**Natural Language Processing Assignment 2**

**Name: Shiang Hu**

**PID: A53267858**

**Supervised: Improve the Basic Classifier:**

***stop\_words*:**

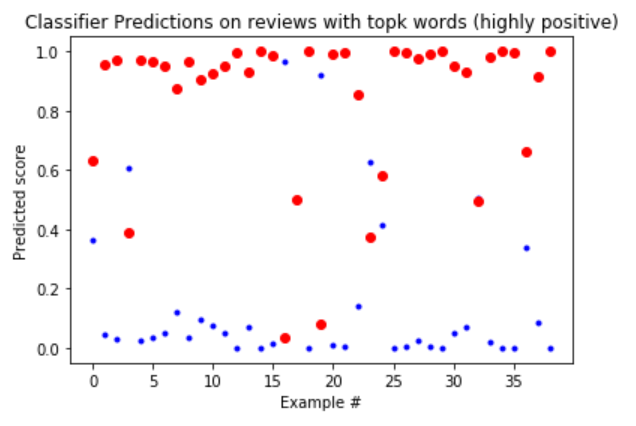
Select words that are trivial, and set them to stop words. First, we produce a naïve count\_vector model, we then sort the coefficient trained by logistic regression and select the top k highest coefficients meaning the top k most influential coefficients for positive prediction, and top k lowest coefficients meaning the top k most influential coefficients for negative prediction, and by ruling out these words in our training bag of words what’s left are trivial words. We set the “stop\_words” parameter in CountVectorizer to these trivial words so that these words will be set to stop\_words at training. We experiment of several k values and try to find the best k value to set.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | 0 | 500 | 1000 | 2000 | 3000 | 3750 | 4000 | 4500 |
| accuracy on dev | 0.777292 | 0.744541 | 0.768558 | 0.762008 | 0.770742 | 0.783842 | 0.781659 | 0.779475 |

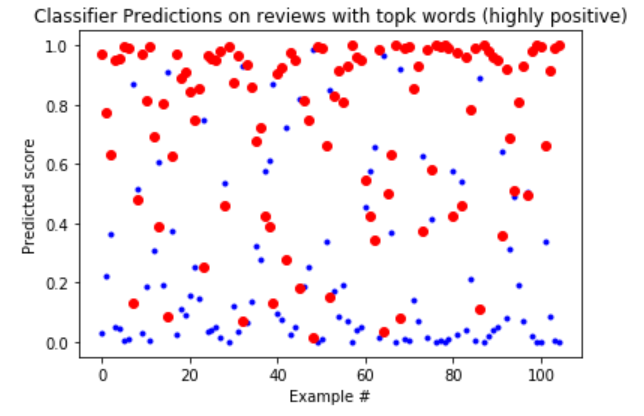
the intuition of selecting stop words in this way is that sentences with these words that has high coefficient

has higher prediction confidence and has better accuracy. Confidence here means the difference between the possibility of predicting positive or negative in the logistic regression model. The higher the difference the higher the confidence and the classifier classify these sentences with ease. By selecting the sentences with words with the highest coefficients and test the accuracy of these sentences, and plot their confidence we see that these data have higher accuracy and confidence. So perhaps if we let sentence have more portion of these words we might increase accuracy.

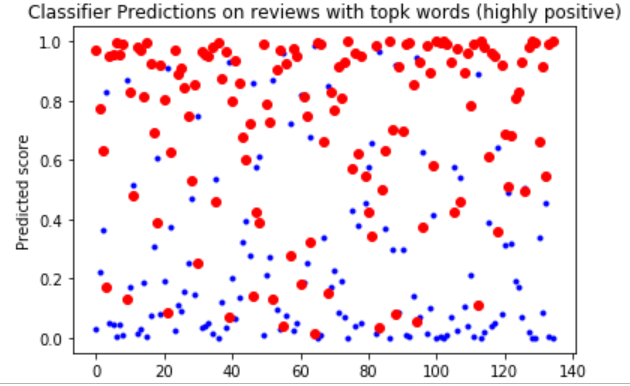
accuracy of sentence with top 5 highest coefficient words: 0.897435



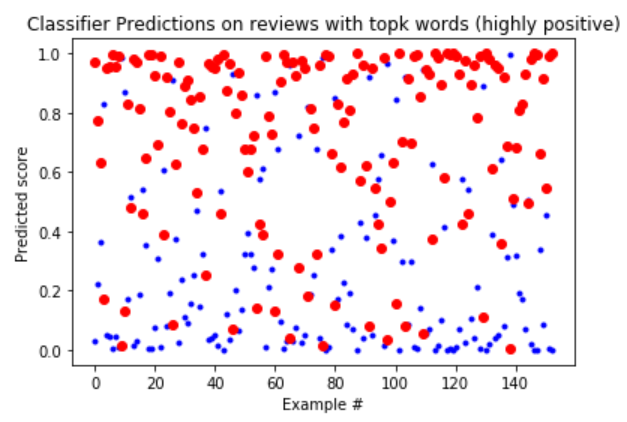
accuracy of sentence with top 10 highest coefficient words: 0.876190



