2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

Providing Cooperative Data Analytics for Real Applications Using Machine Learning

Arun Iyengar, Jayant Kalagnanam, Dhaval Patel, Chandra Reddy, and Shrey Shrivastava
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA
Email: {aruni, jayant, pateldha, creddy, shrey}@us.ibm.com

***Abstract*—This paper presents a data analytics system which determines optimal analytics algorithms by selectively testing a wide range of different algorithms and optimizing parameters using Transformer-Estimator Graphs. Our system is applicable to situations in which multiple clients need to perform calculations on the same data sets. Our system allows clients to cooperate in performing analytics calculations by sharing results and avoiding redundant calculations. Computations may be distributed across multiple nodes, including both client and server nodes. We provide multiple options for dealing with changes to data sets depending upon the data consistency requirements of applications. Another key contribution of our work is the Transformer-Estimator Graph, a system for specifying a wide variety of options to use for machine learning modeling and prediction. We show how Transformer-Estimator Graphs can be used for analyzing time series data. A key feature that we provide for making our system easy to use is solution templates which are customized to problems in specific domains.**

***Index Terms*—artificial intelligence, data analytics, machine learning**

I. Introduction

Many fields have started making extensive use of enhanced data analytics and machine learning techniques that have become prevalent in recent years. Science, health care, business, finance, and many other fields have benefited enormously from increasing usage of powerful data analytics systems which use machine learning and artificial intelligence. In order for applications to take advantage of advanced data analytics systems, the applications must produce a sufficient amount of quality data data for the analytics to yield useful results.

Several frameworks have been developed for building data analytics and machine learning applications including scikit-learn [1], TensorFlow [2], PyTorch [3], Caffe [4], Keras [5], and MXNet [6]. Distributed processing frameworks such as Apache Spark [7] and Hadoop [8] can also be used to scale up data analytics calculations. These machine learning frameworks are freely available as open source software and can readily be used by software developers and data scientists using programming languages such as Python.

In addition to these machine learning frameworks, there are Web services offered by IBM [9], Microsoft [10], Amazon [11], Google [12], and others which provide machine learning and artificial intelligence capabilities such as natural language understanding, speech recognition, and visual analysis. These AI Web services are accessed via http. While some of them are offered for free, getting premium service typically requires paying money.

While these open source machine learning frameworks and Web services make it feasible to build advanced AI applications, there still is often a considerable amount of work that needs to be done to be done to build substantial machine learning and data analytics applications. The data needs to be managed in an appropriate way. Getting a sufficient amount of data can be a problem. Furthermore, the data may need to be preprocessed in order to get meaningful results. Data which constitute erroneous and/or outlying values may need to be identified and discarded. Missing data may need to be imputed by an appropriate method. The data may be constantly changing. When this is the case, appropriate methods are needed to determine how frequently and when to perform updated analytics calculations. Data privacy issues can also be an important factor.

In this paper, we present an end-to-end data analytics system which provides a broad range of machine learning algorithms. Our system can test wide variety of different machine learning algorithms to identify the best ones. It can also handle data which is distributed across multiple nodes and may be changing.

Another key aspect of our system is the *Transformer-Estimator Graph* which provides users with a methodology for specifying multiple possibilities for end-to-end machine learning computations using graphs. Our system can pick the best performing path (s) in the graph, while optimizing parameters.

Transformer-Estimator Graphs are inspired by common methodologies data scientists have started using while solving AI modeling problems. Given a data set and type of modeling activity (such as classification, regression, clustering, etc) to be performed, data scientists may follow a series of steps such as feature scaling, feature transformation, feature selection, model learning, etc to find the best approach for the data sets. The machine learning community refers to such a series of steps as a "Pipeline". Note that for each step in a Pipeline, there are multiple options (i.e., methods/models) available for trial. For example, the feature scaling step can be done using a standard scaler, 0-1 normalization, an outlier-aware robust scaler, etc. Similarly, there are multiple machine learning algorithms available for building supervised classification models. It is often not possible to know in advance what options will

2575-8411/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDCS.2019.00186

work well for a given data set. Thus, an efficient tool like ours that allows data scientists to easily explore multiple options is needed.

This paper makes the following contributions:

- We present a distributed system for analyzing data using machine learning and AI techniques which allows multiple clients to cooperate across different nodes. Computations may take place across both client and cloud server nodes. We provide a novel method for the nodes in the system to cooperate in performing machine learning and AI computations so that they are aware of computations performed by other nodes, and redundant computations can be avoided. Our system also provides consistent data storage and allows updates to data while maintaining consistency across different nodes. We provide delta encoding to minimize data transfers.
- We present the Transformer-Estimator graph, a new framework for organizing machine learning and AI computations into multiple stages. Transformer-Estimator Graphs allow different computational options to be specified at different stages in the pipeline. They allow users to determine an optimal set of steps from a wide range of possibilities. Transformer-Estimator Graphs provide a powerful new tool to optimize an end-to-end data analytics task.
- We show how graphs can be configured for analyzing time series data based on real data analytics problems from heavy industry. In order to make our system usable by non-experts, we have developed solution templates which are targeted to specific problem domains and are considerably easier to use than general-purpose machine learning frameworks such as scikit-learn or TensorFlow. We describe some of the solution templates which are useful for data analytics problems in heavy industry.
- We present an overview of the research and technical challenges for providing usable systems for data analytics using machine learning. We also discuss some of the real challenges we have faced in analyzing data from real customers.

The remainder of this paper is structured as follows. Section II discusses research and technical challenges in developing machine learning frameworks which are readily usable by nonexperts as well as some of the real difficulties encountered when analyzing real data. Section III describes the overall architecture of our system. Section IV describes Transformer-Estimator Graphs. Section V presents related work. Finally, Section VI concludes the paper.

II. Research and Technical Challenges

We are concerned with providing usable data analytics systems that people who are not experts in data science and software engineering can use. There are several existing machine learning frameworks which are widely used, as mentioned earlier. A key problem is that while these frameworks are certainly useful to data scientists and software engineers with enough knowledge of the field they are analyzing data for, there are many others with interest in analyzing data who simply do not have the required knowledge and skills to properly use these machine learning frameworks. There is thus a need for machine learning frameworks which are easier to use.

It should be noted that it is extremely difficult to provide a machine learning framework which is both readily consumable by non-experts and provides the range of features of a general framework such as scikit-learn. Thus, in order to make a framework or tool easier to use, it may be necessary to restrict it to solving a narrower range of problems and use cases. We have taken this approach in designing *AI solution templates* which are customized to solving data analytics problems faced by customers in heavy industry (Section IV-E).

The range of machine learning algorithms which can be applied can be significant. This may necessitate trying out multiple different algorithms. Furthermore, machine learning algorithms typically have parameter settings which can be varied. Methods are needed both to optimize parameters and systematically test several algorithms on the data sets. We have developed the *Transformer-Estimator Graph,* a powerful and versatile tool which allows users to easily test out a wide range of data analytics algorithms and parameter settings (Section IV).

