NLP Assignment-3

Akshala Bhatnagar | 2018012
Arundhati Bhattacharya | 2018225

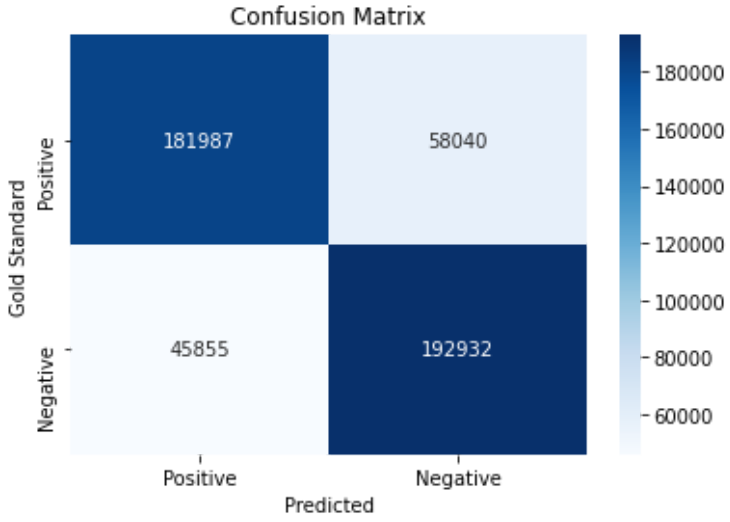
A. Sentiment Analysis of Tweets

a. Naive Bayes

The data (Train and Test) was first loaded into Pandas dataframes.

Preprocessing: (Done on the entire dataset) Every tweet was preprocessed to remove any URLs or handles (@_). Then it was lowered. The tweets were also tokenized (using nltk.tokenize.TweetTokenizer).

A dictionary is made, where the words are the keys and the value is a list with 3 elements containing the number of times the word occurred for each of the labels (positive, negative, neutral). The vocabulary is the list of keys in this dictionary. For sentences in the test data, the probability of belonging to each of the labels is calculated using the Naive Bayes formula along with Laplace smoothing. The label corresponding to the highest probability is assigned to the sentence.



Confusion Matrix for Naive Bayes Model

b. Decision Tree, MLP, SVM

The data (Train and Test) was first loaded into Pandas dataframes.

Preprocessing: (Done on the entire dataset) Every tweet was preprocessed to remove any URLs or handles (@_). Then it was lower-cased and POS-tagged. The tweets were also tokenized (using nltk.tokenize.TweetTokenizer) and then joined with the ' * ' string (to be used to create Non-contiguous Bigrams).

Features Implemented:

N-gram: Only Unigram, contiguous Bigram, non-contiguous Bigram were implemented (To save computation time). Their feature vectors were created using sklearn.feature_extraction.text.CountVectorizer.

POS: The preprocessed POS-tagged (of training+testing data) tweets were used to create count-vectors of POS tags using sklearn.feature_extraction.text.CountVectorizer.

Count Hashtags: Implemented by doing a simple Regex search for '#' in a tweet.

Count Words in all Caps: Implemented by counting tokens that have all letters in Caps.

Check Punctuation (?,!) in:

a. Entire tweet: Implemented by doing a regex search for a contiguous sequence of '?' and '!' characters and reporting count.
b. '?' in last token: Regex on last token
c. '!' in the last token: Regex on the last token

Count elongated words: Regex search in for tokens with one or more letters repeated

Check emoticons:

a. With Positive Sentiment
b. With Negative sentiment
c. With Positive Sentiment in the last token
d. With Negative Sentiment in the last token

All were implemented using Regex with pattern adopted from http://sentiment.christopherpotts.net/tokenizing.html#emoticons

