Visualizations for Model Tracking and Predictions in Machine Learning

by

Wei-En Lee

Submitted to the Department of Electrical Engineering and Computer Science on May 26, 2017, in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering

Abstract

Building machine learning models is often an exploratory and iterative process. A data scientist frequently builds and trains hundreds of models with different parameters and feature sets in order to find one that meets the desired criteria. However, it can be difficult to keep track of all the parameters and metadata that are associated with the models. ModelDB, an end-to-end system for managing machine learning models, is a tool that solves this problem of model management. In this thesis, we present a graphical user interface for ModelDB, along with an extension for visualizing model predictions. The core user interface for model management augments the ModelDB system, which previously consisted only of native client libraries and a backend. The interface provides new ways of exploring, visualizing, and analyzing model data through a web application. The prediction visualizations extend the core user interface by providing a novel prediction matrix that displays classifier outputs in order to convey model performance at the example level. We present the design and implementation of both the core user interface and the prediction visualizations, discussing at each step the motivations behind key features. We evaluate the prediction visualizations through a pilot user study, which produces preliminary feedback on the practicality and utility of the interface. The overall goal of this research is to provide a powerful, user-friendly interface that leverages the data stored in ModelDB to generate effective visualizations for analyzing and improving models.

Thesis Supervisor: Samuel Madden
Title: Professor, EECS
Acknowledgments

I would like to thank Professor Samuel Madden for his advice and guidance on my research project. I would also like to thank Manasi Vartak, for providing the constant support and feedback that helped me make progress on my research. I am also grateful to all of the people who have contributed to ModelDB along the way. Finally, I would like to give a huge thank you to all of my friends and family who have been there for me during my time at MIT.
Contents

1 Introduction 13

2 Background 17
   2.1 Machine Learning Workflows ................................. 17
   2.2 Overview of ModelDB ........................................... 18
      2.2.1 Goals .................................................. 18
      2.2.2 ModelDB Architecture .................................. 18
      2.2.3 Use Cases .............................................. 20
   2.3 Supervised Learning ........................................... 21

3 Related Work 23
   3.1 Scientific Workflow Management .............................. 23
   3.2 Visualizing Model Performance ............................... 24
      3.2.1 Summary Statistics .................................... 24
      3.2.2 Ensembles .............................................. 24
      3.2.3 High-dimensional Feature Sets ......................... 25
      3.2.4 Predictions ............................................ 25
      3.2.5 Composite Systems .................................... 26

4 ModelDB User Interface 27
   4.1 Overview .................................................... 27
   4.2 Design Goals ............................................... 27
   4.3 Frontend Architecture ...................................... 28
4.4 Components ................................................. 29
4.5 Projects Page ........................................... 31
4.6 Models Page ............................................. 33
  4.6.1 Overview and Charts ................................. 33
  4.6.2 Models Table ......................................... 38
  4.6.3 Querying Models ..................................... 42
4.7 Single Model Page ....................................... 47

5 Prediction Visualizations 49
  5.1 Overview ................................................ 49
  5.2 Preliminary Design ...................................... 49
    5.2.1 Existing Research ................................. 50
    5.2.2 Interviews ......................................... 51
    5.2.3 Design Goals ...................................... 51
  5.3 Overall Layout .......................................... 52
  5.4 Prediction Matrix ...................................... 54
    5.4.1 Overview ......................................... 54
    5.4.2 Encoding Schemes ................................. 55
    5.4.3 Sorting ............................................. 57
    5.4.4 Clustering ......................................... 58
    5.4.5 Selection ......................................... 59
  5.5 Summary Visualizations ................................. 60
    5.5.1 ROC Curves ....................................... 61
    5.5.2 PR Curves ......................................... 62
    5.5.3 Confusion Matrices ................................. 62
  5.6 Example-level Debugging ............................... 63
    5.6.1 Highlighting Nearest Neighbors ................. 63
    5.6.2 Quick Inspection of Data .......................... 65
    5.6.3 Raw Data Table .................................... 65
  5.7 Filtering and Grouping .................................. 66
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>ModelDB system architecture</td>
<td>19</td>
</tr>
<tr>
<td>4-1</td>
<td>Frontend architecture</td>
<td>29</td>
</tr>
<tr>
<td>4-2</td>
<td>User interface flow diagram</td>
<td>30</td>
</tr>
<tr>
<td>4-3</td>
<td>Navigation bar</td>
<td>30</td>
</tr>
<tr>
<td>4-4</td>
<td>Projects page</td>
<td>31</td>
</tr>
<tr>
<td>4-5</td>
<td>Projects page loading screen</td>
<td>32</td>
</tr>
<tr>
<td>4-6</td>
<td>Project overview</td>
<td>32</td>
</tr>
<tr>
<td>4-7</td>
<td>Project search</td>
<td>33</td>
</tr>
<tr>
<td>4-8</td>
<td>Models page</td>
<td>34</td>
</tr>
<tr>
<td>4-9</td>
<td>Models page summary graph</td>
<td>35</td>
</tr>
<tr>
<td>4-10</td>
<td>Summary graph brushing</td>
<td>36</td>
</tr>
<tr>
<td>4-11</td>
<td>Summary graph drill down</td>
<td>37</td>
</tr>
<tr>
<td>4-12</td>
<td>Custom model comparison visualizations</td>
<td>38</td>
</tr>
<tr>
<td>4-13</td>
<td>Models table</td>
<td>39</td>
</tr>
<tr>
<td>4-14</td>
<td>Model annotations modal window</td>
<td>41</td>
</tr>
<tr>
<td>4-15</td>
<td>JSON metadata modal window</td>
<td>42</td>
</tr>
<tr>
<td>4-16</td>
<td>Filtering models</td>
<td>44</td>
</tr>
<tr>
<td>4-17</td>
<td>Grouping models</td>
<td>46</td>
</tr>
<tr>
<td>4-18</td>
<td>Sorting models</td>
<td>47</td>
</tr>
<tr>
<td>4-19</td>
<td>Single model page</td>
<td>47</td>
</tr>
<tr>
<td>5-1</td>
<td>Predictions page</td>
<td>53</td>
</tr>
<tr>
<td>5-2</td>
<td>Prediction matrix</td>
<td>55</td>
</tr>
</tbody>
</table>
5-3 Prediction matrix encoding schemes ........................................ 56
5-4 Prediction matrix clustering .................................................. 59
5-5 Selecting rows or columns in the prediction matrix ...................... 60
5-6 ROC and PR curves .............................................................. 61
5-7 Confusion matrices ............................................................. 63
5-8 Nearest neighbors in the prediction matrix .................................. 64
5-9 Inspecting examples from the prediction matrix ............................ 65
5-10 Raw data table ................................................................. 66
5-11 Menus for filtering and grouping .......................................... 67
5-12 Aggregate matrix ............................................................... 68
Chapter 1

Introduction

Building machine learning models is often an exploratory and iterative process. Model building begins with a hypothesis about the underlying data, followed by the construction and evaluation of a few models. Based on the results, the models are then iteratively refined. A data scientist might have to train hundreds of models for a single project, tweaking hyperparameters and feature sets in order to find a model that meets certain criteria.

Despite the iterative nature of model building, there is currently no easy way to manage all of the models built over time. Data scientists often resort to ad-hoc methods for organizing models, or end up wasting valuable resources to regenerate old results. The issue here is one of model management, which is the tracking, storing and indexing of large numbers of machine learning models so they may subsequently be shared, queried, and analyzed.

An effective system for model management provides various benefits on top of the basic improvements to organization. Such a system can provide an overview of previously built models, allowing users to gain valuable insights about how to make improvements. The ability to compare across models and visualize trends also helps with the process of sensemaking, which involves getting a better understanding of the underlying phenomenon being studied. Finally, model management promotes and facilitates collaboration, so that teammates can easily build on top of each other’s work.
ModelDB is a novel, end-to-end system for managing machine learning models that aims to solve the problem of model management. The system automatically tracks models in their native machine learning environments, intelligently indexes them by extracting and storing relevant metadata, and provides a graphical user interface for easy querying and visualization. ModelDB is able to manage the entire workflow, which includes all of the steps from data preprocessing to training and testing.

In this paper we present the design and implementation of the graphical user interface component of ModelDB. The interface is intended to augment the ModelDB backend by making certain operations easier to perform, and by providing useful visualizations that help achieve some of the goals of ModelDB. In particular, the interface provides the following benefits:

1. Data is laid out in a format that is easier to understand than the raw tables in which they are stored. Relevant metadata are grouped together intelligently and organized so that users can easily read and analyze the information.

2. The graphical user interface provides an easier method of querying the data without requiring any knowledge of how to write SQL commands or any familiarity with the underlying database schema.

3. Users are able to visually compare models, which is not possible if exploration is restricted to a simple data table. By offering visualizations and flexible charting, we empower users to quickly discover trends that can help them understand errors or realize improvements. Although these techniques have been explored in the past, this system simplifies the visual analysis process by leveraging ModelDB’s data logging to automatically generate visualizations without the usual overhead.

4. The interface provides a simple and intuitive avenue for updating the stored data. For example, users are able to annotate models with comments by simply entering it into a text box in the interface.

