Classifier.py

Noah Brackenbury and Alex Mathson
CS 322, Fall 2015

To train our classifying program, we tokenized our training set by using a tokenizer adapted from our spellchecker. This tokenizer separates all words (at spaces) and strips them of all punctuation using regular expression substitution. This process works well in combination with the system dictionary that we use, because that dictionary includes many contractions and other words without their punctuation. Finally, after observing the tokenized output a few times, we decided to replace dashes with a space instead of removing them. Before, text like "so-but" would become "sobut". Now, replacing the dash with a space properly separates the words, "so but".

We incorporated the system dictionary so that we can replace training set words not in the dictionary with the <unk> tag. This stands for an unknown word, and improves the accuracy of the classifier. These training sets feature many proper nouns and other specific words that we don't want to be confused with an author's style. For example, if we tested a sentence with the word "Darcy" in it, it would not immediately be assigned to Jane Austen due to her use of the name in Pride and Prejudice (her training set text). Lastly, as checking every word the program encounters against a list (the dictionary) of 200,000+ words is extremely slow, we converted the list into a set. Sets in python are made using hash tables, and so lookup time is O(1) as opposed to O(n). This sped up our program immensely.

Again for purposes of memory and speed, we used bigram probability. This greatly improves accuracy from unigrams, and does not require nearly as much time and memory as trigrams. We also implemented Good-Turing smoothing on any bigram with an occurrence count of less than five. Due to the running time of this program, however, we were unable to run a full test and produce a measurement of accuracy.