Name: ARPITA CHAKRABORTY
B-Number: B00709870
Project: Classification of news Data.
Technologies: Scala, Apache Spark.

News Text Classification:

The goal of text classification is the classification of documents into a fixed number of predefined categories. In this project, I consider text classification problem for 20 Newsgroups data set using Spark machine learning algorithms. The classification of documents is based on two categories - a binary classification.

Project Objective:

The goal of the project is to train a model to classify documents from the 20 Newsgroups data set into two categories according to whether or not the documents are computer related. Afterwards it will be evaluated and tuned against a test data set with documents that the model was not trained against. Spark MLlib Logistic Regression algorithm has been used to classify the documents into topics. The Logistic Regression method requires a numeric label of type double and cannot work directly with the text topic categories that has been extracted from input dataset. The goal is to classify documents in terms of whether they are computer related or not. The documents that are computer related reside in directories that begin with "comp". I have assigned a numeric column called 'label' with a value of 0 for all non-computer related documents (those without "comp" in the topic name) and a value of 1 for all computer related documents (those with "comp" in the topic name).

Spark MLLib:

MLlib is Spark’s machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs. The work involves exploring how to employ feature extraction transformations and classification algorithms for document classification as well as how to combine these into a single pipeline or workflow. I have utilized the spark.ml package of Spark MLlib as it provides a high-level API built on top of DataFrames, a Spark abstraction layer that provides for a distributed collection of data organized into named columns, for constructing ML pipelines.

Data Set:

The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups, each corresponding to a different topic. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering. In this project, use a subset of the 20 Newsgroups data set consisting of 2000 articles - 100 articles from each of the 20 newsgroups. Some of the newsgroups are very closely related to each other (e.g. comp.sys.ibm.pc.hardware / comp.sys.mac.hardware), while others are highly unrelated (e.g misc.forsale / soc.religion.christian). Here is a list of the 20 newsgroups, partitioned (more or less) according to subject matter:

* comp.graphics
* comp.os.ms-windows.misc
* comp.sys.ibm.pc.hardware
* comp.sys.mac.hardware
* comp.windows.x
* rec.autos
* rec.motorcycles
* rec.sport.baseball
* rec.sport.hockey
* sci.crypt
* sci.electronics
* sci.med
* sci.space
* misc.forsale
* talk.politics.misc
* talk.politics.guns
* talk.politics.mideast
* talk.religion.misc
* alt.atheism
* soc.religion.christian

Acknowledgement: Hettich, S. and Bay, S. D. (1999). The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science.

<u>Steps:</u>

1. Validate Spark context and create a Spark SQL context.
2. Import required machine learning libraries(Spark MlLib).
3. Analyze sample (filepath, content) pair.
4. Extract the document text from the (filepath, content) pair. Validate that just the text was extracted from the (filepath, content) pair. Extract the filename from the full filepath. For example, extract the filename '54200' from the full filepath '/resources/data/mini_newsgroups/talk.politics.guns/54200'.
5. Check that filenames have been extracted from the full filepath.
6. Documents in the data set are stored in directories according to topic. The lowest level directory represents the topic classification.

   Code:
   ```
   topic.distinct().take(20).foreach(println)
   ```

   Output:
   ```
   sci.electronics
   rec.motorcycles
   comp.graphics
   alt.atheism
   comp.windows.x
   ```

```
rec.sport.baseball
talk.politics.mideast
rec.autos
talk.politics.guns
misc.forsale
sci.space
sci.crypt
comp.sys.ibm.pc.hardware
soc.religion.christian
sci.med
talk.politics.misc
talk.religion.misc
comp.sys.mac.hardware
rec.sport.hockey
comp.os.ms-windows.misc
```

7. Put the data into a DataFrame. Define a case class and convert it to a DataFrame
8. Show the DataFrame schema and display 5 rows of the DataFrame.

Code:

```
newsgroups.printSchema()
newsgroups.sample(false,0.005,10L).show(5)
```

Output:

```
root
 |-- id: string (nullable = true)
 |-- text: string (nullable = true)
 |-- topic: string (nullable = true)

+------+--------------------+------------------+
|    id|                text|             topic|
+------+--------------------+------------------+
| 54160|Path: cantaloupe....|       alt.atheism|
|101624|Newsgroups: rec.a...|         rec.autos|
|103667|Newsgroups: rec.a...|         rec.autos|
| 55278|Xref: cantaloupe....|talk.politics.guns|
| 53772|Newsgroups: sci.e...|   sci.electronics|
+------+--------------------+------------------+
only showing top 5 rows
```

