CP322 Project Report

March 30th, 2022

Group 9:

Phoebe Schulman, Anojan Balendra, Harchit Bhatoia,

Shing Hei Mau, Alvin Onabolu

Introduction:

Context and Motivations

With online shopping trending more than ever, many customers look to reviews to decide whether an item is a good purchase. It is up to the business to ensure that customers are leaving good reviews so that new customers can continue buying their products (i.e., women's clothing). However, scanning through thousands of reviews and making judgment on a customer's feelings towards a product can be repetitive and time consuming. Thus, it would be much more cost-effective for the company if a machine was able to determine the sentiment of customer reviews. By doing this, the business can learn what makes customers happy or disappointed, so that they can tailor products and services to meet their customers' needs. The automatic sentiment detection of customer reviews on women's clothing is the purpose of the model we have built. The motivation behind the building of this project is delivering a machine that can save businesses money while also helping consumers be heard in the long run.
Sentiment analysis of a customer's review on a product and other factors such as positive feedback count, were key in determining the sentiment of reviews.

The question our classifier aims to solve is how happy the customer is with their product on a scale from is 1 to 5. 5 being very happy, 4 being happy, 3 average, 2 unhappy, 1 very unhappy. One of the techniques we used to analyze the sentiment of reviews was tokenization. We intuitively decided that a sentence is good if it contains more positive words. It is bad if it contains more negative words. Thus, breaking up each review into words and seeing which the majority is, is the reason we used the technique of tokenization. Another technique we used was lexicon handling. In order to distinguish between good words and bad words, we needed to keep a dictionary of the good type of words. This allowed us to classify words as good or bad. It was crucial to ensure that just because a word was not in the good lexicon, does not mean that it should be classified as bad, for example words such as 'the' or 'and' should not increase the count for negative words. The classifier algorithm we used was K-Nearest Neighbors (KNN)

based. Originally, we planned to use a Naïve Bayes algorithm, however after evaluation of both methods, performance with our KNN model was much better than our Naïve Bayes.

In summary, we were given a data set of clothing reviews to analyze. The goal of our project was to find **how to develop a sentiment classifier that can assign reviews to a rating from 1-5.** Using sentiment analysis means analyzing text to determine if someone's opinion is positive or negative. Why this was important is because learning from reviews means understanding the business's customers/products more.

	A	B	C	D	E	F	G	H	I	J	K	L
1		Clothing ID	Age	Title	Review Tex	Rating	Recommen	Positive Fe	Division Na	Departmer	Class Name	
2	0	767	33		Absolutely	4	1	0	Initmates	Intimate	Intimates	
3	1	1080	34		Love this di	5	1	4	General	Dresses	Dresses	
4	2	1077	60	Some majo	I had such l	3	0	0	General	Dresses	Dresses	
5	3	1049	50	My favorit	I love, love,	5	1	0	General Pe	Bottoms	Pants	
6	4	847	47	Flattering s	This shirt is	5	1	6	General	Tops	Blouses	
7	5	1080	49	Not for the	I love tracy	2	0	4	General	Dresses	Dresses	
8	6	858	39	Cagrcoal sh	I aded this	5	1	1	General Pe	Tops	Knits	
9	7	858	39	Shimmer, s	I ordered tl	4	1	4	General Pe	Tops	Knits	
10	8	1077	24	Flattering	I love this c	5	1	0	General	Dresses	Dresses	
11	9	1077	34	Such a fun	I'm 5"5' an	5	1	0	General	Dresses	Dresses	
12	10	1077	53	Dress looks	Dress runs	3	0	14	General	Dresses	Dresses	

Part of the data set

Related Work:

How do baseline methods differ from your method?

When looking at the previous work regarding the automatic sentiment analysis of this dataset, there are multiple methods that use a different approach to the K-Nearest Neighbors model that was the most successful for us. The simplest method is using Naive Bayes as it can be the fastest in comparison to its competitors when regarding CPU power and the size of the dataset. Multinomial Naive Bayes works by utilizing Bayes theorem to predict sentiment by assuming that the features of the data set are independent. This assumes the use of a bag of words model which only concerns itself with the number of occurrences of each word and not their place in the review. This can then be used to determine if the sentiment of the review is positive or negative.

