ApproxML: Efficient Approximate Ad-Hoc ML Models Through Materialization and Reuse

Sona Hasani[‡], Faezeh Ghaderi[‡], Shohedul Hasan[‡], Saravanan Thirumuruganathan[‡†], Abolfazl Asudeh[†], Nick Koudas[‡‡], Gautam Das[‡]
[‡]University of Texas at Arlington; [‡†]QCRI, HBKU;
[†]University of Illinois at Chicago; [‡‡]University of Toronto
[‡]{sona.hasani@mavs,faezeh.ghaderi@mavs,shohedul.hasan@mavs, gdas@cse}.uta.edu,
[‡†]sthirumuruganathan@hbku.edu.qa, [†]asudeh@uic.edu, [‡‡]koudas@cs.toronto.edu

ABSTRACT

Machine learning (ML) has gained a pivotal role in answering complex predictive analytic queries. Model building for large scale datasets is one of the time consuming parts of the data science pipeline. Often data scientists are willing to sacrifice some accuracy in order to speed up this process during the exploratory phase. In this paper, we propose to demonstrate ApproxML, a system that efficiently constructs approximate ML models for new queries from previously constructed ML models using the concepts of *model materialization* and *reuse*. ApproxML supports a variety of ML models such as generalized linear models for supervised learning, and K-means and Gaussian Mixture model for unsupervised learning.

PVLDB Reference Format:
Sona Hasani, Faezeh Ghaderi, Shohedul Hasan, Saravanan Tirumuruganathan, Abolfazl Asudeh, Nick Koudas, Gautam Das. ApproxML: Efficient Approximate Ad-Hoc ML Models Through Materialization and Reuse. *PVLDB*, 12(12): 1906-1909, 2019.
DOI: https://doi.org/10.14778/3352063.3352096

1. INTRODUCTION

Machine learning has become a fundamental tool to gain insight from data. During the exploratory phase data scientists repetitively build numerous ML models in order to achieve higher accuracy. Consider a typical workflow of a data scientist. She issues a query to retrieve data from a data warehouse and builds an ML model (classification, clustering, etc.) on the retrieved data. The model is then used for analytic processing such as predicting revenue of a particular product. These analytic queries on ML models often have properties that allow a faster approach compared to building models from scratch. First, they usually have a specific business interpretation rather than being chosen at random. For example, a data scientist may want to retrieve data for a specific time period (month, semester, year) or for a specific location (city, state, country), etc. Moreover, data scientists are often willing to sacrifice some accuracy in the exploratory phase if they can obtain *good enough* approximate ML models very fast. In addition, data scientists and engineers from the same organization create many ML models for exploratory purposes that are discarded after one-time use. There is a very high chance that in future another member of that organization wants to build an ML model using the same data or a superset of it. Such properties render analytic queries good candidates for approximation as well as enable the potential to reuse their results fully or partially.

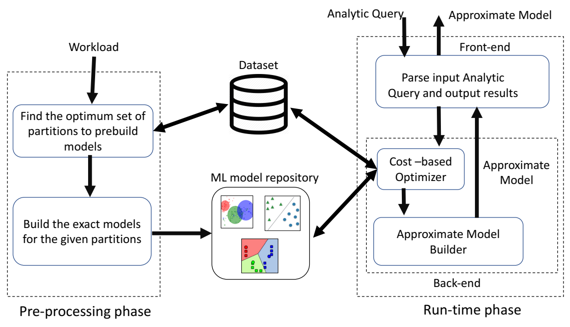
ML development revolves around experimentation. Recently systems such as mlflow [5] and modelDB [4] are developed to streamline the process by treating ML models as first class citizens and allow them to be stored with associated metadata. Nevertheless, building an ML model still remains a major bottleneck and consumes huge amount of time and resources due to sheer size of the datasets. If we can speed up the model building process by producing approximate models during an exploratory phase, it will dramatically improve the efficiency of the data scientist. In this paper, we introduce ApproxML, a system based on [1] that rapidly builds approximate ML models for analytic queries by utilizing two fundamental techniques *materialization* and *reuse* from database optimization. Suppose the analyst has access to pre-built ML models for each month and wants to build a model for the entire year. Currently, one builds the model from scratch using the data from the desired year. ApproxML allows one to combine the pre-built models to create model for that year much more efficiently.