While the comparison of models via analysis of their hyperparameters and metrics
is useful in the context of model management, it is often insufficient for understanding the underlying differences between models. In these cases, it is helpful to analyze the predictions themselves to gain a better sense of model performance. For example, a data scientist might ask questions like “What features are common amongst data points that are misclassified?”, or “What data points are repeatedly misclassified by a given set of models?”

In order to support a deeper understanding of models and their outputs, we also present an extension to the core user interface of ModelDB. In this extension, we provide a novel interface for visualizing predictions that is designed to answer the various questions data scientists might have when debugging models.

Together, the graphical user interface for ModelDB and the prediction visualizations make up the main focus of this paper. Section 2 provides background information on machine learning workflows and the ModelDB system as a whole. Section 3 mentions some related work in visualizations for machine learning which served as inspiration. Sections 4 and 5 present the ModelDB user interface and the prediction visualizations respectively. In both sections, we begin by discussing the high-level goals and architecture before diving deeper into the design and implementation. Section 6 includes the results of a pilot user study of the prediction visualizations, while section 7 outlines some possible directions for future work. Section 8 concludes the paper with some final remarks.
Chapter 2

Background

2.1 Machine Learning Workflows

This section provides a description of the general process a data scientist follows when building machine learning workflows.

1. **Data preprocessing** - This step involves taking the raw data and processing it so that it is ready for use with the specific machine learning environment. This might involve massaging the data into a particular format, cleaning the data by removing rows with missing fields, or sampling data points if there is an excess.

2. **Feature extraction/transformation** - In this step, data scientists select which features to use when training the model. Not all features are relevant, so the scientist might pick and choose. This step can also involve transformations of the data. For example, a transformation might be applied to convert categorical features into numbers or to scale a certain feature down to a given range.

3. **Training** - This is where the models are trained on the data. This involves fitting a model to the processed data, and is generally the most computationally intensive step in the workflow. The models selected depend on the type of the
problem, but in this paper we focus mainly on supervised learning, in which the input data is associated with target values to be learned. For the prediction visualizations, we will focus specifically on the problem of binary classification.

4. **Testing** - Once trained, the models are evaluated based on some accuracy metric that is appropriate for the given model and problem.

### 2.2 Overview of ModelDB

ModelDB is an end-to-end system for managing machine learning models that aims to solve the problem of model management. ModelDB “automatically tracks machine learning models in their native environment, indexes them intelligently, and allows flexible exploration of models via SQL as well as a visual interface.” [17] The system is currently being developed by members of the MIT Database Group, and various parts of the system have been the focus of a few previous theses. In this section, we include a brief overview of the system that will provide the background necessary to understand the significance of our contributions to the user interface.

#### 2.2.1 Goals

Building machine learning models is an iterative process. A data scientist might train hundreds of models, tweaking hyperparameters and feature sets before finding one that meets some target criteria. It is difficult to keep track of all of these models and their associated metadata, and there are few solutions for the problem of model management. ModelDB is intended to fill the void by providing a system to manage models. The overarching goal for ModelDB is to allow users to track, store, and index models so that they can subsequently be reproduced, shared, queried, and analyzed.

#### 2.2.2 ModelDB Architecture

ModelDB is designed as a modular client-server system and consists of three main components: native client libraries for various machine learning environments, a backend
with key abstractions that stores data, and a web-based frontend interface (Figure 2-1). The backend exposes a thrift interface, which is used by both the native clients and the frontend interface to communicate with the backend.

![Diagram of ModelDB system architecture](image)

**Figure 2-1: ModelDB system architecture**

**Client Libraries**

The native client libraries allow data scientists to track models and store relevant metadata straight from the machine learning environment. The clients are implemented in such a way that the data scientist only needs to make minor modifications to the code of their existing workflows. The system then automatically and transparently logs the relevant data to the backend.

One advantage of the native clients is that ModelDB does not require the user to use a GUI to create a workflow in advance, which is necessary in other workflow management programs like VisTrails [4, 2, 5]. Instead, ModelDB minimizes the changes necessary to incorporate the system into existing workflows.

ModelDB currently has native clients for two popular machine learning environments, spark.ml and scikit-learn. However, it is straightforward to add support for additional environments.
ModelDB Backend

The backend defines key abstractions and provides access to the storage layer. The data is stored in a SQL database, and the backend exposes a thrift interface which allows communication with both the client libraries and the frontend.

Within the database, models are stored in a grouping hierarchy that captures how they are organized. At the highest level are Projects, which have a name, author, and description. Projects can contain multiple Experiments, which are the second level in the grouping hierarchy. Experiments can contain multiple Experiment Runs, which finally contain a set of Models. Models contain a variety of data points, including information about the model type, hyperparameters, metrics, and metadata.

User Interface

The web-based frontend provides a graphical user interface which allows users to query and visualize the data that is stored in the database. The frontend is implemented as a Node.js application and communicates with the ModelDB backend via an Apache Thrift interface. The user interface is the main focus of this paper, and its design and implementation are discussed in further detail in sections 4 and 5.

2.2.3 Use Cases

In setting out to solve the problem of model management, ModelDB fulfills various use cases.

1. **Experiment Tracking** - ModelDB logs all of the relevant hyperparameters and metadata so that they will be accessible later on.

2. **Versioning** - The storage of information about past models allows data scientists to preserve and access previous versions of models they have built.

3. **Reproducibility** - All of the parameters necessary to reproduce an experiment are logged so that data scientists can regenerate results. This can be useful when
verifying the results of previous experiments.

4. **Comparisons, Queries, and Search** - Models logged with ModelDB can be easily compared in terms of their parameters, metadata, or metrics. ModelDB also allows users to query models based on the associated data that have been logged. Finally, the indexed models can also be efficiently searched.

5. **Collaboration** - ModelDB acts as a central repository of models which facilitates collaboration between members of a team. Data scientists can review the work of others and build on top of existing results. The ability to annotate models is also a useful feature for collaboration.

### 2.3 Supervised Learning

This section describes some of the machine learning concepts relevant to the prediction visualizations to be presented later in section 5. **Supervised learning** is the task of using labeled training data to infer a function that maps new observations to predictions. The training data, also called training examples, consist of feature vectors along with their labels. The goal of supervised learning is to build a model that can predict the label of new, unseen examples. Models are often evaluated by comparing predictions with the ground truth, or what the labels are actually known to be.

The prediction visualizations presented in this paper deal with a specific type of supervised learning known as classification. Classification is the task of identifying which of a set of categories an observed example belongs to, based on the set of training data. Here, the class or category is the label that is predicted in the supervised learning task. In particular, the prediction visualizations focus on binary classification, or the subset of classification problems where there are exactly two categories. For these problems, the output of the model on an observed example is a prediction value between 0 and 1. The value represents the probability that the observed example has a label of 1, according to the model.
Chapter 3

Related Work

In this section, we describe previous work related to the user interfaces presented in this paper. These related works act as a starting point for our research on ModelDB, and help provide some context on the existing technologies that our system hopes to supplement or improve upon.

3.1 Scientific Workflow Management

There has been much research done in the topic of scientific workflow management. For example, the KEPLER System [10] and the Taverna workflow suite [18] are two such systems that have demonstrated applications in managing workflows related to biological studies. Commercial systems such as the AzureML\textsuperscript{1} suite and the SeaHorse\textsuperscript{2} suite also allow users to graphically construct machine learning workflows. The system most similar to ModelDB is Vistrails [4, 2, 5], which has grown over time to support generalized scientific workflows, including those using scikit-learn.

In contrast to ModelDB, almost all of the existing tools for scientific workflow management require scientists to use a graphical user interface to define workflows prior to execution. According to interviews conducted with data scientists, graphical user interfaces are often too restrictive for constructing machine learning workflows.

\textsuperscript{1}https://azure.microsoft.com/en-us/services/machine-learning/
\textsuperscript{2}https://seahorse.deepsense.io/
In these interfaces, users are limited to the set of operators and drag-and-drop components provided by the system. In addition, data scientists mentioned that they constructed workflows extemporaneously, so predetermined workflows would not be sufficient. ModelDB provides the desired level of flexibility by including native client libraries to infer the workflows directly from code, and using visualization only for post hoc analysis.

3.2 Visualizing Model Performance

There is also a great deal of previous research on using visualizations to understand the performance of models. Models are complicated and deal with high-dimensional data, so there are many ways to visualize their parameters, features, and outputs. Existing studies explore the various ways that visualization can help data scientists interpret machine learning models.

3.2.1 Summary Statistics

Some systems focus on providing summary statistics about the aggregate performance of models, usually through the use of various graphs. ROCR, a visualization package in the R language, generates parameterized performance curves by varying classification threshold levels [15]. ManiMatrix provides summary statistics in the form of a confusion matrix, adding the ability to specify desired changes to certain cells of the matrix [7]. The system then automatically updates the costs of misclassifications to try and achieve the desired changes. Systems that provide summary statistics are useful, but they lack the ability to drill down and enable debugging at the example level.

3.2.2 Ensembles

Other systems deal with visualizing ensembles, or weighted combinations of classifiers. EnsembleMatrix provides a novel way of combining classifiers and exploring
each classifier’s relative merits [16]. In another study, researchers experiment with a radial visualization of ensemble classifier contributions [14]. While ensembles are quite powerful and visualizations can help scientists understand and refine ensembles, the interface presented in this paper chooses to focus on the more general problem of visualizing single classifier outputs instead.