Computational overhead can be a major issue. Analysis of large data sets can have high overhead. Deep learning can require significant computing resources which is a key reason that GPUs are often used. Even if a data set is not inordinately large and no one particular computationally intensive analysis approach is being used, computational overhead can arise from running several different algorithms, each with many different parameter settings. It is often possible to run multiple analytics tasks in parallel to reduce completion times.

Another key issue is that the data analytics platform itself may be distributed across several nodes. This is discussed further in Section III. Crucial data may reside on nodes which do not have much computational power. It may be necessary to perform analytics computations on a node without a high degree of processing power as communication with a significantly more powerful computer would incur latency and may not be possible if connectivity is poor. How to optimize computational resources in such a distributed system is a major challenge. Maintaining data consistency across different nodes can also be a problem. In Section III, we discuss these issues further and present an approach for allowing several distributed clients to cooperate and avoid duplicating the same analytics calculations. Our system also provides data consistency.

There are several practical considerations in analyzing real data sets that need to be taken into consideration. Simply applying the latest machine learning or AI algorithms to the data at hand will often not give very meaningful results.

Apart from choosing a model based on its accuracy (on a test set or cross-validation error), one must consider whether the model is *interpretable*: (1) can it be described using simple rules? (2) can it provide sensitivity analysis—i.e., how much contribution a factor is making to the predicted value, or how

does it compare to another factor in terms of importance? For example, some ensemble methods and neural networks fall short on this count. A key point is that simply making a data prediction is not sufficient. Explanations behind the prediction and sensitivity analysis may be just as important.

Another consideration is whether the model includes *actionable* factors. When it does, does the model allow *root-cause analysis* (what factors contributed to the outcome) , *intervention* (what factors, and by how much, should I change to get a desired outcome) and *what-if analysis* (what would have happened if this factor were not effective).

The nature and the availability of the data impose some considerations. There may only be *limited training data* which makes it difficult to draw definitive conclusions. The data may be incomplete, in which case data imputation may be needed. The data may be inaccurate. In some cases, the data may not be representative for the modeling problem at hand. For example, human data samples might be biased towards certain subgroups and may not be representative of the general population as a whole. Sometimes there are *class imbalances* — e.g., rare failure cases, but many successful cases. There is the issue of *censored data*. Obtaining labeled data for supervised learning may be costly or error prone. Apart from these, *data cleansing* issues can be significant.

In some domains, a lot of domain knowledge may be available without doing sophisticated data analysis and machine learning. This may help offset the issue of lack of data and reduce training times. How to integrate this domain knowledge with data analytics and machine learning is a key issue. Modeling should effectively *utilize available domain knowledge*. Yet another practical issue is convincing the human domain experts. If the model corroborates their thinking, they may under-appreciate the value of the model. On the other hand, if the model results are far or novel from the way they think, they may dismiss the model as being too esoteric. It may require the modeler to work with the experts to convince them.

Yet another consideration is regarding *managing model life-cycles* in which data analytics and machine learning are performed over a long period of time. Availability of more data may require the model to be retrained or even changed. The frequency of retraining (or changing) models needs to be properly selected. Too frequent retraining can result in high overhead, while too infrequent retraining can result in obsolete models which are less accurate. There may be concept drifts. Equipment or environments may change. Some new metrics may become available, and some previous metrics may become unavailable. This means that the data which are available can change over time, as can the both the computational requirements and computational resources for analyzing data.

III. System Architecture

Figure 1 depicts the overall architecture of our system which allows analytics calculations to take place from several client nodes. This is an important computational paradigm when

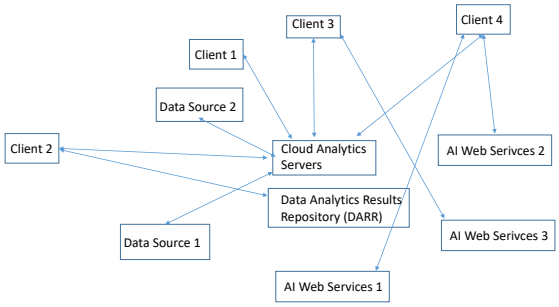

Fig. 1. Our distributed data analytics system.

multiple clients who are geographically distributed need to access a common data set. The data may be replicated across multiple geographic areas for high availability and disaster recovery in case one site fails.

The data analytics calculations can be performed on one or more cloud virtual machines remote from the client. These virtual machines reside in the cloud analytics servers depicted in Figure 1. In this case, the cloud virtual machines can be scaled as needed to handle the computations. Users need to generate multiple predictive models to determine which ones work best. Different predictive models can be run in parallel. The same predictive models may also need to be run with multiple parameter sets to optimize the parameter settings. These parameter optimizations can be done via parallel invocations.

Our system is distributed, so the client nodes in Figure 1 can perform data analytics calculations remotely from the cloud analytics servers. That can reduce the latency since the client will not have to communicate with remote cloud nodes. It also allows the client to develop individualized analytics algorithms which do not have to be implemented on the cloud nodes. It also allows the client to perform analytics calculations when it does not have connectivity with the cloud.

Figure 1 depicts multiple AI Web services. These correspond to AI services offered over the Web such as IBM Watson, Microsoft Azure Cognitive Services, Amazon Machine Learning on AWS, and Google Cloud AI products. These Web services complement the machine learning capabilities at the clients and cloud analytics servers. Clients benefit by having the full range of analytics capabilities from multiple parties depicted in the figure.

Our data prediction system supports distributed computations across multiple nodes. In the computational model, there is a main database. That database might be in a central location. Alternatively, the database might be distributed across multiple nodes. Figure 1 depicts multiple data sources providing input to the analytics calculations, and data can reside on any of the nodes. A key aspect is that the data may be changing. As the data changes, analytics calculations may need to be executed over the updated data. Therefore, our data analytics system is integrated with the data management system. The data are monitored for changes. When the amount

of change in the data exceeds a threshold, then analytics calculations are recalculated on the data. There are a number of ways to determine if data has changed enough to warrant updated analytics calculations:

- The number of updates since the last time analytics calculations were run exceeds a threshold.
- The total size of updates since the last time analytics calculations were run exceeds a threshold.
- Application-specific methods can be applied to determine how much the data have changed. This is the best way to determine when to perform updated analytics calculations. However, it is harder to implement this option than the previous ones.

The computational overhead for data analytics calculations is also an important factor that should be considered in making decisions to perform analytics calculations. If the computational overhead is low, it becomes more feasible to perform analytics calculations more frequently, and vice versa.

Integration of the data store with the data analytics software is a key area of future research. Existing machine learning frameworks such as scikit-learn, TensorFlow, and PyTorch focus on providing machine learning algorithms but not on providing an efficient data tier. Future research in general-purpose machine learning frameworks should look at better ways of integrating data management. This would allow analytics calculations to take place at exactly the right times in response to data updates. It would also allow data to be processed most efficiently. In a distributed computing environment, obtaining data remotely can entail high overhead. A better solution is to have data close to the computational nodes. This can be accomplished using appropriate choices for data stores as well as using techniques such as caching.

