Intelligent Web System Assignment Two

1. My name and student number

Name: Ling Liu

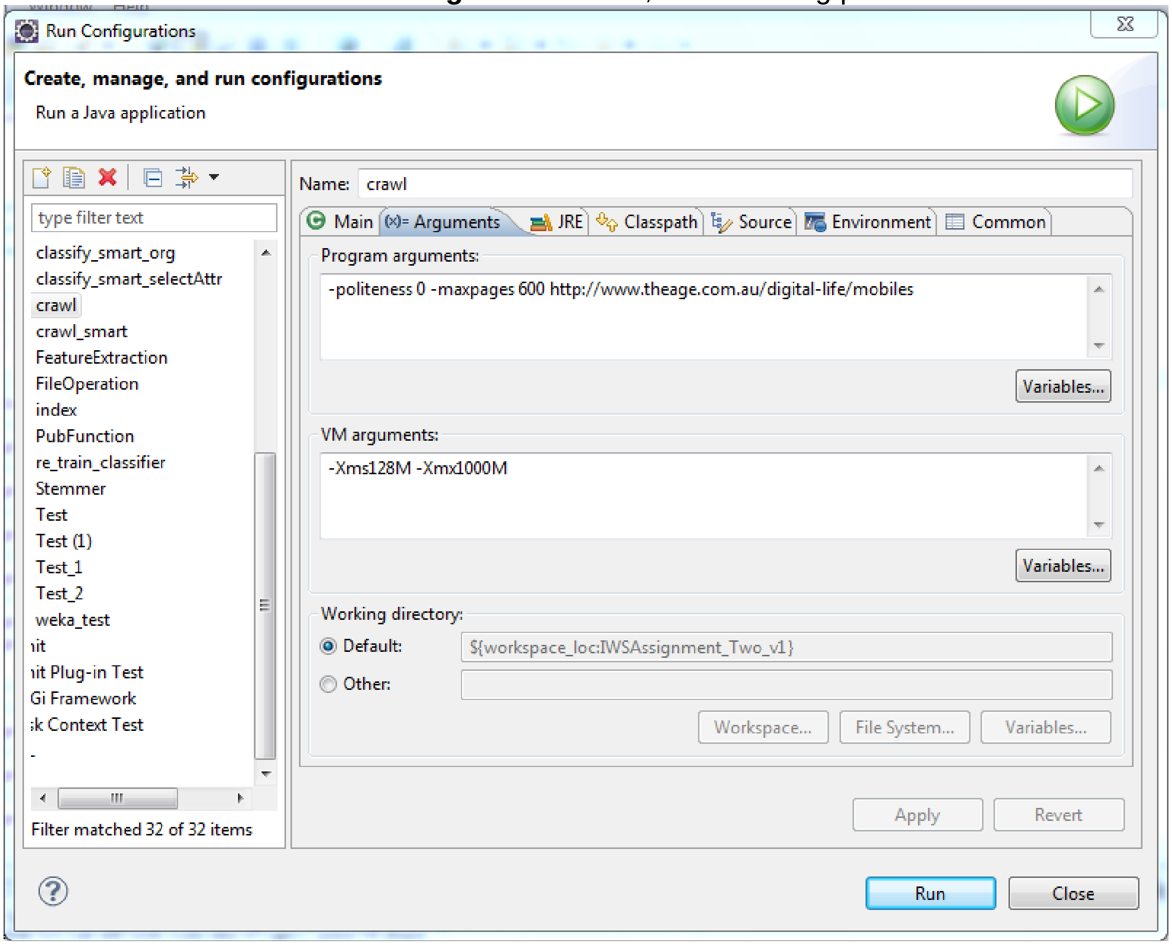
Student Number: s3300154

2. Operating instructions: a short paragraph giving clear instructions on how to compile and run your code

There are two ways to run this project:

Method one: Using Eclipse

- Create a new **JPA** project from Eclipse
- Copy all the **java** files, **stoplist_sorted** file and **directories** into this project.
- Add **jsoup-1.6.2.jar** and **weka.jar** into project.
- Run **crawl.java(fetch pages from source website)** as below(have to set running parameters): Please click **Run->Run Configuration** menu, add following parameters:



Run Configurations

Create, manage, and run configurations

Run a Java application

Name: crawl

Program arguments:

```
-politeness 0 -maxpages 600 http://www.theage.com.au/digital-life/mobiles
```

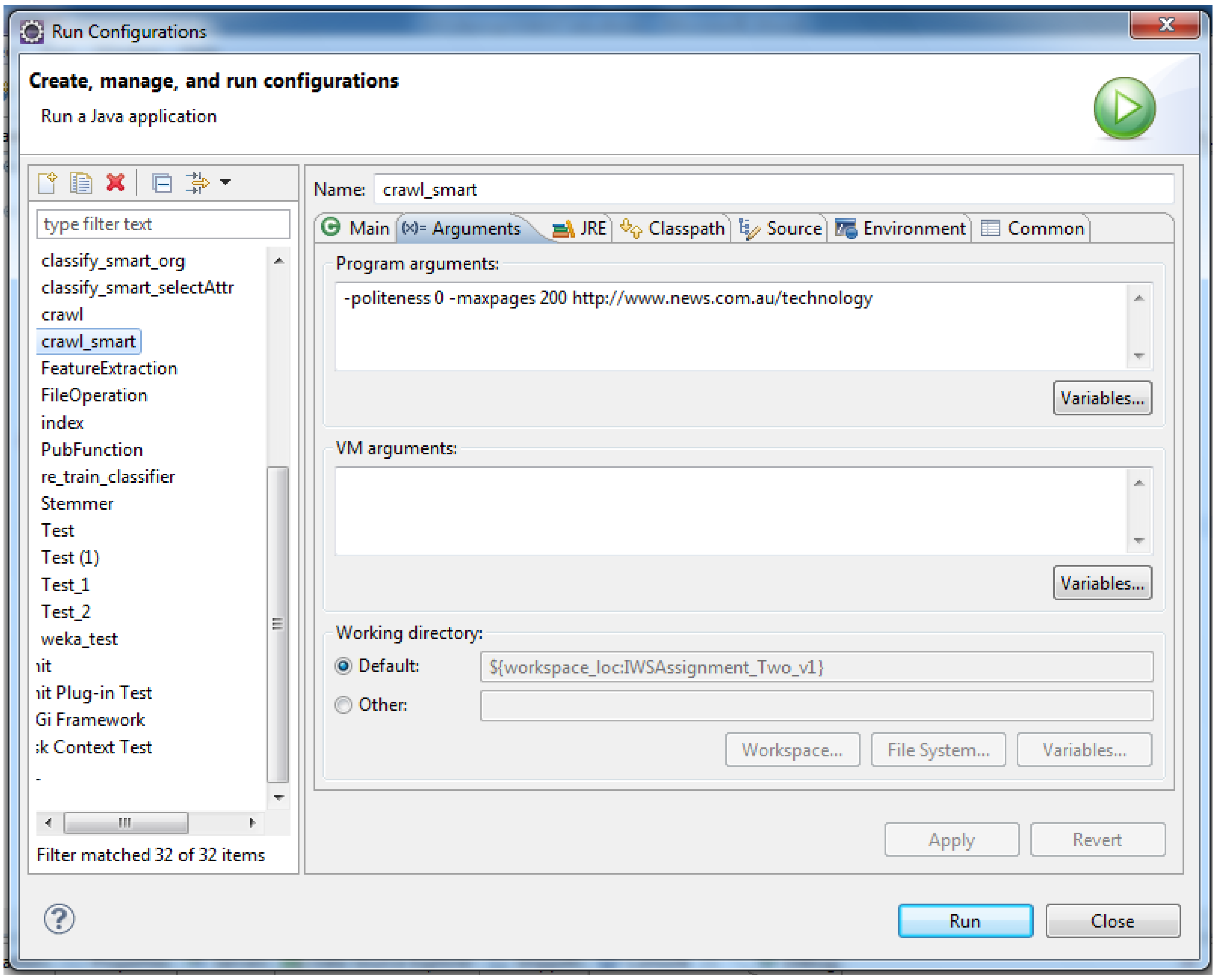
VM arguments:

```
-Xms128M -Xmx1000M
```

Working directory:
[✓ Default:] ${workspace_loc:IWSAssignment_Two_v1}
Other:

Then just click **Run** button.

- Run **index.java(generate term inverted list and training data set)**, just click **Run** button
- Run **crawl_smart.java(fetch pages from new websites)** as below(have to set running parameters):
  Please click **Run->Run Configuration** menu, add following parameters:



Name: crawl_smart

classify_smart_org
classify_smart_selectAttr
crawl
crawl_smart
FeatureExtraction
FileOperation
index
PubFunction
re_train_classifier
Stemmer
Test
Test (1)
Test_1
Test_2
weka_test
ıit
ıit Plug-in Test
Gi Framework
:k Context Test
-

Filter matched 32 of 32 items

Program arguments:

```
-politeness 0 -maxpages 200 http://www.news.com.au/technology
```

VM arguments:

Working directory:
Default: ${workspace_loc:IWSAssignment_Two_v1}
Other:

Then just click **Run** button.

