1.1

The dataset is movie.csv, which list user_Id, movie_Id and ratings columns. The size of the movie.csv is around 250KB, and is used to judge movie's rating.

```
import findspark
import random
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.regression import LinearRegression
from pyspark.sql import Row
from pyspark.sql import SparkSession
findspark.init()
```

```
spark = SparkSession.builder.appName("ALS-hw1").getOrCreate()
ratings_file = spark.read.text("hdfs://localhost:1234/user/hz2558/movie.csv").rdd
ratings_header = ratings_file.take(1)[0]
ratings_data = ratings_file.filter(lambda line: line!=ratings_header).map(lambda row: row.value.split(","))
ratingsRDD = ratings_data.map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]),
                                     rating=int(p[2])))
ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol = 'movieId', ratingCol="rating", coldStartStrategy="drop")
model = als.fit(training)

predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

userRecs = model.recommendForAllUsers(3)
userRecs.show()
movieRecs = model.recommendForAllItems(3)
movieRecs.show()

user = ratings.select(als.getUserCol()).distinct().limit(1)
userSubsetRecs = model.recommendForUserSubset(user, 5)
userSubsetRecs.show()
movie = ratings.select(als.getItemCol()).distinct().limit(1)
movieSubSetRecs = model.recommendForItemSubset(movie, 5)
movieSubSetRecs.show()
```

```
Root-mean-square error = 5.547845823289263
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|   471|[[2096673, 1.3182...|
|   463|[[1872194, 16.494...|
|   833|[[1392214, 8.9986...|
|   496|[[114369, 12.4466...|
|   148|[[120737, 13.0918...|
|   540|[[1291580, 12.562...|
|   392|[[120737, 6.61183...|
|   243|[[2692904, 18.729...|
|   623|[[2357129, 7.9987...|
|  1025|[[2234003, 15.300...|
|   737|[[2872718, 10.074...|
|   897|[[109830, 17.2875...|
|   858|[[109830, 6.99652...|
|    31|[[60196, 9.096024...|
|   516|[[120737, 8.99967...|
|   580|[[1821549, 11.141...|
|   451|[[120737, 4.25920...|
|    85|[[482571, 9.75730...|
|   137|[[1815862, 9.8202...|
|   808|[[99685, 10.38100...|
+------+--------------------+
only showing top 20 rows
```

```
+-------+--------------------+
|movieId|     recommendations|
+-------+--------------------+
|1706620|[[816, 37.3444], ...|
|  46521|[[486, 21.460392]...|
|2114461|[[227, 37.6526], ...|
| 105665|[[816, 29.854437]...|
| 379725|[[486, 38.93641],...|
|2258345|[[565, 21.74422],...|
|1843866|[[486, 38.00049],...|
|2818178|[[486, 20.239824]...|
|  64519|[[227, 33.207985]...|
| 133189|[[227, 28.722837]...|
| 101761|[[816, 22.20648],...|
| 436331|[[816, 11.905194]...|
|1068641|[[816, 20.22474],...|
| 804443|[[93, 2.6473155],...|
| 120524|[[969, 10.9380665...|
| 878804|[[565, 16.043749]...|
| 462465|[[816, 20.45872],...|
| 120749|[[217, 5.978611],...|
| 373469|[[486, 6.7921934]...|
|1370429|[[794, 7.7651777]...|
+-------+--------------------+
only showing top 20 rows

+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|    26|[[2345737, 8.9996...|
+------+--------------------+

+-------+--------------------+
|movieId|     recommendations|
+-------+--------------------+
|1972571|[[227, 24.994196]...|
+-------+--------------------+
```