ApproxML is a system that enables ML model approximation and reuse for popular supervised and unsupervised learning approaches. A demonstration session of ApproxML consists of three parts. The first part details the core features of the system by showcasing the approximate models supported. Next, the user will experience tradeoffs in accuracy of these approximate models compared with exact approaches. Finally, the user will be exposed on the practical utility of the materialization of ML models for a given workload and the ability to reuse them experiencing the associated speedups.

2. APPROXML OVERVIEW

ApproxML enables the user to build approximate model for popular supervised and unsupervised ML models for a given analytic query and a set of materialized models.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352096



Analytic Query
Approximate Model
Front-end
Workload
Dataset
Parse input Analytic Query and output results
Find the optimum set of partitions to prebuild models
ML model repository
Cost –based Optimizer
Approximate Model
Build the exact models for the given partitions
Approximate Model Builder
Back-end
Pre-processing phase
Run-time phase

Figure 1: ApproxML Overview

Technical Challenges There are several challenges to tackle in order to build efficient approximate ML models such as a)If we have access to a set of pre-built models, would it be possible to combine them in a few milliseconds to construct an approximate model instead of spending minutes/hours to build a model from scratch? b)How can we efficiently identify the relevant models among many possible choices? c)What information should be materialized for each model to make it reusable in future?

ApproxML generates approximate ML models in a two-phase approach. During "pre-processing phase", the model passively stores the ML models built by the data analyst to a model DB along with small amount of additional meta-data such as the data used and model parameters; During the "run-time phase", for a new query, it identifies the relevant and reusable pre-built ML models and efficiently constructs an approximate ML model from them.

ApproxML offers two orthogonal methods for generating approximate ML models, a)*model merging* approach, and b)*coreset-based* approach. In model merging approach, the relevant models are carefully merged without going back to the data. The merging based approach is often extremely fast. If there is a need for approximate ML models with tunable approximation ratio, ApproxML offers coreset-based approach where it builds a model from a chosen set of coresets. Coresets are a small weighted set of tuples such that ML models built from the coresets are *provably closer* to ML models built on the entire data. There has been extensive work on coresets for various ML models. For more details about the algorithms we used please refer to [1]. We implemented logistic Regression and linear SVMs as examples of Generalized Linear Models (GLMs) for supervised learning. In unsupervised learning, we implemented K-means and Gaussian Mixture Model (GMM). Figure 1 demonstrates the system overview of ApproxML.

2.1 Run-time phase

During the run-time phase, we assume we have access to a repository of the pre-built models. The user submits an analytic query through the front-end. Front-end will parse the the information about the dataset, the intended ML algorithm, the approximation method (model merging, coreset-based), etc. and will pass them to the cost-based optimizer in the back-end. The optimizer will retrieve all relevant pre-built models from the pre-built model repository. It will identify which of the retrieved pre-built models should be reused and what additional partial models have to be built from scratch. Then these partial models are passed to the "Approximate model builder" component to be combined efficiently to get the final approximate model. Details of each step is explained as follows.

Cost-based optimizer To identify optimum pre-built models to reuse is a major problem. Consider a dataset with 1 million tuples from year 2010 to 2015. Let's assume the relevant ML models for every month and every quarter are materialized. Given a new query for the entire year of 2014, there are many ways to answer it. One can build a model from scratch. Another option is to retrieve the materialized models for all 12 months of the year 2014 and build an approximate model by combining them. Alternatively, one can build a model by combining the materialized models for four quarters of year 2014. There are many more possible options for this simple example. Cost-based optimizer will retrieve all relevant pre-built models from the repository of pre-built models for the given analytic query. The cost-based optimizer finds the optimum set of pre-built models by taking into account different costs involved such as (a) cost of building a model from original data, (b) cost of merging a model and (c) cost of building a coreset. It is possible that some parts of the input query are not covered by the pre-built models. In that case, the exact model for those partitions has to be built using the relevant data from original dataset. At the end, a set of ML models are passed to the *"approximate model builder"* component.

