University of Sheffield

Text Processing Assignment

Jagpreet

December 8, 2022

Abstract

Performing Sentiment Analysis on two datasets to understand the strength and limitations of Bayesian text classification and Lexicon-based approach. I also attempted to implement my own rule-based system to improve upon the lexicon-based system given.

List of Tables

Contents

Chapter 1

STEP 2: Familiarise:

The code splits the Rotten Tomatoes Data into a training and test set in readFiles(), then builds the p(word—sentiment) model on the training data in trainBayes(), and finally applies Naive Bayes to the test data in testBayes().

1.1 Write a function which will print out Accuracy, Precision, Recall and F-measure for the test data.

I have defined a function called evaluation which takes in 8 arguments:

```python
#------------- Evaluation Function ---------------------

def evaluation(correct,correctpos,correctneg,totalpospred,totalnegpred,
    totalpos,totalneg,total):
    """
    Takes in 8 arguments and prints out Accuracy, Precision, and F1 score
    where (1)correct = number of values classified correctly(correctpos+
    correctneg){True Positve + True Negative}
          (2)correctpos,correctneg = correctly classified positive and
    negative values(TP and TN)
          (3)totalpospred,totalnegpred = number of values predicted as
    positive, and negative
          (4)totalpos,totalneg = total number of positive and negative values
          (5)Total = number of values (totalpos + totalneg)
    """
```

Code 1.1: Evaluation Function

and calculates :

1. $Accuracy = \frac{correct}{total}$

```
try:
        # Number of True Positive and True Negative values divided by
    total number of values:-
        accuracy = (correct)/float(total)
except ZeroDivisionError:
        accuracy = 0

```

Code 1.2: Evaluation Function - Accuracy

2. $Precision_{pos} = \frac{correctpos}{totalpospred}$

```
try:
        # Number of True Positives divided by Sum of True Positives and
    False Positives:-
        precision_pos = correctpos/float(totalpospred)
except ZeroDivisionError:
        precision_pos = 0
```

Code 1.3: Evaluation Function- Precision$_{pos}$

3. $Precision_{neg} = \frac{correctneg}{totalnegpred}$

```
try:
        # Number of True Negatives divided by Sum of True Negatives and
    False Negatives :-
        precision_neg = correctneg/float(totalnegpred)
except ZeroDivisionError:
        precision_neg = 0

```

Code 1.4: Evaluation Function- Precision$_{neg}$

4. $Recall_{pos} = \frac{correctpos}{totalpos}$

```
try:
        # Number of True Positives divided by Sum of True Positives and
    False Negatives :-
        recall_pos = correctpos/float(totalpos)
except ZeroDivisionError:
        recall_pos = 0

```

Code 1.5: Evaluation Function- Recall$_{pos}$

5. $Recall_{neg} = \frac{correctneg}{totalneg}$

```
try:
        # Number of True Negatives divided by Sum of True Negatives and
    False Positives :-
        recall_neg = correctneg/float(totalneg)
except ZeroDivisionError:
```

```python
        recall_neg = 0

```

Code 1.6: Evaluation Function- Recall$_{neg}$

6. $F1_{pos} = \frac{2*Precision_{pos}*Recall_{pos}}{Precision_{pos}+Recall_{pos}}$

```python
try:
        #Performance matric giving equal weight to both Precison and Recall :-
        f_measure_pos = (2 * ((precision_pos) * (recall_pos))/float(precision_pos + recall_pos))
except ZeroDivisionError:
        f_measure_pos = 0

```

Code 1.7: Evaluation Function- F-1 Score$_{pos}$

7. $F1_{neg} = \frac{2*Precision_{neg}*Recall_{neg}}{Precision_{neg}+Recall_{neg}}$

```python
try:
        f_measure_neg = (2 * ((precision_neg) * (recall_neg))/float(precision_neg + recall_neg))
except ZeroDivisionError:
        f_measure_neg = 0

```

Code 1.8: Evaluation Function- F-1 Score$_{neg}$

8. Printing Code:

```python
    # Prints Accuracy
    print("Accuracy_total:",accuracy*100)
    # Prints Precision for Positive sentiment and negative sentiment
    print("precision_pos:"+(str)(precision_pos*100)+"\nprecision_neg:"+(str)(precision_neg*100))
    # Prints Recall for Positive sentiment and negative sentiment
    print("recall_pos:"+(str)(recall_pos*100)+"\nrecall_neg:"+(str)(recall_neg*100))
    # Prints F-measure for Positive sentiment and negative sentiment
    print("f_measure_pos:"+(str)(f_measure_pos*100)+"\nf_measure_neg:"+(str)(f_measure_neg*100))


```

Code 1.9: Evaluation Function- Printing

9. Evaluation Call for testBayes:

```python
  evaluation(correct,correctpos,correctneg,totalpospred,totalnegpred,totalpos,totalneg,total)


```

Code 1.10: Evaluation Function Call

[1]

[1]Full Function in Sentiment.py in Appendices

1.2 Run the code and report the classification results.

Results of the evaluation of the Naive Bayes model on Test Data (rotten Tomatoes).

