Lab4- Spark

CS5590 BigData Programming - Lab Assignment 4 .

Team Id: 3 .

Member 1: Raju Nekadi .

Class Id: 7 .

Member 2: Sushma Manne

Class Id: 3 .

Raju's GitHub Link:

https://github.com/rnekadi/CSEE5590_BIGDATA_PROGAMMING_Fall2018/tree/master/Lab4 .

Video Link

https://youtu.be/HsXkKkyaiNw

Contribution:

Raju Nekadi : Question 1 50%

Sushma Manne : Question 2 50% .

Introduction:

In Lab4, We will the Word Count on Twitter Data on real time and machine learning algorithm on Absenteeism dataset.

Objective:

To Perform Twitter Word Count analysis.

Approaches:

Spark Streaming API consumes the tweets from Twitter. We can apply the transformation on stream data to push further downstream.

We will use the Spark Streaming to connect with Twitter.

We need to get the consumer token, access token from twitter.

Spark Streaming creates the streams of data and pass it to downstream. Then we will use map and reduce by transformation, action to perform word count.

Dataset:

Not applicable as we are consuming data directly from twitter.

Workflow:

First, of all we will create the Spark Config and Context variable and then we will pass our consumerKey, consumerSecret, accessToken, accessTokenSecret through arguments as shown below.

```
object TwitterWordCount {

  val conf = new SparkConf().setMaster("local[4]").setAppName("Spark Streaming - PopularHashTags")
  val sc = new SparkContext(conf)

  def main(args: Array[String]) : Unit  {

    sc.setLogLevel("WARN")

    val Array(consumerKey, consumerSecret, accessToken, accessTokenSecret) = args.take(4)
    val filters = args.takeRight(args.length - 4)

    // Set the system properties so that Twitter4j library used by twitter stream
    // can use them to generat OAuth credentials
    System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
    System.setProperty("twitter4j.oauth.consumerSecret", consumerSecret)
    System.setProperty("twitter4j.oauth.accessToken", accessToken)
    System.setProperty("twitter4j.oauth.accessTokenSecret", accessTokenSecret)
```

Then will create the stream of data from TwitterUtil on which apply map and reducebyKey as shown below.

```
    // Set the Spark StreamingContext to create a DStream for every 5 seconds
    val ssc = new StreamingContext(sc, Seconds(5))
    // Pass the filter keywords as arguements

    //  val stream = FlumeUtils.createStream(ssc, args(0), args(1).toInt)
    val stream = TwitterUtils.createStream(ssc, None, filters)

    // Split the stream on space and extract hashtags
    val words = stream.flatMap(status => status.getText.split( regex = " ").filter(_.startsWith("#")))

    // Count each word in each batch
    val pairs = words.map(word => (word, 1))
    val wordCounts = pairs.reduceByKey(_ + _)

    // Print the first ten elements of each RDD generated in this DStream to the console
    wordCounts.print()

    ssc.start()
    ssc.awaitTermination()
```

Evaluation:

Spark Streaming easily handles the streaming of Twitter data and pushing down the further down to count the word in very less time.

```
-------------------------------------------
Time: 1543726680000 ms
-------------------------------------------
(#AndaiNikahanku10M,1)
(#UnlockedIPhoneX,1)

18/12/01 22:58:00 WARN BlockManager: Block input-0-1543726680000 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:00 WARN BlockManager: Block input-0-1543726680400 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:01 WARN BlockManager: Block input-0-1543726680800 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:01 WARN BlockManager: Block input-0-1543726681400 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:02 WARN BlockManager: Block input-0-1543726682400 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:02 WARN BlockManager: Block input-0-1543726682600 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:03 WARN BlockManager: Block input-0-1543726683200 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:03 WARN BlockManager: Block input-0-1543726683400 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:03 WARN BlockManager: Block input-0-1543726683600 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:04 WARN BlockManager: Block input-0-1543726684600 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:05 WARN BlockManager: Block input-0-1543726684800 replicated to only 0 peer(s) instead of 1 peers
-------------------------------------------
Time: 1543726685000 ms
-------------------------------------------
(#atrialfibrillation,1)

18/12/01 22:58:05 WARN BlockManager: Block input-0-1543726685000 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:05 WARN BlockManager: Block input-0-1543726685400 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:06 WARN BlockManager: Block input-0-1543726685800 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:07 WARN BlockManager: Block input-0-1543726687000 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:07 WARN BlockManager: Block input-0-1543726687200 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:07 WARN BlockManager: Block input-0-1543726687400 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:07 WARN BlockManager: Block input-0-1543726687600 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:09 WARN BlockManager: Block input-0-1543726688800 replicated to only 0 peer(s) instead of 1 peers
18/12/01 22:58:09 WARN BlockManager: Block input-0-1543726689400 replicated to only 0 peer(s) instead of 1 peers
-------------------------------------------
Time: 1543726690000 ms
-------------------------------------------
(#iPhone,1)
(#WorldAIDSDay,1)
(#Apple,1)
(#Official髭男dism,1)
```

Conclusion:

Spark Streaming handles Twitter data very swiftly.

Objective:

a.Use the following Classification Algorithms: Naïve Bayes, Decision Tree and Random Forest for the same attribute classification

b.Report the Confusion matrix, Accuracy based on FMeasure, Precision & Recall for all the algorithms

Approaches:

We have used the Absenteeism dataset and performed the Assembler libraries to extract the feature for our models. Then we have deduced the label for output target and tied to assembler features.

Workflow:

We have split the dataset into 70% and 30% ratio. Finally we evaluated the model and prediction on test data. Then we calculated confusion matrix using the above columns and find the precision and recall values.

1.Naive Bayes

```
data = data.withColumn("MOA", data["Month of absence"] - 0).withColumn("label", data['Seasons'] - 0).
    withColumn("ROA", data["Reason for absence"] - 0). \
    withColumn("distance", data["Distance from Residence to Work"] - 0). \
    withColumn("BMI", data["Body mass index"] - 0)

assem = VectorAssembler(inputCols=["label", "MOA"], outputCol='features')

data = assem.transform(data)
# Split the data into train and test
splits = data.randomSplit([0.7, 0.3], 1000)
train = splits[0]
test = splits[1]

# create the trainer and set its parameters
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

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

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

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
                                              metricName="accuracy")

y_true = data.select("BMI").rdd.flatMap(lambda x: x).collect()
y_pred = data.select("ROA").rdd.flatMap(lambda x: x).collect()


accuracy = evaluator.evaluate(predictions)

confusionmatrix = confusion_matrix(y_true, y_pred)

precision = precision_score(y_true, y_pred, average='micro')

recall = recall_score(y_true, y_pred, average='micro')


print("Naive Bayes - Test set accuracy = " + str(accuracy))
```

Invali
The d

2. Random Forest .

```
data = spark.read.load("/Users/sai/Documents/GitHub/ml_data/Absenteeism.csv", format="csv", header=True, delimiter=";")

data = data.withColumn("MOA", data["Month of absence"] - 0).withColumn("label", data['Transportation expense'] - 0). \
    withColumn("ROA", data["Reason for absence"] - 0). \
    withColumn("distance", data["Distance from Residence to Work"] - 0). \
    withColumn("BMI", data["Body mass index"] - 0)

# Index labels, adding metadata to the label column.
# Fit on whole dataset to include all labels in index.

assem = VectorAssembler(inputCols=["label", "distance"], outputCol='features')

data = assem.transform(data)

labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)

# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer =\
    VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)

# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a RandomForest model.
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=10)

# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

y_true = data.select("BMI").rdd.flatMap(lambda x: x).collect()
y_pred = data.select("ROA").rdd.flatMap(lambda x: x).collect()

# Chain indexers and forest in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf, labelConverter])

# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)

# Make predictions.
predictions = model.transform(testData)
```

