End-to-End Machine Learning with Apache AsterixDB

Wail Alkowaileet
University of California Irvine
w.alkowaileet@ics.uci.edu

Sattam Alsubaiee
Center for Complex Engineering Systems at KACST and MIT
ssubaiee@kacst.edu.sa

Michael J. Carey, Chen Li
University of California Irvine
mjcarey,chenli@ics.uci.edu

Heri Ramampiaro
Norwegian University of Science and Technology (NTNU)
heri@ntnu.no

Phanwadee Sinthong
University of California Irvine
psinthon@ics.uci.edu

Xikui Wang*
University of California Irvine
xikuiw@ics.uci.edu

ABSTRACT

Recent developments in machine learning and data science provide a foundation for extracting underlying information from Big Data. Unfortunately, current platforms and tools often require data scientists to glue together and maintain custom-built platforms consisting of multiple Big Data component technologies. In this paper, we explain how Apache AsterixDB, an open source Big Data Management System, can help to reduce the burden involved in using machine learning algorithms in Big Data analytics. In particular, we describe how AsterixDB's built-in support for user-defined functions (UDFs), the availability of UDFs in data ingestion pipelines and queries, and the provision of machine learning platform and notebook inter-operation capabilities can together enable data analysts to more easily create and manage end-to-end analytical dataflows.

ACM Reference Format:
Wail Alkowaileet, Sattam Alsubaiee, Michael J. Carey, Chen Li, Heri Ramampiaro, Phanwadee Sinthong, and Xikui Wang. 2018. End-to-End Machine Learning with Apache AsterixDB. In *DEEM'18: International Workshop on Data Management for End-to-End Machine Learning, June 15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209889.3209894

1 INTRODUCTION

Increasingly our world is being digitized, and there is much to be learned from its analysis. Data analysis on Big Data is a very challenging task as it can be very difficult to handle big datasets. The good news is that recent developments in machine learning and data science provide solid foundations for extracting information from large volumes of data and using it in understanding our rapidly evolving world. The bad news is that the current platforms and tools for machine learning, data analysis, and data management – especially at scale – tend to be disjoint. As a result, data scientists must glue together and maintain custom-built heterogeneous platforms involving multiple Big Data components – e.g., HDFS, Pig, Hive, Spark, TensorFlow, and so on – which requires significant effort, introduces ETL delays, and demands computer systems expertise from analysts who should instead be focused on data modeling, selection of machine learning techniques, algorithms, and data exploration.

In order to reduce the expertise and effort levels required for data scientists to work with Big Data, we need a system that provides scalability for conducting data analyses on large-scale datasets, extendability with machine learning libraries, and integrability with Big Data analytics tools. To move forward towards this goal, we use Apache AsterixDB [4] and enhance it with data analytics features. Apache AsterixDB provides support for the scalable storage and analysis of large volumes of semi-structured data. It offers a flexible data model to enrich data with outputs from machine learning models, data feeds for fast data ingestion, and a user-defined function (UDF) framework for creating customized dataflows with external libraries. In this paper, we describe how to leverage these features for supporting end-to-end data analytics with Machine Learning.

The rest of the paper is organized as follows. Section 2 discusses some existing systems that are widely used within Big Data analytics, including their features and the difficulties of using them as-is for end-to-end data analytics. Section 3 introduces the building blocks that we use to build our end-to-end data analytics platform. Section 4 explains how one can incorporate machine learning algorithms into AsterixDB using the components introduced in Section 3. Section 6 summarizes the paper and outlines our plans for future work.

2 RELATED WORK

As noted earlier, most of the current approaches to end-to-end data analytics require gluing together multiple disjoint components. Here, we discuss some related systems that are used in supporting machine learning on large-scale datasets.

Redis. Redis [19] is an in-memory, key-value database. It is often used as a cache and message broker. Redis now provides a *redis-ml* package as an extension to enable users to apply machine learning algorithms on stored data. It supports several popular algorithms, including clustering, logistic regression, and linear regression, which cover many common machine learning tasks. However, it does not support integrating external libraries, which means that users have

*Corresponding author

DEEM'18, June 15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-5828-6/18/06…$15.00
https://doi.org/10.1145/3209889.3209894

to restrict their data analytics tasks to applying the provided algorithms. It also lacks query capabilities for interactively exploring data.

Hive. Apache Hive [21] is a data warehouse based on Apache Hadoop. It provides file-based storage and query support for archived distributed datasets. Hive provides a UDF interface for users to add customized data processing and analytics. As Hive depends on the Hadoop ecosystem, it requires its users to have significant Hadoop/HDFS expertise.

Spark. Apache Spark [22] is a general-purpose cluster computing system that provides in-memory parallel computation on a cluster with scalability and fault tolerance. MLlib [15], which is built on top of Spark, provides the capability of running machine learning algorithms on large-scale datasets. Although Spark supports querying and accessing structured data inside a Spark program, it does not provide data storage and management. It typically works with HDFS for distributed data processing.

TensorFlow. TensorFlow [2] is a machine learning platform that provides computation support for various analytics tasks. It utilizes computing resources across multiple nodes and different computational devices. TensorFlow is solely a computing platform; similar to Spark and Hive, it relies on a distributed file system for data management.

Weka. Weka [1] is an open-source platform written in Java aiming at providing machine learning algorithms for data mining tasks. It provides supports for regression, classification, clustering, association rule mining, and attribute selection. Being merely a machine learning algorithm suite, it needs additional runtime and data management platforms to be able to work in a distributed environment.

MonetDB. MonetDB [16] is an open-source columnar database management system. It was designed for data warehouse applications and offers significant speedups through its storage and query execution models. To address users familiar with R, MonetDB provides embedded R support to enable native R script execution through SQL functions. While it excels at in-memory execution, MonetDB has limited scale-out capabilities that therefore limit its usefulness for Big Data use cases.

