CS4830 Big Data Laboratory

Predict star ratings - YELP dataset

ME17B144 | ME17B120 | ME17B112

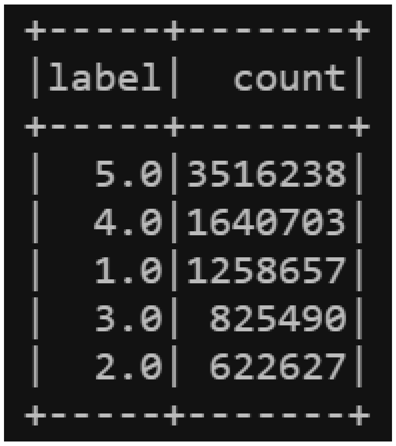
Introduction

The aim of the project was to predict the star ratings of yelp data using NLP. Preprocessing and other necessary tasks were to be performed in a DataProc cluster using Spark and access the trained model for real-time predictions

Data Preprocessing & Model Building

Data Description

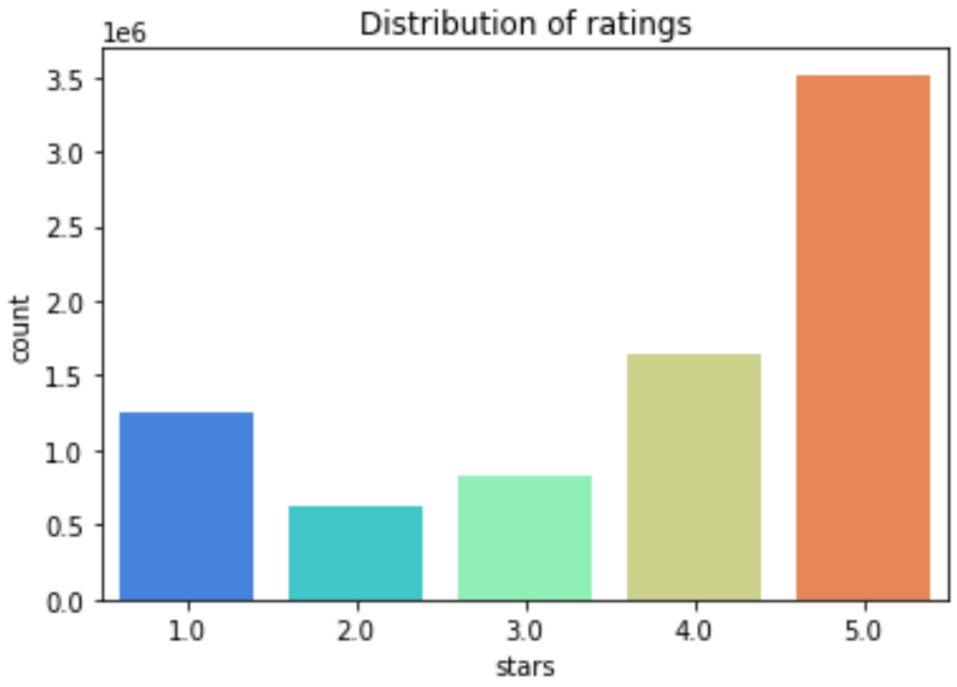
Data had a total of 4 features along with business_id, review_id, user_id, and the date of the review. The 4 features of the review described whether the review was 1) cool, 2) funny 3) useful, and finally the review itself as text. The label column was stars, which is a 5 class categorical column, describing the rating given by the user. The count of data from each class is as follows.