3.2.3 High-dimensional Feature Sets

The input data for machine learning models are feature vectors that often contain too many dimensions to properly visualize. Some existing research is dedicated to exploring novel ways of visualizing this data space effectively. In one study, researchers use a projection method based on self ordering maps to project down to lower dimensions [12]. This method results in a view where neighboring locations in the display space correspond to neighboring locations in the data space. However, this type of projection can be confusing because there is no intuitive interpretation of the projected space. Another system called SmartStripes allows users to investigate the dependencies between different feature subsets by providing visualizations to aid the feature selection process [11]. VizRank selects the most useful data projections by ranking them based on their ability to visually discriminate between classes [9].

3.2.4 Predictions

For example-level debugging, it is often useful to view the actual output, or predictions, of the machine learning models. Prospector is one such system that deals with visualizing classifier predictions [8]. Prospector builds on existing methods of plotting partial dependence of features, and adds functionality by allowing users to see how model predictions change in response to changes in feature values. ModelTracker is another system that visualizes predictions in a way that conveys overall model performance and enables direct inspection of the data [1]. ModelTracker also partially addresses the iterative nature of model building by allowing users to compare between consecutive iterations of models. There are also other systems which are inspired by
ModelTracker, such as an alternate version that supports multi-class classification and data binning [13].

The prediction visualizations presented in this paper are also inspired by ModelTracker, and many of our features are intended to improve on the existing functionality. In particular, we explore how the addition of various visual encoding schemes can help the user understand prediction values. We also enable the user to easily compare across multiple models, rather than showing just the differences between consecutive iterations. Finally, we add various ways of sorting and clustering the data to help highlight similarities between examples.

3.2.5 Composite Systems

While the research described in previous sections do well in exploring novel uses of visualization for specific aspects of model performance, the most practical systems generally combine multiple aspects in order to provide a more functional tool. For example, summary statistics and example level debugging are both useful to some extent in isolation, but they are much more powerful when combined. In one study, researchers aim to combine the various aspects of model performance by providing summary statistics in conjunction with a scatterplot matrix of predictions that can show the dependence between any two features [3].

The benefits of having a composite, well-rounded system is also discussed in a design study of the current landscape of machine learning visualization tools [6]. The design study investigated user needs by recruiting non-expert machine learning practitioners for interviews. The study also analyzed various existing visualization systems to discover what was lacking, and presented various design themes that it found to be relevant for successful systems. These themes are discussed in greater detail in section 5.2.1, and support the idea that the best-performing systems are those which combine different types of visualizations.
Chapter 4

ModelDB User Interface

4.1 Overview

This section describes the design and implementation of the ModelDB user interface. We begin with a discussion of overarching design goals, followed by a description of the system architecture and key components. We then dive into each of the components individually for more thorough explanations of their features.

4.2 Design Goals

While other scientific workflow management tools utilize graphical user interfaces for workflow creation, the interface of ModelDB is intended for post hoc analysis of logged data. Consequently, the user interface is designed to support the operations necessary for this analysis. To ensure that the interface focuses on the appropriate functionality, we begin by outlining various goals:

1. **Display logged data** - The interface should provide a clean and intuitive method of displaying the data stored in the ModelDB backend. The logged data can be complicated and messy - models may have a great number of hyperparameters and metadata. The system should be able to display this in a way that facilitates better exploration and understanding in comparison to standard database tables.
2. **Support model querying** - ModelDB is intended for machine learning workflows that may contain hundreds of models. With so many models, the system needs an effective method of querying models based on their stored attributes. For example, a data scientist might want to look at all of the models of a certain type, or all models with a certain accuracy.

However, there is a tradeoff between ease-of-use and flexibility when it comes to querying. The system should make it easy for users to query models without having to learn SQL or understand the underlying database schema. At the same time, it still needs to provide enough flexibility that the user is able to select the right models based on desired properties. Ideally, the interface should strike a right balance between these two goals.

3. **Visually compare models** - The comparison of previously built models is part of the iterative process of machine learning model building. Comparing a large number of models can be difficult and time consuming, however, especially when the models have many attributes. The interface should provide visualizations that help data scientists compare across models easily and quickly. These visualizations should empower users to discover trends or insights that lead to a better understanding of the data.

4. **Provide flexibility** - There are many different types of models, each of which may be associated with a different set of hyperparameters and metadata. Furthermore, data scientists might want to augment logged data to suit their own needs. For example, users might want to tag specific models or annotate them for future reference. The interface should support a set of default functions, but provide enough flexibility for a wide range of usage patterns.

### 4.3 Frontend Architecture

The ModelDB frontend is a web application that connects to the rest of the ModelDB system through a Thrift API that the backend exposes. The web application is
implemented as a simple client-server Node.js application, as shown in Figure 4-1. Node.js was chosen for its ease-of-use as well as its compatibility with a large library of npm modules. For example, we make use of a Thrift module that enables a seamless connection with the ModelDB backend. The application also uses the Express framework and the EJS templating engine. EJS was selected for its simple syntax and gentle learning curve, which is helpful for opening up the system to open source contributions.

![Diagram of frontend architecture](image)

Figure 4-1: Frontend architecture

Clients communicate with the Node server by sending HTTP requests via a web browser. The server fetches some data for the initial page load, but much of the data is loaded into the client subsequently via asynchronous Ajax calls. This allows pages to render more quickly and lazily load additional data as needed. Clients fetch stylesheets and scripts from the server, which include ModelDB scripts as well as local versions of third-party visualization libraries like D3 and Vega.

4.4 Components

The ModelDB interface consists of several key components, each of which are accessible via various URLs that are exposed through links in the interface. In this section, we list the main components and explain how they fit together. Later sections discuss individual components in greater detail.

The main components of the user interface are the projects page, the models page, and the single model page. The projects page is the landing page for the
entire interface, and contains all of the projects stored within ModelDB. The models page is associated with a specific project, and contains information and visualizations relevant to the models of that project. The single model page provides information about a specific model.

Figure 4-2 shows the user interface flow that connects the components together. The components are hierarchical - projects contain a set of models, and a set of models contains single models. As shown in Figure 4-2, a user can select a project on the projects page to get to the models page for that project. On the models page, a user can select a specific model to view additional information about that single model.

Figure 4-2: User interface flow diagram

To enable quick navigation between the hierarchical components, we also include a fixed navigation bar at the top of every page (Figure 4-3). For a given page, the navigation bar provides links to components that exist higher up in the hierarchy. Naturally, links to more specific components in the hierarchy cannot be provided because they have not yet been selected. For simplicity, these links are not pictured in the user interface flow diagram.

Figure 4-3: Navigation bar
4.5 Projects Page

The projects page acts as the main landing site for the ModelDB frontend (Figure 4-4). As described in section 2.2.2, projects in ModelDB are the highest level in the grouping hierarchy and can contain many experiment runs and experiments, which in turn contain models. While it is up to the users to decide how to group their experiments, a project will usually address a single overarching machine learning task (e.g., predicting IMDB movie ratings).

![Figure 4-4: Projects page](image)

The page is initially rendered with basic information for each project, including the name, author, and description (Figure 4-5). Afterwards, the system makes Ajax calls to the server to load additional project information. This is done asynchronously so that the initial page load time is shorter and users are able to interact with basic project data while waiting for additional data to load.

As the asynchronous calls finish loading additional data, the projects are augmented with relevant information. This includes the last timestamp at which the project was updated (Figure 4-6 (a)), the number of models in the project, and a visual breakdown of the various types of models (Figure 4-6 (b)). The breakdown shows a bar with one colored section for each model type, and hovering on a section
reveals a tooltip with the type name and number of models. Each section’s width is proportional to the percentage of models that the chosen type comprises. This allows users to quickly get a sense of the relative number of models for each type.

As shown in Figure 4-6 (c), the system also calculates summary statistics for all of the metrics it finds within the project. The system automatically looks through the models to see what metrics are included, so there is no need to specify metric names in advance. This provides the user with an overview of how the project is performing thus far. If similar projects exist, the statistics can also be useful for selecting a particular project to focus on.

If ModelDB is used to log a large number of projects, it may be difficult to find the right one from within the projects page. To address this issue, the navigation bar includes a search box that allows users to search for a specific project (Figure 4-7). This is implemented using the hideseek jQuery plugin for customizable live search.
Projects are shown or hidden immediately as the user types into the textbox, resulting in instantaneous feedback. The system searches all of the text within a project div element, so users can search by the project’s name, author, or description.

![Figure 4-7: Project search](image)

Each project div element is a link to the models page for the given project. To convey this affordance to the user, each div is subtly highlighted and outlined when the mouse hovers over.

### 4.6 Models Page

The models page is associated with a chosen project, and contains data about all of the models in that project (Figure 4-8). The page contains a fixed menu on the left and a main content section to the right. The page is organized into two tabs, which are rendered in the main content section and accessible via buttons in the menu. The first tab shows a visual overview of the models and allows users to generate charts. The second tab shows the data in a tabular format and allows interactions with key value pairs. Both tabs are discussed in more detail in the following sections.

