|  |  |  |
| --- | --- | --- |
| **Discrete Mathematics** | Section | 02 |
| Student number | 22000690 |
| **HW4 – Spam filter** | Name | Jung, Yi Ju |

*If your explanation is less informative and insufficient, then you may not get any points.*

*Also, you should provide discussion, otherwise you will get penalty.*

* General information

|  |  |
| --- | --- |
| Item | Your answer |
| The number of lines in your code. | 464 lines. |
| The number of functions in your code. | 9 functions. |

* Functions

|  |  |
| --- | --- |
| Function name | Function Description |
| **void** read\_ham() | Read a dataset\_ham\_train100.csv.  Cut only the text portion of the mail into word units and place it in a structure that stores the words of train mail without overlapping.  When saving, save while marking whether the word is extracted from the ham file.  structure :  struct Word {  char word[100];  int count;  int ham\_num;  int spam\_num;  int check;  double h\_rw;  int isham;  };  struct Word w[1500000]; |
| **void** read\_spam() | Read a dataset\_spam\_train100.csv.  Cut only the text portion of the mail into word units and place it in a structure that stores the words of train mail without overlapping.  When saving, save while marking whether the word is extracted from the spam file.  structure :  struct Word {  char word[100];  int count;  int ham\_num;  int spam\_num;  int check;  double h\_rw;  int isham;  };  struct Word w[1500000]; |
| **void** count\_ham\_word() | Each word collected from the spam file and the ham file is to count how many mails in the dataset\_ham\_test20.csv file came out. To calculate q(each word). |
| **void** count\_spam\_word() | Each word collected from the spam file and the spam file is to count how many mails in the dataset\_spam\_test20.csv file came out. Need to calculate p(each word). |
| **void** cal\_rw() | This function calculate r(each word).  r(each word) = p(each word)/ (p(each word) + q(each word)).  Need each word’s rw is used to determine each mail is spam or non-spam. |
| **void** total\_rw() | This function find the rw value of each mail By calculate r(w1,..,wn).  r(w1,..,wn) = p(w1,..,wn) / (p(w1,..,wn) + q(w1,..,wn)).  Compare the obtained rw value and the threshold value to determine whether this mail is spam or non-spam. |
| **void** make\_spamWord() | It is a function of save the selected word in the structure.  Words will select that fulfill certain criteria.  structure :  struct Select\_Spam\_Word {  char word[100];  int ham\_num;  int spam\_num;  double rw;  };  struct Select\_Spam\_Word selectsw[1500000]; |
| **void** predict\_spamfile() | Read the dataet\_spam\_test20.csv file to determine whether each mail is spam mail or non-spam, and print the num (line) of spam mail this program predict and the each rw value. |
| **void** predict\_hamfile() | Read the dataet\_ham\_test20.csv file to determine whether each mail is spam mail or not, and print the num (line) of spam mail this program predict and the each rw value. |

