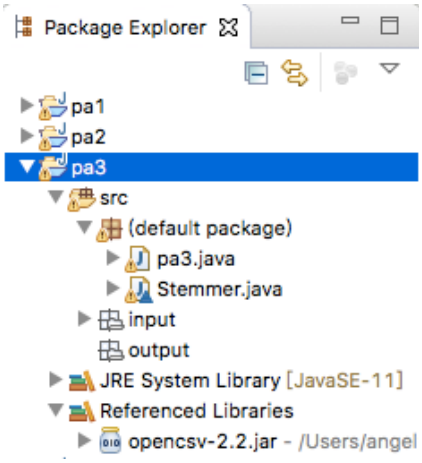
1. 執行環境：Eclipse 2019-06

2. 程式語言：Java (version: 1.8.0_101)

3. 執行方式：（＊有提供助教整個 pa3 java project）



(1) 新增名為 pa3 的 java project，在 src 下建立新的 package 及 pa3 class (pa3.java)

(2) 在 src 之下新增名為 “input” 的資料夾，將下載的 1095 個文件複製進去，供後續讀取檔案用

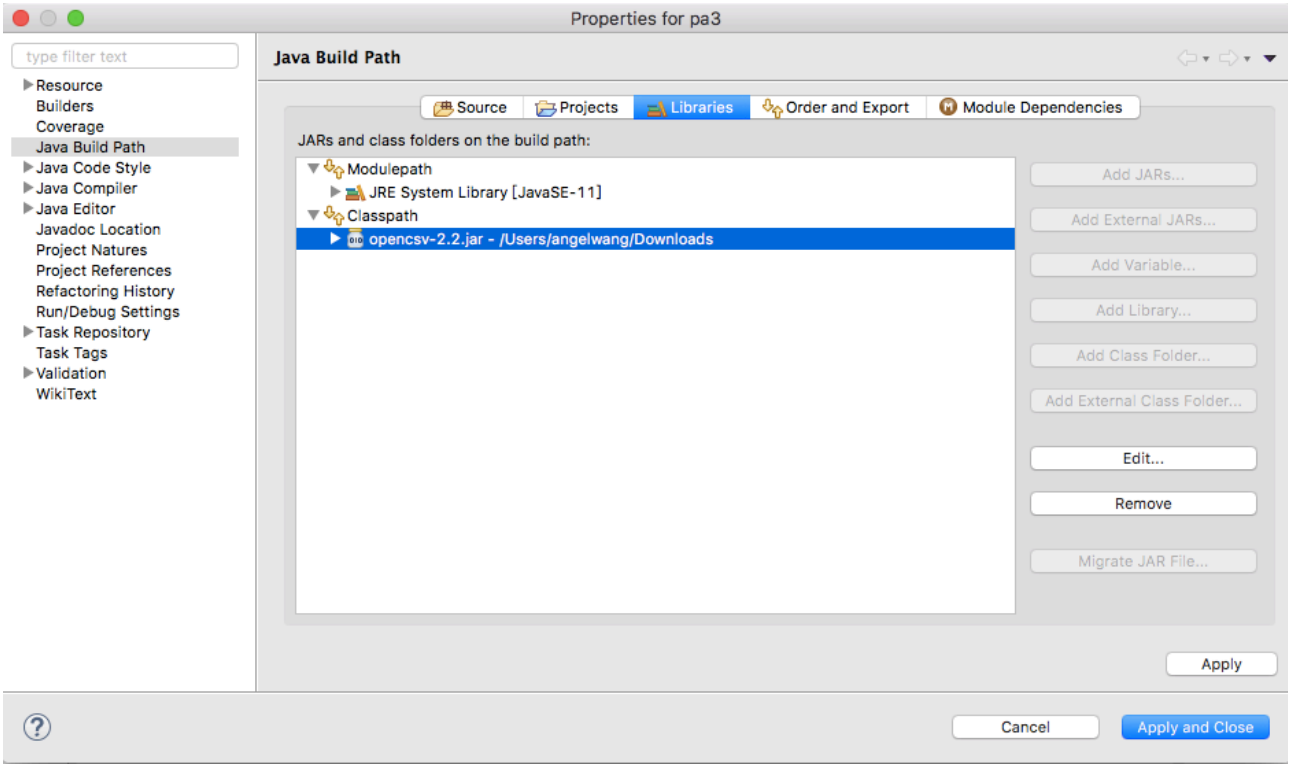
(3) 在前述名為 “input” 的資料夾新增名為 “training.txt” 的檔案，將 training documents 的 docID 貼進去（來源：https://ceiba.ntu.edu.tw/course/88ca22/content/training.txt）