1.2

The dataset is books.csv, which list user_Id, book_Id and ratings columns. The size of the books.csv is around 12.4MB, and is used to judge book's rating.

```
spark = SparkSession.builder.appName("ALS-hw1").getOrCreate()
ratings_file = spark.read.text("hdfs://localhost:1234/user/hz2558/books.csv").rdd
ratings_header = ratings_file.take(1)[0]
ratings_data = ratings_file.filter(lambda line: line!=ratings_header).map(lambda row: row.value.split(","))
ratingsRDD = ratings_data.map(lambda p: Row(userId=int(p[1]), bookId=int(p[0]), rating=int(p[2])))
ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])

als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol = 'bookId', ratingCol="rating", coldStartStrategy="drop")
model = als.fit(training)
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

userRecs = model.recommendForAllUsers(3)
userRecs.show()
bookRecs = model.recommendForAllItems(3)
bookRecs.show()

user = ratings.select(als.getUserCol()).distinct().limit(1)
userSubsetRecs = model.recommendForUserSubset(user, 5)
userSubsetRecs.show()
book = ratings.select(als.getItemCol()).distinct().limit(1)
bookSubSetRecs = model.recommendForItemSubset(book, 5)
bookSubSetRecs.show()
```

```
Root-mean-square error = 1.7447290867538077
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|   148|[[5413, 9.315692]...|
|   463|[[8451, 10.323016...|
|   471|[[8372, 5.464236]...|
|   496|[[8129, 11.487731...|
|   833|[[7191, 9.279278]...|
|  1088|[[9011, 9.470474]...|
|  1238|[[8022, 8.875776]...|
|  1342|[[9624, 9.623194]...|
|  1580|[[7685, 7.690616]...|
|  1591|[[2100, 6.2039547...|
|  1645|[[6247, 6.038144]...|
|  1829|[[9664, 7.592435]...|
|  1959|[[9486, 5.4087696...|
|  2122|[[9479, 9.898839]...|
|  2142|[[9320, 6.849844]...|
|  2366|[[9111, 9.9902725...|
|  2659|[[9775, 5.7152743...|
|  2866|[[5493, 13.200363...|
|  3175|[[6089, 12.766644...|
|  3749|[[8903, 11.386503...|
+------+--------------------+
only showing top 20 rows
```

```
+------+--------------------+
|bookId|     recommendations|
+------+--------------------+
|  1580|[[27570, 13.19890...|
|  4900|[[27570, 10.87201...|
|  5300|[[6034, 14.540917...|
|  6620|[[34427, 13.25181...|
|  7240|[[24100, 26.09579...|
|  7340|[[15412, 14.10536...|
|  7880|[[15225, 11.27574...|
|  9900|[[19379, 17.99734...|
|   471|[[27570, 15.42034...|
|  1591|[[27570, 15.32350...|
|  4101|[[30545, 11.55127...|
|  1342|[[27570, 17.84894...|
|  2122|[[49136, 11.68425...|
|  2142|[[49136, 15.76191...|
|  7982|[[35338, 15.39753...|
|  8592|[[12183, 13.56327...|
|  9852|[[50248, 13.88689...|
|   463|[[27570, 17.01122...|
|   833|[[27570, 13.01139...|
|  5803|[[12183, 10.58484...|
+------+--------------------+
only showing top 20 rows
```

```
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
| 11945|[[5493, 7.288356]...|
+------+--------------------+
```

```
+------+--------------------+
|bookId|     recommendations|
+------+--------------------+
|    26|[[27570, 15.06262...|
+------+--------------------+
```

