CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

This project is implemented in Java Language. In this sub-chapter will explain how to use the program and how the program works. There are two steps in this program, which is : training data and testing data.

5.1.1 Training Data

In this section, the program imported data that will be used as training data in the classification process. The form of data is CSV file then imported into database.

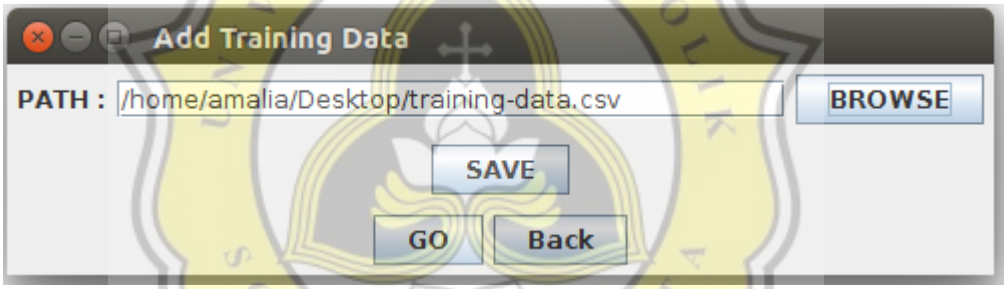

Illustration 5.1: Training Data Graphic Interface

```
1.public void actionPerformed(ActionEvent e) {
2.    System.out.println("Btn Go");
3.      try {
4.       CaseFoldingTraining cs = new CaseFoldingTraining();
5.       System.out.println("Case Folding done");
6.
7.       TokenizeTraining tt = new TokenizeTraining();
8.       System.out.println("Tokenize done");
9.
10.         StopWordTraining sw = new StopWordTraining();
11.         System.out.println("StopWord done");
12.
```

```
13.         StemmingTraining stm = new StemmingTraining();
14.         System.out.println("Stemming done");
15.
16.       TfIdfTrain ti = new TfIdfTrain();
17.       ti.getTF();
18.       System.out.println("ti done");
19.       JOptionPane.showMessageDialog(null, "Selesai");
20.           } catch (SQLException ex) {
21.
Logger.getLogger(TrainingMenu.class.getName()).log(Level.SEVERE,
null, ex);
22.           } catch (IOException ex) {
23.
Logger.getLogger(TrainingMenu.class.getName()).log(Level.SEVERE,
null, ex);
24.           }
25.}
```

Like the illustration and codes above, there are several steps that needed to do before proceeding to the classification process. First the training data will be imported and saved into the database. After that the program will call the CaseFoldingTraining class (line 4) which functions to convert all characters to lowercase character and delete numberic and other characters. Then the program will call TokenizeTraining class (line 7) to parsing all word contents. Next process are Stop Word and Stemming, StopWordTraining class (line 10) is used to delete all stopword and StemmingTraining class (line 13) is used to stem word of contents. Function of those steps is to processing text into more accurate data which called Text Preprocessing. The next step is TF-IDF used to calculate TF-IDF value of training data.

```
26.       int df = listTfMap.get(term).size();
27.
28.       //banyak dokumen atau hitungan idf
29.       Double idf = Math.log10(countDoc/df);
30.
31.       BobotObject bo = new BobotObject();
```

```
32.     bo.setTf(Double.valueOf(df));
33.     bo.setDf(idf);
34.     bo.setListTf(listTfMap);
35.
36.     //perkalian tf*idf
37.     ArrayList<Bobot> listBobot = new ArrayList<Bobot>();
38.     System.err.println("\n"+term);
39.
40.     for(int k = 0; k < countDoc;k++){
41.      Bobot bobot = new Bobot();
42.      Double tf = 0.0;
43.
44.      for( int l = 0 ; l < listTfMap.get(term).size(); l++){
45.
f(listTfMap.get(term).get(l).getTiId().equals(listDocument.get(k))){
46.              tf =
Double.valueOf(listTfMap.get(term).get(l).getFrekuensi());
47.
48.          }
49.      }
50.
51.      Double w = tf * idf;
52.      bobot.setId(listDocument.get(k));
53.      bobot.setW(w);
54.      listBobot.add(bobot);
```

