APPENDIX

Bobot.java

```
1       public class Bobot {
2       String doc_id;
3       Double bobot;
4
5       public Bobot(){}
6
7       public void setId(String doc_id) {
8         this.doc_id =doc_id;
9       }
10
11      public String getId() {
12        return doc_id;
13      }
14
15      public void setW(Double bobot) {
16        this.bobot =bobot;
17      }
18
19      public Double getW() {
20        return bobot;
21      }
22      }
```

BobotObject.java

```
23      public class BobotObject {
24      Double tf, df;
25      HashMap<String,ArrayList<Tf_idf>> listTf = new
HashMap<>();
26      ArrayList<Bobot> listBobot = new ArrayList<>();
27      String term;
28
29      public BobotObject(){}
30
31      public BobotObject(Double tf,Double df){
32        this.tf=tf;
33        this.df=df;
34      }
35
36      public void setTf(Double tf) {
37        this.tf = tf;
38      }
39
40      public Double getTf() {
41        return tf;
42      }
43
44      public void setDf(Double df) {
45        this.df = df;
46      }
47
48      public Double getDf() {
49        return df;
```

```
50      }
51
52      public      void      setListBobot(ArrayList<Bobot>
listBobot) {
53        this.listBobot = listBobot;
54      }
55
56      public ArrayList<Bobot> getListBobot() {
57        return listBobot;
58      }
59
60      public                                          void
setListTf(HashMap<String,ArrayList<Tf_idf>> listTf){
61        this.listTf = listTf;
62      }
63
64      public          HashMap<String,ArrayList<Tf_idf>>
getlistTf(){
65        return listTf;
66      }
67
68      public String getTerm() {
69        return term;
70      }
71
72      public void setTerm(String term) {
73        this.term = term;
74      }
75      }
```

CaseFolding.java

```
76      public class CaseFolding {
77
78      private String web_scraper_order, training_id,
title, author, deskripsi, kategori;
79
80      public CaseFolding(){}
81
82      public CaseFolding(String web_scraper_order,
String training_id, String title,
83                String author, String deskripsi, String
kategori){
84        this.web_scraper_order=web_scraper_order;
85        this.training_id=training_id;
86        this.title=title;
87        this.author=author;
88        this.deskripsi=deskripsi;
89        this.kategori=kategori;
90      }
91
92      public       void       setweb_scraper_order(String
web_scraper_order){
93        this.web_scraper_order=web_scraper_order;
94      }
95
96      public String getweb_scraper_order(){
97        return web_scraper_order;
98      }
99
```

```
100   public void settraining_id(String training_id){
101     this.training_id=training_id;
102   }
103
104   public String gettraining_id(){
105     return training_id;
106   }
107
108   public void settitle(String title){
109     this.title=title;
110   }
111
112   public String gettitle(){
113     return title;
114   }
115
116   public void setauthor(String author){
117     this.author=author;
118   }
119
120   public String getauthor(){
121     return author;
122   }
123
124   public void setdeskripsi(String deskripsi){
125     this.deskripsi=deskripsi;
126   }
127
128   public String getdeskripsi(){
129     return deskripsi;
130   }
131
132   public void setkategori(String kategori){
133     this.kategori=kategori;
134   }
135
136   public String getkategori(){
137     return kategori;
138   }
139   }
CaseFoldingTesting.java
140   public class CaseFoldingTesting {
141   Connection connection;
142   private PreparedStatement statement;
143   private String deskripsi, judul;
144
145   public CaseFoldingTesting() throws
SQLException, IOException {
146       selectData();
147     }
148
149   public String ClearPunctDescription(String
deskripsi) {
150       return deskripsi.replaceAll("[^a-zA-Z'  ']",  "
").toLowerCase().replaceAll("0-
9","").replaceAll("[\\t\\n\\r]+","");
151   }
152
```

```
153  public void selectData()throws SQLException,
IOException
154  {
155    String query= "SELECT * FROM `testing_data`";
156
157    connection = SQLConnection.getConnection();
158                                  statement =
connection.prepareStatement(query);
159
160    ArrayList<CaseFolding> listCaseFolding = new
ArrayList<CaseFolding>();
161                          ResultSet rs =
statement.executeQuery(query);
162        while (rs.next()) {
163                    CaseFolding casefolding = new
CaseFolding();
164
casefolding.settraining_id(rs.getString("web-scraper-
order"));
165          casefolding.settitle(rs.getString("title"));
166
casefolding.setdeskripsi(ClearPunctDescription(rs.getS
tring("deskripsi")));
167          listCaseFolding.add(casefolding);
168        }
169    insertCaseFolding(listCaseFolding);
170  }
171
```

```
172  public void
insertCaseFolding(ArrayList<CaseFolding>
listCaseFolding)throws SQLException, IOException
173  {
174              String query= "insert into
casefolding_testing(testing_id,title,deskripsi)
values(?,?,?)";
175
176    connection = SQLConnection.getConnection();
177                                  statement =
connection.prepareStatement(query);
178
179    for (int i = 0; i < listCaseFolding.size(); i++) {
180
statement.setString(1,listCaseFolding.get(i).gettrainin
g_id());
181
statement.setString(2,listCaseFolding.get(i).gettitle());
182
statement.setString(3,listCaseFolding.get(i).getdeskri
psi());
183          statement.addBatch();
184        }
185
186    try {
187          statement.executeBatch();
188        }
189    catch (SQLException e) {
```

```
            System.out.println("Error message: " + e.getMessage());
            return;
        }
    }
    }
CaseFoldingTraining.java
    public class CaseFoldingTraining {
    Connection connection;
    private PreparedStatement statement;
    private String deskripsi, kategori, judul;

    public CaseFoldingTraining() throws SQLException, IOException {
        selectData();
      }

    public String ClearPunctDescription(String deskripsi) {
        return deskripsi.replaceAll("[^a-zA-Z' ']", "").toLowerCase().replaceAll("0-9","").replaceAll("[\\t\\n\\r]+","");
    }

    public void selectData()throws SQLException, IOException
    {
            String query= "SELECT * FROM `training_data`";

    connection = SQLConnection.getConnection();
                                    statement = connection.prepareStatement(query);

     ArrayList<CaseFolding> listCaseFolding = new ArrayList<CaseFolding>();
                                ResultSet rs = statement.executeQuery(query);
        while (rs.next()) {
                    CaseFolding casefolding = new CaseFolding();
casefolding.settraining_id(rs.getString("web-scraper-order"));
            casefolding.settitle(rs.getString("title"));
casefolding.setdeskripsi(ClearPunctDescription(rs.getString("deskripsi")));
casefolding.setkategori(rs.getString("kategori"));
            listCaseFolding.add(casefolding);
        }
    insertCaseFolding(listCaseFolding);
    }

    public                                  void insertCaseFolding(ArrayList<CaseFolding> listCaseFolding)throws SQLException, IOException
```

```
229    {
230        String query= "insert into casefolding_trainer
(training_id,title,deskripsi,kategori) values(?,?,?,?)";
231
232       connection = SQLConnection.getConnection();
233                                        statement          =
connection.prepareStatement(query);
234
235       for (int i = 0; i < listCaseFolding.size(); i++) {
236
statement.setString(1,listCaseFolding.get(i).gettrainin
g_id());
237
statement.setString(2,listCaseFolding.get(i).gettitle());
238
statement.setString(3,listCaseFolding.get(i).getdeskri
psi());
239
statement.setString(4,listCaseFolding.get(i).getkatego
ri());
240             statement.addBatch();
241          }
242
243       try {
244             statement.executeBatch();
245          } catch (SQLException e) {
246                System.out.println("Error message: " +
e.getMessage());
247             return;
248          }
249    }
250    }
Classification.java
251    public class Classification {
252
253    Connection connection;
254    PreparedStatement statement;
255
256    double alpha;
257    int epoch;
258
259    ArrayList<Id_Class> listTerm;
260
261    public Classification() throws SQLException {
262       listTerm = indexKategori();
263    }
264
265    public ArrayList<Id_Class>       indexKategori()
throws SQLException{
266             String query = "SELECT * FROM
`casefolding_trainer` order by `id` asc";
267
268       connection = SQLConnection.getConnection();
269                                        statement          =
connection.prepareStatement(query);
270       ResultSet rs = statement.executeQuery( );
271       rs = statement.executeQuery();
272
```

```
273             ArrayList<Id_Class> listTerm = new
ArrayList<>();
274     while(rs.next()){
275                                 listTerm.add(new
Id_Class(rs.getString("kategori"),
rs.getString("training_id")));
276     }
277     return listTerm;
278   }
279
280   public    ArrayList<BobotObject>    LVQ(double
alpha, int epoch,
281             ArrayList<BobotObject> listBobotData,
ArrayList<String> hasilranking){
282
283       ArrayList<BobotObject> listWClass = new
ArrayList<>();
284       ArrayList<BobotObject> listWData = new
ArrayList<>();
285
286       //object utk training data. pemisah antara
initiation sm yg dihitung
287     for (int i = 0; i < listBobotData.size(); i++) {
288       ArrayList<Bobot> listBobotDatatemp = new
ArrayList<Bobot>();
289       ArrayList<Bobot> listBobotClasstemp = new
ArrayList<Bobot>();
290
291                     for (int j = 0; j <
listBobotData.get(i).getListBobot().size(); j++) {
292                                         if