1.3

The dataset is anime.csv, which list anime_Id, user_Id and ratings columns. The size of the movie.csv is around 11.6MB, and is used to judge anime's rating.

```
spark = SparkSession.builder.appName("ALS-hw1").getOrCreate()
ratings_file = spark.read.text("hdfs://localhost:1234/user/hz2558/anime.csv").rdd
ratings_header = ratings_file.take(1)[0]
ratings_data = ratings_file.filter(lambda line: line!=ratings_header).map(lambda row: row.value.split(","))
ratingsRDD = ratings_data.map(lambda p: Row(userId=int(p[0]), animeId=int(p[0]), rating=int(p[2])))
ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])

als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol = 'animeId',   ratingCol="rating", coldStartStrategy="drop
model = als.fit(training)
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",        predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

userRecs = model.recommendForAllUsers(3)
userRecs.show()
animeRecs = model.recommendForAllItems(3)
animeRecs.show()

user = ratings.select(als.getUserCol()).distinct().limit(1)
userSubsetRecs = model.recommendForUserSubset(user, 5)
userSubsetRecs.show()
anime = ratings.select(als.getItemCol()).distinct().limit(1)
animeSubSetRecs = model.recommendForItemSubset(anime, 5)
animeSubSetRecs.show()
```

```
Root-mean-square error = 1.3711564237069782
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|  1580|[[1580, 8.887652]...|
|  4900|[[4900, 8.998778]...|
|  5300|[[5300, 9.55799],...|
|  6620|[[6620, 8.779998]...|
|  7240|[[7240, 9.9989], ...|
|  7340|[[3955, 8.250226]...|
|  7880|[[6432, 8.374762]...|
|  9900|[[9900, 9.9989], ...|
|   471|[[4766, 8.143131]...|
|  1591|[[3502, 8.297211]...|
|  4101|[[8111, 9.059622]...|
|  1342|[[1342, 9.362462]...|
|  2122|[[2122, 8.855902]...|
|  2142|[[2142, 9.998902]...|
|  7982|[[9822, 8.812114]...|
|  8592|[[8592, 8.498707]...|
|  9852|[[9852, 7.67714],...|
|   463|[[6016, 8.499827]...|
|   833|[[833, 8.561216],...|
|  5803|[[9790, 7.948205]...|
+------+--------------------+
only showing top 20 rows
```

```
+-------+--------------------+
|animeId|     recommendations|
+-------+--------------------+
|   1580|[[1580, 8.887652]...|
|   4900|[[4900, 8.998778]...|
|   5300|[[5300, 9.55799],...|
|   6620|[[6620, 8.779998]...|
|   7240|[[7240, 9.9989], ...|
|   7340|[[7340, 7.4417863...|
|   7880|[[7880, 7.663936]...|
|   9900|[[9900, 9.9989], ...|
|    471|[[471, 6.9150786]...|
|   1591|[[1591, 6.9889946...|
|   4101|[[4101, 6.2141423...|
|   1342|[[1342, 9.362462]...|
|   2122|[[2122, 8.855902]...|
|   2142|[[2142, 9.998902]...|
|   7982|[[7982, 8.156548]...|
|   8592|[[8592, 8.498707]...|
|   9852|[[9852, 7.67714],...|
|    463|[[463, 8.47143], ...|
|    833|[[833, 8.561216],...|
|   5803|[[5803, 7.742171]...|
+-------+--------------------+
only showing top 20 rows

+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|    26|[[26, 8.998779], ...|
+------+--------------------+

+-------+--------------------+
|animeId|     recommendations|
+-------+--------------------+
|     26|[[26, 8.998779], ...|
+-------+--------------------+
```

2.1

The dataset is abalone.txt. The dataset is to predict the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the age. Further information, such as weather patterns and location (hence food availability) may be required to solve the problem. The size is 279KB

```
import findspark
findspark.init
from pyspark.sql import SparkSession
```

```
from pyspark.ml.regression import LinearRegression
spark = SparkSession.builder.appName("LinearRegression").getOrCreate()
lr_training = spark.read.format("libsvm").load("hdfs://localhost:1234/user/hz2558/regression.txt")

lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
lrModel = lr.fit(lr_training)
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))

trainingSummary = lrModel.summary
print("numIterations: %d" % trainingSummary.totalIterations)
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
trainingSummary.residuals.show()
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
```

```
Coefficients: [-0.10992653682444373,-0.0,0.8206144807830859,8.882098162004942,0.0,0.0,0.0,9.152366318768674]
Intercept: 6.38886342311966
numIterations: 11
objectiveHistory: [0.5, 0.4512890716752172, 0.37167888074274397, 0.3640336879910565, 0.3593702890691347, 0.3581499062
8690463, 0.356216558130987, 0.3538543269280125, 0.3509533614546618, 0.34816820384107205, 0.3476613953001989]
+--------------------+
|           residuals|
+--------------------+
|   6.204884555013187|
| -0.9364542005969856|
| -0.6347486102117568|
|  0.8926597785591959|
|-0.48228850573868876|
|-0.24735144052396407|
|   9.137839031509841|
|   5.992350983065943|
|-0.20296695703240708|
|   8.208847332677951|
|  4.3536654782095425|
|  1.2212477945763887|
|  1.4711967585704961|
| 0.33450145697378275|
|   0.958273924678533|
|  2.0415766438264242|
| -1.0963563376944112|
|  1.4739632894225565|
|  -1.146822642963489|
|  0.5177345369953041|
+--------------------+
only showing top 20 rows

RMSE: 2.514112
r2: 0.391812
```

