Association for
Computing Machinery

Advancing Computing as a Science & Profession

Austin SIGKDD Spark talk

Random Forest and Decision Trees in Spark MLlib

Tuhin Mahmud
Sep 20th, 2017

Agenda

- Spark and MLLib Overview
- Decision Trees in Spark MlLib
- Random Forest in Spark MlLib
- Demo

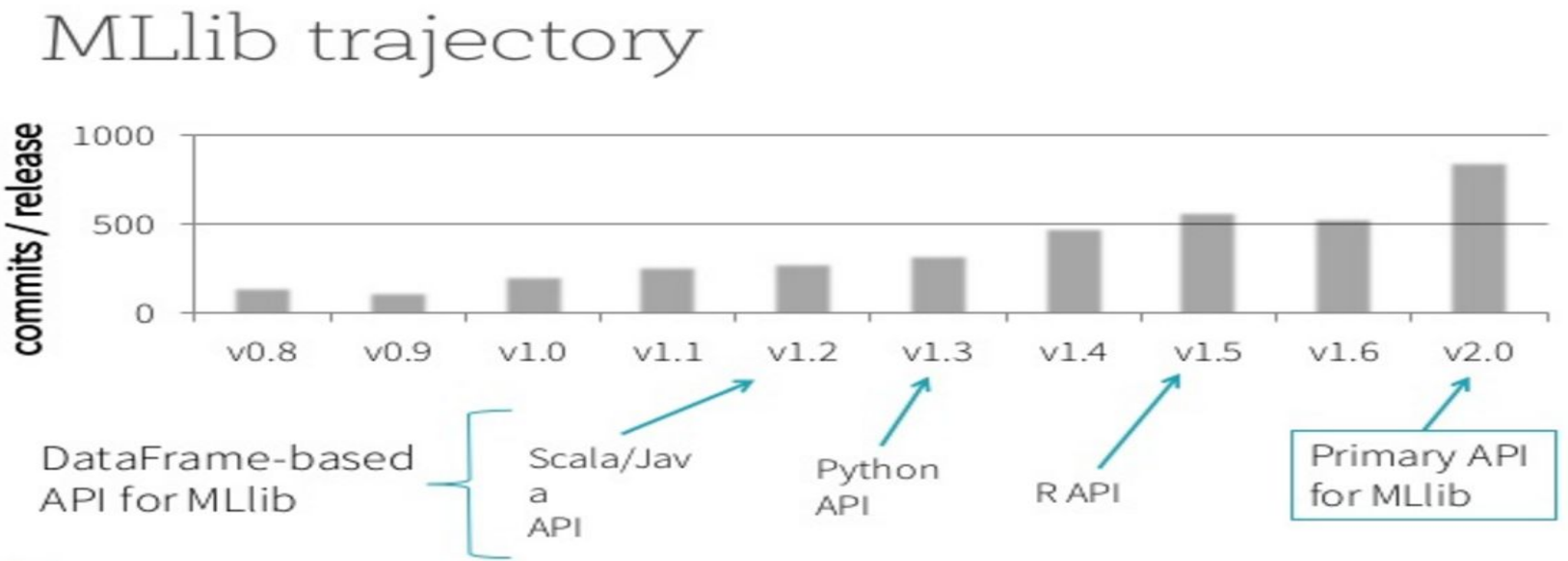
What is MLlib?

- MLlib is Spark's machine learning (ML) library.
- Its goal is to make practical machine learning scalable and easy.
- ML algorithms include Classification , Regression , Decision Trees and random Forests, Recommendation, Clustering,Topic Modeling (LDA), Distributed linear Algebra(SVD,PCA) and many more.

https://spark.apache.org/docs/latest/ml-guide.html

Spark MLlib

Apache Spark MLlib 2.0 Preview: Data Scien... - *Databricks*



https://www.slideshare.net/databricks/apache-spark-mllib-20-preview-data-science-and-production