- Run **classify_smart_J48.java(generate testing data set, apply classifier to testing data set and classify new instances)**, just click **Run** button. Then you can look at **retrain/classified_doc_J48** file which gives the classified results.

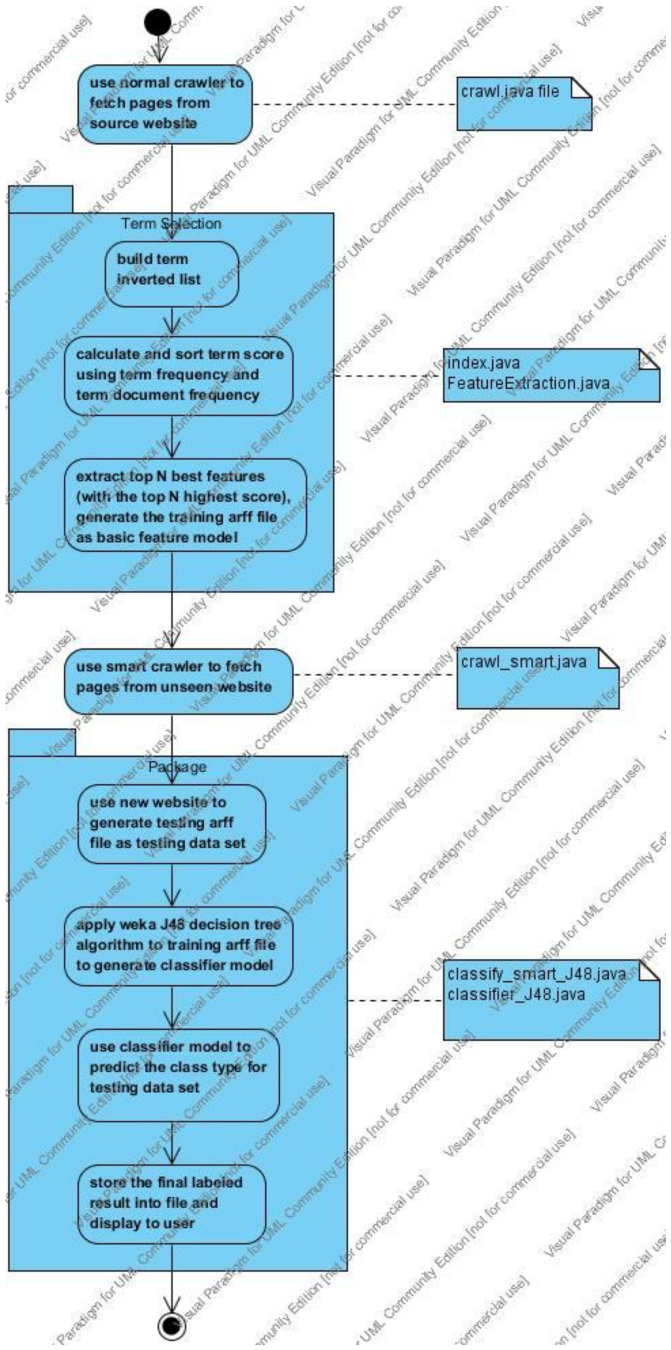
- Open the **retrain/classified_doc_J48** file to mark the result (**please look at this part at current Report: How user feedback (if any) is used to re-train the classifier?**) and save modified result.
- Run **re_train_classifier.java,** just click **Run** button. Then you can look at final improved result in **retrain/ classified_doc_J48_improve** file.

Method two: Using command line on yallara.

- Copy all **the java files, stoplist_sorted file, jsoup-1.6.2.jar,weka.jar,all empty directories and iws_script** into directory which uses for test on yallara.
- Run this command: **chmod +x iws_script**
- Run script: **./iws_script**
  Note: if you want to set any parameters please modify the content of **iws_script** file.
- Open the **retrain/classified_doc_J48** file to mark the result (**please look at this part at current Report: How user feedback (if any) is used to re-train the classifier?**) and save modified result.
- Run command: **java -classpath "jsoup-1.6.2.jar:weka.jar:." re_train_classifier**
  Then you can look at final improved result in **retrain/ classified_doc_J48_improve** file

3. Implementation

The following picture shows the whole structure of my project:



use normal crawler to fetch pages from source website
crawl.java file
Term Selection
build term inverted list
calculate and sort term score using term frequency and term document frequency
index.java
FeatureExtraction.java
extract top N best features (with the top N highest score), generate the training arff file as basic feature model
use smart crawler to fetch pages from unseen website
crawl_smart.java
Package
use new website to generate testing arff file as testing data set
apply weka J48 decision tree algorithm to training arff file to generate classifier model
classify_smart_J48.java
classifier_J48.java
use classifier model to predict the class type for testing data set
store the final labeled result into file and display to user

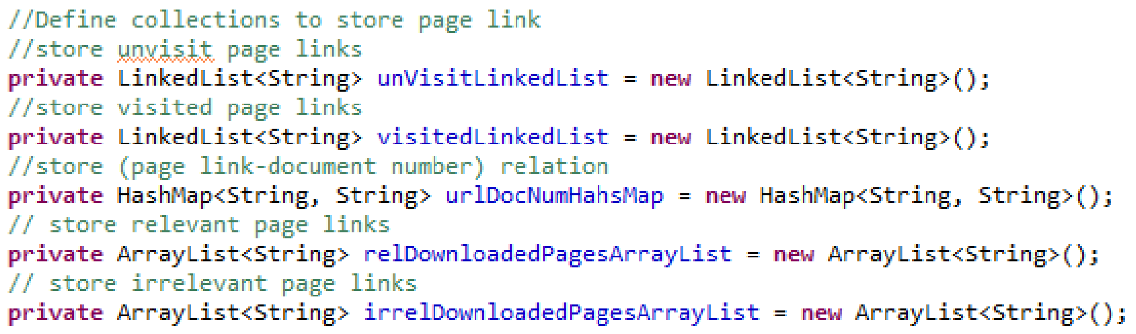
- **An overview of how your crawler is implemented, and how it operates, including politeness policies:**
  The **crawl.java** file implements the crawler. It follows these steps:
  a. user provides some running parameters which include:
  maxpages: limit the total number of pages that your crawler will fetch and store. If this option is not specified, the default maximum of 20 pages.(In order to collect the better training data for part2(Data Mining) and part3(Smart Web Crawler), I have made the crawler more smarter to fecth the equal number of the relevant and irrelevant pages from the source website. For example, the value of maxpages equals 100, the final result set should include 50 relevant pages and 50 irrelevant pages).
  seed_url: it gives the source website, crawler will start from this page(the url has to be type text/html).
  politeness: this parameter defines the time(in seconds) that crawler must wait before attempting to fetch another page from the same host. If this item is not provided, the default delay of 10 seconds.

b. First of all, we will introduce the rules for visiting and downloading page:
b.1 only open URLs that are type of text/html (such as images and videos). I will use sending head request method to Check Content-Type.
b.2 only follow links to pages that exist on the same host as that provided as the seed URL. For example: given seed url http://www.theage.com.au/digital-life/mobiles, only visit the pages which starts with http://www.theage.com.au host.
b.3 only consider the url which starts with "http:" protocol or "/" which means the pages on the same host.
c. Now we can start to fetch pages from seed url.
In order to keep tracking the page information, we provide **visited** and **unvisited collection** to store the visited and unvisited pages respectively. We also define **relevant** and **irrelevant collection** to store mobile and not mobile downloaded pages.
c.1 set seed url as current dealing page.
c.2 fetch all the legal url(use rules from step b) from source code of the current dealing page.(extract all the link from <a herf=""> tag).
c.3 move current dealing page into visited collection and add new links into unvisited collection. If the link has already existed in collection, ignore this link.
c.4 classify the current dealing page following this rule: assume the articles under the seed URL are all relevant to mobile and pages from outside are all irrelevant to mobiles.
c.5 download current dealing page: the relevant files are stored into "rel" directory and the irrelevant files are stored into "irrel" directory. Increase the number of the download pages by one.
c.6 check whether the number of the download pages reach the maxpages number. If yes, the program terminates. If no, go next step.
c.7 fetch a link from unvisited collection, set this url as current dealing page. then go back to step c.2

Note: As I mentioned at step a: maxpages part, I have made the crawler smarter to fetch the equal number of the relevant and irrelevant pages from the source website. The program tries to control the number of the downloaded pages for relevant and irrelevant pages. That means if the number of the downloaded relevant pages reach the half number of the maxpages, the crawler will stop fetching the relevant pages and try to find irrelevant pages. For example, the maxpages equals 100, the crawler will fetch 50 relevant pages and 50 irrelevant pages.

d. For **politeness policies**, the crawler will store the latest download timestamp, every time if we want to download the new page, the time interval between the latest download and the current time will be calculated and compared with the politeness parameter. If the time interval is smaller than politeness, the program will call **delayTimer function** to wait (politeness-interval) seconds. If not, the download operation executes immediately.