(4) 在 src 之下新增名為 “output” 的資料夾，輸出各篇文章的分類結果（sub.csv）

(5) 在 default package 新增一個名為 “Stemmer” 的 class，貼入 Porter’s Stemmer 的 code（來源：https://tartarus.org/martin/PorterStemmer/java.txt）

(6) 為了讀取及寫入 txt 檔案，有 import java.io 的部分套件（同 PA1 及 PA2）

(7) 為了輸出成 csv 檔有 import au.com.bytecode.opencsv.CSVWriter，需在 project 中加入 jar 檔（有提供給助教，但可能需要手動加入），如下圖所示：



＊因為 pa3.java 要在 project 中執行，所以 input 和 output 的路徑設定為 src/input 及 src/output

4. 作業處理邏輯說明：

(1) 前置作業

a. 讀入 training.txt 的內容

前面執行方式有提及在 src 之下新建了 input 的資料夾，並建立 training.txt 將老師提供之 training documents 的 docID 貼進去，讀入方式和 PA1、PA2 一致，有 import 部分 java.io 套件來讀取檔案。

Code

```
/**
 * Read src/input/training.txt
 * @return an ArrayList
 * @throws Exception
 */
public static ArrayList<String> readTrainingFile() throws Exception {
    ArrayList<String> a = new ArrayList<String>();
    String pathname = "src/input/training.txt";
    File filename = new File(pathname);
    InputStreamReader reader = new InputStreamReader(new FileInputStream(filename));
    BufferedReader br = new BufferedReader(reader);
    String line = br.readLine();
    a.add(line);
    while (line != null) {
        line = br.readLine();
        if (line != null) {
            a.add(line);
        } else {
            br.close();
        }
    }
    return a;
}
```

b. 建立 trainingList

在 main method 中呼叫 createtrainingList(readTrainingFile()) method ，將剛才所讀入的 training.txt 依照 docID 存進 ArrayList<ArrayList<String>> 中。過程是先找出所有 splitter（這裡是空格）的 index，除了第一個 docID 以外，每個 docID 的開頭在 splitter 之後，而包括第一個及最後一個在內的每個 docID 的結尾都在 splitter 之前，找出每個 docID 的開頭和結尾 index 就能用迴圈的方式取出 docID，並存入 trainingList 中。

Code

```
/**
 * Create trainingList from training.txt
 * @param a
 */
public static void createtrainingList(ArrayList<String> a) {
    char split = ' ';
    trainingList = new ArrayList<ArrayList<String>>();
    for(int i = 0; i < a.size(); i++) {
        ArrayList<String> element = new ArrayList<String>();
        /* 找出 splitter 的 index */
        ArrayList<Integer> splitIndex = new ArrayList<Integer>();
        String str = a.get(i);
        for(int j = 0; j < str.length(); j++) {
            if(str.charAt(j) == split) {
                splitIndex.add(j);
            }
        }
```

```
        /* 找出每個詞的開頭 index */
        ArrayList<Integer> starIndex = new ArrayList<Integer>();
        starIndex.add(0);
        for(int j = 0; j < splitIndex.size()-1; j++) {
            int star = splitIndex.get(j) + 1;
            starIndex.add(star);
        }
        /* 找出每個詞的結尾 index */
        ArrayList<Integer> endIndex = new ArrayList<Integer>();
        for(int j = 0; j < splitIndex.size(); j++) {
            int end = splitIndex.get(j) - 1;
            endIndex.add(end);
        }
        /* create trainingList element */
        for (int j = 0; j < starIndex.size(); j++) {
            int star = starIndex.get(j);
            int end = endIndex.get(j);
            String id = "";
            for (int k = star; k <= end; k++) {
                id += str.charAt(k) ;
            }
            element.add(id);
        }
        trainingList.add(element);
    }
    /* check trainingList */
    System.out.println("trainingList: ");
    for (int i = 0; i < trainingList.size(); i++) {
        System.out.println(trainingList.get(i));
    }
}
```

c. Method：trainingDoc()

