Demonstration of ModelDB: a system for managing machine learning models

Manasi Vartak  
MIT CSAIL  
Cambridge, MA 02139  
vartak@csail.mit.edu

Harihar Subramanyam  
MIT  
Cambridge, MA 02139  
hsubraman@mit.edu

Wei-En Lee  
MIT  
Cambridge, MA 02139  
weienlee@mit.edu

Srinidhi Viswanathan  
MIT  
Cambridge, MA 02139  
vviswana@mit.edu

Samuel Madden  
MIT CSAIL  
Cambridge, MA 02139  
madden@csail.mit.edu

Matei Zaharia  
Stanford University  
Stanford, California 94305  
matei@cs.stanford.edu

Abstract

ModelDB is an end-to-end system for managing machine learning models. It automatically tracks ML models and pipelines in their native environments, stores and indexes relevant data associated with pipelines, and allows users to run complex queries on this data. At the 2016 ICML ML Systems workshop, we presented our vision and early work on ModelDB. The system got a lot of interest from attendees as well as practicing data scientists. We have since expanded ModelDB capabilities and refined ModelDB to better support diverse workflows. In this paper, we propose to demonstrate ModelDB through two case studies from our early users. By means of these case studies, we seek to show how a model management system such as ModelDB can make the modeling process more efficient and make modeling data more accessible. We argue that by providing a system that tracks ML pipelines and metadata, supports model versioning, and allows easy comparison, we can give data scientists the power to quickly iterate on models, make results reproducible, and find insights faster.

1 Model management

Building a machine learning model is a trial-and-error-based iterative process. A data scientist will build tens to hundreds of models before arriving at one that meets some acceptance criteria. However, the current style of model building is ad-hoc and there is no practical way for a data scientist to track models that are built over time, reproduce an old model or even compare results between models. Lack of tooling can waste data science efforts, make the modeling process inefficient, and cause insights to be overlooked.

The iterative, ad-hoc nature of model building highlights an important problem, namely machine learning (ML) model management. Model management is the problem of tracking, storing and indexing ML models (and metadata) so that they can later be searched, reproduced, compared and analyzed. In [5], we introduced the vision for ModelDB, a novel end-to-end system for the management of machine learning models. Specifically, we proposed a system to automatically track ML models and pipelines, store and index this data, and allow users to run complex queries on it. The system generated a lot of interest at the ICML ML Systems workshop and from practicing data scientists. We have since expanded ModelDB capabilities and refined it to support diverse modeling workflows. In this paper, we propose to demonstrate ModelDB through two case studies from our early users.

ModelDB is related to workflow systems such as Airflow\(^1\) and FBLearner Flow\(^2\) which have recently been developed at large web companies, as well as scientific workflow management systems such as Kepler\(^3\), Taverna\(^6\), and VisTrails\(^1\)\(^3\)\(^4\). ModelDB differs from generalized workflow systems in that it is designed specifically for machine learning; as a result, it supports ML-specific operations such as storing models and hyperparameters, querying models, and model comparisons that are largely absent from other workflow systems. ModelDB also differs from Airflow and FBLearner Flow in that it does not introduce a new language or environment for defining workflows. ModelDB APIs and native clients allow data scientists to continue using their favorite ML environment and get the functionality of ModelDB with minimal code changes. Moreover, ModelDB avoids the drawbacks of many scientific workflow systems that require the user to specify workflows using tool-specific GUIs and limit workflow components to predefined libraries.

In the following section, we describe the ModelDB architecture and functionality. In Section 3 we describe two case studies based on early users of ModelDB.

2 ModelDB Architecture and Functionality

ModelDB is an end-to-end system for ML model management. It supports all parts of the model management process starting with ingesting ML models and pipelines, to storing and indexing this data, and ending with exposing APIs as well as a user interface to access the data in ModelDB. Figure 1 shows the high level architecture of ModelDB.

ModelDB APIs and clients are designed to be compatible with a wide variety of ML environments. Data scientists often use diverse ML environments (e.g. scikit-learn\(^4\) in Python, spark.ml\(^5\) in Spark, R libraries, Torch in Lua\(^6\)) and occasionally also build workflows spanning different environments or libraries (e.g. scikit-learn + gensim\(^7\)). To accommodate the variety in ML workflows, ModelDB defines a set of high-level APIs that can be called from any environment using the corresponding language bindings. For example, for a workflow spanning

\(^{1}\)http://nerds.airbnb.com/airflow/
\(^{3}\)https://thrift.apache.org/
\(^{4}\)http://scikit-learn.org/
\(^{5}\)http://spark.apache.org/docs/latest/ml-guide.html
\(^{6}\)https://github.com/torch
\(^{7}\)https://radimrehurek.com/gensim/
scikit-learn and gensim, ModelDB APIs (generated for Python) can be called through both libraries to store information such as hyperparameters, intermediate results, and metrics.