- **Any data structures that I have used, including to keep track of pages to visit, and pages already visited;**

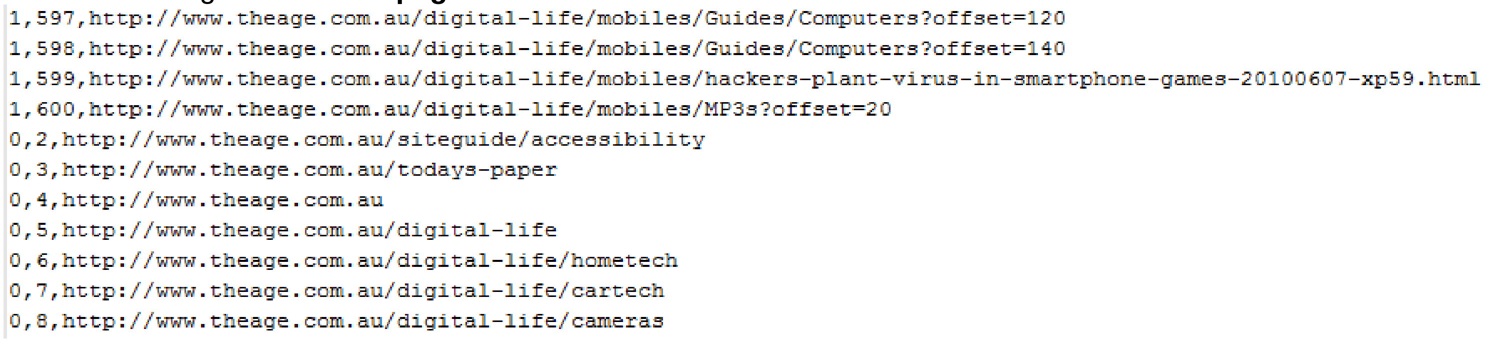


```
//Define collections to store page link
//store unvisit page links
private LinkedList<String> unVisitLinkedList = new LinkedList<String>();
//store visited page links
private LinkedList<String> visitedLinkedList = new LinkedList<String>();
//store (page link-document number) relation
private HashMap<String, String> urlDocNumHahsMap = new HashMap<String, String>();
// store relevant page links
private ArrayList<String> relDownloadedPagesArrayList = new ArrayList<String>();
// store irrelevant page links
private ArrayList<String> irrelDownloadedPagesArrayList = new ArrayList<String>();
```

Explanation:
➢ The crawler uses two **LinkedList<String>** data structure to store visited and unvisited url information respectively. A linked list is a data structure consisting of a group of nodes which

together represent a sequence. By using LinkedList structure, we can keep track of pages by added order.

- The crawler uses **HashMap<String,String>** data structure to store **page url** and **page assigned number(integer number)** pair. This assigned number is also the name of file stored on hard disk. So later we can use given url to find the corresponding file on disk.
- The crawler uses two **ArrayList<String>** data structure to store classified page url. All the relevant downloaded pages' url will be stored in **relDownloadPagesArrayList** collection and the irrelevant downloaded pages' url will be stored in **irreldownloadPagesArrayList** collection.
- The **crawl.java** file will generate following files:
  All the download **relevant** and **irrelevant** files will be stored under **./basic/rel** and **./basic/irrel** directory respectively. The files under these directories are named as integer number.
  It also generates one **pages** file which records the information about each file:



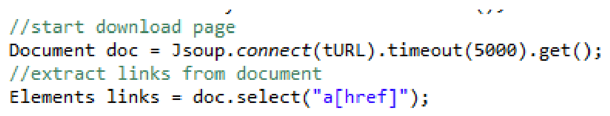
```
1,597,http://www.theage.com.au/digital-life/mobiles/Guides/Computers?offset=120
1,598,http://www.theage.com.au/digital-life/mobiles/Guides/Computers?offset=140
1,599,http://www.theage.com.au/digital-life/mobiles/hackers-plant-virus-in-smartphone-games-20100607-xp59.html
1,600,http://www.theage.com.au/digital-life/mobiles/MP3s?offset=20
0,2,http://www.theage.com.au/siteguide/accessibility
0,3,http://www.theage.com.au/todays-paper
0,4,http://www.theage.com.au
0,5,http://www.theage.com.au/digital-life
0,6,http://www.theage.com.au/digital-life/hometech
0,7,http://www.theage.com.au/digital-life/cartech
0,8,http://www.theage.com.au/digital-life/cameras
```

Explanation: The first column indicates the page type (1-Mobile or 0-Not Mobile). The second column indicates the file name; The third column stores the original url for this page.

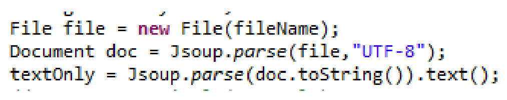
- **A brief explanation of any language-specific libraries or functions that I have used as part of my crawler or project**

  I have used following libraries as parts of my crawler:

  - **Jsoup**: jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods. It can parse HTML from a URl or file; find and extract element from HTML file using DOM traversal or CSS; modify the HTML elements, attributes and text content. In my crawler, I use Jsoup to download pages, extract link and fetch text content from file. The source code shows below:



```
//start download page
Document doc = Jsoup.connect(tURL).timeout(5000).get();
//extract links from document
Elements links = doc.select("a[href]");
```

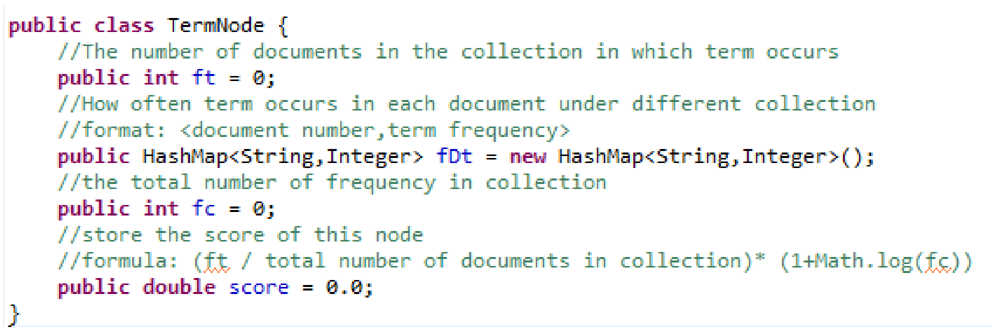


```
File file = new File(fileName);
Document doc = Jsoup.parse(file,"UTF-8");
textOnly = Jsoup.parse(doc.toString()).text();
```

Explanation: The left side code shows how to download pages, extract links from given url. The right side code shows how to fetch text content from file.

  - **The Porter Stemming Algorithm:** The **Stemmer.java** implements Porter Stemming algorithm. This Algorithm is a process for removing the commoner morphological and inflexional endings from words in English. The project applies this algorithm to each download pages and the stop word list when we generate term statistic information.
  - **Weka API(weka.jar)**: Weka is a collection of machine learning algorithms for solving data mining problems. It is written in Java. In this project we will use different classifiers to test and build our model.

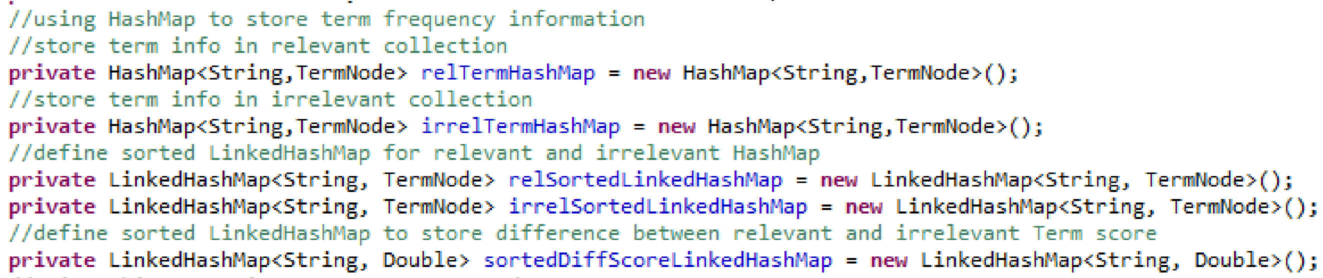
- **My design of features for classification**
  First of all, I will introduce the data structure that I have used for choosing features:
  TermNode: use for store term information:



```
public class TermNode {
    //The number of documents in the collection in which term occurs
    public int ft = 0;
    //How often term occurs in each document under different collection
    //format: <document number,term frequency>
    public HashMap<String,Integer> fDt = new HashMap<String,Integer>();
    //the total number of frequency in collection
    public int fc = 0;
    //store the score of this node
    //formula: (ft / total number of documents in collection)* (1+Math.log(fc))
    public double score = 0.0;
}
```

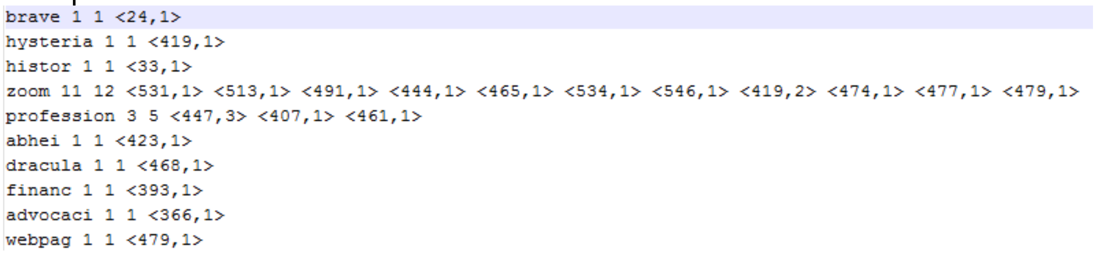
Explanation: this class includes four fields which store term information in each collection(relevant and irrelevant collection). Please look at the **score** field part which shows the **calculation formula**.

