CARLSON SCHOOL OF MANAGEMENT
UNIVERSITY OF MINNESOTA

Scalable Machine Learning with Spark MLlib

MSBA 6330 Prof Liu

Slides credits: Bryan Yang, Joseph Bradley, Xiangrui Deng; Cloudera

Carlson School of Management

Topics

- Introduction to Spark MLlib
- Compare Scalable ML Frameworks
- MLlib Data types and file formats
- Transformer, Estimator, and Pipeline
- MLlib Feature engineering APIs
- Evaluation & Hyper Parameter Tuning
- Appendix: Algorithms in MLlib

Carlson School of Management

Scalable Machine Learning with Spark MLlib

INTRODUCTION TO SPARK MLLIB

Introduction to Spark MLlib

- **Spark MLlib** is Spark's Scalable ML library
  - Contains common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, and underlying optimization primitives.
  - Is a popular choice for large-scale ML.
  - Support Scala, Python, Java, and R APIs
- Spark MLlib consists of two packages
  - org.apache.spark.mllib / pyspark.mllib (older APIs based on RDD)
    - **RDD-based APIs**, expected to be deprecated in Spark 3 (not yet released).
  - org.apache.spark.ml / pyspark.ml (**we will focus on this**)
    - **DataFrame-based APIs**, support pipeline
    - Feature parity with RDD-based APIs estimated for spark 2.3.x (February 2018)

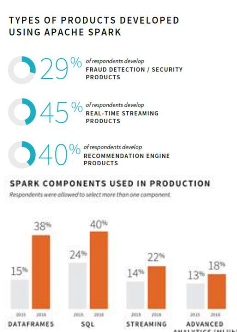
Why DataFrame-based APIs?

- DataFrame-based APIs (pyspark.ml)
  - DataFrames provide more user-friendly APIs than RDDs
  - DataFrame APIs support practical ML pipelines
  - Support DataFrame Datasources (text, csv, json, parquet, image etc)
  - Support SQL/DataFrame queries
  - Tungsten and Catalyst Optimizations
  - Has reached feature parity with older RDD-based APIs.

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

Use Cases of Spark MLlib

- 3M Heath Information Systems
  - Use machine learning to predict patient outcomes
- Customer 360° at Toyota
  - Monitoring customer social media interactions
  - Classify incoming social media interactions into buckets of campaign opinions, customer feedback, product feedback, and noise



Source:Apache Spark Survey 2016
By Databricks

Scalable Machine Learning with Spark MLlib

COMPARE SCALABLE ML FRAMEWORKS

Comparing Scalable ML Frameworks

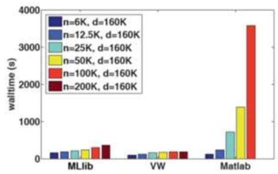
Framework	Description
H2O	Open source machine learning project for distributed machine learning much like Apache Spark. Python and R. Uses specialized data format .Hex
Mahout	Providing a number of java-based distributed ML algorithms for Hadoop. Recently shift to become backend-independent, and compatible with Spark, H2O and Flink.
Turi	(formerly known as GraphLab, purchased by Apple) Graph-based high-performance distributed computation framework written in C++. Topic modeling, clustering graph analytics, CF, computer vision etc.
Vowpal Wabbit (VW)	(originally by Yahoo, now Microsoft Research) Open-source fast out-of-core machine learning library.
Spark MLlib	Part of a general purpose high-performance distributed computing platform. Growing rapidly
SciKit-Learn	Popular single-node ML package but can leverage Spark distributed computing by running multiple models in parallel.

References: Xiangrui Deng's slides: MLLib: Scalable Machine Learning on Spark; additional references in page notes

Why Spark MLlib?