MADlib. Apache MADlib [9] introduced an innovative approach to implementing machine learning, data mining, and statistical algorithms using SQL running on a database engine. It minimizes the cost of data movement for import/export and utilizes the parallelism provided by modern parallel database engines. This library provides many popular algorithms that can be used to build a customized analytics dataflow. To use these implementations, users need to adapt their current analytical pipelines to the MADlib library.

Each of the systems mentioned above provides some support for utilizing machine learning algorithms in Big Data analytics. Some systems provide a limited number of machine learning models (Hive, Redis), while others rely on other systems to work against large-scale datasets (TensorFlow, Spark, Weka). MADlib focuses on enabling machine learning on database engines, but it requires users to re-implement their pipelines. All of them demand extra work from users who may have limited expertise in computer systems to solve the end-to-end data analytics puzzle.

3 OUR BUILDING BLOCKS

To build an end-to-end data analytics system, one needs a storage engine that manages distributed data, a runtime execution engine that supports distributed computation, a user-friendly interface for accessing the data, and an extensible framework that enables the use of external libraries. Instead of starting from scratch, we have chosen to enhance Apache AsterixDB with machine learning libraries/systems for data analytics using its extensibility features.

3.1 Apache AsterixDB

Apache AsterixDB is an open source Big Data Management System that provides distributed data management for large-scale semi-structured data. Here we focus on several important features that can be used in data analytics.

3.1.1 User Model. Analyzing data in the current digitized world requires a system that can handle different data formats and provide support for various data manipulation operations. To meet this requirement, an important feature of AsterixDB is its flexible data model. Before storing data into AsterixDB, a user can create a *Datatype*, which describes known aspects of the data being stored, and a *Dataset*, which is a collection of records of a Datatype. AsterixDB allows a Datatype to be "open", in that the description of the data does not need to be complete prior to storing it. This means that a user can insert data records with additional fields, such as semantic annotations, into a dataset directly. For example, as shown in Fig. 1, only "id" and "text" are specified at creation time, but the inserted records each have an additional field, i.e., "sentiment" and "entities", respectively. The data model is a superset of JSON and supports complex objects with nesting and collections. This gives the user more flexibility in data analytics applications as compared with other systems. In addition, AsterixDB provides SQL++ [17] as the query language that allows data scientists to query their datasets without scripting or low-level programmings.

```
-- Create an open datatype
CREATE TYPE Tweet {
    id: int64,
    text: string
};
-- Create a dataset with a given datatype
CREATE DATASET Tweets(Tweet);
-- Insert records with additional attributes
-- into a dataset.
INSERT INTO Tweets
([{"id":1,"text":"This is a happy tweet!",
  "sentiment":"Positive"},
  { "id":2, "text": "We are located in Irvine CA,
  30 mins from Disneyland!!",
  "entities": [ "Disneyland", "Irvine", "CA"]}]);
```

Figure 1: Insertion of records with additional attributes into a dataset.

3.1.2 Data Feeds. In many Big Data applications, data are generated continuously and at high speed. How to capture "fast moving"

data and persist it in a database is an important problem that has to be solved when building Big Data Analytics applications. AsterixDB provides "data feeds" for ingesting/parsing/storing data from external data sources with scalability and fault tolerance [8]. A user can ingest data from files, a prescribed socket, Twitter, RSS feeds, and other Internet sources by creating or using feed adapters and connecting them to AsterixDB datasets. By making connections between a feed adapter and a dataset, data records ingested from the external data source will be stored in this dataset directly. As an example, Fig. 2 shows the sequence of DDL statements needed to create a Twitter feed and begin collecting live tweets in the previously defined dataset.

```
CREATE FEED TwitterFeed with {
   "adapter-name" : "push_twitter",
   "type-name" : "Tweet",
   "format" : "twitter-status",
   "consumer.key" : "***",
   "access.token" : "***",
   "access.token.secret" : "***"
 };
CONNECT FEED TwitterFeed to Tweets;
START FEED TwitterFeed;
```

Figure 2: Create a Twitter feed to collect tweets.

3.1.3 External User Defined Functions. There are use cases in which a user wants to perform complex operations on stored data that cannot be neatly expressed in a declarative query language. Like various other RDBMSs and Big Data platforms [20], AsterixDB provides an external user-defined function (external UDF) framework that enables users to plug in their own functions written in popular programming languages (e.g., Java and Python). By implementing the function interface, a user can read and manipulate records using external UDFs. Once defined, UDFs can be used like native functions in queries. For more complex processing tasks, like running machine learning models against collections of records, users can leverage functions from external machine learning libraries in external UDFs as well.

To illustrate, assume a user wants to implement a simple function, called "`getNaiveSentiment`", that extracts the sentiment of an input sentence based on the aggregated sentiment score of each word in the sentence. To create his/her function with AsterixDB, the user implements a pre-defined external UDF interface and a factory class that is used for creating function instances at runtime. Then, to deploy the function, he/she creates a configuration file, as shown in Fig. 3, which specifies metadata information about the function, including the expected parameter type for the function, the output type, and the name to be used in queries for invoking the function[1]. As an example of how a user can invoke "`getNaiveSentiment`" in a query, consider the sample query in Fig. 4. Here, the query reads an input list of sentences, processes them using a UDF, and returns a list of records containing both the text and the corresponding sentiments.

The evaluation process for UDF consists of the following three stages: initialization, evaluation, and deinitialization, as shown in

[1]Notice that the function name ("`getNaiveSentiment`") used in queries can be different from the implementation name ("`NaiveSentimentExtractor`").

```
<libraryFunction>
  <function_type>SCALAR</function_type>
  <name>getNaiveSentiment</name>
  <arguments>ASTRING</arguments>
  <return_type>TextSentimentType</return_type>
  <definition>org.apache.asterix.external.library.
    NaiveSentimentExtractorFactory
  </definition>
</libraryFunction>
```

Figure 3: A sample function description.

```
/* We assume the following datatype is created.
CREATE TYPE TextSentimentType {
    text: string,
    sentiment: string
};
*/
SELECT VALUE simplelib#getNaiveSentiment(text) FROM
["DEEM is the best workshop.",
"The authors are frustrated.",
"The best author is very sad.",
"This sentence is neutral."] as text;
/* Result:
{ "text": "DEEM is the best workshop.",
"sentiment": "Positive" }
{ "text": "The authors are frustrated.",
"sentiment": "Negative" }
{ "text": "The best author is very sad.",
"sentiment": "Positive" }
{ "text": "This sentence is neutral.",
"sentiment": "Neutral" }
*/
```

Figure 4: A sample query that calls an external UDF.