The **index.java** file implements term index construction, term score calculation and feature selection functions. The data structures which used for this part shows here:



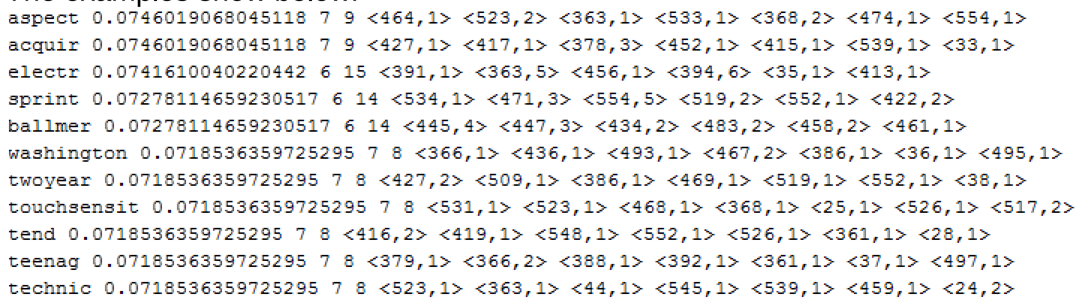
```
//using HashMap to store term frequency information
//store term info in relevant collection
private HashMap<String,TermNode> relTermHashMap = new HashMap<String,TermNode>();
//store term info in irrelevant collection
private HashMap<String,TermNode> irrelTermHashMap = new HashMap<String,TermNode>();
//define sorted LinkedHashMap for relevant and irrelevant HashMap
private LinkedHashMap<String, TermNode> relSortedLinkedHashMap = new LinkedHashMap<String, TermNode>();
private LinkedHashMap<String, TermNode> irrelSortedLinkedHashMap = new LinkedHashMap<String, TermNode>();
//define sorted LinkedHashMap to store difference between relevant and irrelevant Term score
private LinkedHashMap<String, Double> sortedDiffScoreLinkedHashMap = new LinkedHashMap<String, Double>();
```

The program uses first two **HashMap<String,TermNode>** data structure to store terms information for relevant and irrelevant collection respectively. The key part of this HashMap is term string and the value of the HashMap is term information which introduced before. We use **relTerms** and **irrelTerms** files which are located under basic directory to store unsorted terms information. The examples show below:



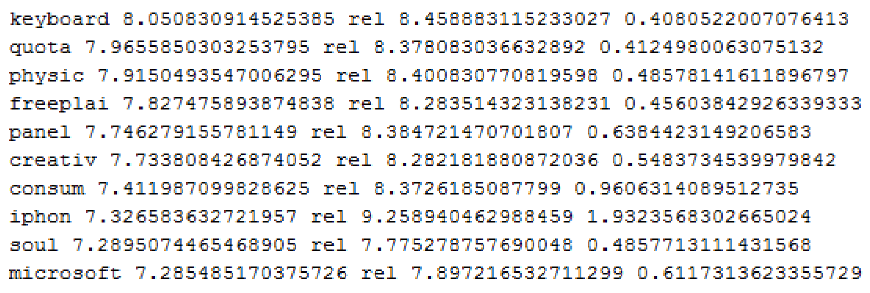
```
brave 1 1 <24,1>
hysteria 1 1 <419,1>
histor 1 1 <33,1>
zoom 11 12 <531,1> <513,1> <491,1> <444,1> <465,1> <534,1> <546,1> <419,2> <474,1> <477,1> <479,1>
profession 3 5 <447,3> <407,1> <461,1>
abhei 1 1 <423,1>
dracula 1 1 <468,1>
financ 1 1 <393,1>
advocaci 1 1 <366,1>
webpag 1 1 <479,1>
```

After calculating the score for each term, the program use two **LinkedHashMap<String,TermNode>** to store sorted term information by score's descending order. We use **relSortedTerms** and **irrelSortedTerms** files which are located under basic directory to store sorted terms information. The examples show below:



```
aspect 0.0746019068045118 7 9 <464,1> <523,2> <363,1> <533,1> <368,2> <474,1> <554,1>
acquir 0.0746019068045118 7 9 <427,1> <417,1> <378,3> <452,1> <415,1> <539,1> <33,1>
electr 0.0741610040220442 6 15 <391,1> <363,5> <456,1> <394,6> <35,1> <413,1>
sprint 0.07278114659230517 6 14 <534,1> <471,3> <554,5> <519,2> <552,1> <422,2>
ballmer 0.07278114659230517 6 14 <445,4> <447,3> <434,2> <483,2> <458,2> <461,1>
washington 0.0718536359725295 7 8 <366,1> <436,1> <493,1> <467,2> <386,1> <36,1> <495,1>
twoyear 0.0718536359725295 7 8 <427,2> <509,1> <386,1> <469,1> <519,1> <552,1> <38,1>
touchsensit 0.0718536359725295 7 8 <531,1> <523,1> <468,1> <368,1> <25,1> <526,1> <517,2>
tend 0.0718536359725295 7 8 <416,2> <419,1> <548,1> <552,1> <526,1> <361,1> <28,1>
teenag 0.0718536359725295 7 8 <379,1> <366,2> <388,1> <392,1> <361,1> <37,1> <497,1>
technic 0.0718536359725295 7 8 <523,1> <363,1> <44,1> <545,1> <539,1> <459,1> <24,2>
```

The last **LinkedHashMap<String,TermNode>** data structure includes all the terms from relevant and irrelevant collection, calculates the difference score between them and also sorts the results by score's descending order. We use **TermDiff** files which are located under basic directory to store compared terms information. The example shows below:



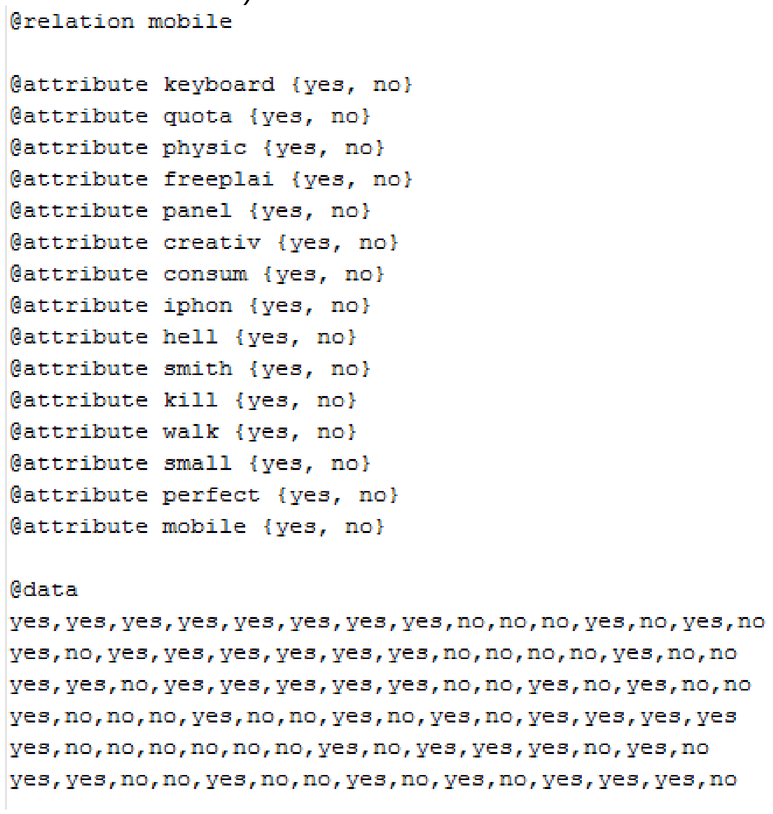
```
keyboard 8.050830914525385 rel 8.458883115233027 0.4080522007076413
quota 7.9655850303253795 rel 8.378083036632892 0.4124980063075132
physic 7.9150493547006295 rel 8.400830770819598 0.48578141611896797
freeplai 7.827475893874838 rel 8.283514323138231 0.45603842926339333
panel 7.746279155781149 rel 8.384721470701807 0.6384423149206583
creativ 7.733808426874052 rel 8.282181880872036 0.5483734539979842
consum 7.411987099828625 rel 8.3726185087799 0.9606314089512735
iphon 7.326583632721957 rel 9.258940462988459 1.9323568302665024
soul 7.2895074465468905 rel 7.775278757690048 0.4857713111431568
microsoft 7.285485170375726 rel 7.897216532711299 0.6117313623355729
```

Explanation: the first column is the term string; the second column shows the difference score between relevant and irrelevant score; the third column shows this term should put more weight on relevant or irrelevant collection; the last two columns show the term score in relevant and irrelevant collection respectively.