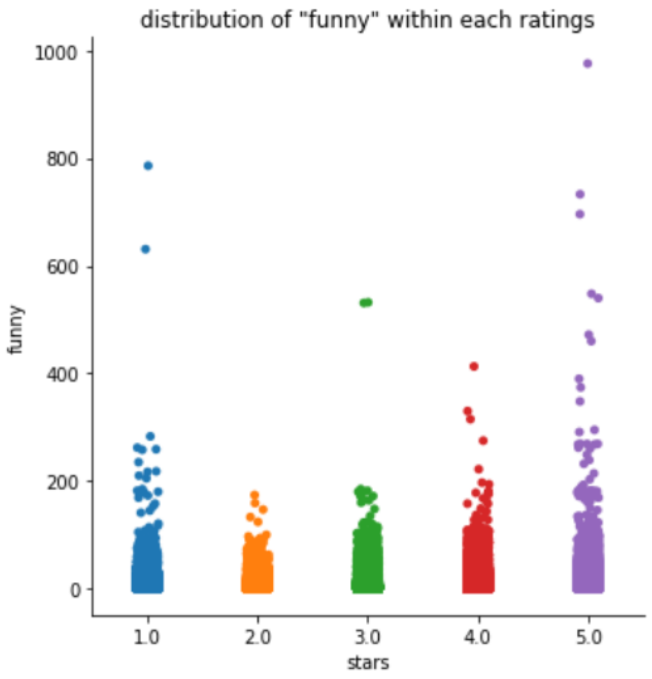


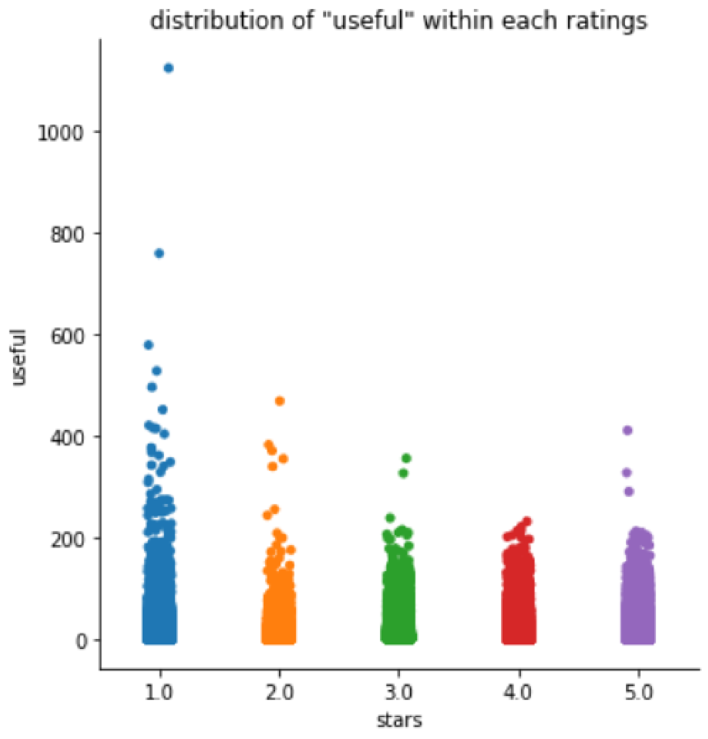
```
+-----+-------+
|label|  count|
+-----+-------+
|  5.0|3516238|
|  4.0|1640703|
|  1.0|1258657|
|  3.0| 825490|
|  2.0| 622627|
+-----+-------+
```

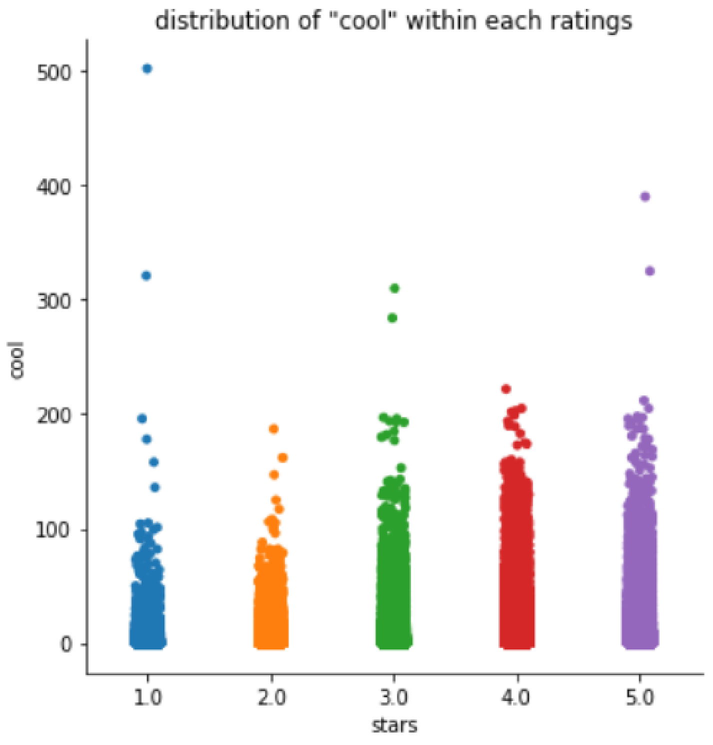
Fig 1: Data Count per label class

Exploratory Data Analysis






distribution of "useful" within each ratings
useful
0
200
400
600
800
1000
1.0
2.0
3.0
4.0
5.0
stars


distribution of "cool" within each ratings
cool
0
100
200
300
400
500
1.0
2.0
3.0
4.0
5.0
stars

Data Preprocessing

Regex Tokenizer: A RegexpTokenizer splits a string into substrings using a regular expression and creates tokens

Stopwords removal: Stopwords are English words that do not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. For example, the words like the, he, have, etc.