- It is build **on Apache Spark**, a fast and general engine for large-scale data processing.
- Inherits many merits of Spark:
  - **Fast** compared to or even better than other libraries specialized in large-scale machine learning
  - **Simplicity**: a few lines of code can accomplish much.
  - **Multi-language** support: scala, python, R etc
  - **Easy integration** with storage systems
  - **Unified stack**: a special-purpose ML package may be better, but the cost of context switching is high (different language, data formats).

Logistic Regression Performance



Vw: Vowpal Wabbit

- Full dataset: 200K images, 160K dense features.
- Similar weak scaling.
- MLlib within a factor of 2 of VW's wall-clock time.

Credit: Xiangrui Deng's slides: MLLib: Scalable Machine Learning on Spark

Recommender (ALS) – Wall-clock time

System	Wall-clock time (seconds)
Matlab	15443
Mahout	4206
GraphLab	291
MLlib	481

- Dataset: scaled version of Netflix data (9X in size).
- Cluster: 9 machines.
- MLlib is an order of magnitude faster than Mahout.
- MLlib is within factor of 2 of GraphLab.

Credit: Xiangrui Deng's slides: MLLib: Scalable Machine Learning on Spark

Scalable Machine Learning with Spark MLlib

MLLIB DATA TYPES AND FILE FORMATS

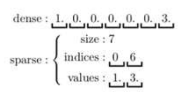
MLlib Data types: Vectors

these are default col names, but you can also other names.

- `label` is represented by a double column
  - for binary classification, it should be either 0.0 or 1.0.
  - For multiclass classification, it should be 0.0, 1.0, 2.0, ...
  - For regression, it should be float or double types
- `features` are represented by a vector column
  - In unsupervised learning, only the features column is needed.

```
+-----+---------+
|label| features|
+-----+---------+
|    1|[0.0,3.0]|
|    0|[1.0,2.0]|
+-----+---------+
```

- Dense vs Sparse Vectors
  - A dense vector is a regular array of doubles
  - A sparse vector is backed by two parallel arrays: one for indicator of elements that are present, and the other for double values of these elements



Create Dense & Sparse Vectors

- Vectors are defined in Spark Mllib's `linalg` module

```python
from pyspark.ml.linalg import Vectors
# Create a dense vector from a Python array
dv = Vectors.dense([1.0, 0.0, 3.0])
# Create a sparse vector using two lists of indices and values or a dict of {pos:value}
sv = Vectors.sparse(3, [0, 2], [1.0, 3.0])
sv = Vectors.sparse(3, {0:1.0, 2:3.0})
```

```scala
import org.apache.spark.ml.linalg.Vectors
val vector = Array(1.0, 0.0, 3.0)
val dv = Vectors.dense(vector)
// sparse vector: length, arrays of indices and values
val sv1 = Vectors.sparse(3, Array(0, 2), Array(1.0, 3.0))
// sparse vector: length, sequence of index:value pairs.
val sv2 = Vectors.sparse(3, Seq((0, 1.0), (2, 3.0)))
```


Why Use Sparse Vectors?

- Not only save storage, but also speed up computation

	Dense	Sparse
Storage	47GB	7GB
Time	240s	58s

Data Set:
- number of cases: 12 million
- number of features: 500
- **sparsity: 10%**

Create ML DataFrames in the Programming Way

- Directly provide the rows; useful for demos and testing
  - define an array of `(features)` or `(label, features)`, where `features` are sparse/dense vectors
  - Convert it to Spark DataFrame