* Screenshot of your program running
* Threshold : 0.6

텍스트이(가) 표시된 사진

자동 생성된 설명

* Threshold : 0.7

텍스트이(가) 표시된 사진

자동 생성된 설명

* Threshold : 0.8

텍스트이(가) 표시된 사진

자동 생성된 설명

* Threshold : 0.9

텍스트이(가) 표시된 사진

자동 생성된 설명

* Threshold : 0.95

텍스트이(가) 표시된 사진

자동 생성된 설명

* Results
  + Probability and predicted label (spam or non-spam) for different threshold (T) (0.6, 0.7, 0.8, 0.9 and 0.95)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| num  True | True Label | T = 0.6 | | T = 0.7 | | T = 0.8 | | T = 0.9 | | T = 0.95 | |
|  | predicted  label |  | predicted  label |  | predicted  label |  | predicted  label |  | predicted  label |
| s01 | Spam | 0.75 | spam | 0.75 | spam | 0.75 | non | 0.75 | non | 0.75 | non |
| s02 | Spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| s03 | Spam | 0.75 | spam | 0.75 | spam | 0.75 | non | 0.75 | non | 0.75 | non |
| s04 | Spam | 0.96 | spam | 0.96 | spam | 0.96 | spam | 0.96 | spam | 0.96 | spam |
| s05 | Spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| s06 | Spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| s07 | Spam | 0.89 | spam | 0.89 | spam | 0.89 | spam | 0.89 | non | 0.89 | non |
| s08 | Spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| s09 | Spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| s10 | Spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| s11 | Spam | 0.89 | spam | 0.89 | spam | 0.89 | spam | 0.89 | non | 0.89 | non |
| s12 | Spam | 0.96 | spam | 0.96 | spam | 0.96 | spam | 0.96 | spam | 0.96 | spam |
| s13 | Spam | 0.89 | spam | 0.89 | spam | 0.89 | spam | 0.89 | non | 0.89 | non |
| s14 | Spam | 0.75 | spam | 0.75 | spam | 0.75 | non | 0.75 | non | 0.75 | non |
| s15 | Spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| s16 | Spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| s17 | Spam | 0.96 | spam | 0.96 | spam | 0.96 | spam | 0.96 | spam | 0.96 | spam |
| s18 | Spam | 0.89 | spam | 0.89 | spam | 0.89 | spam | 0.89 | non | 0.89 | non |
| s19 | Spam | 0.89 | spam | 0.89 | spam | 0.89 | spam | 0.89 | non | 0.89 | non |
| s20 | Spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h01 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h02 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h03 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h04 | Non-spam | 0.75 | spam | 0.75 | spam | 0.75 | non | 0.75 | non | 0.75 | non |
| h05 | Non-spam | 0.89 | spam | 0.89 | spam | 0.89 | spam | 0.89 | non | 0.89 | non |
| h06 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h07 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h08 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h09 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h10 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h11 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h12 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h13 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h14 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h15 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h16 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h17 | Non-spam | 0.75 | spam | 0.75 | spam | 0.75 | non | 0.75 | non | 0.75 | non |
| h18 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h19 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| h20 | Non-spam | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non | 0.00 | non |
| Accuracy (%) | | 70% | | 70% | | 67.5% | | 57.5% | | 57.5% | |

※ Accuracy is calculated from the following equation.

* Discussion (your interpretation of the results and possible strategy to improve the algorithm)
* After cutting the emails in the train folder into words, This program learned the word exclude words composed of Korean characters. After putting the words in the structure Word, calculate the rw values of each word based on the mails in the test folder. To calculate rw value of each word this program find out how many mails each word has been counted by count\_ham\_word() and count\_spam\_word() functions. When we count all word get from test folder. Calculate rw value using cal\_rw() function.

Two representative words were selected by comparing the rw values of each words and give a specific criteria. And save a selected word in structure Select\_Spam\_Word. After that, calculate the rw value corresponding to each mail in the test folder by using total\_rw() function, and then compared with the threshold value to predict whether each mail was spam mail or non-spam mail by using predict\_hamfile() and predict\_spamfile() functions.

If the threshold value is high, all values predicted to be spam mail are spam mail, and none of the predicted values are spam mail. However, we cannot confirm 100% that it is spam mail. Therefore, the accuracy may be low compared to the high threshold value, but the precision will be high.

If the threshold value is low, more spam mail can be predicted compared to the high threshold value, however non-spam mail may be included among the mails predicted to be spam mail. Therefore, the accuracy may be high compared to the high threshold value, but the precision will be low. From these results, there is a difference in accuracy and precision according to the threshold value.

Depending on the situation, the threshold value may be determined according to the selection of a high value between accuracy and precision, or the threshold value in which the two values have an average value may be selected. As a one example of a situation where accuracy must be high, it is COVID-19 PCR test. To improve accuracy in this case is very important problem because it can reduce the damage to many people, such as predicting all negative results, rather than predicting that it is positive once it is negative.

To improve this algorithm, data will need to be training with more words. Then more words will fulfil the criteria that must be met at the time of selection, which will improve accuracy. And also, if the test is performed by further subdividing the range of threshold values, the accuracy will vary depending on each threshold value, so you can choose the threshold with the highest accuracy and high precision.

□ Codes

// Put code here, and you should also submit your original executable C code.

// If C code does not run, then No point will be given.