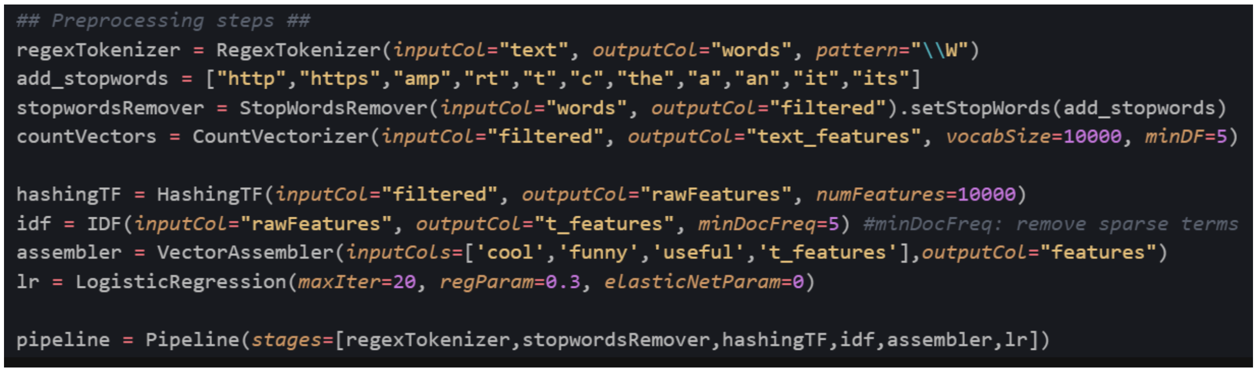
Count Vectorizer: The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

HashingTF: HashingTF is a transformation that takes in a set of terms and converts them into fixed-size vectors.

IDF: Inverse document frequency. A statistical weight used for measuring the importance of a term in a text document collection. The document frequency DF of a term is defined by the number of documents in which a term appears.

After these steps, VectorAssembler is used to combine all the features into a single feature vector.

The entire dataset was split into trainingData and validationData (0.7,0.3).



```
## Preprocessing steps ##
regexTokenizer = RegexTokenizer(inputCol="text", outputCol="words", pattern="\\W")
add_stopwords = ["http","https","amp","rt","t","c","the","a","an","it","its"]
stopwordsRemover = StopWordsRemover(inputCol="words", outputCol="filtered").setStopWords(add_stopwords)
countVectors = CountVectorizer(inputCol="filtered", outputCol="text_features", vocabSize=10000, minDF=5)

hashingTF = HashingTF(inputCol="filtered", outputCol="rawFeatures", numFeatures=10000)
idf = IDF(inputCol="rawFeatures", outputCol="t_features", minDocFreq=5) #minDocFreq: remove sparse terms
assembler = VectorAssembler(inputCols=['cool','funny','useful','t_features'],outputCol="features")
lr = LogisticRegression(maxIter=20, regParam=0.3, elasticNetParam=0)

pipeline = Pipeline(stages=[regexTokenizer,stopwordsRemover,hashingTF,idf,assembler,lr])
```

Model Building

A logistic regression model with params maxIter = 20, regParam = 0.3, and elasticNetParam = 0 was trained on the trainingData.

Results

Validation Accuracy = 0.626888717420035

Train Accuracy = 0.6280051979445153

Train F1- score = 0.5726721694739169

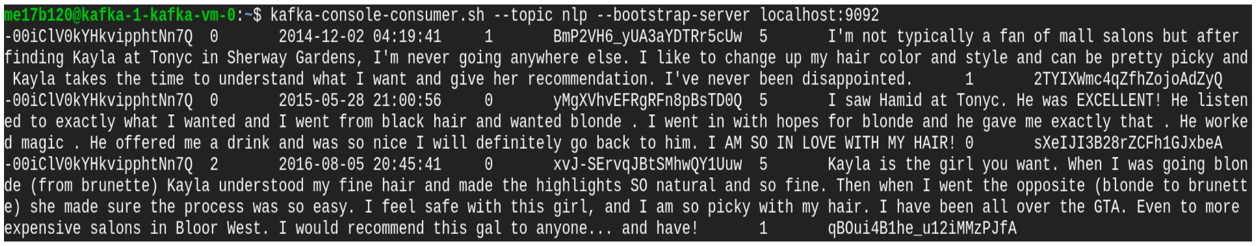
Kafka streaming

The model in our case contains the preprocessing pipeline along with the actual Logistic Regression model. Due to this, there is no need to manually preprocess the data that's being streamed, it's only required to pass the required rows to the model.

