CSE 256 Assignment 1 Report
Vignesh Nanda Kumar: A59010704

The problem of text classification is an important problem to target in Natural Language Processing. It is useful in many NLP tasks like sentiment analysis, token classification, opinion mining, topic modelling. In this report, I will be going through sentiment classification. The objective is to classify the sentiment of user reviews into negative and positive reviews using a supervised text classifier.

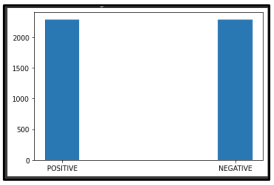


Figure 1: Dataset label distribution

2.1 Guided Feature Engineering

In this problem the basic CountVectorizer is swapped out with a vectorizer that utilizes TFIDF weighing on lexical features and logistic regression is used to build the supervised classifier.

Experimental description and analysis: Hyperparameter tuning was done to find out the best n-gram length "n" and the logistic regression regularization parameter "C". In the first experiment, which uses GridSearchCV from scikit-learn API, the parameters that were tuned were the maximum n-gram range and the regularization parameter C. The grid search is done by taking a combination of all n and C values and k-fold cross validation is performed over the training dataset to find out the best parameter configuration. But the issue with cross validation is that a separate validation set cannot be provided as input. So according to the output of Grid Search the best parameter setting is C = 3, and n = 2 whose performance on the dev data is the same as baseline.

In the second experiment, I ran a for loop over n and C values to build models and find out the best dev accuracy and for getting plots. I plotted the curves for accuracy vs C for different values of n. These curves were plotted for training set and the dev set. From the training set curves, it can be observed that higher C values (near 20) are giving 100% training accuracy which means model could be overfitting. This is expected because increasing C means reducing regularization which can lead to model overfitting. So, from this experiment we get to know about the desired ranges of C. So, a moderate C value < 20 (to ensure some regularization) would lead to the model generalizing well. This could be attributed to the fact that the unregularized model itself gives 98% training accuracy so increasing C would lead to fitting the training set perfectly.

Now coming to the best n-gram ranges, the training accuracy curves are not able to give much information about the best n-gram range values, but the dev accuracy plots is able to give a much clear picture. We also see that as n increases the training data is overfitting quickly on increasing C value. We can see that for n=1 (unigram) we are getting the best performance with a moderate C value < 20. We don't want too high n because we will suffer from sparsity issues due to not enough training data. Even for n = 3 (unigram, bigram and trigrams), the model is performing better than the baseline with good training accuracy and dev accuracy.

n ↓/C→	1	2	4	6	8	10	12	14	16	18	20
1,1	0.902	0.927	0.955	0.968	0.978	0.985	0.990	0.992	0.994	0.994	0.996
1,2	0.957	0.981	0.996	0.998	0.999	1	1	1	1	1	1
1,3	0.980	0.994	0.999	1	1	1	1	1	1	1	1
1,4	0.990	0.998	1	1	1	1	1	1	1	1	1
1,5	0.997	0.999	1	1	1	1	1	1	1	1	1

Table 1: Training accuracy

n ↓/C→	1	2	4	6	8	10	12	14	16	18	20
1,1	0.766	0.766	0.790	0.794	0.788	0.790	0.792	0.792	0.792	0.792	0.792
1,2	0.770	0.781	0.783	0.781	0.783	0.788	0.788	0.786	0.788	0.790	0.790
1,3	0.770	0.779	0.790	0.790	0.790	0.790	0.790	0.790	0.790	0.790	0.790
1,4	0.762	0.768	0.775	0.777	0.779	0.783	0.783	0.786	0.783	0.783	0.783
1,5	0.759	0.770	0.772	0.775	0.775	0.779	0.779	0.779	0.779	0.779	0.779

Table 2: Dev accuracy

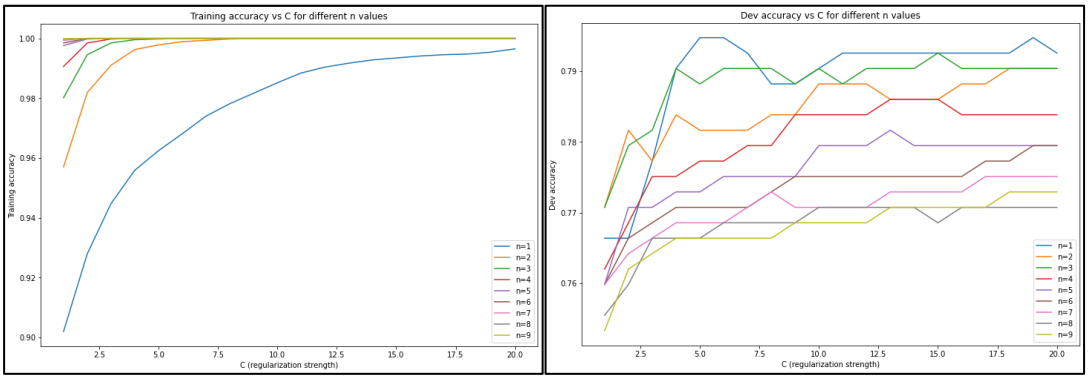


Figure 2: training and dev curves for different n values (accuracy vs C)