Fig. 5. When an external UDF is invoked for the first time within a query, the "`initialize`" method is executed for all necessary initialization, such as loading resource files. In the example, the initialization method first reads two lists of positive and negative words, respectively, which are used to map words to sentiment scores. The "`evaluate`" method processes an input value, computes the aggregated sentiment scores, and returns the result. The "`deinitialze`" method is executed to, e.g., do necessary clean-ups at system shutdown time.

3.2 Machine Learning Platforms

With the external UDF framework introduced in the previous section, users can integrate models from their favorite machine learning libraries into AsterixDB for complex data analytics tasks. Machine learning libraries come with a variety of usability, scalability, and extensibility characteristics. Examples include Weka [1], Spark MLLib [15], Caffe [12], Scikit-Learn [18], and TensorFlow [2]. In this paper, we focus on how AsterixDB can be used to facilitate typical machine learning tasks in both lab-scale experiments for model

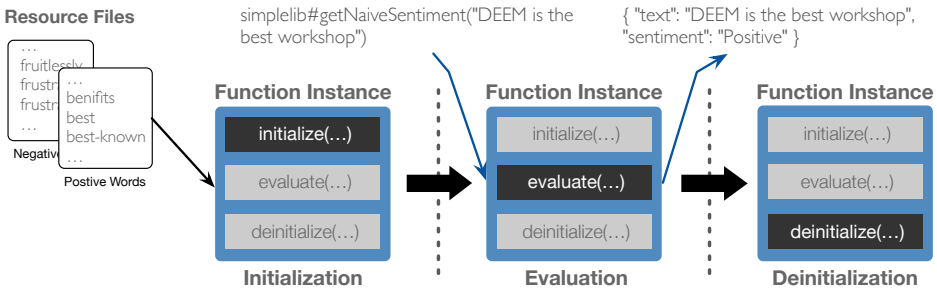


Figure 5: The lifecycle of the "getNaiveSentiment" UDF.

validation and large-scale data analytics for real-world applications using Weka, Spark MLlib, and scikit-learn. We refer the reader to Section 2 for the highlights of the first two platforms.

4 MACHINE LEARNING IN ASTERIXDB

4.1 The Lifecycle of Big Data Analytics

A common data analytics workflow generally includes the following three main tasks [7]:

(1) Collecting and loading data;
(2) Training models using machine learning algorithms; and
(3) Making predictions with the trained models.

As noted earlier, AsterixDB provides data feeds for fast data ingestion from different sources (i.e., task (1)). In this section, we discuss how AsterixDB utilizes components from the building blocks in Section 3 to support the other two tasks. For illustration purposes, we will use records from the dataset `Tweets` in Fig. 6 throughout this section.

4.2 Feature Extraction with AsterixDB

AsterixDB provides different ways of serving data to different machine learning platforms for feature extraction. We discuss the following two cases depending on the size of the dataset that a user wants to work with. For a small training dataset, a user can utilize the AsterixDB query service to export the stored data. AsterixDB provides different formats for returning query results, including both JSON and CSV, that can be fed into a machine learning platform like Weka directly. For big training datasets, we provide a Spark-connector [3] that can move data in parallel from AsterixDB to Spark in order to train machine learning models efficiently.

4.2.1 Small training datasets. When the dataset is small, a user can query data from AsterixDB and feed the result into a separate feature extraction program, or they can embed feature extraction code into a UDF to query and export data features directly. Here we use Weka Explorer as an example to train and build classification models using features exported from AsterixDB. To illustrate, consider the query and the corresponding result in Fig. 7. In this example, we apply a pre-coded UDF called `''extractFeatures''` to extract some pre-specified simple counting features, including "terms" (the number of unique terms), "topics" (number of hashtags), "tags" (number of mentions), "links" (number of URLs), and "sentiment_rate" (number of sentiment-related words).

The main advantage of this approach is that a Weka-familiar user can use a desired sample of the data stored in the database to extract features of interest, export the result to a JSON or CSV file, and import the features into Weka in the usual Weka manner for analysis or experimentation. Further, a more advanced user can use the UDF framework to implement his/her own UDF to extract more complex features (e.g., nested or non-numerical features), thus making the feature extraction UDF a useful part of a complete feature engineering process.

4.2.2 Large training datasets. For large datasets, AsterixDB can be coupled with a scalable machine learning platform such as Apache Spark. The AsterixDB-Spark connector [3] provides a connection between the two distributed systems and has the ability to exploit data locality to minimize the data transfer cost whenever possible. This coupling allows data scientists to leverage the capabilities of both systems to answer a specific question or to test a hypothesis efficiently. For instance, a user can fetch only a desired subset of the data that satisfies certain spatial, temporal and/or textual predicates using the indexes provided by AsterixDB and stream the query results into Spark to perform a complex analysis or to train a machine learning model.

To illustrate, we consider a case where we wish to analyze a large volume of tweets to determine the discussion topics in a specific geographic region of interest. To get the tweets that have been posted in the region, a spatial query can be submitted to AsterixDB from Spark using the AsterixDB-Spark connector to get the result into Spark for analysis as shown in Fig. 8. [3] The Spark variable `''df''` is a Spark DataFrame that contains the query results, which can be used for further processing with Spark's built-in algorithms. In this case, suppose we want to train a topic model with the stored tweets in AsterixDB using the Latent Dirichlet Allocation (LDA) algorithm[6]. As a first step, we can prepare the tweets (using the function `''preprocess''` in Fig. 8) by removing stopwords and computing term frequencies. Then, we can set the LDA parameters and run the algorithm to obtain the topic model. The trained model is stored in the variable `''ldamodel''`, which can be used in the current Spark program or be exported to a local file. Note that all of the steps can be done in the data scientist's familiar Spark world.