3. Decision Tree .

```
data = spark.read.load("/Users/sai/Documents/GitHub/ml_data/Absenteeism.csv", format="csv", header=True, delimiter=";")
data = data.withColumn("MOA", data["Month of absence"] - 0).withColumn("label", data['Height'] - 0). \
    withColumn("ROA", data["Reason for absence"] - 0). \
    withColumn("distance", data["Distance from Residence to Work"] - 0). \
    withColumn("BMI", data["Body mass index"] - 0)
#data.show()

assem = VectorAssembler(inputCols=["label", "distance"], outputCol='features')
data = assem.transform(data)

# Index labels, adding metadata to the label column.
# Fit on whole dataset to include all labels in index.
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer =\
    VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)

# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a DecisionTree model.
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

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

# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)

# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("prediction", "indexedLabel", "features").show(5)

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

y_true = data.select("BMI").rdd.flatMap(lambda x: x).collect()
y_pred = data.select("ROA").rdd.flatMap(lambda x: x).collect()
```

Data set and Parameter:

DataSet Can be Found at https://archive.ics.uci.edu/ml/datasets/Absenteeism+at+work .

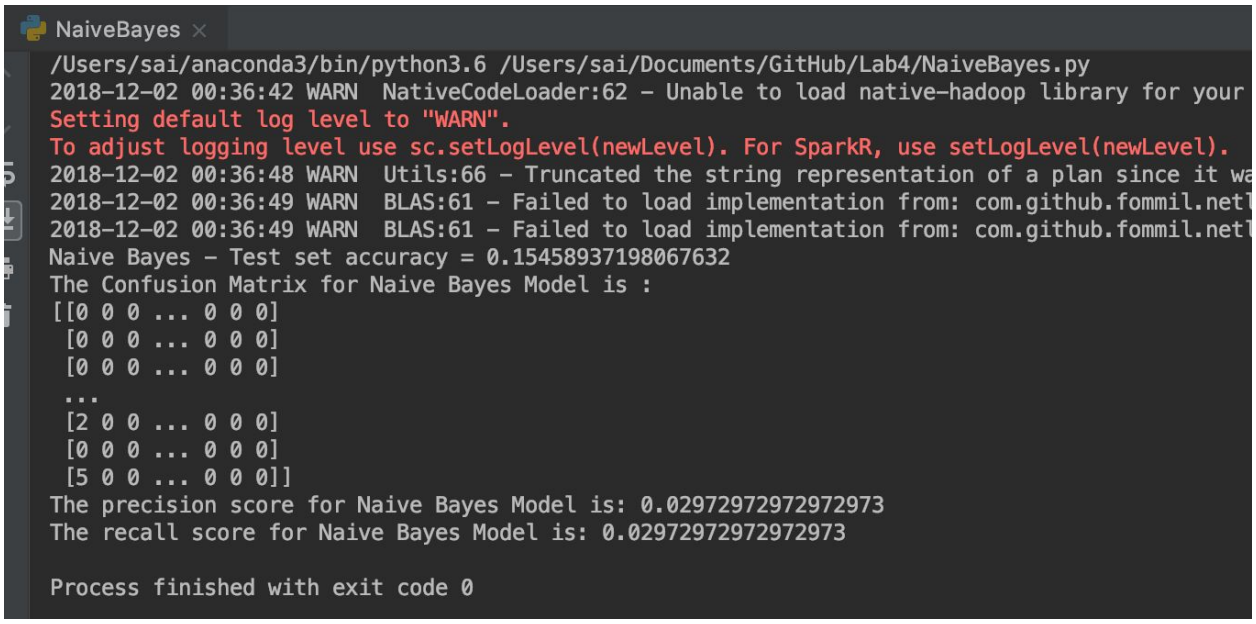
Evaluation .

Accuracy and Error Calculated using the 3 different Classifier and picked the best possible algorithm.