#### 4.6.1 Overview and Charts

The default tab for the models page includes a summary visualization that provides an overview of the models, as well as charting capabilities that allow users to generate visualizations with finer control.
Summary Graph

The summary graph is the centerpiece of the first tab of the models page, and its main purpose is to provide a quick overview of the project’s models. The entire graph, including the supported interactions, is implemented using the Vega visualization grammar.

Each point in the graph corresponds to a model in the project. The y-axis shows metric values while the x-axis shows model IDs. Since model IDs are increasing in ModelDB, having the IDs on the x-axis means that the graph is able to illustrate model performance over time. The menu also provides an option to replace the x-axis with explicit timestamps instead.

Since the project may contain different models that have different metrics, the
metric name for each point is encoded using different shapes. In Figure 4-9, we see that the root-mean-square error (rmse) is encoded by a circular mark, whereas f1 scores are encoded by a plus sign. The points are also color coded to indicate what type of model each point refers to.

Based on the design choices mentioned above, data scientists can use the summary graph to easily answer various questions about the project’s models. For example, the graph answers the following questions:

- Which models perform best/worst overall?
- What type of models perform best/worst?
- For a given type of model, which are the best/worst ones?
- Have my models been getting better or worse over time?
- What are the points where models suddenly get better or worse?

All of these questions can also be answered via inspection of the data in the tabular display, but the summary graph provides a faster, visual method of discovering these insights.

For projects that contain hundreds of models or more, the summary graph can get rather cluttered. To address the issue of scalability with the number of models, the
summary graph utilizes a user interface technique called brushing to select a subset of the data. Underneath the main plot is a simplified profile of the data points on the same initial x-axis. The user can drag on this section to limit the main plot’s x-axis, as shown in Figure 4-10. In this way, the user can focus in on a certain window of the data. Filters which are applied to the models will also limit the points that are shown on the graph. This is described in greater detail during section 4.6.3.

Figure 4-10: Summary graph brushing
A view of the summary graph which only shows a subset of all points. The user selected this subset by brushing on the simplified profile view below the main chart.

The summary graph also allows users to inspect the hyperparameters and metadata associated with each model. The graph surfaces some basic information through tooltips that show up when hovering over points. For more complete information, the user can click on a point to slide out a card that contains the rest of the data. The tooltips and model card are shown in Figure 4-11. Together, these two features enrich the summary visualization by allowing users to drill down into a specific model. This combination of overview and detail results in a flexible and powerful way of exploring the data visually.
The summary graph can be used to drill down into model parameters and metadata. Hovering over a point shows a tooltip with basic information, whereas clicking on a point brings up a card with more data on the right.

**Custom Visualizations**

The summary graph shows metric values as a function of Model ID because this plot can answer many initial questions the user might have about the models of a given project. However, data scientists might be interested in creating other charts to explore trends. For example, the user might want to know how accuracy is affected by the regularization parameter of the models.

To support the flexible generation of charts, we include a section below the summary graph that allows users to specify the y-axis and x-axis for the plot using drop-down lists (Figure 4-12). These lists are automatically populated with all of the fields that the system finds during its scan through the models. This means that users can graph by any field the model might have, such as hyperparameters or metadata.

In addition to selecting the fields for the two axes, users can also select a field to group by and a function to aggregate with. Figure 4-12 shows an example of a generated chart which plots root-mean-square error versus maxDepth, grouped by model type.

The ability to generate charts is useful when the data scientist has specific hy-
Figure 4-12: Custom model comparison visualizations

This is an example of a generated chart that plots rmse vs maxDepth. The chart is grouped by model type and aggregated by averaging the data points.

Hypotheses about how metric values are affected by various hyperparameters. However, the charts can also be used in an exploratory fashion to discover trends that are not as obvious. Moreover, some of these trends are difficult to observe in a data table without visualization. These hidden insights can potentially help data scientists gain a better understanding of how to improve their models.

4.6.2 Models Table

The second tab of the models page is a tabular view of the model data (Figure 4-13). Here, users are able to interact directly with the various parameters and metadata of the models. Since models have a wealth of information associated with them, the system takes care to organize this data in a way that is easier to explore and understand than a standard database table. In addition, the models are inserted into the page lazily in order to limit the time it takes to render all of the div elements. This is useful for the initial page load, as well as the sorting operations which would otherwise have to redraw hundreds of div elements.

Each row in the table represents a single model in the project. The table is
Figure 4-13: Models table

divided up into five columns, each of which contains related key-value pairs. These pairs can be dragged and dropped for the purpose of filtering, which is described later in section 4.6.3. The organization of the key-value pairs is based on the data that gets logged in ModelDB. Relevant pairs are grouped together, and the sections delineate different types of important information that is captured by the system. The five column sections are indicated below.

1. IDs - This section contains various identifying information by providing the IDs that the model is associated with. A model in ModelDB has a unique model ID, which is included here and also serves as a link to the single model page. This section also contains the IDs of the experiment and experiment run that the model belongs to. If users decide to assign a human-readable name for a model, that is also displayed here.

2. DataFrame - This section contains the various pieces of information related to the model’s dataframe. A dataframe is a 2-dimensional labeled data structure where columns can be potentially different types. In the context of ModelDB, the dataframe represents the set of data that was used to build the model. In
addition to the dataframe ID, this column also contains optional dataframe tags and file paths.

3. **Specifications** - This section contains the model type as well as various hyperparameters that are associated with the model. Since there may be a large number of hyperparameters, they are initially collapsed. Clicking on the blue “hyperparameters” text toggles between the collapsed and expanded states.

4. **Metrics** - This section contains the various metrics that are used to evaluate the model performance. A single model may have multiple metrics, and different models may have different metrics. All metrics are displayed here as key-value pairs.

5. **Miscellaneous** - The last section contains miscellaneous information about the model. This includes information like the timestamp and the model file path. The miscellaneous section also contains a way to add annotations and view custom metadata, both of which are described below.

ModelDB is constantly being updated with new features, so new types of information might be logged which do not fit into the existing sections. To prepare for the potential addition of new fields, each row also contains a section for additional information. This section is accessible via the “See More” button at the bottom right corner of each row. The additional information section contains various key-value pairs that do not fit in the main view, and the placement of these pairs is not subject to the five organizational columns.

**Annotations**

During the model building process, a data scientist might find it useful to make certain annotations on models. For example, the user might want to leave a note on a model that performs particularly poorly so that it can be revisited later. Leaving notes on models is also useful for collaboration, since team members can share insights or comments with each other.
In ModelDB, users can add annotations to models via the miscellaneous section of the models table. Under this column, there is an entry that shows the most recent note. There is also a button next to this entry that opens a modal window for entering new annotations. As seen in Figure 4-14, the modal window shows all of the previous annotations on the model and includes a text box to add new annotations.

![Figure 4-14: Model annotations modal window](image)

**JSON Metadata**

While ModelDB automatically logs many of the relevant model hyperparameters and metadata, data scientists may wish to include their own custom attributes that are not supported by default. ModelDB provides the appropriate flexibility by allowing users to import their own custom metadata using JavaScript Object Notation (JSON).

Models that have been augmented with custom JSON metadata have an additional button labeled “Metadata” under the miscellaneous column of the models table. The button opens a modal window that displays the JSON metadata with proper formatting (Figure 4-15). The pretty-json library is used here to format the JSON, and it supports interactive folding of nested objects as well as color-coded values. The
key-value pairs shown in this view also work with the drag-and-drop filtering which is described later in section 4.6.3.

Figure 4-15: JSON metadata modal window

4.6.3 Querying Models

One of the main goals of the ModelDB user interface is to provide a method of querying models that is simple and intuitive, yet powerful enough to fulfill most needs. In this section, we describe the various ways that the interface allows users to filter, group, and sort the models in a given project. Together, these operations are sufficient for selecting a desired subset of models to focus on for further analysis.

Filtering

A project may have hundreds of models, each with different parameters and metadata. During analysis of previously built models, users might want to focus only on a subset of models, such as those with a specific type or a certain parameter. ModelDB supports this through an interface of drag-and-drop filtering.
Each piece of data associated with a model in the database exists as a key-value pair. At a high-level, filtering on a certain key-value pair involves showing only the models that contain the selected key with the specified value. Filters can also be combined so that models must satisfy multiple filters in order to be shown. By combining filters, users can make arbitrarily complicated selections. For example, a data scientist might ask for all linear regression models with a regularization parameter of 0.1 and an accuracy greater than 0.8.

While filters can be expressed with a structured query language, our goal in the ModelDB interface is to support filters without requiring users to learn any complicated language for querying. Instead, filters are specified by dragging and dropping any key-value pair that is found in the models table. Existing tools for data visualization like Tableau have already demonstrated the effectiveness of a drag-and-drop interface\(^1\). In particular, the ability to directly drag and manipulate key-value pairs in the models table minimizes any mental discontinuity between model exploration and querying. A data scientist who finds an interesting parameter while inspecting a model can immediately choose to filter on that value without having to switch modes. Compared to the alternative solution of using complicated menus, the drag-and-drop interface is also cleaner and more intuitive.

