Jacob Fuehne

# LING 406 Term Project: Sentiment Classifier

## 1 Introduction

Sentiment analysis is essential for allowing us to automatically identify positive and negative expressions in language, just as our brains do. Knowing what sentiment we are expressing and knowing the sentiment that others are expressing is a key part of communication that affects how we interact with each other. As such an important part of communication, one obvious use case is in automating dialogue systems and customer service. Sentiment analysis can be used to automate aspects of customer service and allow companies to get customer feedback at a scale not possible otherwise. In the world of Big Data and ever advancing algorithms in natural language processing, sentiment analysis is becoming more important and it is getting better as research progresses.

In this project, I study the performance of different feature sets and machine learning classifiers for the purpose of sentiment analysis. As a baseline, I implemented a bag of words and a Naive Bayes unigram model. In testing features, I've implemented five textual features, non-alphabetical filtering, stopword removal, stemming, POS tagging, unique words, and negation. For testing classifiers, I implemented the SciKitLearn models for Logistic Regression, Decision Tree, and Support Vector Machine. For this project, I use the Cornell annotated movie review dataset, consisting of 1000 positive reviews and 1000 negative reviews, found at http://www.cs.cornell.edu/people/pabo/movie-review-data/. Using movie reviews as a dataset, I split the labelled reviews into a positive set and a negative set, and after shuffling the reviews, I split those into training and test data, with a 80/20 ratio, I feed them into my classifiers.

## 2 Problem definition:

Sentiment analysis or opinion mining is the process of trying to determine emotion or polarity (positive vs negativity) from text. Two possible viewpoints for performing sentiment analysis exist, the computational linguistics perspective and the machine learning perspective. In machine learning, a more statistics and mathematical approach is used to classify text. In computational linguistics, you would focus on leveraging the knowledge of how languages work and finding patterns and features to use to improve performance that way.

The input to a sentiment analysis model is a labelled or annotated dataset of text. In Computational Linguistics, this is called a corpus, or corpora. Corpora are used in Computational Linguistics so that models are able to view training examples and, using the model structure, identify patterns to predict output on a test dataset. The output to sentiment analysis models will be either positive or negative if you are evaluating polarity, or it may be expressions relating to opinions, evaluations, feelings, etc. if you are evaluating subjectivity. For the purposes of this paper, my models will be taking the movie reviews corpus as an input, and outputting the predicted polarity on the test set.

## 3 Previous work:

In handling this project, I referred to the resources as suggested in our rubric. As all of my methods build on top of the bag-of-words model, Kuat Yessenov and Sasa Misailovic's paper "Sentiment Analysis of Movie Review Comments" was very useful. In their paper, they give a description of how a bag of words model works, they describe syntactic and semantic properties, the importance of handling negation, restricting to adjectives and adverbs, and more. In their paper, they point out the limitations of a unigram model by its "inability to capture the subjectivity/polarity relations between words, distinguishing between parts of speech, inability to handle the negation, and different meanings of one word." (Yessenov & Misailovic 2009). They also point out that while a logical extension of the unigram model would be to instead use a bigram, so that a relation between pairs of words could be established, they did not evaluate this model for two reasons. Firstly, the sparsity of the movie review corpus might mean that the bigram may not perform well. And secondly, they found that bigrams do not show an advantage over unigrams in sentiment analysis. From this paper, I incorporated the features of negation, restricting to adjectives and adverbs, and using a unigram model.

In the paper, "NRC-Canada-2014: Detecting Aspects and Sentiment in Customer Reviews" by Svetlana Kiritchenko, Xiaodan Zhu, Colin Cherry, and Saif M. Mohammad, they describe their submissions to the SemEval-2014 Task 4, which sought to detect aspect categories, detect sentiment towards aspect categories, detect aspect terms, and detect sentiment towards aspect terms in the laptop and restaurant domains. In their subtask 4 submission, which was to detect the sentiment towards a given aspect category given a sentence, they used an SVM classifier. In conjunction with their SVM classifier, they reaffirmed the use of POS tags, among other features.

## 4 Approach:

In my approach, I used the movie reviews corpus from Cornell for training and testing. The corpus consists of 1000 positive movie reviews and 1000 negative movie reviews. For each of my trials, I would read in the algorithms from separate directories, and then I would shuffle the positive reviews amongst themselves, and the negative reviews amongst themselves. Then I would split each of the datasets into a ratio of 80/20, with 80% of the data for each polarity being used for training, and 20% for testing. After they were split, I would concatenate the datasets together into a training set of 1600 reviews and a testing set of 400 reviews. Because the reviews were shuffled initially, the process of which data is in the training and testing sets is random, while at the same time, it would be guaranteed that in both the training and test sets, the first 50% of the reviews would be positive, and the last 50% would be negative.

