Yelp Dataset Project Report

Class: EECS 4412

Name: Jaffarjeet Singh Brar
ID: 215939614
Instructor: Aijun An
Date: Dec 07, 2021

1. Objective

The yelp dataset comprises of online reviews of businesses such as restaurants, home services and auto parts etc, from customers. Many potentially new customers read online reviews before using a service. Therefore, it is necessary for business to distinguish between thousands of customer reviews and analyze those reviews to improve their services. Thus, the objective of this project is to build a classifier that labels these customer reviews in yelp dataset as positive, negative and neutral while achieving the maximum accuracy.

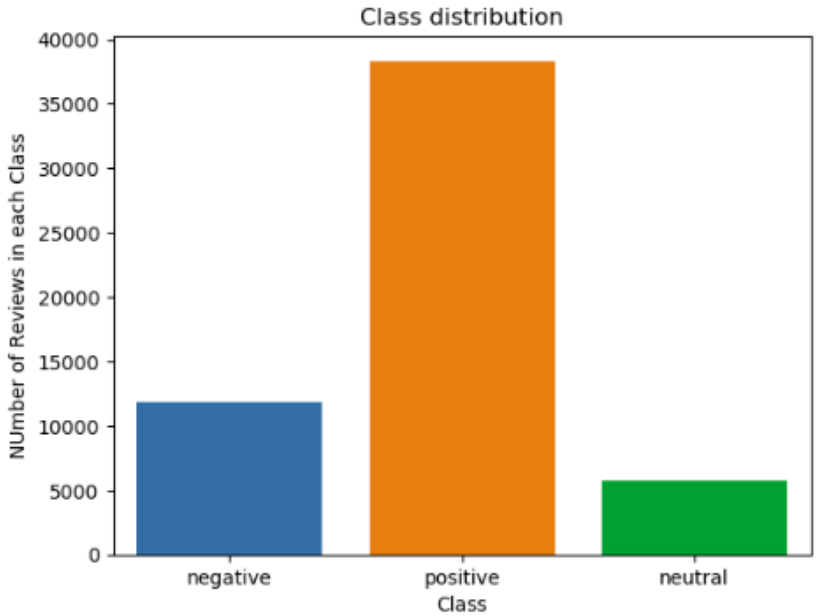
2. Data Analysis

Class Imbalance: If data suffers from class imbalance problem, the classifier has low predictive accuracy towards less frequent class. By looking at the data I observed that, The dataset was very imbalanced towards positive instances. The count for neutral class was the lowest: 5755 instances.
There we need to **oversample** the minority classes by increasing their instances

Ratio = *Positive: Negative: Neutral* = 38348:11897:5755 = *6.66: 2.06: 1*
For every 1 neutral instance, there are about 2 negative instances and 7 positive instances.

```
[7]: df = pd.read_csv("train3.csv")
     graph=sb.countplot(x="Class", data=df)
     graph.set(ylabel="NUmber of Reviews in each Class")
     plt.title('Class distribution')
     print(df.groupby('Class').count())
     plt.show()
```

```
           Text     ID
Class
negative  11897  11897
neutral    5755   5755
positive  38348  38348
```



Stopwords: Next, I looked at number of stop words and punctuation in dataset. About 0.58% of the text contains special characters like !!!, ? etc, because some customers write angry reviews using exclamation marks etc. These characters wouldn't help us much in classifying reviews.
Further, I compared the text file with *stopwords.txt* provided in the project and found that 8.6 % of the text comprised of commonly occuring words like a, an, the etc. These words don't contribute towards classification and are common among all reviews.
Therefore, we observe that about **9% of data is useless** and we need to get rid of this data before we train a classifier