We use both dense and sparse vectors for illustration purposes. You normally have only one kind

```
from pyspark.ml.linalg import Vectors
data = [(1.0, Vectors.sparse(4, [(0, 1.0), (3, -2.0)])),
        (0.0, Vectors.dense([4.0, 5.0, 0.0, 3.0])),
        (1.0, Vectors.dense([6.0, 7.0, 0.0, 8.0])),
        (1.0, Vectors.sparse(4, [(0, 9.0), (3, 1.0)]))]
#convert to DataFrame with col names
df = spark.createDataFrame(data,["label", "features"])
```

```
import org.apache.spark.ml.linalg.Vectors
// define a sequence of tuples
var data = Seq((1.0, Vectors.sparse(4, Seq((0, 1.0), (3, -2.0)))),
        (0.0, Vectors.dense(4.0, 5.0, 0.0, 3.0)),
        (1.0, Vectors.dense(6.0, 7.0, 0.0, 8.0)),
        (1.0, Vectors.sparse(4, Seq((0, 9.0), (3, 1.0)))))
//convert the seq to dataframe with col names
var df = data.toDF("label", "features")
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  1.0|(4,[0,3],[1.0,-2.0])|
|  0.0|   [4.0,5.0,0.0,3.0]|
|  1.0|   [6.0,7.0,0.0,8.0]|
|  1.0| (4,[0,3],[9.0,1.0])|
+-----+--------------------+
```


Load Data from LIBSVM source files

- ***LIBSVM*** is a compact text format for encoding data (usually representing training data sets)
  - Widely used in MLlib to represent sparse feature vectors
  - The layout is as follows:
    - **class_label index1:value1 index2:value2 …**
    - where the numeric indices represent features; values are separated
    - MLlib expects you to start class labeling from 0
    - Feature indices are one-based in ascending order (1,2,3, etc.);
    - If a feature is not present in the record, it is omitted

```
0 2:1 8:1 14:1 21:1 23:1 25:1
0 1:1 9:1 11:1 13:1 18:1 20:1
1 1:1 5:1 15:1 18:1 21:1 23:1
2 6:1 9:1 12:1 14:1 16:1 24:1
2 8:1 13:1 21:1 22:1 27:1 30:1
3 6:1 8:1 10:1 15:1 17:1 21:1
```

```
df = spark.read.format("libsvm").load("mllib/sample_libsvm_data.txt")
```

```
val df = spark.read.format("libsvm").load("mllib/sample_libsvm_data.txt")
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(692,[127,128,129...|
|  1.0|(692,[158,159,160...|
|  1.0|(692,[124,125,126...|
|  1.0|(692,[152,153,154...|
|  1.0|(692,[151,152,153...|
|  0.0|(692,[129,130,131...|
```

Download and Install **spark_sample_data.zip** for examples here

Libsvm: https://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html

Transform an Existing DataFrame

- Spark MLlib provides a `VectorAssembler` API that combines a given list of columns into a single vector column.

```
+---+----+------+--------+-------+----------------+
| id|hour|mobile|userAttr|clicked|        features|
+---+----+------+--------+-------+----------------+
|  0|  18|   1.0|    10.0|    1.0|[18.0,1.0,10.0]|
+---+----+------+--------+-------+----------------+
```

Assembled by a VectorAssembler

- See official document and labs for examples.

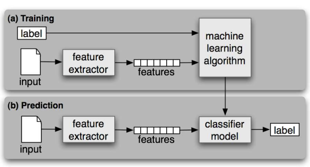
Scalable Machine Learning with Spark MLlib

TRANSFORMER, ESTIMATOR, AND PIPELINE

Carlson School of Management

Overview of DataFrame-Based APIs

- Machine learning tasks consist of a series of steps
- These steps can be viewed as a **pipeline** through which the data travels
- Training and prediction typically follow the same data processing steps.



http://www.nltk.org/book/ch06.html

Carlson School of Management

MLlib's Abstraction of Data, Models, Algorithms, and Pipelines

- `MLlib` aims to standardize interface for machine learning pipelines
- `MLlib` main abstractions are:
  - `DataFrame`: Spark MLlib's dataset type (with a feature vector column)
  - `Transformer`: Transforms one DataFrame into another
  - `Estimator`: Runs an algorithm on a data set to fit a model
  - `Pipeline`: Chains multiple steps to define a machine learning workflow

Transformers

- **Transformers** take a DataFrame as input, and return a new DataFrame
  - Generally append one or more columns to the input DataFrame
  - Abstraction for *feature transformation* and *learned models*
    - A **feature transformer**: e.g., standardize a numerical field.
    - A **learned model**: e.g. read the features column, predict the label for each feature vector, and output a new DataFrame with the predicted label as a new column
- All transformers implement a `transform(df)` method.
- When defining the transformer, one often need to provide
  - **`inputCol`** for the name of the column to be transformed
  - **`outputCol`** for the name of the column for storing the transformed data in the output DataFrame.