Approximate model builder In the coreset-based approach, the approximate model is built by training the intended ML model using the *union* of the coresets as training data. In model merging approach, for each ML algorithm the parameters of partial models are combined through a principled manner such that the objective function of the approximate model is very close to the objective function of the model built from the scratch. The reuse strategy for each algorithm is briefly explained as follows.

For K-means, given a set of K centroids and their corresponding weights, the weighted variant of K-means++ algorithm is applied to the union of the given centroids and the final K cluster centroids are returned as the output. For GMM, the mean vectors and covariance matrices of previously built GMM models on different partitions of the data are combined by running a hierarchical clustering algorithm and iteratively merging two closest Gaussian components till only K of them are left. We used Bhattacharyya distance to measure the distance between two Gaussian components. For supervised learning, the parameters of the approximate model are calculated by averaging the corresponding parameters of the pre-built models. For Logistic Regression, model parameter corresponds to the coefficients while for linear SVM, it corresponds to the coefficients of the separating hyperplane. Please refer to [1] for further details.

2.2 Pre-processing phase

In the pre-processing phase, a set of models are built and stored in the pre-built model repository. These models are selected to be reused for the future queries in the best way using a workload or analytic query logs from the past. To identify the best models to materialize, first, the list of possible ML models to build for a given workload history is enumerated. In the next step a greedy algorithm is applied to identify the models with highest benefit for the given workload. These selected models are materialized and stored in the pre-built model repository.

Repository of ML models In the pre-processing phase, exact ML models are built for several partitions of data and their corresponding metadata is stored in a repository. These pre-built models may contribute to future approximate models. For model merging approach the model parameters are materialized while for coreset-based approach the coresets and their corresponding weights are recorded. In the model merging scenario ,for K-means ApproxML stores K centroids and the weight associated with each cluster. In GMM, it stores the mean vector and the covariance matrix of each component along with their relative weights. For Logistic Regression it stores the coefficients and for SVM it stores the coefficients of the separating hyperplane.

2.3 User Interface

The user interface of ApproxML consists of three main sections including configuration panel, results section, and building partial models. Each section is described in detail as follows.

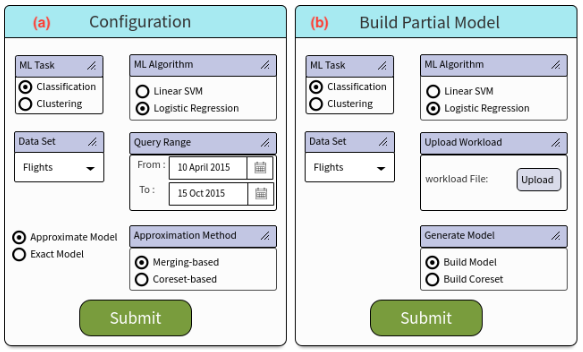
2.3.1 *Configuration section*

In this section the user can submit an analytic query and customize the following options for approximate ML model.
ML task: The user can choose if she wants to build a classifier or a clustering model. If she chooses clustering option, she can specify the number of clusters as well.
ML algorithm: The user can choose between Logistic Regression and Linear SVM for classification task and K-means and GMM for clustering task.
Dataset: The user will select a dataset in this section. An overview of the dataset will be shown to the user. For any selected dataset, appropriate query range or OLAP hierarchy options for customizing the query becomes available. For example, for *Flights* dataset the user can customize the analytic query range by specifying the *FROM* and *TO* parameters as shown in Figure 2(a).
Approximate/Exact: The user has the option to select between the approximate and exact models. If she chooses the exact model, the entire data for the given analytic query will be retrieved from the selected dataset, and the exact model will be built on the entire data from scratch. If the approximate option is selected, the user can then choose between model merging and coreset-based methods.
Model merging/Coreset-based: Based on the user's input in this section, the approximate model will be built using either model merging or the coreset-based methods. If coreset-based method is chosen, an approximation ratio should also be selected.

Figure 2(a) illustrates the configuration panel for building a Logistic Regression classifier on Flights dataset using the data from 10 April 2015 to 15 October 2015 through a model merging approach. Figure 3(a)shows the configuration for building a K-means clustering model on the FIFA2019 dataset using coreset-based approach with approximation ratio of 10% for the data of Europe.