There are a wide variety of machine learning frameworks and Web services that are available. Open source frameworks such as scikit-learn, TensorFlow, and PyTorch provide powerful machine learning capabilities which are constantly being updated and improved as the technical community advances. Several companies offer machine learning via Web services. It is important for data scientists to be aware of the latest tools and techniques so that they can properly take advantage of them. The plethora of existing analytics tools means that data scientists can often use existing software for their core algorithms and not have to implement substantially new ones. However, building analytics platforms for real systems is still often a substantial effort due to the fact that the analytics platforms need to be customized to the applications at hand. Getting the data in the right format so that it can be properly analyzed often involves substantial work. The data has to be obtained from the right data sources and preprocessed to remove erroneous data and outliers. There can be missing data, and data imputation techniques for dealing with the missing data are crucially important. The appropriate transformations to make the data most amenable for analysis can be substantial.

A key feature of our system is its mechanisms for keeping data at the various nodes consistently updated. The update rate can be high. It may not be feasible to send updates to clients every time they are available due to the overhead this entails. Data are comprised of *objects.* An object has a version number associated with it. Each time an object is updated, its version number increases. Each data object has an associated home data store which contains the current version of an object and its version number. The home data store can send complete versions of an object $o1$ to other nodes. Alternatively, it uses delta encoding to send deltas between a previous version of an object and the latest version. Delta encoding can significantly reduce the overhead for updating objects. The key idea is for the home data source to send the delta between the latest version of $o1$ and a previous version of $o1$. $d(o1, 2, 3)$ represents a delta between version 2 and version 3 of object $o1$. This delta may be considerably smaller than version 3 of $o1$. If this is the case, then sending $d(o1, 2, 3)$ to a node which already has version 2 of $o1$ will save considerable bandwidth over sending the entire copy of $o1$.

Suppose the latest version of $o1$ is k. The home data store maintains recent versions of $o1$ as well as deltas between the latest version of $o1$ and these recent versions, $d(o1, k-1, k)$, $d(o1, k-2, k)$, $d(o1, k-3, k)\ldots$ When a remote node $n1$ requests the latest version of $o1$ from $o1$'s home data store and $n1$ has an earlier version e of $o1$, $n1$ passes the version number, e, to the home data store. If the home data store has a delta between version k and version e of $o1$ and that delta is considerably smaller than version k of $o1$, the home data store passes the delta to $n1$. Otherwise, the home data store passes version k (i.e. the latest version) of $o1$ to $n1$.

Clients can obtain updated data from home data stores using either a pull or a push paradigm. In a pull paradigm, the client is responsible for querying a home data store when it wants to obtain updated data. In a push paradigm, clients can subscribe to updates for data objects from home data stores for a specified period of times. Such subscriptions have also been referred to as leases in the literature [13]. After a lease expires, the client must contact the home data store to renew the lease to continue receiving update messages. A client is also expected to cancel its leases early for data for which it no longer needs to automatically receive information about updates.

In the push paradigm, after $o1$ is updated, the home data stores can send either the entire current value of $o1$ or a delta between a previous version and the current one. However, either of these approaches may be inefficient if the client does not need the updated data immediately. Another approach is for the home data store to send information about the update to the client, such as the new version number and how much the new version differs from the previous one. The client can then decide if and when it needs to obtain the latest version from the home data store.

Our system allows multiple clients to cooperate on performing data analytics calculations on common data sets. That way, the clients can share the results with each other and not have to repeat calculations. Clients can place their data analytics results, along with an explanation of how the results were achieved, in a data analytics results repository (DARR) in the

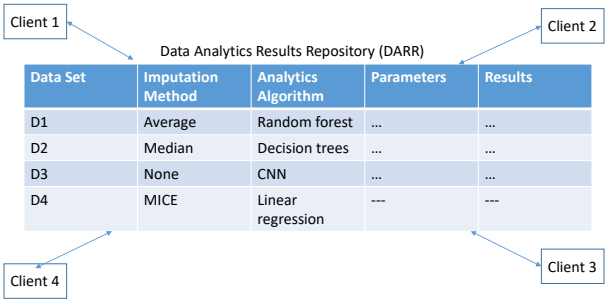

Fig. 2. Clients can share analytics results in the data analytics results repository (DARR).

cloud. The DARR can be accessed and written to by multiple clients, allowing them to both store and retrieve analytics information (Figure 2).

In some cases, clients will be performing analytics using well-defined methods. For example, the steps could include cleaning the data and removing outliers using one or more of a fixed set of techniques and performing data imputation using one or more of a fixed set of techniques (e.g. mean, median, mode, multiple imputation by chained equations, matrix factorization, k nearest neighbors, etc.).

Machine learning algorithms can use a variety of techniques for training models, such as linear regression, neural networks, random forest, decision trees, k nearest neighbors, gradient boosting, and others. Various methods can be used to determine how well a model fits the training data. For example, we can perform k-fold cross validation on the training data and use metrics such as mean absolute error, mean squared error, median absolute log error, mean squared log error, root mean squared error, root mean squared log error, etc.

Once a model has been trained, it has to be tested on data. Various methods can be used to determine how well a model predicts a test set, such as mean squared error, coefficient of determination (R^2), mean absolute error, root mean squared error, etc.

Our system implements a pre-defined set of methods for various steps in data analytics, including data cleansing, outlier detection, data imputation, model training, and model testing. Users can specify the options that they want for each step, as well as the input parameters and output results to collect. The system will then run the appropriate data analytics calculations and optionally store the results in the data analytics results repository (DARR). When these types of structured calculations are run, the DARR can keep track of all analytics calculations that have been run for a particular data set. The total number of possible calculations for a data set is generally too large to exhaustively determine. This is particularly true given the large number of parameter settings.

Therefore, the DARR provides a wide variety of data analytics results which can be used by multiple clients. Users can determine from the DARR which calculations have been run for a certain data set. Clients can then use previous results stored in the DARR. They can also perform additional calculations which do not overlap with those already stored in the DARR. Using this approach, different clients can cooperate to solve data analytics problems.

IV. Transformer-Estimator Graph

Typically during the exploratory stages of a machine learning problem, data scientists iterate through dozens of features and model combinations. In Table I, we have highlighted a typical thinking process a data scientist might have for a regression task.