Transformer Example

```
from pyspark.ml.feature import Tokenizer

sentenceDataFrame = spark.createDataFrame([
    (0, "Hi I heard about Spark"),
    (1, "I wish Java could use case classes"),
    (2, "Logistic,regression,models,are,neat")
], ["id", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")

tokenized = tokenizer.transform(sentenceDataFrame)
tokenized.select("sentence", "words").show(truncate=False)
```

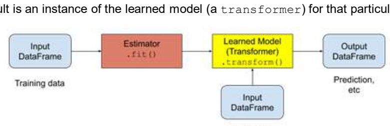
Initialize the tokenizer

Use it to transform data

```
+-----------------------------------+------------------------------------------+
|sentence                           |words                                     |
+-----------------------------------+------------------------------------------+
|Hi I heard about Spark             |[hi, i, heard, about, spark]              |
|I wish Java could use case classes |[i, wish, java, could, use, case, classes]|
|Logistic,regression,models,are,neat|[logistic,regression,models,are,neat]     |
+-----------------------------------+------------------------------------------+
```


Estimators

- Estimators take a DataFrame as input and produce a learned model
  - Learning algorithms are implemented as `Estimators`
  - A learned model is a `Transformer`
- All estimators implement a `fit(df)` method,
  - Estimators are initialized with the specific set of parameters to be used when the algorithm is run
  - `fit` method runs the algorithm on the provided data (`df`)
  - The result is an instance of the learned model (a `transformer`) for that particular algorithm



Estimator Example

- A MinMaxScaler rescales a feature to the [0,1] range.

```
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.linalg import Vectors

dataFrame = spark.createDataFrame([
    (0, Vectors.dense([1.0, 0.1, -1.0]),),
    (1, Vectors.dense([2.0, 1.1, 1.0]),),
    (2, Vectors.dense([3.0, 10.1, 3.0]),)
], ["id", "features"])

scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

# Compute summary statistics and generate MinMaxScalerModel
scalerModel = scaler.fit(dataFrame)

# rescale each feature to range [min, max].
scaledData = scalerModel.transform(dataFrame)
```

```
+--------------+--------------+
|      features|scaledFeatures|
+--------------+--------------+
|[1.0,0.1,-1.0]| [0.0,0.0,0.0]|
| [2.0,1.1,1.0]| [0.5,0.1,0.5]|
|[3.0,10.1,3.0]| [1.0,1.0,1.0]|
+--------------+--------------+
```

training (fit) is required because it needs to know the min/max of each feature

Carlson School of Management

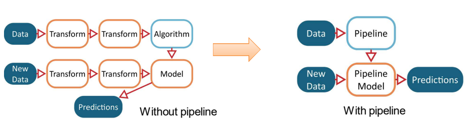
Relationship Between Transformer and Estimator

- Both have the ability to (eventually) transform data
- The key difference lies in whether "*training*" is required.
  - Estimator requires training (using the `fit` method)
    - `fit` produces a `model` (which is a transformer)
    - can be thought of a transformer that requires training before use.
    - E.g., `scaler.fit(df).transform(df)`
  - Transformer does not require training
    - can be directly used
    - E.g., `tokenizer.transform(df)`

Carlson School of Management

Pipelines

- A Pipeline represents a series of steps in a machine learning workflow
  - Each pipeline step can be either a `transformer` or an `estimator`
  - A Pipeline takes a DataFrame as input and produces a `PipelineModel` as output
  - A pipeline itself is therefore an `estimator;` a `PipelineModel` is a `transformer`
- Pipelines simplify the ML workflow:
  - Without pipeline, you must manually carry out each step for both training and prediction. With pipeline, you can define the sequence of steps once and re-used it for both training and prediction.