Hovering over a draggable key-value pair changes the cursor and highlights the pair to indicate that it is indeed draggable. Users can then drag the pair over to the filters section in the left menu, where the it will snap into place. Users are able to drag any number of filters into the menu. Having multiple filters corresponds to a logical AND, so models will only be shown if they satisfy all of the filters.

As seen in Figure 4-16, users can make modifications to filters once they have been added to the menu. By pressing a toggle to expand the filter, users are presented with the options to invert the filter or edit the desired value. When editing the value, users can modify the filter to accept multiple values by separating them with commas in the text box. A filter with multiple values corresponds to a logical OR, so models will satisfy the filter as long as they match any of the specified values for the given

\(^1\text{https://www.tableau.com/}\)
key. For filters on keys that have numerical data, users are also able to specify a range of values.

The draggable key-value pairs are not limited to the ones that are immediately visible in the model table. The system design is standardized so that key-value pairs are draggable wherever they occur in the interface. This provides both consistency and convenience, allowing users to filter whenever they are inspecting any sort of model data. In particular, key-value pairs can also be found in collapsed hyperparameters (Figure 4-16), JSON metadata modal windows (Figure 4-15), and model cards shown by the summary graph (Figure 4-11). For the JSON metadata, a short tag is prepended to the keys so that they do not conflict with existing ModelDB keys.

Once all of the desired filters have been dragged into place, clicking the “Filter” button will apply the filters to the models. The table is reloaded, and the model div elements that satisfy the filters are re-inserted lazily as the user scrolls down the table. In addition to the models table, the filtering is also applied across the entire models page. This means that the summary graph and custom visualizations described in

Figure 4-16: Filtering models

In this example, the user has added three filters. The filter on model type has been expanded to show additional filter options. The cursor is currently over the regParam hyperparameter, which can be dragged directly from the table into the menu.
section 4.6.1 will both be updated to display the filtered model data as well.

**Grouping**

A useful operation during model analysis is grouping models by a particular field. This allows the data scientist to easily compare across all models with a specific field value. For example, the user might want to group by model type and then compare across models of each type. This could in theory be done by filtering the models for a specific type, but the user would have to update the filters to look at each new type. Instead, it is easier to create several groups of models and then switch between them seamlessly. In addition, filters can be added on top of groups for even more flexibility.

To group the models, users simply select the field that the data should be grouped on. This is accessible via a drop-down list in the left menu. Grouping is only relevant to the models table, so this section of the menu is only visible when the user is viewing the models table tab. Once a field is selected, the system goes through the models and makes note of all possible values of the selected field. One group is made for each unique value that occurs, and each model is assigned to the group corresponding to its value for the selected field.

As shown in Figure 4-17, groups are visualized in the interface with a vertical bar that contains different colored sections. Each section is a group that corresponds to a certain key and a value that the key takes on. The group contains all of the models that have the specified value for that key. The heights of the sections are proportional to the percentage of models that are in the corresponding group. This allows users to quickly get a sense of the relative number of models in each group. In the event that there are too many groups, buttons are added above and below the vertical bar so that users can scroll through the groups.

Hovering over a certain section in the groups bar reveals a tooltip that shows the key-value pair associated with the group as well as the number of models in the group. Clicking on a section selects the group and reloads the table so that only models in the group are visible. If any filters exist, they are applied normally to the models in the group.
In this example the user groups by model type, which creates four groups. These are visualized by a vertical bar with four sections. Highlighting over a section shows a tooltip with the group name and size, whereas clicking a section selects the group.

### Sorting

For projects with hundreds of models, the ability to sort the models table is an essential feature. Users might want to sort on a certain metric to look at the models that have the best or worst performance. They might also want to sort on the other parameters and metadata that the models have. The ModelDB interface is flexible and allows users to sort on various fields in the model table.

Some columns may have multiple fields that the models can be sorted on. For example, the metrics column can contain different types of metrics to be sorted on. Users are able to select the specific field to sort on via a drop-down located in the table header, as shown in Figure 4-18. Otherwise, a default field is used for each column. Once the field is selected, users can sort on a column’s field by using the arrows located in the table header. Clicking on the arrow repeatedly will toggle between ascending and descending order.

It is possible that some models do not contain the field that is sorted on. In this case, the models that do contain the field are sorted normally, and the ones without...
Figure 4-18: Sorting models

The up and down arrows are used to sort on a column. If a column has several fields that can be sorted on, the triangle opens a drop-down list to select the desired field.

the field are displayed afterwards in increasing model ID order. When the page is first loaded, the models table is also initially sorted by increasing model ID.

4.7 Single Model Page

Within the models table on the models page, the model ID field in each row links to the single model page of the associated model. The single model page is intended to be a more complete view of all of the information about the model that is stored in ModelDB.

For simplicity and consistency, most of the information about the model is displayed in the same format as a single row in the models table (Figure 4-19). A more complete description about the layout of this data is included in the discussion of the models table in section 4.6.2.

Figure 4-19: Single model page
The single model page also includes a visualization which shows the pipeline for the model. This pipeline shows all of the transformers and dataframes which are part of the workflow that generated the model. By inspecting this pipeline, data scientists can see the various pre-processing and training steps that were taken when building the model. The pipeline visualization is implemented using the network module of vis.js.
Chapter 5

Prediction Visualizations

5.1 Overview

This section describes the design and implementation of prediction visualizations which are intended to augment the core ModelDB user interface and help users better understand model performance. We begin with a discussion of overarching design goals, followed by a description of the overall page layout. We then dive into each of the components individually for more thorough explanations of their features.

5.2 Preliminary Design

The overall goal of the prediction visualizations is to help data scientists better understand model performance. An effective interface would help users interpret the outputs of models and gain insights about how to debug or improve models. To understand what such an interface might include, we begin with an initial survey of existing research on visualizing model performance, followed by interviews with two machine learning practitioners. Based on these results, we outline several high level design goals for the interface.
5.2.1 Existing Research

To get a sense of the landscape of machine learning visualizations, we include a brief discussion of existing research. These studies are described in more detail in the related works (section 3), but we reiterate some of the major takeaways.

Existing research has often focused on a particular aspect of machine learning models. For example, various studies have investigated visualization techniques for showing summary statistics [15, 7], dealing with high-dimensional feature sets [12, 11, 9], and presenting predictions [8, 1]. While these papers made no novel contributions to specific aspects of visualizing model performance, the best systems often combine various components for a more complete experience.

One particularly useful design study also analyzed various existing tools for machine learning visualization [6]. The study interviewed participants to discover what made each system effective or ineffective, and it summarized findings by providing seven major design themes. We focus on some of the design themes provided and use them as guidelines for our own design:

1. **Overview and detail** - Participants stated that overview statistics provide a good summary of overall performance, but can also be misleading or not very informative. An overview should be used as a starting point, but the interface should also allow users to drill down into the raw data to determine what is actually going on.

2. **Group and compare** - Users found it useful to group instances by certain attributes and look for common patterns or notable differences.

3. **Data space, feature space, and prediction space** - All the visualizations studied contained one or two out of the three, but never all three. Participants often referred to the missing space as something that would be useful. Showing multiple spaces together can help answer questions and provide more insight than just a single space.

4. **Diagnostic assistance** - Participants wanted the visualization to highlight
places that were problematic and indicate errors requiring immediate attention.

For our system, we aim to provide a combination of visualizations that addresses these various design themes.

5.2.2 Interviews

In addition to the initial survey of existing literature on the subject, we also conducted interviews with two machine learning practitioners to discover what kinds of features and operations would be useful in their own work. Both participants have had machine learning experience in industry internships as well as research projects.

Both participants indicated that they usually started off with summary statistics to get a general sense of performance, before drilling down into the raw examples. When exploring predictions, participants were interested in questions like “What kinds of examples are consistently misclassified?” or “What was the model most unsure about?” Participants were also generally excited about using visualizations to help debug, but explained that the overhead for generating visualizations is often too high.

The interviews confirmed some of the themes found in the design study mentioned in the previous section, such as including an overview with detail or providing diagnostic assistance. They also helped to substantiate the utility of prediction visualizations for debugging models.

5.2.3 Design Goals

Based on the results of existing research on visualizing model performance and our own interviews, we adopted various high-level goals for the design of the interface:

1. Visualize model outputs - To begin understanding model performance, the user should be able to see the outputs of the models. For this interface, we assume that the models are built for binary classification and the outputs are prediction probabilities. The interface should show these predictions in a format that is easy to understand and explore. This can include visualizations of individual predictions as well as aggregate statistics.
2. **Comparison across models** - While the absolute accuracy of a model is useful, models are generally not built in isolation. A data scientist will have a variety of different models with modified parameters, and the interface should support comparison of performance between multiple models. Such an interface could answer questions like “Which models perform best on this set of examples?” or “Which examples do these models classify differently?” By comparing predictions across models, users can get a better sense of why some models work well and why others do not. They can then apply these insights in order to improve existing models.

3. **Example-level debugging** - To figure out where models perform poorly, data scientists often have to drill down to the features of misclassified examples in order to understand errors. The interface should allow users to inspect the raw data of examples easily. An effective system might also highlight examples of interest so that users can focus on areas that require immediate attention.