TestBayes							
Dataset	Accuracy	Precision_{pos}	Precision_{neg}	Recall_{pos}	Recall_{neg}	F-1 Score_{pos}	F-1 Score_{neg}
Testing Data(RT)	77.67	78.43	76.90	77.28	78.07	77.85	77.48

Table 1.1: Results of Running Evaluation Function on test Bayes on Test Data (RT).

Some Observations:

- As shown in Table 2.1, the Naive Bayes model which was trained on Rotten Tomatoes training data performs accurately about 89% of the time, On the test data It suffers an decrease in accuracy of 10%.
- An accuracy of 77 percent is an okay score as we know the model is balanced and has similar number of positive and negative words.
- The model is well balanced as all the metrics are in close range to each other signifying no one factor such as false positive or false negative influences the model more than others.

Chapter 2

STEP 3: Run Naive Bayes on other data:

Results of the evaluation of the Naive Bayes model on Training Data(Rotten Tomatoes), Test Data (rotten Tomatoes) and Nokia Data.

TestBayes							
Dataset	Accuracy	Precision_{pos}	Precision_{neg}	Recall_{pos}	Recall_{neg}	F-1 Score_{pos}	F-1 Score_{neg}
Training Data(RT)	89.03	89.42	88.66	88.50	89.56	88.96	89.11
Testing Data(RT)	77.67	78.43	76.90	77.28	78.07	77.85	77.48
Nokia Data	59.02	78.10	38.75	57.52	62.5	66.25	47.84

Table 2.1: Results of Running Evaluation Function on test Bayes on all three datasets.

2.1 What do you observe? Why are the results so different?

The model suffers a big loss in all metrics on the Nokia dataset from 77% on training data to 59%, Some of the reasons are:

- As the model was trained on Rotten tomatoes training data, and as Naive Bayes assumes independence of words from one other, the model suffers because the domain of knowledge of Nokia dataset is in Mobile Phone review, Whereas it was trained on the domain of movie reviews which can contain words and sentences which function as misnomers due to names and type of language used to describe Movies.
- The Nokia dataset is also unbalanced as it contains more than twice as much positive examples than negative ones, which is reflected in high Precision i.e 78% for Positives and low Precision for negatives i.e 38%.

- The model has better recall than precision signifying it is making many false negative predictions but is able to identify large number of relevant examples from teh dataset.
- For positive, the reverse is true where it is not identifying as much relevant examples from the dataset, but its identifications are more accurate.
- The disparity in F-1 scores also shows that the positive case have more impact on the model and shows it is biased towards positive sentiment.

If the model was trained on a better balanced dataset and in the domain of mobile reviews it will perform better.

Chapter 3

STEP 4: What is being learnt by the model?

3.1 Which are the most useful words for predicting sentiment? The code you have downloaded contains another function mostUseful() that prints the most useful words for deciding sentiment.

```
mostUseful(pWordPos, pWordNeg, pWord, 50)
```

Code 3.1: Most Useful Words

Output

```
NEGATIVE:
['unfunny', 'boring', 'badly', 'mediocre', 'routine', 'generic', 'poorly', 'mindless',
'pointless', 'disguise', 'stale', 'bore', "wasn't", 'shoot', 'dreary', 'annoying',
'offensive', 'stupid', 'meandering', 'harvard', 'plodding', 'disaster', 'unless',
'inept', 'amateurish', 'horrible', 'chan', 'product', 'fatal', 'lifeless', 'apparently',
'animal', 'pinocchio', 'flat', 'mixed', 'ill', 'junk', 'waste', 'banal', 'adam', 'sadly',
'tiresome', 'incoherent', 'stealing', 'uninspired', 'conceived', 'retread', 'dull',
'supposed', 'literally']

POSITIVE:
['enjoyed', 'wrenching', 'format', 'entertain', 'understands', 'heartbreaking', 'timely',
 'pianist', 'transcends', 'gradually', 'record', 'richly', 'smarter', 'sadness', 'iranian',
 'sides', 'scale', 'unexpected', 'flawed', 'jealousy', 'powerful', 'captures', 'resonant',
'tour', 'grown', 'polished', 'vividly', 'bittersweet', 'provides', 'touching', 'tender',
'detailed', 'lively', 'spare', 'respect', 'challenging', 'wonderful', 'wry', 'heartwarming',
'mesmerizing', 'wonderfully', 'gem', 'realistic', 'chilling', 'refreshingly', 'riveting',
'refreshing', 'intimate', 'inventive', 'engrossing']
```