2.2

```
from pyspark.ml.regression import GeneralizedLinearRegression
spark = SparkSession.builder.appName("GeneralizeLinearRegression").getOrCreate()
glr_training = spark.read.format("libsvm").load("hdfs://localhost:1234/user/hz2558/regression.txt")

glr = GeneralizedLinearRegression(family="gaussian", link="identity", maxIter=10, regParam=0.3)
glr_model = glr.fit(glr_training)
print("Coefficients: " + str(glr_model.coefficients))
print("Intercept: " + str(glr_model.intercept))

summary = glr_model.summary
print("Coefficient Standard Errors: " + str(summary.coefficientStandardErrors))
print("T Values: " + str(summary.tValues))
print("P Values: " + str(summary.pValues))
print("Dispersion: " + str(summary.dispersion))
print("Null Deviance: " + str(summary.nullDeviance))
print("Residual Degree Of Freedom Null: " + str(summary.residualDegreeOfFreedomNull))
print("Deviance: " + str(summary.deviance))
print("Residual Degree Of Freedom: " + str(summary.residualDegreeOfFreedom))
print("AIC: " + str(summary.aic))
print("Deviance Residuals: ")
summary.residuals().show()
```

```
Coefficients: [-0.3870075551097861,2.4151226859199495,5.950512809801432,13.354711640650674,0.6590316066778111,-6.4653
03019802343,-1.427561397501458,11.36328983768107]
Intercept: 4.455813792619174
Coefficient Standard Errors: [0.046195477856590936, 0.7334872767968283, 0.8938897774362811, 1.3691828521684388, 0.202
98229283981378, 0.3611391759351124, 0.741555346442586, 0.5562879810940599, 0.2716874623844711]
T Values: [-8.377606923154056, 3.2926579128501015, 6.656875332960837, 9.753782425408113, 3.2467443216729, -17.9025247
06884737, -1.9250908301717506, 20.426991457433104, 16.400513124575657]
P Values: [0.0, 0.0010007264088354173, 3.160649519884373e-11, 0.0, 0.001176571373496138, 0.0, 0.05428597001971869, 0.
0, 0.0]
Dispersion: 5.345879170446775
Null Deviance: 43410.63059612122
Residual Degree Of Freedom Null: 4176
Deviance: 22281.624382422156
Residual Degree Of Freedom: 4168
AIC: 18866.816563646706
Deviance Residuals:
+--------------------+
|   devianceResiduals|
+--------------------+
|   5.944086666566417|
|  -0.924416657531677|
| -1.2360202588497629|
|   0.481908216223232|
|   0.197411563637389|
| 0.05192962622147679|
|    8.03751076471049|
|   5.457280439549802|
| -0.7373988738428281|
|   7.391594508453508|
|   3.598247966686788|
|  0.7379530122593323|
|  0.7096337625716629|
|-0.09495367485739514|
|  0.5174692430429104|
|  1.3001125636213136|
| -0.7810883079818023|
|  1.4621256921933021|
|  -1.390296254274542|
|  0.2562851868364504|
+--------------------+
only showing top 20 rows
```

2.3

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

spark = SparkSession.builder.appName("GeneralizeLinearRegression").getOrCreate()
data = spark.read.format("libsvm").load("hdfs://localhost:1234/user/hz2558/regression.txt")

featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)
(trainingData, testData) = data.randomSplit([0.7, 0.3])
dt = DecisionTreeRegressor(featuresCol="indexedFeatures")
pipeline = Pipeline(stages=[featureIndexer, dt])
model = pipeline.fit(trainingData)


predictions = model.transform(testData)
predictions.select("prediction", "label", "features").show(5)

evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

treeModel = model.stages[1]
print(treeModel)
```

```
+-----------------+-----+--------------------+
|       prediction|label|            features|
+-----------------+-----+--------------------+
|4.305084745762712|  1.0|(8,[0,1,2,3,4,5,6...|
|4.305084745762712|  3.0|(8,[0,1,2,3,4,5,6...|
|4.305084745762712|  3.0|(8,[0,1,2,3,4,5,6...|
|4.305084745762712|  4.0|(8,[0,1,2,3,4,5,6...|
|4.305084745762712|  4.0|(8,[0,1,2,3,4,5,6...|
+-----------------+-----+--------------------+
only showing top 5 rows

Root Mean Squared Error (RMSE) on test data = 2.45578
DecisionTreeRegressionModel (uid=DecisionTreeRegressor_4a878dd0b11a96b4575e) of depth 5 with 63 nodes
```

3.1

This data is used in a competition on click-through rate prediction jointly hosted by Avazu and Kaggle in 2014. The participents were asked to learn a model from the first 10 days of advertising log, and predict the click probability for the impressions on the 11th day. The size of the dataset is 78KB