(hasilranking.contains(listBobotData.get(i).getListBob
ot().get(j).getId())) {
293             Bobot bobot = new Bobot();
294                                             bobot =
listBobotData.get(i).getListBobot().get(j);
295             listBobotClasstemp.add(bobot);
296           }
297           else{
298             Bobot bobot = new Bobot();
299                                             bobot =
listBobotData.get(i).getListBobot().get(j);
300             listBobotDatatemp.add(bobot);
301           }
302         }
303       BobotObject boClass = new BobotObject();
304       boClass.setTf(listBobotData.get(i).getTf());
305       boClass.setDf(listBobotData.get(i).getDf());
306
boClass.setListTf(listBobotData.get(i).getlistTf());
307       boClass.setListBobot(listBobotClasstemp);
308
boClass.setTerm(listBobotData.get(i).getTerm());
309       listWClass.add(boClass);
310
311       BobotObject boData = new BobotObject();
312       boData.setTf(listBobotData.get(i).getTf());
```

```
313         boData.setDf(listBobotData.get(i).getDf());
314 
boData.setListTf(listBobotData.get(i).getlistTf());
315 
boData.setTerm(listBobotData.get(i).getTerm());
316         boData.setListBobot(listBobotDatatemp);
317         listWData.add(boData);
318     }
319                 //pecahan jumlah data antara
representatif&data training
320                 System.err.println("w class :
"+listWClass.get(0).getListBobot().size());
321                 System.err.println("w data :
"+listWData.get(0).getListBobot().size());
322 
323     this.alpha = alpha;
324     for (int i = 0; i < epoch; i++) {
325 
System.err.println("========================
====="+"epoch                                :
"+epoch+"============================");
326                 listWClass = hitungClass(listWClass,
listWData);
327         this.alpha = 0.1*alpha;
328     }
329     return listWClass;
330   }
331 
```

```
332 public ArrayList<BobotObject> hitungClass
(ArrayList<BobotObject> listWClass,
ArrayList<BobotObject> listWData){
333 
334                         int sizedata =
listWData.get(0).getListBobot().size();
335 
336     // hitung pengurangan eucclidean distance
337     for (int i = 0; i < sizedata; i++) {
338                         double[] w = new
double[listWClass.get(0).getListBobot().size()];
339         for (int j = 0; j < listWData.size(); j++) {
340                         for (int k = 0; k <
listWClass.get(j).getListBobot().size(); k++) {
341                         double wtemp =
listWData.get(j).getListBobot().get(i).getW()
342                                 -
listWClass.get(j).getListBobot().get(k).getW();
343             wtemp = wtemp*wtemp;
344             w[k] += wtemp;
345         }
346       }
347       // hitung akar eucclidean distance
348       for(int l = 0 ; l < w.length;l++){
349         w[l] = Math.sqrt(w[l]);
350 
351         // start print W + index + kategori
352         System.err.println(" w : "+w[l]);
353       }
```

```
354         //cari nilai minimum
355         int index =  FindFirstMinIndex(w);
356                 System.err.println("index  :  "+index+"
"+listWClass.get(0).getListBobot().get(index).getId()
+" "+
357 
listWData.get(0).getListBobot().get(i).getId());
358 
359                                     String     wData     =
listWData.get(0).getListBobot().get(i).getId();
360                                    String     wClass     =
listWClass.get(0).getListBobot().get(index).getId();
361 
362         String katergoriData= getKatgori(wData);
363         String kategoriClass= getKatgori(wClass);
364 
365                  System.err.println("Category  data  =
"+wData+" is "+ katergoriData);
366                  System.err.println("Category  class  =
"+wClass+" is "+ kategoriClass);
367         //end of print W + index + kategori
368 
369         if (katergoriData.equals(kategoriClass)) {
370            for (int j = 0; j < listWClass.size(); j++) {
371                                                 double   c   =
listWClass.get(j).getListBobot().get(index).getW();
372                                                 double   d   =
listWData.get(j).getListBobot().get(i).getW();
373                 double temp= c+alpha*(d-c);
374 
listWClass.get(j).getListBobot().get(index).setW(temp)
;
375              }
376           } else {
377              for (int j = 0; j < listWClass.size(); j++) {
378                                                 double   c   =
listWClass.get(j).getListBobot().get(index).getW();
379                                                 double   d   =
listWData.get(j).getListBobot().get(i).getW();
380                 double temp= c-alpha*(d-c);
381 
listWClass.get(j).getListBobot().get(index).setW(temp)
;
382              }
383           }
384        }
385        return listWClass;
386     }
387 
388     public int FindFirstMinIndex(double[] w)
389     {
390        int minIndex = 0;
391        for (int i = 1; i < w.length; i++)
392           if(w[i] < w[minIndex])
393              minIndex=i;
394        return minIndex;
395     }
396 
```

```
397    public String getKatgori(String findString){
398
399       String kategori = "Notfound";
400       for (int j = 0; j < listTerm.size(); j++) {
401                                                    if
(findString.equals(listTerm.get(j).getId())) {
402                                                  kategori   =
listTerm.get(j).getKategori();
403                 break;
404              }
405           }
406       return kategori;
407    }
408    }
Id_Class.java
409    public class Id_Class {
410    String kategori, id;
411
412       public Id_Class() {
413       }
414
415       public Id_Class(String kategori, String id) {
416          this.kategori = kategori;
417          this.id = id;
418       }
419
420       public String getKategori() {
421          return kategori;
422       }
423
424       public void setKategori(String kategori) {
425          this.kategori = kategori;
426       }
427
428       public String getId() {
429          return id;
430       }
431
432       public void setId(String id) {
433          this.id = id;
434       }
435    }
MainMenu.java
436    public       class       MainMenu       implements
ActionListener {
437    JButton button1, button2;
438    public MainMenu(String title) throws Exception {
439       JFrame frame=new JFrame();
440                      Dimension       screenSize       =
Toolkit.getDefaultToolkit().getScreenSize();
441       Dimension windowSize = frame.getSize();
442        int windowX = Math.max(0, (screenSize.width
- windowSize.width ) / 2);
443       int windowY = Math.max(0, (screenSize.height
- windowSize.height) / 2);
444
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
E);
```

```
445             frame.getContentPane().add(setLayout(),
BorderLayout.CENTER);
446       frame.setTitle("LVQ Project");
447         frame.setLocation(windowX-500/2,  windowY-
500/2);
448       frame.pack();
449       frame.setVisible(true);
450       setLayout();
451
452       System.out.println("Masuk Main Menu");
453     }
454
455     public JButton TrainingButton(String b1){
456      JButton button1 = new JButton(b1);
457      button1.addActionListener(this);
458      return button1;
459     }
460
461     public JButton TestingButton(String b2){
462      JButton button2 = new JButton(b2);
463      button2.addActionListener(this);
464      return button2;
465     }
466
467     public JPanel setLayout() {
468       //initUIComponents();
469       JPanel panel1 = new JPanel();
470                                   panel1.setLayout(new
BoxLayout(panel1,BoxLayout.Y_AXIS));
471
472             JPanel panelgo = new JPanel(new
FlowLayout(FlowLayout.CENTER));
473       panelgo.add(TrainingButton("Training Menu"));
474       panelgo.add(TestingButton("Testing Menu"));
475
476       panel1.add(panelgo);
477
478       JPanel Pane = new JPanel();
479                                         Pane.setLayout(new
BoxLayout(Pane,BoxLayout.Y_AXIS));
480       Pane.add(panel1);
481       return Pane;
482     }
483
484     public void actionPerformed(ActionEvent event)
{
485           if  (event.getActionCommand()=="Training
Menu") {
486             try {
ObjectSum.java
487     public class ObjectSum {
488     String doc_id;
489     String kategori;
490     double sum;
491
492     public ObjectSum(){}
493
494     public void setDocId(String doc_id) {
```

```
495     this.doc_id =doc_id;
496   }
497
498   public String getDocId() {
499     return doc_id;
500   }
501
502   public void setkategori(String kategori){
503     this.kategori=kategori;
504   }
505
506   public String getkategori(){
507     return kategori;
508   }
509
510   public void setSum(double sum){
511     this.sum=sum;
512   }
513
514   public double getSum(){
515     return sum;
516   }
517   }
Project.java
518   public class Project {
519
520   public static void main(String[] args) throws
SQLException, IOException, Exception {
521         MainMenu coba = new MainMenu("LVQ
Project");
522   }
523   }
SQLConnection.java
524   public class SQLConnection {
525   private static Connection connection;
526   private static Statement statement;
527
528   public static Connection getConnection() throws
SQLException {
529     if(connection == null){
530       MysqlDataSource dataSource = null;
531       try{
532         dataSource = new MysqlDataSource();
533
dataSource.setUrl("jdbc:mysql://localhost/testproject6"
);
534         dataSource.setUser("root");
535         dataSource.setPassword("");
536                                          connection    =
dataSource.getConnection();
537                                           statement    =
connection.createStatement();
538       }catch(SQLException exc){