3.2 Uncomment the call to mostUseful(pWordPos, pWordNeg, pWord, 50) at the bottom of the program, and run the code again. This prints the words with the highest predictive value. Are the words selected by the model good sentiment terms? How many are in the sentiment dictionary?

```
    mostUseful(pWordPos, pWordNeg, pWord, 50)



neg = ['unfunny', 'boring', 'badly', 'mediocre', 'routine', 'generic', 'poorly', 'mindless', 'pointless',
 'disguise', 'stale', 'bore', "wasn't", 'shoot', 'dreary', 'annoying', 'offensive', 'stupid', 'meandering',
 'harvard', 'plodding', 'disaster', 'unless', 'inept', 'amateurish', 'horrible', 'chan', 'product', 'fatal',
 'lifeless', 'apparently', 'animal', 'pinocchio', 'flat', 'mixed', 'ill', 'junk', 'waste', 'banal', 'adam',
 'sadly', 'tiresome', 'incoherent', 'stealing', 'uninspired', 'conceived', 'retread', 'dull', 'supposed', 'literally']

pos = ['enjoyed', 'wrenching', 'format', 'entertain', 'understands', 'heartbreaking', 'timely', 'pianist',
 'transcends', 'gradually', 'record', 'richly', 'smarter', 'sadness', 'iranian', 'sides', 'scale', 'unexpected',
 'flawed', 'jealousy', 'powerful', 'captures', 'resonant', 'tour', 'grown', 'polished', 'vividly', 'bittersweet',
 'provides', 'touching', 'tender', 'detailed', 'lively', 'spare', 'respect', 'challenging', 'wonderful', 'wry',
 'heartwarming', 'mesmerizing', 'wonderfully', 'gem', 'realistic', 'chilling', 'refreshingly', 'riveting', 'refreshing',
 'intimate', 'inventive', 'engrossing']

lis = {} #saves words from pos and neg which are in sentiment dictionary
posi,negi = 0,0
for i in sentimentDictionary.keys():
    if i in pos :

        lis[i] = 'positive'
        posi+=1 #number of positves in dictionary
    if i in neg:

        lis[i]= 'negative'
        negi+=1 #number of negatives in dictionary

print(len(lis))  #number of words in sentiment dictionary
print(lis) #most useful words in sentiment dictionary
print(posi)#number of positive words in sentiment dictionary and most useful
print(negi)#number of negative words in setiment dictionary and most useful
```

Code 3.2: mostUsefulwords in Sentiment Dictionary

The words chosen as most Useful are mostly accurate and are in correct sentiment Class such as :

- unfunny,boring,mindless,fatal,incoherent,pinocchio,stealing,dull,stupid etc in the negative class.
- enjoyed, format, entertain, understands,vividly,respect,wonderful etc in the positive class.

However many words are context dependent and may represent opposite sentiment value outside of the domain such as movie reviews or are neutral in representing any sentiment value, for example:

- routine(neutral), harvard(neutral), product(neutral), apparently(neutral), adam(neutral), literally(neutral) in the negative class.
- wrenching, format(neutral), heartbreaking(opposite), pianist(neutral),iranian(neutral),sides(neutral),scale and jealousy(opposite),challenging(opposite) in the positive class.

Output

```
{’engrossing’: ’positive’, ’enjoyed’: ’positive’, ’entertain’: ’positive’, ’gem’: ’positive’,
’heartwarming’: ’positive’, ’intimate’: ’positive’, ’inventive’: ’positive’, ’lively’: ’positive’,
’mesmerizing’: ’positive’, ’polished’: ’positive’, ’powerful’: ’positive’, ’realistic’: ’positive’,
’refreshing’: ’positive’, ’respect’: ’positive’, ’richly’: ’positive’, ’smarter’: ’positive’,
’tender’: ’positive’, ’timely’: ’positive’, ’wonderful’: ’positive’, ’wonderfully’: ’positive’,
’annoying’: ’negative’, ’badly’: ’negative’, ’banal’: ’negative’, ’bore’: ’negative’,
’boring’: ’negative’,’challenging’: ’positive’, ’disaster’: ’negative’, ’dreary’: ’negative’,
’dull’: ’negative’, ’fatal’: ’negative’, ’flawed’: ’positive’, ’heartbreaking’: ’positive’,
’horrible’: ’negative’, ’incoherent’: ’negative’, ’inept’: ’negative’,’jealousy’: ’positive’,
’junk’: ’negative’, ’lifeless’: ’negative’,’mediocre’: ’negative’, ’mindless’: ’negative’,
’offensive’:’negative’, ’pointless’: ’negative’, ’poorly’: ’negative’, ’sadly’: ’negative’,
’sadness’: ’positive’, ’stale’: ’negative’,’stealing’: ’negative’, ’stupid’: ’negative’,
’tiresome’: ’negative’, ’unexpected’: ’positive’,
’waste’: ’negative’}
Number of words in sentiment dictionary which are also in mostuseful: 51
Number of postive words in sentiment dictionary and mostUseful: 26
Number of negative words in sentiment dictionary and mostUseful: 25
```