Spark MlLib

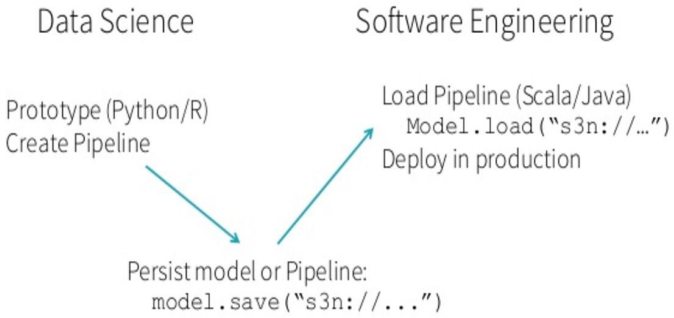
- Initially developed by MLbase team in AmpLab (UC berkeley) - supported only Scala and Java
- Shipped to spark v0.8 (Sep 2013)
- Current release Spark 2.2.0 released (Jul 11, 2017)
- MLLib 2.2:
  - **DataFrame-based Api** becomes the primary API for MLlib
    - org.apache.spark.ml and pysprak.ml
  - Ease of use
    - Java, Scala, **Python, R**
    - interoperates with NumPy in Python
    - Any Hadoop data source (e.g. HDFS, HBase, or local files), making it easy to plug into **Hadoop workflows.**

Spark Mllib in production

- ML persistence
  - Saving and loading
- **Pipeline** MLlib standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow. This section covers the key concepts introduced by the Pipelines API, where the pipeline concept is mostly inspired by the scikit-learn project.

Some concepts[5]

- **DataFrame**
- **Transformer** - one dataframe to another
- **Estimator** - fits on a dataframe to produce a transformer
- **Pipeline** A pipeline chains multiple transformers and Estimators to specif
- **Parameter**



MLlib 2.2.0 API

1. **MLlib: RDD-based API (maintanace mode)**
2. **MLlib: DataFrame-based API**

Why is MLlib switching to the DataFrame-based API?

- **DataFrames** provide a more **user-friendly API** than RDDs.
  a. The many benefits of DataFrames **include Spark Datasources,**
  b. **SQL/DataFrame queries**, Tungsten and Catalyst optimizations,
  c. uniform APIs across languages.
- The DataFrame-based API for MLlib provides a uniform API across ML algorithms and across multiple languages.
- DataFrames facilitate practical **ML Pipelines,** particularly feature transformations. See the Pipelines guide for details.

What is "Spark ML"?

- "Spark ML" is **not an official name** but occasionally used to refer to the MLlib DataFrame-based API.

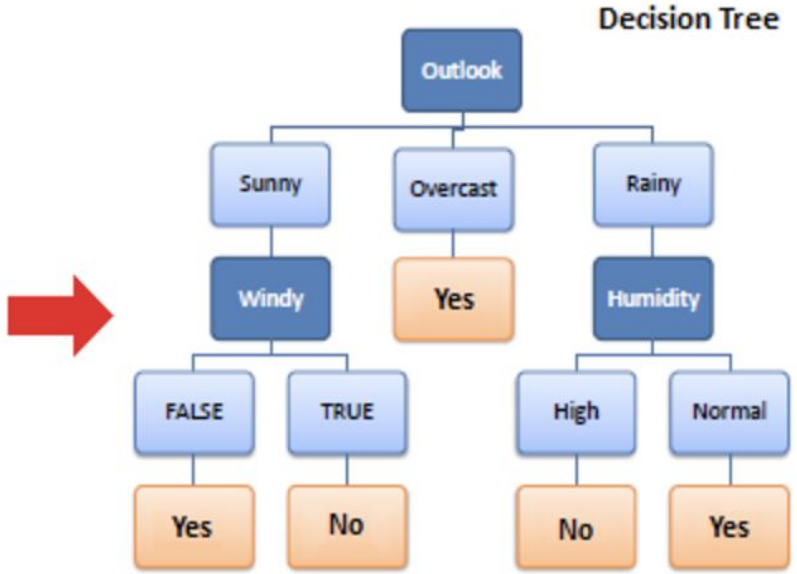
https://spark.apache.org/docs/2.2.0/ml-guide.html

