Distributed Computing

DIANE WOODBRIDGE, PH.D

Distributed Computing

Course Objectives

Understand needs and concepts of distributed computing.

Understand the Spark and its stack.

Being competent to work with Spark on a distributed computing environment.

- Programming with RDDs.
- Work with key/value pairs.
- Work with DataFrame.
- Work with SparkSQL, MLlib, ML, Spark Streaming, (and GarphX).
- Work on Amazon AWS.

Spark Interview Questions

What is Apache Spark?

Explain the key features of Spark.

~~What is RDD?~~

~~How to create RDD.~~

~~What is "partitions"?~~

~~Types or RDD operations?~~

~~What is "transformation"?~~

~~What is "action"?~~

~~Functions of "spark core"?~~

~~What is "spark context"?~~

~~What is an "RDD lineage"?~~

~~Which file systems does Spark support?~~

~~List the various types of "Cluster Managers" in Spark.~~

~~What is "YARN"?~~

~~What is "Mesos"?~~

~~What is a "worker node"?~~

~~What is an "accumulator"?~~

~~What is "Spark SQL" (Shark)?~~

What is "SparkStreaming"?

What is "GraphX"?

What is "MLlib"?

https://www.edureka.co/blog/interview-questions/top-apache-spark-interview-questions-2016/

Spark Interview Questions

~~What are the advantages of using Apache Spark over Hadoop MapReduce for big data processing?~~

~~What are the languages supported by Apache Spark for developing big data applications?~~

~~Can you use Spark to access and analyze data stored in Cassandra databases?~~

~~Is it possible to run Apache Spark on Apache Mesos?~~

~~How can you minimize data transfers when working with Spark?~~

~~Why is there a need for broadcast variables?~~

~~Name a few companies that use Apache Spark in production.~~

~~What are the various data sources available in SparkSQL?~~

What is the advantage of a Parquet file?

~~What do you understand by Pair RDD?~~

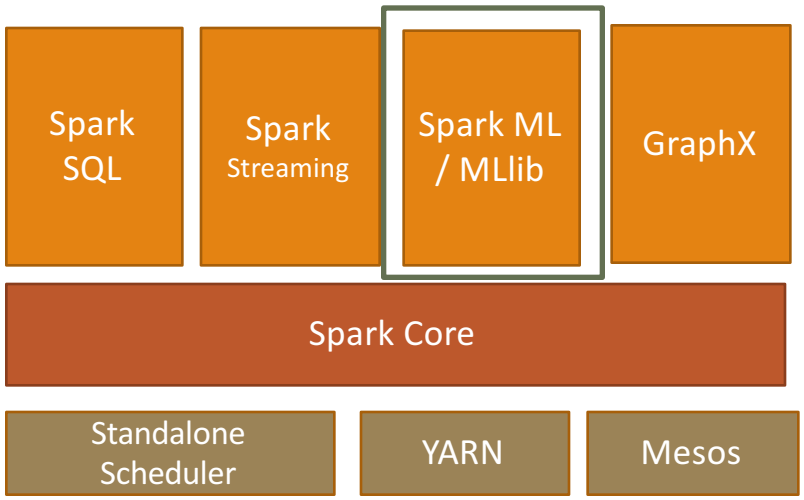
Is Apache Spark a good fit for Reinforcement learning?

https://www.dezyre.com/article/top-50-spark-interview-questions-and-answers-for-2016/208

Machine Learning with Spark

Designed to run in parallel on clusters.

Spark lets you perform machine learning tasks all in the same systems, using the same API.



Machine Learning with Spark

ML

- New (Spark v 1.2).
- Support Pipelines of estimators, transformer and evaluators.
- Use DataFrame.

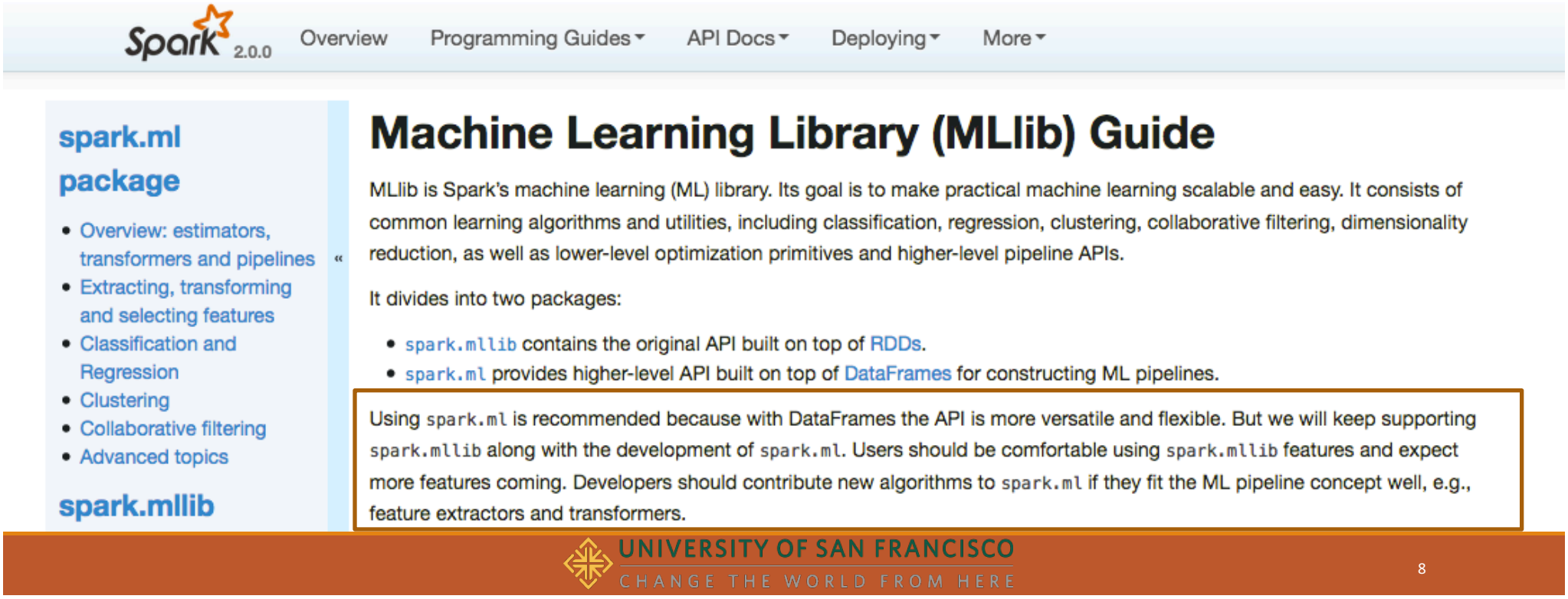
MLlib

- Use RDD.
- Support more features.

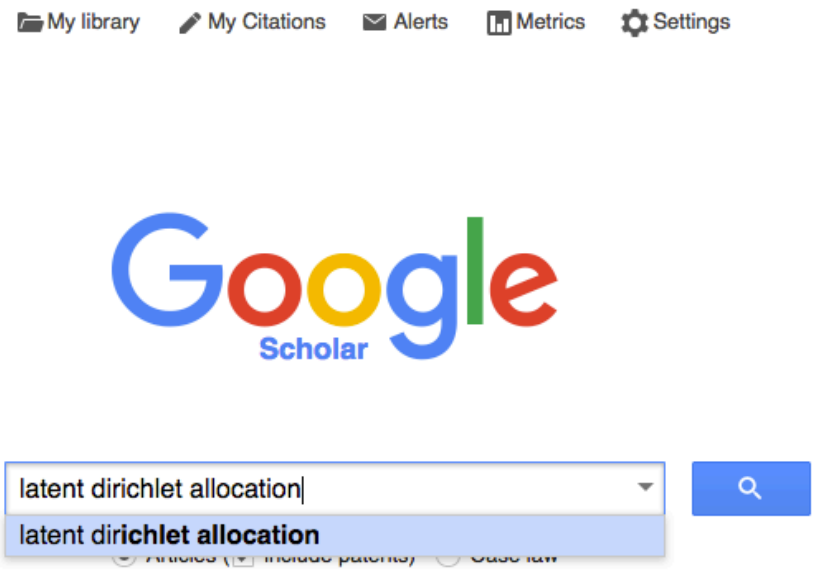
Some of the algorithms are not included because they were not designed for parallel platforms.

Machine Learning with Spark

ML vs MLlib ??

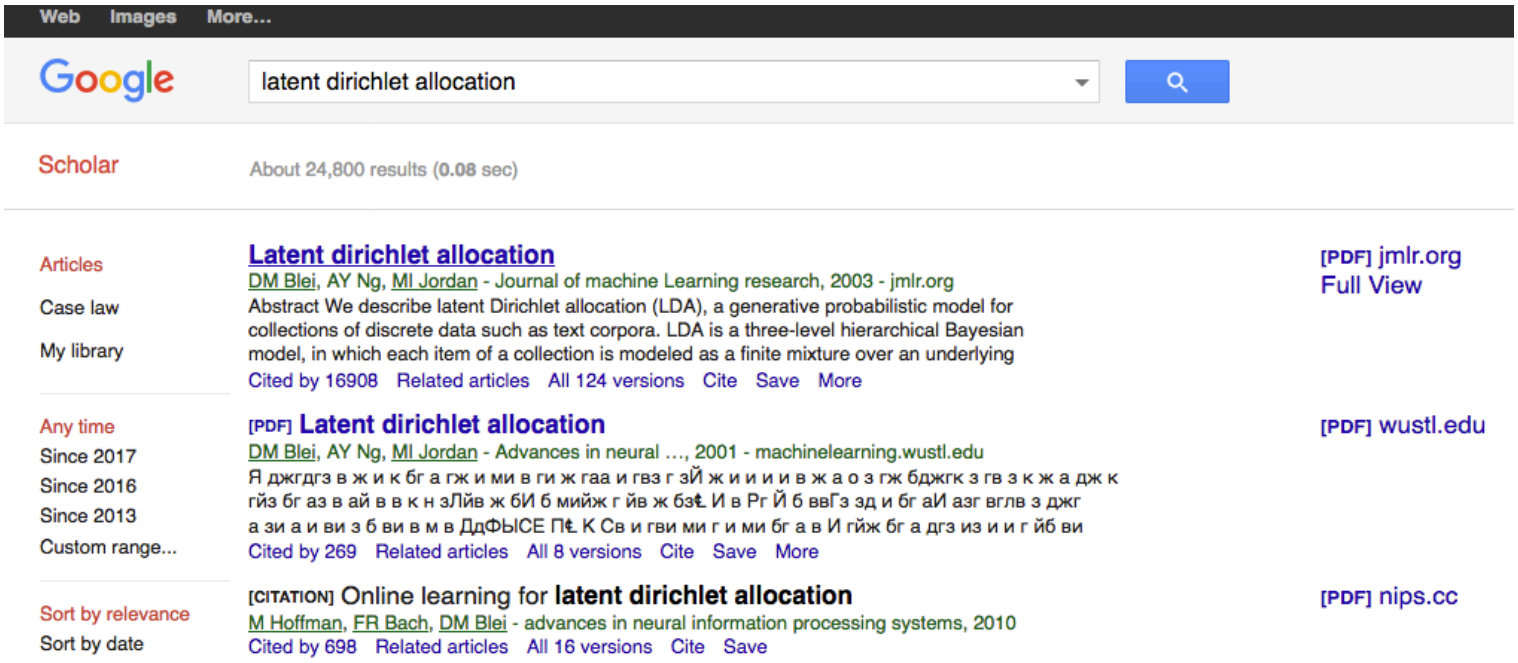


Algorithm Details?



Stand on the shoulders of giants

Algorithm Details?



Spark ML

Why Spark ML?

- MLlib was not scalable and extendable enough.
- Started to develop a new library to generalize machine learning operations and processes.
  ➔Spark ML.

Spark ML

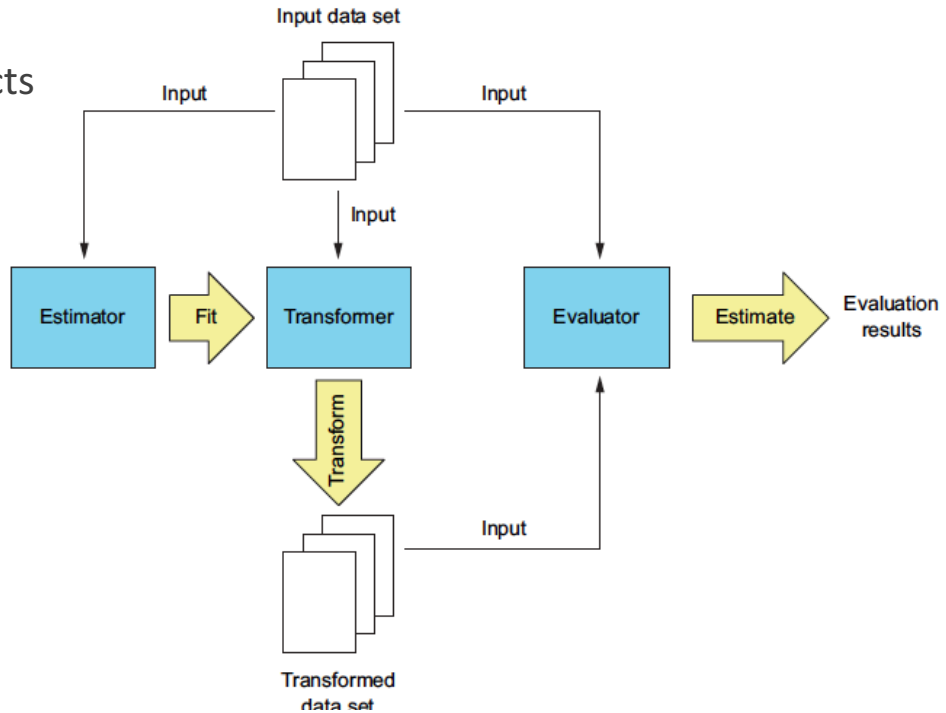
Uses DataFrame and Dataset.