accuracy of sentence with top 15 highest coefficient words: 0.837037



accuracy of sentence with top 20 highest coefficient words: 0.836601



from the result of the stop words selection we can see that it’s only when k is above 3000 when the accuracy on dev set starts to improve, when k is too low too many words are set to stop words for instance when k=500 there are 9882 – 500\*2 = 8882 stop words and it’s not surprising that the accuracy is low in this case, and at high k values the accuracy improves only slightly, for example these are the stop words when k =4750

we can see that these words have little correspondence with negative or positive meaning, so by setting these words to stop words the and increase the accuracy seems reasonable, at k = 3750 the accuracy of the dev set increases the most though still not very much. The hyper parameter stop\_words only turn the selected list of words into stop words which is not the same as selecting features so the improvement has its limits

***TfidfVectorizer:***

as opposed to CountVectorizer which convert the collection of text into a matrix of

Convert a collection of text documents to a matrix of token counts, the TfidVectorizer converts the collection of text documents to a matrix of TF-IDF features (frequency rather than count) or term-frequency times inverse document-frequency. The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus.

by using TfidfTransformer we can transform count matrix produce by CountVectorize into TfidfVectorizer

However, the result didn’t improve.

|  |  |  |
| --- | --- | --- |
|  | CountVectorizer | TfidVectorizer |
| accuracy on dev set | 0.777292 | 0.762008 |

It’s possible that countvectorizer out performed tf-idf in this case, it’s also possible that in this case some common words(words with high frequency) are helpful in distinguishing positive and negative  
It may be that common words (words which will appear in multiple documents) are helpful in distinguishing between classes. Some words like pronouns are very common and would be down weighted in tf-idf, but given equal weight to rare words in countvectorizer.

***max\_df:***

ignores words having high frequency, the vocabs frequency higher than the threshold are ignored at training

the usage of max\_df is to ignore common words that has little effect on the classification

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| max\_df | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| Accuracy on dev | 0.766375 | 0.783842 | 0.788209 | 0.790393 | 0.783842 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| max\_df | 0.6 | 0.7 | 0.8 | 0.9 | 1 default |
| Accuracy on dev | 0.783842 | 0.781659 | 0.777292 | 0.777292 | 0.777292 |

we can see that max\_df = 0.4 has the highest accuracy, when max\_df is too high, not much words are ignored so it’s reasonable that the accuracy doesn’t change, whereas when max\_df is too low, too many words are ignoreds and the accuracy declines

***min\_df:***

ignores terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature, the value of min\_df means the lowest count threshold

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| min\_df | 1 default | 2 | 3 | 4 |  |
| Accuracy on dev | 0.777292 | 0.777292 | 0.764192 | 0.759825 | 0.753275 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| min\_df | 6 | 7 | 8 | 9 | 10 |
| Accuracy on dev | 0.755458 | 0.751091 | 0.753275 | 0.759825 | 0.751091 |

we can see that when min\_df is low at 1,2 the accuracy is the same as the baseline, since only a few words are ignored the results are the same, when min\_df increases we can see that the accuracy is well below the baseline, it’s possible that the words that has great influence on the prediction are ignored since these words doesn’t appear a lot, such as awesome, delicious, amazing, they don’t appear a lot but it’s apparent that if these words appear in a sentence the prediction will be positive.

***ngram\_range:***

The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min\_n <= n <= max\_n will be used

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ngram\_range | (1,1) default | (1,2) | (1,3) | (2,3) |
| accuracy on dev | 0.777292 | 0.783842 | 0.779475 | 0.733624 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ngram\_range | (3,4) | (1,4) | (2,4) | (3,3) |
| accuracy on dev | 0.670305 | 0.775109 | 0.713973 | 0.652838 |

for ngram\_range selection we see that unigram along with bigram has the highest accuracy, for only unigram the accuracy are low since it doesn’t consider context, but when we use unigram along with bigram and trigram the accuracy declines, this might be the result of overfitting, since for classifying a sentence it might be that the appearance of strong positive or negative words is more important than the context in classification for instance in the training set there is this sentence: “*Dr. Greenberg is attentive, caring and listened to all of my concerns. He made me feel comfortable before my laser procedure and calmed my nerves. I recommend him to all”* when our model uses

only unigram and bigram it sees *calmed, calmed my* but when it uses trigram as well it sees *calmed my nerves* the word nerves might miss-guide the model

for supervised learning we eventually select min\_df = 1, max\_df = 0.31, ngram\_range = (1,2), stop\_words with k = 3750, we get accuracy on dev set equals 0.792576, and accuracy on kaggle equals 0.78923

**Semi-supervised: Exploiting Unlabeled Data:**

after training the supervised classifier, we use the trained logistic regression model to predict the unlabeled data, for selection of what data to add to the labeled train data I use the confidence of the prediction. Confidence here is defined as the absolute value of difference between the negative and positive probability, we first sort the unlabeled data according to the confidence from in reverse order and pick the highest batch amount of data as the data to add, batch size is the amount of sentences we want to add to the labeled data,

every time after adding the new data we retrained the model and