#include <stdio.h>

#include <string.h>

**float** threshold = 0.6;

**double** rw = 0.0;

**struct** Word {

**char** word[100];

**int** count;

**int** ham\_num;

**int** spam\_num;

**int** check;

**double** h\_rw;

**int** isham; //0: ham , 1: spam

};

**struct** Word w[1500000];

**int** w\_listindex = 0;

**struct** Select\_Spam\_Word {

**char** word[100];

**int** ham\_num;

**int** spam\_num;

**double** rw;

};

**struct** Select\_Spam\_Word selectsw[1500000];

**int** select\_w\_listindex = 0;

**void** read\_ham() {

**int** line = 0;

FILE \*pFile = fopen("/Users/jeong-yiju/Desktop/Homework4/datasets/train/dataset\_ham\_train100.csv","r");

**char** data[1024];

**char** \*temp[150000] = {**NULL**, };

**while** (!feof(pFile) ) {

**int** dupulicate\_count = 0;

fscanf(pFile, "%s", data);

**if**(line != 0){

**if**(strstr(data, ",ham,") == **NULL**){

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(strcmp(w[i].word, data) == 0){

w[i].count = (w[i].count) + 1;

dupulicate\_count++;

}

}

**if**(dupulicate\_count == 0){

strcpy(w[w\_listindex].word, data);

w[w\_listindex].count = 1;

w[w\_listindex].check = 0;

w[w\_listindex].isham = 0;

w\_listindex++;

}

}

**else**{

**int** i = 0;

**char** \*cut = strtok(data, ",");

**while** (cut != **NULL**){

temp[i] = cut;

i++;

cut = strtok(**NULL**, ",");

}

//printf("%s\n", temp[2]);

strcpy(w[w\_listindex].word, temp[2]);

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(strcmp(w[i].word, temp[2]) == 0){

w[i].count = (w[i].count) + 1;

dupulicate\_count++;

}

}

**if**(dupulicate\_count == 0){

strcpy(w[w\_listindex].word, temp[2]);

w[w\_listindex].count = 1;

w[w\_listindex].isham = 0;

w\_listindex++;

}

}

}

line++;

}

fclose(pFile);

}

**void** read\_spam() {

**int** line = 0;

FILE \*pFile = fopen("/Users/jeong-yiju/Desktop/Homework4/datasets/train/dataset\_spam\_train100.csv","r");

**char** data[1024];

**char** \*temp[150000] = {**NULL**, };

**while** (!feof(pFile) ) {

**int** dupulicate\_count = 0;

fscanf(pFile, "%s", data);

**if**(line != 0){

**if**(strstr(data, ",spam,") == **NULL**){

//strcpy(w[w\_listindex].word, data);

//printf("%s\n", data);

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(strcmp(w[i].word, data) == 0){

w[i].count = (w[i].count) + 1;

dupulicate\_count++;

}

}

**if**(dupulicate\_count == 0){

strcpy(w[w\_listindex].word, data);

w[w\_listindex].count = 1;

w[w\_listindex].isham = 1;

w\_listindex++;

}

}

**else**{

**int** i = 0;

**char** \*cut = strtok(data, ",");

**while** (cut != **NULL**){

temp[i] = cut;

i++;

cut = strtok(**NULL**, ",");

}

//printf("%s\n", temp[2]);

strcpy(w[w\_listindex].word, temp[2]);

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(strcmp(w[i].word, temp[2]) == 0){

w[i].count = (w[i].count) + 1;

dupulicate\_count++;

}

}

**if**(dupulicate\_count == 0){

strcpy(w[w\_listindex].word, temp[2]);

w[w\_listindex].count = 1;

w[w\_listindex].isham = 1;

w\_listindex++;

}

}

}

line++;

}

fclose(pFile);

}

**void** count\_ham\_word(){

**int** line = 0;

FILE \*pFile = fopen("/Users/jeong-yiju/Desktop/Homework4/datasets/test/dataset\_ham\_test20.csv","r");

**char** data[1024];

**char** \*temp[150000] = {**NULL**, };

**int** currnum = 0;

**int** num;

**while** (!feof(pFile) ) {

fscanf(pFile, "%s", data);

**if**(line != 0){

//printf("%s\n", data);

**if**(strstr(data, ",ham,") == **NULL**){

//strcpy(w[w\_listindex].word, data);