539               JOptionPane.showMessageDialog(null,
exc.getMessage(),          "Error          Connection",
JOptionPane.ERROR_MESSAGE);
540       }
```

```
541       }
542       return connection;
543     }
544
545     static                                PreparedStatement
prepareStatement(String query) {
546                                         throw              new
UnsupportedOperationException("Not          supported
yet.");
547     }
548     }
Stemming.java
549     public class Stemming {
550     private String tn_id, title, deskripsi, kategori;
551
552     public Stemming() {}
553
554     public Stemming(String tn_id, String title, String
deskripsi, String kategori){
555       this.tn_id=tn_id;
556       this.title=title;
557       this.deskripsi=deskripsi;
558       this.kategori=kategori;
559     }
560
561     public void settnid(String tn_id){
562       this.tn_id=tn_id;
563     }
564
565     public String gettnid(){
566       return tn_id;
567     }
568
569     public void settitle(String title){
570       this.title=title;
571     }
572
573     public String gettitle(){
574       return title;
575     }
576
577     public void setdeskripsi(String deskripsi){
578       this.deskripsi=deskripsi;
579     }
580
581     public String getdeskripsi(){
582       return deskripsi;
583     }
584
585     public void setkategori(String kategori){
586       this.kategori=kategori;
587     }
588
589     public String getkategori(){
590       return kategori;
591     }
592     }
StemmingTesting.java
```

```
593   public class StemmingTesting {
594   Connection connection;
595   IndonesianStemmer stemmer;
596   public String id,title,deskripsi;
597   PreparedStatement statement;
598
599   public StemmingTesting() throws SQLException,
IOException{
600     lastStemming();
601   }
602
603   public String stem(String word){
604     stemmer = new IndonesianStemmer();
605     char[] kata = word.toCharArray();
606       int len = stemmer.stem(kata, kata.length,
true);
607     String stem = new String(kata, 0, len);
608
609     return stem;
610   }
611   public void lastStemming() throws
SQLException, IOException{
612           String query = "SELECT * FROM
`stopword_testing`";
613
614     connection = SQLConnection.getConnection();
615                                 statement =
connection.prepareStatement(query);
616
617           ArrayList<Stemming> listStem = new
ArrayList<Stemming>();
618                                 ResultSet rs =
statement.executeQuery(query);
619         while (rs.next()) {
620           Stemming stemming = new Stemming();
621           stemming.settnid(rs.getString("st_id"));
622           stemming.settitle(rs.getString("title"));
623
stemming.setdeskripsi(stem(rs.getString("deskripsi")))
;
624           listStem.add(stemming);
625         }
626         insertStem(listStem);
627   }
628
629   public void insertStem(ArrayList<Stemming>
listStem) throws SQLException, IOException
630   {
631     String query= "insert into stemming_testing
(tn_id,title,deskripsi) values(?,?,?)";
632
633     connection = SQLConnection.getConnection();
634                                 statement =
connection.prepareStatement(query);
635
636     for (int i = 0; i < listStem.size(); i++) {
637
statement.setString(1,listStem.get(i).gettnid());
```

```
statement.setString(2,listStem.get(i).gettitle());
statement.setString(3,listStem.get(i).getdeskripsi());
            statement.addBatch();
        }
    try {
            statement.executeBatch();
        } catch (SQLException e) {
            System.out.println("Error message: " +
e.getMessage());
            return;
        }
    }
    }
StemmingTraining.java
    public class StemmingTraining {
    Connection connection;
    IndonesianStemmer stemmer;
    public String id,title,deskripsi,kategori;
    PreparedStatement statement;

    public StemmingTraining() throws SQLException,
IOException{
      lastStemming();
    }

    public String stem(String word){
      stemmer = new IndonesianStemmer();
      char[] kata = word.toCharArray();
        int len = stemmer.stem(kata, kata.length,
true);
      String stem = new String(kata, 0, len);

      return stem;
    }

    public void lastStemming() throws
SQLException, IOException{
            String query = "SELECT * FROM
`stopword_trainer`";

      connection = SQLConnection.getConnection();
                                    statement =
connection.prepareStatement(query);

          ArrayList<Stemming> listStem = new
ArrayList<Stemming>();
                                ResultSet rs =
statement.executeQuery(query);
        while (rs.next()) {
            Stemming stemming = new Stemming();
            stemming.settnid(rs.getString("st_id"));
            stemming.settitle(rs.getString("title"));

stemming.setdeskripsi(stem(rs.getString("deskripsi")))
;
```

```
682 stemming.setkategori(rs.getString("kategori"));
683             listStem.add(stemming);
684         }
685         insertStem(listStem);
686     }
687
688     public void insertStem(ArrayList<Stemming>
listStem) throws SQLException, IOException
689     {
690         String query= "insert into stemming_trainer
(tn_id,title,deskripsi,kategori) values(?,?,?,?)";
691
692       connection = SQLConnection.getConnection();
693                                     statement          =
connection.prepareStatement(query);
694
695       for (int i = 0; i < listStem.size(); i++) {
696
statement.setString(1,listStem.get(i).gettnid());
697
statement.setString(2,listStem.get(i).gettitle());
698
statement.setString(3,listStem.get(i).getdeskripsi());
699
statement.setString(4,listStem.get(i).getkategori());
700             statement.addBatch();
701         }
702
703       try {
704             statement.executeBatch();
705         } catch (SQLException e) {
706                 System.out.println("Error message: " +
e.getMessage());
707             return;
708         }
709     }
710     }
StopWord.java
711     public class StopWord {
712     private String st_id, title, deskripsi, kategori;
713
714     public StopWord(){}
715
716     public StopWord(String st_id, String title, String
deskripsi, String kategori){
717       this.st_id=st_id;
718       this.title=title;
719       this.deskripsi=deskripsi;
720       this.kategori=kategori;
721     }
722
723     public void setstid(String st_id){
724       this.st_id=st_id;
725     }
726
727     public String getstid(){
728       return st_id;
```

```
729    }
730
731    public void settitle(String title){
732      this.title=title;
733    }
734
735    public String gettitle(){
736      return title;
737    }
738
739    public void setdeskripsi(String deskripsi){
740      this.deskripsi=deskripsi;
741    }
742
743    public String getdeskripsi(){
744      return deskripsi;
745    }
746
747    public void setkategori(String kategori){
748      this.kategori=kategori;
749    }
750
751    public String getkategori(){
752      return kategori;
753    }
754    }
StopWordTesting.java
755    public class StopWordTesting {
756    Connection connection;
757    public String id,title,deskripsi;
758    PreparedStatement statement;
759    IndonesianAnalyzer indonesianAnalyzer;
760    public StopWordTesting() throws SQLException,
IOException{
761                    indonesianAnalyzer = new
IndonesianAnalyzer(readFile("/home/amalia/NetBeansP
rojects/project/src/project/stopword-758.txt"));
762      lastStopWord();
763    }
764
765    public CharArraySet readFile(String filename)
766    {
767       CharArraySet setData = new CharArraySet(0,
true);
768      try
769      {
770                        FileInputStream fis=new
FileInputStream(filename);
771      Scanner sc=new Scanner(fis);
772        while(sc.hasNextLine())
773        {
774          setData.add(sc.nextLine());
775        }
776      sc.close();
777      }
778        catch(IOException e)
779      {
780        e.printStackTrace();
```

```
781      }
782      return setData;
783    }
784    public boolean isStopword(String word){
785                              CharArraySet c =
indonesianAnalyzer.getStopwordSet();
786      Iterator iter = c.iterator();
787         while(iter.hasNext()) {
788            char[] stopWord = (char[]) iter.next();
789            String stopword = new String (stopWord);
790            if(word.equals(new String (stopWord))){
791               return true;
792            }
793         }
794       return false;
795    }
796
797    public void lastStopWord() throws SQLException,
IOException{
798              String query = "SELECT * FROM
`tokenize_testing`";
799
800      connection = SQLConnection.getConnection();
801                                    statement =
connection.prepareStatement(query);
802
803        ArrayList<StopWord> listStopWords = new
ArrayList<StopWord>();
804      List<String> words = new ArrayList<String>();
805
806                                ResultSet rs =
statement.executeQuery(query);
807          while (rs.next()) {
808             StopWord sw = new StopWord();
809             sw.setstid(rs.getString("cf_id"));
810             sw.settitle(rs.getString("title"));
811
sw.setdeskripsi((rs.getString("deskripsi")));
812
813             if(!isStopword(sw.getdeskripsi())){
814                listStopWords.add(sw);
815             }
816          }
817          insertStopWord(listStopWords);
818    }
819
820    public                                        void
insertStopWord(ArrayList<StopWord> listStopWords)
throws SQLException, IOException
821    {
822        String query= "insert into stopword_testing
(st_id,title,deskripsi) values(?,?,?)";
823
824      connection = SQLConnection.getConnection();
825                                    statement =