For the features that I tested, I incorporated them into a tokenize method, which takes the string of text uploaded from the review file and splits the lines by space into the individual words which are returned as string elements in an array of lines in an array of reviews. Using global boolean variables that I used to mark which set of features was being tested, my tokenize method had a series of IF statements where it would apply a filtering or tagging of certain features during the tokenization process. The order that the IF statements are applied is the same as the order in which I tested my feature.  The feature Alpha, will filter out any word that is not alphabetical using the built-in Python function isalpha().  The feature Stop will filter out all stop words given with the Python library nltk.corpus, without removing “not”, as this would be necessary for the Negation feature to be applied. While the Stop feature could have been included after the negation, thus allowing “not” to be removed while keeping other elements, this would also mean that any word marked for negation that would have been normally removed by the Stop feature would no longer be recognized, thus not allowing negation to be evaluated fairly. As a measure of thoroughness in evaluating Stop at its full proposed potential, when I tested the Stop feature in my leave one out testing, I branched the trials into the Stop feature without “not” and the Stop feature with “not”. For all following trials that include Stop, the version without “not” is used (ie, “not” will not be filtered out). The next feature I tested was Stemming, which will stem the tokens via the PorterStemmer from nltk.stem.porter, thus chopping off the end of words. The

POS feature uses the part of speech tagger from NLTK to mark the tags of all of the words in the line, then filters the line to only leave words that have been tagged as some form of an adjective or adverb. The negation feature will mark all tokens following a negating word ("not, never, etc.") so that they will be trained as separate tokens from the base form.

Outside of the tokenizer, I implemented two additional optimizations unique to the bayes unigram model, and the scikit learn classifiers respectively. For the Naive Bayes unigram model, I added a prior value on top of all other features tested, thus allowing the unigram model to roughly predict the distribution. While commonly used in Naive Bayes algorithms, it was left out until the end because I wanted my testing to be more reflective of the linguistic features expected in this project than of my ability to optimise a variable. The prior value used was determined previously through trial and error on a previous sentiment analysis unigram model that I created. While not optimized perfectly, it is added simply as an optional optimization on top of the many others. The other optimization is for the Logistic Regression, Decision Tree, and Support Vector Machine models. For these, I implemented an optional optimization called Unique, which seeks to trim down the positive and negative word bags so that no words from the negative bag will appear in the positive bag and vice versa. Inherent to the structures of the models, these features are not implemented on the other group. Naive Bayes unigram requires that words appear in both dictionaries so that it can compare the probability of being in one versus another, while the prior is simply part of the Naive Bayes algorithm.

In testing the performance of my models and features, I implemented a bag of words and a Naive Bayes unigram model as a baseline, with Logistic Regression, Decision Tree, and Support Vector Machine models implemented to compare against. For the Naive Bayes unigram model, I implement laplace smoothing, or in this case, add one smoothing, as I used 1 as the smoothing parameter. I use math.log when adding together so that the probabilities can be added, rather than multiplied, thus removing issues with extremely small numbers being rounded down. For the Logistic Regression, Decision Tree, and Support Vector Machine models, I used the CountVectorizer from sklearn.feature_extraction.text to fit transform the training data and test data for training. Using the same tokenize method as the NaiveBayes unigram, the CountVectorizer makes sure that the data from the file reviews is compatible with the format that sklearn requires. Also necessary in using these models that was not in the NaiveBayes, the positive and negative word bags must be trimmed to be of the same size, and thus, I have a method trimBags, which is used to reduce the bags to be the same size (the max length is determined by the size of the smaller bag).

## 5 Results:

| Features | NAIVE BAYES | LOGISTIC REGRESSION | SUPPORT VECTOR MACHINE | DECISION TREE |
|---|---|---|---|---|
| BoW (Baseline) | 0.813 | 0.8355 | 0.8265 | **0.63975** |
| BoW+Alpha | 0.8145 | 0.8355 | 0.82575 | 0.625 |
| BoW+Alpha+Stop+Not | **0.81975** | 0.837 | **0.83075** | 0.633 |
| BoW+Alpha+Stop | 0.8125 | 0.8315 | 0.8245 | 0.63475 |
| BoW+Alpha+Stop+Stemming | 0.7975 | **0.83825** | 0.82175 | 0.63225 |
| BoW+Alpha+Stop+Stemming+POS | 0.764 | 0.76 | 0.735 | 0.63625 |
| BoW+Alpha+Stop+Stemming+POS+Negation | 0.771875 | 0.736875 | 0.7175 | 0.6075 |
| BoW+Alpha+Stop+Stemming+POS+Negation+Unique | N/A | 0.7385 | 0.713 | 0.61575 |