Another possible method of use is sentiment analysis using logistic regression. This technique works by representing text as a vector corresponding to vocabulary size. This can then be used to build a frequency dictionary that maps sentiment. The support vector machine (SVM) classifier is another good text classification method that works by mapping feature vectors into a higher dimensional feature space, nonlinearly. A separation between the features can then be found and the data is changed so that the separator can be used as a hyperplane. The last technique uses a decision tree learning approach, which uses child and root nodes to determine target values. Internal nodes in the model represent a check on a feature, branches outline the outcome of the check and the leaves represent the class distribution. This will result in a flowchart like structure where the model is able to appoint a label to any document.

Looking at the K-Nearest Neighbors technique which was the most successful in our case, it works by finding K nearest neighbors in training data, then using the labels with the highest similarity for prediction.

Solution/Method:

What was done

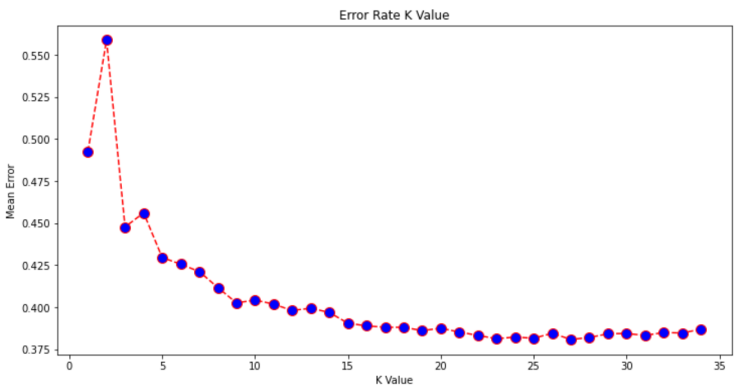
For our project, we performed the sentiment classification during the preprocessing stage. The predictive classifier we used was K-Nearest Neighbors. This was because as mentioned before, it would derive better results than other models such as the Gaussian Naive Bayes classifier. However, in our model we have a changing chosen K value. This was because the best K value to use would be different each time.

Tools and techniques

One technique we used was finding the best K value for KNN. This was done by running many KNN simulations on the same data, then calculating the error for each K value. We would track the least error made (the lowest point on the graph), then use that corresponding K for the actual predictions. The implementation and results are as shown.

```
#find a k value for knn- look for lowest point(error) on graph

minError = 1 #where is least erorr
minErrorK = 0 #which k to choose
errors = [0] #errors = []

# Calculate error for dif K values
for i in range(1,35):#(5,35):#(1, 40)

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(descriptive_train, target_train)
    pred_i = knn.predict(descriptive_test)
    errors.append(np.mean(pred_i != target_test))

    #print(errors[i-1],np.mean(pred_i != target_test),minError,minErrorK)
    if(errors[i]<minError): #if(errors[i-1]<minError):
        minError=errors[i]
        minErrorK=i

errors=errors[1:]
print("choose k = ", minErrorK)

plt.figure(figsize=(12, 6))
plt.plot(range(1, 35), errors, color='red', linestyle='dashed', marker='o', markerfacecolor='blue',

plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Code snippet




```
choose k =  27
```

Code results

In addition, some tools we used were Tokenization (for converting sentences into words) as well as Lemmatization (which would reduce words to their root word). For example, with the

use of Lemmatization, the words “flattering” and “flattered” are both reduced to “flatter”. This means that we don't need to consider every version of a word when traversing through the text.

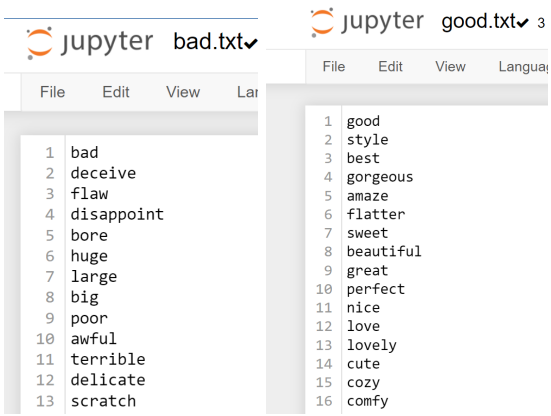
Data and Experiments:

Preprocessing

The preprocessing that was done to the data included reading files containing lists of “good” and “bad” words. Many of which are general words that can be associated with positive or negative meanings, such as “amazing” or “terrible”. But some were manually added after viewing specific words from the data, such as “chic”, “fab”, and “itchy”. These specifically reflect the audience and product of the business.

