The Problem Setup

In this experiment, we shall compare several NLP preprocess techniques on the **logistic regression model** for sentiment analysis of movie reviews. Concretely, we are going to examine 7 preprocessing functions, which are

1. Count the occurrence of unigrams
2. Count the occurrence of unigrams with stopwords excluded
3. Count the occurrence of unigrams with stemming
4. Count the occurrence of unigrams with lemmatization
5. Count the occurrence of bigrams
6. Count the occurrence of bigrams with stopwords excluded
7. Count the occurrence of both unigrams and bigrams

The above feature extractor can all be created via APIs provided by **CountVectorizer** from **sklearn** with aids from **stopwords, WordNetLemmatizer, SnowballStemmer** accommodated by **nltk**.

We anticipate the best preprocess function to be the 3^{rd} one because the corpus is about movie reviews, and usually the sentiment of a review can be captured by a few words, such as ”like,” ”bad,” etc. Adding stemming may further concentrate on these important words since usually the meaning of a word is related to its stem.

The Experimental Procedure

In this experiment, corpus is stored in the folder *rt-polaritydata.*

1. At step one, we shall load the corpus from corresponding files. We label the positive movie reviews by appending a column of 1s to its right. Similarly, we concatenate a column of 0s to the right of negative data. After that, we vertically combine the positive and negative data and shuffle them completely. Finally, we split the shuffled set into training/validation set and test set, where the test set shall take up **30%** of the volume.
2. In order to avoid overfits and underfits, we are going to pick the best **”max_features”** parameter of CountVectorizer for each preprocess function. We plot the training and validation accuracy returned from our **5-fold cross validation** function with respect to ”max_features.” More specifically, our cross-validation function runs 5 times where each iteration uses $\frac{4}{5}$ of the input data for training and the rest for validation. We mainly concern about the mean of **accuracy, macro f1 scores, and weighted f1 scores** of the 5 iterations.
3. We run cross-validation on each tuned preprocess function to select the best model ranked by **”macro f1”**.
4. Finally, we run our models with the best preprocess function on test data and to report the generalization error.

Range of Parameter Settings

- **Model Settings:** We have set **max_iter** of **LogisticRegression** to 500 for it to converge.
- **Feature Extractor: CountVectorizer** from **sklearn.feature_extraction.text**. Globally, we have set the **max_features** parameter to 1500 to reduce overfitting. We toggle **”ngram_range”** among $(1, 1), (2, 2), (1, 2)$ to extract unigram, bigram, or both unigram and bigram features.
- **Stop Words Collection: nltk.corpus.stopwords.words(”english”)**
- **Stemming Utilities: ntlk.stem.SnowballStemmer**
- **Lemmatization: ntlk.stem.WordNetLemmatizer**

Results and Conclusions

The unigram feature extractor with stemming performs the best, with a 74% accuracy, 0.74 macro f1 score, and 0.74 weighted f1 score from cross validation, which conforms to our anticipation. The unigram extractor with lemmatization closely follows, with 73% accuracy. In fact, all unigram preprocessors have similar performance and all scores better than bigram feature extractor as well as processors that extract both unigram and bigram features, as shown in *Table.*1. At the end, the unigram feature extractor with stemming performed at 74% accuracy and 0.74 macro f1 score on the test data.

Name	Accuracy	Macro f1	Weighted f1
bigram	0.64	0.64	0.64
bigram_wrt_stopwords	0.59	0.56	0.56
unigram and bigram	0.72	0.72	0.72
unigram	0.73	0.72	0.72
unigram_lemmatization	0.73	0.73	0.73
unigram_stemming	0.74	0.74	0.74
unigram_wrt_stopwords	0.71	0.71	0.71

Table 1: Performance of Various Preprocess Techniques on Validation Set

It could be due to the fact that the dimension of features extracted by bigram preprocessors is too large, and thus require much more data to train, which is the exactly the curse of dimensionality. Also, it is often enough to capture the sentiment of a review by a few signaling words, and due to the limitation on the size of the data, unigram extractors could be a good fit here. The connotation of a word can usually be captured by its stem, which explains why stemming performs the best.

Limitations

First of all, each variation of unigram feature extractor scores so closely that our observation cannot be conclusive. Secondly, it is untrue to say that unigram feature extractor works well on any sentimental analysis. With more training data, bigram extractors may eventually prevails. Also, we cannot conclude whether bigram feature extractor or other preprocessors will perform better on other models, such as SVM or Neural Networks. Furthermore, the corpus we experiment on may be convenient for unigram preprocessors, and not the others. In addition, there could exist other combinations of preprocess that may surpass unigram_with_stemming. For example, we have not yet tried combining stemming with stopwords exclusion.