延用 PA2 的 code，分別建立 trainTokensList、mergedTrainTokensList、irtm_dict 三個 ArrayList<ArrayList<String>>。在建立 trainTokensList 的過程中會呼叫 method doctTokens(...)，讀入相對應的文章並將單篇文章的所有 tokens 取出，同 PA1、PA2 所做的前處理、stemming 及剔除 stopwords（這次額外剔除了 know, ye, yet, alex, john, lisa, hous, kill, offic，因為不具代表性或是被超過兩個 class 選為 term），再存入參數 tokenList 中（這裡的參數是 trainTokensList）。在 trainingDoc() 的 method 中會取出 training set 中所有文章的 tokens，再對 trainTokensList 依照 tokens 的字母做排序，並將同篇文章相同的 term 合併，trainTokensList 原有的 term、docID 及 classID 皆存入 mergedTrainTokensList，最後同 PA2 的方式去計算每個 term 的 document frequency，建立 irtm_dict。

Code

```
public static void trainingDoc() throws Exception {
    // terms & docID & classID
    trainTokensList = new ArrayList<ArrayList<String>>();
    for (int i = 0; i < trainingList.size(); i++) {
        for (int j = 1; j < trainingList.get(i).size(); j++) {
            String docID = trainingList.get(i).get(j);
            doctTokens(Integer.parseInt(docID), trainingList.get(i).get(0), trainTokensList);
        }
    }
    Collections.sort(trainTokensList, new Comparator<ArrayList<String>>() {
        @Override
        public int compare(ArrayList<String> o1, ArrayList<String> o2) {
            return o1.get(0).compareTo(o2.get(0));
        }
    });
```

```
    // merge same term from the same document
    mergedTrainTokensList = new ArrayList<ArrayList<String>>();
    mergedTrainTokensList.add(trainTokensList.get(0));
    for(int i = 1; i < trainTokensList.size(); i++) {
        int pre = i-1;
        if(!trainTokensList.get(i).equals(trainTokensList.get(pre))) {
            mergedTrainTokensList.add(trainTokensList.get(i));
        }
    }
    // t_index & term & df
    irtm_dict = new ArrayList<ArrayList<String>>();
    int t_index = 1;
    ArrayList<String> firstEle = new ArrayList<String>();
    firstEle.add(Integer.toString(t_index));
    firstEle.add(mergedTrainTokensList.get(0).get(0));
    firstEle.add("1");
    irtm_dict.add(firstEle);
    for(int i = 1; i < mergedTrainTokensList.size(); i++) {
        int pre = i-1;
        ArrayList<String> element = new ArrayList<String>();
        if(!mergedTrainTokensList.get(i).get(0).equals(mergedTrainTokensList.get(pre).get(0))) {
            t_index++;
            element.add(Integer.toString(t_index));
            element.add(mergedTrainTokensList.get(i).get(0));
            element.add("1");
            irtm_dict.add(element);
        } else {
            int count = Integer.parseInt(irtm_dict.get(irtm_dict.size()-1).get(2));
            count++;
            irtm_dict.get(irtm_dict.size()-1).set(2, Integer.toString(count));
        }
    }
}
```

(2) Feature Selection

a. 在 main method 中的步驟：

除了前述的前置作業外，有呼叫 createN11Lists() method 來建立各個 term 在各個 class 中之 n11 的 list，並存入 ArrayList<ArrayList<String>> n11Lists 中供後續計算使用。接著呼叫另一個 method feature() 來建立 feature selection 後的的字典。