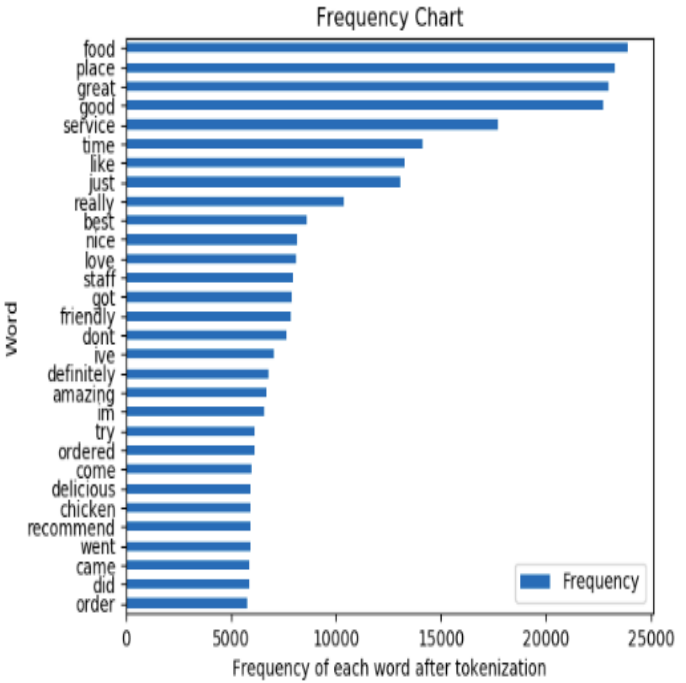
```
[8]: punct = lambda l1,l2: sum([1 for x in l1 if x in l2])
     p= punct(entire_text,set(string.punctuation))
     print("The total number of words in text are %d" % length)
     print("The total number of punctuation characters are %d (%3f%%) " %(p, (p/length)*100))
     for word in entire_text.split():
         if word in stop_words:
             count = count + 1
     print ("The total number of stop words in text are %d (%3f%%)" % (count, (count/length)*100))

     The total number of words in text are 22723446
     The total number of punctuation characters are 132638 (0.583705%)
     The total number of stop words in text are 1953267 (8.595822%)
```

Most Frequent Words: Next I looked at the words which occur the most to get sense of what most reviews are about. Words like food, place and great are frequent due to large set of positive reviews. I also observed that words like *did* and *got* appear as most frequent even after removing stop words. This means we have to convert them to their base form like *do and get*.
Therefore, to solve this problem we need to **undersample** positive reviews, perform **stemming and lemmatization**

```
# WordCloud generation
word_cloud= WordCloud(background_color='white', stopwords=stop_words).generate(entire_text)
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```