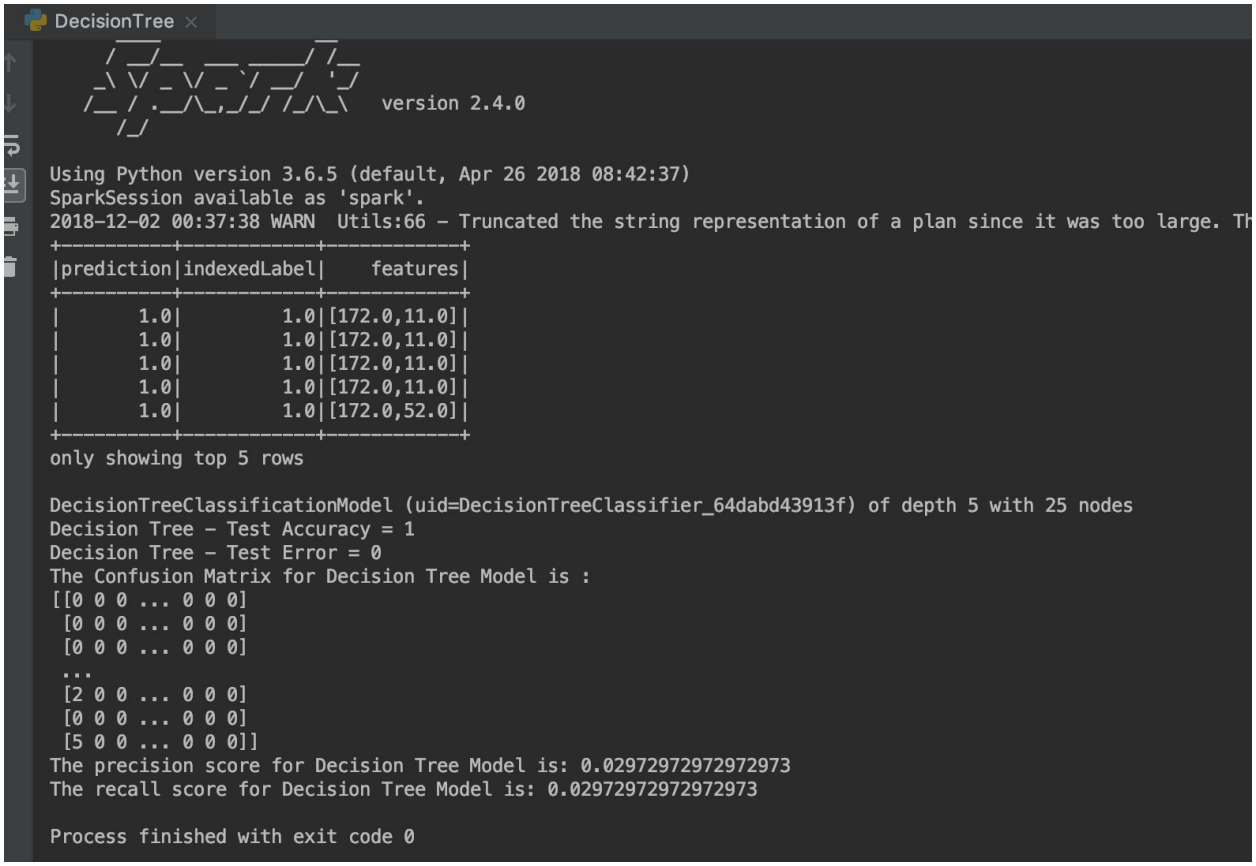
Output:

The output of all 3 algorithm given below.