There are 51 words in the sentiment dictionary which are also mostUseful, 26 of which are positive and 25 negative.

Chapter 4

STEP 5: How does a rule-based system compare?

4.1 Add some code for the function testDictionary() which will print out Accuracy, Precision, Recall and F-measure for the test data. Uncomment out the three lines towards the end of the program that call the function testDictionary() and run the program again. All this code does is add up the number of negative and positive words present in a review and predict the larger class.

```
print("pos:"+(str)(totalpos)+"\nneg:"+(str)(totalneg))
print("pospred:"+(str)(totalpospred)+"\nnegpred:"+(str)(totalnegpred))
evaluation(correct,correctpos,correctneg,totalpospred,totalnegpred,totalpos,
    totalneg,total) # Calls the evaluation function defined in 1.1 and Code
    1.1
```

Code 4.1: TestDictionary Evaluation Call

```
print("\n testdata:")
testDictionary(sentencesTest,  "Films  (Test Data, Rule-Based)\t",
    sentimentDictionary, 1)
print("\ntestDictionary on Train Data")
testDictionary(sentencesTrain,  "Films (Train Data, Rule-Based)\t",
    sentimentDictionary, 1)
print("\ntestDictionary on Nokia:")
testDictionary(sentencesNokia, "Nokia   (All Data, Rule-Based)\t",
    sentimentDictionary, 1)
```

Code 4.2: TestDictionary Run

4.2 How does the dictionary-based approach compare to Naive Bayes on the two domains? What conclusions do you draw about statistical and rule-based approaches from your observations?

The differences in evaluation metrics are as follows:

TestBayes v/s TestDict							
Dataset	Accuracy	Precision$_{pos}$	Precision$_{neg}$	Recall$_{pos}$	Recall$_{neg}$	F-1 Score$_{pos}$	F-1 Score$_{neg}$
Train (Bayes)	89.03	89.42	88.66	88.50	89.56	88.96	89.11
Train (Dict)	63.9	62.2	66.2	70.9	57.9	66.3	61.2
Test (Bayes)	77.67	78.43	76.90	77.28	78.07	77.85	77.48
Test (Dict)	61.7	59.9	64.3	69.03	54.8	64.1	59.2
Nokia (Bayes)	59.02	78.10	38.75	57.52	62.5	66.25	47.84
Nokia (Dict)	77.8	85	62	82	67	83	64

Table 4.1: Comparing Naive bayes performance against testDictionary.

Following Conclusions can be drawn:

1. For the Rotten Tomatoes training Data and Test Data :
   - The evaluation metrics suffer across the board, as the simplistic nature of approach fails to realise the innuendos and implied meaning of the sentences and words used to review movies, as they often involve more emotional and complex sentiments, and as there is no prior training or exposure, the model suffers.
2. For the Nokia Data:
   - As the sentences and words in the Nokia Dataset are mostly about inanimate objects and simple experiences of using the mobile, thereby limiting the range of words, Nokia dataset is very small compared to the other datasets. However despite the increase in accuracy, the model still is biased towards positive classification as shown by disapriteis in precision,recall,F-1 score for the positive and negative class.

4.3 Write a new function to improve the rule-based system, e.g., to take into account negation, diminisher rules, etc. Run the program again and analyse the results on both datasets.

I define two functions, namely, rules() and improved_rule_based().