Steps	Component (Transformer or Estimator)
Select Features	Select K-Best Information Gain Entropy
Feature Normalization	Min-Max Normalization Standard Scalar
Feature Transformation	PCA kernal-PCA LDA
Model Training	Random Forest CNN Linear Regression
Model Evaluation	k-fold cross-validation monte-carlo simulation
Model Score	Root mean square error Mean Average Percentage Error

TABLE I
Different Steps in Machine Learning Modeling

Several open source frameworks for machine learning have been developed which ensure that model training and testing go through a similar process. Scikit and SparkML are two popular libraries for machine learning model building. These open source development frameworks have designed their libraries based on a "Pipeline" construct. A pipeline construct gives a declarative interface where it is easy to see the entire data extraction, transformation, and model training workflow. In particular, a (spark or scikit) Pipeline is constructed that specifies a sequence of stages, and each stage is either a Transformer or an Estimator. These stages are run in order, and the input data (mostly tabular) is transformed as it passes through each stage. The pipeline, transformer, and estimator are three important constructs in machine learning modeling and are defined subsequently in this paper.

In a co-operative data analytic framework, multiple data scientists try different pipelines on the same dataset, and it is highly possible that some pipeline stages are common across different pipelines. Thus, it is important to define a new construct that can effectively summarize the pipelines designed by different data scientists. In this paper, we describe a new programming model which can be facilitated to capture an end-to-end machine learning pipeline for collaborative exploration. Our programming model provides an API to define a Transformer-Estimator Graph that can be implemented across one or more of the nodes depicted in Figure 1. A Transformer-

Estimator Graph has the capability to specify multiple options at each stage.

Figure 3 provides a graphical representation of a Transformer-Estimator Graph created for a regression task. In this regression task, the user is setting multiple operations in stages: i) feature scaling, ii) feature selection, and iii) regression models. The feature scaling stage contains standard methods: MinMaxScaler, RobustScaler, StandardScaler and an option to exclude the stage (i.e., no operation). Next, feature selection uses co-variance based principal component analysis (PCA), SelectKBest or no operation. The final stage consists of the regression model, where the user wants to try a number of well-known classifiers such as Random Forest, MLP Regression, and Decision Trees. In summary, Transformer-Estimator Graph provides a neat and clean way of managing dozens of feature engineering steps and feature scaling operations crisscrossed with pre-selected hand-tuned models.

A Transformer-Estimator Graph, denoted as $G(V, E)$, is a directed acyclic rooted graph (DAG) with a set V of vertices and a set E of edges. Each vertex $v_i \in V$ in the graph represents a meaningful AI/ML operation to be performed on the in-coming data, and edge $e_i \in E$ in the graph represents data/function flow between vertices. Each vertex v_i in graph G consists of its name and the operation it performs, and is represented by a tuple v_i = $(name_i,\ operation_i)$. For example, tuple $(\text{“}pca\text{”}, PCA())$ represents a node, with name “pca”, that performs principle component analysis. Each node in Figure 3 represents the node name. The name given to each node in the pipeline graph should be unique and is a placeholder that enables users to supply external information (e.g. parameters) that can be used to control/change the node operation.

For example, if users want to try “PCA()” with a different number of components ($n\_components$), they can specify the value using “pca__n_components”. The naming convention “pca__n_components” (node name followed by two underscore “_” sign followed by attribute name) is adopted from sklearn.

The operation performed by a node is one of two types: Transform(_.transform) or Estimate(_.fit). An Estimate operation is typically applied to a collection of data items to produce a trained model. A Transform operation uses a trained model on individual data items or a collection of items to produce a new data item. For example, learning a direction of a principal component is done using an estimate operation, whereas projecting a data point to a new dimension is done using a “transform” operation. Ideally, an estimate operation is performed first. In our above example, tuple $(\text{“}pca\text{”}, PCA)$ is type of Transform operation.

A. Pipeline

A Pipeline is a sequence of adjacent connected graph nodes that starts from root node v_{root} and ends at leaf node v_k. Briefly, a pipeline chains together meaningful machine learning operations to build a machine learning Pipeline. For example, one of the Pipelines in Figure 3 is given as follow:

$$P_1 = \{Input \rightarrow robustscaler \rightarrow Select{-}K \rightarrow DecisionTree\}$$

The total number of Pipelines for our working example given in Figure 3 is 36. In summary, graphs provide a convenient way for end users to specify multiple options to be tried in each stage of the modelling process.

Listing 1 shows an example of constructing a graph for a regression task using Python. There should be multiple such methods to add the different operators. For example, method ‘add_feature_selector’ takes a list of transformers to enable the feature selection. In summary, these methods allow users to configure stages in a graph, where each stage provide a list of options to be tested. In this example, we discussed three stages by calling different methods: feature scaling (‘add_feature_scaler’), feature selection (‘add_feature_selector’), and classification (‘add_regression_model’).

```
def prepare_graph():
  Task = TransformerEstimatorGraph()
  Task.add_feature_scalers([MinMaxScaler(),
       StandardScaler(),RobustScaler(),NoOp()])
  Task.add_feature_selector([[Covariance(),
       PCA()], SelectKBest(), NoOp()])
  Task.add_regression_models([DecisionTree(),
       MLPRegressor(), RandomForest()])
  Task.create_graph()
  return Task
```

Listing 1: A programming construct to generate a regression graph

The feature scaling stage contains three popular methods: MinMaxScaler(), RobustScaler(), StandardScaler() and one NoOp() operation. The NoOp operation allows users to skip the operation in that stage. The second stage is feature selection using PCA(), a SelectKBest() method or a NoOp() operation. The final stage is the regression stage, where users try a number of well-known regression models. Users can add any model compatible with sklearn's pipeline construct. The design is flexible, and users are free to add as many stages and as many options per stage as they prefer.

In the final line of Listing 1, the method “create_graph” generates a graph for visual inspection. The output would be similar to Figure 3.

B. Model Validation and Selection

Given a dataset D and a Transformer-Estimator Graph G, the objective of model validation and selection process is to identify a pipeline from the Transformer-Estimator Graph that performs reasonably well for a given dataset. Basically, each pipeline in a Graph is evaluated for a given dataset D, and a path with good model performance is selected. The model selection process requires an agreement across users in terms of the scoring mechanism. The scoring mechanisms are task dependent and calculated using cross validation. For example, Accuracy, Area under the Curve (AUC), and F1-score are

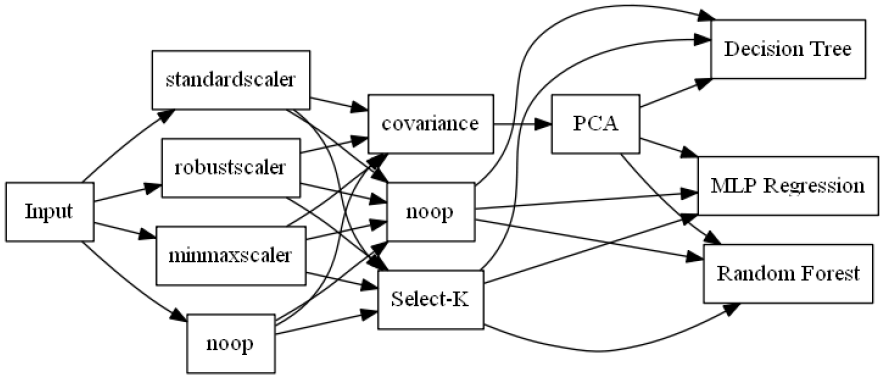

Fig. 3. Example of a Transformer-Estimator Graph for a Regression Task.

commonly used performance measures for classification tasks. Similarly, root mean square error (RMSE), mean square error (MAE), and r^2 are performance measures for regression tasks, etc.