Fig1. Performance of features on models measured in accuracy

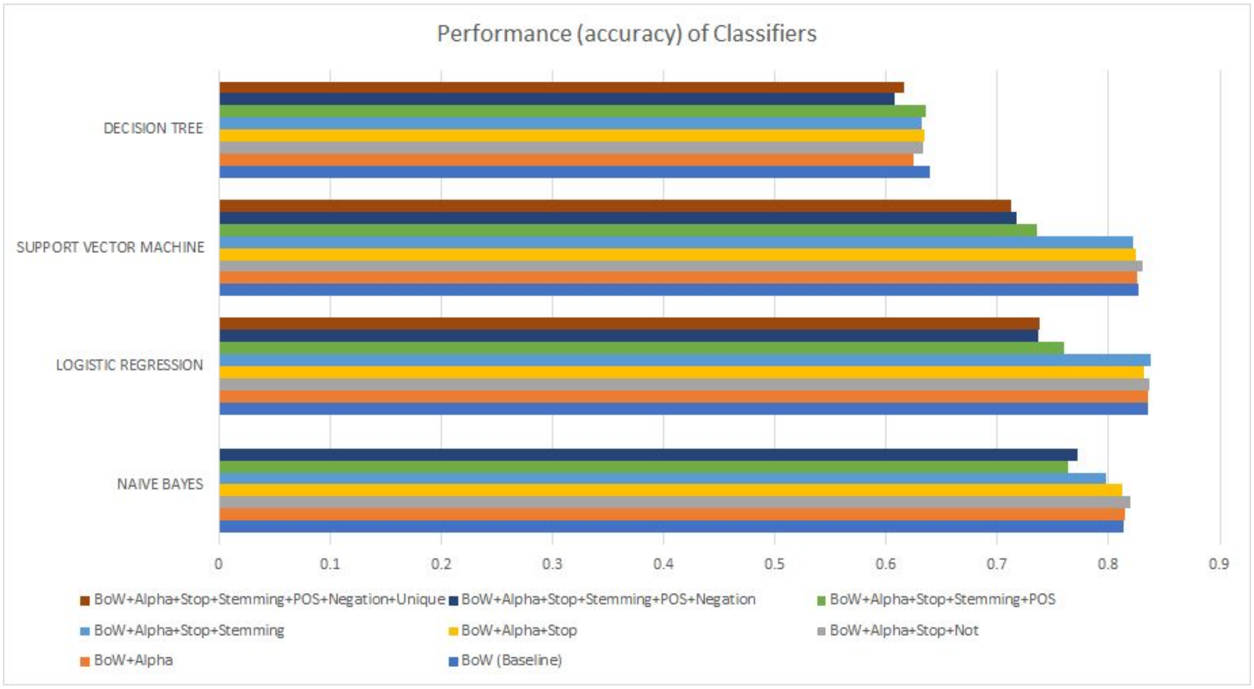
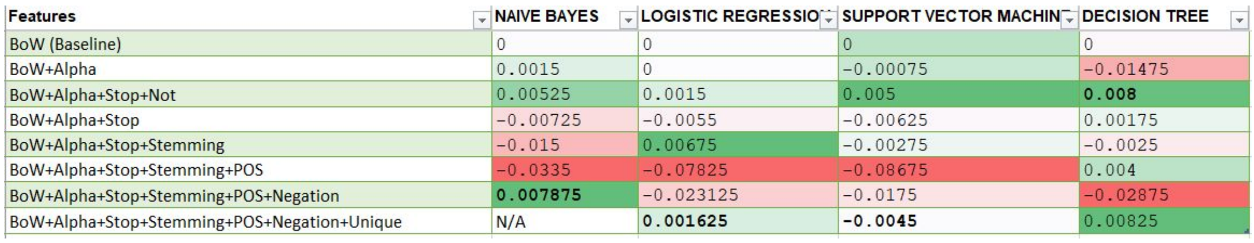


Fig2. Clustered bar graph showing performance of features on models measured in accuracy



| Features | NAIVE BAYES | LOGISTIC REGRESSIO | SUPPORT VECTOR MACHIN | DECISION TREE |
|---|---|---|---|---|
| BoW (Baseline) | 0 | 0 | 0 | 0 |
| BoW+Alpha | 0.0015 | 0 | -0.00075 | -0.01475 |
| BoW+Alpha+Stop+Not | 0.00525 | 0.0015 | 0.005 | **0.008** |
| BoW+Alpha+Stop | -0.00725 | -0.0055 | -0.00625 | 0.00175 |
| BoW+Alpha+Stop+Stemming | -0.015 | 0.00675 | -0.00275 | -0.0025 |
| BoW+Alpha+Stop+Stemming+POS | -0.0335 | -0.07825 | -0.08675 | 0.004 |
| BoW+Alpha+Stop+Stemming+POS+Negation | **0.007875** | -0.023125 | -0.0175 | -0.02875 |
| BoW+Alpha+Stop+Stemming+POS+Negation+Unique | N/A | **0.001625** | **-0.0045** | 0.00825 |