//printf("%s\n", data);

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(strcmp(w[i].word, data) == 0 && w[i].check == 0){

//w[i].ham\_num = (w[i].ham\_num) + 1;

**if**(num != currnum){

w[i].ham\_num = (w[i].ham\_num) + 1;

w[i].check = 1;

}

}

}

}

**else**{

num = currnum;

**for**(**int** i = 0; i < w\_listindex; i++) {

w[i].check = 0;

}

**int** i = 0;

**char** \*cut = strtok(data, ",");

**while** (cut != **NULL**){

temp[i] = cut;

i++;

cut = strtok(**NULL**, ",");

}

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(strcmp(w[i].word, temp[2]) == 0){

w[i].ham\_num = (w[i].ham\_num) + 1;

}

}

currnum++;

}

}

line++;

}

fclose(pFile);

}

**void** count\_spam\_word(){

**int** line = 0;

FILE \*pFile = fopen("/Users/jeong-yiju/Desktop/Homework4/datasets/test/dataset\_spam\_test20.csv","r");

**char** data[1024];

**char** \*temp[150000] = {**NULL**, };

**int** currnum = 0;

**int** num;

**while** (!feof(pFile) ) {

fscanf(pFile, "%s", data);

**if**(line != 0){

//printf("%s\n", data);

**if**(strstr(data, ",spam,") == **NULL**){

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(strcmp(w[i].word, data) == 0 && w[i].check == 0){

//w[i].spam\_num = (w[i].spam\_num) + 1;

**if**(num != currnum){

w[i].spam\_num = (w[i].spam\_num) + 1;

w[i].check = 1;

}

}

}

}

**else**{

num = currnum;

**for**(**int** i = 0; i < w\_listindex; i++) {

w[i].check = 0;

}

**int** i = 0;

**char** \*cut = strtok(data, ",");

**while** (cut != **NULL**){

temp[i] = cut;

i++;

cut = strtok(**NULL**, ",");

}

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(strcmp(w[i].word, temp[2]) == 0){

w[i].spam\_num = (w[i].spam\_num) + 1;

}

}

currnum++;

}

}

line++;

}

fclose(pFile);

}

**void** cal\_rw(){

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(w[i].ham\_num == 0 && w[i].spam\_num != 0){

w[i].h\_rw = (**float**)1;

}

**else** **if**(w[i].spam\_num == 0 && w[i].ham\_num != 0){

w[i].h\_rw = (**float**)0;

}

**else** **if** (w[i].ham\_num != 0 && w[i].spam\_num != 0){

w[i].h\_rw = ((**float**)w[i].spam\_num / 20) / (((**float**)w[i].ham\_num / 20) + ((**float**)w[i].spam\_num / 20));

}

}

}

**void** total\_rw(){

**double** pw = 1.0;

**double** qw = 1.0;

**for**(**int** i = 0; i < select\_w\_listindex; i++) {

pw = pw \* ((**float**)selectsw[i].spam\_num / 20);

qw = qw \* ((**float**)selectsw[i].ham\_num / 20);

}

rw = pw / (pw + qw);

printf("Total rw: %.2f\n", rw);

}

**void** make\_spamWord(){

**for**(**int** i = 0; i < w\_listindex; i++) {

**if**(w[i].h\_rw != 1.0 && w[i].h\_rw > 0.6 && w[i].spam\_num > 4 && strlen(w[i].word) > 2 && w[i].ham\_num < 3){

strcpy(selectsw[select\_w\_listindex].word, w[i].word);

selectsw[select\_w\_listindex].ham\_num = w[i].ham\_num;

selectsw[select\_w\_listindex].spam\_num = w[i].spam\_num;

selectsw[select\_w\_listindex].rw = w[i].h\_rw;

select\_w\_listindex++;

//printf("word: %s \*\*HAM: %d \*\*SPAM: %d $$h\_rw: %.2f\n", w[i].word, w[i].ham\_num, w[i].spam\_num, w[i].h\_rw);

}

}

}

**void** predict\_spamfile() {

**char** w[100];

**char** h[100];

**double** rw\_w;

**double** rw\_h;

**int** correct\_h = 0;

**int** correct\_w = 0;

**int** line = 0;

FILE \*pFile = fopen("/Users/jeong-yiju/Desktop/Homework4/datasets/test/dataset\_spam\_test20.csv","r");