The model performance of a pipeline is calculated using cross validation methodology. Some examples of cross validations include K-fold, Nested K-fold, and Monte-carlo. Figure 4 outlines the process of calculating the performance measure using K-Fold cross validation. A K-fold cross validation strategy is a widely used cross validation technique for independent and identically distributed (iid) data. In K-fold cross validation, input dataset D is randomly partitioned into K equally sized folds without replacement. Next, the data from $K-1$ folds are used to train a given pipeline, and data from the remaining (single) fold is used to obtain predictions using trained paths. The process is repeated for all the other folds one by one, and average performance is computed as a final performance measure. In summary, we obtain K models and K performance estimates. Then, we take their average as the final performance estimate. We can apply K-fold cross validation to either the hyperparameter tuning, performance reporting, or both. Some alternate cross validation strategies are Train-Test Split, Time Series Split, etc.

Fig. 4. Example of K-Fold Cross-validation based evaluation

Note that the cross-validation strategy needs each pipeline to support two important operations, namely training and prediction. For the working example given in Figure 3, the total number of Pipelines for evaluation, using a K-Fold cross-validation strategy, is now K times higher.

1) Training and Prediction of Pipeline: Figure 5 visually explains training (pipeline.fit) and prediction (pipeline.predict) operations performed on sample pipeline: $P_i = \{start \rightarrow robustscaler \rightarrow Select-k \rightarrow MLPRegressor\}$. The training operation should precede the prediction operation.

As shown in Figure 5, the training operation takes input dataset (X) and class label information (Y). Next, input data (X,Y) is passed through the pipeline nodes in turn. In the training operation, the internal nodes of the pipeline path perform "fit & transform" operations, whereas the last node performs "fit" operations. The "fit & transform" operation on internal nodes refresh the data for subsequent modeling. The end result of the training operation is a trained machine learning model.

The predict operation takes only input dataset (X) and passes dataset X through a trained model to generate the prediction (class label, real value, etc). Input data X is also passed through the pipeline nodes in turn. For this operation, the internal nodes of the pipeline path perform "transform" operations to generate the prediction.

2) API for Pipeline Graph Evaluation.: Listing 2 shows an example of a method implementing pipeline graph evaluation written in Python. Methods 'set_cross_validation' and 'set_accuracy' are used to adjust the parameters based on the task at hand. In this example, we are interested in 10 fold cross-validation and f1-score as an accuracy measure.

```
def pipeline_evaluation(Task,D):
    Task.set_cross_validation(k=10)
    Task.set_accuracy('f1-score')
    Execute Task
    return model, best_score, best_path
```

Listing 2: Pipeline Evaluation

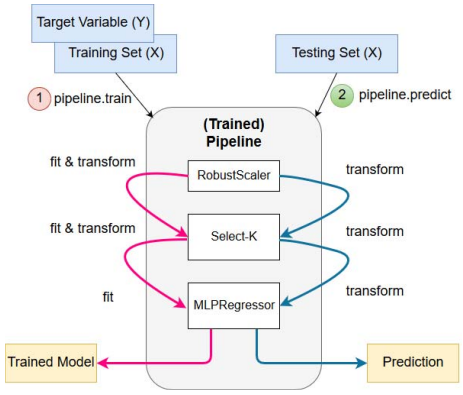

Fig. 5. Training and prediction operation on a sample pipeline

C. AI Functions for Transformer-Estimator graphs

Previously, we discussed traditional sklearn-based transformers and estimator components that are mostly designed for iid data. In this section, we introduce an advanced version that can deal with complex data such as Time Series Data. We call the advanced version of these components as an **AI function**. To explain details of AI functions, we choose one of our main widely used pipelines, the **Time Series Prediction pipeline,** to show how it benefits from the Transformer-Estimator graphs. This pipeline has major applications in the industrial domain and is a good example of the utility of Transformer-Estimator graphs.

An AI function involves a set of activities which are associated with training and validating a time series model. From the name, it is suggested that the transformer and estimators are prime examples of such AI functions. Transformers are AI functions that can handle the scaling and preprocessing for a dataset satisfying individual model requirements. After the transformation, the modelling is done by the Estimators, and the top performing model is selected from this stage. For each AI function, there is a set of parameters defined, and the performance of a path in the graph is evaluated using time series. These AI functions can be connected to form large graphs with an API for Transformer-Estimator Graph evaluation.

In the next section, we shall define the the components of the graph in the context of Time Series Prediction.

We summarize various state of the art time series prediction models from the literature. For time series prediction models, we mainly observed three types of estimators: standard deep neural networks, temporal deep neural networks, and the traditional statistical models.

1) Statistical Models.: The statistical models include algorithms like Zero model, which acts as the baseline model for our prediction problem. This model basically outputs the previous timestamp's ground truth at the next timestamp's prediction. Another model in this category is ARIMA. We did not use this model due to complexity in adding the time series prediction pipeline.

2) Temporal Models.: Temporal deep neural networks include Long Short Term Memory (LSTM) networks and traditional Convolutional Neural Networks (CNN) and its variants WaveNet and SeriesNet.

Long Short-Term Memory (LSTM). Long short-term memory (LSTM) networks have gained popularity in recent years due to their ability to model the sequence in the given data. They provide a scalable and effective way to train as they have recurrent units that are good at handling exploding and vanishing gradients [14]. For our use case, we have built two different architectures for the LSTM models based on the complexity. Both these architectures are generic enough to encompass a wide variety of time series prediction tasks in industrial datasets and model them effectively. The two architectures used here follow a similar structure - repetition of a LSTM layer followed by a dropout layer. The first model is a simple architecture which just has one LSTM layer followed by a dropout layer, whereas the other model is more complex as it has four LSTM layers, each followed by their own dropout layers. Both these architectures have a fully connected linear activation layer at the end.

Convolutional neural network (CNN). In recent years, convolutional neural networks have proven to perform well on tasks like image classification [15]. They are effective at capturing internal patterns in the data, and hence they can also be applied to time series data. They provide faster performance when compared to LSTMs and also take temporal characteristics into consideration.

We have implemented a convolutional neural network (CNN) in the time series prediction pipeline. It consists of multiple hidden layers and convolutional layers. It is essentially a feed-forward neural network in which the convolution of the sequence is calculated internally. The max pooling layer helps in reducing the dimension of the input sequence. Similar to LSTMs and standard DNNs, we have made use of simple as well as deep architectures of the Convolutional Neural Networks. Both of these architectures have layers such as a 1D convolutional layer, a max pooling layer, a dense non-linear layer with ReLU activation, and a densely connected linear layer [16]. The deep CNN is a more complex implementation of the simple CNN explained above. Along with the standard CNN, we have also implemented other popular CNN based models described below.