[3]The AsterixDB-Spark connector allows Spark worker nodes to read query result partitions from AsterixDB nodes in parallel. When AsterixDB and Spark worker nodes are co-located the connector tries to exploit data locality to minimize data movement.

```
{ "id":1, "text": "I'm so happy to be here!", "geo_coordinates": point("47.44,80.65")}
{ "id":2, "text": "We are located in Irvine CA, 30 mins from Disneyland!!", "geo_coordinates": point("36.50,60.65"),
"city":"Irvine, CA"}
{ "id":3, "text": "Honored to welcome Georgia Prime Minister, Giorgi Kvirikashvili to the @WhiteHouse today with @VP Mike Pence.",
"geo_coordinates": point("31.24,98.61"), "city":"Washington, DC"}
{ "id": 4, "text": "getting ready to test out some burger receipes this weekend. Bobby Flay has some great receipes to try. Thanks
Bobby.", "geo_coordinates": point("26.42,42.13"), "city":"Phoenix, AZ"}
{ "id": 5, "text": "i lam so in love with Bobby Flay... he is my favorite. RT @terrysimpson: @bflay you need a place in Phoenix. We
have great peppers here!", "geo_coordinates": point("12.44,42.66"), "city":"Phoenix, AZ"}
{ "id": 6, "text": "using Linux and loving it - so much nicer than windows... Looking forward to using the wysiwyg latex editor!",
"geo_coordinates": point("44.42,86.61"), "city":"Irvine, CA"}
{ "id": 7, "text": "Reading the tweets coming out of this accident... The whole thing is terrifying and incredibly sad...",
"geo_coordinates": point("34.44,86.65"), "city":"Los Angeles, CA"}
```

Figure 6: Contents of the sample "Tweets" dataset.

```
SELECT t.id as id,
wekalib#extractFeatures(t) as features
FROM Tweets t;
/* Result:
{ "id": 1, "features": { "terms": 2.0, "topics": 0.0,
"tags": 0.0, "links": 0.0, "sentiment_rate": 1.0,
"class": "?" } }
{ "id": 2, "features": { "terms": 6.0, "topics": 0.0,
"tags": 0.0, "links": 0.0, "sentiment_rate": 0.0,
"class": "?" } }
{ "id": 3, "features": { "terms": 12.0, "topics": 0.0,
"tags": 2.0, "links": 0.0, "sentiment_rate": 2.0,
"class": "?" } }
...
*/
```

Figure 7: Example extraction of features from the sample tweets using a UDF.[2]

4.3 Bringing Machine Learning to AsterixDB

With the external UDF framework introduced in Section 3, AsterixDB can utilize machine learning libraries for data analytics on large-scale datasets. Depending on the application needs, a user can either use a specialized library for specific tasks, or they can use more general tools such as Weka and Spark with trained models.

4.3.1 External UDFs from a specialized machine learning library. Some libraries are designed to perform analytical tasks within specific application contexts. AsterixDB enables a user to integrate such libraries by using external UDFs so that they can be utilized for data analytics. Assume that we want to extract the sentiment of an incoming tweet based on its text content. One possible approach is to use the Stanford CoreNLP library [14], which focuses on text annotation and semantic analytics, via an external UDF. We can utilize its sentiment analysis algorithm to create the desired AsterixDB UDF as illustrated in Fig. 9. For simplicity, we segment the tweet text into sentences and use the sentiment for the longest sentence as the primary sentiment for the tweet.

Like our example in Section 3, the configuration for the sentiment analysis external UDF is shown in Fig. 10. Here, we define the function name to be `getSentiment`. We specify `Tweet` as the input record type for describing Tweets from Twitter and `TweetSentimentType` as the result data type for annotated sentiment information. A sample SQL++ query for invoking sentiment analysis on all of the records in our sample dataset is shown in Fig. 11. Similarly, we can utilize other capabilities from the CoreNLP library with external UDFs to extract more semantic information from Twitter text. Fig. 12 shows two additional examples that extract locations and names from tweets.

```
...
// load data into DataFrame
val df = sqlContext.sqlpp("""
SELECT t.text, t.city
FROM Tweets t
WHERE spatial_intersect(
    create_rectangle(
        create_point(-107.27, 33.06),
        create_point(-89.1, 38.9)),
    t.geo_coordinates)
""")

// remove stop words and compute term frequency
val documents = preprocess(df)
// initialize the LDA algorithm
val lda = new LDA()

// set the running parameters for the LDA
lda.setOptimizer(new EMLDAOptimizer)
    .setK(14)
    .setMaxIterations(1000)
val ldamodel = lda.run(documents)
```

Figure 8: Get tweets from AsterixDB into Spark as a DataFrame and run the LDA algorithm on the resulting data using Spark MLlib.

4.3.2 External UDFs from general machine learning libraries. In situations where a user cannot find an appropriate specialized library or wants to use customized models, AsterixDB provides support for incorporating customized models via external UDFs with general machine learning libraries. A user can pick up a provided general machine learning algorithm implementation and configure it with his/her own models depending on the applications. Here we use Weka as an example to show how we could extract similar sentiment information from Twitter text with a general "Random Forest" classifier and a trained Weka model file.

