# Use PySpark with version 2.0

>>> SPARK\_MAJOR\_VERSION=2 pyspark

>>> sc.version

u'2.0.0.2.5.3.0-37'

**1. Pandas and numpy in Spark**

Pandas and numpy are available in Spark. But they are not widely used.

>>> import pandas as pd

>>> import numpy as np

>>> sample = sc.parallelize([np.array([1,10,100]),np.array([2,20,200]),np.array([3,30,300])])

>>> sample

ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:475

# Pandas and Numpy in Spark are still based on RDD, not DataFrame

# MLlib is based on RDD

# Example 1

>>> from pyspark.mllib.stat import Statistics

>>> statics = Statistics.colStats(sample)

>>> print statics.mean()

[ 2. 20. 200.]

>>> print statics.variance()

[ 1.00000000e+00 1.00000000e+02 1.00000000e+04]

# Example 2

>>> from pyspark.mllib.stat import Statistics

>>> seriesX = sc.parallelize([1.0, 2.0, 3.0, 3.0, 5.0])

>>> seriesY = sc.parallelize([11.0, 22.0, 33.0, 33.0, 555.0])

>>> print("Correlation is: " + str(Statistics.corr(seriesX, seriesY, method="pearson")))

Correlation is: 0.850028676877

# Example 3

>>> x = sc.parallelize([1.0, 0.0, -2.0], 2)

>>> y = sc.parallelize([4.0, 5.0, 3.0], 2)

>>> zeros = sc.parallelize([0.0, 0.0, 0.0], 2)

>>> Statistics.corr(x, y, "pearson")

0.6546536707079771

>>> Statistics.corr(x, y, "spearman")

0.5

# Example 4: vectors

>>> from pyspark.mllib.linalg import Vectors

>>> rdd = sc.parallelize([Vectors.dense([1, 0, 0, -2]), Vectors.dense([4, 5, 0, 3]),Vectors.dense([6, 7, 0, 8]), Vectors.dense([9, 0, 0, 1])])

>>> pearsonCorr = Statistics.corr(rdd)

18/05/12 16:10:42 WARN PearsonCorrelation: Pearson correlation matrix contains NaN values.

>>> print(str(pearsonCorr))

[[ 1. 0.05564149 nan 0.40047142]

[ 0.05564149 1. nan 0.91359586]

[ nan nan 1. nan]

[ 0.40047142 0.91359586 nan 1. ]]

**2. IRIS example**

**(1). Input**

IRIS dataset.

# Get and put the dataset to HDFS

$ hdfs dfs –get /user/Jason/data/iris.csv

$ hdfs dfs –put ./iris.csv /user/aisjiaw/data/iris.csv

$ hdfs dfs -ls /user/aisjiaw/data

Found 2 items

-rwxrwxrwx 3 aisjiaw student 5107 2017-11-07 12:16 /user/aisjiaw/data/iris.csv

-rw-r--r-- 3 aisjiaw student 5 2017-09-16 19:15 /user/aisjiaw/data/test.txt

# data samples

Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species

1,5.1,3.5,1.4,0.2,Iris-setosa

2,4.9,3.0,1.4,0.2,Iris-setosa

3,4.7,3.2,1.3,0.2,Iris-setosa

4,4.6,3.1,1.5,0.2,Iris-setosa

5,5.0,3.6,1.4,0.2,Iris-setosa

6,5.4,3.9,1.7,0.4,Iris-setosa

7,4.6,3.4,1.4,0.3,Iris-setosa

8,5.0,3.4,1.5,0.2,Iris-setosa

9,4.4,2.9,1.4,0.2,Iris-setosa

10,4.9,3.1,1.5,0.1,Iris-setosa

11,5.4,3.7,1.5,0.2,Iris-setosa

12,4.8,3.4,1.6,0.2,Iris-setosa

13,4.8,3.0,1.4,0.1,Iris-setosa

…

**(2). Processing and model building**

# Import libraries

>>> from pyspark.sql import SQLContext

>>> from pyspark.sql.types import \*

>>> from pyspark.ml.feature import VectorAssembler, StringIndexer

>>> from pyspark.ml.classification import NaiveBayes

>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Build a dataframe

>>> struct = StructType([

StructField('Id', IntegerType(), True),

... StructField('SepalLengthCm', DoubleType(), True),

... StructField('SepalWidthCm', DoubleType(), True),

... StructField('PetalLengthCm', DoubleType(), True),

... StructField('PetalWidthCm', DoubleType(), True),

... StructField('Species', DoubleType(), True)

... ])

# Read data to the dataframe

>>> df\_iris = spark.read.csv('hdfs:///user/aisjiaw/data/iris.csv', header=True, schema=struct)

>>> df\_iris

DataFrame[Id: int, SepalLengthCm: double, SepalWidthCm: double, PetalLengthCm: double, PetalWidthCm: double, Species: string]

>>> df\_iris.show(2)

+---+-------------+------------+-------------+------------+-----------+

| Id|SepalLengthCm|SepalWidthCm|PetalLengthCm|PetalWidthCm| Species|

+---+-------------+------------+-------------+------------+-----------+

| 1| 5.1| 3.5| 1.4| 0.2|Iris-setosa|

| 2| 4.9| 3.0| 1.4| 0.2|Iris-setosa|

+---+-------------+------------+-------------+------------+-----------+

only showing top 2 rows

>>> df\_iris.describe().show()

18/05/12 16:24:50 WARN Utils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.debug.maxToStringFields' in SparkEnv.conf.

