Big Data Project

Kaushik Indukuri, Naveen Reddy, B V Murari

May 19, 2021

1 Objective

This report is based on the yelp data set given to us where based on users written reviews there was a rating given to a place, restaurant, etc. Our aim was to build a Machine learning based model which can be used to predict these ratings for further new data inputs given. So we tried out varies Natural language processing techniques to take text as input and predict the ratings as output.

In the end we built a model using the data given to us and saved it, which we used for batch processing given any new input data file, or also for real-time computation using Kafka to stream data. We also gave f1-score and accuracy for the purpose of this demonstration as we had output for test data too.

2 Pre-processing

Preprocessing steps employed by us :

- String Tokeniser : To get each word separately as a vector form of the sentence.
- N grams : Here we have used n=2, so it pairs words continuously occurring into one and forms another vector.
- Count Vectoriser : Create a dictionary out of the words, and store the counts.
- Tf-idf : Another preprocessing which takes in the input of 1-gram and 2-gram and weighs based on occurences the importance of each word.
- Vector assembler : Final step to combine the previous columns into one for input to model.

```
In [19]: n=2
         ngrams = [

                 NGram(n=i, inputCol="words", outputCol="{0}_grams".format(i))
                 for i in range(1, n + 1)
             ]
         Ngrams = Pipeline(stages = ngrams)
         NgramsFit = Ngrams.fit(tokenizerOutputs)
         NgramsOutputs = NgramsFit.transform(tokenizerOutputs)

In [20]: NgramsOutputs.show(5)

+--------------------+-----+--------------------+--------------------+--------------------+
|                text|stars|               words|             1_grams|             2_grams|
+--------------------+-----+--------------------+--------------------+--------------------+
|Great food ok pri...|  2.0|[great, food, ok,...|[great, food, ok,...|[great food, food...|
|Local hole in the...|  3.0|[local, hole, in,...|[local, hole, in,...|[local hole, hole...|
|I'm giving this p...|  4.0|[i'm, giving, thi...|[i'm, giving, thi...|[i'm giving, givi...|
|Really good pizza...|  5.0|[really, good, pi...|[really, good, pi...|[really good, goo...|
|One of the worst ...|  1.0|[one, of, the, wo...|[one, of, the, wo...|[one of, of the, ...|
+--------------------+-----+--------------------+--------------------+--------------------+
only showing top 5 rows
```

Figure 1: input text : Individual words : ngrams

```
In [22]: cv = [
             CountVectorizer(vocabSize=1024,inputCol="{0}_grams".format(i),
                 outputCol="{0}_tf".format(i))
             for i in range(1, n + 1)
         ]
cv = Pipeline(stages = cv)
cvFit = cv.fit(NgramsOutputs)
cvOutputs = cvFit.transform(NgramsOutputs)

In [24]: cvOutputs[['1_grams', '2_grams', '1_tf', '2_tf']].show(5)

+--------------------+--------------------+--------------------+--------------------+
|             1_grams|             2_grams|               1_tf|               2_tf|
+--------------------+--------------------+--------------------+--------------------+
|[great, food, ok,...|[great food, food...|(1024,[0,1,5,8,11...|(1024,[5,11,46,60...|
|[local, hole, in,...|[local hole, hole...|(1024,[0,1,3,7,8,...|(1024,[1,6,11,35,...|
|[i'm, giving, thi...|[i'm giving, givi...|(1024,[0,1,2,3,4,...|(1024,[10,11,13,2...|
|[really, good, pi...|[really good, goo...|(1024,[1,3,8,15,2...|(1024,[33,64,91,1...|
|[one, of, the, wo...|[one of, of the, ...|(1024,[0,3,5,7,8,...|(1024,[1,11,40,60...|
+--------------------+--------------------+--------------------+--------------------+
only showing top 5 rows
```

