KeystoneML

VASANTH METHKUPALLI[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: mvasanthiiit@gmail.com

Paper-2, April 3, 2017

KeystoneML is a software framework, written in Scala, from the UC Berkeley AMPLab designed to simplify the construction of large scale, end-to-end, machine learning pipelines with Apache Spark. KeystoneMl and spark.ml share many features, but however there are a few important differences, particularly around type safety and chaining, which lead to pipelines that are easier to construct and more robust. KeystoneML also presents a richer set of operators than those present in spark.ml including featurizers for images, text, and speech, and provides several example pipelines that reproduce state-of-the-art academic results on public data sets.

Keywords: Cloud, I524, KeystoneML, MLlib, API, Spark

https://github.com/cloudmesh/sp17-i524/blob/master/paper2/S17-IR-2019/report.pdf

This review document is provided for you to achieve your best. We have listed a number of obvious opportunities for improvement. When improving it, please keep this copy untouched and instead focus on improving report.tex. The review does not include all possible improvement suggestions and if you sea comment you may want to check if this comment applies elsewhere in the document.

1. INTRODUCTION

MLlib's goal is to make practical machine learning (ML) scalable and easy. Besides new algorithms and performance improvements that are seen in each release, a great deal of time and effort has been spent on making MLlib easy. Similar to Spark Core, MLlib provides APIs in three languages: Python, Java, and Scala, along with user guide and example code, to ease the learning curve for users coming from different backgrounds. In Apache Spark 1.2, Databricks, jointly with AMPLab, UC Berkeley, continues this effort by introducing a pipeline API to MLlib for easy creation and tuning of practical ML pipelines [1] .

A practical ML pipeline often involves a sequence of data pre-processing, feature extraction, model fitting, and validation stages. For example, classifying text documents might involve text segmentation and cleaning, extracting features, and training a classification model with cross-validation. Though there are many libraries we can use for each stage, working with large-scale datasets is not easy as it looks. Most ML libraries are not designed for distributed computation or they do not provide native support for pipeline creation and tuning. Unfortunately, this problem is often ignored in academia, and it has received largely ad-hoc treatment in industry, where development tends to occur in manual one-off pipeline implementations [1] .

2. DESIGN PRINCIPLES

KeystoneML is built on several design principles: supporting end-to-end workflows, type safety, horizontal scalability, and composibility. By focusing on these principles, KeystoneML allows for the construction of complete, robust, large scale pipelines that are constructed from reusable, understandable parts [2] .

3. KEY API CONCEPTS

At the center of KeystoneML are a handful of core API concepts that allow us to build complex machine learning pipelines out of simple parts:

- Pipelines
- Nodes
- Transformers
- Estimators

4. PIPELINES

A Pipeline is a dataflow that takes some input data and maps it to some output data through a series of nodes. By design, these nodes can operate on one data item (for point lookup) or many data items: for batch model evaluation [2] .

In a sense, a pipeline is just a function that is composed of simpler functions. Here's part of the Pipeline definition:

From this we can see that a Pipeline has two type parameters: its input and output types. We can also see that it has methods to operate on just a single input data item, or on a batch RDD of data items [2] .

```
package workflow

trait Pipeline[A, B] {
  // ...
  def apply(in: A): B
  def apply(in: RDD[A]): RDD[B]
  // ...
}
```

Fig. 1. Pipeline Definition

5. NODES

Nodes come in two flavors:

- Transformers
- Estimators

Transformers are nodes which provide a unary function interface for both single items and RDD of the same type of item, while an Estimator produces a Transformer based on some training data.

5.1. Transformers

As already mentioned, a Transformer is the simplest type of node, and takes an input, and deterministically transforms it into an output. Here's an abridged definition of the Transformer class.

```
package workflow

abstract class Transformer[A, B : ClassTag] extends TransformerNode[B] with Pipeline[A, B] {
  def apply(in: A): B
  def apply(in: RDD[A]): RDD[B] = in.map(apply)
  //...
}
```

Fig. 2. Transformer Class

While transformers are unary functions, they themselves may be parameterized by more than just their input. To handle this case, transformers can take additional state as constructor parameters. Here's a simple transformer which will add a fixed vector from any vector it is fed as input [2] .

```
import pipelines.Transformer
import breeze.linalg._

class Adder(vec: Vector[Double]) extends Transformer[Vector[Double], Vector[Double]] {
  def apply(in: Vector[Double]): Vector[Double] = in + vec
}
```

Fig. 3. Transformer Class-Additional states

5.2. Estimators

Estimators are what puts the ML in KeystoneML. An abridged Estimator interface looks like this:

Estimator takes in training data as an RDD to its fit() method, and outputs a Transformer.Suppose you have a big list of vectors and you want to subtract off the mean of each coordinate across all the vectors (and new ones that come from the same distribution). We could create an Estimator to do this like so.