[3]Here, "?" in a class field means that the class of the record is to be set by the classifier later.

```
// Create the Stanford CoreNLP pipeline in initialize(...).
...
props.setProperty("annotators", "tokenize, ssplit, parse, sentiment");
StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
...
// Process the incoming tweet in evaluate(...).
...
String tweet = ((JString) inputRecord.getValueByName("text")).getValue();
if (tweet != null && tweet.length() > 0) {
  CoreMap longestSentence = getLongestSentence(pipeline.process(tweet));
  Tree annotatedTree = longestSentence.get(SentimentCoreAnnotations.SentimentAnnotatedTree.class);
  mainSentiment = RNNCoreAnnotations.getPredictedClass(annotatedTree);
}
...
```

Figure 9: A wrapper code snippet for utilizing the Stanford CoreNLP library in a UDF.

```
<libraryFunction>
  <function_type>SCALAR</function_type>
  <name>getSentiment</name>
  <arguments>Tweet</arguments>
  <return_type>TweetSentimentType</return_type>
  <definition>org.apache.asterix.external.
  library.SentimentAnalysisFactory</definition>
</libraryFunction>
```

Figure 10: A sample configuration for a sentiment extraction UDF.

```
LET item=snlplib#getSentiment(t)
SELECT item.id, item.sentiment
FROM Tweets t;
/* Result:
{ "id": 1, "sentiment": "Very Positive" }
{ "id": 2, "sentiment": "Negative" }
{ "id": 3, "sentiment": "Negative" }
{ "id": 4, "sentiment": "Negative" }
{ "id": 5, "sentiment": "Positive" }
{ "id": 6, "sentiment": "Positive" }
{ "id": 7, "sentiment": "Negative" }
*/
```

Figure 11: Invoking an external UDF using a query.

As discussed in Section 3, AsterixDB allows UDFs to load resource files during the function initialization. A UDF that implements a general algorithm can be implemented once and then adapted to different applications by loading application-specific model files. For example, the same UDF implementing the "Random Forest" algorithm can be used to apply multiple different trained models by specifying the needed model files in the UDF configuration file in Fig. 13. Thus, a user can use the same underlying UDF implementation for sentiment analysis, spam detection, and other analytics tasks depending on the application requirements without reprogramming. Together with the feature extraction UDF discussed in Section 4.2, we can realize a data analytics cycle with

```
SELECT snlplib#getLocation(t)
FROM Tweets t
WHERE t.id = 2;
/* Result:
{ locations: [ "Disneyland",
"Irvine", "CA"] }
*/
SELECT snlplib#getName(t)
FROM tweets t
WHERE t.id = 3;
/* Result:
{ names: [ "Kvirikashvili", "Mike",
"Giorgi", "Pence"] }
*/
```

Figure 12: External UDFs adding more semantic information to a tweet record.

Weka and AsterixDB as dictated in Fig. 13. Note that while this figure shows the coupling of the configured UDF with a feed adapter, this UDF can be invoked in queries as well. To compare with the sentiment analysis result in Fig. 11, the corresponding result for the sample dataset using Weka is shown in Fig. 14.

Choosing an external UDF with a general machine learning library offers more flexibility as compared to using prepackaged specialized libraries, allowing users to make a trade-off between algorithm/model complexity and processing costs. By leveraging the UDF framework, users are able to invoke machine learning models on feeds (streaming input data) and on stored data as well as using them to extract features from a query-filtered data subset.

4.3.3 Applying Native Spark Models in AsterixDB. To complete the data analytics cycle with the Spark trained models from Section 4.2, we also support executing native Spark model in AsterixDB. This feature is still under active development, but we summarize it briefly here. Our aim is for the users to be able to run native Spark models on an AsterixDB cluster. Spark allows a user to export a trained machine learning model or workflow pipeline (a series of models) into local files. The exported file contains necessary configurations that can be used for reconstructing Spark models at runtime. A UDF can read this configuration file during

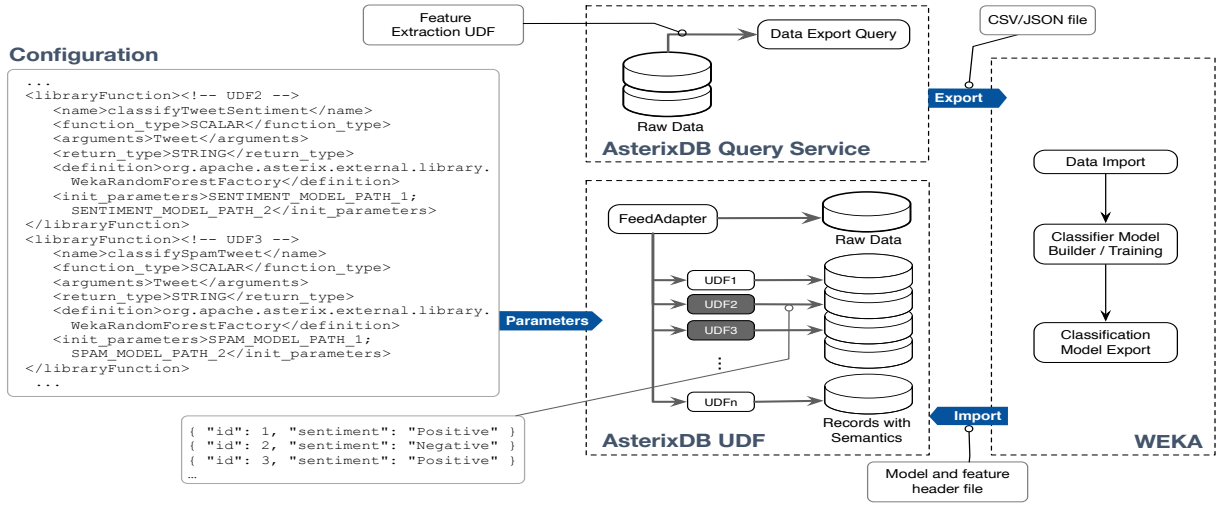


Figure 13: Data analytics cycle with Weka and AsterixDB.

```
SELECT t.id AS id, wekalib#classifyTweetSentiment(t)
       AS sentiment FROM Tweets t;
/* Result:
{ "id": 1, "sentiment": "Positive" }
{ "id": 2, "sentiment": "Negative" }
{ "id": 3, "sentiment": "Positive" }
{ "id": 4, "sentiment": "Positive" }
{ "id": 5, "sentiment": "Positive" }
{ "id": 6, "sentiment": "Positive" }
{ "id": 7, "sentiment": "Negative" }
*/
```

Figure 14: Example of sentiment classification with a Weka-based UDF.

initialization and rebuild the workflow for per-item evaluation. By implementing these interfaces between AsterixDB and Spark, we enable evaluating native Spark models in AsterixDB feeds and queries in parallel.