+-------+------------------+------------------+-------------------+------------------+------------------+

|summary| Id| SepalLengthCm| SepalWidthCm| PetalLengthCm| PetalWidthCm|

+-------+------------------+------------------+-------------------+------------------+------------------+

| count| 150| 150| 150| 150| 150|

| mean| 75.5| 5.843333333333335| 3.0540000000000007|3.7586666666666693|1.1986666666666672|

| stddev|43.445367992456916|0.8280661279778637|0.43359431136217375| 1.764420419952262|0.7631607417008414|

| min| 1| 4.3| 2.0| 1.0| 0.1|

| max| 150| 7.9| 4.4| 6.9| 2.5|

+-------+------------------+------------------+-------------------+------------------+------------------+

>>> df\_iris.printSchema()

root

|-- Id: integer (nullable = true)

|-- SepalLengthCm: double (nullable = true)

|-- SepalWidthCm: double (nullable = true)

|-- PetalLengthCm: double (nullable = true)

|-- PetalWidthCm: double (nullable = true)

|-- Species: string (nullable = true)

# Select features

>>> vecAssembler = VectorAssembler(inputCols=[“SepalLengthCm”, “SepalWidthCm”, “PetalLengthCm”, “PetalWidthCm”, outputCol=”features”])

# Change to Dataframe format

>>> df\_features = vecAssembler.transform(df\_iris)

>>> df\_features.show(5)

+---+-------------+------------+-------------+------------+-----------+-----------------+

| Id|SepalLengthCm|SepalWidthCm|PetalLengthCm|PetalWidthCm| Species| features|

+---+-------------+------------+-------------+------------+-----------+-----------------+

| 1| 5.1| 3.5| 1.4| 0.2|Iris-setosa|[5.1,3.5,1.4,0.2]|

| 2| 4.9| 3.0| 1.4| 0.2|Iris-setosa|[4.9,3.0,1.4,0.2]|

| 3| 4.7| 3.2| 1.3| 0.2|Iris-setosa|[4.7,3.2,1.3,0.2]|

| 4| 4.6| 3.1| 1.5| 0.2|Iris-setosa|[4.6,3.1,1.5,0.2]|

| 5| 5.0| 3.6| 1.4| 0.2|Iris-setosa|[5.0,3.6,1.4,0.2]|

+---+-------------+------------+-------------+------------+-----------+-----------------+

# Transfer species into number (label encoding)

>>> stringIndexer = StringIndexer(inputCol= "Species", outputCol= "label")

>>> df\_features\_label = stringIndexer.fit(df\_features).transform(df\_features)

>>> df\_features\_label.show(5)

+---+-------------+------------+-------------+------------+-----------+-----------------+-----+

| Id|SepalLengthCm|SepalWidthCm|PetalLengthCm|PetalWidthCm| Species| features|label|

+---+-------------+------------+-------------+------------+-----------+-----------------+-----+

| 1| 5.1| 3.5| 1.4| 0.2|Iris-setosa|[5.1,3.5,1.4,0.2]| 0.0|

| 2| 4.9| 3.0| 1.4| 0.2|Iris-setosa|[4.9,3.0,1.4,0.2]| 0.0|

| 3| 4.7| 3.2| 1.3| 0.2|Iris-setosa|[4.7,3.2,1.3,0.2]| 0.0|

| 4| 4.6| 3.1| 1.5| 0.2|Iris-setosa|[4.6,3.1,1.5,0.2]| 0.0|

| 5| 5.0| 3.6| 1.4| 0.2|Iris-setosa|[5.0,3.6,1.4,0.2]| 0.0|

+---+-------------+------------+-------------+------------+-----------+-----------------+-----+

>>> df\_features\_label.select('label').distinct().show()

+-----+

|label|

+-----+

| 0.0|

| 1.0|

| 2.0|

+-----+

# Separate dataset to 80% and 20% for training and testing

>>> df\_train, df\_test = df\_features\_label.randomSplit([.8, .2])

>>> df\_train.count()

122

>>> df\_train.show(3)

+---+-------------+------------+-------------+------------+-----------+-----------------+-----+

| Id|SepalLengthCm|SepalWidthCm|PetalLengthCm|PetalWidthCm| Species| features|label|