9. Show document counts by topic:

Code:

```
 newsgroups.groupBy("topic").count().show()
```

Output:

```
 +--------------------+-----+
|               topic|count|
+--------------------+-----+
|    rec.sport.hockey|  100|
|     sci.electronics|  100|
|             sci.med|  100|
|           rec.autos|  100|
|comp.sys.mac.hard...|  100|
|      comp.windows.x|  100|
|  rec.sport.baseball|  100|
|comp.sys.ibm.pc.h...|  100|
|        misc.forsale|  100|
|     rec.motorcycles|  100|
|           sci.crypt|  100|
|  talk.politics.misc|  100|
|       comp.graphics|  100|
|         alt.atheism|  100|
|talk.politics.mid...|  100|
|soc.religion.chri...|  100|
|  talk.politics.guns|  100|
|comp.os.ms-window...|  100|
|           sci.space|  100|
|  talk.religion.misc|  100|
+--------------------+-----+
```

10. Show only documents that are related to computer topics. That is have "comp" in the topic name using Dataframe API.

Code:

```
newsgroups.filter(newsgroups("topic").like("comp%")).sample(false,
0.01,10L).show(5)
```

Output:

```
+-----+--------------------+--------------------+
|   id|                text|               topic|
+-----+--------------------+--------------------+
| 9519|Newsgroups: comp....|comp.os.ms-window...|
| 9814|Xref: cantaloupe....|comp.os.ms-window...|
|39078|Xref: cantaloupe....|       comp.graphics|
|38839|Newsgroups: comp....|       comp.graphics|
|38244|Newsgroups: comp....|       comp.graphics|
+-----+--------------------+--------------------+
```

11. Train the model using Logistic Regression algorithm using binary classifier. Label is set to 0 for non-computer related topics and 1 for computer related topics.

Code:

```
labelednewsgroups.sample(false,0.003,10L).show(5)

labelednewsgroups.filter(newsgroups("topic").like("comp%")).sample(f
alse,0.007,10L).show(5)
```

Output:

```
+-----+--------------------+--------------------+-----+
|   id|                text|               topic|label|
+-----+--------------------+--------------------+-----+
|55278|Xref: cantaloupe....|  talk.politics.guns|  0.0|
|38839|Newsgroups: comp....|       comp.graphics|  1.0|
|52403|Newsgroups: comp....|comp.sys.mac.hard...|  1.0|
|51996|Path: cantaloupe....|comp.sys.mac.hard...|  1.0|
|83717|Xref: cantaloupe....|  talk.religion.misc|  0.0|
+-----+--------------------+--------------------+-----+
only showing top 5 rows

+-----+--------------------+--------------------+-----+
|   id|                text|               topic|label|
+-----+--------------------+--------------------+-----+
| 9519|Newsgroups: comp....|comp.os.ms-window...|  1.0|
| 9814|Xref: cantaloupe....|comp.os.ms-window...|  1.0|
|39078|Xref: cantaloupe....|       comp.graphics|  1.0|
|38839|Newsgroups: comp....|       comp.graphics|  1.0|
|38244|Newsgroups: comp....|       comp.graphics|  1.0|
+-----+--------------------+--------------------+-----+
```

12. Split data set into separate training (90%) and test (10%) data sets

```
Total Document Count = 2000
Training Count = 1777, 88.85%
Test Count = 223, 11.15%
```

13. **Configure an ML Pipeline:**

In machine learning, it is common to run a sequence of algorithms to process and learn from data. Spark ML represents such a workflow as a Pipeline, which consists of a sequence of PipelineStages (Transformers and Estimators) to be run in a specific order. The pipeline we are using in this example consists of five stages: Tokenizer, StopWordsRemover, HashingTF, Inverse Document Frequency (IDF) and LogisticRegression.

Tokenizer splits the raw text documents into words, adding a new column with words into the dataset.

StopWordsRemover takes as input a sequence of strings and drops all the stop words from the input sequences. Stop words are words which should be excluded from the input, typically because the words appear frequently and don't carry as much meaning. A list of stop words by default. Optionally you can provide a list of stopwords. We will just use the defualt list of stopwords.

HashingTF takes sets of terms and converts those sets into fixed-length feature vectors. Inverse Document Frequency (IDF) is a numerical measure of how much information a term provides. If a term appears very often across the corpus, it means it doesn't carry special information about a particular document. IDF down-weights terms which appear frequently in a corpus.

LogisticRegression is a method used to predict a binary response. The current implementation of logistic regression in spark.ml only supports binary classes. Support for multiclass regression will be added in the future.

Code:
```
val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
val remover = new StopWordsRemover().setInputCol("words").setOutputCol("fi
ltered").setCaseSensitive(false)
val hashingTF = new HashingTF().setNumFeatures(1000).setInputCol("filtered
").setOutputCol("rawFeatures")
val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features").se
tMinDocFreq(0)
val lr = new LogisticRegression().setRegParam(0.01).setThreshold(0.5)
val pipeline = new Pipeline().setStages(Array(tokenizer, remover, hashingT
F, idf, lr))
```

14. Examined the list of default Stop Words that were applied in the pipeline.
15. Fit the pipeline to the training documents.