- Dataset : A strongly typed collection of objects (This includes DataFrame.).

Main Components

- Transformers
- Estimators
- Evaluators

- ML Parameters
- ML Pipeline



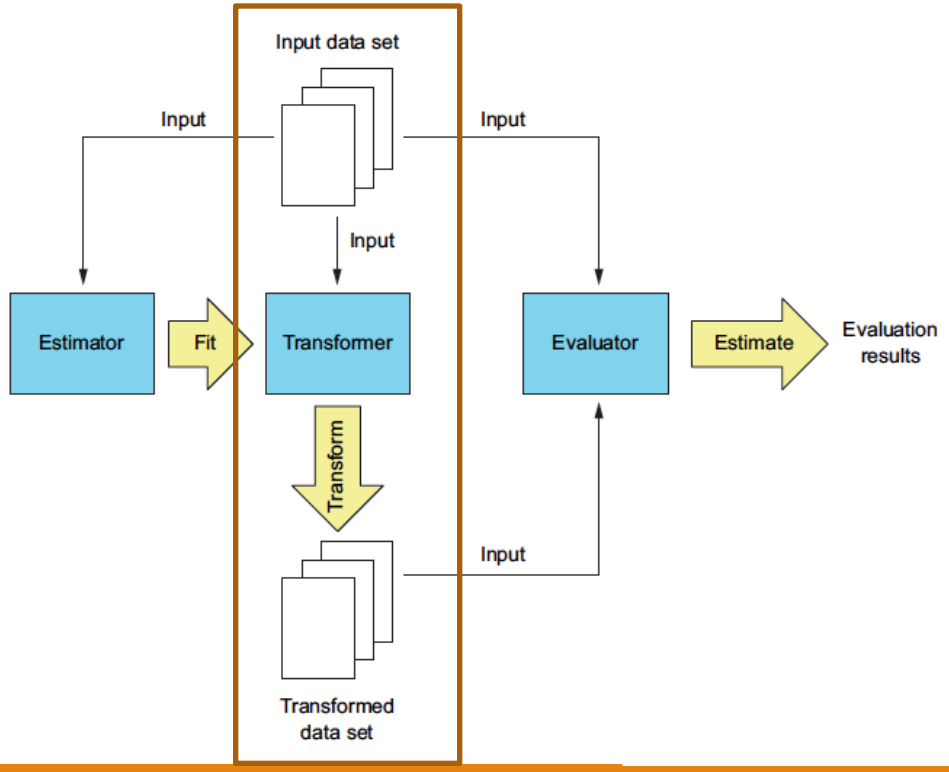
https://spark.apache.org/docs/2.0.1/api/java/org/apache/spark/sql/Dataset.html
https://spark.apache.org/docs/latest/ml-pipeline.html

Spark ML

Main Components

- Transformers
  - Convert a dataset to another.
  - This transforms datasets by adding predictions.
  - `transform()` : takes DataFrame and optional parameters.
- Estimators
- Evaluators

- ML Parameters
- ML Pipeline



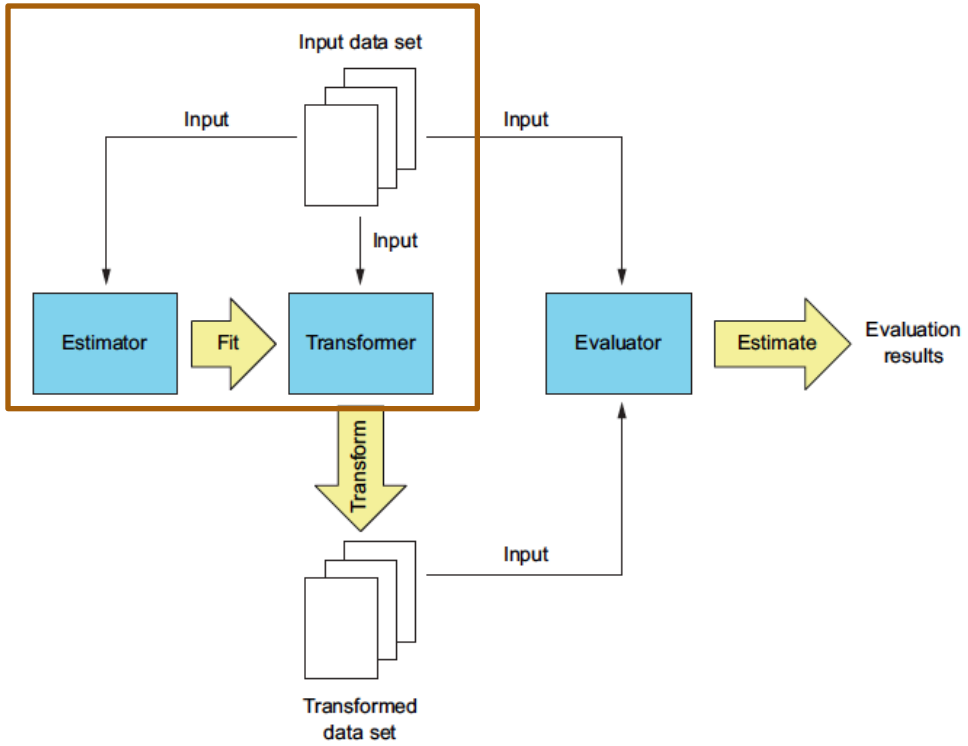
https://spark.apache.org/docs/latest/ml-pipeline.html

Spark ML

Main Components

- Transformers
- Estimators
  - Produces transformers by fitting on a dataset.
  - `fit()` : takes a DataFrame and parameters.
  - Ex. Linear regression produces a linear regression model with fitted weights and an intercepts, which is a transformer.
- Evaluators

- ML Parameters
- ML Pipeline

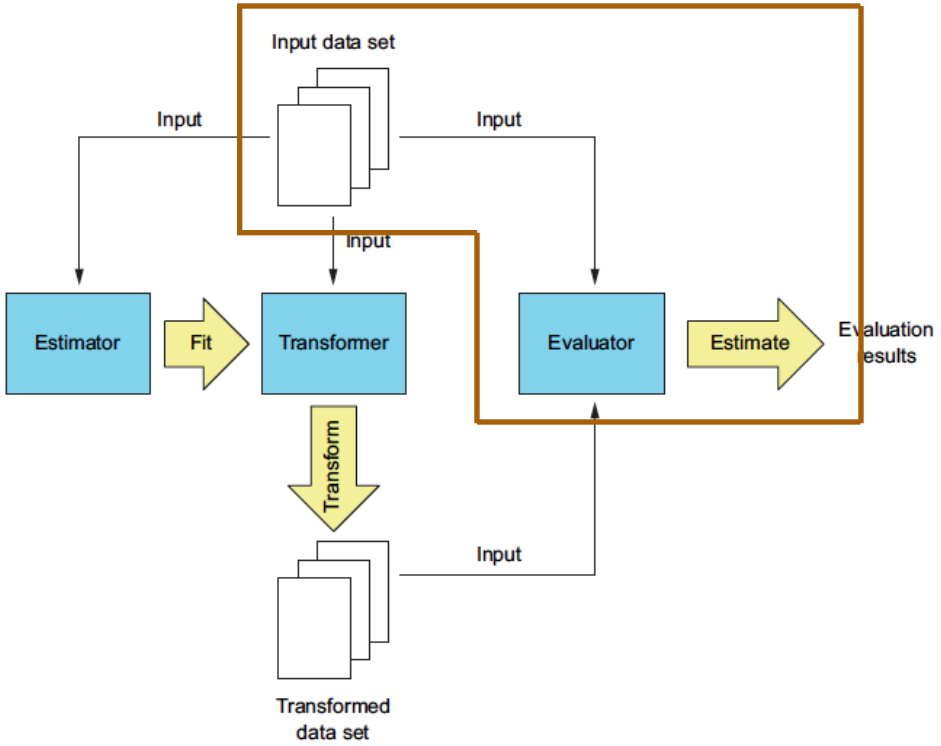


https://spark.apache.org/docs/latest/ml-pipeline.html

Spark ML

Main Components

- Transformers
- Estimators
- Evaluators
  - Evaluate the performance of a model.
  - `Evaluator()`
- ML Parameters
- ML Pipeline



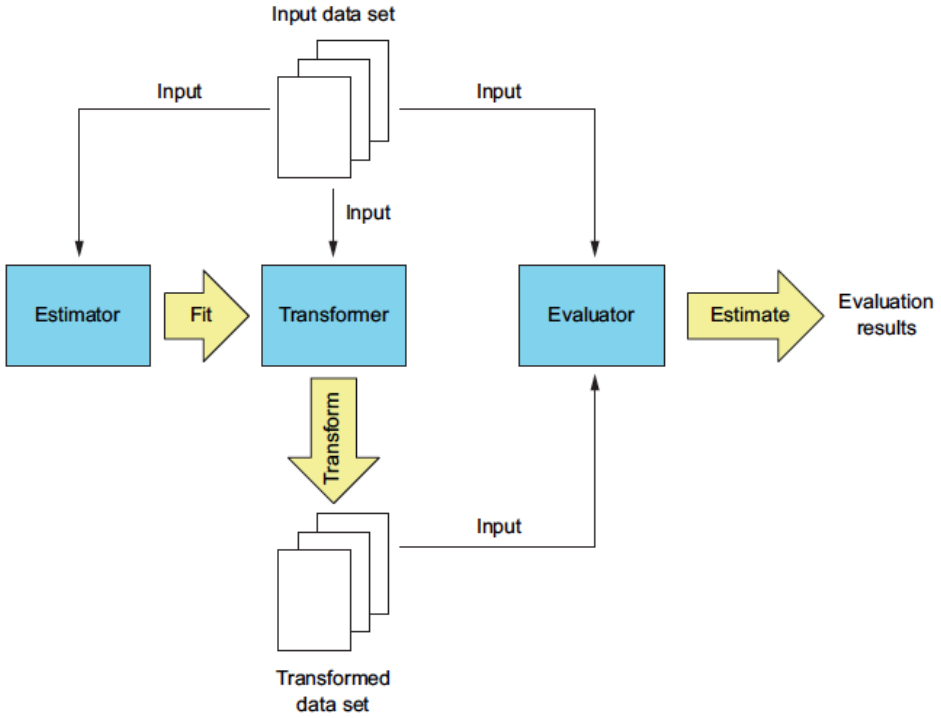
https://spark.apache.org/docs/latest/ml-pipeline.html

Spark ML

Main Components

- Transformers
- Estimators
- Evaluators

- ML Parameters
  - Specify parameters for estimators and transformers.
  - `Param(), ParamPair(), ParamMap()`
- ML Pipeline

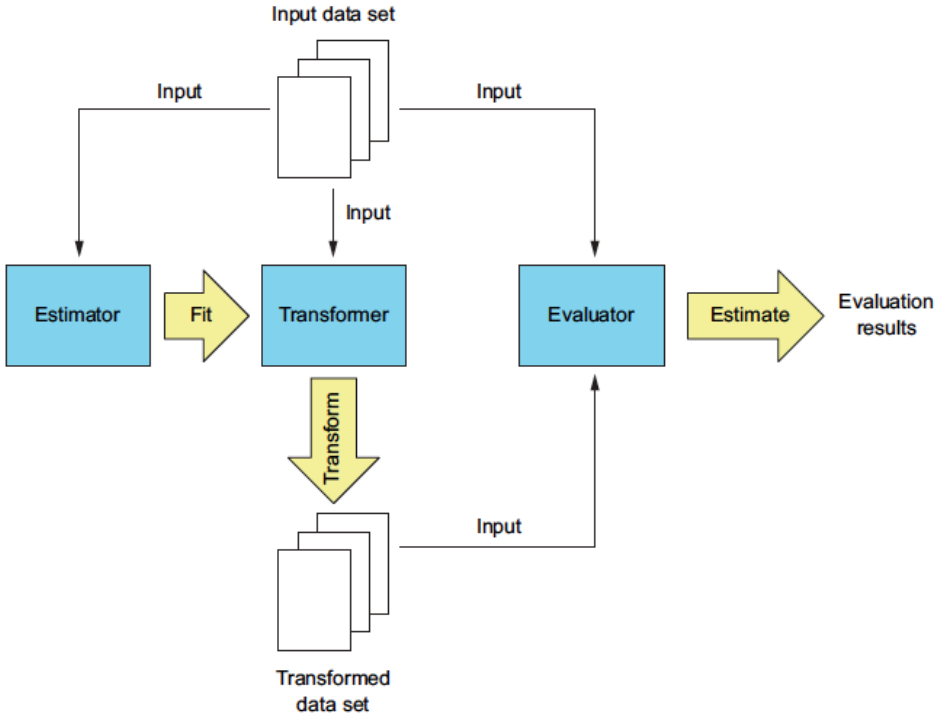


https://spark.apache.org/docs/latest/ml-pipeline.html

Spark ML

Main Components

- ML Pipeline (`PipelineModel()`)
  - In machine learning, the same steps are often repeated with slightly different parameters to find the best results.
  - A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.



