[SGB$^+$18] Hendrik Strobelt, Sebastian Gehrmann, Michael Behrisch, Adam Perer, Hanspeter Pfister, and Alexander M Rush. S eq 2s eq-v is: A visual debugging tool for sequence-to-sequence models. *IEEE transactions on visualization and computer graphics*, 25(1):353–363, 2018.

[SGPR18] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):667–676, 2018.

[SHB$^+$14] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Möller. Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2161–2170, Dec 2014.

[She] Wade Shen. Data-driven discovery of models (d3m). `https://www.darpa.mil/program/data-driven-discovery-of-models`. Accessed: 2018-03-24.

[Shn96] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[Sin12] A. Singhal. Introducing the Knowledge Graph: Things, not Strings. `https://googleblog.blogspot.com/2012/05/introducing-`

knowledge-graph-things-not.html, May 2012. Online; accessed 22-Apr-2020.

[SK19] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[SKB+18] Dominik Sacha, Matthias Kraus, Jürgen Bernard, Michael Behrisch, Tobias Schreck, Yuki Asano, and Daniel A Keim. Somflow: Guided exploratory cluster analysis with self-organizing maps and analytic provenance. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):120–130, 2018.

[SKKC19] Dominik Sacha, Matthias Kraus, Daniel A Keim, and Min Chen. Vis4ml: An ontology for visual analytics assisted machine learning. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):385–395, 2019.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2015.

[SMM12] Michael Sedlmair, Miriah Meyer, and Tamara Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE transactions on visualization and computer graphics*, 18(12):2431–2440, 2012.

[SMWH17] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE*

Transactions on Visualization and Computer Graphics, 23(1):341–350, 2017.

[SPH+16] Carson Sievert, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. plotly: Create interactive web graphics via plotly.js. *R package version*, 3(0), 2016.

[SPM] Hendrik Strobelt, Evan Phibbs, and Mauro Martino. Ffqd-mnist – Forma Fluens quickdraw model decay indicator dataset. `http://www.formafluens.io/client/mix.html`. Version: 0.1.

[spo] spotfire. `https://www.tibco.com/products/tibco-spotfire`. Accessed: 2018-12-12.

[SPW98] Emad W Saad, Danil V Prokhorov, and Donald C Wunsch. Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Transactions on neural networks*, 9(6):1456–1470, 1998.

[SR17] Yangqiu Song and Dan Roth. Machine learning with world knowledge: The position and survey. *arXiv preprint arXiv:1705.02908*, 2017.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

[SSS+14] Dominik Sacha, Andreas Stoffel, Florian Stoffel, Bum Chul Kwon, Geoffrey Ellis, and Daniel A Keim. Knowledge generation model for visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1604–1613, 2014.

[SSW15] Shiliang Sun, Honglei Shi, and Yuanbin Wu. A survey of multi-source domain adaptation. *Information Fusion*, 24:84–92, 2015.

[SSW+18] Gaurav Sheni, Benjamin Schreck, Roy Wedge, James Max Kanter, and Kalyan Veeramachaneni. Prediction factory: automated development and collaborative evaluation of predictive models. *arXiv preprint arXiv:1811.11960*, 2018.

[SSZ+16] Dominik Sacha, Michael Sedlmair, Leishi Zhang, John A. Lee, Daniel Weiskopf, Stephen C. North, and Daniel A. Keim. Human-Centered Machine Learning Through Interactive Visualization: Review and Open Challenges. In *Proceedings of the 24th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, Bruges, Belgium*, April 2016.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[SZ14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[tab] Tableau. `https://www.tableau.com/`. Accessed: 2018-12-12.

[THHLB13] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.

[TLW+19] Wenbo Tao, Xiaoyu Liu, Yedi Wang, Leilani Battle, Çağatay Demiralp, Remco Chang, and Michael Stonebraker. Kyrix: Interactive pan/zoom visualizations at scale. In *Computer Graphics Forum*, volume 38, pages 529–540. Wiley Online Library, 2019.

[TM05] F-Y Tzeng and K-L Ma. Opening the black box-data driven visualization of neural networks. In *Visualization, 2005. VIS 05. IEEE*, pages 383–390. IEEE, 2005.

[TMD+06] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[TP08] Michael W Trosset and Carey E Priebe. The out-of-sample problem for classical multidimensional scaling. *Computational statistics & data analysis*, 52(10):4635–4642, 2008.