4. **Scalability** - Machine learning models often deal with a lot of data. Models may be used to classify a large number of examples, and examples may have a large number of features. Scalability in visualization refers to the problem of displaying all of this information effectively in a limited space. A scalable interface should be able to support thousands of examples with many features without getting too cluttered or introducing too much lag.

### 5.3 Overall Layout

The layout of the page for prediction visualizations is similar to that of the models page. As shown in Figure 5-1, there is a fixed navigation bar at the top and a fixed menu on the left. The rest of the page contains the main content.

The center section of the predictions page contains a matrix that shows the predictions of various models and examples. On the right side of the matrix are options for sampling predictions, as well as options for grouping and filtering. This menu is
Figure 5-1: Predictions page
collapsible so that matrices with a large number of columns can expand to take up the entire screen space.

Below the matrix is a space for visualizations related to summary statistics. These are initially empty and will populate once the user selects a model from the matrix. The selection mechanism is described later in section 5.4.5, whereas the summary visualizations are described in section 5.5. Finally, there is a table at the bottom which shows the raw data of the examples.

5.4 Prediction Matrix

5.4.1 Overview

The prediction matrix is the centerpiece of the page for prediction visualizations (Figure 5-2). It displays predictions, which are the outputs of models on input examples. With the appropriate color encodings and interactions, the matrix is able to convey a wealth of information about both the aggregate and example-specific outputs of models. The matrix is implemented using the D3.js library, and borrows from an existing example of an interactive heatmap.¹

For this interface, we restrict ourselves to the problem of binary classification. The models classify examples as either 0 or 1, and the prediction value for each example is the probability that the example is classified as 1. Intuitively, a lower value means that the example is more likely to have a label of 0, whereas a higher value means that the example is more likely to have a label of 1. In section 7, we discuss how the system might be generalized to handle multi-class classification.

The rows of the matrix correspond to the data examples which are being classified, whereas the columns of the matrix correspond to models in ModelDB. The first column is reserved for the ground truth (GT), which is the actual value of the label being predicted. Besides the first column, each cell in the matrix is at the intersection of a model and an example. Cells show the model’s prediction for the given example

¹http://bl.ocks.org/ianychang/8119685
and scale in size to fit the page properly.

Hovering over column labels shows a tooltip with the model type and overall accuracy. Hovering over a cell shows a tooltip with the model ID, example ID, and prediction value. This is useful because the color of the cell does not always correspond to the raw prediction value, as described in section 5.4.2. Hovering over a cell also highlights the nearest neighbors of the corresponding example, as described in section 5.6.1.

5.4.2 Encoding Schemes

The interface provides a few different ways of visually encoding the raw prediction values. A legend is included above the prediction matrix to help the user better understand the current encoding scheme, and users can switch between schemes using the drop-down list located in the left menu. As shown in Figure 5-3, the four in-
cluded schemes show correctness, raw prediction values, binary classification outputs, and distances from ground truth. By providing different schemes that are useful for performing different types of visual analysis, the matrix is able to convey a lot more information than a standard spreadsheet.

![Prediction matrix encoding schemes](image)

Figure 5-3: Prediction matrix encoding schemes

The four encoding schemes are correctness (top left), raw values (top right), binary classification (bottom left), and distance from ground truth (bottom right).

The four encoding schemes are described below:

1. Correctness - This is the default scheme that the matrix uses upon initial page load. The correctness scale shows how “correct” a prediction probability is, based on the current threshold. Probabilities that are close to the threshold show up as white or near-white, indicating that the model is not quite sure one way or the other. The farther the prediction is from the threshold in the
correct direction, the more green the cell gets. On the other hand, the farther the prediction is in the incorrect direction, the more red the cell gets. This encoding scheme is useful for quickly identifying misclassified examples and their severities.

2. **Raw prediction values** - This scheme simply maps the prediction probabilities onto a linear scale from 0 to 1. As indicated by the legend, orange corresponds to a prediction of 0 and blue corresponds to a prediction of 1. Intermediate shades correspond to intermediate values. These colors were chosen because they are relatively neutral and do not have any implicit meanings, which might be present for colors like red or green.

3. **Binary classification** - This encoding scheme shows whether the prediction probability would be classified as 0 or 1, based on the current threshold. The view is updated as the user modifies the threshold, and provides a quick way of seeing how prediction probabilities translate to actual classifications. It also allows users to visually explore the effects of various threshold values.

4. **Distance from ground truth** - This scheme shows how far a prediction probability is from the ground truth. Light cells are closer to the ground truth, whereas darker cells are farther. Using this scheme, users can see which examples are very far from the ground truth and require further debugging. Since the scheme is only based on ground truths, the view is not affected by changes to the threshold value.

### 5.4.3 Sorting

As with the models table, sorting is a useful operation in the prediction matrix that allows data scientists to focus on models with the best or worst results. Users can sort the rows and columns of the prediction matrix in various ways. The drop-down list in the left menu section allows users to sort rows by ascending or descending example IDs, and columns by ascending or descending model IDs. Users can also sort the
matrix by the raw prediction values of any row or column by single-clicking on a row or column label. Subsequent single clicks on the labels will toggle between ascending and descending order. By default, the examples are sorted by ground truth and the models are sorted in increasing model ID order.

5.4.4 Clustering

When analyzing projects that contain a large number of models and examples, it may be useful to cluster the columns and rows so that similar items appear next to each other. First of all, this allows users to see which examples are similar to each other, a task that can be difficult if the examples have a large number of features. Rearranging columns and rows in this way can also help data scientists find patterns in the matrix, such as a patch of similar examples that are repeatedly misclassified. Coupled with the various encoding schemes, clustering helps users quickly notice classes of examples or models to focus on.

To cluster the matrix, users select a specific type of clustering using the drop-down list in the left menu section. There are options for hierarchical clustering on prediction values, hierarchical clustering on raw data, and k-means clustering on prediction values (Figure 5-4).

Hierarchical clustering is particularly useful when the number of clusters is not known ahead of time, which is often the case when exploring examples and their predictions. The system supports hierarchical clustering on both prediction values and raw data. On the other hand, k-means clustering allows the user to specify the desired number of clusters. Due to limitations of the clusterfck.js library that is used to perform the clustering, k-means clustering is currently only implemented for prediction values. The three types of clustering provided by the system are described below:

1. **Hierarchical clustering on prediction values** - The system clusters both the rows and columns on raw prediction values. Each row or column of prediction values is represented as a vector containing values between 0 and 1. This type of
clustering results in neighboring rows and columns that have similar prediction values.

2. **Hierarchical clustering on raw data** - The system uses the raw data vectors associated with each example. To prepare those vectors for clustering, the system first uses z-score standardization to scale them to an appropriate range. This allows the features to be compared using a simple euclidean distance measure. This type of clustering results in neighboring rows with similar raw data fields. Naturally, clustering on the raw data does not change the order of the columns.

3. **K-means clustering on prediction values** - Upon selecting this option, users are prompted to enter a value of $k$. The system then clusters the rows into $k$ clusters, using the raw prediction values as vectors. Again, this results in neighboring rows that have similar prediction values.

![Figure 5-4: Prediction matrix clustering](image)

Users can cluster the prediction matrix using hierarchical clustering on prediction values (left), hierarchical clustering on raw data (middle), or k-means clustering on prediction values (right).

### 5.4.5 Selection

The prediction matrix also supports selection of specific models and examples for further analysis. Selection of a row or column is done by double-clicking the corresponding row or column labels. The system responds by highlighting the selected row or column in the prediction matrix (Figure 5-5). As described in section 5.5,
selecting a model will bring up summary visualizations for the model. As described in section 5.6.2, selecting an example will reveal a card that contains the raw data as well as a plot of the normalized data vector. Users can select any number of models or examples in order to compare across selections. Double-clicking a previously selected row or column will undo the selection.

![Figure 5-5: Selecting rows or columns in the prediction matrix](image)

5.5 Summary Visualizations

Existing studies and our own interviews with machine learning practitioners both demonstrated user interest for aggregate statistics and summary visualizations. These summary visualizations can show the overall model performance, giving users a solid starting point for further analysis. In our prediction visualizations interface, we offer
three commonly used visualizations for summarizing classifier output quality: ROC curves, PR curves, and confusion matrices.

![ROC and PR curves](image)

**Figure 5-6: ROC and PR curves**

### 5.5.1 ROC Curves

Receiver Operating Characteristic (ROC) curves are commonly used to evaluate the quality of a classifier’s output. The curve plots the true positive rate on the Y-axis against the false positive rate on the X-axis by varying the threshold value for binary classification. The ideal point on the graph would be the top left, where the true positive rate is 1 and the false positive rate is 0. Since this is not very realistic in practice, data scientists usually aim to find a model that maximizes the area under the curve instead. Using the ROC curve, users can also observe the tradeoff between the true positive and false positive rates in order to select the best threshold.