Fig3. Change of Performance (accuracy) as features are added to models

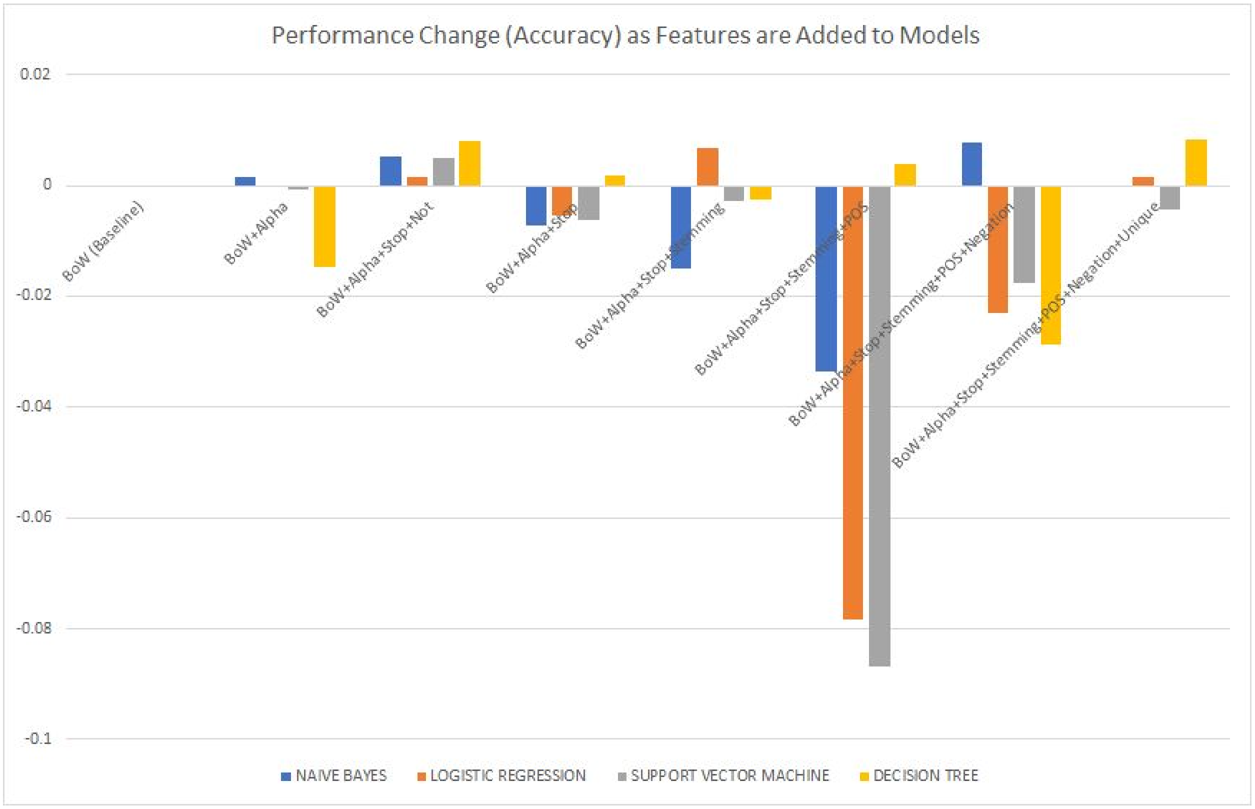
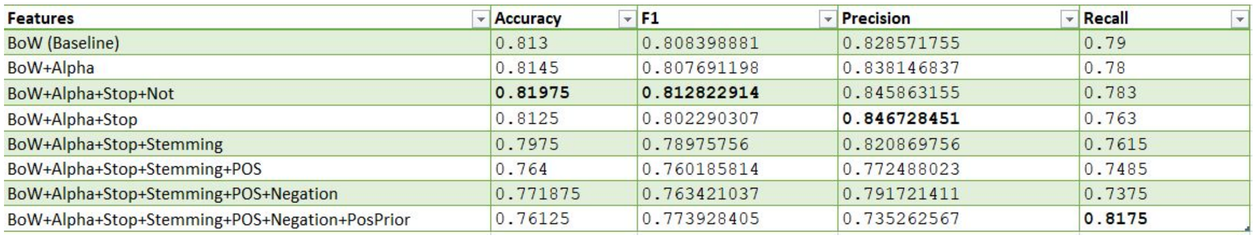


