1.Experiment explanation

1. 1-4 Read training files one by one.

2. Open a file and read it one line at a time.

3. In the article you read, "?! '," Quot ;, and & quot; space & quot;

4. Divide the revised text by a space and read it as a single word.

5. Remove redundancy by storing the word in the map data structure.

6. Proceed through files 1 to 4 in order.

7. 1 ~ 4 files Read the file once again, store the category number based on the words in the category, and store the number of documents per category.

Also, store how many words in a category are displayed based on words that have been previously saved. (However, even if the same word occurs multiple times in one category, it is assumed to be number 1.)

8. Use the numbers above to find the ky square value (select MAX value).

9. Sort the words in descending order based on each word.

10. Finally, it reads 5 test files, extracts the words from the training file, and sorts them based on the chi-squared values ​​of the words.

2. Preprocessing explanation

* In the article you read, "?! ',[]@\" Quot ;, and & quot; space & quot;
* Divide the revised text by a space and read it as a single word.
* Remove redundancy by storing the word in the map data structure.
* Proceed through files 1 to 4 in order.
* 1 ~ 4 files Read the file once again, store the category number based on the words in the category, and store the number of documents per category. Also, store how many words in a category are displayed based on words that have been previously saved. (However, even if the same word occurs multiple times in one category, it is assumed to be number 1.)

3. Performance Analysis

환경

OS : Window10

CPU : Intel i7-4712MQ 2.30GHz

RAM : 8GB

**Time to create training file and test file** : 1min 38sec

**Line** : 532

**Training Time** : 6분 48

**Test Time** : 47초

**Accuracy** : 70.537% (2837/4022)

4. Error analysis

**Number of false negative and false positive**

--------------------------------

Category 1

Total number of 1 category = 101

False Negative = 55

False Positive = 29

--------------------------------

--------------------------------

Category 2

Total number of 2 category = 1235

False Negative = 253

False Positive = 434

--------------------------------

--------------------------------

Category 3

Total number of 3 category = 141

False Negative = 111

False Positive = 41

--------------------------------

--------------------------------

Category 4

Total number of 4 category = 104

False Negative = 45

False Positive = 27

--------------------------------

--------------------------------

Category 5

Total number of 5 category = 601

False Negative = 171

False Positive = 158

--------------------------------

--------------------------------

Category 6

Total number of 6 category = 883

False Negative = 156

False Positive = 113

--------------------------------

--------------------------------

Category 7

Total number of 7 category = 882

False Negative = 354

False Positive = 365

--------------------------------

--------------------------------

Category 8

Total number of 8 category = 75

False Negative = 45

False Positive = 23

5. Conclusion

Inadequate point: The investigation of 을, 를, 와, and 의 is unnecessary and recognizes it as another word. In addition, words such as a word indicating time (today, tomorrow) are unnecessary elements for category discrimination.

Future Experimental Plan: We want to improve the items that degrade the accuracy mentioned above. Separate surveys from words that contain unnecessary surveys and classify them into the same words.

Time words are also classified and removed and not included in the word.

We will improve the accuracy by applying two improvement measures.

**Code**

#include "stdafx.h"

using namespace std;

// Temporary storage for sorting

map<string, int> temp\_count;

vector<pair<double,int>> temp\_X2;

string::size\_type lastPos;

string::size\_type pos;

/\*

Function the token

reference : https://wiki.kldp.org/HOWTO/html/C++Programming-HOWTO/standard-string.html

\*/

void Tokenize(const string& str, map<string, double>& tokens, map<string, int> number\_doc[8], int category\_index, const string& delimiters = " ") {

// Ignore if the first character is a delimiter

lastPos = str.find\_first\_not\_of(delimiters, 0);

// Find the first non-delimiter

pos = str.find\_first\_of(delimiters, lastPos);

string word;

// Cut one line by ""

// Save the classified features in categories & one map without duplicates

while (string::npos != pos || string::npos != lastPos) {

word = str.substr(lastPos, pos - lastPos);

if (tokens.count(word) == 0) tokens.insert(pair<string, double>(word, 0.0));

number\_doc[category\_index].insert(pair<string, int>(word, 1));

// Skip the delimiter. Note the "not\_of"

lastPos = str.find\_first\_not\_of(delimiters, pos);