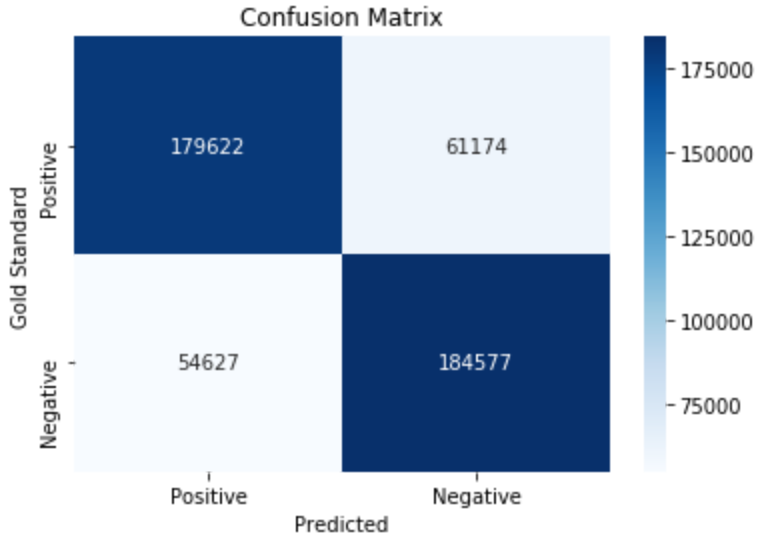
Count Negated contexts: By the definition of Negated contexts, it is sufficient to count the number of tokens that are negated. This was checked against a list of negated words from nltk.sentiment.vader.VaderConstants.

Lexicon features: 5 lexicons were used to score every token in a tweet and the following features were extracted from the scores:

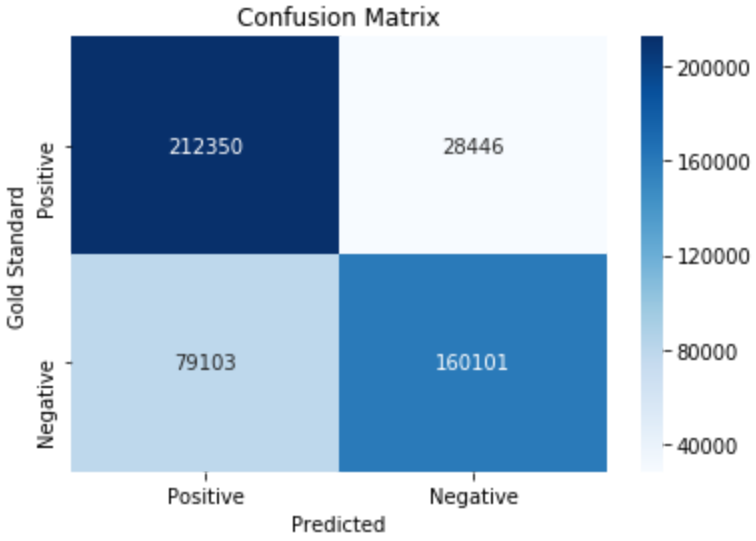
a. Count of tokens with a positive score
b. Total score of all tokens in the tweet
c. Maximum score of a token in the tweet
d. Score of the last token with a positive score in the tweet

Evaluation

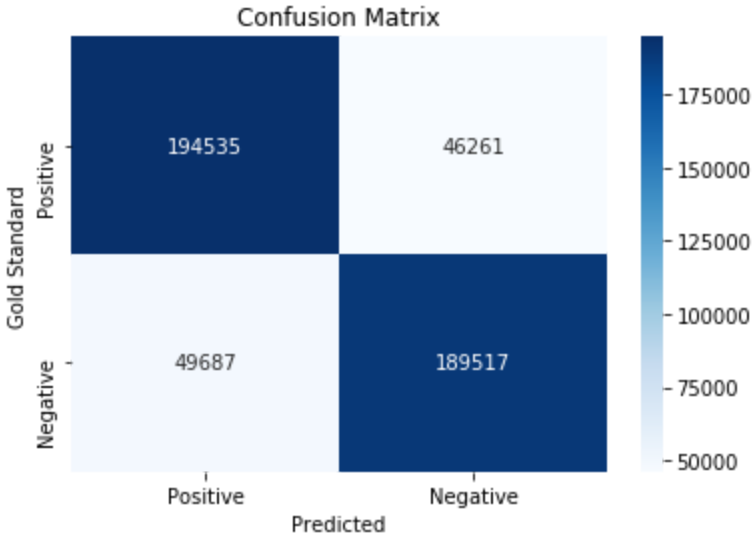
	Accuracy	Precision	Recall	F-Score
Decision Tree	0.7549625	0.755441599	0.755031943	0.754879211
MLP	0.775939583	0.788860672	0.775587089	0.773252908
SVM	0.800108333	0.800172054	0.800082461	0.800086485
Naive Bayes	0.783015951	0.783740626	0.783080401	0.782902515



Confusion Matrix for Decision Tree Model



Confusion Matrix for MLP model



Confusion Matrix for SVM model

B. Emotion Intensity Prediction

The data (Train and Test) was first loaded into Pandas dataframes.