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

Spark ML

Algorithms

- Feature Extractors, Transformers, and Selectors.
  - Feature Extractors : TF-IDF, Word2Vec, CountVectorizer
  - Feature Transformer : Tokenizer StopWordsRemover *n*-gram Binarizer PCA PolynomialExpansion Discrete Cosine Transform (DCT) StringIndexer IndexToString OneHotEncoder VectorIndexer Normalizer StandardScaler MinMaxScaler MaxAbsScaler Bucketizer ElementwiseProduct SQLTransformer VectorAssembler QuantileDiscretizer
  - Feature Selectors : VectorSlicer, Rformula,ChiSqSelector

https://spark.apache.org/docs/2.0.0-preview/ml-features.html

Spark ML

Algorithms

- Classification and Regression
  - Classification : Logistic regression, Decision tree classifier, Random forest classifier, Gradient-boosted tree classifier, Multilayer perceptron classifier, One-vs-Rest classifier (a.k.a. One-vs-All), Naive Bayes
  - Regression: Linear regression, Decision tree regression, Random forest regression, Gradient-boosted tree regression, Survival regression
  - Decision trees
  - Tree Ensembles: Random Forests, Gradient-Boosted Trees (GBTs)
- Clustering : K-means, Latent Dirichlet allocation (LDA), Bisecting k-means, Gaussian Mixture Model (GMM)
- Collaborative filtering

https://spark.apache.org/docs/2.0.0-preview/ml-features.html

Spark ML

Data Sets

- Adult Data Set : Prediction task is to determine whether a person makes over 50K a year.
  - 48842 instances.
  - Attribute Information:
    - 14 attributes.

      >50K, <=50K.

      age: continuous.
      workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
      fnlwgt: continuous.
      education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
      education-num: continuous.
      marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
      occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
      relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
      race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
      sex: Female, Male.
      capital-gain: continuous.
      capital-loss: continuous.
      hours-per-week: continuous.
      native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba,….

https://archive.ics.uci.edu/ml/datasets

Spark ML

Data Sets

- Optical Recognition of Handwritten Digits Data Sets. :
  - 10992 instances.
  - Attribute Information:
    - 16 pixels with intensity values in the range 0..100.
    - The last attribute is the class code 0..9

https://archive.ics.uci.edu/ml/datasets

Spark ML – Example 1

Logistic Regression Example.

- Adult Data Set : Prediction task is to determine whether a person makes over 50K a year.
  - 48842 instances.
  - Attribute Information:
    - 14 attributes.

      >50K, <=50K.

      age: continuous.
      workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
      fnlwgt: continuous.
      education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
      education-num: continuous.
      marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
      occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
      relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
      race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
      sex: Female, Male.
      capital-gain: continuous.
      capital-loss: continuous.
      hours-per-week: continuous.
      native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba,….

Spark ML – Example 1

Logistic Regression Example.

- Use adult.dat to generate a regression model.

1. Create an RDD.

```
def toDoubleSafe(v):
    try:
        return float(v)
    except ValueError:
        return v #if it is not a float type return as a string.
```

```
#load and convert the data
census_raw = sc.textFile("../adult.raw", 4).map(lambda x:  x.split(", "))
census_raw = census_raw.map(lambda row:  [toDoubleSafe(x) for x in row])
```

```
census_raw.take(1)
```

```
[[39.0,
  u'State-gov',
  77516.0,
  u'Bachelors',
  u'Never-married',
  u'Adm-clerical',
  u'Not-in-family',
  u'White',
  u'Male',
  2174.0,
```

Spark ML – Example 1

Logistic Regression Example.

2. Convert the RDD to DataFrame

```
:
# Create a dataframe.
from pyspark.sql.types import *
columns = ["age", "workclass", "fnlwgt", "education", "marital_status",
    "occupation", "relationship", "race", "sex", "capital_gain", "capital_loss",
    "hours_per_week", "native_country", "income"]
adultschema = StructType([
    StructField("age",DoubleType(),True),
    StructField("capital_gain",DoubleType(),True),
    StructField("capital_loss",DoubleType(),True),
    StructField("education",StringType(),True),
    StructField("fnlwgt",DoubleType(),True),
    StructField("hours_per_week",DoubleType(),True),
    StructField("income",StringType(),True),
    StructField("marital_status",StringType(),True),
    StructField("native_country",StringType(),True),
    StructField("occupation",StringType(),True),
    StructField("race",StringType(),True),
    StructField("relationship",StringType(),True),
    StructField("sex",StringType(),True),
    StructField("workclass",StringType(),True)
])
from pyspark.sql import Row
dfraw = sqlContext.createDataFrame(census_raw.map(lambda row: Row(**{x[0]: x[1] for x in zip(columns, row)})), adultsch
```

Spark ML – Example 1

Logistic Regression Example.

3. Clean the data. (I simply imputed the common value, although I prefer to use classification/regression for missing data imputation.)

```
#Check the most commonly used vals.
dfraw.groupBy(dfraw["workclass"]).count().show()
```

```
+----------------+-----+
|       workclass|count|
+----------------+-----+
|Self-emp-not-inc| 3862|
|       Local-gov| 3136|
|       State-gov| 1981|
|         Private|33906|
|     Without-pay|   21|
|     Federal-gov| 1432|
|    Never-worked|   10|
|               ?| 2799|
|    Self-emp-inc| 1695|
+----------------+-----+
```

Spark ML – Example 1

Logistic Regression Example.

3. Clean the data. (I simply imputed the common value, although I prefer to use classification/regression for missing data imputation.)

na : returns a DataFrameNA Function for handling missing values.

replace(*to_replace, value, subset=None*)

```
#Missing data imputation
dfrawrp = dfraw.na.replace(["?"], ["Private"], ["workclass"])
dfrawrpl = dfrawrp.na.replace(["?"], ["Prof-specialty"], ["occupation"])
dfrawnona = dfrawrpl.na.replace(["?"], ["United-States"], ["native_country"])
```

https://spark.apache.org/docs/1.6.2/api/python/pyspark.sql.html

Spark ML – Example 1

Logistic Regression Example.

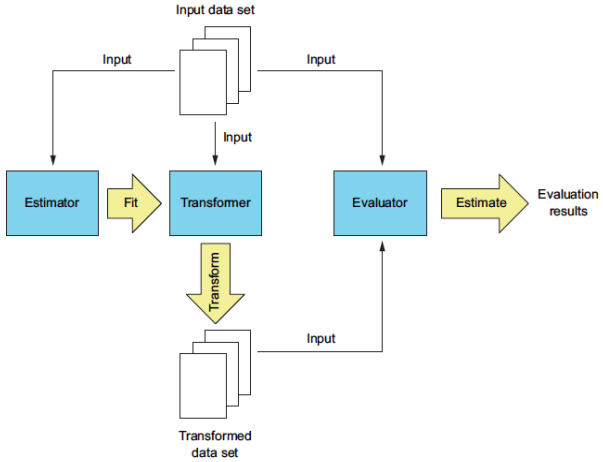
3. Clean the data.

Convert Strings to Categorical Values.

- String Indexer:
  - Convert string categorical values into integer indexes.
  - Takes a DataFrame and fits a `StringIndexerModel()` and used it for transformation.
- One-hot encoding :
  - Expand a column to as many columns as there are distinct strings in it and only one column contains a 1 and others are 0.
  - Create a new column as a one-hot-encoded sparse vector. (Replace a column with a vector.)

https://spark.apache.org/docs/1.6.2/api/python/pyspark.sql.html

Spark ML – Example 1



Logistic Regression Example.

3. Clean the data.

Convert Strings to Categorical Values.

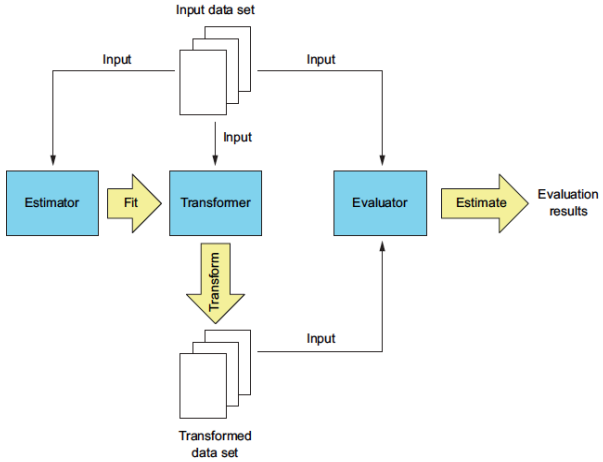
- String Indexer:
  - Convert string categorical values into integer indexes.
  - Takes a DataFrame and fits a `StringIndexerModel()` and used it for transformation.

```
#converting strings to numeric values
def indexStringColumns(df, cols):
    from pyspark.ml.feature import StringIndexer
    #variable newdf will be updated several times
    newdf = df
    for c in cols:
        si = StringIndexer(inputCol=c, outputCol=c+"-num")
        sm = si.fit(newdf)
        newdf = sm.transform(newdf).drop(c)
        newdf = newdf.withColumnRenamed(c+"-num", c)
    return newdf

dfnumeric = indexStringColumns(dfrawnona, ["workclass", "education", "marital_status", "occupation", "relationship", "r
```

https://spark.apache.org/docs/1.6.2/api/python/pyspark.sql.html

Spark ML – Example 1



Logistic Regression Example.

3. Clean the data.

Convert Strings to Categorical Values.

- String Indexer:
  - Convert string categorical values into integer indexes.
  - Takes a DataFrame and fits a `StringIndexerModel()` and used it for transformation.=

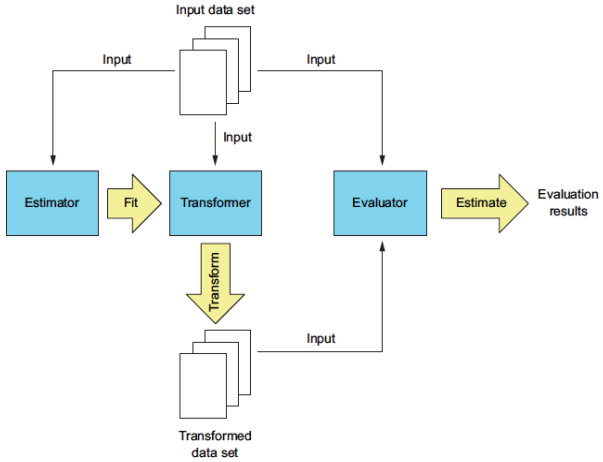
Before

```
dfraw.show()
```

```
+----+------------+------------+------------+--------+--------------+------+--------------------+--------------+-----
------------+------------------+-------------+------+----------------+
| age|capital_gain|capital_loss|   education|  fnlwgt|hours_per_week|income|
marital_status|native_country|      occupation|              race| relationship|   sex|       workclass|
+----+------------+------------+------------+--------+--------------+------+--------------------+--------------+-----
------------+------------------+-------------+------+----------------+
|39.0|      2174.0|         0.0|   Bachelors| 77516.0|          40.0| <=50K|       Never-married| United-States|
Adm-clerical|             White|Not-in-family|  Male|       State-gov|
|50.0|         0.0|         0.0|   Bachelors| 83311.0|          13.0| <=50K|  Married-civ-spouse| United-States|
Exec-managerial|             White|      Husband|  Male|Self-emp-not-inc|
|38.0|         0.0|         0.0|     HS-grad|215646.0|          40.0| <=50K|            Divorced| United-
States|Handlers-cleaners|             White|Not-in-family|  Male|         Private|
|53.0|         0.0|         0.0|        11th|234721.0|          40.0| <=50K|  Married-civ-spouse| United-
States|Handlers-cleaners|             Black|      Husband|  Male|         Private|
|28.0|         0.0|         0.0|   Bachelors|338409.0|          40.0| <=50K|  Married-civ-spouse|          Cuba|
```

https://s

Spark ML – Example 1



Logistic Regression Example.

3. Clean the data.

Convert Strings to Categorical Values.