4.3.4 Applying Scikit-Learn Models in AsterixDB. In addition to Weka and Spark MLlib, we also support applying Scikit-Learn trained models as a UDF in the same fashion as that of executing native Spark models. We offer an ability to call a serializable Scikit-Learn model/pipline in AsterixDB. This feature is implemented using a third-party library called Jep [10], which embeds CPython in Java through JNI. It enables communication between AsterixDB's Java-centric system and Scikit-Learn's Python models. Python users can easily load in their data, train an ML model, and then apply it seamlessly using AsterixDB and predefined UDFs. To simulate a complete data analytics pipeline, we first train a sentiment classifier using dataset from UC Irvine Machine Learning Repository[13] containing 3000 labeled sentences from Yelp, IMDB, and Amazon reviews. The dataset is labeled using scores of 1 (for positive) and 0 (for negative). A sample code shown in Fig. 15 illustrates how to export the trained pipeline as an on-disk file called "sentiment_pipeline". Second, to apply this model on an incoming

```
pipeline = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('classifier', MultinomialNB())])
                    .
                    .
          Train a Scikit-Learn Pipeline
                    .
                    .
pickle.dump(pipeline, open("sentiment_pipeline", 'wb'))
```

Figure 15: Sample code to export a Scikit-Learn trained model.

feed or to a large dataset distributed across multiple nodes, users only have to edit the UDF configuration file (Fig. 16) to include the path to the previously exported model. After updating the configuration file and relaunching the AsterixDB instance, the model is accessible as a function call. Fig. 17 shows a standard Python's URL-handling module that calls our HTTP API and loads the data into pandas DataFrame. Finally, we present a Jupyter notebook (Fig. 18) showing sample queries applying the sentiment function to a large Twitter dataset. The notebook includes bar graphs of sentiment scores on Tweet sentences mentioning three 2016 US presidential candidates.

```
<libraryFunction>
  <function_type>SCALAR</function_type>
  <name>getSentiment</name>
  <arguments>ASTRING</arguments>
  <return_type>AINT32</return_type>
  <definition>ScikitLearnSentimentFactory</definition>
  <init_parameters>/home/user/sentiment_pipeline
  </init_parameters>
</libraryFunction>
```

Figure 16: A sample configuration for a Scikit-Learn Sentiment UDF.

```
def send_request(query: str):
        host = 'http://localhost:19002/query/service'
        data = dict()
        data['statement'] = query
        data = urllib.parse.urlencode(data).encode('utf-8')
        with urllib.request.urlopen(host, data) as handler:
            result = json.loads(handler.read())
            df = json.read_json(json.dumps(result['results']))
            return df
```

Figure 17: A function to send a SQL++ query to the AsterixDB HTTP API.

4.4 Utilizing Machine Learning in AsterixDB

As we have discussed, depending on the users' preferences and analytics requirements, an external UDF for data analysis can be placed at different points in the data analytics pipeline. For a data-enriching use case, a user may want the analytics to be applied upon data arrival and then to be able to query data annotated with the enriched information at query time. For an ad-hoc analysis use case, a user may just want to apply the analytics function at runtime together with other query predicates and expressions. AsterixDB provides mechanics for supporting data analytics in both use cases.

To evaluate functions upon data arrival, the user can attach the analytic function to a feed and have AsterixDB evaluate it during the ingestion process. Fig. 19 shows an example of adding the sentiment analysis function to a feed adapter ("localfs") for incrementally loading data from the local file system. By attaching the function to the feed, all ingested records will be processed by the attached function as they enter the database. In this example, all incoming tweets will be annotated with a sentiment based on their text.

To evaluate such analytical UDFs at query time, users may include function calls in their queries to apply them to the data records directly. An example is shown in Fig. 20. Functions can be applied to any subset of records with proper predicates, and the result can be returned to either the query interface or another dataset. Since the function evaluation becomes part of the query execution, the overall execution time will be affected by the choice of machine learning algorithms and implementation libraries.

4.5 A Data Scientist Sandbox

To further alleviate the effort of working with multiple data sources and data engines for data scientists, the notebook interface has come into play for many data analytics platforms. It helps data scientists become more productive by providing a unified interface for organizing and executing codes and visualizing and exporting results without referring to each low-level system details. Apache Zeppelin [5] is one of the widely used notebook environments; it is a collaborative, web-based notebook with built-in support for Python, Scala, and SQL as well as many types of data visualizations. Internally, Zeppelin uses an `Interpreter` to manage different languages and data-processing engines. An advanced Zeppelin user is able to add support for additional languages by implementing its `Interpreter` interface and can extend the visualization library by adding new charts and graphs.

We have extended Zeppelin for AsterixDB by adding an AsterixDB SQL++ interpreter. As shown on the left-hand side of Fig. 21, a Zepplin user can formulate a SQL++ query using the AsterixDB interpreter and see the resulting data using a Zepplin visualization. Alternatively, he/she can invoke the LDA algorithm from Spark's MLlib to explore the topics of the data returned by the query using the same Zeppelin interface (as shown on the right-hand side of the figure). The query's results can also be exported to other analytics tools for further analysis.

5 ILLUSTRATIVE EXPERIMENTS

In this section, we demonstrate experimentally the benefit of using AsterixDB for end-to-end machine learning based data analytics. These experiments aim to show the speed-up and scale-out performance of the system for computing-intensive analytical tasks. Following the social media theme, we created a feed pipeline using a socket adapter that receives data from a external data source. A sentiment analysis UDF was attached to the feed pipeline to annotate the incoming records. We use a socket client to push data into this pipeline to measure the overall time that the system spends on processing the incoming data.