- rules() Function for applying rule effects:

```
#---------------------------Rule applying Function Function----------------------------------
def rules(temp):
    """
    Takes in a list of  integers and strings and performs rule-based scoring
    Returns the list to assign score of a sentence
    Uses recursion to apply the effects
    Values preceeding get a preference for manipulation
    Further improvement can be achieved using conjunctions to treat compound sentences efficiently
    Order of Precedence adopted is : Intensification >> Dimnishing >> Negation
    """
    for i in temp:
        #---------------applying intensification------------
        if (i == 'i'):
            int_ = temp.index(i)
            try:

                if type(temp[int_-1]) == int:
                    if temp[int_-1]>0:
                        temp[int_-1] = temp[int_-1] + 1
                        temp.pop(int_)
                        return rules(temp)
                    if temp[int_-1]<0:
                        temp[int_-1] = temp[int_-1] - 1
                        temp.pop(int_)

                        return rules(temp)
                if type(temp[int_+1]) == int:
                    if temp[int_+1]<0:
                        temp[int_+1] = temp[int_+1] - 1
                        temp.pop(int_)
                        return rules(temp)
                    if temp[int_+1]>0:
                        temp[int_+1] = temp[int_+1] + 1
                        temp.pop(int_)

                        return rules(temp)
            except IndexError:
                pass
```

```
    #---------------applying diminshing effect------------
    for i in temp:
        if (i == 'd'):
            dim_ = temp.index(i)
            try:
                if type(temp[dim_-1])== int:
                    if temp[dim_-1]<0:
                        temp[dim_-1] = temp[dim_-1] + 1
                        temp.pop(dim_)
                        return rules(temp)
                    if temp[dim_-1]>0:
                        temp[dim_-1] = temp[dim_-1] -1
                        temp.pop(dim_)

                        return rules(temp)
                if type(temp[dim_+1])== int:
                    if temp[dim_+1]>0:
                        temp[dim_+1] = temp[dim_+1] -1
                        temp.pop(dim_)
                        return rules(temp)
                    if temp[dim_+1]<0:
                        temp[dim_+1] = temp[dim_+1] + 1
                        temp.pop(dim_)

                        return rules(temp)

            except IndexError:
                pass
    #--------------------------------Negation can be applied backward or forward depending upon the negation word used----
    for i in temp:
        if (i == 'nf'):

            ind = temp.index(i)
            try:
                temp.pop(ind)
                temp.insert(ind+1,-3)
                return rules(temp)
            #-------To acccount for Index errors
            except IndexError:
                pass
        if (i == 'nb'):
            ind = temp.index(i)
            try:
                temp.pop(ind)
                temp.insert(ind-1,-3)
                return rules(temp)
            except IndexError:
                pass
    #-----------------------Finally returns the manipulated list--------------------------
    return temp
#---------------------------End Rule applying Function----------------------------------
```