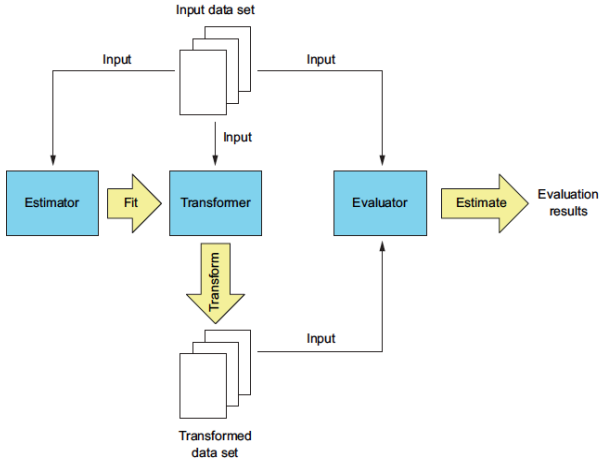
- String Indexer:
  - Convert string categorical values into integer indexes.
  - Takes a DataFrame and fits a `StringIndexerModel()` and used it for transformation.=

After

```
dfnumeric.show()
```

```
+----+------------+------------+--------+-------------+---------+---------+-------------+----------+------------+--
--+---+--------------+------+
| age|capital_gain|capital_loss|  fnlwgt|hours_per_week|workclass|education|marital_status|occupation|relationship|ra
ce|sex|native_country|income|
+----+------------+------------+--------+-------------+---------+---------+-------------+----------+------------+--
--+---+--------------+------+
|39.0|      2174.0|         0.0| 77516.0|         40.0|      3.0|      2.0|          1.0|       3.0|         1.0| 0
.0|0.0|           0.0|   0.0|
|50.0|         0.0|         0.0| 83311.0|         13.0|      1.0|      2.0|          0.0|       2.0|         0.0| 0
.0|0.0|           0.0|   0.0|
|38.0|         0.0|         0.0|215646.0|         40.0|      0.0|      0.0|          2.0|       8.0|         1.0| 0
.0|0.0|           0.0|   0.0|
|53.0|         0.0|         0.0|234721.0|         40.0|      0.0|      5.0|          0.0|       8.0|         0.0| 1
.0|0.0|           0.0|   0.0|
|28.0|         0.0|         0.0|338409.0|         40.0|      0.0|      2.0|          0.0|       0.0|         4.0| 1
```

Spark ML – Example 1



Logistic Regression Example.

3. Clean the data.

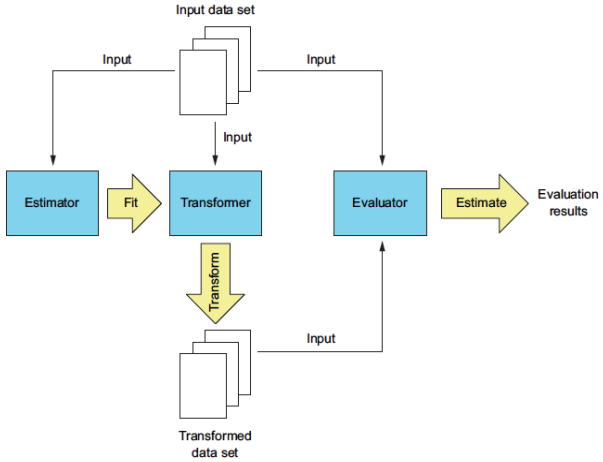
Convert Strings to Categorical Values.

- One-hot encoding :
  - Expand a column to as many columns as there are distinct strings in it and only one column contains a 1 and others are 0.
  - Create a new column as a one-hot-encoded sparse vector. (Replace a column with a vector.)

```
def oneHotEncodeColumns(df, cols):
    from pyspark.ml.feature import OneHotEncoder
    newdf = df
    for c in cols:
        onehotenc = OneHotEncoder(inputCol=c, outputCol=c+"-onehot", dropLast=False)
        newdf = onehotenc.transform(newdf).drop(c)
        newdf = newdf.withColumnRenamed(c+"-onehot", c)
    return newdf

dfhot = oneHotEncodeColumns(dfnumeric, ["workclass", "education", "marital_status", "occupation", "relationship", "race
```

Spark ML – Example 1



Logistic Regression Example.

3. Clean the data.

Convert Strings to Categorical Values.

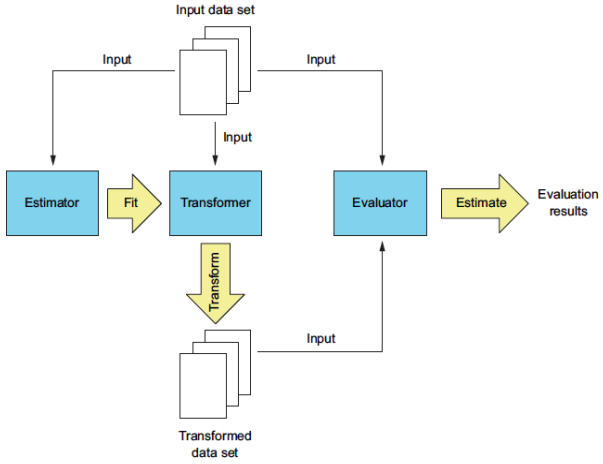
- One-hot encoding :
  - Expand a column to as many columns as there are distinct strings in it and only one column contains a 1 and others are 0.
  - Create a new column as a one-hot-encoded sparse vector. (Replace a column with a vector.)

Before

```
dfnumeric.show()
```

```
+----+------------+------------+--------+-------------+---------+---------+-------------+----------+------------+--
--+---+--------------+------+
| age|capital_gain|capital_loss|  fnlwgt|hours_per_week|workclass|education|marital_status|occupation|relationship|ra
ce|sex|native_country|income|
+----+------------+------------+--------+-------------+---------+---------+-------------+----------+------------+--
--+---+--------------+------+
|39.0|      2174.0|         0.0| 77516.0|         40.0|      3.0|      2.0|          1.0|       3.0|         1.0| 0
.0|0.0|           0.0|   0.0|
|50.0|         0.0|         0.0| 83311.0|         13.0|      1.0|      2.0|          0.0|       2.0|         0.0| 0
.0|0.0|           0.0|   0.0|
|38.0|         0.0|         0.0|215646.0|         40.0|      0.0|      0.0|          2.0|       8.0|         1.0| 0
.0|0.0|           0.0|   0.0|
|53.0|         0.0|         0.0|234721.0|         40.0|      0.0|      5.0|          0.0|       8.0|         0.0| 1
```

Spark ML – Example 1



Logistic Regression Example.

3. Clean the data.

Convert Strings to Categorical Values.

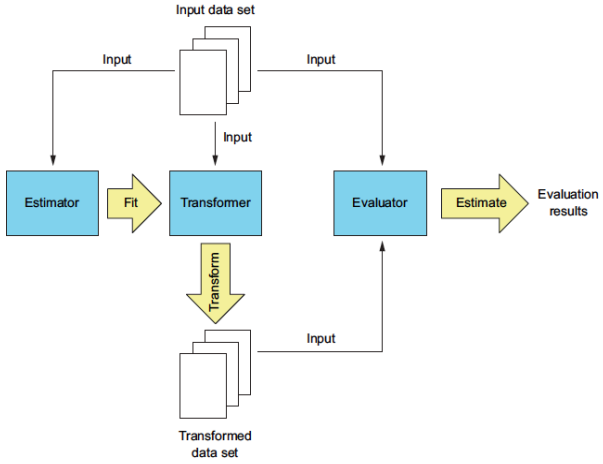
- One-hot encoding :
  - Expand a column to as many columns as there are distinct strings in it and only one column contains a 1 and others are 0.
  - Create a new column as a one-hot-encoded sparse vector. (Replace a column with a vector.)

After

```
dfhot.show()
```

```
+----+------------+------------+--------+--------------+---+------+-------------+---------------+--------------+-----
---------+-------------+-------------+---------------+
| age|capital_gain|capital_loss|  fnlwgt|hours_per_week|sex|income|    workclass|      education|marital_status|    o
ccupation| relationship|         race| native_country|
+----+------------+------------+--------+--------------+---+------+-------------+---------------+--------------+-----
---------+-------------+-------------+---------------+
|39.0|      2174.0|         0.0| 77516.0|          40.0|0.0|   0.0|(8,[3],[1.0])| (16,[2],[1.0])| (7,[1],[1.0])|(14,[
3],[1.0])|(6,[1],[1.0])|(5,[0],[1.0])| (41,[0],[1.0])|
|50.0|         0.0|         0.0| 83311.0|          13.0|0.0|   0.0|(8,[1],[1.0])| (16,[2],[1.0])| (7,[0],[1.0])|(14,[
2],[1.0])|(6,[0],[1.0])|(5,[0],[1.0])| (41,[0],[1.0])|
|38.0|         0.0|         0.0|215646.0|          40.0|0.0|   0.0|(8,[0],[1.0])| (16,[0],[1.0])| (7,[2],[1.0])|(14,[
8],[1.0])|(6,[1],[1.0])|(5,[0],[1.0])| (41,[0],[1.0])|
|53.0|         0.0|         0.0|234721.0|          40.0|0.0|   0.0|(8,[0],[1.0])| (16,[5],[1.0])| (7,[0],[1.0])|(14,[
```

Spark ML – Example 1



Logistic Regression Example.

3. Clean the data.

VectorAssembler()

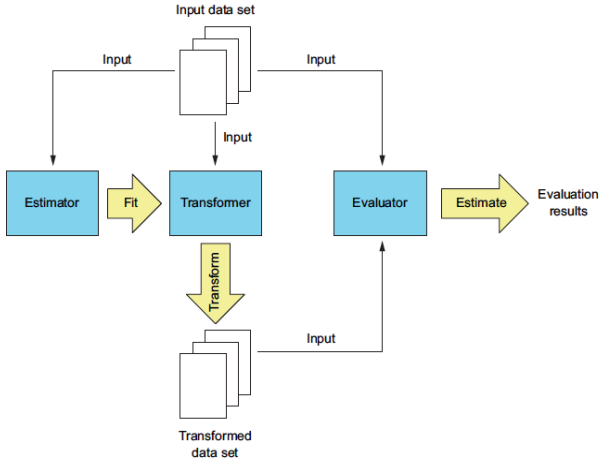
- Merge all the new vectors and the original columns into a single vector.
- Useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression and decision trees.

Before

```
dfhot.show()
```

```
+----+------------+------------+--------+--------------+---+------+-------------+---------------+--------------+-----
---------+-------------+-------------+---------------+
| age|capital_gain|capital_loss|  fnlwgt|hours_per_week|sex|income|    workclass|      education|marital_status|    o
ccupation| relationship|         race| native_country|
+----+------------+------------+--------+--------------+---+------+-------------+---------------+--------------+-----
---------+-------------+-------------+---------------+
|39.0|      2174.0|         0.0| 77516.0|          40.0|0.0|   0.0|(8,[3],[1.0])| (16,[2],[1.0])| (7,[1],[1.0])|(14,[
3],[1.0])|(6,[1],[1.0])|(5,[0],[1.0])| (41,[0],[1.0])|
|50.0|         0.0|         0.0| 83311.0|          13.0|0.0|   0.0|(8,[1],[1.0])| (16,[2],[1.0])| (7,[0],[1.0])|(14,[
2],[1.0])|(6,[0],[1.0])|(5,[0],[1.0])| (41,[0],[1.0])|
|38.0|         0.0|         0.0|215646.0|          40.0|0.0|   0.0|(8,[0],[1.0])| (16,[0],[1.0])| (7,[2],[1.0])|(14,[
8],[1.0])|(6,[1],[1.0])|(5,[0],[1.0])| (41,[0],[1.0])|
|53.0|         0.0|         0.0|234721.0|          40.0|0.0|   0.0|(8,[0],[1.0])| (16,[5],[1.0])| (7,[0],[1.0])|(14,[
```

Spark ML – Example 1



Logistic Regression Example.

3. Clean the data.

VectorAssembler()

- Merge all the new vectors and the original columns into a single vector.
- Useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression and decision trees.

After

```
lpoints.show()
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|(103,[0,1,3,4,9,1...|  0.0|
|(103,[0,3,4,7,16,...|  0.0|
|(103,[0,3,4,6,14,...|  0.0|
|(103,[0,3,4,6,19,...|  0.0|
|(103,[0,3,4,5,6,1...|  0.0|
|(103,[0,3,4,5,6,1...|  0.0|
|(103,[0,3,4,5,6,2...|  0.0|
```

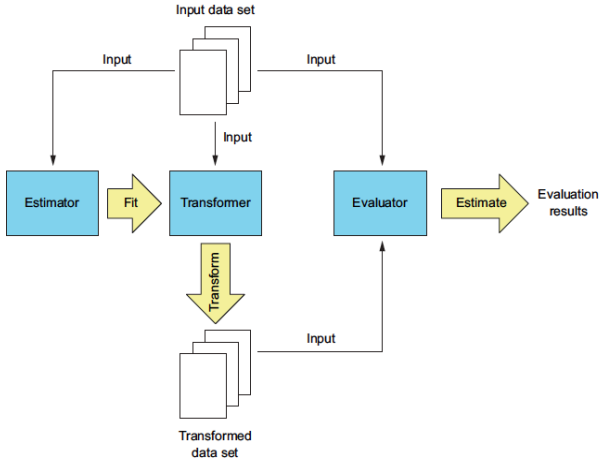
http://spark.apache.org/docs/latest/ml-f

Spark ML – Example 1

Logistic Regression Example.