5.1 Configurations

Cluster: The experiments were conducted on a 8-node cluster with a Gigabit Ethernet switch. Each node had a Dual-Core AMD Opteron Processor 2212 2.0GHz, 8GB of RAM, and a 900GB hard disk. **Data:** We used an offline synthetic tweet generator to generate the data. The size of each generated tweet was about 300bytes, which contained id, timestamp, language, message_text, etc. The sentiment analysis was based on the "message_text" field. **Library:** The Standford NLP library was used for extracting sentiment out of the incoming records. Part of our UDF implementation was shown in Fig. 9. With the hardware used here, a single thread sentiment analysis UDF with StandfordNLP can process about 6 records per second. **Measurements:** We measured the execution time that the system spent on processing all incoming data, in seconds. Also, we kept track of the aggregated time that each processor spent on processing the records. We compared the per-core evaluation time with the execution time to visualize the system overhead.

5.2 Scale-out Experiment

In this experiment, we investigated whether the system scales with the growing size of the cluster and input workload. We started with 2 nodes processing 160, 000 records and increased that by 2, up to 8 nodes processing 640, 000 records. Fig. 23 plots the overall execution time and per core evaluation time against the number of nodes in the cluster. As indicated in the figure, the system shows good scale-out performance. The system overhead increased a bit as the size of cluster grew, but the margin between the execution time and the per-core evaluation time was relatively small compared to the execution time.

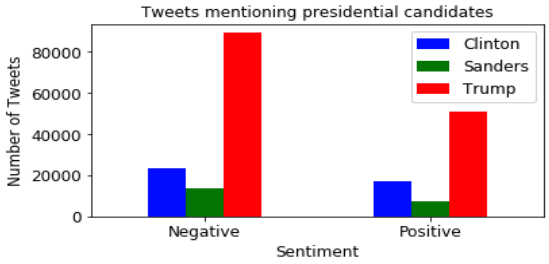
5.3 Speed-up Experiment

In this experiment, we looked at how the system speed-up with a fixed amount of workload and an increasing cluster size. We fed the system with 400, 000 records and measured the execution time on 2, 4, 6, and 8 nodes. The result shows that the system speed up accordingly with more nodes joining the evaluation.

```
In [3]: trump = send_request("FROM Tweets t WHERE contains(t.text,\"Trump\")\
                  GROUP BY sklib#getSentiment(t.text) sentiment SELECT sentiment, COUNT(*) as cnt;")
        sanders = send_request("FROM Tweets t WHERE contains(t.text,\"Sanders\")\
                  GROUP BY sklib#getSentiment(t.text) sentiment SELECT sentiment, COUNT(*) as cnt;")
        clinton = send_request("FROM Tweets t WHERE contains(t.text,\"Clinton\")\
                  GROUP BY sklib#getSentiment(t.text) sentiment SELECT sentiment, COUNT(*) as cnt;")

In [4]: merged=pd.DataFrame({'Clinton':clinton['cnt'], 'Sanders':sanders['cnt'],
                     'Trump':trump['cnt']}).rename({0: 'Negative', 1: 'Positive'}, axis='index')

In [5]: ax = merged.plot(kind='bar',title ="Tweets mentioning presidential candidates")
        ax.set_xlabel("Sentiment", fontsize=12)
        ax.set_ylabel("Number of Tweets", fontsize=12)
        plt.show()
        print(merged)
```



	Clinton	Sanders	Trump
Negative	23646	13868	89115
Positive	17092	7315	51153

Figure 18: A sample Jupyter notebook that calls a Scikit-Learn trained model through HTTP API.

```
CREATE FUNCTION annotation(t) {
object_merge({"sentiment":
snlplib#getSentimentScore(t.text)}, t)};

CREATE FEED TwitterFileFeed USING localfs
(("type-name"="Tweet"),
("path"="127.0.0.1:///PATH_TO_THE_DATA_FILE"),
("format"="adm"));
CONNECT FEED TwitterFileFeed TO DATASET
Tweets apply function annotation;
```

Figure 19: Adding an external UDF to a feed.

```
SELECT snlplib#getSentiment(t) FROM Tweets t
WHERE t.id > 2;
```

Figure 20: Evaluating an external UDF on-the-fly.

6 CONCLUSION

In this paper, we have explained how machine learning can be used in conjunction with Apache AsterixDB for Big Data analytics. We introduced the data model, user-defined functions, and data feeds in AsterixDB, and explained how they can be used together with off-the-shelf machine learning libraries to provide a full end-to-end analytic experience for data analysts using their preferred machine learning tools. We also explained how to export data from AsterixDB for machine learning model training using SQL++ with its query service or Spark connector, and discussed how to train models using Weka, Spark MLlib, and scikit-learn. For applying models for data analytics, we further discussed how to extract specific information using a specialized library, e.g., Stanford CoreNLP, or to use general libraries, e.g., Weka and Spark, with trained model files. With this AsterixDB support for the loading-training-prediction life cycle in data analytics, we can provide data analysts with an integrated distributed Big Data analytics platform with customizable dataflows.

Going forward, we plan to further extend the AsterixDB support for end-to-end machine learning data analytics with the following features:

Visualization of Semantic Data. Cloudberry [11] is a distributed analytics middleware solution built on top of AsterixDB. It provides interactive data analysis on temporal, spatial, and textual dimensions. To extend its ability in visualizing data, we plan to integrate data analysis via UDFs into its query interface. This improvement will make it possible to visualize things such as the sentiment distribution of tweets about certain topics in the U.S.

Integration with Advanced Machine Learning Architectures. Modern machine learning libraries are provided in different programming languages, e.g., Python (TensorFlow), Scala (Spark MLlib), and C++ (Shogun), as well as on various computational devices, such as multi-core, GPU, and TPU hardware. We want to add support for libraries from additional languages and to execute them efficiently on different computing resources.