[Tuk77] John W Tukey. *Exploratory data analysis*, volume 2. Reading, Mass., 1977.

[Tuk93] John W Tukey. Exploratory data analysis: past, present and future. Technical report, PRINCETON UNIV NJ DEPT OF STATISTICS, 1993.

[une20] Us unemployment rate by county, 1990-2016. `https://www.kaggle.com/jayrav13/unemployment-by-county-us#output.csv`, 2020. Online; accessed 10-Mar-2020.

[uni17] Uniprot: the universal protein knowledgebase. *Nucleic acids research*, 45(D1):D158–D169, 2017.

[Vap92] Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

[var20a] various. Alteryx. `https://www.alteryx.com`, 2020. Online; accessed 19-Mar-2020.

[var20b] various. D:Swarm. `http://www.dswarm.org/`, 2020. Online; accessed 19-Mar-2020.

[var20c] various. Google Data Studio. `http://datastudio.google.com`, 2020. Online; accessed 19-Mar-2020.

[var20d] various. OpenRefine. `https://openrefine.org`, 2020. Online; accessed 19-Mar-2020.

[var20e] various. Tableau. `https://www.tableau.com`, 2020. Online; accessed 19-Mar-2020.

[VDDP18] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[VDEvW11] Stef Van Den Elzen and Jarke J van Wijk. Baobabview: Interactive construction and analysis of decision trees. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pages 151–160. IEEE, 2011.

[VDR17] Gust Verbruggen and Luc De Raedt. Towards automated relational data wrangling. In *Proceedings of AutoML 2017@ ECML-PKDD: Automatic selection, configuration and composition of machine learning algorithms*, pages 18–26, 2017.

[VvRBT13] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[VW05] Jarke J Van Wijk. The value of visualization. In *Visualization, 2005. VIS 05. IEEE*, pages 79–86. IEEE, 2005.

[WC$^+$08] Hadley Wickham, Winston Chang, et al. ggplot2: An implementation of the grammar of graphics. *R package version 0.7, URL: `http://CRAN.R-project.org/package=ggplot2`*, 2008.

[WCL$^+$18] Zhe Wang, Dylan Cashman, Mingwei Li, Jixian Li, Matthew Berger, Joshua A Levine, Remco Chang, and Carlos Scheidegger. Neuralcubes: Deep representations for visual data exploration. *arXiv preprint arXiv:1808.08983*, 2018.

[Wer90] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[WGSY19] Junpeng Wang, Liang Gou, Han-Wei Shen, and Hao Yang. Dqnviz: A visual analytics approach to understand deep q-networks. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):288–298, 2019.

[WLS$^+$10] Furu Wei, Shixia Liu, Yangqiu Song, Shimei Pan, Michelle X Zhou, Weihong Qian, Lei Shi, Li Tan, and Qiang Zhang. Tiara: a visual exploratory text analytic system. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 153–162. ACM, 2010.

[WLSL17] Junpeng Wang, Xiaotong Liu, Han-Wei Shen, and Guang Lin. Multi-resolution climate ensemble parameter analysis with nested parallel coordinates plots. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):81–90, 2017.

[Won10] Bang Wong. Gestalt principles (part 1). *nature methods*, 7(11):863–864, 2010.

[WSW$^+$18] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mané, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):1–12, 2018.

[WZM$^+$16] Xu-Meng Wang, Tian-Ye Zhang, Yu-Xin Ma, Jing Xia, and Wei Chen. A survey of visual analytic pipelines. *Journal of Computer Science and Technology*, 31:787–804, 2016.

[XBK$^+$15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[YCN$^+$15] Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.

[YGCC12] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 97–108, 2012.

[ZF14] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[ZHC17] Erkang Zhu, Yeye He, and Surajit Chaudhuri. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment*, 10(10):1034–1045, 2017.

[ZL16] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[ŽPG16] Indrė Žliobaitė, Mykola Pechenizkiy, and Joao Gama. An overview of concept drift applications. In *Big data analysis: new algorithms for a new society*, pages 91–114. Springer, 2016.

[ZVSL17] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.

[ZWM$^+$18] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[ZYZC18] Weiguo Zheng, Jeffrey Xu Yu, Lei Zou, and Hong Cheng. Question answering over knowledge graphs: question understanding via template decomposition. *Proceedings of the VLDB Endowment*, 11(11):1373–1386, 2018.

[ZZXW19] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 2019.