Without pipeline

With pipeline

Pipeline Example

- Pipeline itself is an estimator, thus you must call its `fit()` method before using it.

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)

# Make predictions on test documents and print columns of interest.
prediction = model.transform(test)
```

* Construction of training and test DataFrames are omitted for brevity

Scalable Machine Learning with Spark MLlib

MLLIB FEATURE ENGINEERING API'S

Supported Feature Engineering APIs

Tranformation	Description
TF-IDF (HashTF, IDF*)	Calculates importance of words in a given body of Text
Word2Vec*	Converts words in a given body of text to vectors to enable numerical calculation of similarity
Tokenizer	Breaking test into individual terms (words)
StopWordsRemover	Remove stop words from a seq of strings
NGram	Takes a seq of strings and computes n-grams
StandardScaler*	Scaling and centering for features to standard deviation one and zero mean
Normalizer	Scales features to have length 1 when viewed as a vector
MinMaxScaler*	Rescaling each feature to a specific range (often [0, 1])
SQLTransformer	Implement transformations defined by SQL statement

* These are estimators (thus require `fit` before use)

Tranformation	Description
Imputer*	completes missing values in a dataset, either using the mean or the median
OneHotEncoderEstimator*	Encodes a single categorical variable into set of binary continuous features, one for each category of the original variable
StringIndexer*	Encodes a string representation of a categorical variable into integer values. Requires fitting to data.
VectorAssembler	Aggregates DataFrame columns into a single column containing a vector.
VectorIndexer*	Class for indexing categorical feature columns in a dataset of vector. Automatically turn a dataset of vectors into one with some continuous features and some categorical features (depending on maxCategories)

Incomplete list, for more please check the documentation.
https://spark.apache.org/docs/latest/ml-features.html

Carlson School of Management

`StringIndexer`: encode (string) categorical features into numerical features

- Many algorithms only take numerical features
  - Need to convert (string) categorical values into numerical values
- `StringIndexer(inputCol, outputCol)`
  - is an **estimator** that encodes categorical features.
  - Operates on a single column of a Spark DataFrame
    - Need a separate `StringIndexer` for each categorical column
  - Requires `fit` before `transform`.
- `IndexToString(inputCol, outputCol)`
  - is a **transformer** that transforms an indexed column back to strings using information in the column's meta data.
  - Implements `.transform()`
  - Reverses `StringIndexer`

`stringIndexer` `indexToString`

stated	state_enc	restored
NY	23.0	NY
CA	6.0	CA
CA	6.0	CA
VA	42.0	VA
TX	36.0	TX
AL	2.0	AL

** StringIndexer by default uses `frequencyDesc` order. e.g., CA has a code of 6 because it is No. 7 ranked by frequency.*

Carlson School of Management

StringIndexer & IndexToString Example

- `StringIndexer` is often used as a stage of a pipeline.

```
+---+--------+-----------+------------+
| id|category|cat_indexed|cat_restored|
+---+--------+-----------+------------+
|  0|       a|        0.0|           a|
|  1|       b|        2.0|           b|
|  2|       c|        1.0|           c|
|  3|       a|        0.0|           a|
|  4|       a|        0.0|           a|
|  5|       c|        1.0|           c|
+---+--------+-----------+------------+
```

```python
from pyspark.ml.feature import StringIndexer, IndexToString
df = spark.createDataFrame(
    [(0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")],
    ["id", "category"])
indexer = StringIndexer(inputCol="category", outputCol="cat_indexed")
df_indexed = indexer.fit(df).transform(df)
index2str = IndexToString(inputCol="cat_indexed", outputCol="cat_restored")
df_index2str = index2str.transform(df_indexed)
```