Agenda

- Spark and MLLib Overview
- Decision Tree in Spark MlLib
- Random Forest in Spark MlLib
- Demo

Decision Tree

Predictors				Target
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



http://www.saedsayad.com/decision_tree.htm

DecisionTrees packages in MLlib

1. from pyspark.**mllib**.tree import DecisionTree, DecisionTreeModel (RDD based)
2. from **pyspark.ml.**classification import DecisionTreeClassifier (Dataframe based)

https://spark.apache.org/docs/2.2.0/ml-classification-regression.html#decision-trees

DecisionTree Classifier (MLlib Dataframe based API)

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> dt = DecisionTreeClassifier(maxDepth=2, labelCol="indexed")
>>> model = dt.fit(td)
>>> model.numNodes
```

https://spark.apache.org/docs/2.2.0/api/python/pyspark.ml.html#pyspark.ml.classification.DecisionTreeClassifier

Decison Tree

hyper-parameters for decision tree in MlLib

- **numClasses:** How many classes are we trying to classify?
- **categoricalFeaturesInfo:** A specification whereby we declare what features are **categorical features** and should not be treated as numbers
- **impurity:** A measure of the homogeneity of the labels at the node. Currently in Spark, there are two measures of impurity with respect to classification: **Gini** and **Entropy**
- **maxDepth:** A s**topping criterion** which limits the depth of constructed trees. Generally, deeper trees lead to more accurate results but run the risk of overfitting.
- **maxBins:** Generally, increasing the number of bins allows the tree to consider more values but also increases computation time.

$$Entropy = \sum_{j} -p_j \log_2 p_j$$

Where p_j is the frequency of label j at a node.

The **Gini Index**: is a measure of how often a randomly chosen element would be misclassified if it were randomly given a label according to the distribution of labels at a given node.

$$Gini\ Index = 1 - \sum_{j} p_j^2$$

Agenda

- Spark and MLLib Overview
- Decision Tree in Spark MlLib
- Random Forest in Spark MlLib
- Demo

Random Forest

A random forest (RF) is a collection of tree predictors

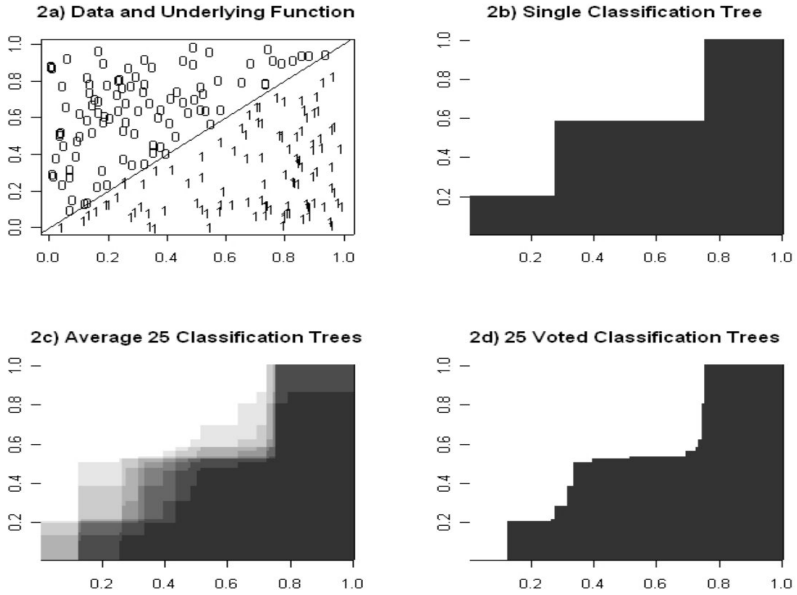
$f (x, T, \Theta k), k = 1,\ldots,K$

where the Θk are i.i.d random vectors.

The trees are combined by

- voting (for classification)
- averaging (for regression).

Random Forest



http://www.math.usu.edu/adele/RandomForests/ENAR.pdf

Random Forests - algorithm

Grow each tree on an **independent bootstrap sample** from the training data.

At each node:

1. Randomly select m variables out of all M possible variables (independently for each node).
2. Find the best split on the selected m variables.

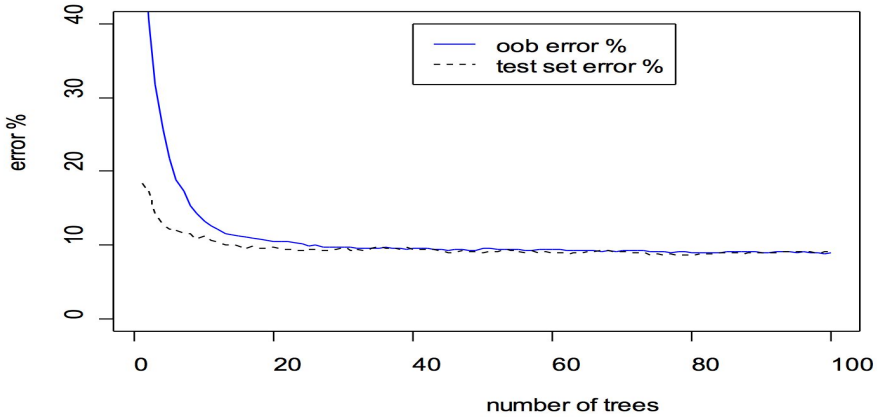
Grow the tree to maximum depth.

Vote or average the trees to get predictions.

Random Forest - Out of bag Error

Out-of-bag Error Estimate

- Average over the cases within each class to get a classwise out-of-bag error rate.
- Average over all cases to get an overall out-of-bag error rate.



RandomForest in MLLib

DataFrame Based API

class **pyspark.ml.**classification.**RandomForestClassifier**(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, ***maxDepth=5***, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity="gini"*, ***numTrees=20***, *featureSubsetStrategy="auto"*, *seed=None*, *subsamplingRate=1.0*)[source]¶