The code above show how TF-IDF function works in TfIdfTrain class. The function will resulting TF-IDF values at each word of description within the data's id. At line 26, it is used to count the number of documents (df) that contains the same term/word. After that at line 29, IDF is calculated using the logarithmic formula. At line 51, the results of the IDF calculation will be multiplied by the value of tf which produces the weight value of each terms.

With the results from TF-IDF process, it will find representation data from each categories for the initiation weight for LVQ algorithm by using text summarization.

```
55.     ObjectSum sum = new ObjectSum();
56.     sum.setDocId(listDocument.get(k));
57.     sum.setkategori(listKategori.get(k));
58.     sumw[k] += w;
59.     sum.setSum(sumw[k]);
60.     listSum.put(k, sum);
61.
62.     public ArrayList<String>
rankingSum(HashMap<Integer,ObjectSum> listSum){
63.     ArrayList<String> hasilranking = new ArrayList<>();
64.
65.     ArrayList<String> listKategori2 = new ArrayList<>(new
HashSet<>(listKategori));
66.
67.     for (int n = 0; n < listKategori2.size(); n++) {
68.       double temp =0;//listSum.get(0).getSum();
69.       String tempid="";//this.listSum.get(0).getId();
70.       for(int m = 0; m < listSum.size(); m++){
71.          if
(listKategori2.get(n).equals(listSum.get(m).getkategori())) {
72.             if (temp<listSum.get(m).getSum()) {
73.                temp=listSum.get(m).getSum();
74.                tempid=listSum.get(m).getDocId();
75.             }
76.          }
77.       }
78.       hasilranking.add(tempid);
79.       System.err.println("Initiation Weight for Training");
80.       System.err.println("tempid = "+tempid+" sum="+temp);
81.     }
82.     return hasilranking;
83.     }
```

At line 55 to line 58, it will get calculate summarization of all the terns based on each categories. And at line 71 to line 74, it will compare each data that has the same terns in each categories.

In Classification class will produce some value for the classification calculation for the training data. The result of this class could show the accuracy of LVQ algorithm using the training data.

```
84.  // hitung pengurangan eucclidean distance
85.  for (int i = 0; i < sizedata; i++) {
86.  double[] w = new
double[listWClass.get(0).getListBobot().size()];
87.  for (int j = 0; j < listWData.size(); j++) {
88.  for (int k = 0; k < listWClass.get(j).getListBobot().size(); k++)
{
89.  double wtemp = listWData.get(j).getListBobot().get(i).getW()
90.        - listWClass.get(j).getListBobot().get(k).getW();
91.  wtemp = wtemp*wtemp;
92.  w[k] += wtemp;
93.  }
94.  }
95.  // hitung akar eucclidean distance
96.  for(int l = 0 ; l < w.length;l++){
97.  w[l] = Math.sqrt(w[l]);
98.
99.  //print W + index + kategori
100. System.err.println(" w : "+w[l]);
```

This function will calculate eucclidean value of each training data. At line 87 to 92 it will calculate the differences of eucclidean distance value and square root the values at line 97.

```
101. public int FindFirstMinIndex(double[] w)
102. {
103.    int minIndex = 0;
104.    for (int i = 1; i < w.length; i++)
105.       if(w[i] < w[minIndex])
106.          minIndex=i;
107.    return minIndex;
108. }
```