```scala
import org.apache.spark.ml.feature.{StringIndexer, IndexToString}
val df = spark.createDataFrame(
  Seq((0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c"))
).toDF("id", "category")
val indexer = new StringIndexer().setInputCol("category").setOutputCol("cat_indexed")
val df_indexed = indexer.fit(df).transform(df)
val index2str = new
IndexToString().setInputCol("cat_indexed").setOutputCol("cat_restored")
val df_index2str = index2str.transform(df_indexed)
```

Carlson School of Management

VectorAssembler

- VectorAssembler is a **transformer** that combines a given list of columns into a single vector column.
  - It is useful for combining raw features and features generated by different feature transformers into a single feature vector.
  - VectorAssembler accepts all numeric types, boolean type, and vector type.
  - The values of the input columns will be concatenated into a vector in the given order.

```python
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

dataset = spark.createDataFrame(
    [(0, 18, 1.0, Vectors.dense([0.0, 10.0, 0.5]), 1.0)],
    ["id", "hour", "mobile", "userFeatures", "clicked"])

assembler = VectorAssembler(
    inputCols=["hour", "mobile", "userFeatures"],
    outputCol="features")

output = assembler.transform(dataset)

output.show(truncate=False)
```

```
+---+----+------+--------------+-------+-----------------------+
|id |hour|mobile|userFeatures  |clicked|features               |
+---+----+------+--------------+-------+-----------------------+
|0  |18  |1.0   |[0.0,10.0,0.5]|1.0    |[18.0,1.0,0.0,10.0,0.5]|
+---+----+------+--------------+-------+-----------------------+
```

OneHotEncoderEstimator

- `OneHotEncoderEstimator` maps a categorical feature, represented as a label index, to a binary vector.
  - This encoding allows algorithms which expect continuous features, such as Logistic Regression, to use categorical features.
  - For string type input data, it is common to encode categorical features using StringIndexer first.

```
+-------------+-------------+
|categoryIndex|  categoryVec|
+-------------+-------------+
|          0.0|(2,[0],[1.0])|
|          1.0|(2,[1],[1.0])|
|          2.0|    (2,[],[])|
|          0.0|(2,[0],[1.0])|
|          0.0|(2,[0],[1.0])|
|          2.0|    (2,[],[])|
+-------------+-------------+
```

```
from pyspark.ml.feature import OneHotEncoderEstimator
df = spark.createDataFrame([(0.0,), (1.0,), (2.0,), (0.0,), (0.0,), (2.0,)],
    "categoryIndex double")

encoder = OneHotEncoderEstimator(inputCols=["categoryIndex"],outputCols=["categoryVec"])
encoded = encoder.fit(df).transform(df)
```

Carlson School of Management

Scalable Machine Learning with Spark MLlib

EVALUATION & HYPER PARAMETER TUNING

Carlson School of Management

Evaluation and Cross Validation in Spark MLlib

- `ml.evaluation.MulticlassClassificationEvaluator(predictionCol='prediction', labelCol='label', metricName='f1'):`
  - Other metric include: `weightedPrecision|weightedRecall|accuracy`
- `ml.evaluation.BinaryClassificationEvaluator(rawPredictionCol='rawPrediction', labelCol='label', metricName='areaUnderROC')`
  - The other metric is "`areaUnderPR`"
- `ml.tunning.CrossValidator(estimator, estimatorParamMaps, evaluator, numFolds=3)`
  - K-fold cross validation
  - **`CrossValidator`** is also an estimator, which implements the fit method.