connection.prepareStatement(query);
826
827      for (int i = 0; i < listStopWords.size(); i++) {
```

```
828 statement.setString(1,listStopWords.get(i).getstid());
829 statement.setString(2,listStopWords.get(i).gettitle());
830 statement.setString(3,listStopWords.get(i).getdeskripsi());
831             statement.addBatch();
832         }
833     try {
834             statement.executeBatch();
835         } catch (SQLException e) {
836                 System.out.println("Error message: " + e.getMessage());
837             return;
838         }
839     }
840     }
StopWordTraining.java
841     public class StopWordTraining {
842     Connection connection;
843     public String id,title,deskripsi,kategori;
844     PreparedStatement statement;
845     IndonesianAnalyzer indonesianAnalyzer;
846     public StopWordTraining() throws SQLException, IOException{
847                     indonesianAnalyzer = new IndonesianAnalyzer(readFile("/home/amalia/NetBeansProjects/project/src/project/stopword-758.txt"));
848       lastStopWord();
849     }
850
851     public CharArraySet readFile(String filename)
852     {
853       CharArraySet setData = new CharArraySet(0, true);
854       try
855       {
856                         FileInputStream fis=new FileInputStream(filename);
857       Scanner sc=new Scanner(fis);
858         while(sc.hasNextLine())
859         {
860           setData.add(sc.nextLine());
861         }
862       sc.close();
863       }
864         catch(IOException e)
865       {
866         e.printStackTrace();
867       }
868       return setData;
869     }
870
871     public boolean isStopword(String word){
872                         CharArraySet c = indonesianAnalyzer.getStopwordSet();
873
```

```
874      Iterator iter = c.iterator();
875         while(iter.hasNext()) {
876            char[] stopWord = (char[]) iter.next();
877            String stopword = new String (stopWord);
878
879            if(word.equals(new String (stopWord))){
880               return true;
881            }
882         }
883       return false;
884    }
885
886    public void lastStopWord() throws SQLException,
IOException{
887             String   query   =   "SELECT   *   FROM
`tokenize_trainer`";
888
889      connection = SQLConnection.getConnection();
890                                    statement          =
connection.prepareStatement(query);
891
892        ArrayList<StopWord> listStopWords = new
ArrayList<StopWord>();
893      List<String> words = new ArrayList<String>();
894
895                                ResultSet          rs          =
statement.executeQuery(query);
896         while (rs.next()) {
897            StopWord sw = new StopWord();
898            sw.setstid(rs.getString("cf_id"));
899            sw.settitle(rs.getString("title"));
900
sw.setdeskripsi((rs.getString("deskripsi")));
901            sw.setkategori(rs.getString("kategori"));
902
903            if(!isStopword(sw.getdeskripsi())){
904               listStopWords.add(sw);
905            }
906         }
907         insertStopWord(listStopWords);
908    }
909
910    public                                                 void
insertStopWord(ArrayList<StopWord>   listStopWords)
throws SQLException, IOException
911    {
912       String query= "insert into stopword_trainer
(st_id,title,deskripsi,kategori) values(?,?,?,?)";
913      connection = SQLConnection.getConnection();
914                                          statement          =
connection.prepareStatement(query);
915
916      for (int i = 0; i < listStopWords.size(); i++) {
917
statement.setString(1,listStopWords.get(i).getstid());
918
statement.setString(2,listStopWords.get(i).gettitle());
```

```
statement.setString(3,listStopWords.get(i).getdeskripsi());
statement.setString(4,listStopWords.get(i).getkategori());
            statement.addBatch();
        }

    try {
            statement.executeBatch();
        } catch (SQLException e) {
            System.out.println("Error message: " + e.getMessage());
            return;
        }
    }
}
```

TestingData.java

```
public class TestingData {
private String web_scraper_order, testing_id, title, author, deskripsi, kategori;

public TestingData() {}

public TestingData(String web_scraper_order, String training_id, String title, String author, String deskripsi, String kategori) {
    this.web_scraper_order = web_scraper_order;
    this.testing_id = training_id;
    this.title = title;
    this.author = author;
    this.deskripsi = deskripsi;
}

public String getKategori() {
    return kategori;
}

public void setKategori(String kategori) {
    this.kategori = kategori;
}

public String getWeb_scraper_order() {
    return web_scraper_order;
}

public void setWeb_scraper_order(String web_scraper_order) {
    this.web_scraper_order = web_scraper_order;
}

public String getTesting_id() {
    return testing_id;
}

public void setTesting_id(String training_id) {
    this.testing_id = training_id;
```

```
967     }
968
969     public String getTitle() {
970       return title;
971     }
972
973     public void setTitle(String title) {
974       this.title = title;
975     }
976
977     public String getAuthor() {
978       return author;
979     }
980
981     public void setAuthor(String author) {
982       this.author = author;
983     }
984
985     public String getDeskripsi() {
986       return deskripsi;
987     }
988
989     public void setDeskripsi(String deskripsi) {
990       this.deskripsi = deskripsi;
991     }
992     }
```

TestingMenu.java

```
993     public class TestingMenu extends
javax.swing.JFrame{
994     private static final long serialVersionUID = 1L;
995
996     JScrollPane jScrollPane1;
997     JFrame frame;
998     JButton go, back, browse, save, savetotrain;
999     private JTextField textPath;
1000    public JTextArea textBox;
1001
1002    PreparedStatement statement;
1003    Connection connection;
1004    public ArrayList<TestingData> listHasilData;
1005
1006    public TestingMenu() throws Exception {
1007      frame=new JFrame();
1008                  Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
1009     Dimension windowSize = frame.getSize();
1010      int windowX = Math.max(0, (screenSize.width
- windowSize.width ) / 2);
1011     int windowY = Math.max(0, (screenSize.height
- windowSize.height) / 2);
1012
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
E);
1013
1014     frame.setTitle("Add Testing Data");
1015       frame.setLocation(windowX-500/2,  windowY-
500/2);
1016     frame.setVisible(true);
```

```
1017    JPanel setLayout = setLayout();
1018
1019            frame.getContentPane().add(setLayout(),
BorderLayout.CENTER);
1020    frame.pack();
1021
1022    System.out.println("Masuk Testing Menu");
1023  }
1024
1025  public JLabel LabelPath(String l,int alignment){
1026   JLabel label = new JLabel(l,alignment);
1027   return label;
1028  }
1029
1030  public JLabel LabelResult(String l,int alignment)
{
1031   JLabel label = new JLabel(l,alignment);
1032   return label;
1033  }
1034
1035  public JPanel setLayout(){
1036    initComponents();
1037
1038            JPanel panel1 = new JPanel(new
FlowLayout(FlowLayout.CENTER));
1039    panel1.add(LabelPath("PATH :", 0));
1040    panel1.add(textPath);
1041    panel1.add(browse);
1042    JPanel centerPane1=new JPanel();
1043                    centerPane1.setLayout(new
BoxLayout(centerPane1,BoxLayout.Y_AXIS));
1044    centerPane1.add(panel1);
1045
1046            JPanel panel2 = new JPanel(new
FlowLayout(FlowLayout.CENTER));
1047    panel2.add(save);
1048    panel2.add(go);
1049
1050    JPanel centerPane2 = new JPanel();
1051                    centerPane2.setLayout(new
BoxLayout(centerPane2,BoxLayout.Y_AXIS));
1052    centerPane2.add(panel2);
1053
1054            JPanel panel3 = new JPanel(new
FlowLayout(FlowLayout.CENTER));
1055    panel3.add(back);
1056    JPanel centerPane3 = new JPanel();
1057                    centerPane3.setLayout(new
BoxLayout(centerPane3,BoxLayout.Y_AXIS));
1058    centerPane3.add(panel3);
1059
1060            JPanel panel4 = new JPanel(new
FlowLayout(FlowLayout.LEFT));
1061    panel4.add(LabelResult("RESULT :", 0));
1062    JPanel centerPane4 = new JPanel();
1063                    centerPane4.setLayout(new
BoxLayout(centerPane4,BoxLayout.Y_AXIS));
1064    centerPane4.add(panel4);
```

```
1065
1066            JPanel panel5 = new JPanel(new
FlowLayout(FlowLayout.LEFT));
1067    panel5.add(jScrollPane1);
1068    JPanel centerPane5 = new JPanel();
1069                        centerPane5.setLayout(new
BoxLayout(centerPane5,BoxLayout.Y_AXIS));
1070    centerPane5.add(panel5);
1071
1072    JPanel Pane = new JPanel();
1073                                Pane.setLayout(new
BoxLayout(Pane,BoxLayout.Y_AXIS));
1074    Pane.add(centerPane1);
1075    Pane.add(centerPane2);
1076    Pane.add(centerPane4);
1077    Pane.add(centerPane5);
1078    Pane.add(centerPane3);
1079
1080    return Pane;
1081  }
1082
1083  private void initComponents() {
1084    try {
1085          textPath=new JTextField(30);
1086          browse=new JButton("BROWSE");
1087                      browse.addActionListener(new
ButtonBrowse());
1088
1089          save=new JButton("SAVE");
1090                      save.addActionListener(new
ButtonSave());
1091
1092          go=new JButton("GO");
1093          go.addActionListener(new ButtonGo());
1094
1095          textBox = new JTextArea();
1096          textBox.setEditable(false);
1097
1098          jScrollPane1 = new JScrollPane(textBox);
1099                  jScrollPane1.setPreferredSize(new
Dimension(500, 300));
1100
jScrollPane1.setVerticalScrollBarPolicy(jScrollPane1.VE
RTICAL_SCROLLBAR_ALWAYS);
1101
jScrollPane1.setHorizontalScrollBarPolicy(jScrollPane1.