```
d = pd.DataFrame(tokens_distribution.most_common(30), columns=['Word', 'Frequency'])
g=d.plot(x='Word', y='Frequency', kind='barh')
g.set(xlabel="Frequency of each word after tokenization", ylabel="Word")
plt.gca().invert_yaxis()
plt.title('Frequency Chart')
plt.show()
```



3. Data Preprocessing

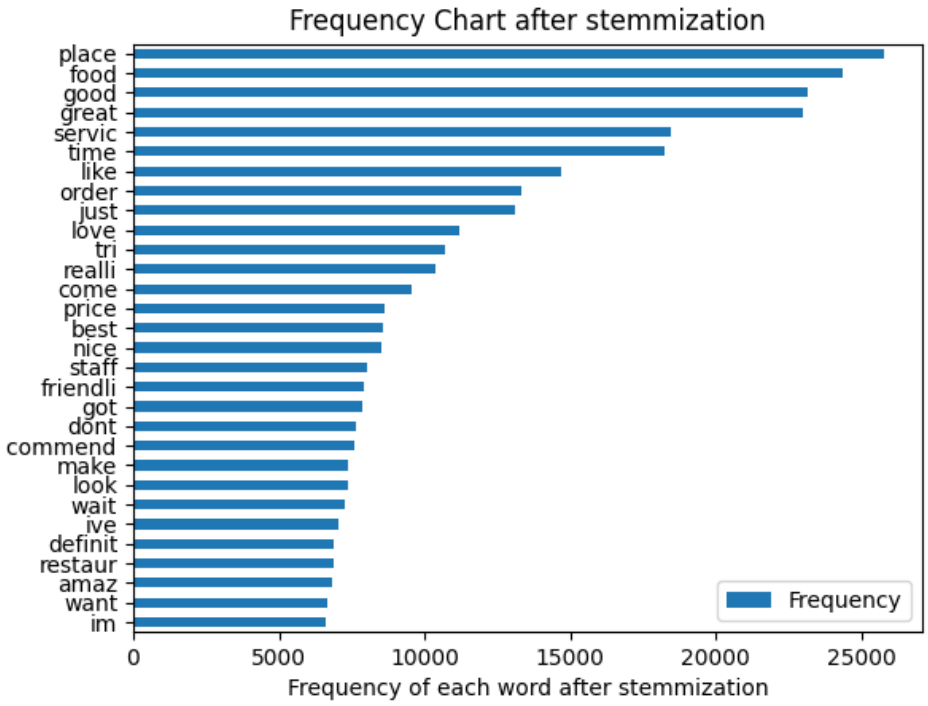
1. Removing stop words: First step is to remove stop words. The dataset was compared with stop words and all the stop words and unnecessary special characters were removed. In this way we focus on the 91% of data which is actually the useful data for our classification

2. Class balancing: The no. Of instances in positive class was reduced from 38348 to 25000 using under-sampling technique without replacement. The minority class neutral was up-sampled to match the number of negative instances, about 12000. The negative class was also up-sampled to 14000 instances in order to follow original ratio

Old ratio = *Positive: Negative: Neutral* = 38348:11897:5755 = *6.66: 2.06: 1*
New Ratio = *Positive:Negative:Neutral* = 25000:16500:12000 = *2.08: 1.38: 1*

The newly sample dataset is more balanced and isn't skewed towards the positive class

3. Stemming and Lemmatization: In order to remove the words that contain the same meaning and context, stemming and lemmatization was performed.
Stemming resulted in significant increase in performance (~5.22%) during the final classifying stage while lemmatization resulted in slight increase in performance (~0.68%) with running time as overhead cost. Although lemmatization wasn't favourable, it was included in the final classification in order to achieve highest accuracy.
Porter's stemming (Snowball stemmer, updated version of Porter's stemmer) was used to perform stemming.



4. Feature Extraction

TF-IDF Vectorizer was used to perform feature selection. The term frequency TF- counts the number of unique words in a document while inverse document frequency IDF- counts the frequency of rare words in all documents. Here document refers to different reviews sorted by customer ID. Using TF-IDF vectorizer, I was able to convert text into features. After TF-IDF vectorization, there were more than 90000 features in this document. In order to make the process the faster- the number of max features=[100,300,1000,10000] were studied using the Grid Search. However, reducing the max features shot down the accuracy from 91% to 84%. Therefore, all the features were kept.

Although TF-IDF vectorizer performs better than Count Vectorizer in general, because it normalizes the frequency count, it was observed that Count Vectorizer performed better with logistic regression and increased accuracy by 2% . It is due to the fact the performance of logistic regression doesn't improve with scaling and it is able to calculate the loss function more concretely without normalization.

Therefore, I decided to use 2 vectorizers. TF-IDF with multinomial Bayes and Count Vectorizer with logistic regression.

Parameter Selection:

1. N-Grams: Since the tradition Bag of Words (BoW) approach used by TF-IDF Vectorizer doesn't take int account the order of words in which they appear in the document and only counts individual words, it becomes more prone to false classification especially in reviews in which the order of words is important. For example if "good" is associated with positive review, then "not good" would also classify as positive review since it BoW only looks at individual words. Using N-Grams, increased the accuracy in both classifiers. n_grams=[2-5] was tested on validation set and accuracy increased by 0.01% after tri-gram. There n_grams=(1,3) was chosen for both vectorizers .

2. ***min_df***: This features acts as a minimum threshold, allowing the vectorizer to ignore words whose df values fall below the minimum threshold.

min_df values tested for Count Vectorizer with Logistic Regression						
min_df =	1	**2**	3	4	5	6
Accuracy	83.55	**85.22**	84.68	84.29	83.92	81.89

min_df=2 was chosen for Count Vecotrizer.

In Case of TF-IDF vectorizer with multinomial bayes, it results in gradual increase in accuracy with increase in min_df values. min_df=[1,10] was tested for TF-IDF vectorizer in combination with multinomial bayes using the Grid Search and min_df=10 was chosen.

[6]:
```
steps = [('vec',
          TfidfVectorizer(strip_accents='unicode',  min_df=10, stop_words=stop_words,
                          ngram_range=(1, 3))), ('mnb', MultinomialNB(alpha=0.1))]
nb = Pipeline(steps)

steps = [('tfidf',
          CountVectorizer(strip_accents='unicode',  stop_words=stop_words, min_df=2,
                          ngram_range=(1, 3))),
         ('lr', LogisticRegression(n_jobs=-1))]
LR = Pipeline(steps)
models = [('lr', LR), ('nb', nb)]
```

5. Classification

Data was split into training set and holdout validation set for testing purposes with 80-20 ratio.

Classifier 1: ***Multinomial Bayes:*** Various Algorithms like multi layer perceptron, SVM, random forest were initially tested on training data. However, their running time was very large and not feasible to perform data insights. Therefore, multinomial Bayes was chosen due to its easy implementation and fast learning. Since its feature counts usually require integer values, the TF-IDF values acts as integer features which can then be used to calculate probabilities using the Bayes' Theorem.
A smoothing hyper parameter alpha =0.1 was chosen after performing Grid Search On [0.1, 0.01, 0.001]. Its individual accuracy is 84.8±0.6 % (tested on different validation sets)

```
param = {'vec__min_df':[1, 10], 'mnb__alpha':[0.1, 0.01, 0.001]}
nb = GridSearchCV(pipeline, param, cv = 10, scoring="accuracy", verbose=1)
```

Classifier 2: Logistic Regression: Logistic Regression is especially useful in sentiment analysis since it uses sigmoid function to give a probability between 0 and 1. The default parameters were used. L1 and L2 regularization was considered but it performed similarly to default settings.
Its individual accuracy is 93.8±0.7 %

Final Classifier: **Weighted Voting Ensemble:** I used voting ensemble because logistic regression well with binary classes but has tendency to under perform with multi-label classes as in yelp dataset. Therefore, a voting ensemble was used which relies on accuracy of both classifiers .