4. Train the model.

Divide the datasets into training and testing sets.




```
#Divide the dataset into training and testing sets.
splits = lpoints.randomSplit([0.8, 0.2])
adulttrain = splits[0].cache()
adultvalid = splits[1].cache()
```

WHY??

http://spark.apache.org/docs/latest/ml-features.html#vectorassembler

Spark ML – Example 1

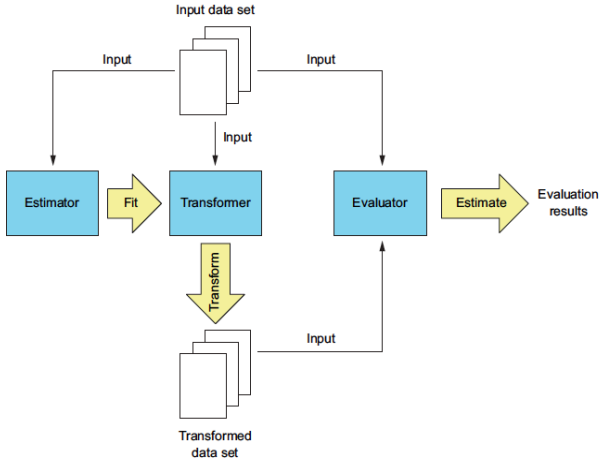
Logistic Regression Example.

4. Train the model.

Use Spark ML's `LogisticRegression`

Set parameters - regParam, maxIter, fitIntercept.

Call fit() passing in a DataFrame.



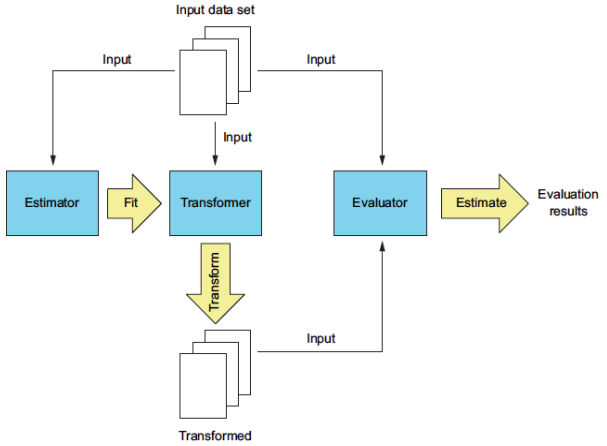

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(regParam=0.01, maxIter=1000, fitIntercept=True)
lrmodel = lr.fit(adulttrain)
lrmodel = lr.setParams(regParam=0.01, maxIter=500, fitIntercept=True).fit(adulttrain)
```

https://spark.apache.org/docs/1.6.1/ml-classification-regression.html#logistic-regression

Spark ML – Example 1

Logistic Regression Example.

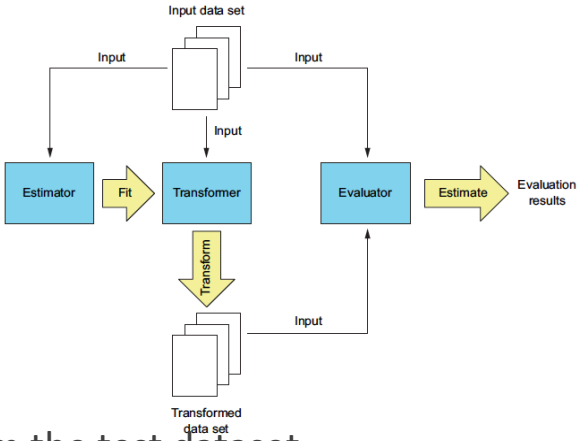
5. Interpret the model parameters.




```
#Interpret the model parameters
print(lrmodel.coefficients)
print(lrmodel.intercept)
```

```
[0.0200635956757,0.000142117910422,0.000549680557822,7.67939881134e-07,0.0270580225506,-0.527453343079,0.009663562261
61,-0.34291667775,0.0395917194118,-0.141854478053,0.280058560177,0.600089208761,-0.387298971983,-0.64014643248,-0.344
749447097,0.000956755374673,0.753174821018,1.08931162847,0.153992779575,-0.918979066595,0.173732711249,-1.11489212755
,-1.35296560719,1.64107789548,-1.23499994442,-0.611294148404,1.52228819816,-1.21910537157,-1.55424341573,-1.633814267
06,0.807335596657,-0.675633544989,-0.273023613173,-0.298616483069,-0.259763608797,-0.174182110048,0.892905297572,0.22
3649546869,0.0406062034707,0.674242185475,-0.0362094985227,0.196109637517,-0.747375692485,-0.271198569957,-0.13472289
5846,-0.616817018666,-0.910237980391,0.446199946491,0.351709412347,-0.895293985467,0.589665239528,0.471866510172,-0.0
964050040335,-0.817006143056,-0.325652759335,1.32289221548,-0.580583289363,0.194206452698,-0.591199710438,0.447122319
076,-0.00743155608916,-0.106991845472,0.403034553484,-0.392782163748,0.0315347018095,0.338559425942,0.749846528935,-0
.445684932018,-1.01727162353,0.207976345143,0.730583078689,-1.04443802189,0.0421863262162,-0.286338264139,0.134407218
438,-0.636935242122,-1.21433121715,-0.0680550984543,0.365524483517,0.0388005241512,-0.476740741173,0.199492582103,-0.
564159833277,-0.541838707731,-0.925450854006,0.656209481137,0.886629033942,-0.885074619199,-0.784944915792,0.88876345
1506,-0.840846549927,1.15928732002,-0.959851499168,-0.578807741163,-0.557760905544,-0.0365170688454,0.474330934434,-1
.14851367742,0.0968734548632,-0.177597852545,0.231903507463,-0.37008350593,-0.0334950683076]
-4.33068546155
```

Spark ML – Example 1



Logistic Regression Example.

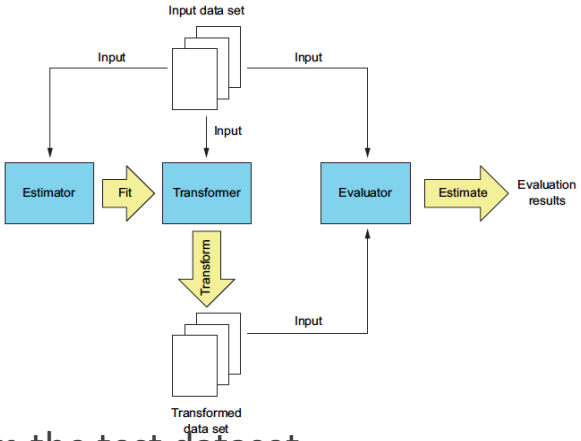
6. Evaluate classification models.

- First use lrmodel (the developedlinear regression model) to transform the test dataset.
- Then use `BinaryClassificationEvaluator()` to evaluate the performance.

```
#Evaluate models using test dataset.
validpredicts = lrmodel.transform(adultvalid)
validpredicts.show()
```

```
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|(103,[0,1,3,4,5,6...|  0.0|[0.65761126454865...|[0.65872358820700...|       0.0|
|(103,[0,1,3,4,5,6...|  1.0|[-0.9590664998231...|[0.27706513608652...|       1.0|
|(103,[0,1,3,4,5,6...|  0.0|[3.68197930538441...|[0.97544502461658...|       0.0|
|(103,[0,1,3,4,5,6...|  0.0|[4.09627501272418...|[0.98363765626483...|       0.0|
|(103,[0,1,3,4,5,6...|  1.0|[0.96345154510005...|[0.72381232948775...|       0.0|
|(103,[0,1,3,4,5,6...|  0.0|[2.28984133291450...|[0.90803220068241...|       0.0|
|(103,[0,1,3,4,5,6...|  0.0|[2.71279447036546...|[0.93777740821339...|       0.0|
|(103,[0,1,3,4,5,6...|  0.0|[4.64042974795558...|[0.99043875182615...|       0.0|
|(103,[0,1,3,4,5,6...|  0.0|[-2.2054793310857...|[0.09925952135483...|       1.0|
|(103,[0,1,3,4,5,6...|  1.0|[-0.3346962461104...|[0.41709839480886...|       1.0|
```

Spark ML – Example 1



Logistic Regression Example.

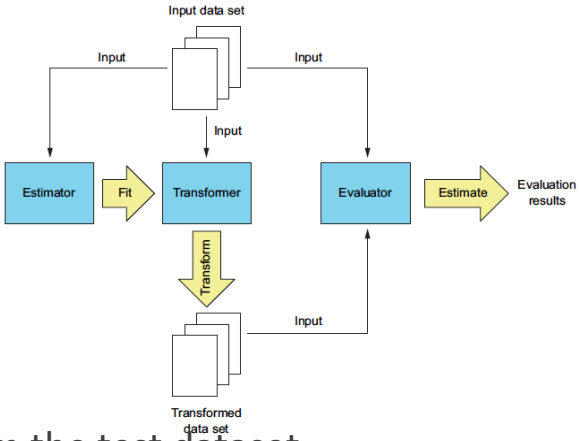
6. Evaluate classification models.

- First use lrmodel (the developedlinear regression model) to transform the test dataset.
- Then use `BinaryClassificationEvaluator()` to evaluate the performance.

```
#Evaluate the model. default metric : Area Under ROC
from pyspark.ml.evaluation import BinaryClassificationEvaluator
bceval = BinaryClassificationEvaluator()
print (bceval.getMetricName() +":" + str(bceval.evaluate(validpredicts)))
```

```
areaUnderROC:0.903469362164
```

Spark ML – Example 1



Logistic Regression Example.

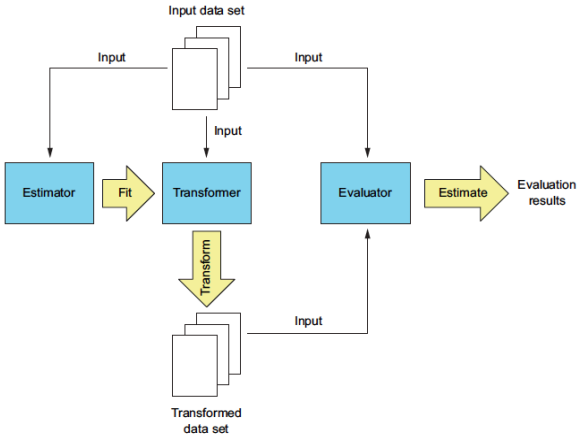
6. Evaluate classification models.

- First use lrmodel (the developedlinear regression model) to transform the test dataset.
- Then use `BinaryClassificationEvaluator()` to evaluate the performance.

```
#Evaluate the model. metric : Area Under PR
bceval.setMetricName("areaUnderPR")
print (bceval.getMetricName() +":" + str(bceval.evaluate(validpredicts)))
```

```
areaUnderPR:0.749049690617
```

Spark ML – Example 1



Logistic Regression Example.

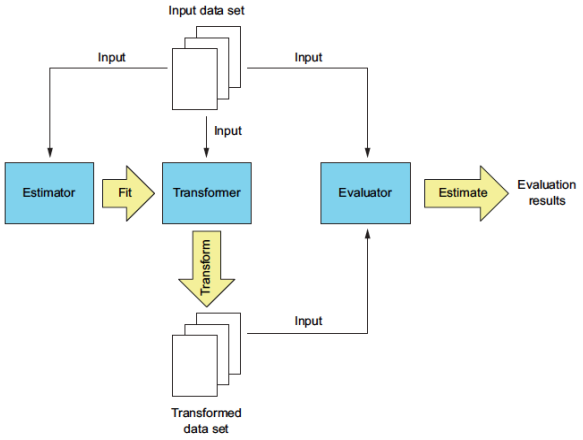
* n-fold cross-validation
  - Divide the dataset into n subsets of equal sizes and train n models excluding a different subset each time and train all n models.
  - Calculate the mean error for all n models and choose the set of parameters with the smallest average error.

```
# n-fold validation and the results.
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.tuning import ParamGridBuilder
cv = CrossValidator().setEstimator(lr).setEvaluator(bceval).setNumFolds(5)
paramGrid = ParamGridBuilder().addGrid(lr.maxIter, [1000]).addGrid(lr.regParam, [0.0001, 0.001, 0.005, 0.01, 0.05, 0.1,
cv.setEstimatorParamMaps(paramGrid)
cvmodel = cv.fit(adulttrain)
```

Spark ML – Example 1

Logistic Regression Example.

* n-fold cross-validation
  - `CrossValidator` class : estimator.
    - Takes estimator, evaluator and number of folds to use.
    - Takes several parameters in setEstimatorParamMaps()

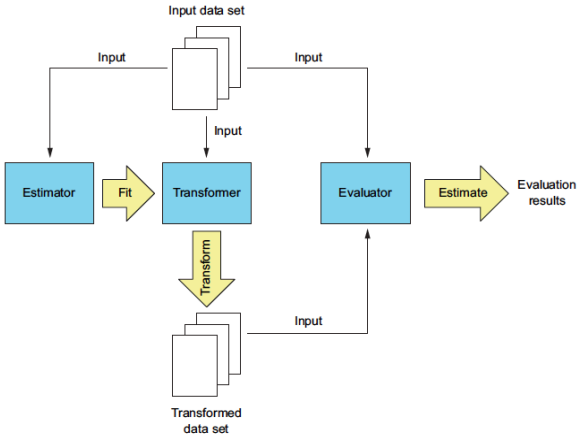



```
# n-fold validation and the results.
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.tuning import ParamGridBuilder
cv = CrossValidator().setEstimator(lr).setEvaluator(bceval).setNumFolds(5)
paramGrid = ParamGridBuilder().addGrid(lr.maxIter, [1000]).addGrid(lr.regParam, [0.0001, 0.001, 0.005, 0.01, 0.05, 0.1,
cv.setEstimatorParamMaps(paramGrid)
cvmodel = cv.fit(adulttrain)
```

Spark ML – Example 1

Logistic Regression Example.

* n-fold cross-validation
  - `CrossValidator`
    - You can access the selected model through `bestModel`.

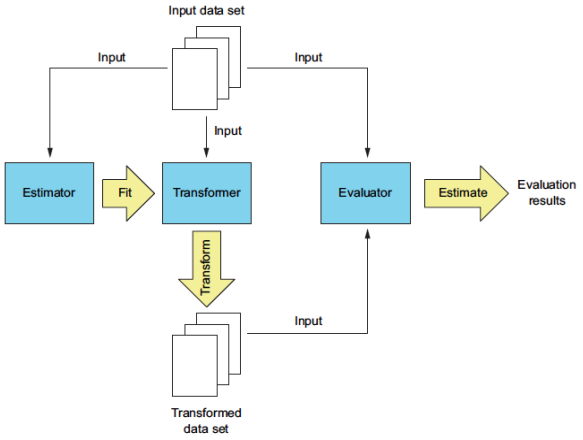



```
cvmodel.bestModel.coefficients
#cvmodel.bestModel.intercept
```

Spark ML – Example 1

Logistic Regression Example.

* n-fold cross-validation
  - `CrossValidator`
    - You can access the selected model through `bestModel`.