We can choose either n = 1 or n = 3 with a moderate C value for regularization. n = 1 can be a good option if there are memory limitations with large data since only unigrams will have to be stored. n = 3 can be a good option for better generalization since the model will be able to learn from unigram, bigram and trigrams of words but this can be memory consuming. We can see n value up to 4 is able to beat the baseline dev data accuracy. We don't want training accuracy to be too high (very close to 100%) because that can mean that the model is overfitting.

Model description	Training set accuracy	Dev set accuracy
Baseline	0.982	0.777
TFIDF vectorizer with n = 1 and classifier regularizer C = 6	0.968	0.794
TFIDF vectorizer with n = 3 and classifier regularizer C = 4	0.999	0.790

Table 3: comparison with baseline model

2.2 Independent Feature Engineering analysis

For this task, I used my intuitions and the training data to add new features and tried different tokenizers which will be described in this section.

Tokenizers and Stop word removal techniques:
For stop word removal, I used the nltk library to download the list of stop words. And for each sentence in the training and validation dataset I iterate over all the words and remove the stop words from the sentence. Now this new sentence is converted to TFIDF vector and used as features for logistic regression model training. The validation accuracy degraded compared to performance with stop words. This can be attributed to the fact that stop words will be contributing towards the model performance.

The other tokenizers I tried out are WordNet lemmatizer and Porter stemmer. Stemming is a more crude process (Stanford reference) that chops off the ends of words in the hope of getting the word's base form correctly. But lemmatization does this more correctly by using a vocabulary (in this case WordNet) and to return the base or dictionary form of a word called as lemma. Using stemming and lemmatization might not necessarily generalize well. Stemming and lemmatization tokenization highly depend on the dataset and the task in hand.

Sentence	This place is an absolute joke! I don\'t even know how this lady "Nina" has a shop. Her work is very poor quality, she is rude and does nothing but
Stop word removal	place absolute joke! Even know lady "nina" shop. Work poor quality, rude nothing
Lemma tokenizer	This place is an absolute joke ! I do n't even know how this lady Nina ha a shop Her work is very poor quality she is rude and doe nothing but
Stem tokenizer	thi place is an absolut joke ! I do n't even know how thi ladi nina ha a shop her work is veri poor 2uality she is rude and doe noth but

Table 4: different tokenization example

Adding new features:
In this section I discuss the different new features that I derived from the existing text in the dataset. The features that I tried and built models on are numerical features (on top of the existing TFIDF vectorizer). I also check the distribution of the feature values with the labels to see if a feature is useful in distinguishing between the two labels. Features like word count, character count, average word length (intuition was that negative reviews

might have more words) were computed from the data provided and their performance was tested on dev data. The distributions of the features vs the labels are also shown below which show that these features are not able to distinguish between the two classes by much. I also tried computing the number of adjectives, number of verbs (again the intuition was that adjectives and verbs can help distinguish between positive and negative reviews) in the sentence using part of speech tagging support provided by the spacy library. These features did not contribute much in improving the accuracy of the model instead degraded the accuracy on dev data (because from the distributions we can see that the features are not able to distinguish between the 2 labels).

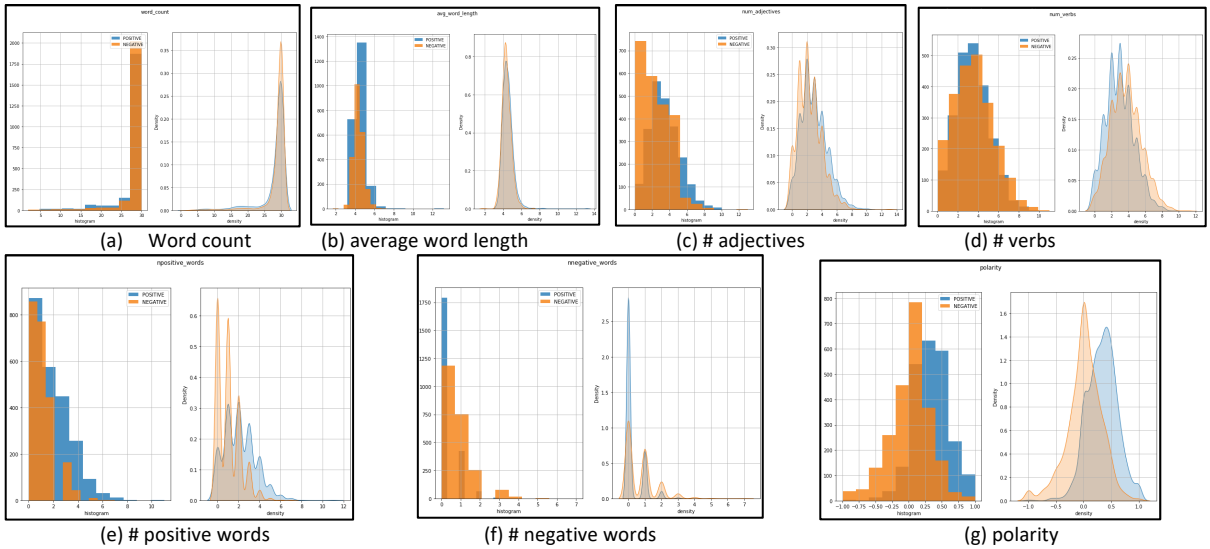
Then I tried to find the number of positive words and number of negative words in each sentence. For this, I used the textblob library to get the sentiment polarity of words and count the feature values accordingly. Finally, I tried using the textblob library to get the sentiment polarity of the sentence itself and used that as a feature. This contributed in improving the performance significantly of the classifier compared to baseline on the dev dataset as shown in Table 6.

