|  |  |  |
| --- | --- | --- |
| **Discrete Mathematics** | Section | 2 |
| Student number | 21900706 |
| **HW6 – Spam filter** | Name | 조영관 |

*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. | 273 lines |
| The number of functions in your code. | 7 functions |

* Functions

|  |  |
| --- | --- |
| Function name | Function Description |
| void copy | Copying string type to char \* type to use in data parsing in tokenizing function |
| Void tokenizing | Using delimiters, parse string data type in to words. |
| Void makefile | Check number of occurrences of words and make text file. On the text file, each words and number of emails each words showed up is written |
| Void readfile (2 functions, different parameters) | Read textfiles created from makefile function to use in main function for spam mail filtering. |
| Void preProcessing | Read csv files and read each lines in to string type. call tokenizing and makefile functions to parse data for appropriate usage. |
| Void main | Use data created by preProcessing function to do mathematical calculations for spam check and accuracy of each thresholds. |

* Screenshot of your program running

Text

Description automatically generatedT=0.6

T=0.7

A picture containing calendar

Description automatically generated

Text

Description automatically generated T=0.8

T=0.9

A picture containing text

Description automatically generated

T=0.95

Text

Description automatically generated

* 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 | 0.972503 | spam | 0.972503 | spam | 0.972503 | spam | 0.972503 | spam | 0.972503 | spam |
| s02 | Spam | 0.733238 | spam | 0.733238 | spam | 0.733238 | Not spam | 0.733238 | Not spam | 0.733238 | Not spam |
| s03 | Spam | 0.999786 | spam | 0.999786 | spam | 0.999786 | spam | 0.999786 | spam | 0.999786 | spam |
| s04 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| s05 | Spam | 0.997746 | spam | 0.997746 | spam | 0.997746 | spam | 0.997746 | spam | 0.997746 | spam |
| s06 | Spam | 0.993974 | spam | 0.993974 | spam | 0.993974 | spam | 0.993974 | spam | 0.993974 | spam |
| s07 | Spam | 0.999889 | spam | 0.999889 | spam | 0.999889 | spam | 0.999889 | spam | 0.999889 | spam |
| s08 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| s09 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| s10 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| s11 | Spam | 0.996686 | spam | 0.996686 | spam | 0.996686 | spam | 0.996686 | spam | 0.996686 | spam |
| s12 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| s13 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| s14 | Spam | 0.999524 | spam | 0.999524 | spam | 0.999524 | spam | 0.999524 | spam | 0.999524 | spam |
| s15 | Spam | 0.5 | Not spam | 0.5 | Not spam | 0.5 | Not spam | 0.5 | Not spam | 0.5 | Not spam |
| s16 | Spam | 0.694444 | spam | 0.694444 | Not spam | 0.694444 | Not spam | 0.694444 | Not spam | 0.694444 | Not spam |
| s17 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| s18 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| s19 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| s20 | Spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| h01 | Non-spam | 7.20252e-13 | Not spam | 7.20252e-13 | Not spam | 7.20252e-13 | Not spam | 7.20252e-13 | Not spam | 7.20252e-13 | Not spam |
| h02 | Non-spam | 3.7344e-06 | Not spam | 3.7344e-06 | Not spam | 3.7344e-06 | Not spam | 3.7344e-06 | Not spam | 3.7344e-06 | Not spam |
| h03 | Non-spam | 0.010185 | Not spam | 0.010185 | Not spam | 0.010185 | Not spam | 0.010185 | Not spam | 0.010185 | Not spam |
| h04 | Non-spam | 0.942831 | spam | 0.942831 | spam | 0.942831 | spam | 0.942831 | spam | 0.942831 | Not spam |
| h05 | Non-spam | 7.20757e-11 | Not spam | 7.20757e-11 | Not spam | 7.20757e-11 | Not spam | 7.20757e-11 | Not spam | 7.20757e-11 | Not spam |
| h06 | Non-spam | 2.49234e-08 | Not spam | 2.49234e-08 | Not spam | 2.49234e-08 | Not spam | 2.49234e-08 | Not spam | 2.49234e-08 | Not spam |
| h07 | Non-spam | 1.45708e-67 | Not spam | 1.45708e-67 | Not spam | 1.45708e-67 | Not spam | 1.45708e-67 | Not spam | 1.45708e-67 | Not spam |
| h08 | Non-spam | 5.4322e-08 | Not spam | 5.4322e-08 | Not spam | 5.4322e-08 | Not spam | 5.4322e-08 | Not spam | 5.4322e-08 | Not spam |
| h09 | Non-spam | 0.99646 | spam | 0.99646 | spam | 0.99646 | spam | 0.99646 | spam | 0.99646 | spam |
| h10 | Non-spam | 0 | Not spam | 0 | Not spam | 0 | Not spam | 0 | Not spam | 0 | Not spam |
| h11 | Non-spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam | 1 | spam |
| h12 | Non-spam | 0.192791 | Not spam | 0.192791 | Not spam | 0.192791 | Not spam | 0.192791 | Not spam | 0.192791 | Not spam |
| h13 | Non-spam | 1.1897e-38 | Not spam | 1.1897e-38 | Not spam | 1.1897e-38 | Not spam | 1.1897e-38 | Not spam | 1.1897e-38 | Not spam |
| h14 | Non-spam | 0.0187811 | Not spam | 0.0187811 | Not spam | 0.0187811 | Not spam | 0.0187811 | Not spam | 0.0187811 | Not spam |
| h15 | Non-spam | 0.999088 | spam | 0.999088 | spam | 0.999088 | spam | 0.999088 | spam | 0.999088 | spam |
| h16 | Non-spam | 0.442844 | Not spam | 0.442844 | Not spam | 0.442844 | Not spam | 0.442844 | Not spam | 0.442844 | Not spam |
| h17 | Non-spam | 6.34931e-07 | Not spam | 6.34931e-07 | Not spam | 6.34931e-07 | Not spam | 6.34931e-07 | Not spam | 6.34931e-07 | Not spam |
| h18 | Non-spam | 1.06621e-06 | Not spam | 1.06621e-06 | Not spam | 1.06621e-06 | Not spam | 1.06621e-06 | Not spam | 1.06621e-06 | Not spam |
| h19 | Non-spam | 1.25719e-11 | Not spam | 1.25719e-11 | Not spam | 1.25719e-11 | Not spam | 1.25719e-11 | Not spam | 1.25719e-11 | Not spam |
| h20 | Non-spam | 0.0143857 | Not spam | 0.0143857 | Not spam | 0.0143857 | Not spam | 0.0143857 | Not spam | 0.0143857 | Not spam |
| Accuracy (%) | | 87.5% | | 85% | | 82.5% | | 82.5% | | 85% | |