// Find the next non-delimiter character

pos = str.find\_first\_of(delimiters, lastPos);

}

}

//Function the token

void TokenizeC(const string& str, map<string, double>& tokens, map<string, int> number\_doc[8], int category\_index, const string& delimiters = " ") {\

// Ignore if the first character is a delimiter

lastPos = str.find\_first\_not\_of(delimiters, 0);

// Find the first non-delimiter

pos = str.find\_first\_of(delimiters, lastPos);

string word;

// Cut one line by ""

// Feature stores the total number of documents in each category

while (string::npos != pos || string::npos != lastPos) {

word = str.substr(lastPos, pos - lastPos);

// Save the number of documents

if (number\_doc[category\_index].count(word)>0) {

if (temp\_count.count(word) == 0) temp\_count.insert(pair<string, int>(word, 1));

}

// Skip the delimiter. Note the "not\_of"

lastPos = str.find\_first\_not\_of(delimiters, pos);

// Find the next non-delimiter character

pos = str.find\_first\_of(delimiters, lastPos);

}

}

//Function the token

void TokenizeR(const string& str, map<string, double>& feature, map<string, int>& multi\_feature, const string& delimiters = " ") {

// Ignore if the first character is a delimiter

lastPos = str.find\_first\_not\_of(delimiters, 0);

// Find the first non-delimiter

pos = str.find\_first\_of(delimiters, lastPos);

int index;

string word;

double X2;

map<string, double>::iterator it;

// Search for saved Chi values and indexes and save them temporarily

while (string::npos != pos || string::npos != lastPos) {

word = str.substr(lastPos, pos - lastPos);

if (feature.count(word) > 0) {

X2 = feature.at(word);

index = multi\_feature.at(word);

temp\_X2.push\_back(pair<double, int>(X2, index));

}

// Skip the delimiter. Note the "not\_of"

lastPos = str.find\_first\_not\_of(delimiters, pos);

// Find the next non-delimiter character

pos = str.find\_first\_of(delimiters, lastPos);

}

}

// This function returns true if the first pair is "less"

// than the second one according to some metric

// In this case, we say the first pair is "less" if the first element of the first pair

// is less than the first element of the second pair

bool pairCompare(const pair<double, string>& firstElem, const pair<double, string>& secondElem) {

return firstElem.first > secondElem.first;

}

bool pairCompareR(const pair<double, int>& firstElem, const pair<double, int>& secondElem) {

return firstElem.first > secondElem.first;

}

/\*

1. 1-4 Read training files one by one.

2. Open a file and read it one line at a time.

3. In the article you read, "?! '," Quot ;, and & quot; space & quot;

4. Divide the revised text by a space and read it as a single word.

5. Remove redundancy by storing the word in the map data structure.

6. Proceed through files 1 to 4 in order.

7. 1 ~ 4 files Read the file once again, store the category number based on the words in the category, and store the number of documents per category.

Also, store how many words in a category are displayed based on words that have been previously saved. (However, even if the same word occurs multiple times in one category, it is assumed to be number 1.)

8. Use the numbers above to find the ky square value (select MAX value).

9. Sort the words in descending order based on each word.

10. Finally, it reads 5 test files, extracts the words from the training file, and sorts them based on the chi-squared values ​​of the words.

\*/

int main() {

map <string, double> feature;

map<string, int> multi\_feature;

map <string, int> number\_doc[8];

vector<string>::iterator new\_end;

vector<pair<double, string>> temp\_feature;

int total\_each\_category[8] = { 0, };

int temp\_each\_category[8] = { 0, };

double temp\_X2\_index[8] = { 0.0, };

int N = 0;

//Default file name

string file\_name = "HKIB-20000\_000.txt";

string doc = "@DOCUMENT";

string text = "#TEXT";

string category = "#CAT'03";

string KW = "<KW>";

int text\_flag = 0;

int category\_index = 0;

string temp\_category;

double A, B, C, D;

//Save the features

for (int i = 1; i < 5; i++) {

char buffer[10];

string temp\_line;

\_itoa(i, buffer, 10);

file\_name[13] = buffer[0];

//Read files

ifstream file(file\_name.data());

//Check the file

if (!file.is\_open()) printf("Not file open!!!\n");

//Read string of files

//Emty check

while (getline(file, temp\_line)) {

char \*temp = new char[temp\_line.size() + 1];

strcpy(temp, temp\_line.c\_str());