```
#read files #split text into good or bad meaning

def readFile(file_name):#read files for it
        file = open(file_name, "r")
        words = file.read().splitlines()
        file.close()
        return words


badWords= readFile("bad.txt")
print (badWords)

goodWords= readFile("good.txt")
print (goodWords)
```

Code snippet



Text files containing different words

Sentiment analysis was performed on the Title and Review Text features. We evaluated them by first replacing nan values with 0. Then traversing the text to find the number of good/bad words in each sentence. A calculation of **final_eval = num_good_words - num_bad_words** (the good minus the bad) was made. This means that the output was positive when finding more good words, and negative with more bad words in the sentence. We also attempted to handle the negation of sentences. For example, if “no” or “not” was added to a sentence, it would switch the final outcome. We also handle sentences with extra expression. Such as “!” or “too” being added to the sentence, would mean increasing the final outcome.

```
#title
title_evals= []


for i in range(len(sentiment_df_title)):#for each sentence
    num_good_words = 0
    num_bad_words = 0

    final_eval = 0

    was_negated = False
    was_expression = False


    if (sentiment_df_title.values[i]!=0):#not na

        sentence = sentiment_df_title.values[i]
        print(sentence)#debug
        words=word_tokenize(sentence)

        for j in range(len(words)): #for each word

            word = words[j].lower()
            rootWord = lem.lemmatize(word,"v") #root words

            #meaning of word
            if rootWord in goodWords :
                num_good_words+=1

            elif rootWord  in badWords:
                num_bad_words+=1



            elif rootWord in negateWords:
                was_negated = True
            elif rootWord in expressionWords:
                was_expression = True
```

```
#calc final eval
final_eval = num_good_words-num_bad_words


#check for negation or expresion
if(was_negated):
    final_eval = -final_eval

if(was_expression):
    final_eval = final_eval*2#*5 #*10
```

Code snippet

Here are some examples of running this process.

1. Bad: “Itchy” is a bad word, therefore the final outcome is negative.

```
Itchy tags
was_negated,was_expression =  False False
num_good_words, num_bad_words =  0 1
final_eval = -1
```

2. Negated: “Not impressed”. The word “impressed” is reduced to “impress”, which is a good word. But the “not” added makes it bad overall.

```
Not impressed...
was_negated, was_expression =  True False
num_good_words, num_bad_words =  1 0
final_eval = -1
```

3. Expression: “Great shirt!!!”. The word “Great” is positive. Then the use of an “!” makes the statement even more positive.

```
Great shirt!!!
was_negated, was_expression =  False True
num_good_words, num_bad_words =  1 0
final_eval = 2
```

4. Combination: “Not very flattering”. There is a good word that’s been negated and emphasized. Thus it leads to a larger negative value.

```
Not very flattering
was_negated, was_expression =  True True
num_good_words, num_bad_words =  1 0
final_eval = -2
```

Next, we created our descriptive and target features. The target feature we used is Rating. While the descriptive features we chose were: Title, Review Text, Positive Feedback Count, and Recommended IND. The descriptive features needed to drop any remaining features and replace the Title and Review Text features with their final evaluations. Finally, we randomly split the original data as 70% training : 30% testing data.