2.3.2 *Results Section*

In the results section of ApproxML, the statistics and quantitative measures of the generated ML model is reported to the user. For classifiers, training accuracy and testing accuracy are shown. Cost of building the model is also reported to the user. In clustering scenario, Adjusted Rand Index (ARI) and likelihood are shown for K-means



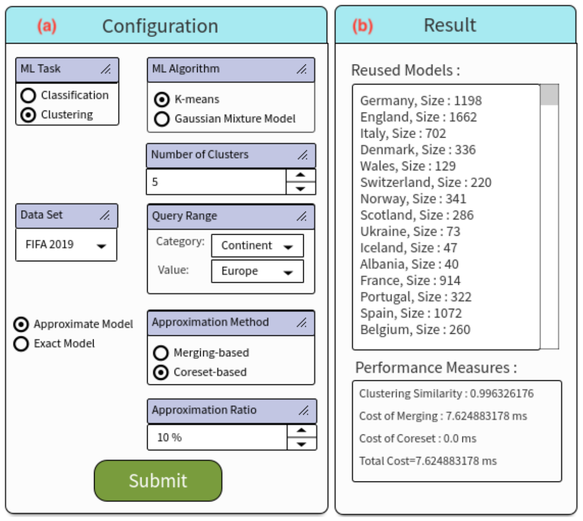
(a) Configuration

ML Task: [✓ Classification] Clustering
ML Algorithm: Linear SVM [✓ Logistic Regression]
Data Set: Flights
Query Range: From : 10 April 2015 To : 15 Oct 2015
[✓ Approximate Model] Exact Model
Approximation Method: [✓ Merging-based] Coreset-based

(b) Build Partial Model

ML Task: [✓ Classification] Clustering
ML Algorithm: Linear SVM [✓ Logistic Regression]
Data Set: Flights
Upload Workload: workload File:
Generate Model: [✓ Build Model] Build Coreset

Figure 2: (a)Configuration panel for approximate Logistic Regression,(b) Build partial Logistic Regression models



(a) Configuration

ML Task: Classification [✓ Clustering]
ML Algorithm: [✓ K-means] Gaussian Mixture Model
Number of Clusters: 5
Data Set: FIFA 2019
Query Range: Category: Continent Value: Europe
[✓ Approximate Model] Exact Model
Approximation Method: Merging-based [✓ Coreset-based]
Approximation Ratio: 10 %

(b) Result

Reused Models :

Germany, Size : 1198
England, Size : 1662
Italy, Size : 702
Denmark, Size : 336
Wales, Size : 129
Switzerland, Size : 220
Norway, Size : 341
Scotland, Size : 286
Ukraine, Size : 73
Iceland, Size : 47
Albania, Size : 40
France, Size : 914
Portugal, Size : 322
Spain, Size : 1072
Belgium, Size : 260

Performance Measures :

Clustering Similarity : 0.996326176
Cost of Merging : 7.624883178 ms
Cost of Coreset : 0.0 ms
Total Cost=7.624883178 ms

Figure 3: (a)Configuration panel for approximate K-means,(b)Results

and GMM respectively. If an approximate model is built, in addition to the total cost of building the approximate model, the partial costs including the merging cost, cost of building coresets, and costs of building the partial models for the new partitions are also reported. Finally, the user can see which pre-built models were retrieved from the pre-built model repository and reused for this particular approximate ML model. Figure 3(b) shows an example of results section for an approximate K-means model with 5 clusters using coreset-based approach.

2.3.3 *Build Partial Models*

In this section, the user can upload a workload, select a dataset, and customize the parameters of an ML model. The data is then retrieved from the dataset, partitioned into optimum partitions, the exact model is built for each partition, and the corresponding metadata for the models are saved in the ML model repository. Figure 2(b) shows an example of this section.

2.4 System Implementation

ApproxML's backend is implemented in Python 3.6. Scikit-Learn (version 0.19.1) was used to train the ML models. Pandas library was used to save the query results in dataframes.

We used Flask for session management and database connection tools.

Datasets: For classification, we selected 5 datasets from the Hamlet repository [2] including Movies, Yelp, Walmart, Books and Flights. In addition, we used Flights dataset [1]. For evaluating clustering algorithms we used Santander customer transaction data [2] and FIFA 2019 dataset [3]. Additionally, we generated a synthetic dataset with 5M data points, 20 features and 10 clusters using publicly available generator [3].