```
BinaryClassificationEvaluator().evaluate(cvmodel.bestModel.transform(adultvalid))
```

```
0.9068028091449186
```

```
BinaryClassificationEvaluator().setMetricName("areaUnderPR").evaluate(cvmodel.bestModel.transform(adultvalid))
```

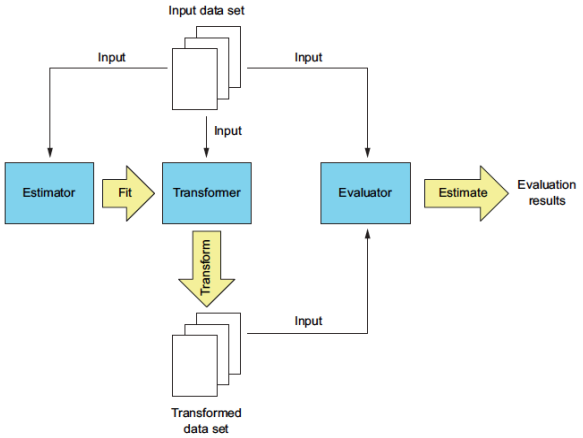
```
0.7609408397804686
```

Spark ML – Example 2

Decision Tree

- Pros
  - Do not require data normalization, can handle numerical/categorical values, and work with missing values.
- Cons
  - Prone to overfitting and is sensitive to the input data.
- `DecisionTreeClassifier()` - Binary Decision Tree

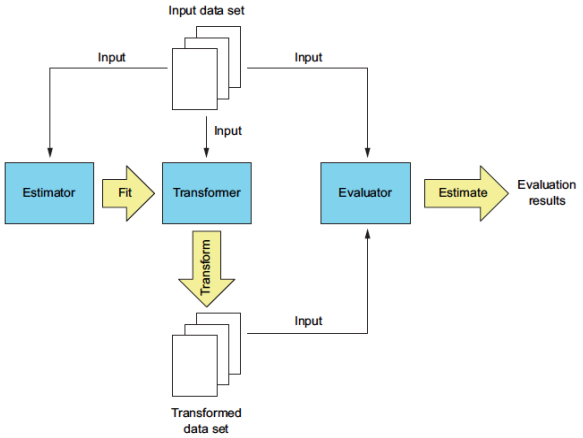
Spark ML – Example 2



Decision Tree Example

- Optical Recognition of Handwritten Digits Data Sets. :
  - 10992 instances.
  - Attribute Information:
    - 16 pixels with intensity values in the range 0..100.
    - The last attribute is the class code 0..9

Spark ML – Example 2



Decision Tree Example

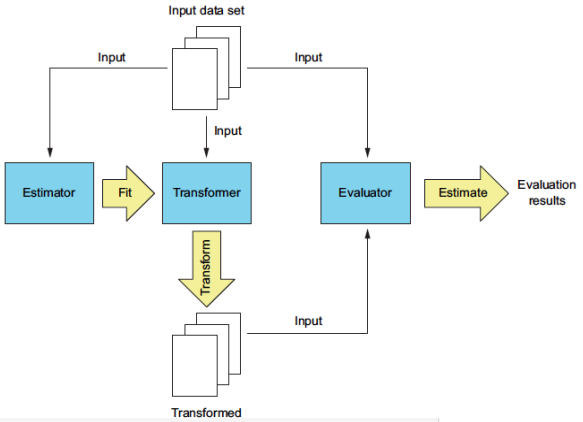
1. Create an RDD.

```
#Load the data and create an RDD (16 pixels and label)
pen_raw = sc.textFile("../penbased.dat", 4).map(lambda x:  x.split(", ")).map(lambda row: [float(x) for x in row])
```

```
pen_raw.take(1)
```

```
[[47.0,
  100.0,
  27.0,
  81.0,
  57.0,
  37.0,
  26.0,
  0.0,
  0.0,
  23.0,
  56.0,
  53.0,
  100.0,
  90.0,
  40.0,
  98.0,
  8.0]]
```

Spark ML – Example 2



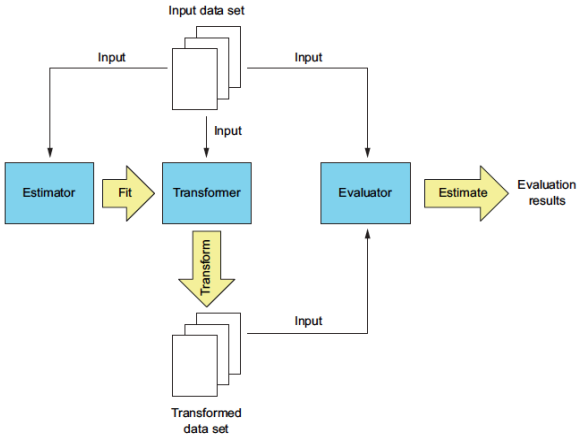
Decision Tree Example

2. Convert the RDD to DataFrame

```
#Create a DataFrame
from pyspark.sql.types import *
from pyspark.sql import Row
penschema = StructType([
    StructField("pix1",DoubleType(),True),
    StructField("pix2",DoubleType(),True),
    StructField("pix3",DoubleType(),True),
    StructField("pix4",DoubleType(),True),
    StructField("pix5",DoubleType(),True),
    StructField("pix6",DoubleType(),True),
    StructField("pix7",DoubleType(),True),
    StructField("pix8",DoubleType(),True),
    StructField("pix9",DoubleType(),True),
    StructField("pix10",DoubleType(),True),
    StructField("pix11",DoubleType(),True),
    StructField("pix12",DoubleType(),True),
    StructField("pix13",DoubleType(),True),
    StructField("pix14",DoubleType(),True),
    StructField("pix15",DoubleType(),True),
    StructField("pix16",DoubleType(),True),
    StructField("label",DoubleType(),True)
])

dfpen = sqlContext.createDataFrame(pen_raw.map(lambda x : Row(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8],x[9],x[10],x
```

Spark ML – Example 2



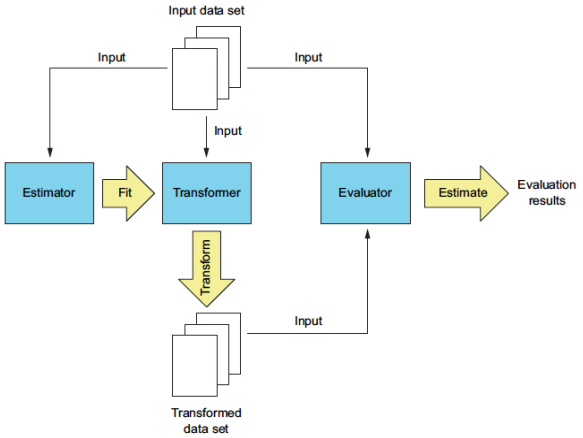
Decision Tree Example

2. Convert the RDD to DataFrame

```
dfpen.show()
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| pix1| pix2|pix3| pix4| pix5| pix6| pix7| pix8| pix9|pix10|pix11|pix12|pix13|pix14|pix15|pix16|label|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 47.0|100.0|27.0| 81.0| 57.0| 37.0| 26.0|  0.0|  0.0| 23.0| 56.0| 53.0|100.0| 90.0| 40.0| 98.0|  8.0|
|  0.0| 89.0|27.0|100.0| 42.0| 75.0| 29.0| 45.0| 15.0| 15.0| 37.0|  0.0| 69.0|  2.0|100.0|  6.0|  2.0|
|  0.0| 57.0|31.0| 68.0| 72.0| 90.0|100.0|100.0| 76.0| 75.0| 50.0| 51.0| 28.0| 25.0| 16.0|  0.0|  1.0|
|  0.0|100.0| 7.0| 92.0|  5.0| 68.0| 19.0| 45.0| 86.0| 34.0|100.0| 45.0| 74.0| 23.0| 67.0|  0.0|  4.0|
|  0.0| 67.0|49.0| 83.0|100.0|100.0| 81.0| 80.0| 60.0| 60.0| 40.0| 40.0| 33.0| 20.0| 47.0|  0.0|  1.0|
|100.0|100.0|88.0| 99.0| 49.0| 74.0| 17.0| 47.0|  0.0| 16.0| 37.0|  0.0| 73.0| 16.0| 20.0| 20.0|  6.0|
|  0.0|100.0| 3.0| 72.0| 26.0| 35.0| 85.0| 35.0|100.0| 71.0| 73.0| 97.0| 65.0| 49.0| 66.0|  0.0|  4.0|
|  0.0| 39.0| 2.0| 62.0| 11.0|  5.0| 63.0|  0.0|100.0| 43.0| 89.0| 99.0| 36.0|100.0|  0.0| 57.0|  0.0|
| 13.0| 89.0|12.0| 50.0| 72.0| 38.0| 56.0|  0.0|  4.0| 17.0|  0.0| 61.0| 32.0| 94.0|100.0|100.0|  5.0|
| 74.0| 87.0|31.0|100.0|  0.0| 69.0| 62.0| 64.0|100.0| 79.0|100.0| 38.0| 84.0|  0.0| 18.0|  1.0|  9.0|
| 48.0| 96.0|62.0| 65.0| 88.0| 27.0| 21.0|  0.0| 21.0| 33.0| 79.0| 67.0|100.0|100.0|  0.0| 85.0|  8.0|
|100.0|100.0|72.0| 99.0| 36.0| 78.0| 34.0| 54.0| 79.0| 47.0| 64.0| 13.0| 19.0|  0.0|  0.0|  2.0|  5.0|
```

Spark ML – Example 2



Decision Tree Example

3. Clean the data.

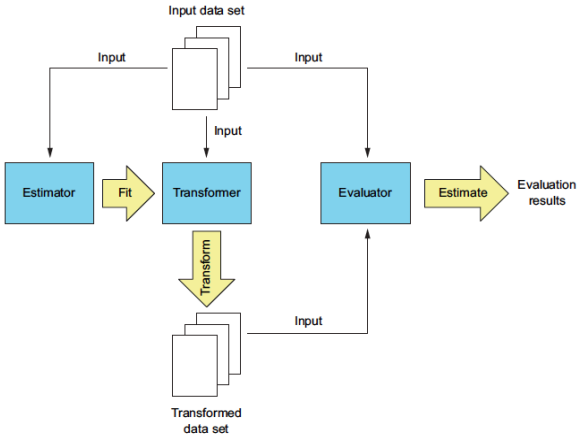
Merge Data.

```
# Merging the data with Vector Assembler.
from pyspark.ml.feature import VectorAssembler
va = VectorAssembler(outputCol="features", inputCols=dfpen.columns[0:-1])
penlpoints = va.transform(dfpen).select("features", "label")
```

```
penlpoints.show()
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[47.0,100.0,27.0,...|  8.0|
|[0.0,89.0,27.0,10...|  2.0|
|[0.0,57.0,31.0,68...|  1.0|
```

Spark ML – Example 2



Decision Tree Example

4. Train the data.

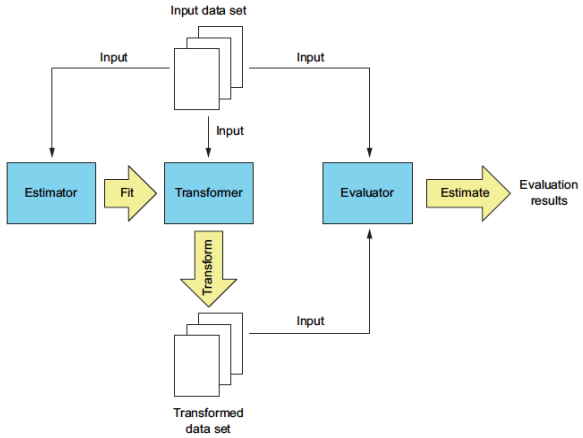
```
# Create Training and Test data.
pendtsets = penlpoints.randomSplit([0.8, 0.2])
pendttrain = pendtsets[0].cache()
pendtvalid = pendtsets[1].cache()
```

```
# Train the data.
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(maxDepth=20)
dtmodel = dt.fit(pendttrain)
```

Spark ML – Example 2

Decision Tree Example

5. Evaluate the model.