+---+-------------+------------+-------------+------------+-----------+-----------------+-----+

| 1| 5.1| 3.5| 1.4| 0.2|Iris-setosa|[5.1,3.5,1.4,0.2]| 0.0|

| 2| 4.9| 3.0| 1.4| 0.2|Iris-setosa|[4.9,3.0,1.4,0.2]| 0.0|

| 4| 4.6| 3.1| 1.5| 0.2|Iris-setosa|[4.6,3.1,1.5,0.2]| 0.0|

+---+-------------+------------+-------------+------------+-----------+-----------------+-----+

**Naïve Bayes method**

# Naïve Bayes: simple multiclass classification algorithm

# with the assumption of independence between every pair of features.

>>> nb = NaiveBayes(featuresCol="features", labelCol="label")

>>> nb\_model = nb.fit(df\_train)

>>> nb.model

NaiveBayes\_40f3b8bd20f26704047c

# Model was built, use transform to predict the test dataset

>>> df\_predicted = nb\_model.transform(df\_test.select('features','label'))

>>> df\_predicted.show(3)

+-----------------+-----+--------------------+--------------------+----------+

| features|label| rawPrediction| probability|prediction|

+-----------------+-----+--------------------+--------------------+----------+

|[4.7,3.2,1.3,0.2]| 0.0|[-11.147123569798...|[0.71509340155031...| 0.0|

|[4.4,2.9,1.4,0.2]| 0.0|[-10.798090391157...|[0.64835530232266...| 0.0|

|[4.3,3.0,1.1,0.1]| 0.0|[-9.8973041771793...|[0.73234440737539...| 0.0|

+-----------------+-----+--------------------+--------------------+----------+

only showing top 3 rows

# Evaluate model

>>> evaluator = MulticlassclassificationEvaluator()

>>> evaluator

MulticlassClassificationEvaluator\_46648c1f55bb5436c82d

# the numbers above usually indicate that it is an object

>>> evaluator.evaluate(df\_predicted)

0.8587301587301588

**Decision tree model**

>>> from pyspark.ml.classification import DecisionTreeClassifier

>>> dtree = DecisionTreeClassifier(maxDepth=5, featuresCol=”features”, labelCol=”label”)

>>> dtree\_model = dtree.fit(df\_train)

>>> dtree\_model.numNodes

17

>>> dtree\_model.depth

5

>>> result2 = dtree\_model.transform(test0).head() # test0 is defined in the last session

>>> resutl2

Row(features=DenseVector([5.1, 3.5, 1.4, 0.2]),

rawPrediction=DenseVector([41.0, 0.0, 0.0]),

probability=DenseVector([1.0, 0.0, 0.0]), prediction=0.0)

>>> result2.prediction

0.0

# Tree model

# Visualization of the tree model is an ongoing work in open resource community

>>> dtree\_model.toDebugString

u'DecisionTreeClassificationModel (uid=DecisionTreeClassifier\_4fccb9229772430bcccc) of depth 5 with 17 nodes\n If (feature 2 <= 1.9)\n Predict: 0.0\n Else (feature 2 > 1.9)\n If (feature 3 <= 1.7)\n If (feature 2 <= 4.9)\n If (feature 3 <= 1.6)\n Predict: 1.0\n Else (feature 3 > 1.6)\n Predict: 2.0\n Else (feature 2 > 4.9)\n If (feature 3 <= 1.5)\n Predict: 2.0\n Else (feature 3 > 1.5)\n If (feature 0 <= 6.7)\n Predict: 1.0\n Else (feature 0 > 6.7)\n Predict: 2.0\n Else (feature 3 > 1.7)\n If (feature 2 <= 4.8)\n If (feature 0 <= 5.9)\n Predict: 1.0\n Else (feature 0 > 5.9)\n Predict: 2.0\n Else (feature 2 > 4.8)\n Predict: 2.0\n'

**(3) Save and load model**

# Save the model

# Define a HDFS path first

>>> output\_dir = “hdfs:///user/aisjiaw/models/myNaiveBayesModel2”

>>> nb\_model.save(output\_dir)

# Read back the model

>>> from pyspark.ml.classification import NaiveBayesModel

>>> nb\_model2 = NaiveBayesModel.load(output\_dir)

>>> nb\_model2

NaiveBayes\_40f3b8bd20f26704047c

**Apply model**

# Use the load back model to predict the new data test0

>>> from pyspark.sql import Row

>>> from pyspark.ml.linalg import Vectors

>>> test0 = sc.parallelize([Row(features=Vectors.dense([5.1, 3.5, 1.4, 0.2]))]).toDF()

>>> result = nb\_model2.transform(test0).head()

>>> print result

Row(features=DenseVector([5.1, 3.5, 1.4, 0.2]),

rawPrediction=DenseVector([-12.0427, -13.5234, -14.026]),

probability=DenseVector([0.7326, 0.1666, 0.1008]), prediction=0.0)

>>> result.prediction

0.0