Namdar Kabolinejad COMP 550, A1
260893536

Problem Setup:
The objective of this study is to examine how different preprocessing strategies improve the classification of sentence-level sentiment analysis using supervised linear regression.
Our data corpus is a collection of movie reviews separated into positive and negative polarities. In the implementation, I will attempt to (in order): 1. Prepare the data corpus 2. Preprocess the data 3. Build features 4. Classify the data 5. Evaluate results

Preparing the data corpus:
Firstly, I assemble the corpus in a way that can be used in the implementation. In the code, after reading the data corpus, I create a randomized data frame where each review has its respective label attached. The data frame contains both positive and negative reviews.

Preprocessing:
In the preprocessing step, I'll try to reduce the complexity of the data in order to help with the classification, but not remove so much information that would have a negative effect.
I have written a function called *preprocess_corpus* that takes in a corpus and 5 flags that determine the preprocessing techniques to use on the data. We have 5 individual preprocessing steps, which can be applied individually or as a combination of steps.

The 5 preprocessing techniques include:

1. Cleaning the data: This step firstly removes all HTML tags and extra newlines. It'll also lowercase all the words and finally, remove extra white space from the reviews.
2. Removing stop words: This step will remove a set of stop words from the data set. I'm using the stop word list is defined in the nltk library, excluding "not", "no".
3. Lemmatization: I'm using the WordNetLemmatizer and pass each word's POS for a more accurate lemmatization. Each review is first tokenized and each token is associated with its POS, then each word-POS tuple is lemmatized. Finally, the lemmatized words are joined together to create the lemmatized review.
4. Stemming: Here we simply use PorterStemmer to stem all the review tokens.
5. Special Char Removal: Here we remove all "special" characters from the data. Special characters are anything but white space, numbers, and letters.

Featured Building:
After the entire dataset is prepared using one or multiple preprocessing steps we will split the data into two sets, training and testing. I will then use fit_transform to build BOW features for training the data set. I will proceed by fitting a LogisticRegression model on the training data and using that to predict the test data set.

Results:
In the experiments, I have tried applying 8 different preprocessing strategies. 5 of the preprocessing strategies were simply applying individual preprocessing steps and the other 3 were using a combination of 3 steps together.
The results for the different data splits follow the same pattern with *Stemming*, *Special Char Removal*, and *Lemmatization* having the best accuracies. Due to a lack of space, I will be discussing the results for only the (90,10) split. I have also run the model on the data directly without being preprocessed to use as a benchmark for comparison.

As seen in Table 1, between the 5 individual preprocessing approaches *Stemming* has the best accuracy followed by *Special Char Removal*.
Stemming has better accuracy than *Lemmatization* when used individually, however, when combined with *Data Cleaning* and *Special Char Removal*, *Lemmatization* performs better.
I expected a better accuracy from *Lemmatization* than *Stemming* (in all combinations) given that *Lemmatization* takes the POS into account and is able to obtain the correct lemma for each word. I believe the drop in accuracy for *Lemmatization* is because by lemmatizing the words we are getting the same lemma for multiple different tokens and essentially grouping multiple words, that may be used in contexts differently, together which may result in incorrect predictions, while in *Stemming* different tokens will still be differentiated.
Stop Word Removal actually ended up reducing the accuracy of the model, this is because the stop word list provided by nltk (even after removing "no" and "not") includes words like "don't", "doesn't", … which may be significant in classifying sentiment polarities.

Data Cleaning doesn't seem to improve the accuracy when used individually, this may be because the reviews in our data are already "clean" and the extra step doesn't change the data that much. However, we could see improvement in the accuracy with more messy data and I believe it should still be used as an extra preprocessing step.

Data Split	Preprocessing Technique	Accuracy
(90,10)	No Pre-processing	74.70
(90,10)	Stop Word Removal	74.13
(90,10)	Data Cleaning	74.70
(90,10)	Lemmatization	74.79
(90,10)	Special Char Removal	75.73
(90,10)	Stemming	76.76
(90,10)	Stop Word Removal + Stemming + Special Char Removal	75.35
(90,10)	Data Cleaning + Stemming + Special Char Removal	75.63
(90,10)	Data Cleaning + Lemmatization + Special Char Removal	77.32

Table 1: Performance Results

In conclusion, preprocessing strategies have a significant impact on the classification process. Depending on the type and style of data each preprocessing step might yield different results, but for sentiment analysis *Stemming, Removal of Special Chrachtars,* and *Lemmatization* have positive effects on the accuracy. A preprocessing step of cleaning the data can be beneficial depending on the nature of the data. The accuracy may also increase when multiple preprocessing steps are applied.

Limitations:
Due to the 1.5 page list, I was not able to dig deeper into the topic and the differences between the strategies. Also, given my introductory level of knowledge in python and the libraries used, the implementation and the results obtained may not be the best.
For some reason that I haven't been able to figure out yet the accuracies change within 1 percent of the values in Table 1 but follow the same pattern.

Recourse:

[1]. The effects of pre-processing strategies in sentiment analysis of online movie reviews. Link
[2]. Sentiment Analysis Using Logistic Regression. Link
[3]. Sentiment analysis with logistic regression in Python with nltk library. Link
[4]. Performing Sentiment Analysis on Movie Reviews. Link
[5]. Analysing Movie Review Using NLP (used in coding part). Link