**char** data[1024];

**int** currnum = 1;

**int** num = 0;

printf("\n\*\*\*\*\*\*\*\*\*\* test spam file \*\*\*\*\*\*\*\*\*\*\n");

printf("spam mail line is...\n");

**while** (!feof(pFile) ) {

fscanf(pFile, "%s", data);

**if**(line != 0){

//printf(" word: %s\n", data);

**if**(strstr(data, ",spam,") == **NULL**){

**if**(threshold < rw){

**for**(**int** i = 0; i < select\_w\_listindex; i++) {

**if**(strcmp(selectsw[i].word, data) == 0){

**if**(strcmp(selectsw[i].word, "com")){

correct\_w++;

rw\_w = selectsw[i].rw;

strcpy(w, selectsw[i].word);

}

**if**(strcmp(selectsw[i].word, "here")){

correct\_h++;

rw\_h = selectsw[i].rw;

strcpy(h, selectsw[i].word);

}

}

}

}

}

**else**{

**if**(correct\_h > 0 && correct\_w > 0){

printf("line: %d spam mail!!\n", num);

}

**else** **if**(correct\_h > 0){

**if**(rw\_h > threshold){

printf("line: %d word: %s wr: %.2f", num, h, rw\_h);

printf(" spam mail!!\n");

}

}

**else** **if**(correct\_w > 0){

**if**(rw\_w > threshold){

printf("line: %d word: %s wr: %.2f", num, w, rw\_w);

printf(" spam mail!!\n");

}

}

correct\_h = 0;

correct\_w = 0;

num = currnum;

currnum++;

}

}

line++;

}

fclose(pFile);

}

**void** predict\_hamfile() {

**char** w[100];

**char** h[100];

**int** correct\_h = 0;

**int** correct\_w = 0;

**int** line = 0;

FILE \*pFile = fopen("/Users/jeong-yiju/Desktop/Homework4/datasets/test/dataset\_ham\_test20.csv","r");

**char** data[1024];

**int** currnum = 1;

**int** num = 0;

**double** rw\_w;

**double** rw\_h;

printf("\n\*\*\*\*\*\*\*\*\*\* test ham file \*\*\*\*\*\*\*\*\*\*\n");

printf("spam mail line is…\n");

**while** (!feof(pFile) ) {

fscanf(pFile, "%s", data);

**if**(line != 0){

**if**(strstr(data, ",ham,") == **NULL**){

**if**(threshold < rw){

**for**(**int** i = 0; i < select\_w\_listindex; i++) {

**if**(strcmp(selectsw[i].word, data) == 0){

**if**(strcmp(selectsw[i].word, "com")){

correct\_w++;

rw\_w = selectsw[i].rw;

strcpy(w, selectsw[i].word);

}

**if**(strcmp(selectsw[i].word, "here")){

correct\_h++;

rw\_h = selectsw[i].rw;

strcpy(h, selectsw[i].word);

}

}

}

}

}

**else**{

**if**(correct\_h > 0 && correct\_w > 0){

printf("line: %d spam mail!!\n", num);

}

**else** **if**(correct\_h > 0){

**if**(rw\_h > threshold){

printf("line: %d word: %s wr: %.2f", num, h, rw\_h);

printf(" spam mail!!\n");

}

}

**else** **if**(correct\_w > 0){

**if**(rw\_w > threshold){

printf("line: %d word: %s wr: %.2f", num, w, rw\_w);

printf(" spam mail!!\n");

}

}

correct\_h = 0;

correct\_w = 0;

num = currnum;

currnum++;

}

}

line++;

}

fclose(pFile);

}

**int** main() {

read\_ham();

read\_spam();

count\_ham\_word();

count\_spam\_word();

cal\_rw();

make\_spamWord();

total\_rw();

printf("\*\*\*\*\*\*\*\*\*\*Selected words are\*\*\*\*\*\*\*\*\*\*\n");

**for**(**int** i = 0; i < select\_w\_listindex; i++) {

printf("word: %s", selectsw[i].word);

printf(" ");

}

predict\_spamfile();

predict\_hamfile();

}