//Get the total number of each categorys

if (strstr(temp, "#CAT'03")) {

if (strstr(temp, "건강과 의학")) category\_index = 0;

else if (strstr(temp, "경제")) category\_index = 1;

else if (strstr(temp, "과학")) category\_index = 2;

else if (strstr(temp, "교육")) category\_index = 3;

else if (strstr(temp, "문화와 종교")) category\_index = 4;

else if (strstr(temp, "사회")) category\_index = 5;

else if (strstr(temp, "산업")) category\_index = 6;

else if (strstr(temp, "여가생활")) category\_index = 7;

total\_each\_category[category\_index]++;

}

//Find #TEXT

if (strstr(temp, "#TEXT")) {

//Initial flag

text\_flag = 1;

continue;

}

if (strstr(temp, "@DOCUMENT")) {

text\_flag = 0;

continue;

}

if (strstr(temp, "<KW>")) {

text\_flag = 0;

continue;

}

if (text\_flag == 1) {

//Case the empty

if (strlen(temp) == 0) continue;

//Change the specific word

for (int i = 0; i < strlen(temp); i++) {

if (temp[i] == '.' || temp[i] == '-' || temp[i] == ',' || temp[i] == '/'

|| temp[i] == '<' || temp[i] == '>' || temp[i] == '{' || temp[i] == '}'

|| temp[i] == '[' || temp[i] == ']' || temp[i] == '"' || temp[i] == '\*'

|| temp[i] == '(' || temp[i] == ')' || temp[i] == ':' || temp[i] == '!' || temp[i] == '?'

|| temp[i] == '@') {

temp[i] = ' ';

}

}

//Split the line and get the number of each category of feature

Tokenize(temp, feature, number\_doc, category\_index);

}

}

file.close();

}

//Count the document of ecah feature

for (int i = 1; i < 5; i++) {

char buffer[10];

string temp\_line;

\_itoa(i, buffer, 10);

file\_name[13] = buffer[0];

//Read files

ifstream file\_count(file\_name.data());

//Check the file

if (!file\_count.is\_open()) printf("Not file open!!!\n");

//Read string of files

//Emty check

while (getline(file\_count, temp\_line)) {

char \*temp = new char[temp\_line.size() + 1];

strcpy(temp, temp\_line.c\_str());

//Get the total number of each categorys

if (strstr(temp, "#CAT'03")) {

if (strstr(temp, "건강과 의학")) category\_index = 0;

else if (strstr(temp, "경제")) category\_index = 1;

else if (strstr(temp, "과학")) category\_index = 2;

else if (strstr(temp, "교육")) category\_index = 3;

else if (strstr(temp, "문화와 종교")) category\_index = 4;

else if (strstr(temp, "사회")) category\_index = 5;

else if (strstr(temp, "산업")) category\_index = 6;

else if (strstr(temp, "여가생활")) category\_index = 7;

}

//Find #TEXT

if (strstr(temp, "#TEXT")) {

//Initial flag

text\_flag = 1;

continue;

}

if (strstr(temp, "@DOCUMENT")) {

text\_flag = 0;

for (map<string, int>::iterator s = temp\_count.begin(); s != temp\_count.end(); s++) {

if (number\_doc[category\_index].count(s->first) > 0) number\_doc[category\_index].at(s->first)++;

}

//Clear the temp\_count

temp\_count.erase(temp\_count.begin(), temp\_count.end());

continue;

}

if (strstr(temp, "<KW>")) {

text\_flag = 0;

continue;

}

if (text\_flag == 1) {

//Case the empty

if (strlen(temp) == 0) continue;

//Change the specific word

for (int i = 0; i < strlen(temp); i++) {

if (temp[i] == '.' || temp[i] == '-' || temp[i] == ',' || temp[i] == '/'

|| temp[i] == '<' || temp[i] == '>' || temp[i] == '{' || temp[i] == '}'

|| temp[i] == '[' || temp[i] == ']' || temp[i] == '"' || temp[i] == '\*'

|| temp[i] == '(' || temp[i] == ')' || temp[i] == ':' || temp[i] == '!' || temp[i] == '?'

|| temp[i] == '@') {

temp[i] = ' ';