```
>>> import numpy
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> rf = RandomForestClassifier(numTrees=3, maxDepth=2, labelCol="indexed", seed=42)
>>> model = rf.fit(td)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
```

https://spark.apache.org/docs/2.2.0/ml-classification-regression.html#random-forest-classifier

Random Forest

RandomForest - some parameters (older RDD based)

- **numTrees:** Number of trees in the resulting forest. I**ncreasing the number of trees decreases model variance**.
- **featureSubsetStrategy:** Specifies a method which produces a number of how many features are selected for training a single tree.
- **seed:** Seed for random generator initialization, since RandomForest depends on random selection of features and rows

The parameters **numTrees** and **maxDepth** are often referenced as **stopping criteria**.

Random Forest

Spark also provides additional parameters to **stop tree growing** and produce fine-grained trees:

- **minInstancesPerNode:** A node is not split anymore, if it would provide left or right nodes which would contain smaller number of observations than the value specified by this parameter. Default value is 1, but typically for regression problems or large trees, the value should be higher.
- **minInfoGain:** Minimum information gain a split must get. Default value is 0.0.

Sample Random Forest code

class pyspark.ml.classification.**RandomForestClassifier**(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity="gini"*, *numTrees=20*, *featureSubsetStrategy="auto"*, *seed=None*, *subsamplingRate=1.0*)[source]