After the ecculidean values founded, at line 105 calculate minimum value of the training data and the initiation data using its indexing.

```
109.  if (katergoriData.equals(kategoriClass)) {
110.   for (int j = 0; j < listWClass.size(); j++) {
111.                                    double c =
listWClass.get(j).getListBobot().get(index).getW();
112.       double d = listWData.get(j).getListBobot().get(i).getW();
113.       double temp= c+alpha*(d-c);
114.       listWClass.get(j).getListBobot().get(index).setW(temp);
115.   }
116.  } else {
117.   for (int j = 0; j < listWClass.size(); j++) {
118.                                    double c =
listWClass.get(j).getListBobot().get(index).getW();
119.       double d = listWData.get(j).getListBobot().get(i).getW();
120.       double temp= c-alpha*(d-c);
121.       listWClass.get(j).getListBobot().get(index).setW(temp);
122.   }
123.  }
```

At line 109, it compare whether the training data index is the same as the initiation data index. If it is the same, the weights will be updated at line 113 but if it is not the weights will be updated at line 120.

5.1.2 Testing Data

At Testing Menu, it will also imported new data into database that will be used as testing data in the classification process. The form of data is CSV file.

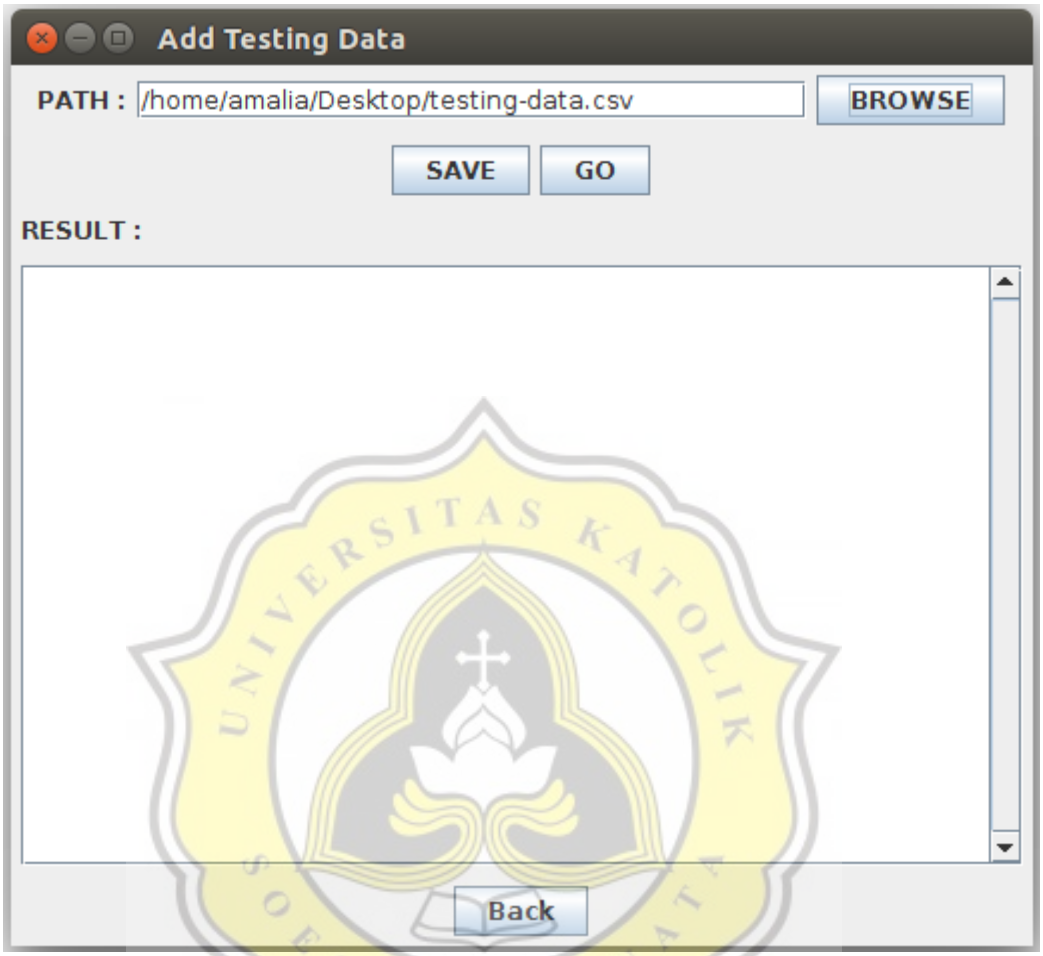

Illustration 5.2: Testing Menu Graphic Interface

```
124.  HashMap<String,String> hasilTest = new HashMap<>();
125.  for (int i = 0; i < listBobotDocument.size(); i++) {
126.  double[] w = new double[LVQ.get(0).getListBobot().size()];
127.  for (int j = 0; j < LVQ.size(); j++) {
128.   for (int k = 0; k < LVQ.get(j).getListBobot().size(); k++) {
129.
130.  double wtemp =
listBobotDocument.get(listDocumentId.get(i)).get(LVQ.get(j).getTerm
()) - LVQ.get(j).getListBobot().get(k).getW();
133.       wtemp = wtemp*wtemp;
134.       w[k] += wtemp;
135.   }
136.  }
```

```
137.  // hitung akar eucclidean distance
138.  for(int l = 0 ; l < w.length;l++){
139.   w[l] = Math.sqrt(w[l]);
140.   System.err.println("w : "+w[l]);
```

At the code above is used to calculate the eucclidean distance value of the testing data. The calculation is executed at line 130 to 139.

```
140.  int index =  FindFirstMinIndex(w);
141.  String wClass = LVQ.get(0).getListBobot().get(index).getId();