```
import findspark
findspark.init
from pyspark.sql import SparkSession
```

```
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
spark = SparkSession.builder.appName("NaiveBayes").getOrCreate()
data = spark.read.format('libsvm').load("hdfs://localhost:1234/user/hz2558/classification.txt")
splits = data.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

# train the model
model = nb.fit(train)

# select example rows to display.
predictions = model.transform(test)
predictions.show()

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
                                              metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
```

```
+-----+--------------------+--------------------+--------------------+----------+
|label|            features|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
| -1.0|(8,[0,1,2,3,4,5,6...|[-615.57516107906...|[0.95429810014249...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-768.71498040481...|[0.99967626982525...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-640.86097899419...|[0.22544679206137...|       1.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-734.12243049127...|[0.99999571380276...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-791.65228999158...|[0.99999989055780...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-685.01056466404...|[3.71691370447461...|       1.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-724.15064761226...|[0.99357059252912...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-837.18025466998...|[0.99991765926265...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-693.12839704668...|[0.99974350429603...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-1044.5804245912...|[1.0,1.7105400467...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-1437.8329958703...|[1.0,3.7758347997...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-544.01134741778...|[0.55849638642155...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-460.85281472169...|[0.02359390886660...|       1.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-624.22437827810...|[0.04167311283376...|       1.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-659.96076551298...|[0.98616532094627...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-751.68220423846...|[0.99910448848787...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-874.95168227221...|[0.99999527521588...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-1248.2723193387...|[1.0,1.8700854163...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-927.92468743731...|[0.99999120006648...|       0.0|
| -1.0|(8,[0,1,2,3,4,5,6...|[-733.80373274485...|[0.00787356507318...|       1.0|
+-----+--------------------+--------------------+--------------------+----------+
only showing top 20 rows

Test set accuracy = 0.4673913043478261
```

3.2

```
import findspark
findspark.init
from pyspark.sql import SparkSession
```

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
spark = SparkSession.builder.appName("RandomForest").getOrCreate()
data = spark.read.format('libsvm').load("hdfs://localhost:1234/user/hz2558/classification.txt")
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)
(trainingData, testData) = data.randomSplit([0.7, 0.3])


rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=10)
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf, labelConverter])
model = pipeline.fit(trainingData)
predictions = model.transform(testData)
predictions.select("predictedLabel", "label", "features").show(5)


evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))

rfModel = model.stages[2]
print(rfModel)  # summary only
```

```
+--------------+-----+--------------------+
|predictedLabel|label|            features|
+--------------+-----+--------------------+
|          -1.0| -1.0|(8,[0,1,2,3,4,5,6...|
|           1.0| -1.0|(8,[0,1,2,3,4,5,6...|
|          -1.0| -1.0|(8,[0,1,2,3,4,5,6...|
|          -1.0| -1.0|(8,[0,1,2,3,4,5,6...|
|          -1.0| -1.0|(8,[0,1,2,3,4,5,6...|
+--------------+-----+--------------------+
only showing top 5 rows

Test Error = 0.248848
RandomForestClassificationModel (uid=RandomForestClassifier_49b9987d6309c822ab5a) with 10 trees
```

3.3

```
import findspark
findspark.init
from pyspark.sql import SparkSession
```

```
from pyspark.ml import Pipeline
from pyspark.sql import Row
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
spark = SparkSession.builder.appName("DecisionTree").getOrCreate()
data = spark.read.format('libsvm').load("hdfs://localhost:1234/user/hz2558/classification.txt")

labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)
(trainingData, testData) = data.randomSplit([0.7, 0.3])
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])
model = pipeline.fit(trainingData)

predictions = model.transform(testData)
predictions.select("prediction", "indexedLabel", "features").show(5)
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g " % (1.0 - accuracy))
treeModel = model.stages[2]
print(treeModel)
```

```
+----------+------------+--------------------+
|prediction|indexedLabel|            features|
+----------+------------+--------------------+
|       0.0|         1.0|(8,[0,1,2,3,4,5,6...|
|       0.0|         1.0|(8,[0,1,2,3,4,5,6...|
|       0.0|         1.0|(8,[0,1,2,3,4,5,6...|
|       1.0|         1.0|(8,[0,1,2,3,4,5,6...|
|       0.0|         1.0|(8,[0,1,2,3,4,5,6...|
+----------+------------+--------------------+
only showing top 5 rows

Test Error = 0.275424
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4dea8e1e7d80d3d19f9e) of depth 5 with 53 nodes
```