```
#Test data.
dtpredicts = dtmodel.transform(pendtvalid)
```

Spark ML – Example 2

Decision Tree Example

5. Evaluate the model.

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(dtpredicts)
print("Test Error = %g" % (1.0 - accuracy))
```

```
Test Error = 0.0449438
```

https://spark.apache.org/docs/2.0.1/api/java/org/apache/spark/ml/evaluation/MulticlassClassificationEvaluator.html

Spark ML –

Decision Tree Example

5. Evaluate the model.

```
from pyspark.mllib.evaluation import MulticlassMetrics
dtresrdd = dtpredicts.select("prediction", "label").rdd #convert DataFrame to RDD.
dtmm = MulticlassMetrics(dtresrdd)
dtmm.precision()
```

```
/usr/local/Cellar/apache-spark/2.0.2/libexec/python/pyspark/mllib/evaluation.py:237: UserWarning:
. Use accuracy.
  warnings.warn("Deprecated in 2.0.0. Use accuracy.")
```

```
0.9550561797752809
```

```
print(dtmm.confusionMatrix())
```

```
DenseMatrix([[ 204.,    1.,    0.,    0.,    2.,    0.,    0.,    0.,    1.,
          0.],
       [   0.,  179.,    6.,    1.,    1.,    1.,    0.,    2.,    1.,
          1.],
       [   0.,    7.,  186.,    0.,    1.,    0.,    0.,    0.,    0.,
          0.],
       [   0.,    1.,    0.,  164.,    0.,    2.,    0.,    0.,    0.,
          3.],
       [   1.,    1.,    0.,    0.,  190.,    0.,    1.,    0.,    0.,
          1.],
       [   0.,    1.,    0.,    2.,    2.,  177.,    0.,    0.,    0.,
          5.],
       [   2.,    0.,    1.,    0.,    1.,    0.,  176.,    0.,    0.,
          1.],
       [   0.,    2.,    0.,    0.,    0.,    0.,    0.,  210.,    1.,
          0.],
       [   4.,    3.,    0.,    0.,    0.,    3.,    1.,    2.,  177.,
          1.],
       [   0.,    3.,    0.,    1.,    0.,    1.,    1.,    4.,    4.,
        156.]])
```

MLlib MulticlassMetrics

- Takes RDD.
- multiclass evaluator.

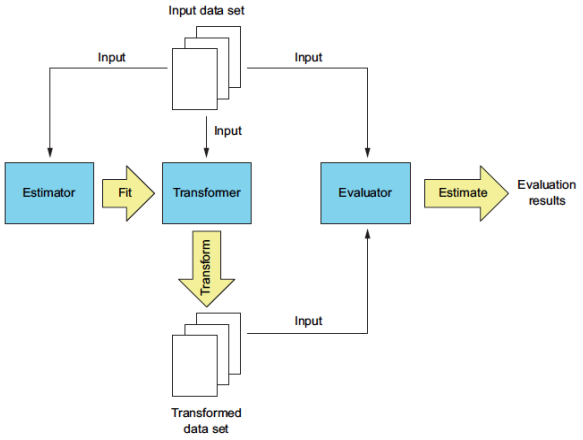
Spark ML – Example 3

Random Forest

- Ensemble learning method
  - Train a certain number of decision trees on data randomly sampled from the original data.
- Performs well on high dimensional datasets.

- `RandomForestClassifier`
  - Parameters
    - `numTrees` : The number of trees to train. Default : 20
    - `featureSubsetStrategy`
      - `all` – use all features.
      - `onethird` – randomly selects 1/3 of the features.
      - `sqrt` –sqrt(randomly selected number of features).
      - `log2`
      - `auto` – sqrt for classification and onethird for regression (default)
  - Make sure to have enough driver memory (configuration)

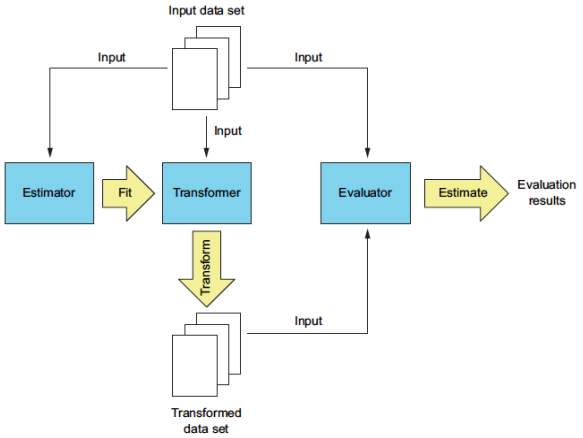
Spark ML – Example 3



Random Forest Example

- Optical Recognition of Handwritten Digits Data Sets. :
  - 10992 instances.
  - Attribute Information:
    - 16 pixels with intensity values in the range 0..100.
    - The last attribute is the class code 0..9

Spark ML – Example 3



Random Forest Example

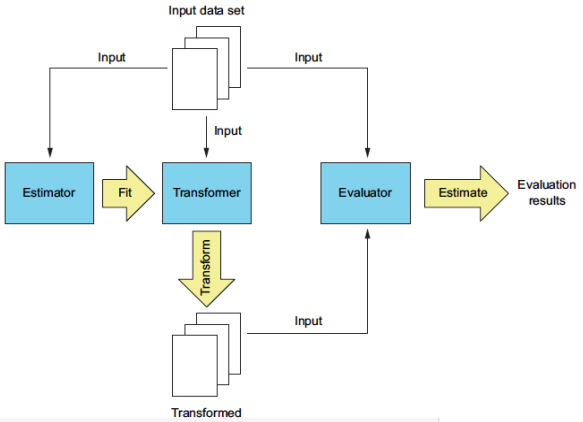
1. Create an RDD.

```
#Load the data and create an RDD (16 pixels and label)
pen_raw = sc.textFile("../penbased.dat", 4).map(lambda x:  x.split(", ")).map(lambda row: [float(x) for x in row])
```

```
pen_raw.take(1)
```

```
[[47.0,
  100.0,
  27.0,
  81.0,
  57.0,
  37.0,
  26.0,
  0.0,
  0.0,
  23.0,
  56.0,
  53.0,
  100.0,
  90.0,
  40.0,
  98.0,
  8.0]]
```

Spark ML – Example 3



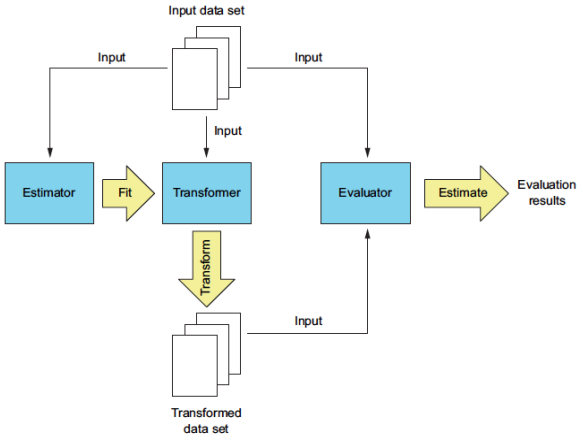
Random Forest Example

2. Convert the RDD to DataFrame

```
#Create a DataFrame
from pyspark.sql.types import *
from pyspark.sql import Row
penschema = StructType([
    StructField("pix1",DoubleType(),True),
    StructField("pix2",DoubleType(),True),
    StructField("pix3",DoubleType(),True),
    StructField("pix4",DoubleType(),True),
    StructField("pix5",DoubleType(),True),
    StructField("pix6",DoubleType(),True),
    StructField("pix7",DoubleType(),True),
    StructField("pix8",DoubleType(),True),
    StructField("pix9",DoubleType(),True),
    StructField("pix10",DoubleType(),True),
    StructField("pix11",DoubleType(),True),
    StructField("pix12",DoubleType(),True),
    StructField("pix13",DoubleType(),True),
    StructField("pix14",DoubleType(),True),
    StructField("pix15",DoubleType(),True),
    StructField("pix16",DoubleType(),True),
    StructField("label",DoubleType(),True)
])

dfpen = sqlContext.createDataFrame(pen_raw.map(lambda x : Row(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8],x[9],x[10],x
```

Spark ML – Example 3



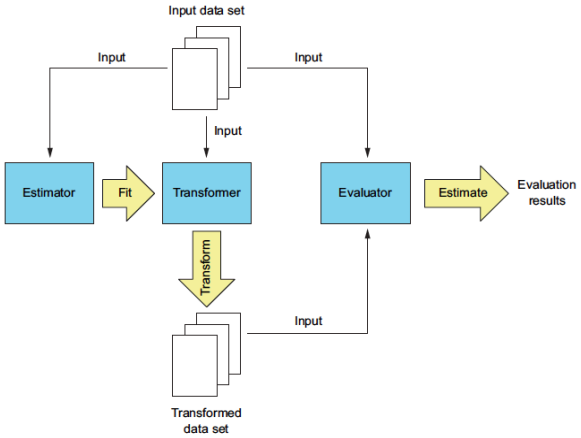
Random Forest Example

2. Convert the RDD to DataFrame

```
dfpen.show()
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| pix1| pix2|pix3| pix4| pix5| pix6| pix7| pix8| pix9|pix10|pix11|pix12|pix13|pix14|pix15|pix16|label|
+-----+-----+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 47.0|100.0|27.0| 81.0| 57.0| 37.0| 26.0|  0.0|  0.0| 23.0| 56.0| 53.0|100.0| 90.0| 40.0| 98.0|  8.0|
|  0.0| 89.0|27.0|100.0| 42.0| 75.0| 29.0| 45.0| 15.0| 15.0| 37.0|  0.0| 69.0|  2.0|100.0|  6.0|  2.0|
|  0.0| 57.0|31.0| 68.0| 72.0| 90.0|100.0|100.0| 76.0| 75.0| 50.0| 51.0| 28.0| 25.0| 16.0|  0.0|  1.0|
|  0.0|100.0| 7.0| 92.0|  5.0| 68.0| 19.0| 45.0| 86.0| 34.0|100.0| 45.0| 74.0| 23.0| 67.0|  0.0|  4.0|
|  0.0| 67.0|49.0| 83.0|100.0|100.0| 81.0| 80.0| 60.0| 60.0| 40.0| 40.0| 33.0| 20.0| 47.0|  0.0|  1.0|
|100.0|100.0|88.0| 99.0| 49.0| 74.0| 17.0| 47.0|  0.0| 16.0| 37.0|  0.0| 73.0| 16.0| 20.0| 20.0|  6.0|
|  0.0|100.0| 3.0| 72.0| 26.0| 35.0| 85.0| 35.0|100.0| 71.0| 73.0| 97.0| 65.0| 49.0| 66.0|  0.0|  4.0|
|  0.0| 39.0| 2.0| 62.0| 11.0|  5.0| 63.0|  0.0|100.0| 43.0| 89.0| 99.0| 36.0|100.0|  0.0| 57.0|  0.0|
| 13.0| 89.0|12.0| 50.0| 72.0| 38.0| 56.0|  0.0|  4.0| 17.0|  0.0| 61.0| 32.0| 94.0|100.0|100.0|  5.0|
| 74.0| 87.0|31.0|100.0|  0.0| 69.0| 62.0| 64.0|100.0| 79.0|100.0| 38.0| 84.0|  0.0| 18.0|  1.0|  9.0|
| 48.0| 96.0|62.0| 65.0| 88.0| 27.0| 21.0|  0.0| 21.0| 33.0| 79.0| 67.0|100.0|100.0|  0.0| 85.0|  8.0|
|100.0|100.0|72.0| 99.0| 36.0| 78.0| 34.0| 54.0| 79.0| 47.0| 64.0| 13.0| 19.0|  0.0|  0.0|  2.0|  5.0|
```

Spark ML – Example 3



Random Forest Example

3. Clean the data.

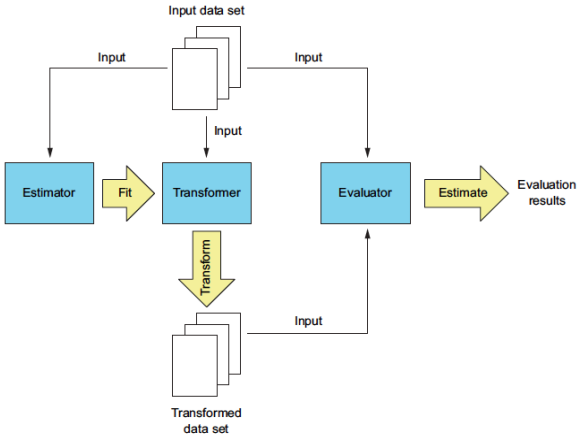
Merge Data.

```
# Merging the data with Vector Assembler.
from pyspark.ml.feature import VectorAssembler
va = VectorAssembler(outputCol="features", inputCols=dfpen.columns[0:-1])
penlpoints = va.transform(dfpen).select("features", "label")
```

```
penlpoints.show()
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[47.0,100.0,27.0,...|  8.0|
|[0.0,89.0,27.0,10...|  2.0|
|[0.0,57.0,31.0,68...|  1.0|
```

Spark ML – Example 3



Random Forest Example

4. Train.

```
# Train the model.
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(maxDepth=20)
rfmodel = rf.fit(pendttrain)
```