Code 4.3: Rules application Function

- improved_rule_based() is implemented as follows:

```
#--------------------------Improved rule based Function ---------------------------------
def improved_rule_based(sentencesTest,threshold):
    """
    takes in sentences, and builds sentiment dictionary from Positive-words and negative-words txt files
    Breaks down the sentences into words and lemmatizes them which can be analyzed against a new sentiment dictionary
    to calculate sentiment.
    The new sentiment dictionary takes out or manipulates values of Negation,Intensifiers and dimnishers words in the
    positive and negative words txt files and removes words (prepos, conjunc) which dont contribute to sentiment value
    and lemmatizes it.
    Conjunction words can also be added to better deal with compound sentences
    After processing the sentences and taking out a intermediate list which contains a crude structure of the sentence,
    we pass the list to another function rules() which performs scoring based on our rules.
    We finally sum the list returned frokm rules() and divide by the number of words considered for sentiment to give out
    the final score which is compared to threshold to classify as positive or negative.
    """
    total_poserrors =0
    total_negerrors =0
    total=0
    correct=0
    totalpos=0
    totalneg=0
    totalpospred=0
    totalnegpred=0
    correctpos=0
    correctneg=0
    new_sentiment_dictionary = {} # to process the sentimentdictionary used by Test Dictionary Function
    conjunc_str = lemma('for and but or yet so because')
    negation_str = lemma(" rather wasnot didnot wouldnot shouldnot werenot donot havenot hasnot wont hadnot musnot without doesnot couldnot   hardly few seldom   even though no not neither never none nobody nowhere nor nothing cannot somehow")
    prepos_str = lemma('an all use be words at to are do does  many you can  that this a the in of  to for with on at from by about as into through after over between out against during without before under around among')
    intense_str = lemma('huge big very quite most really extremely amazingly exceptionally incredibly particularly remarkably unusually likely like')
    dimnish_str = lemma('just however perhaps barely somewhat rarely less little scarcely seldom lack lacking')
    #--------------------------------------------Performing
```

```
   lemmatizationusing Spacy lemmatizer-------
    conjunc,negation,prepos,intense,dimnish = [],[],[],[],[]
    for i in conjunc_str:
        conjunc.append(i.lemma_)
    for i in negation_str:
        negation.append(i.lemma_)
    for i in prepos_str:
        prepos.append(i.lemma_)
    for i in intense_str:
        intense.append(i.lemma_)
    for i in dimnish_str:
        dimnish.append(i.lemma_)
    #--------------------------------Updating and lemmatizing sentiment
   dictionary--------------------
    nposDictionary = open('positive-words.txt', 'r', encoding="ISO
   -8859-1")  #dictionary
    nposWordList = re.findall(r"[a-z\-]+", nposDictionary.read())
    pos_pre = ' '.join(nposWordList)
    pos = lemma(pos_pre)
    pos_list = []
    for i in pos:
        pos_list.append(i.lemma_)

    nnegDictionary = open('negative-words.txt', 'r', encoding="ISO
   -8859-1")
    nnegWordList = re.findall(r"[a-z\-]+", nnegDictionary.read())
    neg_pre = ' '.join(nnegWordList)
    neg = lemma(neg_pre)
    neg_list = []
    for i in neg:
        neg_list.append(i.lemma_)
    #---------------------------------------Assigning Values to words
   --------------------------
    for i in pos_list:
        new_sentiment_dictionary[i] = 3
    for i in neg_list:
        new_sentiment_dictionary[i] = -3
    #--------------------------------------Removing and changing values
   of words which are in our prepos,conjunc,
    #negation,dimnisher,intensifier list.

    for j in new_sentiment_dictionary.copy():
        if j in prepos:
            del new_sentiment_dictionary[j]
        if j in negation:

            del  new_sentiment_dictionary[j]
        if j in intense:

            del new_sentiment_dictionary[j]
        if j in dimnish:

            del  new_sentiment_dictionary[j]
        if j in conjunc:

            del new_sentiment_dictionary[j]
```

```
    #---------------------------------------------------pre-processing & Lemmatizing sentences---------------------------
    sentences_preprocessed = {}
    for i,j in sentencesTest.items():
        Words_2 = re.findall(r"[\w']+", i)
        word_str = lemma(' '.join(Words_2))
        words_1 =[]
        for l in word_str:
            words_1.append(l.lemma_)
        k = " ".join(m for m in words_1)
        sentences_preprocessed[k] = j
            #----------------------------Pre-Processing Done---------------------------------------#
            #----------------------------Sentence Processing Starts--------------------------------
    Words = []
    for sentence, sentiment in sentences_preprocessed.items():
        Words.append(sentence)
        score=0
        length = 0
        Words_final = []
        for i in Words:
            Words_final = (i.split())
        temp = [] #storing and processing the words from the sentence to apply rules based scoring later
        for word in Words_final:
            j = 0
            if word in new_sentiment_dictionary:
                length +=1
                j=new_sentiment_dictionary[word]
                temp.append(j)
                j = 0
            if word in negation:
                length +=1
                if word == 'not':
                    temp.append('nf')
                else:
                    temp.append('nb')
            if word in intense:
                temp.append('i')
                length +=1
            if word in dimnish:
                temp.append('d')
                length +=1
             #----------------------------Words-Processing Done------------------------------------------#
             #----------------------------Applying effect of negation, intensifier and dimnisher----------
        temp_ruled = rules(temp)#<<<<<<<<<<<<-----using function rules defined before
             #----------------------------Rules Applied---------------------------------------------------#
             #----------------------------Final Score Calculation-----------------------------------------
        for i in temp_ruled:
            if type(i) == int:
```

```
                score+= i
        if length != 0:
            score = score/float(length)
        total+=1
        #---------------------------Final Score Calculation Done ---------------------------------------------#
        #---------------------------Analyzing performance and Printing Errors when 1---------------------------------------------------
        if sentiment=="positive":
            totalpos+=1
            if score>=threshold:
                correct+=1
                correctpos+=1
                totalpospred+=1
            else:
                correct+=0
                totalnegpred+=1
                if PRINT_ERRORS:
                    print ("ERROR (pos classed as neg %0.2f):" %score)
                    print("Score:",score)
                    total_poserrors +=1
        else:
            totalneg+=1
            if score<threshold:
                correct+=1
                correctneg+=1
                totalnegpred+=1
            else:
                correct+=0
                totalpospred+=1
                if PRINT_ERRORS:
                    print ("ERROR (neg classed as pos %0.2f):" %score + sentence)
                    print("Score",score)

                    total_negerrors += 1
        #---------------------------Analyzing performance done -------------------------------------------------#
        #-----------------------------Calling Evalution Function -----------------------------------------------
    print("pos:"+(str)(totalpos)+"\nneg:"+(str)(totalneg))
    print("pospred:"+(str)(totalpospred)+"\nnegpred:"+(str)(totalnegpred))
    evaluation(correct,correctpos,correctneg,totalpospred,totalnegpred,totalpos,totalneg,total)

    #print("poserrors:"+(str)(total_poserrors)+"\nnegerrors:"+(str)(total_negerrors))

#---------------------------End improved rule based Function ----------------------------------
```