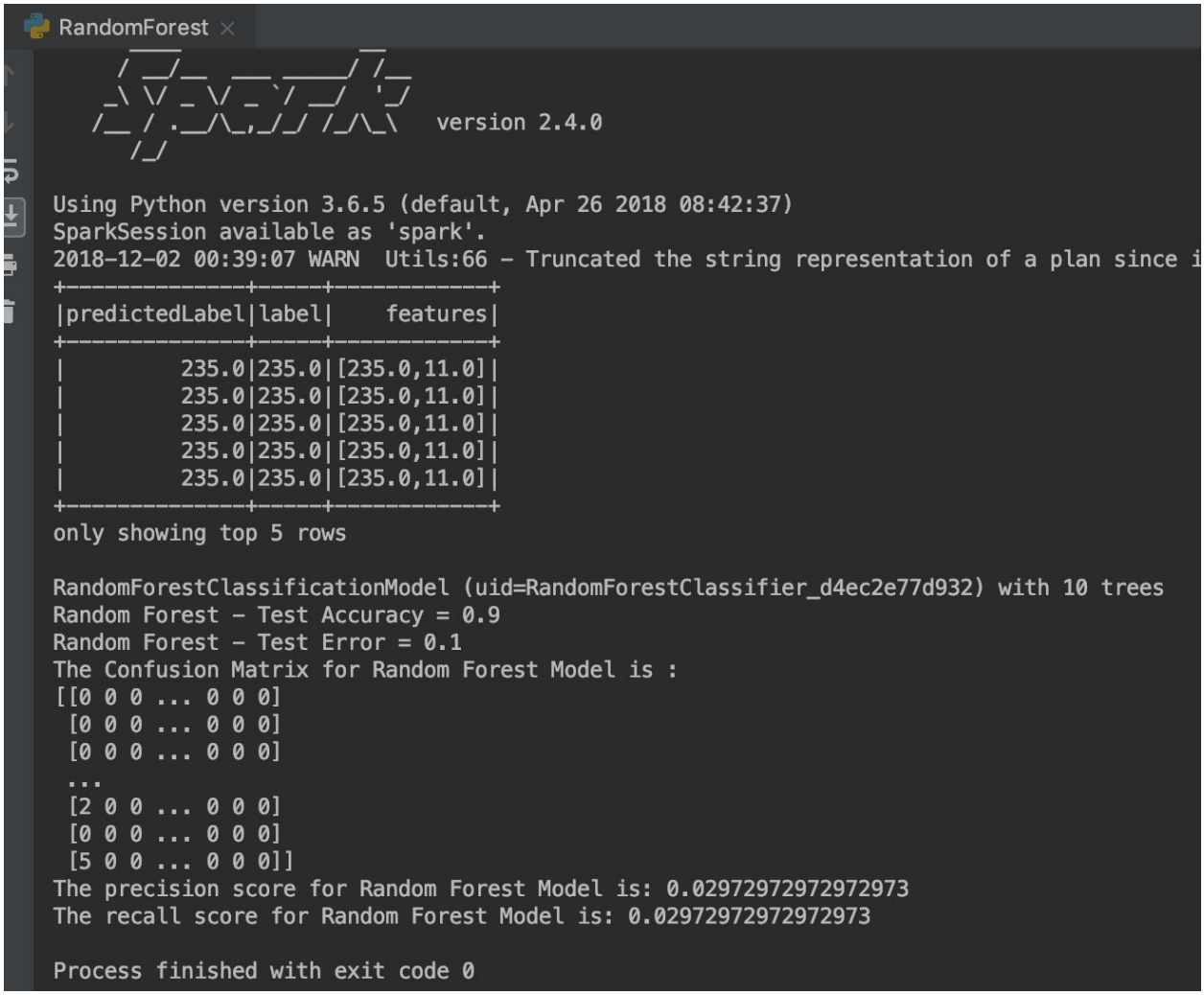
Naive Bayes



Decision Tree



Random Forest



Conclusion:

As per Accuracy parameter Decision Tree has the highest accuracy of 98 % and the lowest error of 2%. Also the confusion matrix is plotted for all the 3 classifications and is shown in the above output images

References:

https://www.programcreek.com/scala/org.apache.spark.streaming.twitter.TwitterUtils