The graph for the ROC curves is located below the prediction matrix (Figure 5-6). It is initially empty on page load, and is populated when users select a model by double-clicking on a column label in the prediction matrix. Once a model is selected, the system generates the curve by varying threshold levels to calculate the true positive and false positive rates for points along the curve. By default, the step size for varying the threshold is 0.01, so each curve has 100 points. If multiple models are selected, the corresponding curves are overlaid on the graph and a legend is included to identify each curve.
5.5.2 PR Curves

The Precision-Recall (PR) curve is another commonly used metric for evaluating classifier output quality. Precision measures how relevant the results are, and is defined as the number of true positives divided by the total number of true positives plus false positives. Recall measures how many relevant results are returned, and is defined as the number of true positives divided by the total number of true positives plus false negatives.

A low precision with high recall means that the model returns many results but most of them are incorrect. On the other hand, a high precision with low recall means that the model returns only a few results, but most of them are correct. An ideal model will have both high precision and high recall, so that it is returning a large number of correct results. Like the ROC curves, the graph of the PR curves is also located below the prediction matrix (Figure 5-6). The PR curves are generated using the same process that is used for ROC curves, and are added to the graph as users select models from the prediction matrix.

5.5.3 Confusion Matrices

Another useful summary visualization for evaluating the quality of a classifier is the confusion matrix. Each column corresponds to a predicted class and each row corresponds to an actual class. For the problem of binary classification, there are two rows and two columns, since the classes can only be 0 or 1. Each cell in the matrix shows the number of instances that have the corresponding predicted and actual classes.

Cells along the main diagonal represent instances where the predicted label and the actual label are equal. These are true positives and true negatives, and are colored green to indicate that they are correct classifications. The remaining cells represent false positives and false negatives, and are colored red to indicate that they are incorrectly classified. An ideal model would only have instances along the main diagonal. Confusion matrices can help data scientists quickly understand the overall model performance as well as the kinds of misclassifications that are being made. The
confusion matrices also update as the user updates the threshold, allowing users to see if different threshold values can produce better results.

The confusion matrices are displayed below the ROC and PR curves when users select models. As shown in Figure 5-7, selecting multiple models will display multiple confusion matrices so that users can compare across models. Hovering over the model name will also show a tooltip that contains basic information about the model, like model type and overall accuracy.

![Confusion Matrices](image)

Figure 5-7: Confusion matrices

5.6 Example-level Debugging

While comparisons of model hyperparameters and overall accuracy can lead to improvements, data scientists will often want to debug models at the example level to determine where the model makes mistakes. This can involve drilling down to the raw data itself to see what features might be the cause for misclassification. To facilitate example-level debugging, the predictions page surfaces raw data and provides various features that enable users to find and compare similar examples.

5.6.1 Highlighting Nearest Neighbors

When debugging a misclassified example, a data scientist might find it useful to look at how the model classifies similar examples. To enable quick exploration of similar examples, the prediction matrix highlights nearest neighbors in the raw data.
space. Hovering over any cell of the matrix shows the three nearest neighbors for the corresponding example, as shown in Figure 5-8.

![Figure 5-8: Nearest neighbors in the prediction matrix](image)

The user has selected example 684, whose ground truth cell is highlighted in black. The three cells highlighted in red and connected with dashed lines are the three nearest neighbors to example 684 in the raw data space.

To calculate the nearest neighbors, the system first normalizes all of the raw data vectors using z-score standardization. This is done so that all values are scaled down to a range where distances between features can be compared. After the normalization, the system picks the three closest neighbors based on a euclidean distance measure. The system then highlights these neighbors in the prediction matrix.

The visualization updates automatically as the user hovers over different cells, so that it always shows the nearest neighbors of the example that the user is currently examining. Users can also click on a cell to freeze the dotted lines in place, which can be useful if they wish to revisit those examples later.
5.6.2 Quick Inspection of Data

By selecting examples from the prediction matrix, users are able to quickly inspect all of the raw data fields of that example without having to refer to the table at the bottom of the page. As shown in Figure 5-9, a card with all of the raw data slides in from the right when the user selects an example. This interaction allows users to seamlessly switch between exploration in the prediction space and exploration in the raw data space. When multiple examples are selected, users can also plot the examples to visually compare their data. Field values are normalized for the plot, so that the relative differences between each feature can be visualized.

![Figure 5-9: Inspecting examples from the prediction matrix](image)

5.6.3 Raw Data Table

The predictions page provides a table of the raw data of examples, allowing users to explore the raw data space in a more traditional tabular format (Figure 5-10). Users can search through the table using the text box, and they can sort on any
column by clicking the triangles found in the table header. The table is implemented using the DataTables library and designed so that it can work with a large number of rows (examples) and a large number of columns (features). Pagination is used to deal with large amounts of examples, and column toggles are used to show or hide desired columns. Finally, the filters (described in section 5.7) that are applied to the prediction matrix also apply to the table.

5.7 Filtering and Grouping

During the model debugging process, it may be useful to group similar examples together for comparison, or focus on a particular subset of examples for analysis. The predictions interface supports these operations through a system of filtering and grouping, which allows users to filter by attributes of the raw data and create groups of examples.

Using the menu to the right of the prediction matrix, users can create a “filter group.” A filter group is basically a subset of examples subject to one or more filters.
Each group has a name and at least one filter, which consists of a selected key and a value to match for. An example is part of a particular filter group if it passes all of the filters of the group. In Figure 5-11, the user has created a filter group named test_group with two filters: one that checks that the ground truth is 1 and another that checks that the age is 19. This group would contain all examples that have a ground truth of 1 and an age of 19.

![Figure 5-11: Menus for filtering and grouping](image)

Users can manually create filter groups (left) or automatically generate filter groups based on a selected key (right).

The filter groups provide a great deal of flexibility, but creating many filter groups can get tedious. The interface allows users to automatically generate filter groups by selecting a key from the drop-down list shown in Figure 5-11. The system then finds all of the possible values of this key and automatically generates a filter group for each one. In Figure 5-11, the user has chosen to automatically generate filter groups based on the age field.

Once the user has created one or more filter groups, an aggregate matrix view is displayed above the normal prediction matrix (Figure 5-12). The columns of this aggregate matrix are models just like the columns of the regular prediction matrix, but each row in the aggregate matrix corresponds to a filter group rather than a single example. Hovering over the ground truth cell in a particular row shows a
tooltip that contains the number of positive and negative examples which belong to that filter group. Instead of raw prediction values, cells in the aggregate matrix show the average accuracy of the model based on the examples in the filter group. This makes it easy to see how the models perform on specific subsets of examples.

Double clicking the row label of a filter group in the aggregate matrix will filter the prediction matrix down to only the examples in that group. For example, in Figure 5-12 the user selected the filter group that contains examples with an age of 29. As a result, the prediction matrix is filtered so that only examples with an age of 29 are shown.

![Figure 5-12: Aggregate matrix](image)

In this example, the user has automatically generated filter groups based on the age field and selected a filter group for the age of 29. This filters the prediction matrix down to nine examples that have an age of 29.

To summarize, the user is able to create filter groups, which are subsets of examples that have desired properties. The aggregate matrix provides a way of visualizing the average performance for each of these filter groups. Selecting a certain filter group limits the prediction matrix to only those examples.
5.8 Classification Threshold

For the problem of binary classification, the classification threshold decides how prediction values are interpreted. In particular, examples with prediction values greater than the threshold are classified as 1, whereas the rest are classified as 0. Data scientists are often interested in how classifier outputs change in response to the threshold. Depending on the task, data scientists may want to shift the threshold to optimize various metrics such as the true positive rate.

The left menu section of the predictions interface contains a slider which adjusts the global classification threshold. This is the threshold used throughout the page, and visualizations are updated automatically in response to changes to the slider. The threshold affects various parts of the page: the prediction matrix under the correctness and binary classification schemes, the average accuracies in the aggregate matrix, and the confusion matrices. To avoid confusion, the threshold slider is only visible when the page contains some visualization that can be affected by changes to the threshold value.

5.9 Scalability

Machine learning deals with a lot of data, so it is important that visualizations are scalable. For visualizing predictions, the interface should be scalable with respect to the number of examples as well as the number of features in a given example. In this section, we describe how the prediction visualizations deal with issues of scalability.

For projects that have hundreds or thousands of examples, the system cannot comfortably fit all of the predictions into the prediction matrix. The matrix would take too long to scroll through, and the number of elements rendered on the page would cause the system to be unresponsive. Instead, the system randomly samples a number of examples to show in the matrix. This option is accessible via the menu to the right of the prediction matrix, and subsequent clicks on the option will refresh the matrix with resampled points. In addition to random sampling, the filtering
and grouping operations described in section 5.7 also help with issue of scalability, since they are used to filter the prediction matrix down to a smaller set of selected examples.

As mentioned in section 5.6.3, the data table also includes some functionality to provide scalability with respect to the number of features. Users are able to toggle the visibility of columns so that they can focus on some subset of features. There is also a button for hiding or showing all of the columns at once, since it can be tedious to select everything individually.

Finally, while the system does not currently provide any way to limit the columns of the prediction matrix, there are straightforward integrations with the rest of the ModelDB interface that could help with scalability in this aspect. In particular, the system can be augmented to leverage existing methods of filtering models on the models page.

5.10 Integration with ModelDB

The prediction visualizations page is currently implemented as a standalone page which reads data from local CSV files. Since limited time was available for implementation, priority was given to the completion of major features rather than a full integration with ModelDB. In this section, we discuss how the predictions page might fit in with the rest of the ModelDB user interface.