※ Accuracy is calculated from the following equation.

* Discussion (your interpretation of the results and possible strategy to improve the algorithm)
  + The accuracy of spam filtering was the highest on the 0.6 thresholds, followed by 0.7, 0.95, 0.8, and 0.9. Thus, on the threshold between 0.6 to 0.95 will have over 80 percent of accuracy, which is a decent result. Some of the given 20 spam test emails had words commonly used in the working environment, and one had no written information. Some of the 20 non-spam test emails included words used in marketing or shopping advertisements. These features affected the overall accuracy, and such emails could not filter out using the Bayesian spam filter. If adequate emails for testing had been provided, the general accuracy of the program would have been much higher.
  + To improve the algorithm, possible strategies would be having more training emails and using appropriate testing emails so that the spam filter works appropriately and increases the algorithm's accuracy. In addition, adding or changing the functions to receive both .csv and .txt file formats would improve the overall program usage and allow people to use it more often.

□ Codes

#include <iostream>

#include <fstream>

#include <string>

#include <queue>

#include <map>

#include <chrono>

using namespace std;

void copy(string first, char \*second)

{

int i;

for (i = 0; first[i] != '\0'; i++)

second[i] = first[i];

second[i] = first[i];

}

void tokenizing(string mail, queue<string> &Queue)

{

char \*token;

char \*line = new char[mail.size() + 1];

copy(mail, line);

char delimiter[] = {" ,.'`÷╥╧{[]}~|?!-=/+%^@&\*$#\_\"<>\\:;1234567890()"};

token = strtok\_r(line, delimiter, &line);

while (token)

{

Queue.push(token);

token = strtok\_r(line, delimiter, &line);

}

}

void makeFile(const string filename)

{

ifstream input(filename);

int flag = 0;

int i;

map<string, unsigned int> word\_data;

map<string, unsigned int> label;

word\_data.clear();

string word;

string hamm("ham");

string spamm("spam");

string compare("subject");

string previous = compare;

while (input >> word)

{

if (word.compare(hamm) == 0)

{

flag++;

continue;

}

if (word.compare(spamm) == 0)

{

flag++;

continue;

}

if ((flag + 1) % 2 == 0 && word.compare(compare) == 0)

{

flag++;

label.clear();

continue;

}

if (flag % 2 == 0)

{

if (word\_data.find(word) != word\_data.end())

{

word\_data[word]++;

label[word]++;

if (label[word] > 1)

word\_data[word]--;

}

else

word\_data[word] = 1;

}

else

flag--;

}

input.close();

ofstream output(filename);

for (auto iter = word\_data.begin(); iter != word\_data.end(); ++iter)

{

output << iter->second << "\t" << iter->first << "\n";

}

}

void readFile(const string filename, map<string, unsigned int> &spamWord)

{

ifstream input(filename);

string word;

unsigned int num;

while (input >> num >> word)

{

spamWord.insert(pair<string, int>(word, num));

}

input.close();

}

void readFile(const string filename, queue<string> &spamWord)

{

ifstream input(filename);

string word;

while (input >> word)

{

spamWord.push(word);

}

input.close();

}

void preProcessing(queue<string> &Queue)