Code 4.4: Improved rule based function

4.3.1 Results

```
print("\nimproved rule based:\n")
improved_rule_based(sentencesNokia,-0.5)
improved_rule_based(sentencesTrain,-0.2)
improved_rule_based(sentencesTest,-0.1)
```

Code 4.5: Implement imrpoved rule based

Improved Rule based(IRB)v/s TestDict							
Dataset	Accuracy	Precision_{pos}	Precision_{neg}	Recall_{pos}	Recall_{neg}	F-1 Score_{pos}	F-1 Score_{neg}
Train (Dict)	63.9	62.2	66.2	70.9	57.9	66.3	61.2
Train (IRB)	64.6	65.28	64.06	63.45	65.88	64.3	64.9
Test (Dict)	61.7	59.9	64.3	69.03	54.8	64.1	59.2
Test (IRB)	64.3	64.2	64	63	65.7	64.9	64.9
Nokia (Dict)	77.8	85	62	82	67	83	64
Nokia (IRB)	79.1	84.5	65.3	85.4	63.7	85.02	64.55

Table 4.2: improved rule based v/s testdict().

Observations:

1. For Rotten Tomatoes Train Data:

    - The improved rules model performs slightly better in most parameters, and is overall more balanced than the testDict() function as the recall and precision of both classes are similar as well as F-1 scores.

2. For Rotten Tomatoes Test Data:

    - The results are similar to the Training Data with slight improvement and a more balanced model.

3. For Nokia Data:

    - The model also performs better on the Nokia Dataset, however due to the inherent bias in the dataset, then model also inherits the bias in similar fashion to testDict()

4. The sentiment dictionary produced by this model, and process of using lemmatisation on the input data may allow wider application of the model to anlyse texts from different domains.

5. However due to the process of lemmatisation and recursion to apply rule effects to sentences the improved model runs slower for what can be considered as marginal improvements.

Chapter 5

STEP 6: Error Analysis.

Comment out all but one of the testBayes/testDictionary calls. At the top of the program, `set PRINT_ERRORS=1`

5.1 Run the program again, and it will print out the mistakes made. List the mistakes in the report

```
testBayes(sentencesNokia, "Nokia   (All Data,  Naive Bayes)\t", pWordPos,
    pWordNeg, pWord,0.3)
testDictionary(sentencesNokia, "Nokia   (All Data, Rule-Based)\t",
    sentimentDictionary, 1)
```

Code 5.1: running testbayes on Nokia and testdict on Nokia for error analysis

5.2 Please explain why the model is making mistakes (e.g., analyse the errors and report any patterns or generalisations).

1. Mistakes made bye testBayes() on Nokia Dataset:

   - Mistakes due to threshold: These mistakes are also affected by biased nature of dataset which contains more positive cases than negative ones.

```
                 ERROR (pos classed as neg 0.01):there is much
    which has been said in other reviews about the features of
    this phone , it is a great phone , mine worked without any
    problems right out of the box .
                 ERROR (pos classed as neg 0.01):the infrared is a
     blessing if you have a previous nokia and want to transfer
    your old phone book to this phone , saved me hours of re-
    entering my numbers .
                 ERROR (pos classed as neg 0.00):while i like the
    performance of the phone in every regard , i would buy
    another one solely upon the apparent indestructibility of it
    .
```

```

```

Code 5.2: Error analysis Nokia Bayes

- These errors can change and switch depending upon the value of the threshold.

- Inherent Naive Bayes Weakness: These are due to failure of Naive Bayes model to appreciate relations between words,structure and semantics.

```
             ERROR (neg classed as pos 0.51):when talking the voice is not very clear .
             ERROR (neg classed as pos 0.68):one complaint ... the screen is too easily scratched !
             ERROR (pos classed as neg 0.09):but , i would definately recommend this phone .
             ERROR (pos classed as neg 0.24):the nokia 6610 is a relatively new phone , and what a great phone it is .
             ERROR (pos classed as neg 0.15):1 month , no problems , great phone i 'm very pleased with my 6610 phone .

```

Code 5.3: Inherent errors Nokia Bayes

- All these are result of not being able to gauge and deal with significance of conjunctions and prepositions which provide the sentence its meaning. Compound sentences may mean different things when interpreted as a whole than taking individual words as separate and inferring their meaning.