HORIZONTAL_SCROLLBAR_NEVER);
1102
1103          back=new JButton("Back");
1104                      back.addActionListener(new
ButtonBack());
1105    } catch (Exception e) {
1106                  System.out.println("Exception   in
initUIComponents() = " + e);
1107  }
1108  }
1109
```

```
1110 public class ButtonBrowse implements ActionListener{
1111         public void actionPerformed(ActionEvent event) {
1112         browseCSV();
1113     }
1114 }
1115
1116 public class ButtonSave implements ActionListener{
1117     @Override
1118         public void actionPerformed(ActionEvent event) {
1119         if(textPath.getText().equals("")){
1120                 JOptionPane.showMessageDialog(null, "No File Choiced !", "Error", JOptionPane.ERROR_MESSAGE);
1121         }else
1122             try {
1123                 loadData();
1124             } catch (SQLException ex) {
1125 Logger.getLogger(TestingMenu.class.getName()).log(Level.SEVERE, null, ex);
1126             }
1127     }
1128 }
1129
```

```
1130 public class ButtonGo implements ActionListener{
1131     @Override
1132     public void actionPerformed(ActionEvent e) {
1133         System.out.println("Btn Go");
1134         try {
1135                     CaseFoldingTesting cs = new CaseFoldingTesting();
1136             System.out.println("Case Folding done");
1137
1138                     TokenizeTesting tt = new TokenizeTesting();
1139             System.out.println("Tokenize done");
1140
1141                     StopWordTesting sw = new StopWordTesting();
1142             System.out.println("StopWord done");
1143
1144                     StemmingTesting stm = new StemmingTesting();
1145             System.out.println("Stemming done");
1146
1147             TfIdfTrain ti = new TfIdfTrain();
1148                     HashMap<String,String> hasil = ti.ujiData();
1149             String textdisplay = "";
1150
1151                     String query = "SELECT * FROM `testing_data`";
```

```
1152                                        connection =
SQLConnection.getConnection();
1153                                         statement =
connection.prepareStatement(query);
1154                                    ResultSet rs =
statement.executeQuery( );
1155              listHasilData = new ArrayList<>();
1156              while (rs.next()) {
1157                 TestingData td = new TestingData();
1158                    td.setTesting_id(rs.getString("web-
scraper-order"));
1159                 td.setTitle(rs.getString("title"));
1160                 td.setAuthor(rs.getString("author"));
1161
td.setDeskripsi(rs.getString("deskripsi"));
1162
td.setKategori(hasil.get(td.getTesting_id()));
1163                 listHasilData.add(td);
1164                              textdisplay += "Kategori
\t:"+td.getKategori()+"\n";
1165                               textdisplay += "Book Id
\t:"+td.getTesting_id()+"\n";
1166                                 textdisplay += "Judul
\t:"+td.getTitle()+"\n";
1167                                textdisplay += "Author
\t:"+td.getAuthor()+"\n";
1168                              textdisplay += "Deskripsi
\t:\n"+td.getDeskripsi()+"\n";
1169                                          textdisplay +=
"================\n";
1170              }
1171           textBox.setLineWrap(true);
1172           textBox.setWrapStyleWord(true);
1173           textBox.setText(textdisplay);
1174           savetotrain.setEnabled(true);
1175
1176       } catch (SQLException ex) {
1177
Logger.getLogger(TestingMenu.class.getName()).log(L
evel.SEVERE, null, ex);
1178       } catch (IOException ex) {
1179
Logger.getLogger(TestingMenu.class.getName()).log(L
evel.SEVERE, null, ex);
1180       }
1181    }
1182 }
1183
1184 public      class      ButtonBack      implements
ActionListener{
1185    @Override
1186          public  void  actionPerformed(ActionEvent
event) {
1187
if("Back".equals(event.getActionCommand())) {
1188
1189           frame.setVisible(false);
```

```
        }
    }
}

public boolean check(){
    boolean check =false;
    if(textPath.getText().equals("")){
                JOptionPane.showMessageDialog(null, "No File Choiced !", "Error", JOptionPane.ERROR_MESSAGE);
            check=false;
    }else{
        check=true;
    }
    return check;
}

private void browseCSV(){
    JFileChooser jfc = new JFileChooser();
    FileFilter fileFilter;
    fileFilter = new FileFilter() {
        @Override
        public boolean accept(File f) {
                                                return f.getName().toLowerCase().endsWith(".csv") || f.isDirectory();
        }
        @Override
        public String getDescription() {
            return "*.csv";
        }
    };
    jfc.addChoosableFileFilter(fileFilter);
    jfc.setMultiSelectionEnabled(false);
    jfc.showOpenDialog(this);

                                String path = jfc.getSelectedFile().getAbsolutePath();
    textPath.setText(path);
}

private void loadData() throws SQLException{
    connection = SQLConnection.getConnection();
    Statement statement = null;
    String path = validatePath(textPath.getText());
    final String delimiter = ";";
    final String enclosed = "\"";
        final String query = "LOAD DATA INFILE '"+path+"' INTO TABLE testing_data FIELDS TERMINATED BY '"+delimiter+
            "' ENCLOSED BY '"+enclosed+"' LINES TERMINATED BY '\n' IGNORE 1 LINES (`web-scraper-order`, `title`, `author`, `deskripsi`);";

    try {
                                    statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
```

```
1237                                    int result =
statement.executeUpdate(query);
1238        if(result>0){
1239                JOptionPane.showMessageDialog(null,
"Berhasil Menyimpan");
1240        }
1241    } catch (SQLException ex) {
1242        System.out.println(ex.getMessage());
1243    }
1244    catch (Exception e){}
1245  }
1246
1247  private String validatePath(String invalidPath){
1248    String validPath;
1249    validPath = invalidPath.replace('\\', '/');
1250    return validPath;
1251  }
1252  }
TfIdfTesting.java
1253  public class TfIdfTesting {
1254  Connection connection;
1255  PreparedStatement statement;
1256  ArrayList<String>     listDocumentId     =     new
ArrayList<>();
1257  ArrayList<Tf_idf> listTF = new ArrayList<>();
1258  ArrayList<Id_Class> listKategori;
1259  Double countDoc = 0.0;
1260
1261  public  TfIdfTesting()  throws  SQLException,
IOException{
1262    listKategori = indexKategori();
1263    getListTF();
1264  }
1265
1266  public                    HashMap<String,String>
Uji(ArrayList<BobotObject>          LVQ)          throws
SQLException{
1267    HashMap<String, HashMap<String, Double>>
listBobotDocument = getTF();
1268
1269    return testingData(LVQ, listBobotDocument);
1270  }
1271
1272  public  ArrayList<Id_Class>      indexKategori()
throws SQLException{
1273          String  query  =  "SELECT  *  FROM
`casefolding_trainer` order by `id` asc";
1274
1275    connection = SQLConnection.getConnection();
1276                                    statement      =
connection.prepareStatement(query);
1277    ResultSet rs = statement.executeQuery( );
1278    rs = statement.executeQuery();
1279
1280        ArrayList<Id_Class>  kategoriList  =  new
ArrayList<>();
1281    while(rs.next()){
```

```
                kategoriList.add(new Id_Class(rs.getString("kategori"), rs.getString("training_id")));
            }
            return kategoriList;
        }

        public void getListTF() throws SQLException{
            //get t
        String query = "SELECT CT.training_id, CT.kategori, ST.deskripsi, count(ST.id) frekuensi " +
        "FROM casefolding_trainer as CT " +
        "LEFT JOIN stemming_trainer as ST " +
        "ON CT.training_id = ST.tn_id " +
        "GROUP BY ST.deskripsi, CT.id " +
        "order by ST.id";

        connection = SQLConnection.getConnection();
                statement = connection.prepareStatement(query);

            ResultSet rs = statement.executeQuery( );

            while (rs.next()) {
                Tf_idf tf = new Tf_idf();
                tf.setTiId(rs.getString("training_id"));
                tf.setTerm(rs.getString("deskripsi"));
                tf.setFrekuensi(rs.getInt("frekuensi"));
                tf.setKategori(rs.getString("kategori"));

                listTF.add(tf);
            }

        //get count document
            String queryDoc = "SELECT count(*) as count FROM `casefolding_trainer`";

        connection = SQLConnection.getConnection();
                statement = connection.prepareStatement(queryDoc);
            rs = statement.executeQuery();
            while(rs.next()){
                countDoc = rs.getDouble("count");
            }
        }

        public HashMap<String,HashMap<String,Double>> getTF() throws SQLException
        {
            String query = "SELECT * FROM `stemming_trainer` group by `deskripsi` order by `id` asc";

        connection = SQLConnection.getConnection();
                statement = connection.prepareStatement(query);
            ResultSet rs = statement.executeQuery( );
```

```
1329     rs = statement.executeQuery();
1330
1331         ArrayList<String>  listTermTraining  =  new
ArrayList<>();
1332     while(rs.next()){
1333
listTermTraining.add(rs.getString("deskripsi"));
1334     }
1335
1336     //get list document from testing
1337         String  queryListDoc  =  "SELECT  *  FROM
`casefolding_testing` order by `id` asc";
1338
1339  connection = SQLConnection.getConnection();
1340                                  statement          =
connection.prepareStatement(queryListDoc);
1341     rs = statement.executeQuery();
1342
1343     while(rs.next()){
1344
listDocumentId.add(rs.getString("testing_id"));
1345     }
1346
1347       HashMap<String,HashMap<String,Double>>