Figure 2: ngrams : countvectorizer on ngrams

```
In [27]: idf = [IDF(inputCol="{0}_tf".format(i), outputCol="{0}_tfidf".format(i), minDocFreq=5) for i in range(1, n + 1)]

idf = Pipeline(stages = idf)
idfFit = idf.fit(cvOutputs)
idfOutputs = idfFit.transform(cvOutputs)

In [29]: idfOutputs[['1_tf', '2_tf', '1_tfidf', '2_tfidf']].show(5)

+--------------------+--------------------+--------------------+--------------------+
|               1_tf|               2_tf|             1_tfidf|             2_tfidf|
+--------------------+--------------------+--------------------+--------------------+
|(1024,[0,1,5,8,11...|(1024,[5,11,46,60...|(1024,[0,1,5,8,11...|(1024,[5,11,46,60...|
|(1024,[0,1,3,7,8,...|(1024,[1,6,11,35,...|(1024,[0,1,3,7,8,...|(1024,[1,6,11,35,...|
|(1024,[0,1,2,3,4,...|(1024,[10,11,13,2...|(1024,[0,1,2,3,4,...|(1024,[10,11,13,2...|
|(1024,[1,3,8,15,2...|(1024,[33,64,91,1...|(1024,[1,3,8,15,2...|(1024,[33,64,91,1...|
|(1024,[0,3,5,7,8,...|(1024,[1,11,40,60...|(1024,[0,3,5,7,8,...|(1024,[1,11,40,60...|
+--------------------+--------------------+--------------------+--------------------+
only showing top 5 rows
```

Figure 3: ngrams count : tfidf on ngrams

```
In [30]: assembler = [VectorAssembler(
             inputCols=["{0}_tfidf".format(i) for i in range(1, n + 1)],
             outputCol="features"
             )]
assembler = Pipeline(stages = assembler)
assemblerFit = assembler.fit(idfOutputs)
assemblerOutputs = assemblerFit.transform(idfOutputs)

In [32]: assemblerOutputs[['text', 'features']].show(5)

+--------------------+--------------------+
|                text|            features|
+--------------------+--------------------+
|Great food ok pri...|(2048,[0,1,5,8,11...|
|Local hole in the...|(2048,[0,1,3,7,8,...|
|I'm giving this p...|(2048,[0,1,2,3,4,...|
|Really good pizza...|(2048,[1,3,8,15,2...|
|One of the worst ...|(2048,[0,3,5,7,8,...|
+--------------------+--------------------+
only showing top 5 rows
```

Figure 4: input data : Final output column

3 Best model and Inferences

Model	Accuracy	F1-score
Naive Bayes	.16	.11
Logistic	.64	.61
DecisionTree	.5	.3
OneVsAll	.45	.25

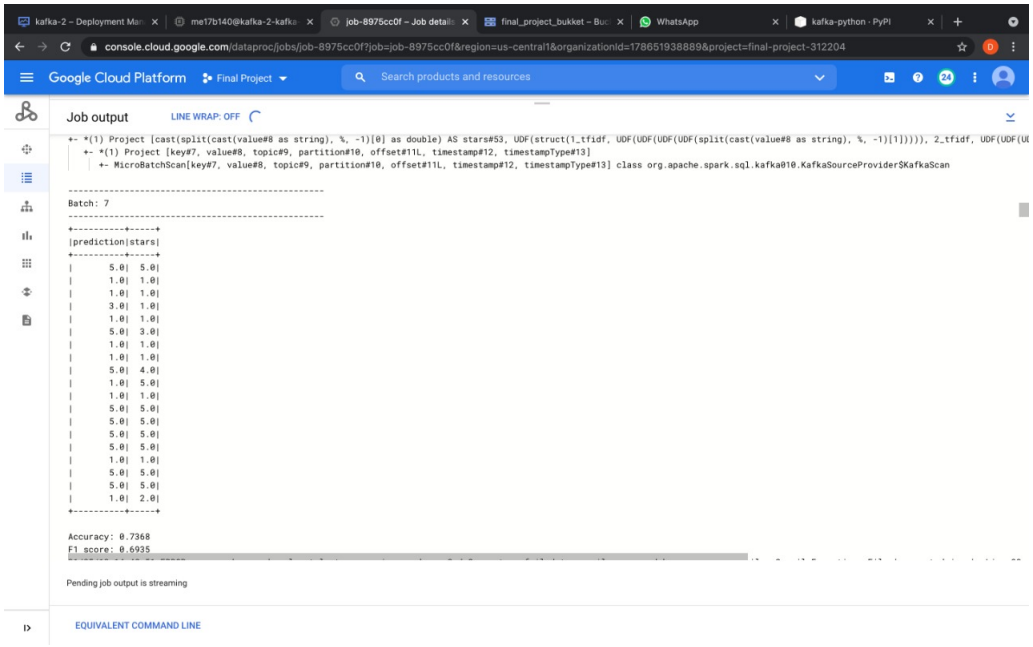
Each of the above model was tried after all the preprocessing steps we used on the data(using one of the files given in the bucket) and we conclude that logistic regression performed the best out of all models and attached the image of f1-score and accuracy of this model.

```
In [12]: print("doing...")
         acc = evaluatorMulti.evaluate(predictionAndTarget, {evaluatorMulti.metricName: "accuracy"})
         f1 = evaluatorMulti.evaluate(predictionAndTarget, {evaluatorMulti.metricName: "f1"})
         print("Accuracy of logistic is = %g"% (acc))
         print("F1 of logistic is = %g"% (f1))

         doing...
         Accuracy of logistic is = 0.640894
         F1 of logistic is = 0.608265
```