3. DEMONSTRATION PLAN

3.1 System Setup and Audience Interactions

We shall provide 3 laptops with ApproxML pre-installed on them. The datasets and repository of the pre-built models are stored on a server. We will also keep local copies of the datasets and the ML repository on the demo laptops in case of a broken internet. We will store a set of pre-built models in the ML model repository for each dataset. Visitors to the demo can freely select the dataset and ML algorithm of interest, specify the query of interest, and then observe the accuracy and efficiency of the output approximate models and compare them with the exact model built from the scratch. Additionally, they can experience the pre-processing phase by materializing ML models for a given workload.

3.2 Demonstration Scenarios

In this section, we describe several scenarios about how the audience can interact with ApproxML.

A: Classification using Flights 2015 data In the configuration panel, the user can choose classification for ML task and Logistic Regression as ML algorithm. After selecting the Flights dataset and adjusting query range, the user can once build the approximate model with model merging and once with coreset-based approach. The pre-built models that are reused for the approximate model will be reported to the user along with different costs involved in making the approximate models. She can then choose to build the exact model from scratch for the same configuration and compare its efficiency and accuracy with those of the approximate models. The same process can be repeated for generating approximate linear SVM model.

B: Clustering using FIFA 2019 data In the configuration panel, the user can select clustering as the ML task, K-means as the ML algorithm, and set the number of clusters. After selecting the FIFA 2019 dataset, and setting "category" to "continent" and "value" to "Europe", she can build an approximate model once with model merging and once with coreset-based approach. To compare the efficiency and accuracy of the approximate models, she can choose to build an exact model with the same configuration and evaluate the Adjusted Rand Index for goodness of clustering, cost of building the model for the exact model and various costs involved in the approximate models. She can repeat the same steps for GMM as well. In order to compare the approximate and exact GMM models, the likelihood measure is shown for clustering similarity.

C: Materialize models for Flights 2015 data In the "Build partial model" panel, user can choose the ML model as clustering, the ML algorithm as K-means, select number of clusters, and choose Flights 2015 dataset. We will provide a text file containing a workload that the user can upload to the system. The user can submit the request once for model merging approach and once for coreset-based approach with approximation ratio set to 10%. The user will see the optimum partitions identified by ApproxML for materialization. We will also show the saved centroids and coresets and their corresponding weights in the pre-built model repository. The same process can be repeated for Logistic Regression, linear SVM, and GMM.

4. SUMMARY

We demonstrate ApproxML, a system that efficiently constructs approximate ML models for new queries from previously constructed ML models by leveraging the concepts of *model materialization* and *reuse*. In order to generate approximate ML models, ApproxML takes a two-phase approach. In the pre-processing phase it partitions the data and builds exact ML models on each partition and saves their meta data in a pre-built model repository. During the run-time phase, it reuses the pre-built models and combines them efficiently to create an approximate model for a new analytic query.

5. ACKNOWLEDGEMENT

The work of Gautam Das was supported in part by grant W911NF-15-1-0020 from the Army Research Office, grant 1745925 from the National Science Foundation, and a grant from AT&T.

6. REFERENCES

[1] S. Hasani, S. Thirumuruganathan, A. Asudeh, N. Koudas, and G. Das. Efficient construction of approximate ad-hoc ml models through materialization and reuse. *PVLDB*, 11(11):1468–1481, 2018.

[2] A. Kumar, J. Naughton, J. M. Patel, and X. Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *SIGMOD*, pages 19–34. ACM, 2016.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 12(Oct):2825–2830, 2011.

[4] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. Modeldb: A system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 14:1–14:3, New York, NY, USA, 2016. ACM.

[5] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar. Accelerating the machine learning lifecycle with mlflow. In *IEEE Data Engineering Bulletin, 41(4)*, 2018.

[1]https://www.kaggle.com/miquar/explore-flights-csv-airports-csv-airlines-csv/data

[2]www.https://www.kaggle.com/c/santander-customer-transaction-prediction/data

[3] https://www.kaggle.com/karangadiya/fifa19