Feed and Query Optimization with Analytic Functions. Due to the computational complexity of analytic functions, having functions in a dataflow can slow down the whole pipeline and introduce system bottlenecks, either on the ingestion side or query-evaluation side. We plan to study how to incorporate further parallelism for expensive function evaluation in the ingestion pipeline and how to best optimize queries containing such expensive functions.

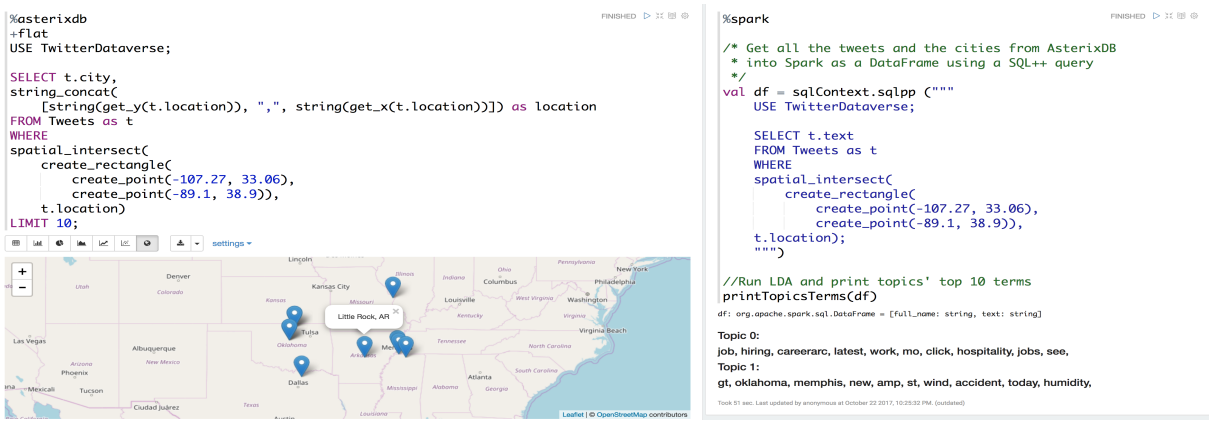


Figure 21: Using Apache Zeppelin to run a SQL++ query on AsterixDB and Spark using Scala.

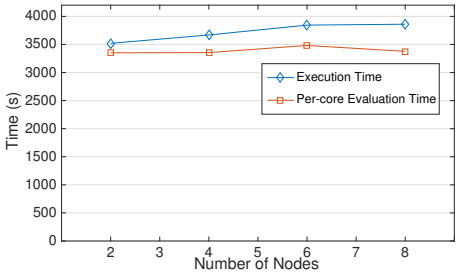


Figure 22: Scale-out experiment with $2, 4, 6, 8$ **nodes processing** $160k$**,** $320k$**,** $480k$**, and** $640k$ **records respectively.**

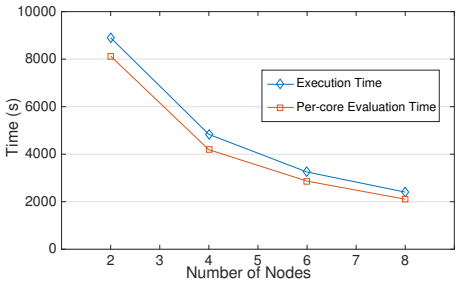


Figure 23: Speed-up experiment with $400,000$ **records on** $2, 4, 6, 8$ **nodes.**

ACKNOWLEDGMENTS

The work reported in this paper was supported in part by NSF CNS award 1305430.

REFERENCES

[1] Appendix b - the WEKA workbench. In I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, editors, *Data Mining: Practical machine learning tools and techniques*, pages 553–571. Morgan Kaufmann, fourth edition edition, 2017.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[3] W. Y. Alkowaileet, S. Alsubaiee, M. J. Carey, T. Westmann, and Y. Bu. Large-scale complex analytics on semi-structured datasets using AsterixDB and Spark. *Proceedings of the VLDB Endowment*, 9(13):1585–1588, 2016.

[4] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. Borkar, Y. Bu, M. Carey, I. Cetindil, M. Cheelangi, K. Faraaz, et al. AsterixDB: A scalable, open source BDMS. *Proceedings of the VLDB Endowment*, 7(14):1905–1916, 2014.

[5] Apache Zeppelin. http://zeppelin.apache.org, 2013.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[7] D. Donoho. 50 years of Data Science. In *Princeton NJ, Tukey Centennial Workshop*, 2015.

[8] R. Grover and M. J. Carey. Data ingestion in AsterixDB. In *Proceedings of the 18th International Conference on Extending Database Technology (EDBT 2015)*, pages 605–616, 2015.

[9] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, et al. The MADlib analytics library: or MAD skills, the SQL. *Proceedings of the VLDB Endowment*, 5(12):1700–1711, 2012.

[10] Jep. Java Embedded Python. https://github.com/ninia/jep, 2013.

[11] J. Jia, C. Li, X. Zhang, C. Li, M. J. Carey, et al. Towards interactive analytics and visualization on one billion tweets. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 85. ACM, 2016.

[12] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[13] D. Kotzias, M. Denil, N. De Freitas, and P. Smyth. From group to individual labels using deep features. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 597–606. ACM, 2015.

[14] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[15] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[16] S. I. F. G. N. Nes and S. M. S. M. M. Kersten. Monetdb: Two decades of research in column-oriented database architectures. *Data Engineering*, 40, 2012.

[17] K. W. Ong, Y. Papakonstantinou, and R. Vernoux. The SQL++ query language: Configurable, unifying and semi-structured. *arXiv preprint arXiv:1405.3631*, 2014.

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

[19] Redis. https://redis.io, 2009.

[20] A. Rheinländer, U. Leser, and G. Graefe. Optimization of complex dataflows with user-defined functions. *ACM Computing Surveys (CSUR)*, 50(3):38, 2017.

[21] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.