Code

```
// Feature Selection
createtrainingList(readTrainingFile());
trainingDoc();
n11Lists = new ArrayList<ArrayList<String>>();
createN11Lists();

// Dictionary
dictionary = new ArrayList<String>();
feature();
System.out.println("Dictionary(" + dictionary.size() + ")" + dictionary);
```

b. 計算某個 class 中所有 term 的 n11

首先取出 mergedTrainTokensList 中所有 classID 同參數者，暫存進在此 method 中新建的 tokensList，接著以在計算 irtm_dict 中的 df 時相同的方式來計算在某個 class 中各個 term 的 df，暫存進名為 df 的 list 中。由於前面從 mergedTrainTokensList 所取出的 tokens 僅為這個 class 所用到的 term，不一定包含所有 irtm_dict 中的 term，因此另外以 irtm_dict 中所有 term 建立了一個名為 n11 的 ArrayList<ArrayList<String>>，再將剛才所算出各個 term 的 df 依照 term 存入 n11 中，並回傳 n11 這個 ArrayList<ArrayList<String>>。

Code

```
public static ArrayList<ArrayList<String>> n11(String classID) {
    // terms & docID & classID for each class
    ArrayList<ArrayList<String>> tokensList = new ArrayList<ArrayList<String>>();
    for (int i = 0; i < mergedTrainTokensList.size(); i++) {
        if (mergedTrainTokensList.get(i).get(2).equals(classID)) {
            tokensList.add(mergedTrainTokensList.get(i));
        }
    }
    // terms & df for each class (n11)
    ArrayList<ArrayList<String>> df = new ArrayList<ArrayList<String>>();
    ArrayList<String> firstEle = new ArrayList<String>();
    firstEle.add(tokensList.get(0).get(0));
    firstEle.add("1");
    df.add(firstEle);
    for(int i = 1; i < tokensList.size(); i++) {
        int pre = i-1;
        ArrayList<String> element = new ArrayList<String>();
        if(!tokensList.get(i).get(0).equals(tokensList.get(pre).get(0))) {
            element.add(tokensList.get(i).get(0));
            element.add("1");
            df.add(element);
        } else {
            int count = Integer.parseInt(df.get(df.size()-1).get(1));
            count++;
            df.get(df.size()-1).set(1, Integer.toString(count));
        }
    }
    // term & n11
    ArrayList<ArrayList<String>> n11 = new ArrayList<ArrayList<String>>();
    for (int i = 0; i < irtm_dict.size(); i++) {
            ArrayList<String> element = new ArrayList<String>();
            element.add(irtm_dict.get(i).get(1));
            element.add("0");
            n11.add(element);
    }
    for (int i = 0; i < n11.size(); i++) {
        for (int j = 0; j < df.size(); j++) {
            if (n11.get(i).get(0).equals(df.get(j).get(0))) {
                n11.get(i).set(1, df.get(j).get(1));
            }
        }
    }
    return n11;
}
```

c. Method：createN11Lists()

接著透過剛才建立的 n11(classID) method，利用 for 迴圈將各個 class 所有 term 的 n11 整合在一個 ArrayList<ArrayList<String>> 中，供後續取用。

Code

```
/**
 * (2) 把各個class的n11併成 n11Lists (term & class1's n11 &...& class13's n11)
 */
public static void createN11Lists() {
    for (int i = 1; i <= classNum; i++) {
        ArrayList<ArrayList<String>> list = n11(Integer.toString(i));
        for (int j = 0; j < list.size(); j++) {
            if (i == 1) {
                ArrayList<String> element = new ArrayList<String>();
                element.add(list.get(j).get(0));
                element.add(list.get(j).get(1));
                for (int k = 1; k <= classNum-1; k++) {
                    element.add("0");
                }
                n11Lists.add(element);
            } else {
                n11Lists.get(j).set(i, list.get(j).get(1));
            }
        }
    }
}
```

d. Method：LLR(int classID)

利用剛才所建立的 n11Lists 取出相對應的 n11，並計算其他如 n10、n01、n00 等值，且對這四個值做 add one smoothing，避免在後續計算 log 時出現 log0 的情況。接著計算某個 class 中所有 term 的 LLR，先依照 LLR 的大小做排序（大到小），再取出前 42 高且 LLR 大於 5 的 term 當作字典，回傳這 42 個 term（取 42 是為了讓最終的字典接近 500 個 term）。