```
nb = Pipeline(steps)
LR = Pipeline(steps)
models = [('lr', LR), ('nb', nb)]
scores = evaluate_models(models, x_train, x_test, y_train, y_test)
ensemble = VotingClassifier(estimators=models, voting='soft', weights=scores)
```

Hyper-parameters:

1. weights: The weights are equal to the individual accuracy scores of each classifier. In most cases, the overall accuracy decreases slightly but it doesn't vary much as in case of logistic regression. The most common weight age is [0.93, 0.85] resulting in accuracy of 93.8%.

2. *voting*: Soft voting is more than hard voting because hard voting works with mode of two outputs and decreases the overall accuracy to 91.4% whereas soft voting classifies the input data based on probabilities of predictions made by different classifiers. The soft voting lead to accuracy of 94%.

The overall accuracy of ensemble classifier is ~ **94%**

```
The weighted scores are [0.94, 0.8415094339622642]
Weighted Average accuracy is:  94.094
              precision    recall  f1-score   support

    positive       0.94      0.95      0.95      3000
    negative       0.92      0.89      0.90      2400
     neutral       0.95      0.96      0.95      5200

    accuracy                           0.94     10600
   macro avg       0.94      0.93      0.94     10600
weighted avg       0.94      0.94      0.94     10600

[[2858   63   79]
 [  79 2131  190]
 [  92  123 4985]]
```

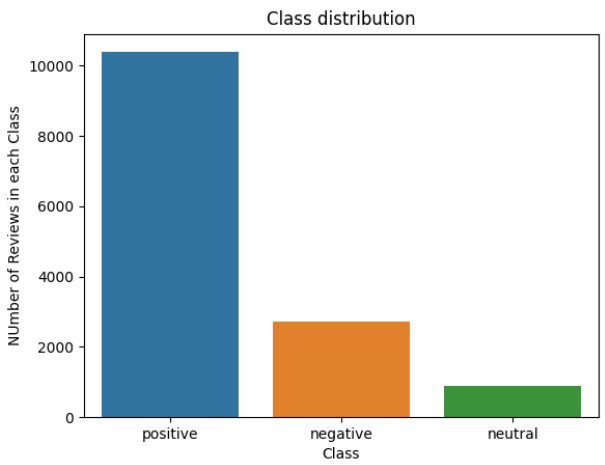
6. Discussion

I was interested in finding the effect of unigram, bi-gram and so on on the classifier. However, it seems that N-grams doesn't help much in improving the accuracy significantly. It might be due to the fact that since TF-IDF produces very large number of features, the bi-gram and tri-gram features, doesn't effect the classifier accuracy. If the number of features were reduced to 300, bi grams resulted in significant increase in accuracy (3.1%) than the original accuracy with 300 features (which was reduced to 84.28% at the expense of regularization).

7. Conclusion

We see that reviews assigned by the classifier closely follows the distribution of original dataset. The final results gets automatically saved in the *prediction.csv* file

```
The number of reviews in the dataset:  14000
Grouping Reviews by class, we get :
 positive    10385
negative     2734
neutral       881
```



8. Appendix

1. Save train3.csv, test.csv, yelp_Jaffar.py, stopwords.txt in the same folder or location.
2. In the linux terminal, type *python 3 yelp_jaffar.py*
3. **Ignore** the convergence ***warning*** by the program. It is due to logistic regression and doesn't stop the program from running
4. The running time of the algorithm is 6 minutes 50 seconds.
5. **prediction.csv** is automatically saved in the same folder along with data analysis file that contains tets_text with reviews .

```
user@user-HP-NOTBOOK-15-P202TU-PC:~/4412_project$ python3 yelp_jaffar.py
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
The weighted scores are [0.94, 0.8415094339622642]
Weighted Average accuracy is:  94.094
              precision    recall  f1-score   support

    positive       0.94      0.95      0.95      3000
    negative       0.92      0.89      0.90      2400
     neutral       0.95      0.96      0.95      5200

    accuracy                           0.94     10600
   macro avg       0.94      0.93      0.94     10600
weighted avg       0.94      0.94      0.94     10600

[[2858   63   79]
 [  79 2131  190]
 [  92  123 4985]]
```