142.  String kategoriClass= getKatgori(wClass);
143.  System.err.println("Id Data Testing = "+listDocumentId.get(i)+"
is "+ kategoriClass);
144.  hasilTest.put(listDocumentId.get(i), kategoriClass);
```

After the eucclidean distance value founded, at line 140 it will find minimum values of the testing data using the updated initiation weight of the training data. At line 143, the resuls of the classification will be printed on.

5.2 Testing

In LVQ algorithm, the initial weights uses data taken from the TF-IDF Text Summarization process. The initial weights is taken from the existing training data which must be taken into vector form.

After the initial weight is determined, the training data on testing process is then executed using several values learning rate (α) and MaxEpoch to determine the best level of accuracy of the LVQ algorithm. In this project, the data is divided into 3 class categories with each class having 20 training data so the total of the training data is 60 data.

Table 5.1: LVQ Algorithm Results on Training Data

No	Learing Rate	MaxEpoch	Training Data Result		Accuracy Percentage
			Correct Data	Incorrect Data	
1.	0.5	10	31	26	54.38%
		25	27	30	47.36%

		50	49	8	85.96%
2.	0.35	10	26	31	45.61%
		25	49	8	85.96%
		50	50	7	87.71%
3.	0.1	10	25	32	43.85%
		25	50	7	87.71%
		50	52	5	91.22%

From the results of the training data above, it was found that the highest accuracy value was 91.22% using the learning rate was 0.1 and MaxEpoch was 50.

To testing the new data to be classified, can use the results of the training data accuracy test above. The results for the new test data can be seen in the table below.

Table 5.2: Testing Results of LVQ Algorithm on Testing Data

Data Id	Final Weights			Recognition As
	Literature Class	Biography Class	Kids Class	
1546861445-234	13.417671737128336	13.253169058412318	13.38761180781970 6	Biography Book
1546861607-288	12.15498736237409	11.737642820444576	11.99931686681544 3	Biography Book
1592294685-51	7.532898037634407	7.929378524771017	6.389286582873257	Kids Book
1592294898-120	11.213123275504564	11.384313639959 28	10.68018626277185 1	Kids Book
1546854385-96	7.380173232220613	8.187529479823855	7.454504985875234	Literature Book
1546851810-32	9.901717810172448	10.321382237329813	9.992451600562982	Literature Book