```
descriptive_features===============

   Recommended IND  Positive Feedback Count  titleNew  textNew
0                1                        0         0        3
1                1                        4         0        8
2                0                        0        -1        6
3                1                        0         2      -14
4                1                        6         1        6
```

Descriptive features

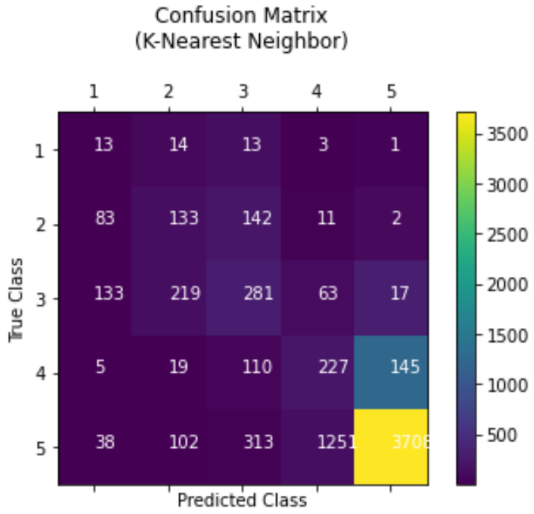
As a side note , we tested with different experiments such as using different K values, a weighted KNN model, Gaussian Naive Bayes, and different split amounts (80:20). But these were not as effective as our current solution.

We also used an external natural language processing library called TFIDF (term-frequency inverse document frequency) which is a method used to arrive at a numerical statistic which reflects how important a word is a certain body of text. In this case, we used a Vectorizer which gave every unique word in the set a certain integer value. A TFIDF transformer was used and each word in the text was given a score from 0-1 regarding its importance relative to the other words. A multinomial naive bayes was performed on these values and we found that when performed on the title data alone, it gave an f1 score of 0.578 and an overall accuracy of 64%. This may be limiting since the title data is not always available

Evaluation and Results:

Evaluation

For our solution, we attempted multiple methods including K-Nearest-Neighbor, Weighted K-Nearest-Neighbor, and the Naive Bayes' method. We chose these methods because they make the most sense intuitively. We did not try, for example, the ID3 method as the model would very likely be affected by overfitting and would take a lot of work to prune. We decided on using confusion matrix-based performance measures, as our target feature was **categorical**. After comparing the precision, recall, and F1-measure of KNN, weighted KNN, and Naive Bayes', we concluded that K-Nearest-Neighbor was the best fit as it had the best metrics. KNN had the best F1-score, maintaining the best balance between precision and recall. Naive Bayes achieved an F1-score of **0.5178** while KNN achieved an F1-score of **0.5643**.



Confusion matrix shows final ratings

For example:1,1 -> true positive of rating 1 (13).5,5 -> true positive of rating 5 (3708)

We also included accuracy as one of our evaluation metrics. While this is useful, it is important to note that it does not tell the whole story. We focused more on recall as we found it more important that our model is able to predict almost all members of a specific rating, while giving up some precision on whether a rating was 5 star or 4 star.

There were many descriptive features that did not improve the model. These features were categorical, and heavily contributed to the curse of dimensionality in our data. On top of that, since we are using KNN, data must be loaded into memory for the learning process. Thus we save both on space and dimensionality for removing these features. The removal of these features did not have any negative impact on the model's performance.

Conclusions

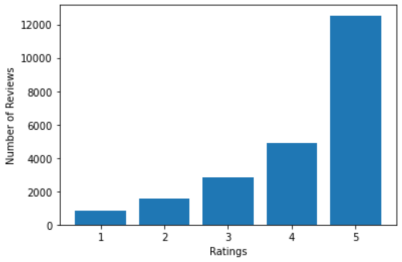
In conclusion, we have **developed a sentiment classifier that can assign reviews to a rating from 1-5.** The solution uses various techniques such as the use of different text files, Tokenization, Lemmatization, and calculating a final evaluation on sentences. Then we use a

KNN classifier (with various K values) to predict the rating based on various descriptive features.

	precision	recall	f1-score	support
1	0.05	0.30	0.08	44
2	0.27	0.36	0.31	371
3	0.33	0.39	0.36	713
4	0.15	0.45	0.22	506
5	0.96	0.69	0.80	5412
accuracy			0.62	7046
macro avg	0.35	0.44	0.35	7046
weighted avg	0.79	0.62	0.68	7046

Some statistical analysis

After running our experiment, we can observe that our result shows that a rating of 1 star has an f1score much lower (0.08) than a rating of 5 stars (with f1score= 0.80). This is either due to the data is highly imbalanced or because we are more biased in predicting the 5 star ratings.



Graph showing the imbalanced data