According to previous introduction, we can start to build our own features model:

- User provides two parameters: one for the size of relevant features, the other for the size of irrelevant features.
- According to the value of previous parameters, the program will select top number of relevant and irrelevant words from **sortedDiffScoreLinkedHashMap**.
- Then **FeatureExtraction.java** file will generate basic model by using these features ("**model_basic**" file stores original model information).
- It also generates the **mobile.arff** training file for building classifier in weka.

- **The details of my arff file for training**
  In my implementation, I have selected 120 relevant features and 30 irrelevant features to build my basic feature model. So there 150 attributes and one class attribute in the arff file(we can modify the size of the attribute by changing the attribute size parameters following previous steps). **Mobile.arff** file which is generated by **index.java** and **FeatureExtraction.java** programs is stored on current working directory. Now let us look at the details of my arff file.(because the file is too big, so I use a small example which includes 7 relevant and 7 irrelevant attributes to explain the structure of the file)



```
@relation mobile

@attribute keyboard {yes, no}
@attribute quota {yes, no}
@attribute physic {yes, no}
@attribute freeplai {yes, no}
@attribute panel {yes, no}
@attribute creativ {yes, no}
@attribute consum {yes, no}
@attribute iphon {yes, no}
@attribute hell {yes, no}
@attribute smith {yes, no}
@attribute kill {yes, no}
@attribute walk {yes, no}
@attribute small {yes, no}
@attribute perfect {yes, no}
@attribute mobile {yes, no}

@data
yes,yes,yes,yes,yes,yes,yes,yes,no,no,no,yes,no,yes,no
yes,no,yes,yes,yes,yes,yes,yes,no,no,no,no,yes,no,no
yes,yes,no,yes,yes,yes,yes,yes,no,no,yes,no,yes,no,no
yes,no,no,no,yes,no,no,yes,no,yes,no,yes,yes,yes,yes
yes,no,no,no,no,no,no,yes,no,yes,yes,yes,no,yes,no
yes,yes,no,no,yes,no,no,yes,no,yes,no,yes,yes,yes,no
```

Explanation: The first line gives the information about the file which is talking about the mobile. The attribute parts display 14 attributes which automatically selected by my program and the last mobile attribute is class label. The type of all the attributes is binary type which **yes** means the term occurs in current page and **no** means the term not occurs in current page. For data part, each line represents one web page and each column corresponds to one attribute which is introduced before.

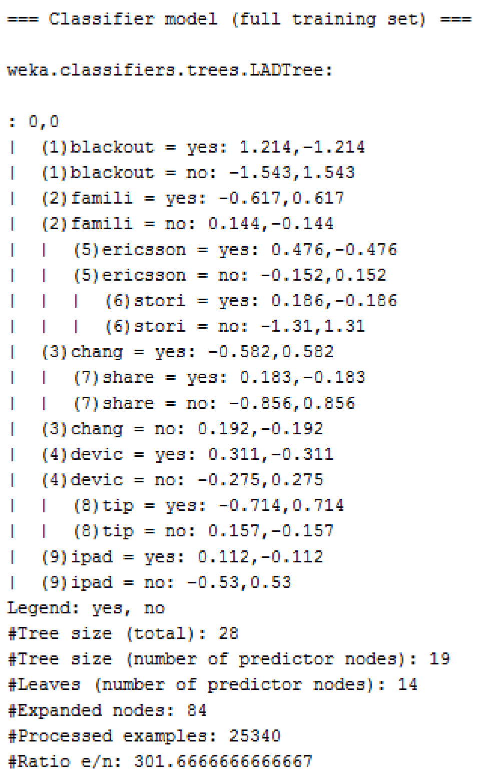
- **The classifiers that I have experimented with and my findings through the exercise**
  After generating the training data set which stored in **mobile.arff** file, I have done the following experiments using different classifiers(**Cross Validation** option has been used during the training).
  ZeroR:

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         300               50      %
Incorrectly Classified Instances       300               50      %
Kappa statistic                          0
Mean absolute error                      0.5
Root mean squared error                  0.5
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances              600

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         1          0.5       1         0.667      0.5      yes
                 0         0          0         0         0          0.5      no
Weighted Avg.    0.5       0.5        0.25      0.5       0.333      0.5

=== Confusion Matrix ===

   a   b   <-- classified as
 300   0 |   a = yes
 300   0 |   b = no
```

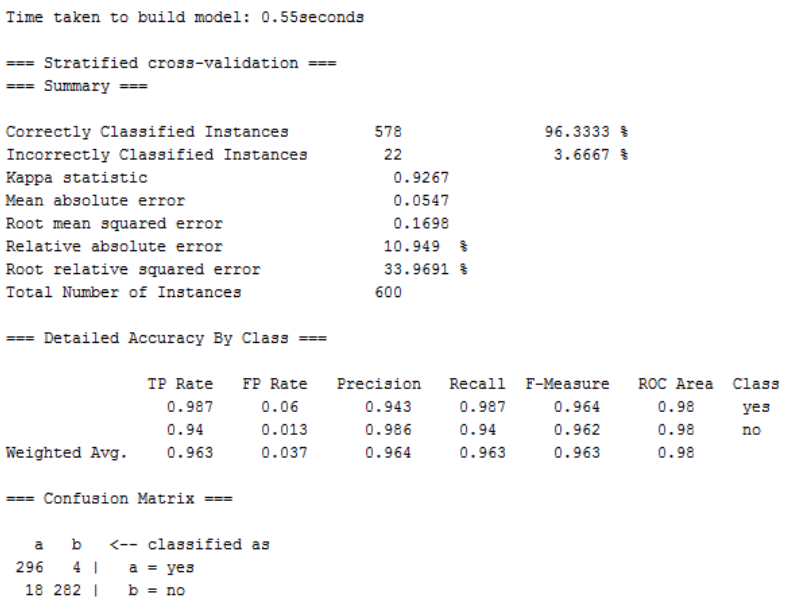
OneR:

```
[list of attributes omitted]
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

blackout:
        yes     -> yes
        no      -> no
(580/600 instances correct)


Time taken to build model: 0.01seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         578               96.3333 %
Incorrectly Classified Instances        22                3.6667 %
Kappa statistic                          0.9267
Mean absolute error                      0.0367
Root mean squared error                  0.1915
Relative absolute error                  7.3333 %
Root relative squared error             38.2971 %
Total Number of Instances              600

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.997     0.07       0.934     0.997     0.965      0.963    yes
                 0.93      0.003      0.996     0.93      0.962      0.963    no
Weighted Avg.    0.963     0.037      0.965     0.963     0.963      0.963

=== Confusion Matrix ===

   a   b   <-- classified as
 299   1 |   a = yes
  21 279 |   b = no
```

LADTree:

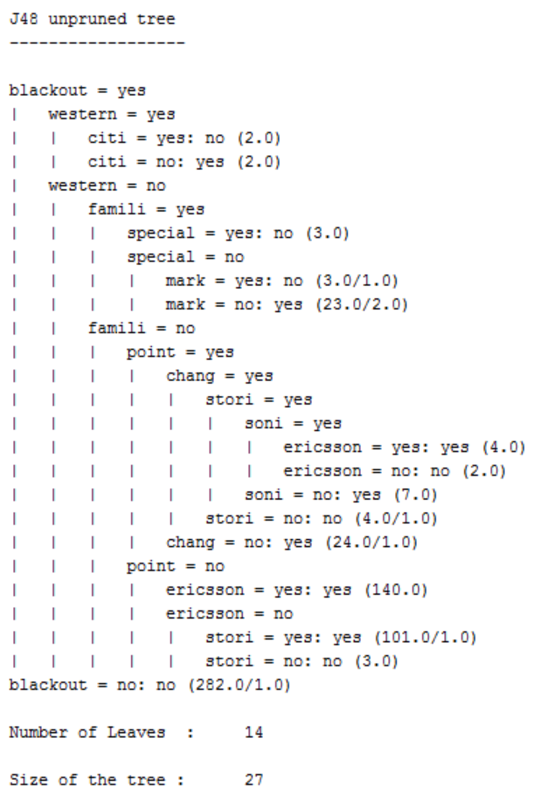


```
=== Classifier model (full training set) ===

weka.classifiers.trees.LADTree:

: 0,0
|  (1)blackout = yes: 1.214,-1.214
|  (1)blackout = no: -1.543,1.543
|  (2)famili = yes: -0.617,0.617
|  (2)famili = no: 0.144,-0.144
|  |  (5)ericsson = yes: 0.476,-0.476
|  |  (5)ericsson = no: -0.152,0.152
|  |  |  (6)stori = yes: 0.186,-0.186
|  |  |  (6)stori = no: -1.31,1.31
|  (3)chang = yes: -0.582,0.582
|  |  (7)share = yes: 0.183,-0.183
|  |  (7)share = no: -0.856,0.856
|  (3)chang = no: 0.192,-0.192
|  (4)devic = yes: 0.311,-0.311
|  (4)devic = no: -0.275,0.275
|  |  (8)tip = yes: -0.714,0.714
|  |  (8)tip = no: 0.157,-0.157
|  (9)ipad = yes: 0.112,-0.112
|  (9)ipad = no: -0.53,0.53
Legend: yes, no
#Tree size (total): 28
#Tree size (number of predictor nodes): 19
#Leaves (number of predictor nodes): 14
#Expanded nodes: 84
#Processed examples: 25340
#Ratio e/n: 301.6666666666667
```

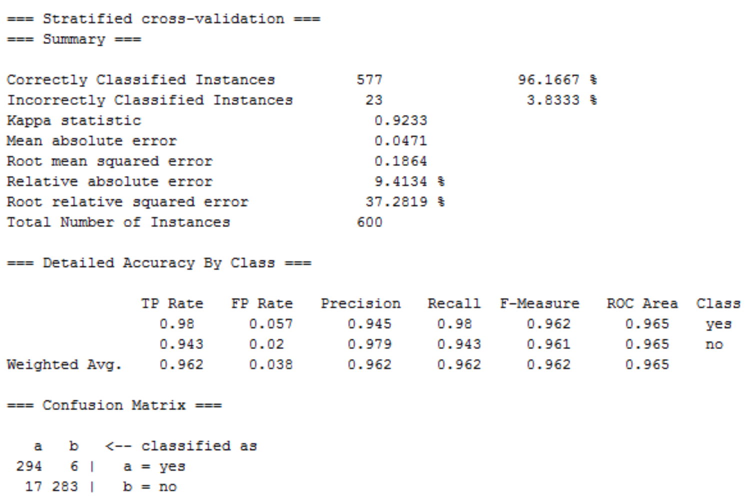


```
Time taken to build model: 0.55seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         578               96.3333 %
Incorrectly Classified Instances        22                3.6667 %
Kappa statistic                          0.9267
Mean absolute error                      0.0547
Root mean squared error                  0.1698
Relative absolute error                 10.949  %
Root relative squared error             33.9691 %
Total Number of Instances              600

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.987     0.06      0.943     0.987     0.964      0.98     yes
                 0.94      0.013     0.986     0.94      0.962      0.98     no
Weighted Avg.    0.963     0.037     0.964     0.963     0.963      0.98

=== Confusion Matrix ===

   a   b   <-- classified as
 296   4 |   a = yes
  18 282 |   b = no
```

J48:



```
J48 unpruned tree
------------------

blackout = yes
|   western = yes
|   |   citi = yes: no (2.0)
|   |   citi = no: yes (2.0)
|   western = no
|   |   famili = yes
|   |   |   special = yes: no (3.0)
|   |   |   special = no
|   |   |   |   mark = yes: no (3.0/1.0)
|   |   |   |   mark = no: yes (23.0/2.0)
|   |   famili = no
|   |   |   point = yes
|   |   |   |   chang = yes
|   |   |   |   |   stori = yes
|   |   |   |   |   |   soni = yes
|   |   |   |   |   |   |   ericsson = yes: yes (4.0)
|   |   |   |   |   |   |   ericsson = no: no (2.0)
|   |   |   |   |   |   soni = no: yes (7.0)
|   |   |   |   |   stori = no: no (4.0/1.0)
|   |   |   |   chang = no: yes (24.0/1.0)
|   |   |   point = no
|   |   |   |   ericsson = yes: yes (140.0)
|   |   |   |   ericsson = no
|   |   |   |   |   stori = yes: yes (101.0/1.0)
|   |   |   |   |   stori = no: no (3.0)
blackout = no: no (282.0/1.0)

Number of Leaves  :     14

Size of the tree :      27
```



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         577               96.1667 %
Incorrectly Classified Instances        23                3.8333 %
Kappa statistic                          0.9233
Mean absolute error                      0.0471
Root mean squared error                  0.1864
Relative absolute error                  9.4134 %
Root relative squared error             37.2819 %
Total Number of Instances              600

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.98      0.057     0.945     0.98      0.962      0.965    yes
                 0.943     0.02      0.979     0.943     0.961      0.965    no
Weighted Avg.    0.962     0.038     0.962     0.962     0.962      0.965

=== Confusion Matrix ===

   a   b   <-- classified as
 294   6 |   a = yes
  17 283 |   b = no
```

From previous experiments we can see all other three classifiers have better performance than ZeroR method and they all get very high correctly accuracy(over 90%). It sounds like we got a good features set for training data set.

Note: I have done some experiments at different time(or day). Because the website keeps changing, so the training data set and model may keep changing as well.

- **The classifier or classifiers you selected for the smart crawler**
  For this project, I will use J48 classifier which implements C4.5 decision tree algorithm for the smart crawler.

- **The structure of my smart crawler, e.g. how does it make decisions based on the trained model?**
  My smart crawler includes two components:
  crawl_smart.java implements the basic crawler functions. When given a seed page URL, the crawler will fetch the page and fetch pages linked from the seed page, until either there are no more pages to fetch or any termination condition is reached. The crawler will only download pages that exist on the same host as that provided as the seed URL.
  classify_smart_J48.Java implements the function which makes decisions based on the training model. The whole process shows below:
  - Read stop words list from **stoplist_sorted** files to remove very frequent terms.
  - Read attributes(features) collection from **Mobile.arff** training data set.
  - Build **term inverted index** list for the new unseen pages which are downloaded from previous step.
  - Using term inverted index list to generate unlabeled testing data set(**mobile_test.arff**).
  - **classifier_J48.java** implements following functions:
    - using training data set(Mobile.arff) to build classifier model.
    - Applying classifier to testing data set(mobile_test.arff). So each unlabeled record will be classified by training model(mobile_test_labeled.arff).
    - Generate feedback file to end user(**classified_doc_J48**). The content shows below:

      ```
      25,http://www.news.com.au/money/investing,Not Mobile
      26,http://www.news.com.au/money/interest-rates,Not Mobile
      166,http://www.news.com.au/couriermail/,Not Mobile
      27,http://www.news.com.au/money/david-and-libby-koch,Not Mobile
      165,http://www.news.com.au/perthnow/,Not Mobile
      28,http://www.news.com.au/money/guides-tools,Not Mobile
      168,http://www.news.com.au/?from=networklinks,Not Mobile
      29,http://www.news.com.au/entertainment,Not Mobile
      167,http://www.news.com.au/heraldsun/,Not Mobile
      ```

      Explanation: the first column is document number; the second column is original url; the third column represents prediction result by using training model.

- **The performance on test or unseen data (websites)**
  Introduction to testing environment:
  - I have downloaded total 600 pages from http://www.theage.com.au/digital-life/mobiles, including 300 relevant pages and 300 irrelevant pages(The pages which under this link are all relevant to mobile and pages from outside are all irrelevant to mobiles). The program uses these pages to generate training model.
  - I have downloaded total 200 pages from http://www.news.com.au/technology. The program uses these pages to generate testing data set.

  The performance of the testing result is not good. All the unseen websites are assigned to **Not Mobile** label. Even if some of them look like talking about Mobile. The following picture shows the training model generated by weka J48 algorithm:

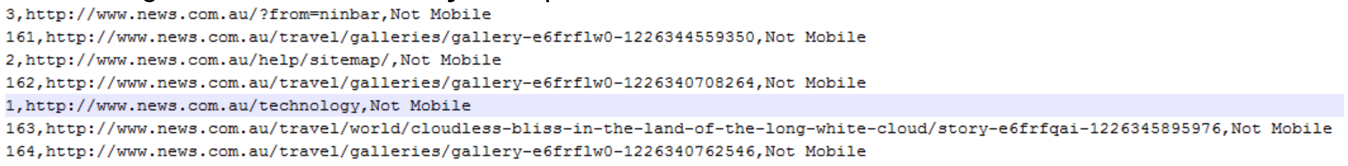
```
J48 unpruned tree
------------------

blackout = yes
|   western = yes
|   |   citi = yes: no (2.0)
|   |   citi = no: yes (2.0)
|   western = no
|   |   famili = yes
|   |   |   special = yes: no (3.0)
|   |   |   special = no
|   |   |   |   mark = yes: no (3.0/1.0)
|   |   |   |   mark = no: yes (23.0/2.0)
|   |   famili = no
|   |   |   point = yes
|   |   |   |   chang = yes
|   |   |   |   |   stori = yes
|   |   |   |   |   |   soni = yes
|   |   |   |   |   |   |   ericsson = yes: yes (4.0)
|   |   |   |   |   |   |   ericsson = no: no (2.0)
|   |   |   |   |   |   soni = no: yes (7.0)
|   |   |   |   |   stori = no: no (4.0/1.0)
|   |   |   |   chang = no: yes (24.0/1.0)
|   |   |   point = no
|   |   |   |   ericsson = yes: yes (140.0)
|   |   |   |   ericsson = no
|   |   |   |   |   stori = yes: yes (101.0/1.0)
|   |   |   |   |   stori = no: no (3.0)
blackout = no: no (282.0/1.0)
```

