William Powers A59008589

Text Classification Report

Introduction

We are given a dataset of reviews limited to their first 140 characters. The dataset is split into 3 sections. Two of those sections additionally include labels that signify whether the dataset has a positive or negative sentiment; One of them is used for training a model and one is used for testing the model while tuning hyperparameters. The third section is unlabeled and can be used for further tweaking the model and feature extraction.

We are provided with code to read data from the tsv files compressed in a tarball. Additionally, we have code that extracts feature vectors from each document in the data and builds a feature vector for use on a model. The feature vectors are created by using sklearn's method *CountVectorizer* to map each feature to a term in all the vocabulary used among the documents and provide as the feature the number of times it appears in a particular document. Additionally, there is code to encode the labels as integer values for use in the model. Finally, there is a simple logistic regression model provided and code to train the model using test data and evaluate the model using development data. The model uses mostly default values.

All that code ensembles to train and test a model to predict the sentiment of a review from the first 140 characters. Using an accuracy classification score the following results are obtained from the model.

Model Accuracy on **Training** Dataset	0.9821038847664775
Model Accuracy on **Development** Dataset	0.777292576419214

Guided Feature Engineering

The goal for these next following steps is to improve upon the accuracy of the model. First, we will replace the *CountVectorizer* with the *TfidfVectorizer*, which modifies the previous term frequency to discount words that appear in every document. Next, we will experiment with a couple of parameters on the feature extraction methods and the model in order to improve the accuracy results of the model.

First, we will vary the maximum n-gram length that is used for feature extraction. So instead of just counting terms (unigrams), we will count unigrams and larger n-grams. The accuracy of the model as we vary the maximum size of the n-grams is shown below.

Maximum N-gram	Accuracy on Dev Dataset
1	0.7663755458515283
2	**0.7707423580786026**
3	0.7707423580786026
4	0.7620087336244541

As we see the best accuracy is achieved when the utilized n-grams are limited to size 2 or 3. These models are better than only using unigrams because they contain contextual information about unigram

sequences that provide additional meaning that is not provided by simply isolated unigrams. They are better models than ones containing larger unigrams because those likely include too many features that increase overfitting and include too many n-grams that are not shared between documents. Surprisingly, both models have the same accuracy on the dev set. To break the tie, we will be using a feature extraction that uses only n-grams of sizes 1 and 2. This will be to limit the size of the vocabulary used and thus limit the size of the features used. In doing so we are reducing the complexity of the model and reducing the chance of overfitting our model to the dev set and underperforming on future unseen data.

Next, we will tune the regularization parameter on the logistic regression model to further improve the model's accuracy. The current parameter (the inverse of the regularization strength) is set to the default of 1.0. The chart below shows the accuracy of the model as we tune that parameter.

Inverse of the Regularization Strength	Accuracy on Dev Dataset
.01	0.7510917030567685
.1	0.7510917030567685
.5	0.7685589519650655
1	0.7707423580786026
5	0.7816593886462883
10	0.7882096069868996
50	0.7860262008733624
100	**0.7925764192139738**
500	0.7903930131004366
1000	0.7903930131004366

The accuracy as a function of the inverse of the regularization strength does not seem to be an especially simple function but given the data, there does seem to be a maxima around 100. This is because that parameter likely smooths out the model's function enough to negate the peculiarities that overfit it to the training data and allows it to generalize better. However, it doesn't smooth it out too much as to erase valuable data that does generalize. So, we will be using 100 as the regularization parameter for our model. Thus, the final accuracies of our current model are shown in the following table.

Model Accuracy on **Training** Dataset	1.0
Model Accuracy on **Development** Dataset	0.7925764192139738

Independent Feature Engineering

For our first step in attempting to improve our model further, we will be removing English stop words when preprocessing our document. We want to do this to minimize the complexity of our model and avoid overfitting and we are choosing these words because they are words that are not used to convey sentiment in the English language. After setting that parameter on *TfidfVectorizer* and rerunning our model, the accuracy on our dev set is **0.7707423580786026**. Since this is lower than our previous accuracy, we will not be using this preprocessing technique in our further experimentation.

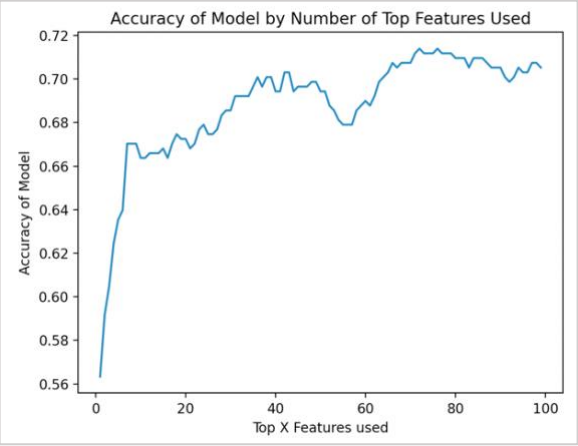
Next, we will change the default whitespace tokenizer and use one that also separates punctuation and other symbols where applicable for a given language. This is so as to make sure words are counted as separate just because one is followed by punctuation and also that all separate segments of meaning are captured, such as expanding contracted words. For this, we will use nltk's method *word_tokenize.* After applying the change, the model's accuracy on the dev set is now **0.8078602620087336**. Thus, we will continue using this tokenizer for our model.

Next, we will convert all the alphabetical letters to lowercase in order to not be case sensitive in our feature engineering of vocab. After doing so, we maintain the same accuracy of **0.8078602620087336**. So likely the text was already converted as such or reviews did not capitalize much or the capitalized words were of little effect. Alternatively, there could be another point in the vectorizer that converts strings to lowercase. Since there was no effect, we shall arbitrarily be keeping this change in our feature engineering code as it is good practice.

Now, we will lemmatize the words to reduce our vocab by combining ones with close meanings that just regard the word's lemma. Hopefully, this will reduce the complexity of the model without losing necessary nuance in the data. For this, we will use nltk's function *WordNetLemmatizer*. After doing so the accuracy for our model on the dev set is **0.8144104803493449**. So, we will continue using this lemmatization process for our feature extraction.

Now we will attempt to remove tokens/features that are special characters using python's *isalnum* function in an attempt to remove unpredictive features. After doing so, we have a dev set accuracy of **0.7969432314410481**. We will not be using this change as it reduces the accuracy, likely because there is some sentiment involved in symbols used for emphasis and use of some can give insight into the temperament and personality of the user, which could predict sentiment in their reviews.

Now we will attempt feature selection by trying to take the most decisive features of the model by looking at their coefficients once trained on the model. On the right, you will see a chart showing the accuracy of the model on the dev set when we only use a certain number of top features. The max accuracy here was **0.7139737991266376** when the top 72 decisive features were used. Since that does not improve our model, we will not use this technique. We attempted this up to 1000 features with similarly disappointing results. There is likely a very large number that could work better than all features. However, such experimentation will not be pursued for the sake of time.



Conclusion

After attempting all of the improvements listed above, there are potentially many more we could attempt although time and maximum page length do not permit us to continue in our experimentations. So, in conclusion the accuracies of the final model on the data sets are shown below.

Model Accuracy on **Training** Dataset	1.0
Model Accuracy on **Development** Dataset	0.8144104803493449