WaveNet. It is a deep learning network which was developed to process speech data. The architecture was built to learn the probabilistic distribution from samples of audio data and generate speech mimicking human speech with high accuracy [17].

SeriesNet. SeriesNet is based on the WaveNet architecture and provides state of the art performance when it comes to time series prediction. It provides similar results to top performing models even without having data pre-processing and ensemble methods [18].

3) IID Models.: Finally, we have also incorporated non-timeseries deep learning models in the pipeline like standard neural networks. These models ignore the temporal char-

acteristics of the data but sometimes provide good models depending on the nature of the data. Each estimator needs special data pre-processing and is discussed subsequently.

Deep Neural Network (DNN): The Deep Neural Networks sometimes provide good performance if the time series data has transactional characteristics. They treat the data as Independent and identically distributed (IID) variables by not considering the sequence in them. The DNNs we are leveraging here have two different architectures. Similar to the case in LSTMs, we have simple and complex architectures based on the number of hidden layers. The simple network is 2 hidden layers and dropout layers, whereas, the complex network is made of 4 hidden layers and dropout layers. One of the advantage standard DNNs offer over LSTMs is their much faster speed of execution compared to LSTMs.

4) Time Series Transformers: Transformers are defined as the AI functions that modify the data before the modelling step using the above mentioned estimators. They are used to perform one of the two activities- improve the data quality so the estimators are able to build better models or modify the data so that estimators are able to ingest the data for modelling. In this section, we provide descriptions of the transformers used in the Time Series Prediction Pipeline. They are either Data Scalers used for normalizing the data or Data Preprocessors used for modelling time series data for estimator consumption.

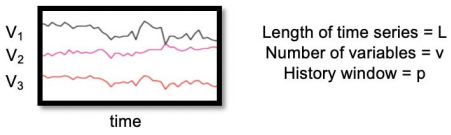

Fig. 6. Multivariate time series data

In time series data, many complications arise in machine learning modelling due to the temporal nature of the data. The underlying assumption that data is independently and identically distributed is no longer valid, requiring careful data manipulation in the pipeline. For typical multivariate time series data as shown in Figure 6, a prediction task is to look at a history of the time series data, usually for a fixed window size called **history window** of length p, and try to predict the value of the next few timestamps, called **prediction window** of a particular variable which has not been observed yet. Since the input to the model here is multivariate time series data (v variables) for some history window (p), the input data becomes 2-dimensional with the shape $(v * p)$. Since estimators have different methods for consuming data, we need to transform this 2D data to comply to these methods. Hence, we make use of Transformers to address the three challenges which arise when building the Transformer-Estimator graphs for time series prediction:

1) Normalization or Standardization of the data.
2) Addressing the data ingesting policies for different estimators
3) Preserving the temporal nature of the data.

Data Scalers: Normalization and Standardization of the time series data are important steps in the model building process. Standardization of data typically involves converting the mean of the time series to 0 and the standard deviation to 1. We leverage scikit-learn's implementation of the 'StandardScaler', 'MinMaxScaler' and 'RobustScaler' for performing data scaling.

Data Preprocessing: We have defined custom time transformers for our pipeline. They address the challenges 2 and 3 in the above list. Descriptions of the these transformers are provided below.

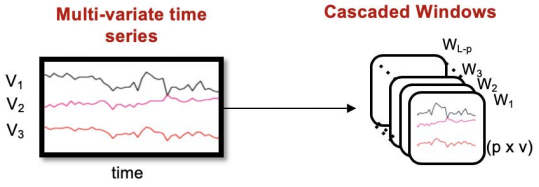

Fig. 7. Time Series Cascaded Windows

Cascaded Windows: In this, the time series data is transformed into a series of cascaded windows as shown in Figure 7. This is used for the Temporal DNN models like LSTMs and CNNs. They contain the temporal history of the data and preserve the order of the time series data.

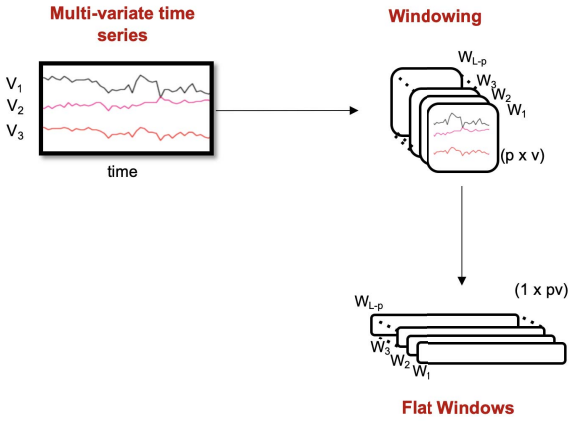

Fig. 8. Flat Time Series Windowing

Flat Windowing: In this, the time series window from the 'Cascaded Windows' step are flattened. For example, if we have built $L - p$ cascaded windows of shape $(p * v)$, after flattening it, we will have $L - p$ windows of shape $(1 * pv)$ as shown in Figure 8. This is done for the standard DNN which

takes transactional data as input. It provides temporal history to the estimator; however, the ordering is lost.

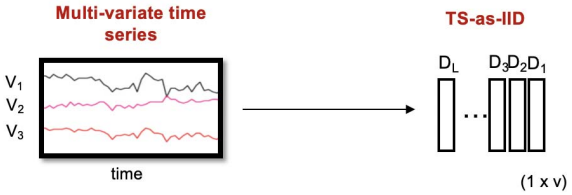

Fig. 9. Time Series as Transactional data

Time Series as transactional data (TS-as-IID): This is also consumed by the standard DNN models but no information about the recent history or temporal order is preserved. Each time stamp is provided to the model as an independently and identically distributed data point as shown in Figure 9.

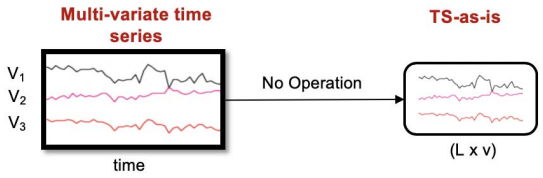

Fig. 10. Time Series with no operation

Time Series with no operation (TS-as-is): In this, the time series is passed to the models which don't require data transformations like Zero model and ARIMA Model as shown in Figure 10.

D. Time Series Prediction Pipeline

The Time Series Prediction pipeline is built on the Transformer-Estimator graphs with the AI functions mentioned above. Using the DAG based graph definition APIs, the multiple AI functions can be connected to form a coherent pipeline for performing time series prediction. This pipeline fulfills a major requirement in the heavy industry domain. Table II provides a high level overview of the various stages of the pipeline along with the AI functions and other components used inside it.