Spark ML

Random Forest Example

5. Evaluate the test dataset.

```
# Evaluate with the test data ste.
from pyspark.mllib.evaluation import MulticlassMetrics
rfpredicts = rfmodel.transform(pendtvalid)
rfresrdd = rfpredicts.select("prediction", "label").rdd
rfmm = MulticlassMetrics(rfresrdd)
rfmm.precision()
```

```
/usr/local/Cellar/apache-spark/2.0.2/libexec/python/pyspark/mllib/evaluation.py:2
. Use accuracy.
  warnings.warn("Deprecated in 2.0.0. Use accuracy.")
```

```
0.9855577689243028
```

```
print(rfmm.confusionMatrix())
```

```
DenseMatrix([[ 218.,    1.,    0.,    0.,    1.,    0.,    0.,    0.,    0.,
                 1.],
             [   0.,  202.,    5.,    2.,    0.,    0.,    0.,    0.,    0.,
                 0.],
             [   0.,    1.,  204.,    0.,    0.,    0.,    0.,    1.,    0.,
                 1.],
             [   0.,    0.,    1.,  189.,    0.,    0.,    0.,    1.,    0.,
                 1.],
             [   0.,    0.,    0.,    0.,  195.,    0.,    0.,    0.,    0.,
                 0.],
             [   0.,    0.,    0.,    3.,    0.,  174.,    0.,    0.,    0.,
                 4.],
             [   0.,    0.,    0.,    0.,    0.,    0.,  209.,    0.,    0.,
                 0.],
             [   0.,    0.,    0.,    0.,    0.,    0.,    0.,  227.,    0.,
                 0.],
             [   0.,    0.,    0.,    0.,    0.,    0.,    0.,    2.,  173.,
                 1.],
             [   0.,    1.,    0.,    0.,    1.,    0.,    0.,    0.,    1.,
               188.]])
```

Spark ML – Example 4

K-mean clustering

- Unsupervised learning
- Dataset should be standardized.
- Example – partition data into groups, anomaly detection, text/topic categorization

- `Kmeans`
  - K : number of clusters to find. (default – 2)

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

Spark ML – Example 4

K-mean clustering

1. Create an RDD.
2. Convert the RDD to DataFrame (This time only features!)

```
# Merging the data with Vector Assembler.
from pyspark.ml.feature import VectorAssembler
va = VectorAssembler(outputCol="features", inputCols=dfpen.columns[0:-1]) #except the last col.
penlpoints = va.transform(dfpen).select("features")
```

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

Spark ML – Example 4

K-mean clustering

3. Train the data.

```
from pyspark.ml.clustering import KMeans
kmeans =  KMeans(k=10)
model = kmeans.fit(penlpoints)
```

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

Spark ML – Example 4

K-mean clustering

4. Evaluate.

```
# Evaluate clustering by computing Within Set Sum of Squared Errors
wssse = model.computeCost(penlpoints)
print("WSSE = " + str(wssse))
```

```
WSSE = 45384621.4799
```

```
# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[  3.2364532   60.59359606  29.77463054  73.22413793  70.51724138
  89.29433498  95.06773399  93.63793103  83.34605911  73.94950739
  67.96305419  48.55172414  53.62068966  23.66995074  50.84975369
   1.33128079]
[ 88.67256637  98.18672566  53.37345133  87.8300885   21.44159292
  61.33893805   7.31150442  30.39115044  34.20619469   8.91150442
  79.47433628  13.90884956  60.06106195  28.37345133  12.58230088
  21.21858407]
[ 26.65292096  89.98969072  18.11168385  60.59621993  61.08419244
                .56872852   2.84536082   6.46735395  18.25945017
                .95876289  40.33161512  93.51890034  97.84364261
  98.104811  ]
```

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

Spark MLlib

Algorithms

- Basic Statistics : summary statistics, correlations, stratified sampling, hypothesis testing, streaming significance testing, random data generation
- Classification and Regression : Linear Support Vector Machines (SVMs), Logistic regression, Linear least squares, Lasso, and ridge regression, naive Bayes decision trees ensembles of trees (Random Forests and Gradient-Boosted Trees) isotonic regression
- Collaborative Filtering : alternating least squares (ALS)
- Clustering : k-means, Gaussian mixture, power iteration clustering (PIC), latent Dirichlet allocation (LDA), bisecting k-means, streaming k-means

https://spark.apache.org/docs/2.0.0-preview/mllib-guide.html

Spark MLlib

Algorithms

- Dimensionality Reduction : singular value decomposition (SVD), principal component analysis (PCA)
- Feature extraction and transformation
- Frequent Pattern Mining : FP-growth, association rules, PrefixSpan

Spark MLlib

Logistic Regression

- Use set of independent variables to make predictions about a target variable (label).
- Assumption : a linear relationship between the independent and target variables.

- `LogisticRegressionWithSGD`
  - Logistic regression using stochastic gradient descent.

Spark MLlib – Example 5

Logistic Regression Example

- Spam Classification

1. Create an RDD.

```
spam = sc.textFile("../spam.txt")
nospam = sc.textFile("../nospam.txt")
```

```
spam.take(2)
```

```
[u'Dear sir, I am a Prince in a far kingdom you have not heard of.  I want to send you money via wire transfer so ple
ase ...',
 u'Get Viagra real cheap!  Send money right away to ...']
```

```
nospam.take(2)
```

```
[u'Dear Spark Learner, Thanks so much for attending the Spark Summit 2014!  Check out videos of talks from the summit
 at ...',
 u'Hi Mom, Apologies for being late about emailing and forgetting to send you the package.  I hope you and bro have b
een ...']
```

Spark MLlib – Example 5

Logistic Regression Example

- Spam Classification

2. Clean the data.

Feature Extraction/Transformation.

- `HashingTF`
  - Maps a sequence of terms to their term frequencies using the hashing trick.

https://spark.apache.org/docs/1.2.0/api/scala/index.html#org.apache.spark.mllib.feature.HashingTF

Spark MLlib – Example 5

Logistic Regression Example

- Spam Classification

2. Clean the data.

```
from pyspark.mllib.feature import HashingTF
# Create a HashingTF instance to map email text to vectors of 100 features.
tf = HashingTF(numFeatures = 100)
# Each email is split into words, and each word is mapped to one feature.
spamFeatures = spam.map(lambda email: tf.transform(email.split(" ")))
nospamFeatures = nospam.map(lambda email: tf.transform(email.split(" ")))
```

```
spamFeatures.collect()
#nospamFeautes.collect()
```

```
[SparseVector(100, {0: 1.0, 1: 1.0, 4: 2.0, 8: 1.0, 9: 1.0, 11: 1.0, 16: 3.0, 18: 1.0, 20: 1.0, 21: 1.0, 28: 1.0, 29:
 1.0, 37: 1.0, 38: 1.0, 42: 1.0, 44: 2.0, 45: 1.0, 48: 1.0, 50: 1.0, 51: 1.0, 72: 1.0, 75: 1.0, 82: 2.0}),
 SparseVector(100, {0: 1.0, 2: 1.0, 21: 1.0, 26: 1.0, 29: 1.0, 37: 1.0, 50: 1.0, 59: 2.0, 76: 1.0, 80: 1.0}),
 SparseVector(100, {1: 1.0, 3: 1.0, 8: 1.0, 12: 1.0, 17: 1.0, 21: 1.0, 25: 1.0, 34: 2.0, 38: 1.0, 44: 1.0, 45: 1.0, 5
9: 3.0, 71: 1.0, 75: 2.0, 76: 1.0, 82: 1.0, 84: 1.0, 95: 1.0}),
 SparseVector(100, {0: 2.0, 6: 1.0, 8: 1.0, 9: 1.0, 19: 1.0, 21: 1.0, 25: 1.0, 33: 2.0, 37: 1.0, 40: 1.0, 55: 1.0, 60
```

Spark MLlib – Example 5

Logistic Regression Example

- Spam Classification

2. Clean the data.

`LabeldPoint (label, features)`

```
from pyspark.mllib.regression import LabeledPoint
# Create LabeledPoint datasets for positive (spam) and negative (ham) examples.
positiveExamples = spamFeatures.map(lambda features: LabeledPoint(1, features))
negativeExamples = nospamFeatures.map(lambda features: LabeledPoint(0, features))
training_data = positiveExamples.union(negativeExamples)
training_data.cache() # Cache data since Logistic Regression is an iterative algorithm.
```

```
UnionRDD[221] at union at NativeMethodAccessorImpl.java:-2
```

```
training_data.collect()
```

```
[LabeledPoint(1.0, (100,[0,1,4,8,9,11,16,18,20,21,28,29,37,38,42,44,45,48,50,51,72,75,82],[1.0,1.0,2.0,1.0,1.0,1.0,3.
0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0])),
 LabeledPoint(1.0, (100,[0,2,21,26,29,37,50,59,76,80],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0])),
 LabeledPoint(1.0, (100,[1,3,8,12,17,21,25,34,38,44,45,59,71,75,76,82,84,95],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0
,1.0,3.0,1.0,2.0,1.0,1.0,1.0,1.0])),
 LabeledPoint(1.0, (100,[0,6,8,9,19,21,25,33,37,40,55,60,62,63,71,72,74,76,79,81],[2.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.
0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])),
```

https://spark.apache.org/docs/0.8.1/api/mllib/org/apache/spark/mllib/regression/LabeledPoint.html

Spark MLlib – Example 5

Logistic Regression Example

- Spam Classification

3. Train the model.

```
#Train
from pyspark.mllib.classification import LogisticRegressionWithSGD
# Run Logistic Regression using the SGD optimizer.
# regParam is model regularization, which can make models more robust.
model = LogisticRegressionWithSGD.train(training_data)
```

```
/usr/local/Cellar/apache-spark/2.0.2/libexec/python/pyspark/mllib/classification.py:313: UserWarning: Deprecated in 2
.0.0. Use ml.classification.LogisticRegression or LogisticRegressionWithLBFGS.
  "Deprecated in 2.0.0. Use ml.classification.LogisticRegression or "
```

Spark MLlib – Example 5

Logistic Regression Example

- Spam Classification

3. Train the model.

```
#Interpret the model
import operator
print(",".join([str(s) for s in sorted(enumerate([abs(x) for x in model.weights.toArray()]), key=operator.itemgetter(0)
```

```
(0, 0.07291910167120795),(1, 0.27641028312863813),(2, 0.18084306504417552),(3, 0.10776168428967071),(4, 0.54050102526
986021),(5, 0.0),(6, 0.13332249583574537),(7, 0.0),(8, 0.67927755040357385),(9, 0.076523781524273499),(10, 0.0),(11,
0.008754005119169973),(12, 0.02702461938482734),(13, 0.0),(14, 0.19206934808011311),(15, 0.0),(16, 0.3732377830870003
4),(17, 0.18651434450400761),(18, 0.078567673616367364),(19, 0.026853653187871338),(20, 0.078567673616367364),(21, 0.
19609183310322334),(22, 0.0),(23, 0.2248672288970984),(24, 0.12061838368254736),(25, 0.2334900860404964),(26, 0.19948
462262476277),(27, 0.0),(28, 0.11843820447040838),(29, 0.41745598205794476),(30, 0.0),(31, 0.067639955906277319),(32,
 0.0),(33, 0.42159583838964215),(34, 0.33965904422936471),(35, 0.0),(36, 0.11735674213677989),(37, 0.0993122390704887
5),(38, 0.12389870038959433),(39, 0.077475423359075785),(40, 0.10328743243450288),(41, 0.0),(42, 0.27063702169648068)
,(43, 0.45780311102801352),(44, 0.30299871838614351),(45, 0.11623413410502156),(46, 0.26768663578393503),(47, 0.0),(4
8, 0.18607816037668568),(49, 0.10751048676031848),(50, 0.57763213108156231),(51, 0.033566577637641808),(52, 0.1920693
4808011311),(53, 0.15495084671815157),(54, 0.0),(55, 0.21079791919482108),(56, 0.11735674213677989),(57, 0.2877914945
5159897),(58, 0.26700021187949907),(59, 0.66996561350538286),(60, 0.21079791919482108),(61, 0.15948972511918019),(62,
```

Spark MLlib – Example 5

Logistic Regression Example

- Spam Classification

4. Test/Evaluate.

You can try with more test data sets and RegressionMetrics to calculate MSE/RMSE.

```
# Test
# Test on a positive example and a negative one.
# First apply the same HashingTF feature transformation used on the training data.
posTestExample = tf.transform("O M G GET cheap stuff by sending money to ...".split(" "))
negTestExample = tf.transform("Hi Dad, I started studying Spark the other ...".split(" "))

# Now use the learned model to predict spam/ham for new emails.
# Can be extended to use"from pyspark.mllib.evaluation import RegressionMetrics"
# to get rootMeanSquaredError and meanSqureError.
print "Prediction for positive test example: %g" % model.predict(posTestExample)
print "Prediction for negative test example: %g" % model.predict(negTestExample)
```

```
Prediction for positive test example: 1
Prediction for negative test example: 0
```

Reference

Spark Online Documentation : http://spark.apache.org/docs/latest/

Karau, Holden, et al. *Learning spark: lightning-fast big data analysis*. O'Reilly Media, Inc., 2015.

Zecevic, Petar, et al. Spark in Action, Manning, 2016.