Jeanette Pranin (jrp338), Nishant Subramani (nso155), Jaiveer Kothari (jvk383)

	Bayes.py	**BayesBest.py**
Avg Pos Precision	0.9439	0.9266
Avg Neg Precision	0.7392	0.6888
Avg Pos Recall	0.9321	0.9209
Avg Neg Recall	0.7745	0.7031
Avg Pos F1-Measure	0.9379	0.9237
Avg Neg F1-Measure	0.7558	0.6953

Our Naive Bayes unigram model (in bayes.py) makes the assumption that given the class of document (5 star or 1 star), that word i's presence is independent of word i+1's or i-1's presence. This assumption is good but limited such that words occur in groups and in some sequential ordering. Furthermore, certain words may not be independent of each other given the class. We utilized add-one smoothing for this model.

In bayesbest.py, we chose to utilize orderings and groupings of words by utilizing bigrams. We stripped each string into two word intervals and used these bigram tokens as the features with which to classify from. Furthermore, we smoothed differently. Since bigrams are infrequent and many will never be seen, but those that are seen should be weighted much higher, we chose to do 0.001 smoothing. We gave a pseudo presence count of 0.001 rather than 1 for unseen bigram tokens. However, these additions reduced our performance from 0.9379 to 0.9237 for average positive f1-measure, and 0.7558 to 0.6953 for average negative f1-measure.

Options for improvement include utilizing trigrams or quad-grams. However, Google has published research that show diminishing returns by increasing from bi to tri to quad to pent. Using various features beyond just presence such as types of words (nouns, adjectives, verbs), considering length of document, creating a stop list of words such as "a" or "the," and using map estimates are also good features to integrate.