Features	Value
Word count	30
Character count	117
Average word length	3.9
Number of adjectives	3
Number of verbs	1
Number of positive words (polarity value > 0.2)	0
Number of negative words (polarity value < -0.2)	2
Sentence Polarity	-0.19

Table 5: Features for sentence: "This place is an absolute joke! I don\'t even know how this lady "Nina" has a shop. Her work is very poor quality, she is rude and does nothing but"

Feature engineering done	Training accuracy	Dev accuracy
None (baseline)	0.982	0.7772
Best Hyperparameter tuning result from 2.1	0.968	0.794
Stop word removal	0.904	0.759
WordNetLemmatizer	0.887	0.759
PorterStemmer	0.896	0.766
Adding word count + TFIDF	0.898	0.753
Adding character count + TFIDF	0.899	0.759
Adding average word length + TFIDF	0.900	0.762
Adding # adjectives + TFIDF	0.895	0.766
Adding # verbs + TFIDF	0.888	0.759
Adding # positive words, # negative words + TFIDF	0.864	0.781
Adding # sentence polarity + TFIDF	0.860	0.788

Table 6: Model performance keeping logistic regression and adding different features



(a) Word count (b) average word length (c) # adjectives (d) # verbs

(e) # positive words (f) # negative words (g) polarity

Figure 3: Distributions of different features between positive and negative classes

To summarise, I observed that using different tokenization schemes is highly dependent on the dataset. Also for adding new features, looking at the feature distribution with the labels is a good interpretation measure to check if a feature is able to help distinguish between different classes. The features like word count, word length, number of adjectives, number of verbs are not able to distinguish between the two classes as seen from the distribution plots. While the sentence polarity, number of positive and negative words plots show different distributions for both classes. So, features like sentence polarity, number of positive and negative words helped improve performance of model without changing the base model.

In conclusion, the assignment gave good insights in targeting a text classification problem through ideas of hyper parameter tuning, feature engineering.

3.1 References

1. https://towardsdatascience.com/text-analysis-feature-engineering-with-nlp-502d6ea9225d
2. https://www.geeksforgeeks.org/python-stemming-words-with-nltk/
3. https://gist.github.com/4OH4/f727af7dfc0e6bb0f26d2ea41d89ee55
4. https://medium.com/mlearning-ai/nlp-tokenization-stemming-lemmatization-and-part-of-speech-tagging-9088ac068768
5. https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a
6. https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp
7. https://stackoverflow.com/questions/25217510/how-to-see-top-n-entries-of-term-document-matrix-after-tfidf-in-scikit-learn
8. https://medium.com/analytics-vidhya/nlp-with-spacy-tutorial-part-2-tokenization-and-sentence-segmentation-352df790a214
9. https://machinelearningknowledge.ai/nltk-tokenizer-tutorial-with-word_tokenize-sent_tokenize-whitespacetokenizer-wordpuncttokenizer/
10. https://machinelearningknowledge.ai/complete-guide-to-spacy-tokenizer-with-examples/
11. https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a
12. https://www.section.io/engineering-education/regularization-to-prevent-overfitting/
13. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
14. https://stackoverflow.com/questions/65359261/can-you-get-all-estimators-from-an-sklearn-grid-search-gridsearchcv
15. https://medium.com/analytics-vidhya/nlp-with-spacy-tutorial-part-2-tokenization-and-sentence-segmentation-352df790a214
16. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
17. https://scikit-learn.org/stable/modules/feature_extraction.html
18. https://scikit-learn.org/stable/modules/grid_search.html
19. https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html#building-a-pipeline
20. https://www.analyticsvidhya.com/blog/2021/04/a-guide-to-feature-engineering-in-nlp/