Twitter Sentiment Analysis

Data Preparation

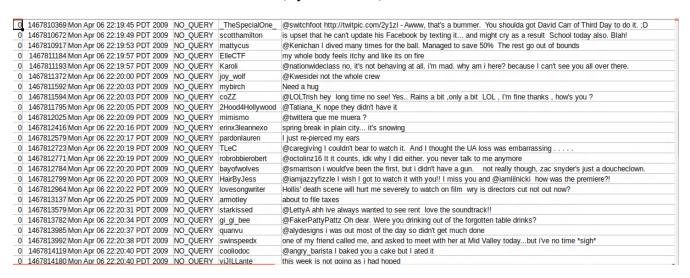
Data is downloaded from

https://www.kaggle.com/kazanova/sentiment140

It contains 1,600,000 tweets.

It contains the following 6 fields:

- 1. target: the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)
- 2. ids: The id of the tweet (*2087*)
- 3. date: the date of the tweet (*Sat May 16 23:58:44 UTC 2009*)
- 4. flag: The query (*lyx*). If there is no query, then this value is NO_QUERY.
- 5. user: the user that tweeted (*robotickilldozr*)
- 6. text: the text of the tweet (*Lyx is cool*)



So firstly we have create schema using case class.

```
case class TweetLabel(label : Int, tweet : String)
```

Then we load the data along with cleaning it. We load only those columns which are needed i.e. label and tweet. We removed special characters, website links, unnecessary spaces at starting and end.

```
val data = sc.textFile("/home/harsh/Desktop/twitter
sentiment/2477_4140_bundle_archive/training.1600000.proces
sed.noemoticon.csv").map(_.split(",")).map(attributes =>
TweetLabel(attributes(0).replace("\"","").toInt,
attributes(5).replace("\"","").toLowerCase()
    .replaceAll("\n", "")
    .replaceAll("rt\\s+", "")
    .replaceAll("\\s+@\\w+", "")
    .replaceAll("@\\w+", "")
    .replaceAll("\\s+#\\w+", "")
    .replaceAll("#\\w+", "")
    .replaceAll("(?:https?|http?)://[\\w/%.-]+", "")
    .replaceAll("(?:https?|http?)://[\\w/%.-]+\\s+", "")
    .replaceAll("(?:https?|http?)//[\\w/%.-]+\\s+", "")
    .replaceAll("(?:https?|http?)//[\\w/%.-]+", "")
    .trim()
)).toDF()
After loading, we need to convert tweets into feature vectors.
val tokenizer = new
Tokenizer().setInputCol("tweet").setOutputCol("words")
val wordsData = tokenizer.transform(data)
```

```
val hashingTF = new HashingTF()
```

.setInputCol("words").setOutputCol("rawFeatures").setNumFeatures(1000)

val featurizedData = hashingTF.transform(wordsData)

val idf = new
IDF().setInputCol("rawFeatures").setOutputCol("features")
val idfModel = idf.fit(featurizedData)

val rescaledData = idfModel.transform(featurizedData)

Then we split the transformed data into two subsets i.e. training and test(ratio 0.7:0.3)

val Array(training, test) = rescaledData.randomSplit(Array(0.7,0.3), seed=1234L)

Model Selection and Model Tuning

We tried Naïve Bays and Gradient Boosted Trees for classification.

val nb = new NaiveBayes()

val paramGrid = new ParamGridBuilder()

.addGrid(nb.modelType,
Array("multinomial","complement","gaussian"))

.build()

```
val cv = new CrossValidator()
        .setEstimator(nb)
        .setEvaluator(new BinaryClassificationEvaluator())
        .setEstimatorParamMaps(paramGrid)
        .setNumFolds(3)
        .setParallelism(2)
val cvModel = cv.fit(training)
val nb_predictions = cvModel.transform(test)
val gbt = new GBTClassifier()
            .setLabelCol("label")
            .setFeaturesCol("features")
           .setMaxIter(15)
           .setFeatureSubsetStrategy("all")
           .setMaxDepth(10)
val paramGrid2 = new ParamGridBuilder()
       .addGrid(gbt.featureSubsetStrategy, Array("auto", "all"))
             .addGrid(gbt.maxDepth, Array(5,10))
             .addGrid(gbt.maxIter, Array(10,15))
             .build()
```

```
val model2 = cv2.fit(training)
val gbt_predictions = model.transform(test)
```

Conclusion

We evaluated accuracy for both Naïve Bays and Gradient boosted trees using MultiClassClassification Evaluator and got 68 % accuracy for Naïve Bays and 71 %

val evaluator = new MulticlassClassificationEvaluator()

```
.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("accuracy")

val nb_accuracy = evaluator.evaluate(nb_predictions)

val gbt_accuracy = evaluator.evaluate(gbt_predictions)
```