}

}

//Split the line and get the number of each category of feature

TokenizeC(temp, feature, number\_doc, category\_index);

}

}

file\_count.close();

}

//Get the totla number of documents

for (int d = 0; d < 7; d++) N += total\_each\_category[d];

int count = 0;

//Write file

ofstream ABCD\_FILE("ABCD.txt");

//Caculate the X2 statistic

for (map<string, double>::iterator s = feature.begin(); s != feature.end(); s++) {

double result = 0.0;

vector<double> temp\_result;

for (int k = 0; k < 8; k++) {

result = A = B = C = D = 0.0;

if (number\_doc[k].count(s->first) > 0) A = number\_doc[k].at(s->first);

for (int c = 0; c < 8; c++) {

if (k == c) continue;

if (number\_doc[c].count(s->first) > 0) C += number\_doc[c].at(s->first);

D += total\_each\_category[c];

}

B = total\_each\_category[k] - A;

D = D - C;

if ((A + B)\*(A + C)\*(B + D)\*(D + C) == 0)

result = 0.0;

else

result = ((double)N\*(pow(((A\*D) - (B\*C)), 2.0))) / ((A + B)\*(A + C)\*(B + D)\*(D + C));

temp\_result.push\_back(result);

//Write file ABCD

ABCD\_FILE << "CATE "<< k << " - " <<"A : " << A << " B : " << B << " C : " << C << " D : " << D << " feature : "<< s->first<<"\n";

}

//Sort the chi value

sort(temp\_result.begin(), temp\_result.end());

//Select the max chi value

s->second = temp\_result.back();

//Select the max chi value

temp\_feature.push\_back(pair<double, string>(s->second, s->first));

//Clear the temp

temp\_result.erase(temp\_result.begin(), temp\_result.end());

//Check count

count++;

}

ABCD\_FILE.close();

//Sort the feature by chi value

sort(temp\_feature.begin(), temp\_feature.end(), pairCompare);

count = 0;

//Write file

ofstream FEATURE\_INDEX("FEATURE\_INDEX.txt");

//Store the feature value

for (vector<pair<double,string>>::iterator it = temp\_feature.begin() ; it != temp\_feature.end(); it++){

multi\_feature.insert(pair<string, int>(it->second, count++));

FEATURE\_INDEX << "Feature : " << it->second << " INDEX : " << count << " CHI : " << it->first << "\n";

}

FEATURE\_INDEX.close();

//Write file

ofstream o\_FILE("training.txt");

int flag = 1;

category\_index = 0;

//Write the training file

for (int i = 1; i < 5; i++) {

char buffer[10];

string temp\_line;

\_itoa(i, buffer, 10);

file\_name[13] = buffer[0];

//Read files

ifstream FILE(file\_name.data());

//Check the file

if (!FILE.is\_open()) printf("Not file open!!!\n");

//Read string of files

//Emty check

while (getline(FILE, temp\_line)) {

char \*temp = new char[temp\_line.size() + 1];

strcpy(temp, temp\_line.c\_str());

//Get the total number of each categorys

if (strstr(temp, "#CAT'03")) {

if (strstr(temp, "건강과 의학")) category\_index = 0;

else if (strstr(temp, "경제")) category\_index = 1;

else if (strstr(temp, "과학")) category\_index = 2;

else if (strstr(temp, "교육")) category\_index = 3;

else if (strstr(temp, "문화와 종교")) category\_index = 4;

else if (strstr(temp, "사회")) category\_index = 5;

else if (strstr(temp, "산업")) category\_index = 6;

else if (strstr(temp, "여가생활")) category\_index = 7;

}

//Find #TEXT

if (strstr(temp, "#TEXT")) {

//Initial flag

text\_flag = 1;

continue;

}

if (strstr(temp, "@DOCUMENT")) {

text\_flag = 0;

if (flag == 1) {

flag = 0;

continue;

}

sort(temp\_X2.begin(), temp\_X2.end(), pairCompareR);

map<int, double>M;

//Stroe the sorted feature index and feature chi value by chi value

for (vector<pair<double, int>>::iterator it = temp\_X2.begin(); it != temp\_X2.end(); it++) {

M.insert(pair<int, double>(it->second, it->first));

}

//Write the category, index and chi value by format

o\_FILE << category\_index + 1 << " ";

for (map<int, double>::iterator it = M.begin(); it != M.end(); it++) {

o\_FILE << it->first + 1 << ":" << it->second << " ";