From this result we can see the model doesn't include many words about Mobile(only got soni,ericsson more likely talking about Mobile). And these words look like too specific for the training data set(the age websites). Here I will display some attributes from **Mobile.arff** file:

```
@attribute byte {yes, no}
@attribute iii {yes, no}
@attribute thinner {yes, no}
@attribute taller {yes, no}
@attribute revolut {yes, no}
@attribute reignit {yes, no}
@attribute toss {yes, no}
@attribute radio {yes, no}
@attribute metal {yes, no}
@attribute microsoft {yes, no}
@attribute mogul {yes, no}
@attribute gen {yes, no}
@attribute openplan {yes, no}
@attribute nook {yes, no}
@attribute bit {yes, no}
@attribute shut {yes, no}
@attribute samsung {yes, no}
@attribute smartphon {yes, no}
@attribute centuri {yes, no}
@attribute tech {yes, no}
@attribute phone {yes, no}
@attribute war {yes, no}
@attribute appl {yes, no}
@attribute iphon {yes, no}
@attribute htc {yes, no}
@attribute blackout {yes, no}
@attribute hook {yes, no}
@attribute eread {yes, no}
@attribute everydai {yes, no}
@attribute chapter {yes, no}
@attribute pirat {yes, no}
@attribute nobl {yes, no}
@attribute consum {yes, no}
@attribute rumour {yes, no}
@attribute shack {yes, no}
@attribute comput {yes, no}
@attribute comic {yes, no}
@attribute nokia {yes, no}
```

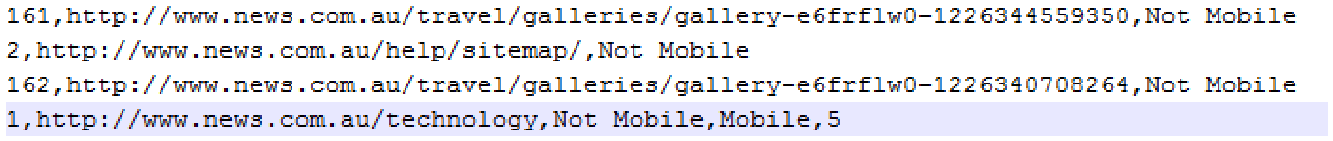
From this picture we can see the selected features set includes good words about Mobile, eg: iii,radio,micorsoft,samsung,smartphon,tech,phone,appl,iphon,htc,comput,nokia. But the J48 algorithm hasn't pick much words from these terms to build the final model. This means our training set is not big enough or the words is too specific for training pages. We will try to improve the result at next stage.

- **How user feedback (if any) is used to re-train the classifier?**
  re_train_classifier.java implements all the function about re-train the classifier. The whole process shows below:
  - user go into **retrain directory** and open the **classified_doc_J48** file, the file shows below:



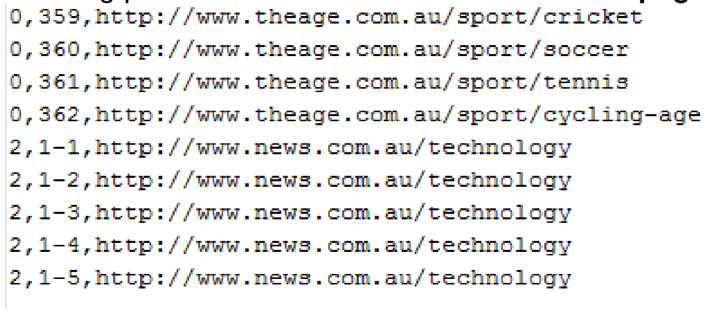
```
3,http://www.news.com.au/?from=ninbar,Not Mobile
161,http://www.news.com.au/travel/galleries/gallery-e6frflw0-1226344559350,Not Mobile
2,http://www.news.com.au/help/sitemap/,Not Mobile
162,http://www.news.com.au/travel/galleries/gallery-e6frflw0-1226340708264,Not Mobile
1,http://www.news.com.au/technology,Not Mobile
163,http://www.news.com.au/travel/world/cloudless-bliss-in-the-land-of-the-long-white-cloud/story-e6frfqai-1226345895976,Not Mobile
164,http://www.news.com.au/travel/galleries/gallery-e6frflw0-1226340762546,Not Mobile
```

  As described before, the third column represents the classified result for current record by using our training model. Currently we cannot find any record is labelled as Mobile in this file. But after checking website for the fifth record(No 1) which is talking about different Mobile device, so I think this page is about Mobile topic. Then the user can mark this record as below:



```
161,http://www.news.com.au/travel/galleries/gallery-e6frflw0-1226344559350,Not Mobile
2,http://www.news.com.au/help/sitemap/,Not Mobile
162,http://www.news.com.au/travel/galleries/gallery-e6frflw0-1226340708264,Not Mobile
1,http://www.news.com.au/technology,Not Mobile,Mobile,5
```

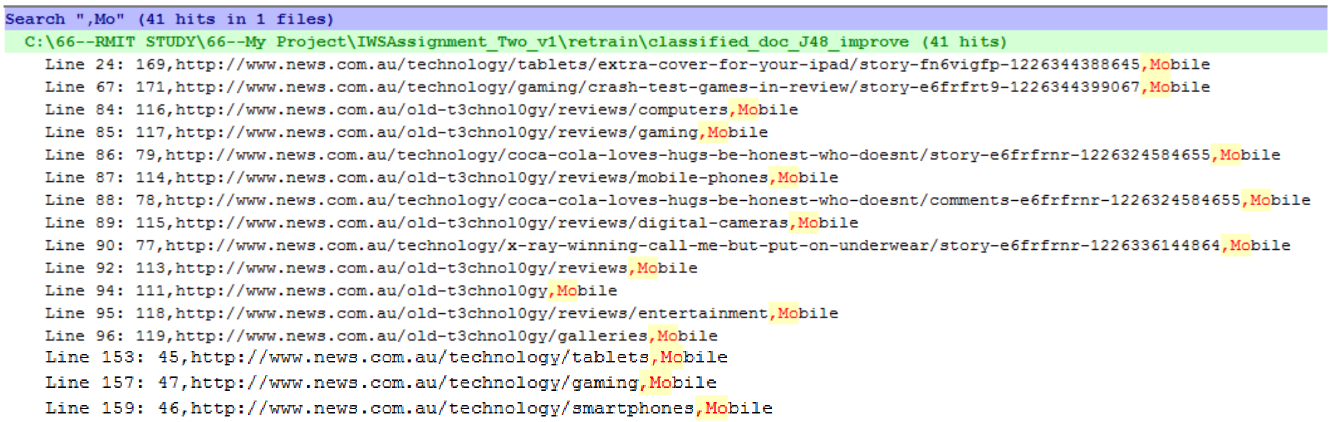
  We add **"Mobile,5"** to the end of this line which means user considers this pages should be talking about Mobile and the similarity score for this page is given 5. For this implementation, we can assign any positive integer number for this score. This score means the weight value for this article. For example, if I assign 5 to this article, the program will generate five records with different document number but the same url to the new training data set(**pages** file). All the words appeared in this page will be considered appear five times term frequency value. The following picture shows the final added results in **pages** file



```
0,359,http://www.theage.com.au/sport/cricket
0,360,http://www.theage.com.au/sport/soccer
0,361,http://www.theage.com.au/sport/tennis
0,362,http://www.theage.com.au/sport/cycling-age
2,1-1,http://www.news.com.au/technology
2,1-2,http://www.news.com.au/technology
2,1-3,http://www.news.com.au/technology
2,1-4,http://www.news.com.au/technology
2,1-5,http://www.news.com.au/technology
```

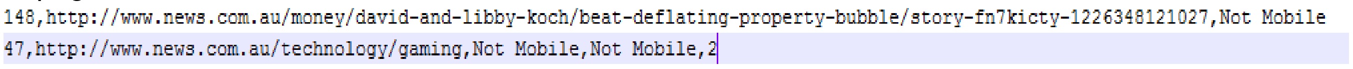
  - After user mark an irrelevant article "Not Mobile" or mark a genuine "Mobile" article as positive if it is indeed marked wrongly by the crawler. The program will generate the new **pages** file(stored under retrain directory) which combine the original pages from training website and the new marked pages from previous step.
  - According to this new training collection, **re_train_index.java** program will regenerate the new term inverted list and all the relative files stored under retrain directory.
  - It also rebuilds the new training data set(**retrain/mobile.arff**) and testing data set(**retrain/mobile_test.arff**).
  - **classify_smart_J48.Java** implements following functions:
    - Using new training data set(retrain/mobile.arff) to build classifier model.
    - Applying classifier to testing data set(retrain/mobile_test.arff). So each unlabeled record will be classified by training model(retrain/mobile_test_labeled.arff).
    - Generate the final feedback file to end user(retrain/classified_doc_J48_imporve).