listBobotDocument = new HashMap<>();
1348     for (int i = 0; i < listDocumentId.size(); i++) {
1349         //get term
1350               String  queryTerm  =  "SELECT  *  FROM
`stemming_testing` "
1351                                        +  "where  `tn_id`  =
'"+listDocumentId.get(i)+"' group by `deskripsi` order
by `id` asc ";
1352                                    connection      =
SQLConnection.getConnection();
1353                                    statement       =
connection.prepareStatement(queryTerm);
1354         rs = statement.executeQuery();
1355
1356  ArrayList<String>      listTermTest      =      new
ArrayList<>();
1357         while(rs.next()){
1358
listTermTest.add(rs.getString("deskripsi"));
1359         }
1360
1361                  ArrayList<Tf_idf>  listTF  =  new
ArrayList<>();
1362               System.err.println("doc  testing  id  :
"+listDocumentId.get(i));
1363
1364         HashMap<String,Double> listBobot = new
HashMap<>();
1365         //get tf
1366         for (int j = 0; j < listTermTraining.size(); j++)
{
1367           int tftemp = 0;
1368           for (int k = 0; k < listTermTest.size(); k++)
{
```

```
                                                    if
(listTermTraining.get(j).equals(listTermTest.get(k))) {
                    tftemp = tftemp + 1;
                }
            }

  Tf_idf tf = new Tf_idf();
            tf.setTiId(listDocumentId.get(i));
            tf.setTerm(listTermTraining.get(j));
            tf.setFrekuensi(tftemp);

            listTF.add(tf);
            //print tf testing data
  //                System.err.println("tf : "+tftemp+"
term :"+listTermTraining.get(j));

            //hitung idf
                                            double    idf    =
hitungIDF(listTermTraining.get(j));

            //hitung tf-idf
            double tfidf = tftemp * idf;
                    listBobot.put(listTermTraining.get(j),
tfidf);
  //              System.err.println("w: "+tfidf+" ");
        }

            listBobotDocument.put(listDocumentId.get(i),
listBobot);
        }

    return listBobotDocument;
    }

    public Double hitungIDF(String term){

        HashMap<String,ArrayList<Tf_idf>> listTfMap
= new HashMap<>();
        ArrayList<Tf_idf> temp = new ArrayList<>();

        for (int j=0; j<listTF.size(); j++){
            if(term.equals(listTF.get(j).getTerm())){
                Tf_idf tf = new Tf_idf();
                tf.setTiId(listTF.get(j).getTiId());

tf.setFrekuensi(listTF.get(j).getFrekuensi());
                temp.add(tf);
                listTfMap.put(term, temp);
  //
System.err.println(term+listTF.get(j).getTerm()
+temp.size()+"\n");
            }
        }
        //jumlah dokumen yg ada termnya
        int df = listTfMap.get(term).size();
```

```
1415
1416    //banyak dokumen atau hitungan idf
1417    Double idf = Math.log10(countDoc/df);
1418
1419      return idf;
1420  }
1421
1422  public                          HashMap<String,String>
testingData(ArrayList<BobotObject> LVQ,
1423
HashMap<String,HashMap<String,Double>>
listBobotDocument ){
1424
1425       HashMap<String,String>  hasilTest  =  new
HashMap<>();
1426     for (int i = 0; i < listBobotDocument.size(); i+
+) {
1427                                  double[]    w    =    new
double[LVQ.get(0).getListBobot().size()];
1428       for (int j = 0; j < LVQ.size(); j++) {
1429                                 for   (int   k   =   0;   k   <
LVQ.get(j).getListBobot().size(); k++) {
1430
1431                                        double   wtemp   =
listBobotDocument.get(listDocumentId.get(i))
1432                      .get(LVQ.get(j).getTerm()) - LVQ
1433
.get(j).getListBobot().get(k).getW();
1434              wtemp = wtemp*wtemp;
1435              w[k] += wtemp;
1436           }
1437       }
1438       // hitung akar eucclidean distance
1439       for(int l = 0 ; l < w.length;l++){
1440          w[l] = Math.sqrt(w[l]);
1441          System.err.println("w : "+w[l]);
1442       }
1443       int index =  FindFirstMinIndex(w);
1444                                          String     wClass     =
LVQ.get(0).getListBobot().get(index).getId();
1445       String kategoriClass= getKatgori(wClass);
1446              System.err.println("Id  Data  Testing  =
"+listDocumentId.get(i)+
1447       " is "+ kategoriClass);
1448  hasilTest.put(listDocumentId.get(i),
kategoriClass);
1449    }
1450    return hasilTest;
1451  }
1452
1453  public int FindFirstMinIndex(double[] w)
1454  {
1455    int minIndex = 0;
1456    for (int i = 1; i < w.length; i++)
1457       if(w[i] < w[minIndex])
1458          minIndex=i;
1459    return minIndex;
1460  }
```

```
1461
1462  public String getKatgori(String findString){
1463
1464    String kategori = "Notfound";
1465    for (int j = 0; j < listKategori.size(); j++) {
1466                                                    if
(findString.equals(listKategori.get(j).getId())) {
1467                                                kategori =
listKategori.get(j).getKategori();
1468              break;
1469          }
1470       }
1471    return kategori;
1472  }
1473  }
TfIdfTrain.java
1474  public class TfIdfTrain {
1475  Connection connection;
1476  PreparedStatement statement;
1477
1478  HashMap<Integer,ObjectSum> listSum = new
HashMap<>();
1479  ArrayList<String>      listKategori      =      new
ArrayList<>();
1480  ArrayList<String>      listDocument      =      new
ArrayList<>();
1481
1482  public     TfIdfTrain()     throws     SQLException,
IOException{}
1483
1484  public ArrayList<BobotObject> getTF() throws
SQLException, IOException
1485  {
1486    //get tf
1487          String query = "SELECT CT.training_id,
CT.kategori, ST.deskripsi, count(ST.id) frekuensi " +
1488  "FROM casefolding_trainer as CT " +
1489  "LEFT JOIN stemming_trainer as ST " +
1490  "ON CT.training_id = ST.tn_id " +
1491  "GROUP BY ST.deskripsi, CT.id " +
1492  "order by ST.id";
1493
1494  connection = SQLConnection.getConnection();
1495                                        statement
=
connection.prepareStatement(query);
1496
1497    ArrayList<Tf_idf> listTF = new ArrayList<>();
1498
1499    ResultSet rs = statement.executeQuery( );
1500
1501    while (rs.next()) {
1502       Tf_idf tf = new Tf_idf();
1503       tf.setTiId(rs.getString("training_id"));
1504       tf.setTerm(rs.getString("deskripsi"));
1505       tf.setFrekuensi(rs.getInt("frekuensi"));
1506       tf.setKategori(rs.getString("kategori"));
1507
1508       listTF.add(tf);
```

```
1509     }
1510   //get term
1511         String queryTerm = "SELECT * FROM
`stemming_trainer` group by `deskripsi` order by `id`
asc";
1512
1513     connection = SQLConnection.getConnection();
1514                                     statement          =
connection.prepareStatement(queryTerm);
1515     rs = statement.executeQuery();
1516
1517             ArrayList<String>    listTerm    =    new
ArrayList<>();
1518     while(rs.next()){
1519        listTerm.add(rs.getString("deskripsi"));
1520     }
1521     //get count document
1522      String queryDoc = "SELECT count(*) as count
FROM `casefolding_trainer`";
1523
1524     connection = SQLConnection.getConnection();
1525                                     statement          =
connection.prepareStatement(queryDoc);
1526     rs = statement.executeQuery();
1527
1528     Double countDoc = 0.0;
1529      while(rs.next()){
1530        countDoc = rs.getDouble("count");
1531     }
1532   //get list document
1533         String queryListDoc = "SELECT * FROM
`casefolding_trainer` order by `id` asc";
1534
1535     connection = SQLConnection.getConnection();
1536                                     statement          =
connection.prepareStatement(queryListDoc);
1537     rs = statement.executeQuery();
1538
1539     while(rs.next()){
1540
listDocument.add(rs.getString("training_id"));
1541         listKategori.add(rs.getString("kategori"));
1542     }
1543     //get frekuensi kata per dokumen
1544        ArrayList<BobotObject> listBobotObjects =
new ArrayList<>();
1545      HashMap<String,ArrayList<Tf_idf>> listTfMap
= new HashMap<>();
1546
1547   //                System.err.println(listTerm.size()
+"-"+listTF.size()+"\n");
1548
1549                   double[]      sumw      =      new
double[countDoc.intValue()];
1550
1551     for (int i=0; i<listTerm.size(); i++){
1552         String term = listTerm.get(i);
```

```
1553                    ArrayList<Tf_idf> temp = new
ArrayList<>();
1554        for (int j=0; j<listTF.size(); j++){
1555            if(term.equals(listTF.get(j).getTerm())){
1556                Tf_idf tf = new Tf_idf();
1557                tf.setTiId(listTF.get(j).getTiId());
1558
tf.setFrekuensi(listTF.get(j).getFrekuensi());
1559                temp.add(tf);
1560                listTfMap.put(term, temp);
1561  //
System.err.println(term+listTF.get(j).getTerm()
+temp.size()+"\n");
1562            }
1563        }
1564
1565        //jumlah dokumen yg ada termnya atau df
1566        int df = listTfMap.get(term).size();
1567
1568        //banyak dokumen atau hitungan idf
1569        Double idf = Math.log10(countDoc/df);
1570
1571        BobotObject bo = new BobotObject();
1572        bo.setTf(Double.valueOf(df));
1573        bo.setDf(idf);
1574        bo.setListTf(listTfMap);
1575
1576                ArrayList<Bobot> listBobot = new
ArrayList<Bobot>();
1577        System.err.println("\n"+term);
1578
1579        for(int k = 0; k < countDoc;k++){
1580            Bobot bobot = new Bobot();
1581            Double tf = 0.0;
1582
1583            //dpt tf
1584                                for( int l = 0 ; l <
listTfMap.get(term).size(); l++){
1585  //
System.out.println(listTfMap.get(term).get(l).getTiId()
+" "+listDocument.get(k));
1586
if(listTfMap.get(term).get(l).getTiId().equals(listDocum
ent.get(k))){
