Authorship Classifier Report

Eric Walker

Data Cleaning

In cleaning data for testing purposes, a very general approach was used. First, all new line characters were replaced with a space to avoid differences between files that have new line characters between paragraphs and those that don't. Next, multiple periods were replaced with a single one, so that no sentences with '…' in them would appear as three sentences. Then, sentence breaks are denoted with the character '@.' Then, all characters that were not letters or numbers were stripped, with the exception of apostrophes directly between two letters. This cleaning separated words and sentences not necessarily the most accurately, but well enough to work in the model. Also, it's very, very fast.

N-gram

In choosing a particular n-gram, the program was coded to allow a general n, which would be set in the beginning of the main. Then, a series of tests were run using dev mode to see which n produced the best results. For n=2, an average of 48% were correct, 13% for n=3 and 5% for n=4. From this, I decided to use n=2, as it was the best at predicting a certain language (though it's really bad at detecting Christie's particular style). However, I wasn't quite convinced with dev mode. So, I ran on test mode, with a different title from each author for n=2 to n=4, and it became clear that this was much more accurate (5/7 correct for n=2, whereas 1/7 correct for n=4). My thoughts on why it gets so much worse at large n is due to the lack of backoff. This means that most 5- or more-grams will not be anywhere in the training set, so it would be best to look at smaller n-grams, but without backoff, GT smoothing comes in and messes up the calculations, making everything appear very unlikely.

Backoff

I tried using backoff to allow higher n-grams, which would theoretically be more accurate. However, this ran into some nasty coding problems that resulted in the author with the smallest vocabulary having the highest probability. Given more time, I would have tried to incorporate this into the program.

Smoothing and Speed

For smoothing our counts, I used Good Turing Smoothing for n-grams with n>1. For unigrams, I used a fixed dictionary and an unknown token as the go-to for unknown words. GT smoothing was only applied to bigrams and unigrams that had count less than 6, because above that it doesn't really change much. Incorporating the dictionary at first was incredibly slow, as for every word found, the program would loop through a 400,000 word dictionary. However, the dictionary was stored in a python dict that had values of the first character of the word, and values of a list of words starting with that character. This sped things up by at least a factor of 26.

Performance in Devset

The –dev mode automatically takes the last tenth of the sample and uses that for the devset. The results from running the test set with the final n=2 were:

Author	# correct	devset	% correct
Austen	314	563	55.77265
Christie	127	526	24.14449
Dickens	167	258	64.72868
Doyle	169	403	41.93548
Thoureau	199	435	45.74713
Whitman	68	115	59.13043
Wilde	39	87	44.82759
Total	**1083**	**2387**	**48.04092**

From this, we can see that it very easily identifies Dickens and Whitman, and has excessive trouble with Christie, who just writes in completely different styles between the first ninth tenths of a book and the last tenth. It was observed that running on my machine, the –dev mode took just over 10 seconds.

Performance in Testing

In testing, another book was chosen for each author, and the total correct sentences was measured:

n=2	Austen	Christie	Dickens	Doyle	Thoureau	Whitman	Wilde
Austen	3586	864	971	450	48	28	4
Christie	355	740	459	230	20	6	4
Dickens	1257	705	2273	637	59	11	29
Doyle	654	655	772	836	35	4	5
Thoureau	448	582	685	283	252	21	5
Whitman	997	392	949	286	66	2366	3
Wilde	422	292	454	197	17	5	6

The cells that represent where an author was, overall guess correctly are highlighted in green. As stated before, this is much better than the n=2 or n=3 cases, and 5/7 authors guessed overall correctly is pretty good. That being said, implementing machine learning algorithms and backoff would likely increase accuracy. It was observed that on my machine, the –test mode took about 15 seconds with a large test file, and closer to 10 with a small test file.

Conclusions

This project was very interesting, though there is a lot of overhead to get it up and running. There is certainly a lot that could be incorporated into it, given more time, but running just bigrams and Good Turing smoothing does a remarkably good job at classifying authors.