2. Mistakes made by testDict() on Nokia Dataset:

- The Test Dict() fails to appreciate the impact of intensifiers,negation, dimnishers on the meaning and is reflected in the following examples:

```
     {Negation ignored} >> ERROR (neg classed as pos 1.00):the headset that comes with the phone has good sound volume but it hurts the ears like you cannot imagine !
     {Compound Statement failure} >>ERROR (pos classed as neg -1.00):but , then again , the ringer can be so loud that i heard it ringing inside my office , when i was already out on the street .. .
     {Negation ignored}>>
     ERROR (neg classed as pos 1.00):when talking the voice is not very clear .
     {Negation ignored}>> ERROR (neg classed as pos 1.00):the volume level of the phone is not all that good .
     {Conjunction and Negation}>>>
     ERROR (neg classed as pos 2.00):i have excellent hearing but the volume level on this phone is especially quiet .

```

Code 5.4: Test Dict Nokia errors

- Cannot handle singular or double word reviews properly:

```
             ERROR (pos classed as neg 0.00):- speakerphone
             ERROR (pos classed as neg 0.00):small size .
```

```
            ERROR (neg classed as pos 1.00):its quiet .


```

Code 5.5: Single word testdict error Nokia

- Implied meaning is not evaluated properly:

```
            ERROR (neg classed as pos 2.00):this model does
    have the traditional key arrangement , it 's just that they
    are really close to one another , and have unconventional
    shapes , so it takes a big getting used to for someone like
    me with big hands .

```

Code 5.6: Implied errors testdict nokia

  Here the implied meaning is that because the keys are close together it is harder for someone with big hands to use, But the model interprets it as a positive sentiment due to words such as have, does

```
            ERROR (neg classed as pos 2.00):so loud , really
    , that it does n't work terribly well as a silent ringer
    option .

```

Code 5.7: Implied errors testdict nokia$_2$

- Limitations due to limited number of words in positive-words.txt and negative-words.txt to draw from for comparison, Model fails to evaluate words which are outside its source. Use of lemmatisation can help in this and make the source sentiment dictionary more applicable.

Appendices

Appendix A

List of Codes

Appendix B

Sentiment.py

```python
def evaluation(correct,correctpos,correctneg,totalpospred,totalnegpred,totalpos,totalneg,total):
    """
    Takes in 8 arguments and prints out Accuracy, Precision, and F1 score
    where (1)correct = number of values classified correctly(correctpos+correctneg){True Positve + True Negative}
          (2)correctpos,correctneg = correctly classified positive and negative values(TP and TN)
          (3)totalpospred,totalnegpred = number of values predicted as positive, and negative
          (4)totalpos,totalneg = total number of positive and negative values
          (5)Total = number of values (totalpos + totalneg)
    """
    try:
        # Number of True Positive and True Negative values divided by total number of values:-
        accuracy = (correct)/float(total)
    except ZeroDivisionError:
        accuracy = 0
    try:
        # Number of True Positives divided by Sum of True Positives and False Positives:-
        precision_pos = correctpos/float(totalpospred)
    except ZeroDivisionError:
        precision_pos = 0
    try:
        # Number of True Negatives divided by Sum of True Negatives and False Negatives :-
        precision_neg = correctneg/float(totalnegpred)
    except ZeroDivisionError:
        precision_neg = 0
    try:
        # Number of True Positives divided by Sum of True Positives and False Negatives :-
        recall_pos = correctpos/float(totalpos)
    except ZeroDivisionError:
        recall_pos = 0
    try:
        # Number of True Negatives divided by Sum of True Negatives and False
```

```
 Positives :-
        recall_neg = correctneg/float(totalneg)
    except ZeroDivisionError:
        recall_neg = 0
    try:
        #Performance matric giving equal weight to both Precison and Recall :-
        f_measure_pos = (2 * ((precision_pos) * (recall_pos))/float(precision_pos + recall_pos))
    except ZeroDivisionError:
        f_measure_pos = 0
    try:
        f_measure_neg = (2 * ((precision_neg) * (recall_neg))/float(precision_neg + recall_neg))
    except ZeroDivisionError:
        f_measure_neg = 0
    # Prints Accuracy
    print("Accuracy_total:",accuracy*100)
    # Prints Precision for Positive sentiment and negative sentiment
    print("precision_pos:"+(str)(precision_pos*100)+"\nprecision_neg:"+(str)(precision_neg*100))
    # Prints Recall for Positive sentiment and negative sentiment
    print("recall_pos:"+(str)(recall_pos*100)+"\nrecall_neg:"+(str)(recall_neg*100))
    # Prints F-measure for Positive sentiment and negative sentiment
    print("f_measure_pos:"+(str)(f_measure_pos*100)+"\nf_measure_neg:"+(str)(f_measure_neg*100))

#----------------------------End of Evaluation Function ----------------#
```

Code B.1: Sentiment.py