6. CHAINING NODES AND BUILDING PIPELINES

Pipelines are created by chaining transformers and estimators with the andThen methods. Going back to a different part of the Transformer interface:

Ignoring the implementation, andThen allows you to take a pipeline and add another onto it, yielding a new Pipeline[A,C] which works by first applying the first pipeline (A => B) and then applying the next pipeline (B => C).

```
package workflow

abstract class Estimator[A, B] extends EstimatorNode {
  protected def fit(data: RDD[A]): Transformer[A, B]
  // ...
}
```

Fig. 4. Estimator Interface

This is where type safety comes in to ensure robustness. As your pipelines get more complicated, you may end up trying to chain together nodes that are incompatible, but the compiler won't let you. This is powerful, because it means that if your pipeline compiles, it is more likely to work when you go to run it at scale [3] .

Estimators can be chained onto transformers via the andThen (estimator, data) or andThen (labelEstimator, data, labels) methods. The latter makes sense if you're training a supervised learning model which needs ground truth training labels. Suppose you want to chain together a pipeline which takes a raw image, converts it to grayscale, and then fits a linear model on the pixel space, and returns the most likely class according to the linear model [3] .

7. WHY KEYSTONEML?

KeystoneML makes constructing even complicated machine learning pipelines easy. Here's an example text categorization pipeline which creates bigram features and creates a Naive Bayes model based on the 100,000 most common features [2] .

```
val trainData = NewsGroupsDataLoader(sc, trainingDir)

val predictor = Trim andThen
    LowerCase() andThen
    Tokenizer() andThen
    NGramsFeaturizer(1 to conf.nGrams) andThen
    TermFrequency(x => 1) andThen
    (CommonSparseFeatures(conf.commonFeatures), trainData.data) andThen
    (NaiveBayesEstimator(numClasses), trainData.data, trainData.labels) andThen
    MaxClassifier
```

Fig. 5. Code for Naive Bayes model

Parallelization of the pipeline fitting process is handled automatically and pipeline nodes are designed to scale horizontally. Once the pipeline has been defined you can apply it to test data and evaluate its effectiveness [3] .

```
val test = NewsGroupsDataLoader(sc, testingDir)
val predictions = predictor(test.data)
val eval = MulticlassClassifierEvaluator(predictions, test.labels, numClasses)

println(eval.summary(newsgroupsData.classes))
```

Fig. 6. Code for testing the data for effectiveness

The result of this code is as follows:

This relatively simple pipeline predicts the right document category over 80 percent of the time on the test set. KeystoneML works with much more than just text. KeystoneML is alpha software, in a very early public release (v0.2). The project is still very young, but it has reached a point where it is viable for general use.

8. LINKING

KeystoneML is available from Maven Central. It can be used in our applications by adding the following lines to the SBT project definition:

libraryDependencies += "edu.berkeley.cs.amplab"

```
Avg Accuracy:   0.980
Macro Precision:0.816
Macro Recall:   0.797
Macro F1:       0.797
Total Accuracy: 0.804
Micro Precision:0.804
Micro Recall:   0.804
Micro F1:       0.804
```

Fig. 7. Output for the above code

9. BUILDING

KeystineML is available on GitHub.

$ git clone https://github.com/amplab/keystone.git

Once downloaded, KeystoneML can be built using the following commands:

$ cd keystone

$ git checkout branch-v0.3

$ sbt/sbt assembly

$ make

You can then run example pipelines with the included bin/run-pipeline.sh script, or pass as an argument to spark-submit.

10. RUNNING AN EXAMPLE

Once you've built KeystoneML, you can run many of the example pipelines locally. However, to run the larger examples, you'll want access to a Spark cluster.

Here's an example of running a handwriting recognition pipeline on the popular MNIST dataset. This should be able to run on a single machine in under a minute.

```
#Get the data from S3
wget http://mnist-data.s3.amazonaws.com/train-mnist-dense-with-labels.data
wget http://mnist-data.s3.amazonaws.com/test-mnist-dense-with-labels.data

KEYSTONE_MEM=4g ./bin/run-pipeline.sh \
  pipelines.images.mnist.MnistRandomFFT \
  --trainLocation ./train-mnist-dense-with-labels.data \
  --testLocation ./test-mnist-dense-with-labels.data \
  --numFFTs 4 \
  --blockSize 2048
```

Fig. 8. Example for running on a MNIST dataset

To run on a cluster, it is recommend using the spark-ec2 to launch a cluster and provision with correct versions of BLAS and native C libraries used by KeystoneML.

More scripts have been provided to set up a well-configured cluster automatically in bin/pipelines-ec2.sh.

11. CONCLUSION

One of the main features of KeystoneML is the example pipelines and nodes it provides out of the box. These are designed to illustrate end-to-end real world pipelines in computer vision, speech recognition, and natural language processing. KeystoneML also provides several utilities for evaluating models once they've been trained. Computing metrics like precision, recall, and accuracy on a test set. Metrics are currently calculated for Binary Classification, Multiclass Classification, and Multilabel Classification, with more on the way [3] .

REFERENCES

[1] 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," *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, 2016.

[2] "KeystoneML." [Online]. Available: http://keystone-ml.org/programming_guide.html

[3] "KeystoneML." [Online]. Available: http://keystone-ml.org/