CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

This chapter describes how to implement the Naive Bayes Algorithm in the coding section. There are several important functions in implementing the algorithm, namely :

Function for storing training data

```
8.  def load_csv(filename):
9.      dataset = list()
10.     with open(filename, 'r') as file:
11.         csv_reader = reader(file)
12.         for row in csv_reader:
13.             if not row:
14.                 continue
15.             dataset.append(row)
16.     return dataset
```

The function above serves to read and save the Training Data that will be used into the variable dataset. Line 8 is the name of the function followed by the name of the Data Training file that will be used. The 9th line functions to declare the dataset variable. On line 10 to line 15 it functions to read the Data Training file of type csv and save it into the dataset variable as a list. The 16th line serves to return the final result of the dataset variable.

Function to convert String to Integer

```
24. def str_column_to_int(dataset, column):
25.     class_values = [row[column] for row in dataset]
26.     unique = set(class_values)
```

```
27.    lookup = dict()
28.
29.    for i, value in enumerate(unique):
30.        lookup[value] = i
31.        print('[%s] => %d' % (value, i))
32.    for row in dataset:
33.        row[column] = lookup[row[column]]
34.    return lookup
```

The function above functions to change the data type of each class in the training data from strings to integers. The 25th line functions to take only the data in the last column on each training data row and enter it into the class_values variable. The 26th line serves to set the class_values variable. Line 27 serves to declare lookup variables that contain a dictionary. Lines 29 to 33 are used to change the data type of the last row of each dataset from a string to an integer. Line 34 serves to return the final result of the lookup variable.

Function to find the Mean value

```
56. def mean(numbers):
57.    a = sum(numbers)/float(len(numbers))
58.    return a
```

The function above functions to find the Mean value in a data. Row 57 functions to calculate the mean value of data and save it into variable a. Line 58 serves to return the final result of variable a.

Function to find Standard Deviation value

```
61. def stdev(numbers):
62.    avg = mean(numbers)
63.    variance = sum([(x-avg)**2 for x in numbers]) /
       float(len(numbers)-1)
```

```
64.     b = sqrt(variance)
65.     return b
```

The function above functions to find the Standard Deviation value of a data. Line 62 serves to declare the avg variable containing the Mean value of the data used. Lines 63 to 64 are used to find the Standard Deviation value and then save it into the variable b. Line 65 serves to return the final result of the variable b.

Function to Combine data

```
79. def summarize_by_class(dataset):
80.     separated = separate_by_class(dataset)
81.     summaries = dict()
82.     for class_value, rows in separated.items():
83.         summaries[class_value] = summarize_dataset(rows)
84.         d = summaries
85.     return d
```

The function above serves to call a function that looks for the Mean value and Standard Deviation value then combines the search results for the two values. Line 80 serves to call the separated_by_class function which functions to separate the training data based on each class and save the results into separated variables. Line 81 functions to declare variable summaries containing the dictionary. Lines 82 to line 84 function to call the summarize_dataset function which contains the Mean value and Standard Deviation value which will be grouped based on their respective classes and then save them into variable d. Line 85 serves to return the final result of the variable d.

Function to apply the Naive Bayes Algorithm

```
89. def calculate_probability(x, mean, stdev):
```

```
90.     exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
91.     c = (1 / (sqrt(2 * pi) * stdev)) * exponent
92.     return c
```

This function requires Data Testing, Mean value, and Standard Deviation value. Line 90 and line 91 function to calculate the probability value for each class using the Naive Bayes algorithm. The calculation result is then stored in variable c and will be returned at line 92.

