DiscreteMathematics_02_PA3
21800147 KimYouYoung
21800294 ParkSuah
21800811 ParkSangBeom

Problem

The goal of this problem is to make text classifier which can determine the sentence is negative or non-negative. Using test.negative.csv and test.non-negative.csv, the program trains that the word which is negative and non-negative. After training, we test it with other csv files to check it trains correctly or we can check with standard input. To train the program, we used method of Naive Bayes classifier. Calculate the probability based on simple computation, conditional probabilities. When a sentence comes in, then predict the sentence is negative or non-negative based on words which compose the sentence after stemming, reduce unnecessary words.

Trainer

a. Explanation

Trainer reads the data from a given file, train.negative.csv and train.non-negative.csv to obtain the probability of being negative and non-negative for each word. It also creates a file “model.csv” in the form of list including ‘word / probability of negative / probability of non-negative’.

b. Algorithm

To make sentences in each file into a set of words, we created functions that serves as tokenization, normalization, counting.

Open the negative file and non-negative file to make all the words inside the file tokenization. In this case, one of the parameter of this function indicates whether the data entered is data of the negative file or non-negative file. If the value of this parameter is 0, increase the number of negative sentences(integer ‘count_n’) by one, and if it is 1, increase the number of non-negative sentences(integer ‘count_nn’) by one. By checking each word divided by one character, non-alphabet characters(such as numbers and special characters) are removed and all capital letters are changed to lowercase. Use sb_stemmer_stem to stemming each word while moving in token units until the end of a sentence. And then use g_hash_table_lookup to check if the word is in stopword. If not stopword, use g_hash_table_lookup, g_hash_table_insert to check if the word is in the counter. If the word is in the counter table, increase the value of the word by one, and if not, put the word in the key and make the value 1 . At this point, the opposite value was zeroed to prevent the garbage value from entering the calculation.

We have generated the file containing stopwords. So we will open this file, put on a g_hash_table calld ‘stop’, and use it to filter out stopwords when we take the words token. This function simply turns stopwords in the form of files into g_hash_table so that we can compare them easily.

Next, generates model.csv file by calculating the data value we get from the probability. Only words with two or more values were put into model.csv within the counter table to filter out words that are rarely used here. In order to narrow the gap between the word

that came out once and the word that came out several times, all the words were set as basic 50 times before calculating the probability.

Predictor

a. Explanation

Predictor's role is to calculate the probability of negative in a sentence by analyzing whether the words in a sentence are negative or not. If the calculated probability exceeds the threshold, it is determined either as a negative sentence or as a non-negative sentence.

b. Algorithm

Use the hash table to put the words stored in the stopwords file as the key value, and another hash table to save the words read from model.csv as the key value, and the probabilities read as the value

We can receive input in two ways. The first can be entered as a file such as test.negative.csv, and the second can be received as a standard input by the user himself. After receiving input in one of two ways, separate the words in the sentence with a tokenization and change the unnecessary words to the appropriate form, such as deleting the stopwords, and convert uppercase letters to lowercase letters, and stemming them. This should be done exactly as we did in the trainer above.

After excluding the things that are not needed in the sentence, each negative and non-negative probability of the remaining words is read from the hash table(model.csv), and then the expression “probabilities = negative/ (negative+non_negative)” is applied to obtain the probability that this sentence is negative. If the probability of being negative exceeds the threshold set by us, the sentence shall be deemed negative. We set the threshold at 0.7, or 70%. The threshold was set to 0.7 because, on average, sentences with a probability of more than 70% were negative when tried several times.

Evaluation test data

After succeed to determine negatives with standard input, we wonder many sentence that we do not make is correct using this program. We use test.negative.csv and test.non-negative.csv to check it. According to this program, we expect 60% correctness but we underestimate it. When we make threshold value 0.7, test.negative.csv has 99% correctness and test.non-negative.csv has 87% correctness. I was disappointed that the test data was shorter than I thought, but I found that the results showed higher results than I expected, and the confidence in the “Naive Bayes text classifier" increased.

Discussion

a. Limitation

There were frequent errors of judgment because there was little data in the predictor. For example, when the sentence "Hi, my name is Jane" is entered, it should be non-negative, but it is judged that hi and name are negative because they have a higher probability of being negative. If there were more data, the results could have been different. Also, if a word has a neutral meaning or if a negative appears in a sentence even several times, it is non-negative,

but it is judged as negative.

b. Improvement

Collect and analyze more data. We can make more accurate judgments by dividing the various possible cases into various ones. As I said earlier, when two negative words like "You're not a bad boy," they'll be able to come up with more accurate results to take into account many of these situations, like non-negative rather than negative.

c. Learned

SangBeom: I used g_hash_table first time in c, its component is useful than I thought. I didn't expect dictionary function in c, but it was quite good. We tried understand the return value and parameters that each g_hash_table function has. Do_while is also very useful loop statements.

Suah: It took a long time to understand this problem, but it was convenient to use useful libraries and functions. From now on, I felt that I should make a good use of them without being reluctant to use them. If several people bring in emotion-classified data from different sources, more accurate and detailed emotional analysis is likely to be possible.

Youyoung: I have used a hash table when learning Java, but I have never known if there is a hash table function in C. Without the Hash table function, it would have taken more time and complexity to save each word and find each word that matches. Through this project, I found out the convenience of the hash table. It was also more effective in comparing the two, using probabilities.