Producer

The dataset exists in the form of a directory filled with .json files which are essentially directories. We use cloud storage API to pass in the bucket and the path of the directory containing the data as arguments to the ***run_producer*** function. We use the pandas library which can read files directly from GCS to read all the files in the given directory and write each file line by line to a given topic in a Kafka cluster.

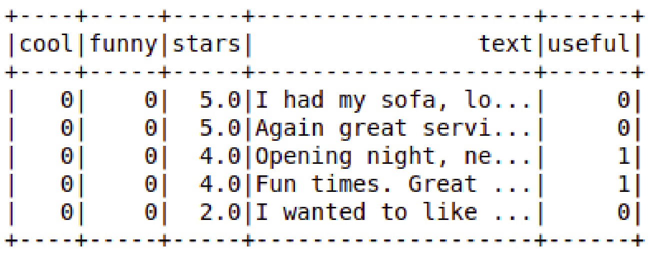
Since the data is written to the topic line by line as strings, we convert the data into tab-separated data so that it could be read and split into columns easily, since splitting normally wouldn't be possible due to one of the fields being text/review which contains a lot of spaces. Following is the sample output of the producer writing each line to a topic, the fields are separated by tabs that look like 4 spaces.



```
me17b120@kafka-1-kafka-vm-0:~$ kafka-console-consumer.sh --topic nlp --bootstrap-server localhost:9092
-00iClV0kYHkvipphtNn7Q  0       2014-12-02 04:19:41     1       BmP2VH6_yUA3aYDTRr5cUw  5       I'm not typically a fan of mall salons but after
finding Kayla at Tonyc in Sherway Gardens, I'm never going anywhere else. I like to change up my hair color and style and can be pretty picky and
 Kayla takes the time to understand what I want and give her recommendation. I've never been disappointed.      1       2TYIXWmc4qZfhZojoAdZyQ
-00iClV0kYHkvipphtNn7Q  0       2015-05-28 21:00:56     0       yMgXVhvEFRgRFn8pBsTD0Q  5       I saw Hamid at Tonyc. He was EXCELLENT! He listen
ed to exactly what I wanted and I went from black hair and wanted blonde . I went in with hopes for blonde and he gave me exactly that . He worke
d magic . He offered me a drink and was so nice I will definitely go back to him. I AM SO IN LOVE WITH MY HAIR! 0       sXeIJI3B28rZCFh1GJxbeA
-00iClV0kYHkvipphtNn7Q  2       2016-08-05 20:45:41     0       xvJ-SErvqJBtSMhwQY1Uuw  5       Kayla is the girl you want. When I was going blon
de (from brunette) Kayla understood my fine hair and made the highlights SO natural and so fine. Then when I went the opposite (blonde to brunett
e) she made sure the process was so easy. I feel safe with this girl, and I am so picky with my hair. I have been all over the GTA. Even to more
expensive salons in Bloor West. I would recommend this gal to anyone... and have!       1       qBOui4B1he_u12iMMzPJfA
```

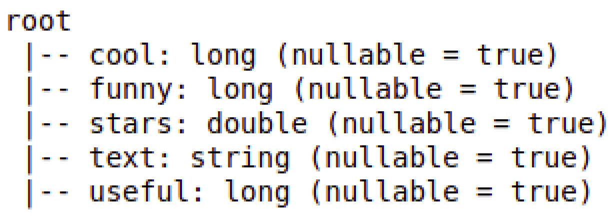
Consumer

The model only uses the following features from the data:



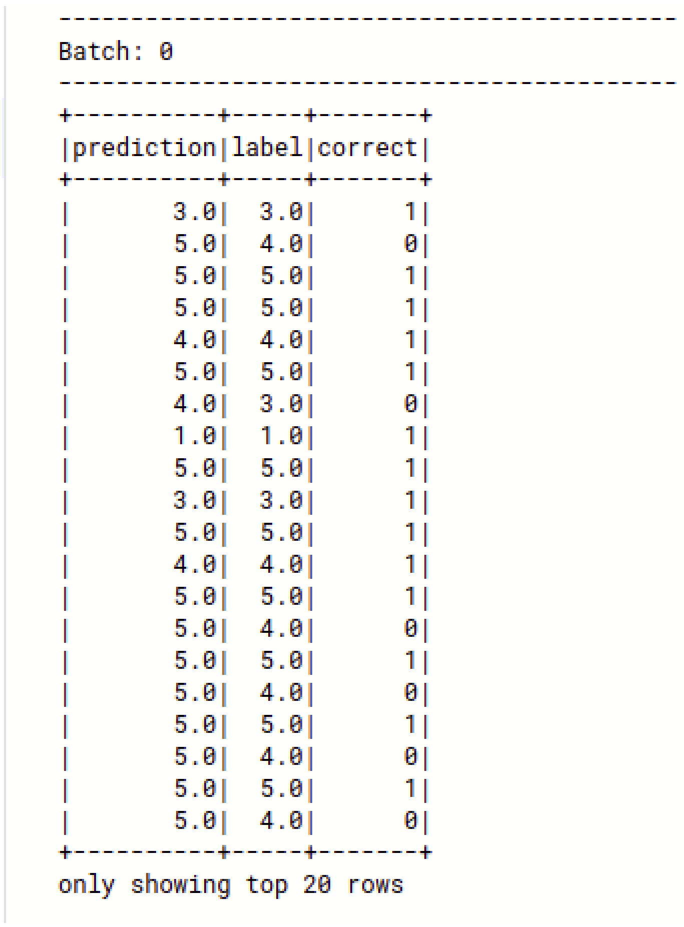
```
+----+-----+-----+--------------------+------+
|cool|funny|stars|                text|useful|
+----+-----+-----+--------------------+------+
|   0|    0|  5.0|I had my sofa, lo...|     0|
|   0|    0|  5.0|Again great servi...|     0|
|   0|    0|  4.0|Opening night, ne...|     1|
|   0|    0|  4.0|Fun times. Great ...|     1|
|   0|    0|  2.0|I wanted to like ...|     0|
+----+-----+-----+--------------------+------+
```

Thus, only the following columns should be passed to the model. The types of all the features are as follows.



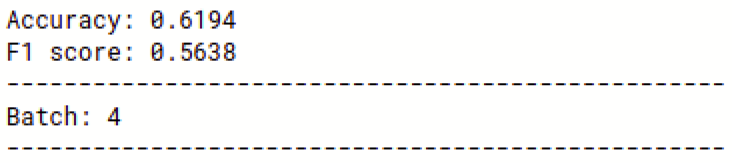
```
root
 |-- cool: long (nullable = true)
 |-- funny: long (nullable = true)
 |-- stars: double (nullable = true)
 |-- text: string (nullable = true)
 |-- useful: long (nullable = true)
```

Thus we 1st read each line splitting it by tabs and select only the required columns and cast them to the required object types. Now, this input is passed on to the model which stores the predictions in the ***"prediction"*** column. For each sample, we create a new column called ***"correct"*** which holds whether the prediction for the particular sample was correct or not.



```
-------------------------------------------
Batch: 0
-------------------------------------------
+----------+-----+-------+
|prediction|label|correct|
+----------+-----+-------+
|       3.0|  3.0|      1|
|       5.0|  4.0|      0|
|       5.0|  5.0|      1|
|       5.0|  5.0|      1|
|       4.0|  4.0|      1|
|       5.0|  5.0|      1|
|       4.0|  3.0|      0|
|       1.0|  1.0|      1|
|       5.0|  5.0|      1|
|       3.0|  3.0|      1|
|       5.0|  5.0|      1|
|       4.0|  4.0|      1|
|       5.0|  5.0|      1|
|       5.0|  4.0|      0|
|       5.0|  5.0|      1|
|       5.0|  4.0|      0|
|       5.0|  5.0|      1|
|       5.0|  4.0|      0|
|       5.0|  5.0|      1|
|       5.0|  4.0|      0|
+----------+-----+-------+
only showing top 20 rows
```

Since the evaluation functions in Spark like ***MulticlassClassificationEvaluator*** work only for static dataframes and not streaming dataframes, we use the ***foreachBatch*** function while writing to the stream to be able to use it. The accuracy and F1 score for each batch are printed on the console before the next batch as shown below.



```
Accuracy: 0.6194
F1 score: 0.5638
-------------------------------------------
Batch: 4
-------------------------------------------
```

The observed latency was nonsignificant and just in order of a few seconds, based on our observations.