Code

```
public static ArrayList<String> LLR(int classID) {
    ArrayList<ArrayList<String>> term_LLR = new ArrayList<ArrayList<String>>();
    for (int i = 0; i < n11Lists.size(); i++) {
        ArrayList<String> element = new ArrayList<String>();
        ArrayList<String> n11List = n11Lists.get(i);
        element.add(n11List.get(0));
        double n11 = Integer.parseInt(n11List.get(classID))+1.0;
        double df = Integer.parseInt(irtm_dict.get(i).get(2))+2.0;
        double pt = df/(trainDocNum + 4);
        double n10 = (15.0 + 2.0) - n11;
        double n01 = df - n11;
        double n00 = (trainDocNum + 4) - (15+2) - n01;
        double p1 = (n11/(n11+n10));
        double p2 = (n01/(n01+n00));
        double LLR = -2*((n11+n01)*Math.log(pt) + (n10+n00)*Math.log((1-pt)) - n11*Math.log(p1) - n10*Math.log(1-p1) - n01*Math.log(p2) -n00*Math.log(1-p2)*1.0);
        LLR =  Math.round(LLR*100.0)/100.0;
        element.add(Double.toString(LLR));
        term_LLR.add(element);
    }
    // 按照 LLR 大小排序
    Collections.sort(term_LLR, new Comparator<ArrayList<String>>() {
        @Override
        public int compare(ArrayList<String> o1, ArrayList<String> o2) {
            return -o1.get(1).compareTo(o2.get(1));
        }
    });
    // top 40 term
    ArrayList<String> top40LLR = new ArrayList<String>();
    for (int i = 0; i < 42; i++) {// 取前40個
        top40LLR.add(term_LLR.get(i).get(0));
    }
    for (int i = 0; i < 42; i++) {// 取前40個
        ArrayList<String> element = term_LLR.get(i);
        if (Double.parseDouble(element.get(1)) > 5) {
            top40LLR.add(element.get(0));
        }
    }
    return top40LLR;
}
```

e. Method：feature()

呼叫 13 次前述的 LLR(int classID) method，將各個 class 的前 42 個 term 暫存進此 method 新建立的 dictionary520，將這些 term 依照字母做排序，透過和前一個 term 比較的方式刪除重複的 term，最終建立將近 500 個 term 的 dictionary。

Code

```
/**
 * (4) 建立字典
 */
public static void feature() {
    // term
    ArrayList<String> dictionary520 = new ArrayList<String>();
    for (int i = 1; i <= classNum; i++) {
        for (int j = 0; j < LLR(i).size(); j++) {
            dictionary520.add(LLR(i).get(j));
        }
    }
    Collections.sort(dictionary520);
    dictionary.add(dictionary520.get(0));
    for(int i = 1; i < dictionary520.size(); i++) {
        int pre = i-1;
        if(!dictionary520.get(i).equals(dictionary520.get(pre))) {
            dictionary.add(dictionary520.get(i));
        }
    }
}
```

(3) Training Phase

a. 在 main method 中的步驟：

建立了名為 condprobAllClass 的 ArrayList<ArrayList<String>> 來儲存字典中所有 term 在各個 class 的 log(condprob(t,c))，且計算過程全都包在同名的 method 中。

Code

```
// Training
condprobAllClass = new ArrayList<ArrayList<String>>();
condprobAllClass();
```

b. Method：countTokensofTerm(int classID)

Code

```
/**
 * (1) 計算某個class中所有文章各個term的出現總次數
 * @param classID
 * @return
 */
public static ArrayList<ArrayList<String>> countTokensofTerm(int classID) {
    // 從 trainTokensList 中取出 classID 相同者
    ArrayList<ArrayList<String>> list = new ArrayList<ArrayList<String>>();
    for (int i = 0; i < trainTokensList.size(); i++) {
        if(trainTokensList.get(i).get(2).equals(Integer.toString(classID))) {
            ArrayList<String> element = new ArrayList<String>();
            element.add(trainTokensList.get(i).get(0));
            element.add(trainTokensList.get(i).get(2));
            list.add(element);
        }
    }
```

```
    // terms 在 class 中出現次數 (term & 次數)
    ArrayList<ArrayList<String>> term_f = new ArrayList<ArrayList<String>>();
    ArrayList<String> element1 = new ArrayList<String>();
    element1.add(list.get(0).get(0));
    element1.add("1");
    term_f.add(element1);
    for(int i = 1; i < list.size(); i++) {
        int pre = i-1;
        ArrayList<String> element = new ArrayList<String>();
        if(!list.get(i).equals(list.get(pre))) {
            element.add(list.get(i).get(0));
            element.add("1");
            term_f.add(element);
        } else{
            int count = Integer.parseInt(term_f.get(term_f.size()-1).get(1));
            count++;
            term_f.get(term_f.size()-1).set(1, Integer.toString(count));
        }
    }
    return term_f;
}
```