Fig4. Change of Performance (accuracy)  as features are added to models visualized



| Features | Accuracy | F1 | Precision | Recall |
|---|---|---|---|---|
| BoW (Baseline) | 0.813 | 0.808398881 | 0.828571755 | 0.79 |
| BoW+Alpha | 0.8145 | 0.807691198 | 0.838146837 | 0.78 |
| BoW+Alpha+Stop+Not | **0.81975** | **0.812822914** | 0.845863155 | 0.783 |
| BoW+Alpha+Stop | 0.8125 | 0.802290307 | **0.846728451** | 0.763 |
| BoW+Alpha+Stop+Stemming | 0.7975 | 0.78975756 | 0.820869756 | 0.7615 |
| BoW+Alpha+Stop+Stemming+POS | 0.764 | 0.760185814 | 0.772488023 | 0.7485 |
| BoW+Alpha+Stop+Stemming+POS+Negation | 0.771875 | 0.763421037 | 0.791721411 | 0.7375 |
| BoW+Alpha+Stop+Stemming+POS+Negation+PosPrior | 0.76125 | 0.773928405 | 0.735262567 | **0.8175** |

Fig4. Performance of features on Naive bayes unigram baseline measured in accuracy, f1, precision and recall

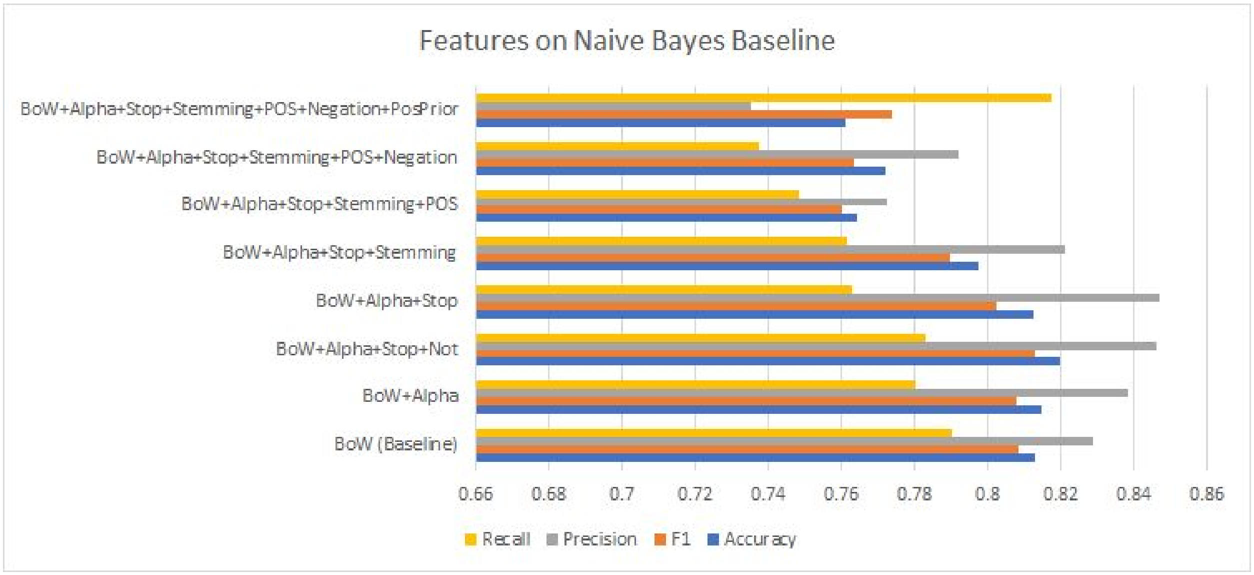
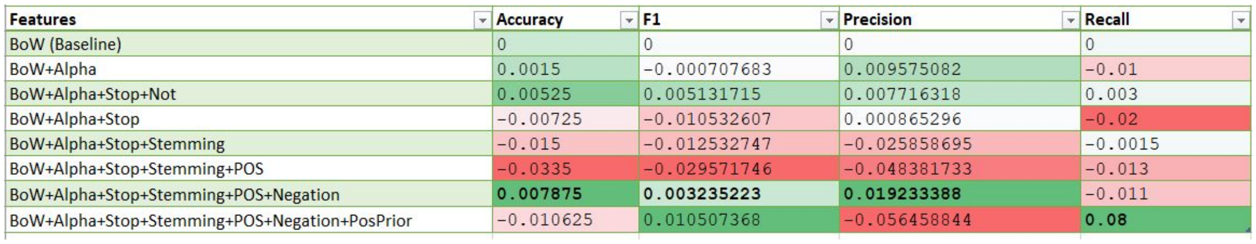