The graph for the Time Series Prediction graph is displayed in Figure 11. The three major layers in the pipeline are:

1) The **Data Scaling** stage applies various standardization and normalization transformations on the data, such as Standard scaling and Min-max scaling as described in Section IV-C4.
2) In the **Data Preprocessing** stage, we transform the data based on the model requirements. The three model categories defined - Temporal DNN, Standard DNN and Statistical models have their separate data inputs. The Transformers defined above address these conditions and prepare the data accordingly.
3) In the **Modelling** stage, we finally train the models with the appropriate data inputs from the data preprocessing stage. The model with the top performance is provided by the pipeline as the output.

Steps	Component (Transformer or Estimator)
Data Scaling	Min-Max Scaling Robust Scaling No Scaling Standard Scalar
Data Preorocessing	Cascaded Windowing Flat Windowing TS-as-IID TS-as-is
Model Training	Temporal DNN IID DNN Statistical Models
Model Evaluation	TimeSeriesSlidingSplit
Model Score	Root mean square error Mean Average Percentage Error

TABLE II
DIFFERENT STEPS IN TIME SERIES PREDICTION PIPELINE

The Transformer-Estimator graph enables us to build the pipeline and connect these separate AI functions with ease. The Data Scaling stage allows us to connect with each of the other paths to help us find which data scaling approach yields the best result. The output of the data scaling stage is sent to the Data Preprocessing stage. In this stage, each AI function is selectively connected to the estimators in the next stage. The $CascadedWindows$ is connected to the $TemporalDNNs$, the $FlatWindowing$ and $TS-as-IID$ are connected to $StandardDNNs$ and finally the $TS-as-is$ is connected to $Statistical$ models. This complex connect graph is automated with the help of the Transformer-Estimator graphs and allows us to build a tool for automatic discovery of the best modelling path for a given time series data set.

When it comes to modelling of time series data, we need to address the cross-validation strategies which can be applied. It is more critical to address in time series applications due to the fact that the test data should have not any information from the training data for a model training. To address this, we have used the $TimeSeriesSlidingSplit$ as cross validation strategy. In this cross-validation technique, we use the size of a training and validation set with a buffer window between them as shown in Figure 12. The windows slide across time to include future data in the training and validation sets for k iterations.

This enables us to trust the results of our Time Series Prediction pipeline.

E. Solution Templates for Domain-Specific Data Analytics

In industrial applications, there is a need for solving unique problems with AI and Machine Learning in a repeatable manner. Although, when actual efforts are made to enable

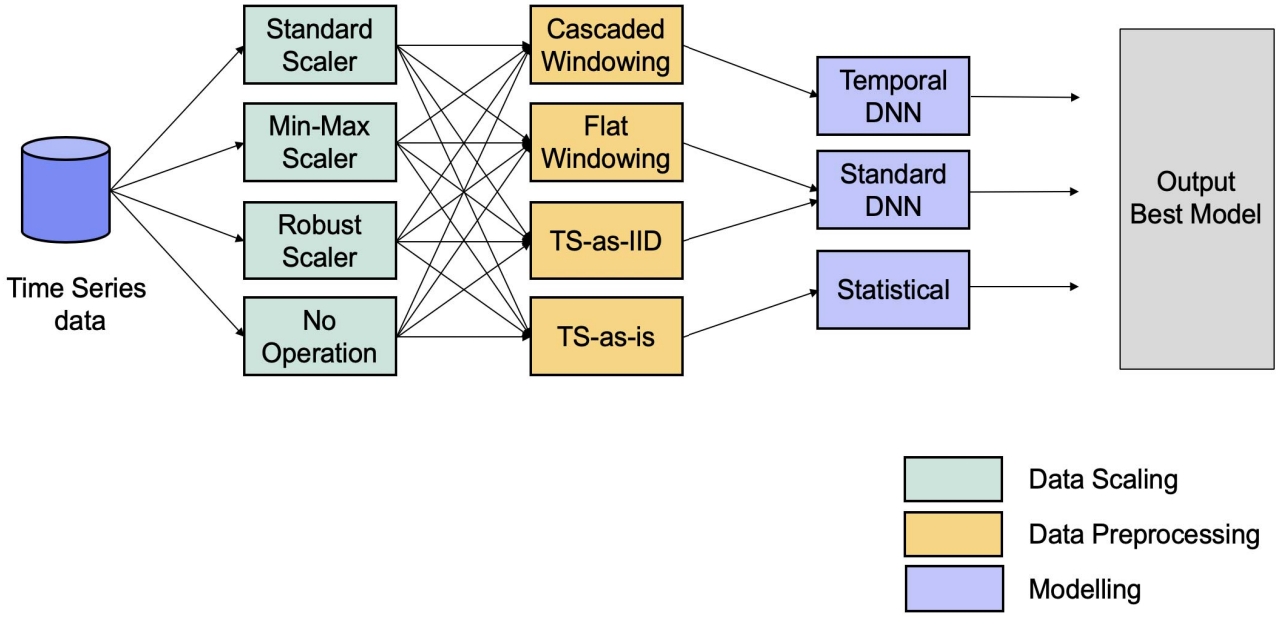

Fig. 11. The time series prediction pipeline graph consisting of three stages: 'Data Scaling', 'Data Preprocessing' and 'Modelling'. The output of the model is the best performing set of Transformers and Estimators.

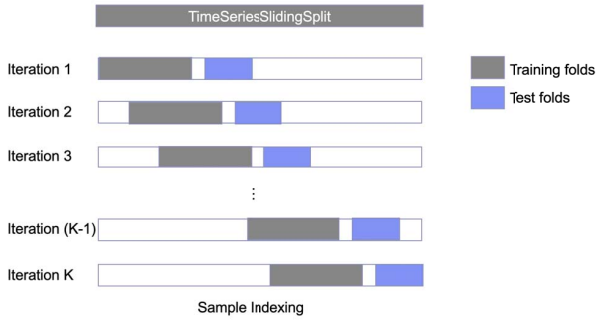

Fig. 12. Cross Validation: Time Series Sliding Split

such solutions in an industrial setting, challenges are faced to develop and adopt such systems. Often the people involved in building, operating and understanding such machine learning frameworks are managers, operators, data scientists and software developers. But building domain specific industrial solutions which leverage AI can be a daunting challenge even for a large organization due to the high skill set requirement for such tasks. We often encounter cases where a customer does not have the prerequisite knowledge to leverage advanced machine learning and AI tools available in the market. With our experience working with IBM clients in different domains, we have addressed this gap by providing industry specific solution templates which solve commonly observed problems in that industry. We leverage the Transformer-Estimator graphs to build such industry specific solution templates quickly. Some of such solutions commonly used in different industries include:

- *Failure Prediction Analysis (FPA)*: This solution pattern allows users to leverage historical sensor data and failure logs to build machine learning models to predict imminent failures.
- *Root Cause Analysis (RCA)*: This solution pattern enables operators to get a better understanding into the statistical reasons for favourable and unfavourable outcomes in industrial processes.
- *Anomaly Analysis*: This solution pattern builds a model to flag data as corresponding to a normal operation mode or an anomalous mode.
- *Cohort Analysis (CA)*: This solution pattern leverages historical sensor data from multiple assets to model their behaviour. Based on the similar patterns, assets are grouped in different buckets or cohorts allowing for a better understanding of industrial asset behavior.