首先建立以 class ID 為 input parameter 的 method，從先前建立的 trainTokensList 中取出所有 classID 和參數相同者，將 token 及 classID 暫存進 list 中。接著計算各個 token 在此 class 中的出現次數，將 token 及次數存進名為 term_f 的 ArrayList<ArrayList<String>> 並回傳。由於此 trainTokensList 是依照 token 字母做排序，因此相同的字會排在一起，計算次數時可以與前者比較是否為相同的字。

c. Method：countTokensofTermV(int classID)

依照字典中的 term 建立一個新的 ArrayList<ArrayList<String>>，將前面步驟所算出各個 term 的出現次數存進此二維的 ArrayList 中，並在此處做 add one smoothing，因此 class 中沒用到的 term 出現次數會是 1。

Code

```
/**
 * (2) 計算某個class中所有文章各個term的出現總次數，只留下有在 vocabulary 的 term (Add 1 smoothing)
 * @param classID
 * @return
 */
public static ArrayList<ArrayList<String>> countTokensofTermV(int classID) {
    // term & 次數
    ArrayList<ArrayList<String>> termV_f = new ArrayList<ArrayList<String>>();
    for (int i = 0; i < dictionary.size(); i++) {
        String term = dictionary.get(i);
        ArrayList<String> element = new ArrayList<String>();
        element.add(term);
        element.add("1");
        ArrayList<ArrayList<String>> list = countTokensofTerm(classID);
        for (int j = 0; j < list.size(); j++) {
            if (list.get(j).get(0).equals(term)) {
                int count = Integer.parseInt(list.get(j).get(1))+1;
                element.set(1, Integer.toString(count));
            }
        }
        termV_f.add(element);
    }
    return termV_f;
}
```

d. Method：condprob(int classID)

此 method 用來計算某個 class 中所有 term 的 log(condprob)，首先透過剛才提及的 method countTokensofTermV(int classID) 將同一個 class 的 Tct 做加總，作為 condprob 的分母，再從中取出各個 Tct 值去計算 condprob，並在此處先取 log 以後再存進 list 中。

Code

```
/**
 * (3) 某個 class 中每個 term 的 condprob
 * @param classID
 * @return
 */
public static ArrayList<ArrayList<String>> condprob(int classID) {
    ArrayList<ArrayList<String>> tct = countTokensofTermV(classID); // term & 次數
    // term & probability
    ArrayList<ArrayList<String>> list = new ArrayList<ArrayList<String>>();
    // 計算 tct 加總
    double sum = 0;
    for (int i = 0; i < tct.size(); i++) {
        sum += Integer.parseInt(tct.get(i).get(1));
    }
    // 算 probability
    for (int i = 0; i < tct.size(); i++) {
        ArrayList<String> element = new ArrayList<String>();
        element.add(tct.get(i).get(0)); // term
        double prob = Integer.parseInt(tct.get(i).get(1))*1.0/sum;
        element.add(Double.toString(Math.round(Math.log(prob)*100.0)/100.0));
        list.add(element);
    }
    return list;
}
```

e. Method：condprobAllClass()

最終將各個 class 的 log(condprob) 存在同一個 ArrayList<ArrayList<String>>，包含了 term 以及 13 個 class 的 log(condprob)，供 testing phase 做使用。

Code

```
/* Training Phase */

/**
 * (4) 將各個class的condprob整合成一個二維list condprobAllClass
 */
public static void condprobAllClass() {
    // term & probability 1 & probability 2 & ...& probability 13
    for (int i = 1; i <= classNum; i++) {
        ArrayList<ArrayList<String>> list = condprob(i);
        for (int j = 0; j < list.size(); j++) {
            if (i == 1) {
                ArrayList<String> element = new ArrayList<String>();
                element.add(list.get(j).get(0));
                element.add(list.get(j).get(1));
                for (int k = 1; k <= classNum-1; k++) {
                    element.add("0");
                }
                condprobAllClass.add(element);
            } else {
                condprobAllClass.get(j).set(i, list.get(j).get(1));
            }
        }
    }
}
```