{

ifstream ham\_train("./train/dataset\_ham\_train100.csv");

ifstream spam\_train("./train/dataset\_spam\_train100.csv");

ifstream ham\_test("./test/dataset\_ham\_test20.csv");

ifstream spam\_test("./test/dataset\_spam\_test20.csv");

string line;

getline(ham\_train, line, '\n');

ofstream fout;

while (!ham\_train.eof())

{

getline(ham\_train, line, '\n');

transform(line.begin(), line.end(), line.begin(), ::tolower);

tokenizing(line, Queue);

fout.open("ham\_train.txt", ios::out | ios::app);

while (!Queue.empty())

{

fout << Queue.front() << "\n";

Queue.pop();

}

fout.close();

}

getline(spam\_train, line, '\n');

while (!spam\_train.eof())

{

getline(spam\_train, line, '\n');

transform(line.begin(), line.end(), line.begin(), ::tolower);

tokenizing(line, Queue);

fout.open("spam\_train.txt", ios::out | ios::app);

while (!Queue.empty())

{

fout << Queue.front() << "\n";

Queue.pop();

}

fout.close();

}

getline(ham\_test, line, '\n');

while (!ham\_test.eof())

{

getline(ham\_test, line, '\n');

transform(line.begin(), line.end(), line.begin(), ::tolower);

tokenizing(line, Queue);

fout.open("ham\_test.txt", ios::out | ios::app);

while (!Queue.empty())

{

fout << Queue.front() << "\n";

Queue.pop();

}

fout.close();

}

getline(spam\_test, line, '\n');

while (!spam\_test.eof())

{

getline(spam\_test, line, '\n');

transform(line.begin(), line.end(), line.begin(), ::tolower);

tokenizing(line, Queue);

fout.open("spam\_test.txt", ios::out | ios::app);

while (!Queue.empty())

{

fout << Queue.front() << "\n";

Queue.pop();

}

fout.close();

}

makeFile("ham\_train.txt");

makeFile("spam\_train.txt");

}

int main()

{

queue<string> Queue;

preProcessing(Queue);

double spam\_chance = 1.0;

double ham\_chance = 1.0;

map<string, unsigned int> spam\_train;

map<string, unsigned int> ham\_train;

queue<string> spam\_test;

queue<string> ham\_test;

string before;

int i = 0;

readFile("ham\_train.txt", ham\_train);

readFile("spam\_train.txt", spam\_train);

readFile("ham\_test.txt", ham\_test);

readFile("spam\_test.txt", spam\_test);

double result\_spam[20] = {0.0};

double result\_ham[20] = {0.0};

int a = 0;

while (!spam\_test.empty())

{

if (a != 2)

{

if (spam\_test.front() == "subject" || spam\_test.front() == "spam")

{

a++;

spam\_test.pop();

continue;

}

}

if (spam\_train.find(spam\_test.front()) != spam\_train.end() && ham\_train.find(spam\_test.front()) != ham\_train.end() && spam\_test.front() != "subject")

{

spam\_chance \*= spam\_train[spam\_test.front()] / 100.0;

ham\_chance \*= ham\_train[spam\_test.front()] / 100.0;

}

if ((spam\_test.front() == "subject" && before == "spam") || spam\_test.size() < 2)

{

double go = spam\_chance / (spam\_chance + ham\_chance);

result\_spam[i++] = go;

spam\_chance = 1.0;

ham\_chance = 1.0;

}

before = spam\_test.front();

spam\_test.pop();

}

spam\_chance = 1.0;

ham\_chance = 1.0;

i = 0;

a = 0;

int cnt = 0;

while (!ham\_test.empty())

{

if (a != 2)

{

if (ham\_test.front() == "subject" || ham\_test.front() == "ham")

{

a++;

ham\_test.pop();

continue;

}

}

if (spam\_train.find(ham\_test.front()) != spam\_train.end() && ham\_train.find(ham\_test.front()) != ham\_train.end() && ham\_test.front() != "subject")

{

spam\_chance \*= spam\_train[ham\_test.front()] / 100.0;

ham\_chance \*= ham\_train[ham\_test.front()] / 100.0;

}

if ((ham\_test.front() == "subject" && before == "ham") || ham\_test.size() < 2)

{

double go = spam\_chance / (spam\_chance + ham\_chance);

result\_ham[i++] = go;

spam\_chance = 1.0;

ham\_chance = 1.0;

}

before = ham\_test.front();

ham\_test.pop();

}

for (int k = 0; k < 20; k++)

{

cout << "spam mail " << k + 1 << " is " << result\_spam[k];

if (result\_spam[k] > 0.95)

{

cout << " and spam mail" << endl;

cnt++;

}

else

cout << " and ham mail" << endl;

}

for (int k = 0; k < 20; k++)

{

cout << "ham mail " << k + 1 << " is " << result\_ham[k];

if (result\_ham[k] > 0.95)

cout << " and spam mail" << endl;

else

{

cout << " and ham mail" << endl;

cnt++;

}

}

float accuracy = cnt / 40.0 \* 100.0;

cout << "Accuracy : " << accuracy << endl;

return 0;

}