CS165A Project1 Report
Shunji Zhan

Architecture
I wrote everything in one file so I didn't have any classes. I first read in the training data and parse it to array of arrays, with some filter to modify the data so that it contains less noise and extra punctuations.

Then I used the training data to build collections of words, so each word has its total frequency, as well as conditional frequency, saved in the collection.

In testing I read in all testing data and also parse them into array of arrays, similar to training process. Then I extract the 1 or 0 in the end of each comment, and formed them into "real score" array, in order to compare my prediction. Then I feed each comment (each element in the array) to my prediction function, which makes use of naïve bayes rule to predict the result, and formed a prediction array. I compare each element of prediction array and "real score" array to get my accuracy.

Preprocessing
I first read the training data into the program, and parse it into a big array, each element contains an array of words. So it is an array of arrays.

Then in order to eliminate too common words, I have an array of 200 most used words, and by trying several times I decided to use 75 of them and removed any appearance of 75 most common words from the array. I also stripped punctuations, plus some word tail, such as "ed", "ing", and "'d".

Then I separate the whole array into two parts, the first part contains only comments of '1', the other only contain comments of '0'.

Model Building
In the training process, I build three collection of words (after striped the 1 or 0 in the end), the first collection is the collection of all words and their frequency, the second is the collection of words and their frequency in all '0' comments, the third is the collection of words and their frequency in all '1' comments.

With these three collection I can calculate the frequency of 1 and 0, which is the 'premier', and I could potentially apply TF-IDF algorithm. What's more, it is easy to get conditionally word probability with the second and third word collection.

Results

My program ran for 6 seconds in training, and 7 seconds in testing. The accuracy is 0.989 for training data, and 0.824 for testing data.

I printed the ten most frequent words in all comments, '1' comment, and in '0' comment, and found out the most important words:

In all comments:
{'all': 4675, 'good': 2929, 'like': 3876, 'just': 3462, 'movie': 8446, 'who': 4104, 'but': 8345, 'not': 6067, 'more': 2863, 'film': 7736}

In '1' comments:
{'all': 2306, 'like': 1767, 'very': 1646, 'movie': 3584, 'who': 2209, 'but': 4070, 'not': 2764, 'more': 1477, 'film': 4053, 'out': 1657}

In '0' comments:
{'all': 2369, 'like': 2109, 'just': 1989, 'no': 1548, 'movie': 4862, 'who': 1895, 'but': 4275, 'not': 3303, 'film': 3683, 'out': 1674}

Challenges

There were three challenges that I faced during this project.

The first one is that at first I didn't really understand Naïve bayes rule, I solved this by discussing with friends, and searched a lot of examples online. And finally I understood it, and was to apply it correctly to the calculation. One trick that was particularly important is that I need to take log of probability and add them together, instead of multiply the probability. Otherwise the result will be too small. I got stuck in this detail for a while, but finally solved it.

The second challenge was that it is hard to manage array of arrays, I had to print them many times to see the result array to make sure the array is correct.

The third challenge was that I didn't use Python for a while, so I forget all the syntaxes. But this problem was not too hard to solve, I just went through some of my old Python program and review the syntax, and search online for any particular functionality.

Weakness

One thing I noticed was that the running time is relatively long, and I am sure this is because I stripped punctuations and things like "ed", and "ing", and there are 75 stop words. Without these stripping, the accuracy will be 2% lower, but the running time will decrease to about 1 or 2 second. I can potentially increase this by stripping less words, by picking the most efficient stripping words.

There can be improve in my algorithm, too. Since right now in predicting every comment, I treated every single word as a single element. I can potentially use TF-IDF algorithm to improve the accuracy. I can also include frequency count for two-word or even three-word frequency, and make the prediction based on all those frequencies and probabilities, I believe this will greatly improve accuracy.

In addition, when I printed out most frequent words I found the ‘br’ is frequently appear, so this html format tag actually counted as a word. If I stripped it from the dictionary it might improve the prediction a little bit, provided the testing data also contains such tag.