Figure 5: Best model(Logistic regression) F1 score and accuracy

4 Real Time computation



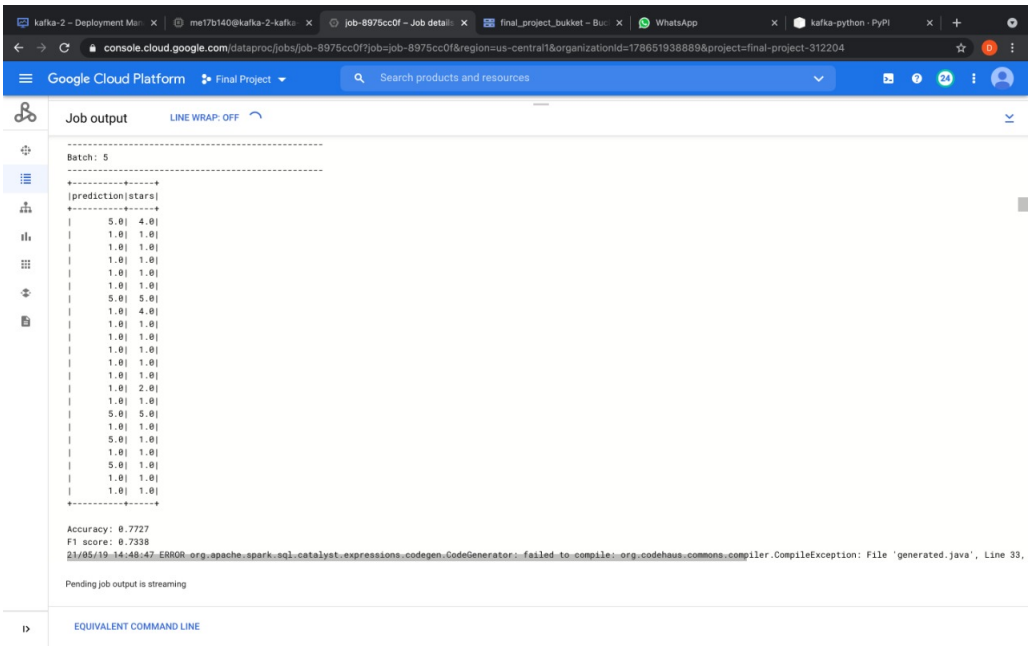


Figure 6: Real time streaming from kafka

Input bucket name and filename used to get the json data provided and the use full columns for prediction published to kakfa producer. This data read in real time by subscribing to the topic and printing accuracy and f1-score.

5 Conclusion

Given this is a natural language multiclass classification problem, getting high accuracy's with just Machine learning models(No deep learning) is a bit difficult. On testing with various methods we found that n-grams with tf-idf gave the best results and for the large data we had we used(n=2 of n-grams) to train the data and built a model giving these results.