```
from pyspark.ml.tuning import ParamGridBuilder
lr = LogisticRegression()
grid = ParamGridBuilder().addGrid(lr.maxIter, [0, 1]).build()
evaluator = BinaryClassificationEvaluator()
cv = CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator)
cvModel = cv.fit(dataset)
evaluator.evaluate(cvModel.transform(dataset))
```

Cross-validation over a grid of parameters is expensive. In this case, it will train 2 (grid size) x 3 (3-fold) = 6 models

Hyper Parameter Tuning: `TrainValidationSplit`

- Cross-validation is a well-established method for choosing parameters which is more statistically sound.
- Spark provides another hyper-parameter tuning estimator called `TrainValidationSplit` that is less expensive but also may not produce reliable results
- `ml.tuning.TrainValidationSplit(estimator, estimatorParamMaps, evaluator, trainRatio=0.75,parallelism=1)`
  - `TrainValidationSplit` only evaluates each combination of parameters once, as opposed to k times in the case of CrossValidation.
  - It does the split of the dataset for you based on the trainRatio.

A Decision Tree Pipeline with Cross Validation and Feature Engineering

```
from pyspark.ml.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

def vectorizeData(data):
    return data.rdd.map(lambda r: [r[-1], Vectors.dense(r[:-1])]).toDF(['label','features'])
vectorized_data = vectorizeData(data)

# Index labels, adding metadata to the label column
labelIndexer = StringIndexer(inputCol='label',outputCol='indexedLabel').fit(vectorized_data)

# Automatically identify categorical features and index them
featureIndexer = VectorIndexer(inputCol='features', outputCol='indexedFeatures',
maxCategories=2).fit(vectorized_data)

# Train a DecisionTree model
dTree = DecisionTreeClassifier(labelCol='indexedLabel', featuresCol='indexedFeatures')

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dTree])
```


A Decision Tree Pipeline with Cross Validation and Feature Engineering (cont.)

```
# Search through decision tree's maxDepth parameter for best model
paramGrid = ParamGridBuilder().addGrid(dTree.maxDepth, [2,3,4]).build()

# Set F-1 score as evaluation metric for best model selection
evaluator = MulticlassClassificationEvaluator(labelCol='indexedLabel', predictionCol='prediction', metricName='f1')

# Set up 3-fold cross validation
crossval = CrossValidator(estimator=pipeline, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=3)

model = crossval.fit(vectorized_data)

# Fetch best model
tree_model = model.bestModel.stages[2]
print tree_model

# prediction on test data
vectorized_test_data = vectorizeData(test_data)

transformed_data = model.transform(vectorized_test_data)
print evaluator.getMetricName(), 'accuracy:', evaluator.evaluate(transformed_data)

predictions = transformed_data.select('indexedLabel', 'prediction', 'probability')
```

Scalable Machine Learning with Spark MLlib

APPENDIX: ALGORITHMS IN MLLIB

Algorithms and Tools in MLlib (pyspark.ml)

Module	Algorithms
.classification	logistic regression, SVM (linearSVC), Naive Bayes, Multilayer Perceptron
.tree	Decision Tree, Random Forest, Gradient Boosted Trees
.regression	linear regression, ridge regression, Lasso Regression, Accelerated Failure Time Survival Regression, Generalized Linear Regression
.recommendation	alternating least squares (ALS)
.clustering	k-means, Bisecting k-means, Gaussian Mixture, LDA, etc
.fpm	FPGrowth (for mining freq itemsets), PrefixSpan (for mining freq sequential patterns)

http://spark.apache.org/docs/latest/api/python/pyspark.ml.html

Algorithms and Tools in MLlib (pyspark.ml)

Module	Algorithms
.linalg	(Dense/Sparse) Vector, (Dense/Sparse) Matrix, etc
.feature	HashTF, IDF, Word2Vec, Normalizer, StandardScaler, ChiSqSelector, PCA, Tokenizer, Binarizer, MinMaxScaler, OneHotEncoder, StringIndexer, VectorIndexer, SQLTransformer, and several more.
.stat	ChiSquareTest, Correlation, Summarizer (2.4), KolmogorovSmirnovTest
.evaluation	BinaryClassificationEvaluator, MulticlassEvaluator, RegressionEvaluator
.param	Param, Params
.tuning	ParamGridBuilder, CrossValidator, TrainValidationSplit