These solution patterns have been implemented as Jupyter notebooks [19]. With such readily available solutions, the goal is to speed up the task of building Data Analytics and AI solutions quickly. The flexibility and easy of use provided by the Transformer-Estimator graphs makes the task of building such solutions considerably easier.

These solution templates provide consumable machine learning and AI capabilities to non-expert users. They are a key component in expanding the range of people who can take advantage of AI and machine learning capabilities. We believe that the use of domain-specific solution templates will become increasingly common as more people use AI and

machine learning techniques to analyze data in a wide variety of disciplines.

V. Related Work

There have been several open source machine learning frameworks which have become widely used in recent years. The data analytics algorithms provided by these frameworks can be used by our system. However, none of these frameworks provide the novel features of our system such as the ability to coordinate computations across multiple clients and to achieve cooperative data analytics across multiple clients and cloud analytics servers. We also offer unique capabilities to identify the best performing data analytics algorithms using Transformer-Estimator Graphs. Our work is complementary to these machine learning frameworks, and by leveraging their capabilities, our system can improve as these frameworks evolve and incorporate improvements in algorithms.

Scikit-learn is a widely used open source framework providing machine learning capabilities [1], [20], [21]. It provides several classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Much of scikit-learn is written in Python, while some of the core algorithms are written in Cython for improved performance.

TensorFlow is an open source framework from Google for building machine learning models [2], [21]. It is at a lower level than scikit-learn. It is frequently used for neural networks and deep learning and provides GPU support. Keras is a neural networks API written in Python and capable of running on top of TensorFlow, the Microsoft Cognitive Toolkit (CNTK), or Theano [5]. The Keras API is at a higher level than the TensorFlow API. People will often use the Keras API on top of TensorFlow. Keras makes it easier to analyze data using neural nets compared with the raw TensorFlow API. On the other hand, the raw TensorFlow API may be needed for users who need to customize computations beyond what Keras provides.

Torch is a machine learning library providing a variety of deep learning algorithms [22]. It is implemented in C++, but it is used via the Lua scripting language. As of 2018, Torch is no longer in active development. PyTorch is a Python package based on Torch that provides tensor computation (like NumPy), GPU acceleration and deep neural networks built on a tape-based autograd system [3]. PyTorch and TensorFlow are both widely used for deep learning. PyTorch is generally easier to use, but there are advantages to both frameworks.

Theano is a Python library for defining, optimizing, and evaluating mathematical expressions involving multi-dimensional arrays efficiently [23]–[25]. It is no longer being developed and enhanced. CAFFE (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework written in C++ but with a Python interface [4]. Caffe2, which was announced by Facebook in March 2017, includes additional features such as Recurrent Neural Networks. Caffe2 was merged into PyTorch after March 2018. The Microsoft Cognitive Toolkit (CNTK) is a deep learning toolkit that describes neural networks as a series of computational steps via a directed graph [26].

Several companies offer Web services which provide machine learning capabilities using an HTTP interface, including IBM [9], Amazon [11], Google [12], H2O.ai [27] and Microsoft [10].

GraphLab is a parallel framework for machine learning using a graph-based data model for representing data and computational dependencies [28], [29]. It is targeted for sparse iterative graph algorithms. Spark MLlib [30] is a distributed machine learning framework for Apache Spark [7] built on top of the Spark Core. Apache Mahout is a project of the Apache Software Foundation containing scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification [31].

VI. Summary and Conclusion

We have presented a distributed system for performing data analytics and machine learning. Our system allows multiple users to cooperate in performing data analytics computations to scale out processing and avoid redundant computations. We introduced the Transformer-Estimator Graph, a powerful framework for determining optimal models for analyzing data sets. We showed how graphs can be effectively used for analyzing time series data common in heavy industry. In order to make our system easily usable by non-experts, we provide solution templates which are customized to problems in specific domains.

References

[1] 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," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[3] N. Ketkar, "Introduction to pytorch," in *Deep learning with python*. Springer, 2017, pp. 195–208.

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[5] A. Gulli and S. Pal, *Deep Learning with Keras*. Packt Publishing Ltd, 2017.

[6] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[7] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016. [Online]. Available: http://doi.acm.org/10.1145/2934664

[8] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[9] IBM, "Watson Machine Learning," https://www.ibm.com/cloud/machine-learning.

[10] Microsoft, "Microsoft Azure Machine Learning Studio," https://azure.microsoft.com/en-us/services/machine-learning-studio/.

[11] Amazon, "Machine Learning on AWS," https://aws.amazon.com/machine-learning/.

[12] Google, "Cloud AI Products," https://cloud.google.com/products/ai/.

[13] C. Gray and D. Cheriton, "Leases: An efficient fault-tolerant mechanism for distributed file cache consistency," in *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, 1989, pp. 202–210.
[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[15] M. Sewak, M. R. Karim, and P. Pujari, *Practical Convolutional Neural Networks: Implement Advanced Deep Learning Models Using Python*. Packt Publishing, 2018.
[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.
[17] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio." in *SSW*, 2016, p. 125.
[18] K. Kristpapadopoulos. (2018) SeriesNet. [Online]. Available: https://github.com/kristpapadopoulos/seriesnet
[19] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, "Jupyter notebooks-a publishing format for reproducible computational workflows." in *ELPUB*, 2016, pp. 87–90.
[20] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler *et al.*, "Api design for machine learning software: experiences from the scikit-learn project," *arXiv preprint arXiv:1309.0238*, 2013.
[21] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.
[22] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," Technical Report IDIAP-RR 02-46, IDIAP, Tech. Rep., 2002.
[23] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proc. 9th Python in Science Conf*, vol. 1, 2010, pp. 3–10.
[24] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, "Theano: new features and speed improvements," *arXiv preprint arXiv:1211.5590*, 2012.
[25] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv preprint arXiv:1605.02688*, 2016.
[26] D. Yu, A. Eversole, M. Seltzer, K. Yao, Z. Huang, B. Guenter, O. Kuchaiev, Y. Zhang, F. Seide, H. Wang *et al.*, "An introduction to computational networks and the computational network toolkit," *Microsoft Technical Report MSR-TR-2014–112*, 2014.
[27] H2O.ai, "H2O.ai: Fast Scalable Machine Learning For Smarter Applications," https://github.com/h2oai.
[28] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.
[29] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.
[30] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.
[31] A. S. Foundation, "Apache Mahout," https://mahout.apache.org/.