Code:
```
   val model = pipeline.fit(training)
```

16. Make predictions on document in the Test data set. Run the model against the test dataset.

Code:
```
  val predictions = model.transform(test)
  predictions.select("id", "topic", "probability", "prediction", "label")
.sample(false,0.01,10L).show(5)
   predictions.select("id", "topic", "probability", "prediction", "label")
.filter(predictions("topic").like("comp%")).sample(false,0.1,10L).show(5)
```

Output:
```
  +-----+--------------------+--------------------+----------+-----+
|   id|               topic|         probability|prediction|label|
+-----+--------------------+--------------------+----------+-----+
|54446|  talk.politics.guns|[0.85907386640572...|       0.0|  0.0|
|53911|     sci.electronics|[0.99375677040263...|       0.0|  0.0|
|67516|      comp.windows.x|[0.78541867608477...|       0.0|  1.0|
|51539|comp.sys.mac.hard...|[0.99035383473721...|       0.0|  1.0|
+-----+--------------------+--------------------+----------+-----+

+-----+--------------------+--------------------+----------+-----+
|   id|               topic|         probability|prediction|label|
+-----+--------------------+--------------------+----------+-----+
| 9151|comp.os.ms-window...|[0.99677353549827...|       0.0|  1.0|
|38942|       comp.graphics|[0.41936760457692...|       1.0|  1.0|
|51613|comp.sys.mac.hard...|[0.07059632616872...|       1.0|  1.0|
|60199|comp.sys.ibm.pc.h...|[0.01268251597292...|       1.0|  1.0|
+-----+--------------------+--------------------+----------+-----+
```

17. **Generate an evaluator for the binary classification:**

I have used the area under the ROC curve as the evaluation metric. Receiver operating characteristic (ROC) is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate against the false positive rate at various threshold settings. The ROC curve is thus the sensitivity as a function of fall-out. The area under the ROC curve is useful for comparing and selecting the best machine learning model for a given data set. A model with an area under the ROC curve score near 1 has very good performance. A model with a score near 0.5 is about as good as flipping a coin.

Code:
```
val evaluator = new BinaryClassificationEvaluator().setMetricName("areaUnd
erROC")
println("Area under the ROC curve = " + evaluator.evaluate(predictions))
```

Area under the ROC curve = 0.8736702127659571

18. **Tune Hyperparameters:**

Generate hyperparameter combinations by taking the cross product of some parameter values. Spark MLlib algorithms provide many hyperparameters for tuning models. These hyperparameters are distinct from the model parameters being optimized by MLlib itself. Hyperparameter tuning is accomplished by choosing the best set of parameters based on model performance on test data that the model was not trained with. All combinations of hyperparameters specified will be tried in order to find the one that leads to the model with the best evaluation result.

Build a Parameter Grid specifying what parameters and values will be evaluated in order to determine the best combination.

Code:
```
val paramGrid = new ParamGridBuilder().
  //addGrid(hashingTF.numFeatures, Array(1000, 10000, 100000)).
  //addGrid(idf.minDocFreq, Array(0,10, 100)).
  addGrid(lr.regParam, Array(0.01, 0.1, 0.2)).
  addGrid(lr.threshold, Array(0.5, 0.6, 0.7)).
  build()
```

19. Create a cross validator to tune the pipeline with the generated parameter grid
Spark MLlib provides for cross-validation for hyperparameter tuning. Cross-validation attempts to fit the underlying estimator with user-specified combinations of parameters, cross-evaluate the fitted models, and output the best one.

20. Cross-evaluate the ML Pipeline to find the best model
using the area under the ROC evaluator and hyperparameters specified in the parameter grid

Code:
```
val cv = new CrossValidator().setEstimator(pipeline).setEvaluator(evaluator).
setEstimatorParamMaps(paramGrid).setNumFolds(2)

val cvModel = cv.fit(training)

println("Area under the ROC curve for best fitted model = " + evaluator.ev
aluate(cvModel.transform(test)))
```

```
println("Area under the ROC curve for non-tuned model = " + evaluator.eval
uate(predictions))
println("Area under the ROC curve for fitted model = " + evaluator.evaluat
e(cvModel.transform(test)))
println("Improvement = " + "%.2f".format((evaluator.evaluate(cvModel.trans
form(test)) - evaluator.evaluate(predictions)) *100 / evaluator.evaluate(p
redictions)) + "%")
```

```
Area under the ROC curve for best fitted model = 0.8629110251450667
Area under the ROC curve for non-tuned model = 0.8736702127659574
Area under the ROC curve for fitted model = 0.8629110251450667
Improvement = -1.23%
```

Conclusion:

We can see the developed model using Logistic regression without tuning hyper-parameters is very close to the actual fitted ideal model.

References:

1. https://www.tutorialspoint.com/scala/
2. https://www.tutorialspoint.com/apache_spark/
3. https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc
4. https://spark.apache.org/mllib/
5. http://www.tfidf.com/

Packages to install to the run the program:

Install Java

1. Java -version
2. sudo apt-get update
3. sudo apt-get install default-jdk

Install Scala:

1. sudo apt-get install scala

Install Spark:

1. sudo apt-get install git
2. tar xvf spark-2.0.2-bin-hadoop2.7.tgz
3. cd spark-2.0.2-bin-hadoop2.7.tgz
4. cd bin
5. ./spark-shell