1587                                                    //
System.out.println(listTf2.get(term).get(l).getId());
1588                                                    tf =
Double.valueOf(listTfMap.get(term).get(l).getFrekuens
i());
1589  //                  System.out.println(tf+"-"+df);
1590                }
1591            }
1592            //perkalian tf*idf
1593            Double w = tf * idf;
1594            bobot.setId(listDocument.get(k));
1595            bobot.setW(w);
1596            listBobot.add(bobot);
1597
```

```
1598            //hasil tfidf per dokumen
1599                                    System.err.print("   id   :
"+listDocument.get(k)+" w :"+w +" ");
1600
1601            //hitung semua jumlah term
1602            ObjectSum sum = new ObjectSum();
1603            sum.setDocId(listDocument.get(k));
1604            sum.setkategori(listKategori.get(k));
1605            sumw[k] += w;
1606            sum.setSum(sumw[k]);
1607            listSum.put(k, sum);
1608        }
1609
1610        System.err.println("");
1611        bo.setListBobot(listBobot);
1612        bo.setTerm(term);
1613        listBobotObjects.add(bo);
1614    } //end looping tfidf
1615
System.err.println(""+listBobotObjects.get(0).getListB
obot().size());
1616                    ArrayList<String>       ranking     =
rankingSum(listSum);
1617    //panggil class Classification
1618    Classification lvq = new Classification();
1619    ArrayList<BobotObject> LVQ = lvq.LVQ(0.1, 50,
listBobotObjects, ranking);
1620
1621    return LVQ;
1622 }
1623 //panggil tfidf testing
1624 public HashMap<String,String> ujiData() throws
SQLException, IOException{
1625    TfIdfTesting ujicoba = new TfIdfTesting();
1626    ArrayList<BobotObject> LVQ = getTF();
1627    return ujicoba.Uji(LVQ);
1628 }
1629 //ranking data
1630 // semua data per kategori di-sum terus yg
paling besar jadiin representatif
1631 public                                  ArrayList<String>
rankingSum(HashMap<Integer,ObjectSum> listSum){
1632          ArrayList<String>  hasilranking  =  new
ArrayList<>();
1633          ArrayList<String>  listKategori2  =  new
ArrayList<>(new HashSet<>(listKategori));
1634
1635    for (int n = 0; n < listKategori2.size(); n++) {
1636       double temp =0;
1637       String tempid="";
1638       for(int m = 0; m < listSum.size(); m++){
1639                                                          if
(listKategori2.get(n).equals(listSum.get(m).getkategor
i())) {
1640             if (temp<listSum.get(m).getSum()) {
1641                temp=listSum.get(m).getSum();
1642                tempid=listSum.get(m).getDocId();
1643             }
```

```
1644            }
1645        }
1646        hasilranking.add(tempid);
1647            System.err.println("Initiation Weight for
Training");
1648            System.err.println("tempid = "+tempid+"
sum="+temp);
1649     }
1650     return hasilranking;
1651  }
1652  }
Tf_idf.java
1653  public class Tf_idf {
1654  int frekuensi;
1655  String ti_id,term,kategori;
1656
1657  public Tf_idf(){}
1658
1659  public  Tf_idf(int  frekuensi,String  ti_id,  String
term, String kategor){
1660     this.frekuensi = frekuensi;
1661     this.ti_id = ti_id;
1662     this.term = term;
1663     this.kategori = kategori;
1664  }
1665
1666  public void setTiId(String ti_id) {
1667     this.ti_id = ti_id;
1668  }
1669
1670  public String getTiId() {
1671     return ti_id;
1672  }
1673
1674  public void setFrekuensi(int frekuensi) {
1675     this.frekuensi = frekuensi;
1676  }
1677
1678  public int getFrekuensi() {
1679     return frekuensi;
1680  }
1681
1682  public void setTerm(String term) {
1683     this.term = term;
1684  }
1685
1686  public String getTerm() {
1687     return term;
1688  }
1689
1690  public void setKategori(String kategori) {
1691     this.kategori = kategori;
1692  }
1693
1694  public String getKategori() {
1695     return kategori;
1696  }
1697  }
```

Tokenize.java

```
1698  public class Tokenize {
1699  private String cf_id, title, deskripsi, kategori;
1700
1701  public Tokenize(){}
1702
1703  public Tokenize(String cf_id, String title, String
deskripsi, String kategori){
1704    this.cf_id=cf_id;
1705    this.title=title;
1706    this.deskripsi=deskripsi;
1707    this.kategori=kategori;
1708  }
1709
1710  public void setcf_id(String cf_id){
1711    this.cf_id=cf_id;
1712  }
1713
1714  public String getcf_id(){
1715    return cf_id;
1716  }
1717
1718  public void settitle(String title){
1719    this.title=title;
1720  }
1721
1722  public String gettitle(){
1723    return title;
1724  }
1725
1726  public void setdeskripsi(String deskripsi){
1727    this.deskripsi=deskripsi;
1728  }
1729
1730  public String getdeskripsi(){
1731    return deskripsi;
1732  }
1733
1734  public void setkategori(String kategori){
1735    this.kategori=kategori;
1736  }
1737
1738  public String getkategori(){
1739    return kategori;
1740  }
1741  }
```

TokenizeTesting.java

```
1742  public class TokenizeTesting {
1743  PreparedStatement statement;
1744  public StringTokenizer token;
1745  public String kata = "";
1746  public ArrayList<String> wordsdesc;
1747  public String cf_id,title,deskripsi;
1748  public String words_desc="";
1749  Connection connection;
1750
1751  public TokenizeTesting() throws SQLException,
IOException{
```

```
1752     lastTokenize();
1753   }
1754   public void lastTokenize() throws SQLException,
IOException{
1755             String query = "SELECT * FROM
`casefolding_testing";
1756
1757     connection = SQLConnection.getConnection();
1758                                   statement =
connection.prepareStatement(query);
1759
1760         ArrayList<Tokenize> listTokenize = new
ArrayList<Tokenize>();
1761                               ResultSet rs =
statement.executeQuery(query);
1762
1763         while (rs.next()) {
1764           Tokenize tokenize = new Tokenize();
1765
tokenize.setcf_id(rs.getString("testing_id"));
1766           tokenize.settitle(rs.getString("title"));
1767
tokenize.setdeskripsi(rs.getString("deskripsi"));
1768           listTokenize.add(tokenize);
1769         }
1770         Tokening(listTokenize);
1771   }
1772
1773   public                              ArrayList<String>
Tokening(ArrayList<Tokenize>  listTokenize)  throws
SQLException {
1774
1775       for (int i = 0; i < listTokenize.size(); i++) {
1776                                 String hapusNewline =
listTokenize.get(i).getdeskripsi().replace(System.getPr
operty("line.separator"), "");
1777                     ArrayList<String> katadesc = new
ArrayList<String>();
1778                                              token    =    new
StringTokenizer(hapusNewline, " ");
1779
1780         System.out.println(token.countTokens());
1781
1782         while (token.hasMoreTokens()) {
1783           insertTokenDesc(
1784                   listTokenize.get(i).getcf_id(),
1785                   listTokenize.get(i).gettitle(),
1786                   token.nextToken());
1787         }
1788       }
1789     return wordsdesc;
1790   }
1791
1792   private void insertTokenDesc(String cf_id, String
title,String deskripsi) throws SQLException{
1793       String query= "insert into tokenize_testing
(cf_id,title,deskripsi) values(?,?,?)";
```

```
1794                                    statement =
connection.prepareStatement(query);
1795    statement.setString(1,cf_id);
1796    statement.setString(2,title);
1797    statement.setString(3,deskripsi);
1798    statement.addBatch();
1799
1800    try {
1801          statement.executeBatch();
1802       } catch (SQLException e) {
1803             System.out.println("Error message: " +
e.getMessage());
1804          return;
1805       }
1806  }
1807  }
TokenizeTraining.java
1808  public class TokenizeTraining {
1809  PreparedStatement statement;
1810  public StringTokenizer token;
1811  public String kata = "";
1812  public ArrayList<String> wordsdesc;
1813  public String cf_id,title,deskripsi,kategori;
1814  public String words_desc="";
1815  Connection connection;
1816
1817  public TokenizeTraining() throws SQLException,
IOException{
1818     lastTokenize();
1819  }
1820  public void lastTokenize() throws SQLException,
IOException{
1821          String query = "SELECT * FROM
`casefolding_trainer`";
1822
1823    connection = SQLConnection.getConnection();
1824                                    statement =
connection.prepareStatement(query);
1825
1826       ArrayList<Tokenize> listTokenize = new
ArrayList<Tokenize>();
1827                              ResultSet rs =
statement.executeQuery(query);
1828
1829       while (rs.next()) {
1830          Tokenize tokenize = new Tokenize();
1831
tokenize.setcf_id(rs.getString("training_id"));
1832          tokenize.settitle(rs.getString("title"));
1833
tokenize.setdeskripsi(rs.getString("deskripsi"));
1834
tokenize.setkategori(rs.getString("kategori"));
1835          listTokenize.add(tokenize);
1836       }
1837       Tokening(listTokenize);
1838  }
1839
```

```
1840  public ArrayList<String> Tokening(ArrayList<Tokenize> listTokenize) throws SQLException {
1841
1842      for (int i = 0; i < listTokenize.size(); i++) {
1843          String hapusNewline = listTokenize.get(i).getdeskripsi().replace(System.getProperty("line.separator"), "");
1844          ArrayList<String> katadesc = new ArrayList<String>();
1845          token = new StringTokenizer(hapusNewline, " ");
1846
1847          System.out.println(token.countTokens());
1848
1849          while (token.hasMoreTokens()) {
1850              insertTokenDesc(
1851                  listTokenize.get(i).getcf_id(),
1852                  listTokenize.get(i).gettitle(),
1853                  token.nextToken(),
1854                  listTokenize.get(i).getkategori());
1855      }
1856    }
1857    return wordsdesc;
1858  }
1859
