[Amazon Fine Food Reviews](http://122.165.140.148/terasql/deep-insights-dsci/-/tree/patch-1/Twitter%20Sentiment%20Analysis)

Data Preparation

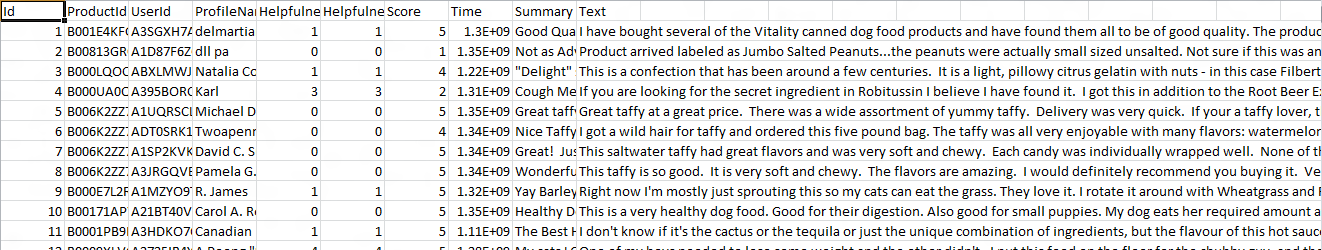
Data is downloaded from

https://www.kaggle.com/snap/amazon-fine-food-reviews

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012.

It contains the following fields:

1. Id
2. ProductId
3. UserId
4. ProfileName
5. HelpfulnessNumerator
6. HelpfulnessDenominator
7. Score : Ranging 1 to 5
8. Time
9. Summary
10. Text



In this, we label those reviews as positive whose score is either 4 or 5 and as negative whose score is either 1 or 2. Records with score 3 are not considered for classification.

Loading data into a dataframe:

val df = spark.read.

format("csv")

.option("header","true")

.option("inferSchema","true")

.load("/home/harsh/Desktop/amazon food/Reviews.csv")

Removing records with score 3:

val n = df.filter("Score !=3")

Making a udf for labelling based on score:

def lo(i:Int) :Int = { if(i>3){1} else{0} }

val labelling = udf(lo \_)

Extracting the required columns along with labelling:

val fine = n.select(labelling($"Score").alias("label"), $"HelpfulnessNumerator", $"HelpfulnessDenominator", $"Summary", $"Text" )

Taking only consistent data:

val consistent = fine.filter($"HelpfulnessNumerator" <= $"HelpfulnessDenominator")

Making an udf for pre-processing (removing emojis, website links, special characters, unnecessary spaces)

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 \_)

val data = consistent.select($"label", $"HelpfulnessNumerator", $"HelpfulnessDenominator", concat(preProcess($"Summary"), lit(" "), preProcess($"Text")).alias("text"))

Then, we want to use nlp stemming, so we convert string into document, then token, then normalizer, then stemmer, then 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 text into feature vectors.

val hashingTF = new HashingTF()

.setInputCol("filtered").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)

Combining HelpfulnessNumerator, HelpfulnessDenominator and features to a vector column:

val assembler = new VectorAssembler() .setInputCols(Array("HelpfulnessNumerator", "HelpfulnessDenominator", "features"))

.setOutputCol("finalFeatures")

val output = assembler.transform(rescaledData)

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 90 % accuracy for both training and testing.

val evaluator = new MulticlassClassificationEvaluator()

.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("accuracy")

val train\_accuracy = evaluator.evaluate(preTr)

val test\_accuracy = evaluator.evaluate(preTs)