(4) Testing Phase

a. 在 main method 中的步驟：

首先建立名為 testingResult 的 ArrayList<ArrayList<String>> 來儲存分類結果，用 for 迴圈的方式找出屬於 testing set 的 docID（testingList 為先前建立用來儲存所有 testing set 的 docID 的 ArrayList<Integer>），分類過程主要在 testingResult(int docID) 的 method 中，接著取出分類結果存進 testingResult。最後寫出成 sub.csv 檔。

Code

```
// Testing
testingResult = new ArrayList<ArrayList<String>>();
ArrayList<String> firstElement = new ArrayList<String>();
firstElement.add("Id");
firstElement.add("Value");
testingResult.add(firstElement);
for (int i = 0; i < testingList.size(); i++) {
    ArrayList<String> element = new ArrayList<String>();
    int docID = testingList.get(i);
    element.add(Integer.toString(docID));
    int result = testingResult(docID);
    element.add(Integer.toString(result));
    testingResult.add(element);
    System.out.print(element.get(1));
}
write(testingResult);
```

b. Method：testingResult(int docID)

處理方式為根據各篇文章的 docID 去算各個 class 的分數，13 個分數都儲存在這個 method 下建立的 ArrayList<Double> score 中，再找出分數最大者的 classID，作為回傳的分類結果。

Code

```
/* Testing Phase */
/**
 * (2) 對文章做分類
 * @param docID
 * @return
 * @throws Exception
 */
public static int testingResult(int docID) throws Exception {
    // 各個 class 的分數
    ArrayList<Double> score = new ArrayList<Double>(); // size = 13
    ArrayList<String> tokens = testingDoc(docID);
    System.out.println("Tokens:" + tokens.size() + ", " + tokens);
    for (int i = 1; i <= classNum; i++) {
        double sum = 0; // 原本應該要是 logPrior 但因為大家都一樣就省略
        // 每個 token 找出 condprob 算分數
        for (int j = 0; j < tokens.size(); j++) {
            for (int k = 0; k < condprobAllClass.size(); k++) {
                if (tokens.get(j).equals(condprobAllClass.get(k).get(0))){
                    double condprob = Double.parseDouble(condprobAllClass.get(k).get(i));
                    sum += condprob;
                }
            }
        }
        score.add(sum);
    }
    System.out.println("Score:" + score);
```

```
    // 找出 13 個 score 中最大者
    double max = score.get(0);
    int classID = 1;
    for (int i = 1; i < score.size(); i++) {
        if (score.get(i) > max) {
            max = score.get(i);
            classID = i+1;
        }
    }
    System.out.println("Result: " +  classID);
    return classID;
}
```

計算分數是根據先前 Training Phase 中已建立的 ArrayList<ArrayList<String>> condprobAllClass 來找出相對應的 log(condprob(t,c))，由於每個 class 都會加上一樣的 log(prior)，因此在計算過程中省略不加，避免運算過程太耗時。而 testingDoc(docID) 的 method 和 PA1、PA2 類似，是讀取 IRTM 文件並對 tokens 做前處理，比較特別的是最終有以 for 迴圈的方式和 feature selection 後的字典做比對，只留下字典中有出現的 tokens，這部分就不再截圖呈現。

c. 輸出 sub.csv

如同前面「3. 執行方式」所述，輸出需匯入 openCSV 的 jar 檔，這裡的 input parameter 為 main method 中建立的 testingResult，欄位包含 Id 及 Value，須先將 ArrayList<String> 轉存為 String[] 並以逗號作為分欄標準。

Code

```
/**
 * Write to csv
 * @param n
 * @throws Exception
 */
public static void write(ArrayList<ArrayList<String>> result) throws Exception {
    CSVWriter writer = new CSVWriter(new FileWriter("src/output/sub.csv"), ',');
    for (int i = 0; i < result.size(); i++) {
        ArrayList<String> list = result.get(i);
        String[] line = {list.get(0),list.get(1)};
        writer.writeNext(line);
    }
    writer.close();
}
```

2022.1.2