- **The performance of the re-trained model on test or unseen (websites)**
  After re-training our classifier model by previous operations, we find the result has been improved. Before re-training the model we cannot find any record which is labelled as Mobile. But now we can find some records are labelled as Mobile. The result(/classified_doc_J48_imporve) shows below:



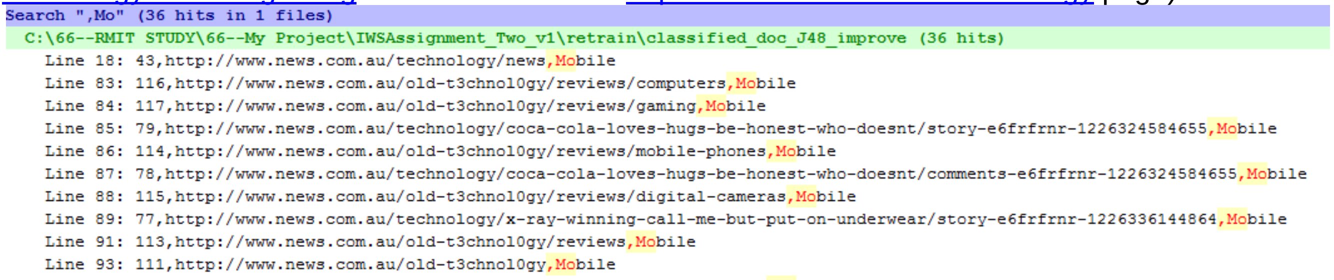
```
Search ",Mo" (41 hits in 1 files)
  C:\66--RMIT STUDY\66--My Project\IWSAssignment_Two_v1\retrain\classified_doc_J48_improve (41 hits)
    Line 24: 169,http://www.news.com.au/technology/tablets/extra-cover-for-your-ipad/story-fn6vigfp-1226344388645,Mobile
    Line 67: 171,http://www.news.com.au/technology/gaming/crash-test-games-in-review/story-e6frfrt9-1226344399067,Mobile
    Line 84: 116,http://www.news.com.au/old-t3chnol0gy/reviews/computers,Mobile
    Line 85: 117,http://www.news.com.au/old-t3chnol0gy/reviews/gaming,Mobile
    Line 86: 79,http://www.news.com.au/technology/coca-cola-loves-hugs-be-honest-who-doesnt/story-e6frfrnr-1226324584655,Mobile
    Line 87: 114,http://www.news.com.au/old-t3chnol0gy/reviews/mobile-phones,Mobile
    Line 88: 78,http://www.news.com.au/technology/coca-cola-loves-hugs-be-honest-who-doesnt/comments-e6frfrnr-1226324584655,Mobile
    Line 89: 115,http://www.news.com.au/old-t3chnol0gy/reviews/digital-cameras,Mobile
    Line 90: 77,http://www.news.com.au/technology/x-ray-winning-call-me-but-put-on-underwear/story-e6frfrnr-1226336144864,Mobile
    Line 92: 113,http://www.news.com.au/old-t3chnol0gy/reviews,Mobile
    Line 94: 111,http://www.news.com.au/old-t3chnol0gy,Mobile
    Line 95: 118,http://www.news.com.au/old-t3chnol0gy/reviews/entertainment,Mobile
    Line 96: 119,http://www.news.com.au/old-t3chnol0gy/galleries,Mobile
    Line 153: 45,http://www.news.com.au/technology/tablets,Mobile
    Line 157: 47,http://www.news.com.au/technology/gaming,Mobile
    Line 159: 46,http://www.news.com.au/technology/smartphones,Mobile
```

There are 41 records are labelled as Mobile. From these results we can see most of the url comes from technology directory, it sounds like get the better results than before(none article is labelled as Mobile). The Line157 are not about Mobile(more like about games). So now we can label this pages as below:



```
148,http://www.news.com.au/money/david-and-libby-koch/beat-deflating-property-bubble/story-fn7kicty-1226348121027,Not Mobile
47,http://www.news.com.au/technology/gaming,Not Mobile,Not Mobile,2
```

Then run re_**train_classifier.java** again. We got the improved results. The pages which is talking about games are labelled as Not Mobile.(Note: the page: http://www.news.com.au/old-t3chnol0gy/reviews/gaming will be redirected to http://www.news.com.au/technology page)



```
Search ",Mo" (36 hits in 1 files)
  C:\66--RMIT STUDY\66--My Project\IWSAssignment_Two_v1\retrain\classified_doc_J48_improve (36 hits)
    Line 18: 43,http://www.news.com.au/technology/news,Mobile
    Line 83: 116,http://www.news.com.au/old-t3chnol0gy/reviews/computers,Mobile
    Line 84: 117,http://www.news.com.au/old-t3chnol0gy/reviews/gaming,Mobile
    Line 85: 79,http://www.news.com.au/technology/coca-cola-loves-hugs-be-honest-who-doesnt/story-e6frfrnr-1226324584655,Mobile
    Line 86: 114,http://www.news.com.au/old-t3chnol0gy/reviews/mobile-phones,Mobile
    Line 87: 78,http://www.news.com.au/technology/coca-cola-loves-hugs-be-honest-who-doesnt/comments-e6frfrnr-1226324584655,Mobile
    Line 88: 115,http://www.news.com.au/old-t3chnol0gy/reviews/digital-cameras,Mobile
    Line 89: 77,http://www.news.com.au/technology/x-ray-winning-call-me-but-put-on-underwear/story-e6frfrnr-1226336144864,Mobile
    Line 91: 113,http://www.news.com.au/old-t3chnol0gy/reviews,Mobile
    Line 93: 111,http://www.news.com.au/old-t3chnol0gy,Mobile
```

These re-training results looks better than previous one. Please look at the following new training model which looks better than previous model.

```
J48 unpruned tree
------------------

blackout = yes
|   jame = yes
|   |   print = yes
|   |   |   fall = yes: no (2.0)
|   |   |   fall = no: yes (7.0)
|   |   print = no: no (4.0)
|   jame = no
|   |   western = yes: yes (3.0/1.0)
|   |   western = no
|   |   |   chang = yes
|   |   |   |   devic = yes: yes (32.0/2.0)
|   |   |   |   devic = no
|   |   |   |   |   googl = yes
|   |   |   |   |   |   point = yes: no (3.0)
|   |   |   |   |   |   point = no: yes (4.0/1.0)
|   |   |   |   |   googl = no: yes (4.0)
|   |   |   chang = no
|   |   |   |   govern = yes: yes (6.0/1.0)
|   |   |   |   govern = no
|   |   |   |   |   tip = yes: yes (6.0/1.0)
|   |   |   |   |   tip = no
|   |   |   |   |   |   michael = yes: yes (6.0/1.0)
|   |   |   |   |   |   michael = no
|   |   |   |   |   |   |   soni = yes: yes (150.0)
|   |   |   |   |   |   |   soni = no
|   |   |   |   |   |   |   |   stori = yes: yes (89.0/1.0)
|   |   |   |   |   |   |   |   stori = no: no (2.0)
blackout = no
|   iii = yes
|   |   screen = yes: no (10.0)
|   |   screen = no
|   |   |   microsoft = yes: yes (5.0)
|   |   |   microsoft = no: no (2.0)
|   iii = no: no (274.0/1.0)
```

- **My conclusions, comments on this mining exercise and possible future works on how to improve the smart crawler in terms of accuracy and efficiency**

  This mining exercise is a really hard work for me, but after finishing this exercise I have learned a lot of things which shows below:
  - How to write my own web crawler with Jsoup tools to fetch pages from website(s).
  - How to build term inverted list for different collection and calculate weight value for different term. When we build the term inverted list, we also have to consider how to remove stop words and perform term stemming operation.
  - How to user weka data mining tools to generate classifier model. Especially we have learned how to write Java code to build dynamic classifier model and apply this model to predict the new instances.
  - How to use different strategies to improve our model.(Although it is not the best one, we still got better results).

  In the future, I may use following ideas to improve my smart crawler:
  - In this stage, we only fetch pages from one website, we can see the terms are too specific for this website. So in next stage, we may try to grab the pages from different websites, it should be provided us more general terms for building training model.
  - When we try to get the content for each page, the current solution extracts all the text between HTML tags. I find some of information is useless, it may disturb the final results. But only depend on removing stop list words cannot remove these words effectively. So in next stage, we can analysis the structure of the web page and try to remove useless information.
  - Now we only try a few classifiers model in weka, we can see the generated model is not very good. In next stage we may try to use different algorithm to build better model.
  - Another problem is we only consider Australia websites, so if user applies this classifier model to a new website from different country, it may not generate good results because different culture and habit. So we can do some research in this field.

Reference:

1. Jsoup   http://jsoup.org

2. The Porter Stemming Algorithm http://tartarus.org/~martin/PorterStemmer/index.html

3. Stop list words list: http://www.webconfs.com/stop-words.php

4. Weka: http://sourceforge.net/projects/weka/