Preprocessing: (Done on the entire dataset) Every tweet was preprocessed to remove any URLs or handles (@_). Then it was lowered. The tweets were also tokenized (using nltk.tokenize.TweetTokenizer).

Features Implemented:

Lexicon Features:

a. *VADER sentiment*: This is calculated using the python library vaderSentiment.vaderSentiment.SentimentIntensityAnalyzer which gives a dictionary with polarity scores for positive, negative, neutral, and compound.
b. *Polar word count*: This is calculated using *MPQA* and *Bing Liu lexicon*. Matching is done with these lexicons to find the number of positive and negative words in the tweet.
c. *Aggregate polarity score*: This is calculated using *Sentiment140*, *AFINN*, and *Sentiwordnet* lexicon. The polarity score is calculated for each word in the tweet which is matching the lexicon. Finally, the aggregate of all positive and negative scores is considered.
d. *Aggregate polarity score (Hashtags)*: The same method as above is used with *the NRC Hashtag Sentiment lexicon*.
e. *Emotion word count*: This is calculated using *the NRC Word-Emotion Association lexicon*. The count of words corresponding to the 10 emotions present in the lexicon is taken.
f. *Aggregate emotion score*: This is calculated using *NRC-10 Expanded lexicon*. For each word, the score associated with each emotion is found. The Sum of score corresponding to each emotion in the lexicon is taken.
g. *Aggregate emotion score (Hashtags)*: The same method as above is used with *the NRC Hashtag Emotion Association lexicon*.

h. *Emoticon score*: This is calculated using afinn.AFINN. Aggregate positive and negative scores for emoticons matching are taken.
i. *Negation count*: This is calculated using nltk.sentiment.vader.VaderConstants. A count of words matching the negated words in this class is taken.

N-gram: This is calculated using sklearn.feature_extraction.text.CountVectorizer. Unigrams and bigrams are considered. The feature vector contains the count of the n-grams in the tweets.

Finally using the above-explained lexicon and n-gram features, the emotion intensity is predicted using decision tree, SVM, and MLP.

Anger

	Pearson correlation between predicted and actual	Spearman correlation between predicted and actual
Decision Tree	0.44956275112189437	0.43260227102135124
SVM	0.6488937227580774	0.6352648421037835
MLP	0.5771913961157183	0.5602679163030928

	Pearson correlation for gold scores in range 0.5-1 between predicted and actual	Spearman correlation for gold scores in range 0.5-1 between predicted and actual
Decision Tree	0.2691760120311348	0.258314184728041
SVM	0.47238428634469704	0.4656535894441568
MLP	0.4316016787606326	0.40732968404064007

Average Pearson correlation	0.5585492899985635
Average Spearman correlation	0.5427116764760759
Average Pearson correlation for gold scores in range 0.5-1	0.3910539923788215
Average Spearman correlation for gold scores in range 0.5-1	0.37709915273761263

Joy

	Pearson correlation between predicted and actual	Spearman correlation between predicted and actual
Decision Tree	0.4116888231933479	0.4023221044794858
SVM	0.7066522032999258	0.7077763520340412
MLP	0.4868568564573238	0.48422928377941477

	Pearson correlation for gold scores in range 0.5-1 between predicted and actual	Spearman correlation for gold scores in range 0.5-1 between predicted and actual
Decision Tree	0.29941938438414056	0.30335213170595593
SVM	0.46013166515902965	0.46098106747522316
MLP	0.30605551140605713	0.25519911124799194

Average Pearson correlation	0.5350659609835325
Average Spearman correlation	0.5314425800976473
Average Pearson correlation for gold scores in range 0.5-1	0.35520218698307576
Average Spearman correlation for gold scores in range 0.5-1	0.3398441034763903

CONTRIBUTION OF EACH MEMBER:

1. **Akshala Bhatnagar:**
    a. **Implemented Naive Bayes for Sentiment Analysis**
    b. **Emotional Intensity Prediction and Evaluation of models (Pearson and Spearman Correlation)**
2. **Arundhati Bhattacharya**
    a. **Sentiment Analysis and Evaluation of models (Confusion matrix, Accuracy, Class-wise and Macro average precision, recall and F-score)**