1860  private void insertTokenDesc(String cf_id, String title,String deskripsi,String kategori) throws SQLException{
1861      String query= "insert into tokenize_trainer (cf_id,title,deskripsi,kategori) values(?,?,?,?)";
1862          statement = connection.prepareStatement(query);
1863    statement.setString(1,cf_id);
1864    statement.setString(2,title);
1865    statement.setString(3,deskripsi);
1866    statement.setString(4,kategori);
1867    statement.addBatch();
1868
1869    try {
1870        statement.executeBatch();
1871      } catch (SQLException e) {
1872          System.out.println("Error message: " + e.getMessage());
1873          return;
1874      }
1875  }
1876  }
```

TrainingMenu.java

```
1877  public class TrainingMenu extends javax.swing.JFrame{
1878  private static final long serialVersionUID = 1L;
1879  private static Connection connection;
1880
1881  JScrollPane jScrollPane1;
1882  JFrame frame;
1883  JButton go, back, browse, save;
1884  private JTextField textPath;
```

```
1885
1886  public TrainingMenu() throws Exception {
1887     frame=new JFrame();
1888                        Dimension       screenSize       =
Toolkit.getDefaultToolkit().getScreenSize();
1889     Dimension windowSize = frame.getSize();
1890      int windowX = Math.max(0, (screenSize.width
- windowSize.width ) / 2);
1891     int windowY = Math.max(0, (screenSize.height
- windowSize.height) / 2);
1892
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
E);
1893
1894     frame.setTitle("Add Training Data");
1895        frame.setLocation(windowX-500/2,  windowY-
500/2);
1896     frame.setVisible(true);
1897     JPanel setLayout = setLayout();
1898
1899             frame.getContentPane().add(setLayout(),
BorderLayout.CENTER);
1900     frame.pack();
1901
1902     System.out.println("Masuk Training Menu");
1903  }
1904
1905  public JLabel MyLabel(String l,int alignment){
1906    JLabel label = new JLabel(l,alignment);
1907    return label;
1908  }
1909
1910  public JPanel setLayout(){
1911    initComponents();
1912
1913             JPanel    panel1    =    new    JPanel(new
FlowLayout(FlowLayout.CENTER));
1914     panel1.add(MyLabel("PATH :", 0));
1915     panel1.add(textPath);
1916     panel1.add(browse);
1917     JPanel centerPane1=new JPanel();
1918                                centerPane1.setLayout(new
BoxLayout(centerPane1,BoxLayout.Y_AXIS));
1919     centerPane1.add(panel1);
1920
1921             JPanel    panel2    =    new    JPanel(new
FlowLayout(FlowLayout.CENTER));
1922     panel2.add(save);
1923     JPanel centerPane2 = new JPanel();
1924                                centerPane2.setLayout(new
BoxLayout(centerPane2,BoxLayout.Y_AXIS));
1925     centerPane2.add(panel2);
1926
1927             JPanel    panel3    =    new    JPanel(new
FlowLayout(FlowLayout.CENTER));
1928     panel3.add(go);
1929     panel3.add(back);
1930     JPanel centerPane3 = new JPanel();
```

```
1931                        centerPane3.setLayout(new
BoxLayout(centerPane3,BoxLayout.Y_AXIS));
1932    centerPane3.add(panel3);
1933
1934    JPanel Pane = new JPanel();
1935                                Pane.setLayout(new
BoxLayout(Pane,BoxLayout.Y_AXIS));
1936    Pane.add(centerPane1);
1937    Pane.add(centerPane2);
1938    Pane.add(centerPane3);
1939    return Pane;
1940  }
1941
1942  private void initComponents() {
1943    try {
1944          textPath=new JTextField(30);
1945          browse=new JButton("BROWSE");
1946                      browse.addActionListener(new
ButtonBrowse());
1947
1948          save=new JButton("SAVE");
1949                        save.addActionListener(new
ButtonSave());
1950
1951          go=new JButton("GO");
1952          go.addActionListener(new ButtonGo());
1953
1954          back=new JButton("Back");
1955                        back.addActionListener(new
ButtonBack());
1956    } catch (Exception e) {
1957                System.out.println("Exception in
initUIComponents() = " + e);
1958  }
1959  }
1960
1961  public class ButtonBrowse implements
ActionListener{
1962        public void actionPerformed(ActionEvent
event) {
1963      browseCSV();
1964    }
1965  }
1966
1967  public class ButtonSave implements
ActionListener{
1968    @Override
1969        public void actionPerformed(ActionEvent
event) {
1970      System.err.println("Button Save");
1971      if(textPath.getText().equals("")){
1972              JOptionPane.showMessageDialog(null,
"No File Choiced !", "Error",
JOptionPane.ERROR_MESSAGE);
1973      }else
1974        try {
1975          loadData();
```

```
1976            } catch (SQLException ex) {
1977 
Logger.getLogger(TrainingMenu.class.getName()).log(L
evel.SEVERE, null, ex);
1978            }
1979        }
1980    }
1981 
1982    public        class        ButtonGo        implements
ActionListener{
1983        @Override
1984        public void actionPerformed(ActionEvent e) {
1985            System.out.println("Btn Go");
1986            try {
1987                        CaseFoldingTraining    cs    =    new
CaseFoldingTraining();
1988            System.out.println("Case Folding done");
1989 
1990                           TokenizeTraining    tt    =    new
TokenizeTraining();
1991            System.out.println("Tokenize done");
1992 
1993                         StopWordTraining    sw    =    new
StopWordTraining();
1994            System.out.println("StopWord done");
1995 
1996                        StemmingTraining    stm    =    new
StemmingTraining();
1997            System.out.println("Stemming done");
1998 
1999            TfIdfTrain ti = new TfIdfTrain();
2000            ti.getTF();
2001            System.out.println("ti done");
2002 
2003                JOptionPane.showMessageDialog(null,
"Selesai");
2004 
2005            } catch (SQLException ex) {
2006 
Logger.getLogger(TrainingMenu.class.getName()).log(L
evel.SEVERE, null, ex);
2007            } catch (IOException ex) {
2008 
Logger.getLogger(TrainingMenu.class.getName()).log(L
evel.SEVERE, null, ex);
2009            }
2010        }
2011    }
2012 
2013    public        class        ButtonBack        implements
ActionListener{
2014        @Override
2015            public    void    actionPerformed(ActionEvent
event) {
2016 
if("Back".equals(event.getActionCommand())) {
2017 
2018            frame.setVisible(false);
```

```
        }
    }
}

public boolean check(){
    boolean check =false;
    if(textPath.getText().equals("")){
                JOptionPane.showMessageDialog(null, "No File Choiced !", "Error", JOptionPane.ERROR_MESSAGE);
            check=false;
    }else{
        check=true;
    }
    return check;
}

private void browseCSV(){
    JFileChooser jfc = new JFileChooser();
    FileFilter fileFilter;
    fileFilter = new FileFilter() {
    @Override
    public boolean accept(File f) {
                return f.getName().toLowerCase().endsWith(".csv") || f.isDirectory();
    }
    @Override
    public String getDescription() {
        return "*.csv";
    }
};
    jfc.addChoosableFileFilter(fileFilter);
    jfc.setMultiSelectionEnabled(false);
    jfc.showOpenDialog(this);

    //import path file
                String path = jfc.getSelectedFile().getAbsolutePath();
    textPath.setText(path);
}

private void loadData() throws SQLException{
    connection = SQLConnection.getConnection();
    Statement statement = null;
    String path = validatePath(textPath.getText());
    final String delimiter = ";";
    final String enclosed = "\"";
        final String query = "LOAD DATA INFILE '"+path+"' INTO TABLE training_data FIELDS TERMINATED BY '"+delimiter+
            "' ENCLOSED BY '"+enclosed+"' LINES TERMINATED BY '\n' IGNORE 1 LINES (`web-scraper-order`, `title`, `author`, `deskripsi`, `kategori`);";

try {
```

```
2066                                    statement       =
connection.createStatement(ResultSet.TYPE_SCROLL_S
ENSITIVE, ResultSet.CONCUR_UPDATABLE);
2067                                int       result      =
statement.executeUpdate(query);
2068        if(result>0){
2069                JOptionPane.showMessageDialog(null,
"Berhasil Menyimpan");
2070        }
2071    } catch (SQLException ex) {
2072        System.out.println(ex.getMessage());
2073    }
2074    catch (Exception e){}
2075  }
2076
2077  private String validatePath(String invalidPath){
2078    String validPath;
2079    validPath = invalidPath.replace('\\', '/');
2080    return validPath;
2081  }
2082  }
```

Report #11030550

Introduction Background In this globalization era, internet technology has grown rapidly and many things can be done using this technology. People can get many informations and facilities so fast is a prove that this technology has become a daily activity. With easy access, many people used internet technology as a business support. Hence this day, many bookstores offer online book sales to make it easier for people to get more information. But to be able to get real information that really needed, then users should be able to sort any websites which can be accessed and provide the right information. However many bookstore websites showed uncategorized books so the search was ineffective because it takes a long time. To solve that problem, those books need to be classified into some categories so users can easily find the right book that needed. Book categories can be determined based on its description because some books at the same categories have similar description each of it. This final project will use a machine learning algorithm which is Learning Vector Quantization (LVQ). LVQ algorithm will learn characteristic of categorized book descriptions on training datas and classify the test data as from LVQ training data before. And finally the book will be classified according to category that matches with the training datas using the learning logic from the algorithm, (case studies