}

o\_FILE << "\n";

temp\_X2.erase(temp\_X2.begin(), temp\_X2.end());

continue;

}

if (strstr(temp, "<KW>")) {

text\_flag = 0;

continue;

}

if (text\_flag == 1) {

//Case the empty

if (strlen(temp) == 0) continue;

//Change the specific word

for (int i = 0; i < strlen(temp); i++) {

if (temp[i] == '.' || temp[i] == '-' || temp[i] == ',' || temp[i] == '/'

|| temp[i] == '<' || temp[i] == '>' || temp[i] == '{' || temp[i] == '}'

|| temp[i] == '[' || temp[i] == ']' || temp[i] == '"' || temp[i] == '\*'

|| temp[i] == '(' || temp[i] == ')' || temp[i] == ':' || temp[i] == '!' || temp[i] == '?'

|| temp[i] == '@') {

temp[i] = ' ';

}

}

//Write the X2 to training file

TokenizeR(temp, feature, multi\_feature);

}

}

FILE.close();

}

o\_FILE.close();

//Write test file

ofstream o\_FILE\_test("test.txt");

int flag = 1;

//Write the test file

for (int i = 5; i < 6; i++) {

char buffer[10];

string temp\_line;

\_itoa(i, buffer, 10);

file\_name[13] = buffer[0];

//Read files

ifstream FILE\_test(file\_name.data());

//Check the file

if (!FILE\_test.is\_open()) printf("Not file open!!!\n");

//Read string of files

//Emty check

while (getline(FILE\_test, temp\_line)) {

char \*temp = new char[temp\_line.size() + 1];

strcpy(temp, temp\_line.c\_str());

//Get the total number of each categorys

if (strstr(temp, "#CAT'03")) {

if (strstr(temp, "건강과 의학")) category\_index = 0;

else if (strstr(temp, "경제")) category\_index = 1;

else if (strstr(temp, "과학")) category\_index = 2;

else if (strstr(temp, "교육")) category\_index = 3;

else if (strstr(temp, "문화와 종교")) category\_index = 4;

else if (strstr(temp, "사회")) category\_index = 5;

else if (strstr(temp, "산업")) category\_index = 6;

else if (strstr(temp, "여가생활")) category\_index = 7;

}

//Find #TEXT

if (strstr(temp, "#TEXT")) {

//Initial flag

text\_flag = 1;

continue;

}

if (strstr(temp, "@DOCUMENT")) {

text\_flag = 0;

if (flag == 1) {

flag = 0;

continue;

}

sort(temp\_X2.begin(), temp\_X2.end(), pairCompareR);

map<int, double>M;

//Stroe the sorted feature index and feature chi value by chi value

for (vector<pair<double, int>>::iterator it = temp\_X2.begin(); it != temp\_X2.end(); it++) {

M.insert(pair<int, double>(it->second, it->first));

}

//Write the category, index and chi value by format

o\_FILE\_test << category\_index + 1 << " ";

for (map<int, double>::iterator it = M.begin(); it != M.end(); it++) {

o\_FILE\_test << it->first + 1 << ":" << it->second << " ";

}

o\_FILE\_test << "\n";

temp\_X2.erase(temp\_X2.begin(), temp\_X2.end());

continue;

}

if (strstr(temp, "<KW>")) {

text\_flag = 0;

continue;

}

if (text\_flag == 1) {

//Case the empty

if (strlen(temp) == 0) continue;

//Change the specific word

for (int i = 0; i < strlen(temp); i++) {

if (temp[i] == '.' || temp[i] == '-' || temp[i] == ',' || temp[i] == '/'

|| temp[i] == '<' || temp[i] == '>' || temp[i] == '{' || temp[i] == '}'

|| temp[i] == '[' || temp[i] == ']' || temp[i] == '"' || temp[i] == '\*'

|| temp[i] == '(' || temp[i] == ')' || temp[i] == ':' || temp[i] == '!' || temp[i] == '?'

|| temp[i] == '@') {

temp[i] = ' ';

}

}

//Write the X2 to training file

TokenizeR(temp, feature, multi\_feature);

}

}

FILE\_test.close();

}

o\_FILE\_test.close();

return 0;

}