Fig 5. Clustered bar graph of performance of features on Naive bayes baseline measured in accuracy, f1, precision and recall



| Features | Accuracy | F1 | Precision | Recall |
|---|---|---|---|---|
| BoW (Baseline) | 0 | 0 | 0 | 0 |
| BoW+Alpha | 0.0015 | -0.000707683 | 0.009575082 | -0.01 |
| BoW+Alpha+Stop+Not | 0.00525 | 0.005131715 | 0.007716318 | 0.003 |
| BoW+Alpha+Stop | -0.00725 | -0.010532607 | 0.000865296 | -0.02 |
| BoW+Alpha+Stop+Stemming | -0.015 | -0.012532747 | -0.025858695 | -0.0015 |
| BoW+Alpha+Stop+Stemming+POS | -0.0335 | -0.029571746 | -0.048381733 | -0.013 |
| BoW+Alpha+Stop+Stemming+POS+Negation | **0.007875** | **0.003235223** | **0.019233388** | -0.011 |
| BoW+Alpha+Stop+Stemming+POS+Negation+PosPrior | -0.010625 | 0.010507368 | -0.056458844 | **0.08** |

Fig 6. Change in accuracy, f1, precision, and recall as features are added for Naive bayes baseline

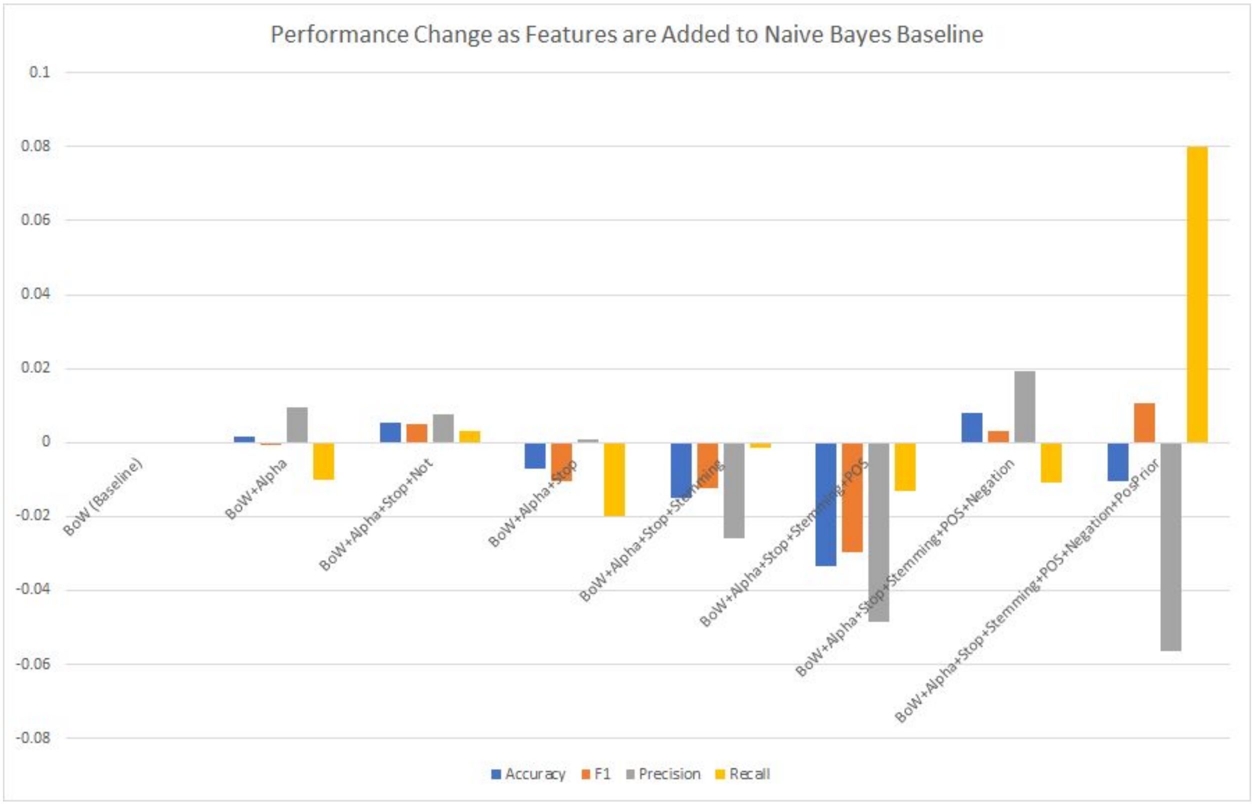


Fig7. Change in accuracy, f1, precision, and recall as features are added for Naive bayes baseline visualized