The predictions shown on the predictions page belong to a set of models, so the visualizations could actually exist as a third tab on the models page. This would allow the system to leverage the existing infrastructure for filtering models. Users could filter models normally on the models page, and only the filtered models would show up within the prediction matrix.

The interface for predictions currently makes minimal use of model data. The only part of the interface that uses model data are the tooltips, which show model type and accuracy. The interface could benefit from the additional data that is stored in ModelDB. Users would be able to inspect model hyperparameters and metadata while
viewing the prediction matrix, allowing them to gain more insights about how these parameters are manifested as different prediction values. Furthermore, the columns of the prediction matrix could be sorted or clustered on model hyperparameters, allowing users to better compare the outputs of similar models.
Chapter 6

Evaluation of Prediction Visualizations

This section evaluates the prediction visualizations through a pilot user study. We begin with the user study methodology, followed by results of the study and a discussion.

6.1 Methodology

Due to time constraints and the incomplete integration of the prediction visualizations with ModelDB, we opt for a pilot experiment rather than a full-fledged user study. The results of the pilot experiment will help refine the design of the user study and provide preliminary feedback on the user interface.

Four participants were recruited for the pilot experiment based on their experiences with machine learning. After signing consent forms, participants filled out pre-study questionnaires which asked about their backgrounds in machine learning and their current practices. Participants were also informed that their screen and audio would be recorded for the experiment.

The four participants were split into two groups of two, with each group testing a specific variant of the prediction visualizations interface described in this paper. In variant A, the interface was modified to show only the predictions of a user selected
model along with the predictions of the previous model. In variant B, the unmodified interface showed the predictions of all models. The two variants are intended to help analyze the benefit of comparisons across multiple models, in contrast to existing systems like ModelTracker which only highlight differences across consecutive iterations [1].

After the pre-study questionnaire, participants were given a demonstration of their specific variant of the interface. The demonstration highlighted the main features of the system and walked through various examples of usage. Participants received access to the entire ModelDB frontend, including both the core model management interface as well as the prediction visualizations page. However, participants were asked to focus mainly on the prediction visualizations page when testing the system.

Once participants became familiar with the system, they were presented with a dataset, as well as 10 models that were previously built using the dataset. The fictitious dataset was taken from the IBM HR Analytics Employee Attrition & Performance kaggle project, and contains relevant employee information.¹ The models were built to predict employee attrition, which is whether or not the employee left the company. This is a binary classification problem, where a label of 1 corresponds to an employee that has left the company.

Participants were also presented with a Jupyter notebook containing the python code that was used to build the models in scikit-learn. The project contained 10 previously built models with varying parameters, including 4 logistic regression models, 3 stochastic gradient descent (SGD) classifiers, and 3 random forest classifiers. The model data and predictions were both imported into the interface for participants to test.

Participants were given the task of using the interface (specifically the prediction visualizations) to gain any insights on how they might improve the models. They were free to suggest concrete modifications to the python code, or simply discuss what high-level changes they would attempt. Participants were also asked to think out loud and provide a stream of consciousness as they explored the interface, so that

¹https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset
the investigator could take notes on interesting interactions and explicit feedback. At any point during the experiment, participants were free to ask about any part of the interface or task that they found confusing.

At the end of the task, participants were given post-study questionnaires to gauge their opinions on the presented interface. The entire pilot experiment lasted up to one hour per participant, and the results were anonymized and stored securely.

6.2 Results and Discussion

The results of the pilot experiment are useful for refining both the design of the user study and the design of the system interface. They also provide some preliminary feedback on how useful the interface is for understanding and debugging models. This section describes some of the major takeaways from the experiment.

- **Participants had a generally positive experience with the interface.** In the post-study questionnaires, participants indicated that they either “somewhat liked” or “extremely liked” the interface. When asked about the tool’s impact on their modeling processes, participants gave two responses for “medium positive impact,” one response for “large positive impact”, and one response for “no impact.” Participants mentioned that they found it useful to see the predictions, and that the system helped them diagnose problems.

- **The interface is effective for identifying problems.** Participants were largely interested in misclassifications, and the correctness scheme quickly directed them towards these errors. Using the prediction matrix, participants immediately noticed that the SGD classifiers performed very poorly compared to the other model types. They observed that these classifiers were outputting prediction values of 0 for all examples, regardless of ground truth. Most participants suggested sticking to linear regression models, which seemed to perform best.

Participants also noticed that a large majority of examples had a ground truth
of 0, and that this might be a problem. They realized that most of the misclassifications had a ground truth of 1, so the model accuracies were skewed by the relative proportions of each class in the dataset. In response to these insights, one participant suggested a change to the loss function that penalizes examples with a ground truth of 1.

- **Summary visualizations provide a good overview of overall performance.** All participants found the summary visualizations (ROC curves, PR curves, confusion matrices) to be helpful for getting a sense of overall model performance. They did not replace the prediction matrix, which users still used for exploring model outputs. However, the summary visualizations were useful as a starting point for analysis.

- **Comparing between multiple models is useful.** The participants who tested variant A of the interface wished that they could compare more models, or have more control over which models to compare. On the other hand, participants who tested variant B found it useful and easy to compare across models using the prediction matrix. This difference in user feedback supports the idea that showing predictions across many models in the prediction matrix is an improvement over existing systems like ModelTracker, which only highlight changes in predictions between consecutive iterations of models [1].

- **The predictions page needs to make more use of model data.** The prediction matrix is currently making limited use of model data. In particular, it just shows the model type and overall accuracy when users hover over the model label. However, ModelDB has a wealth of information about the models, including hyperparameters and metadata. Participants mentioned that it would be useful if operations that were present for examples (e.g., filtering, clustering, sorting) also existed for models. This would allow them to focus on a certain subset of models, or group similar models together for comparison. Access to the model parameters directly from the predictions page would also help users
put predictions in context without having to refer back to the models page.

- **The system needs to be integrated with the backend for the full user study.** Since integration is not yet complete, the users were limited in how they could improve the models during the pilot experiment. They gave suggestions about what changes they would make, but were unable to actually implement them. A more effective user study would have participants actually build additional models with scikit-learn and then evaluate the results in the ModelDB interface. This experience would be closer to the iterative nature of model building, and would illustrate how participants might use the system in practice. In order to achieve this, the remaining API calls need to be implemented so that the interface fetches data from the ModelDB backend rather than from a local file.
THIS PAGE INTENTIONALLY LEFT BLANK
Chapter 7

Future Work

In this section, we describe possible avenues for future research. We discuss updates to the system that are suggested by the results of the pilot experiment, as well as other potential extensions to the system.

As discussed in the previous section, the user study and system interface could benefit from a deeper integration between the prediction visualizations page and the rest of the ModelDB system. The predictions page is implemented as a proof of concept that manually reads in prediction data from local files. An obvious next step is to connect the page to the ModelDB backend via API calls. This allows for a more seamless experience in which the data logged by ModelDB are automatically populated into the interface, allowing users to properly incorporate the tool into their iterative model building processes. Integration with the ModelDB backend would also support additional features in the prediction matrix, such as clustering or filtering based on model attributes.

Aside from infrastructural improvements, the predictions page could also be extended with additional visualizations for the machine learning pipelines associated with each model. As data scientists build models, they apply various steps to preprocess or transform the raw data. Perhaps the interface could visualize these various steps and show how examples evolve as they move through the pipeline. This would help data scientists understand the pipeline and identify the specific steps where errors are introduced.
Finally, while the prediction visualizations are designed for the problem of binary classification, machine learning models can often deal with more than two labels. Further research can be done to generalize the interface to multi-class classification. It is not immediately clear how the prediction matrix should be extended to visually encode predictions for multiple classes. One existing system uses a different color for each class when visualizing classification outputs [13]. Perhaps the prediction matrix could also use a different color for each class, with various intensities to indicate the different class probabilities. While ROC curves and PR curves do have natural extensions to multi-class classification, they will also require some modification in order to support comparison across multiple models.
Chapter 8

Conclusion

In this thesis, we presented the design and implementation of a user interface for ModelDB, an end-to-end system for machine learning model management. With ModelDB, data scientists are able to automatically track machine learning models in their native environment for subsequent analysis. The graphical user interface improves on the existing ModelDB system by providing new ways to explore, visualize, and analyze the logged data.

We also presented a set of prediction visualizations to extend the core ModelDB user interface. These visualizations allow data scientists to analyze the performance of models by looking at their outputs and exploring how different models classify different types of examples. Together, the core ModelDB user interface and the prediction visualizations allow data scientists to not only keep track of models, but to analyze them systematically and find ways to improve performance.

To evaluate the practicality and usefulness of the presented interface, we conducted a pilot experiment in which users performed a realistic model debugging task. We also reported the results of the study, noting that participants responded positively to various features of the interface. At the same time, the pilot experiment highlighted a few areas to improve upon before conducting a more complete user study.

Based on the results from the preliminary pilot study, the presented user interface provides novel and effective ways of visualizing both model data and predictions. The system equips users with the tools necessary to explore and visualize the data stored in
the ModelDB backend. With the addition of the graphical user interface, ModelDB is more useful than ever for data scientists who want to manage and ultimately improve their models.
Bibliography