Since the set of high-level ModelDB APIs are designed to be used in diverse ML environments, by design, they capture only coarse-grained information. To perform fine-grained logging (e.g. operator-level logging), ModelDB provides native clients for multiple popular ML libraries (currently scikit-learn and spark.ml). These native clients have been written to automatically drill down into functions and objects to capture relevant logging information. For example, the ModelDB scikit-learn client can automatically log the hyperparameters used in a TF-IDF operation and can inspect a LogisticRegression model to find its weights. When using these native clients, data scientists can continue to perform experimentation and model building in their favorite ML environment and get fine-grained logging by changing only a few lines of code.

3 Case Studies for Demonstration

In this paper we propose to demonstrate the use of ModelDB using the following two cases studies from our early users.

3.1 Tracking ML pipelines as they are built

Data scientists at a web startup use scikit-learn to model customer churn on their platform. Since even a small gain in the accuracy of their models can lead to a measurable difference in revenue, data scientists in this company are constantly testing out new ideas, algorithms and features to better capture user behavior.

Currently, it is most common for experimentation to be performed by writing a training script, changing the script repeatedly and re-executing it. Consequently, after updating the script a dozen times, the data scientist does not remember all the changes made to the script between the first model run and the current model run. For example, data scientists had a hard time answering questions like “what type of normalization was used in a given run?” or “what l2 parameter was used for this model?”

One of the data scientists at this company used ModelDB’s native scikit-learn client to track the pipelines that were generated across model runs. In this setup, every time the script was run, ModelDB would snapshot the code used for the experiment. The native client would then track dataframes, preprocessing operations (e.g. scaling, dimensionality reduction) as well as fitting operations and store them in ModelDB. Finally, the native client would export the model path and metrics to ModelDB.

Once the data was logged to ModelDB, the data scientist was able to quickly query ModelDB to examine how the models generated by different runs performed on the validation dataset. In particular, one of the experiments the data scientist was performing was varying the regularization parameter during training. Since ModelDB had automatically logged the regularization parameter for each run, ModelDB frontend enabled the data scientist to plot, with minimal effort, the effect of regularization on accuracy. In another case, the data scientist wanted to compare two model runs because the accuracy of one of the runs seemed off. Without ModelDB, the data scientist would not have stored the code for each model run and therefore would not have been able to perform this comparison at all. With ModelDB, however, he could visualize the pipelines used in both runs and compare the pipelines with respect to the operations as well as their parameters. In places where the pipeline differences seemed unclear, the data scientist used the code snapshot to identify differences.

Limitations. While this case study shows that ModelDB logging can significantly simplify model comparison (both with respect to metrics and pipelines), we found that the quality of pipeline comparison was strongly impacted by the native client’s coverage of operations. For example, when the data scientist wrote custom code for complex preprocessing, the native client was unable to extract meaningful information for this operation.

3.2 Central Repository for Models

A mid-sized web company has tens of models in production at a time. These models are updated periodically once new data is available or the data science team has a better model. Currently, each of
the models (which are owned by different teams) are stored in different locations, often without any organization. Answering a seemingly simple query such as “find all versions of the recommendations model” requires a manual search through multiple HDFS directories and another pass to find metadata associated with each model (e.g., training dataset, metrics etc.)

A data scientist at the company tested ModelDB as a central repository of models. Specifically, this meant that every time he trained and deployed a model, he would use the high-level ModelDB API to log the model, training configuration, data used to train, and metrics. Since the API allowed arbitrary key-value pairs to be associated with the model, he could tailor the metadata to be associated with the model.

Once the models had been logged into ModelDB, the data scientist could query models based on the metadata associated with them. For example, the data scientist wanted to track how metrics had changed over different versions of a model. To answer this question, he searched for models through ModelDB first filtering by task, then by dataset and then by time. Once he had identified all the versions of the model, he could plot accuracy vs. model version in a manner very similar to the previous case study. The data scientist also used the annotation functionality in ModelDB to add tags to different objects; for example, he annotated the dataset for a specific week as being anomalous due to unusually high user engagement. A colleague testing on that dataset could look it up in ModelDB and know to interpret metrics on that dataset with care.

Limitations. While the high-level ModelDB API is extremely flexible and can allow data scientists to log arbitrary information to ModelDB, the data stored by different data scientists may have a different schema. For example, one data scientist may store the number of features in a dataset as “num_features” while another may store it as “num_cols.” ModelDB cannot automatically identify these properties as being the same and therefore, it can be difficult to run queries across variable schemas.

The two case studies discussed above highlight the different advantages of a tool like ModelDB. Having an easy means to manage modeling experiments can enable faster iteration and comparison across models with minimal effort. A central model repository, in contrast, can make model-related information broadly accessible, whether it is metrics, pipelines or models. Together, we believe that ModelDB can make the modeling process more efficient and make ML models more accessible.

4 Conclusion

In this paper, we propose to demonstrate ModelDB — an end-to-end system for managing machine learning models. By means of two case studies from our early users, we propose to show how a model management system can make the modeling process more efficient and make information more accessible. We argue that by providing a system that tracks ML pipelines, supports model versioning, and allows easy model comparison, we can give data scientists the power to make results reproducible, to make modeling data accessible, and to find insights faster.

References