In my project, I developed the code to display the accuracy, f1, precision, and recall of all of the classifiers, and I also ensured that I would be able to run multiple trials in one run, with each trial having randomized data, and I would display the average and standard deviations of accuracy, f1, precision, and recall in each trial. This feature worked successfully, and it remains commented out in the code, available for use if needed. However, I found that due to the number/dimensionality of all the variables, there was no practical way that I could think of to visualize all of the data at the same time. Thus, I split my results into two groups, one that focuses entirely on the average accuracy of the classifiers as features are added one at a time (leave one out approach), and another one that focuses entirely on the Naive Bayes unigram baseline as features are added, mapping out the average accuracy, f1, precision, and recall. Each trial is run 10 times and each data entry shown in the tables is the result of the average of 10 runs for that feature set on that model. 10 trials are chosen because the variability in accuracy among my randomly selected training and testing datasets often ranges by a noticeably larger margin than the addition of text features do. A number of trials larger than 10 would be beneficial, however, 10 trials was the most that I could manage, as some of the larger feature sets would go on for long periods of time, only to later cause Jupyter Notebook to crash.

Because some changes were small, some were large, and some were hard to see clearly without doing arithmetic, I decided to produce additional tables using Microsoft Excel that color code the gains and losses of performance as each feature is added, thus allowing me to more easily draw conclusions one which features performed best, and for them to be easier to see to the viewer.

In my results, I was surprised to see such variations among the models in terms of performance as different features were applied. The best performing overall was the Logistic Regression classifier when using Alpha, Stop, and Stemming features as seen in figures . While typically, features that improved the performance of one classifier improved the performance of all of them and the same for negative changes, this was not the case universally. As with stemming, which reduced the performance of the Naive Bayes unigram model, but led to the best performance overall when added to the Logistic Regression classifier.

In my post-testing research into explaining why this particular combination may have worked so well, I came across a published paper on sentiment analysis called "GBSVM: Sentiment Classification from Unstructured Reviews Using Ensemble Classifier" by Madiha Khalid, Imran Ashraf , Arif Mehmood, Saleem Ullah, Maqsood Ahmad, and Gyu Sang Choi published in April 2020. While their paper obviously uses more in depth features for their publication, my intuition on the ordering of my IF statements had coincidentally lined up nearly exactly with the preprocessing that this paper uses. Their ordering was to use tokenization to split up user reviews into tokens, remove punctuation, remove stop words, remove numbers and non-alphabetical characters, and to lastly use stemming to convert words back to the root word. In their paper, they even used the scikit learn implementations (although they had used an ensemble model approach, where they combined the predictions of multiple models to reach a final conclusion). In the paper, they describe Logistic Regression as a predictive analysis model that is "used to describe data and explain the relationship between a dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables." Considering this definition and the effect that the Unique feature that I added also had on logistic regression, I would conclude that Logistic Regression benefited greatly from Stemming because it seems to benefit the most from tokens with clearly defined relationships. By condensing the stems of words into the base pair, and removing ambiguity between positive and negative polarities, I think that Logistic Regression is able to make more accurate predictions.

As for other results of note in my project, I found that the addition of a prior probability value to the Naive Bayes unigram lowered accuracy by 1%, as seen in Fig 4 through 7. However, it also improved the F1 score by a little over 1%, as well as improving the recall by the largest margin of any feature measured at 8% improvement. In order to explain this phenomena, I researched possible explanations, as I had done with Logistic Regression. In my research, I found a published paper called “Techniques for Improving the Performance of Naive Bayes for Text Classification” by Karl-Michael Schneider.

In this Schneider’s paper, he states that for larger documents, classification scores are predominantly affected by word probabilities, and prior probabilities hardly have an effect. However, the inverse is also true in cases where documents are usually very short and the distribution is skewed. He further notes that ignoring prior probabilities in such cases will actually improve the accuracy (Schneider). Due to the fact that I had only tested the prior probability as an addition at the very end, after all other features had filtered out much of the training data, I think it is reasonable to say that the remaining training data was sparse. After the part of speech tagger is run, the document will only contain adjectives and adverbs, which while this means that nothing is left to distract the classifiers in deriving the meaning, this greatly reduces the data set size and it also certainly leads to semantic meaning being lost in removing all other parts of speech.

