[First GOP Debate Twitter Sentiment](http://122.165.140.148/terasql/deep-insights-dsci/-/tree/patch-1/Twitter%20Sentiment%20Analysis)

Data Preparation

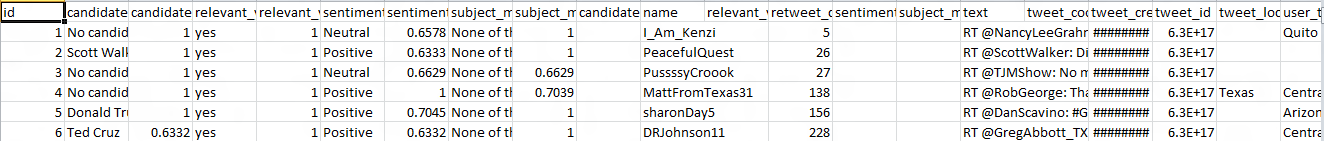
Data is downloaded from

https://www.kaggle.com/crowdflower/first-gop-debate-twitter-sentiment

It contains 14k tweets.

It contains the following fields:

1. id
2. candidate
3. candidate\_confidence
4. relevant\_yn
5. relevant\_yn\_confidence
6. sentiment
7. sentiment\_confidence
8. subject\_matter
9. subject\_matter\_confidence
10. candidate\_gold
11. name
12. relevant\_yn\_gold
13. retweet\_count
14. sentiment\_gold
15. subject\_matter\_gold
16. text
17. tweet\_coord
18. tweet\_created
19. tweet\_id
20. tweet\_location
21. user\_timezone



So firstly we have create schema:

val schema = StructType(Array(StructField("id", IntegerType, true),

StructField("candidate", StringType, true),

StructField("candidate\_confidence", DoubleType, true),

StructField("relevant\_yn", StringType, true),

StructField("relevant\_yn\_confidence", DoubleType, true),

StructField("sentiment", StringType, true),

StructField("sentiment\_confidence", DoubleType, true),

StructField("subject\_matter", StringType, true),

StructField("subject\_matter\_confidence", DoubleType, true),

StructField("candidate\_gold", StringType, true),

StructField("name", StringType, true),

StructField("relevant\_yn\_gold", StringType, true),

StructField("retweet\_count", DoubleType, true),

StructField("sentiment\_gold", StringType, true),

StructField("subject\_matter\_gold", StringType, true),

StructField("text", StringType, true),

StructField("tweet\_coord", StringType, true),

StructField("tweet\_created", StringType, true),

StructField("tweet\_id", StringType, true),

StructField("tweet\_location", StringType, true),

StructField("user\_timezone", StringType, true)

))

Now, we load the data in a dataframe using above schema:

val df = spark.read.

format("csv")

.schema(schema)

.option("header","true")

.load("/home/harsh/Desktop/gop debate twitter/Sentiment.csv")

Selecting only those columns which are useful:

val df1 = df.select("candidate","candidate\_confidence", "relevant\_yn", "relevant\_yn\_confidence", "sentiment", "sentiment\_confidence", "subject\_matter", "subject\_matter\_confidence", "retweet\_count", "text", "tweet\_coord")

Combining text (String ) columns into a single column:

val o = df1.select( concat($"candidate", lit(" "), $"subject\_matter", lit(" "), $"text" ).alias("text"), $"candidate\_confidence", $"relevant\_yn", $"relevant\_yn\_confidence", $"sentiment", $"sentiment\_confidence", $"subject\_matter\_confidence", $"retweet\_count" )

Removing records having null values:

val p = o.na.drop

Column “relevant\_yn” consist of two string values either yes or no. To convert its values in numerical values, we create a udf:

def lo(i:String) : Double = { if(i.equals("no")){0} else{1} }

val rel = udf(lo \_)

Column “sentiment” consist of three string values Negative, Nuetral, Positive. To convert its values in numerical values, we create a udf:

def pr(i:String) : Double = { if(i.equals("Negative")){0}

else if(i.equals("Neutral")){1}

else{2}

}

val sent = udf(pr \_)

For pre-processing (removing website links, emojis, special characters, unnecessary spaces) we create a udf :

def prep(d:String) :String = { d.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/%.-]+", "")

.replaceAll("[^\u0000-\uFFFF]","")

.replaceAll("(\u00a9|\u00ae|[\u2000-\u3300]|\ud83c[\ud000-\udfff]|\ud83d[\ud000-\udfff]|\ud83e[\ud000-\udfff])","")

.trim()

}

val preProcess = udf(prep \_)

Applying all udfs on columns:

val data = p.select(preProcess($"text" ).alias("text"), $"candidate\_confidence", rel($"relevant\_yn").alias("relevant\_yn"), $"relevant\_yn\_confidence", sent($"sentiment").alias("label"), $"sentiment\_confidence", $"subject\_matter\_confidence", $"retweet\_count")

Then to use nlp stemmer, we need to text have to go through document, then through token, then through normalizer, then through stemmer, then finally through finisher:

val document = new DocumentAssembler()

.setInputCol("text")

.setOutputCol("document")

val d1 = document.transform(data)

val token = new com.johnsnowlabs.nlp.annotator.Tokenizer()

.setInputCols("document")

.setOutputCol("token")

val t1 = token.fit(d1).transform(d1)

val normalizer = new Normalizer()

.setInputCols("token")

.setOutputCol("normal")

val n1 = normalizer.fit(t1).transform(t1)

val stemmer = new Stemmer()

.setInputCols("normal")

.setOutputCol("stem")

val s1 = stemmer.transform(n1)

val finisher = new Finisher()

.setInputCols("stem")

.setOutputCols("final")

val f1 = finisher.transform(s1)

After loading, we need to convert tweets into feature vectors.

val hashingTF = new HashingTF()

.setInputCol("final").setOutputCol("rawFeatures").setNumFeatures(10000)

val featurizedData = hashingTF.transform(f1)

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

val idfModel = idf.fit(featurizedData)

val rescaledData = idfModel.transform(featurizedData)

Then we need to assemble all numeric columns and categorical feature vectors to form final Features vector:

val assembler = new VectorAssembler()

.setInputCols(Array("candidate\_confidence", "relevant\_yn", "sentiment\_confidence","subject\_matter\_confidence","retweet\_count", "features"))

.setOutputCol("finalFeatures")

val output = assembler.transform(rescaledData)

Extracting label and Final Features (only they are needed to proceed further)

val limited = output.select($"label",$"finalFeatures".alias("features"))

Then we split the transformed data into two subsets i.e. training and test(ratio 0.8:0.2)

val Array(training, test) = limited.randomSplit(Array[Double](0.8,0.2))

Model Selection and Model Tuning

We tried Logistic Regression for classification.

val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.01).setLabelCol("label").setElasticNetParam(0.5)

val model = lr.fit(training)

val preTr = model.transform(training)

val preTs = model.transform(test)

Conclusion

We evaluated accuracy for model using MultiClassClassification Evaluator and got 75 % accuracy for training and 68 % for testing.

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("accuracy")

val train\_accuracy = evaluator.evaluate(preTr)

val test\_accuracy = evaluator.evaluate(preTs)