```
>>> import numpy
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> rf = RandomForestClassifier(numTrees=3, maxDepth=2, labelCol="indexed", seed=42)
>>> model = rf.fit(td)
>>> model.featureImportances
```

MLlib - Labeled point vector (RDD based)

Labeled point vector

- Prior to running any supervised machine learning algorithm using Spark MLlib, we must **convert our dataset into a labeled point vector.**
  - val higgs = response.zip(features).map {
    case (response, features) =>
    LabeledPoint(response, features) }
    higgs.setName("higgs").cache()

- An example of a labeled point vector follows:

  (1.0, [0.123, 0.456, 0.567, 0.678, ..., 0.789])

MLlib - data caching

Data caching

Many machine learning algorithms are iterative in nature and thus require multiple passes over the data.

Spark provides a way to persist the data in case we need to iterate over it. Spark also publishes several StorageLevels to allow storing data with various options:

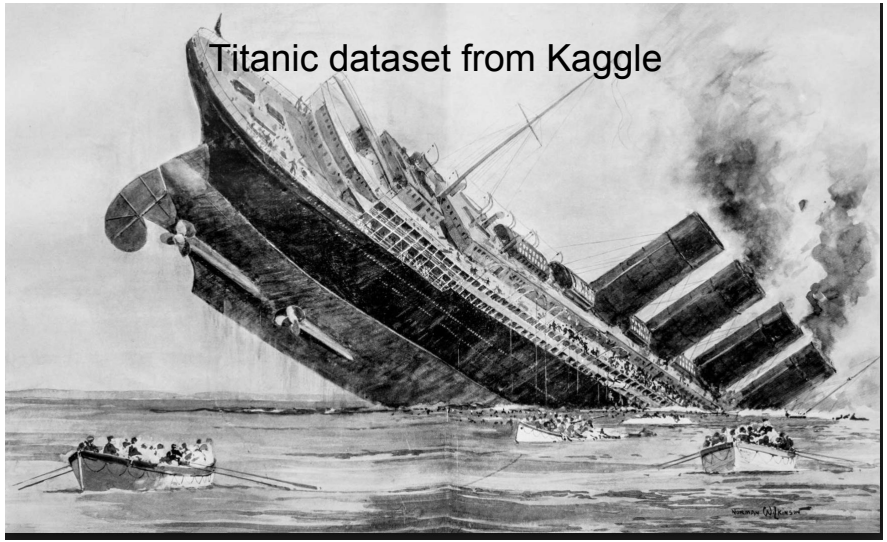
- NONE: No caching at all
- MEMORY_ONLY: Caches RDD data only in memory
- DISK_ONLY: Write cached RDD data to a disk and releases from memory
- MEMORY_AND_DISK: Caches RDD in memory, if it's not possible to offload data to a disk
- OFF_HEAP: Use external memory storage which is not part of JVM heap

Setting up dataset for Machine Learning

Creating a training and testing set

```
// Create Train & Test Splits
val trainTestSplits = higgs.randomSplit(Array(0.8, 0.2))
val (trainingData, testData) = (trainTestSplits(0), trainTestSplits(1))
```

Demo



https://www.kaggle.com/c/titanic/data

Notebooks

1. Notebook using small dataset golf play
   http://nbviewer.jupyter.org/github/tuhinmahmud/sigkdd_austin/blob/master/SparkMllibPyspark.golf.ipynb
2. Notebook using titanic dataset and MLlib spark dataframe based apis for Decision Tree and Random Forest
   http://nbviewer.jupyter.org/github/tuhinmahmud/sigkdd_austin/blob/master/SparkMlLibTitanicNewDFbasedAPI.ipynb

Reference

- https://www.slideshare.net/databricks/apache-spark-mllib-20-preview-data-science-and-production
- https://spark.apache.org/mllib/
- https://spark.apache.org/docs/latest/ml-guide.html
- http://www.saedsayad.com/decision_tree.htm
- http://www.math.usu.edu/adele/RandomForests/ENAR.pdf
- https://spark.apache.org/docs
- https://spark.apache.org/docs/2.2.0/ml-classification-regression.html#random-forest-classifier
- https://spark.apache.org/docs/2.2.0/api/python/pyspark.ml.html#pyspark.ml.classification.DecisionTreeClassifier