Initially, during my implementation and early testing of the program, I had my doubts about the effectiveness of removing all words except for adjectives and adverbs. Many sentences were reduced to simple phrases. And so, to explain the failures of the part of speech tagger in my results, I researched other implementations of part of speech tagging that could explain this. In my post-testing research, I found a publication on weighted part of speech tagging in sentiment analysis called “Word clustering based on POS feature for efficient twitter sentiment analysis” by Yili Wang, KyungTae Kim, ByungJun Lee, and Hee Yong Youn published in 2018.

The results of this paper help explain my issue, while also validating the previous studies that I had referred to. In their model, they propose using the POS tagger for weighted subclasses to put words into. Adjectives, adverbs, and verbs are included in a subclass called emotion, and they are given the most weight, as they directly pertain to the emotional description of a sentence. The second class, norm, is considered in the weights as important, but much less than the emotion subclass. Words evaluated into the norm subclass include nouns and the variations of noun tags. The third class is simply called remaining, and the weight of this class fluctuates. One other thing of note is that the weights and formulas in the paper are designed to be variable. Without diving too deep into the math, there is a variable in the paper that is designed to accommodate the dilemma of weighting the emotion subclass words, while still maintaining a large enough training size.  By default, the subclass of words marked as remaining will be much larger than those marked emotion, but if the dimensionality of the emotion subclass is deemed large enough, then (roughly speaking), the formula will put more weight into the emotion subclass (Wang et al. 2018).  This formula provides a solution to the issue that I encountered.  If your dataset is too small, or in my case, if you have filtered out too many words on an already small dataset, you can leverage the benefits of part of speech tagging without needing to sacrifice your remaining training data.

## 6 Discussion and Conclusions:

In this project, I learned a lot about the different models of machine learning classifiers and how text features impact them differently. I've learned that more features are not necessarily better, and that just because you can find research that supports a certain feature as useful, does not mean that it will be useful for your model or for your use-case. Despite finding two separate papers concluding that adjectives and adverbs would increase performance for sentiment analysis, I found that in my case, it did not. In general, while researching the effects of different features and learning through personal experience, I found that the best method for adding features to your project is to test how it performs with your model, your corpus/test data/domain, and how it performs in real use cases. I think some possibilities for improvement could be to test more models, and more combinations of features for those models. In my research of the issues encountered during my test results, I found the idea of ensemble methods that was proposed by Madiha Khalid, et al., to be a very appealing approach.  In order to construct the sentiment analysis that humans have, our brains, in some sense, have their own ensemble method of taking in the varying parts of context in the world through our 5 senses. I am certainly of the opinion that the best way to achieve better computational linguistic models is to mimic the structure and mechanisms of the brain as best we can, and on all levels, from neurons to more abstract levels such as psychology, sociology, linguistics, etc.  For starters though, the methods in my project could implement better methods to preserve training data, as I believe that stacking feature filters on top of each other was a major source of issues which caused the small movie reviews corpus to continually shrink.  Instead of implementing filters to remove text entirely, the better solution would be to follow the general direction of the several papers that I cited in my post-test analysis and to find ways to weight the emotion subclass more heavily, and preserve remaining words until you've determined through some means, such as a formula, that you can afford to sacrifice less text that is less relevant to the overall sentiment.

## 7 Citations

Yessenov, Kuat, and Sasa Misailovic. “Sentiment Analysis of Movie Review Comments” Massachusetts Institute of Technology, Spring 2009. http://people.csail.mit.edu/kuat/courses/6.863/report.pdf

Svetlana Kiritchenko, Xiaodan Zhu, Colin Cherry, and Saif M. Mohammad. NRC-Canada-2014: Detecting Aspects and Sentimentin Customer Reviews. Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014), pages 437–442,Dublin, Ireland, August 23-24, 2014. https://www.aclweb.org/anthology/S14-2076.pdf

Khalid, Madiha et al. “GBSVM: Sentiment Classification from Unstructured Reviews Using Ensemble Classifier.” *Applied Sciences* 10.8 (2020): 2788. *Crossref*. Web. https://www.mdpi.com/2076-3417/10/8/2788/pdf

Schneider, Karl-Michael. “Techniques for Improving the Performance of Naive Bayes for Text Classification.” *Computational Linguistics and Intelligent Text Processing Lecture Notes in Computer Science*, 2005, pp. 682–693., doi:10.1007/978-3-540-30586-6_76. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.2085&rep=rep1&type=pdf

Wang, Y., Kim, K., Lee, B. *et al.* Word clustering based on POS feature for efficient twitter sentiment analysis. *Hum. Cent. Comput. Inf. Sci.* 8, 17 (2018). https://doi.org/10.1186/s13673-018-0140-y