Function to calculate the Final Probability

```
96. def calculate_class_probabilities(summaries, row):
97.     total_rows = sum([summaries[label][0][2] for label in
        summaries])
98.     probabilities = dict()
99.     for class_value, class_summaries in summaries.items():
100.            probabilities[class_value] =
                summaries[class_value][0][2]/float(total_rows)
101.            for i in range(len(class_summaries)):
102.                mean, stdev, _ = class_summaries[i]
103.                probabilities[class_value]*=
                    calculate_probability(row[i],mean, stdev)
104.    return probabilities
```

The function above is to find the final probability. Line 97 serves to calculate the amount of training data. Line 98 functions to declare variable probabilities that contain the dictionary. Lines 99 through 103 functions to find the final probability by calling the calculate_probability function which functions to calculate the probability of each class by applying the Naive Bayes Algorithm. Line 104 returns the final result of variable probabilities.

Function to determine the Final Result

```
109. def predict(summaries, row):
110.     probabilities = calculate_class_probabilities(summaries,
         row)
111.     best_label, best_prob = None, -1
112.     for class_value, probability in probabilities.items():
113.             if best_label is None or probability > best_prob:
114.                     best_prob = probability
115.                     best_label = class_value
116.     return best_label
```

The function above serves to determine the final result of the classification. Line 110 functions to call the calculate_class_probabilities function to calculate the probability value for each class. Line 111 functions to declare the best_label and best_prob variables. Line 112 to line 115 functions to find the class with the highest probability and then store it in the best_label variable. Line 116 serves to return the final result of the best_label variable.

Main Function

```
128. filename = 'finaltrain.csv'
129. dataset = load_csv(filename)
130. print("filename: ",filename)
131. for i in range(len(dataset[0])-1):
132.     str_column_to_float(dataset, i)
133. str_column_to_int(dataset, len(dataset[0])-1)
134. model = summarize_by_class(dataset)
135. row = [2,15000000,2500000,12,2.5,3]
136. label = predict(model, row)
137.  print('Data Testing=%s, Predicted: %s, Label: %s' % (row,
      final(label), label))
```

The function above serves to call other functions. Line 128 functions to store the name of the training data file that will be used in the variable filename. Line 129 calls the load_csv function and stores the results in the dataset variable. Lines 131 through 132 call the str_column_to_float function to change the data type to float. Line 133 functions to call the str_column_to_int function. Line 134 functions to call the function summarize_by_class and then store it in the model variable. Line 135 functions to store the testing data that will be used into variable row. Line 136 functions to call the predict function and then save the result into variable label. Line 137 functions to display the Testing Data used and also the prediction results.

5.2 Testing

This chapter discusses the testing process carried out in this study. The first test is done using 300 training data, while the second testing is done using 600 training data where both tests use 94 training data. The last test is done using the K-Fold Cross Validation method. The following are the results of the first testing :

Table 5.1: Testing Table using 300 Data

No	Gender	Submission	Salary	Period	Interest	Guarantee	Class	Prediction
1	2	8000000	3500000	12	2.5	3	A	D
2	2	10000000	2400000	10	2.5	3	A	D
3	1	8000000	2500000	12	2.5	3	A	D
4	2	7000000	3000000	12	2.5	3	A	D
5	2	8000000	2500000	12	2.5	3	A	D
6	2	7500000	3000000	12	2.5	3	A	D
7	2	10000000	2500000	10	2.5	3	A	D
8	2	6000000	2500000	10	2.5	3	A	D
9	2	7000000	2500000	6	2.5	3	A	A
10	1	5000000	2700000	12	2.5	3	A	D
11	2	8000000	2500000	12	2.5	3	A	D
12	2	7000000	2500000	12	2.5	3	A	D
13	2	10000000	4000000	12	2.5	3	A	D
14	2	7500000	2500000	10	2.5	3	A	D
15	2	9000000	2600000	12	2.5	3	A	D
16	1	6000000	2500000	10	2.5	3	A	D
17	1	10000000	2500000	12	2.5	3	A	D
18	2	7000000	2500000	12	2.5	3	A	D
19	2	7000000	2500000	6	2.5	3	A	A
20	2	10000000	3800000	12	2.5	3	A	D
21	1	10000000	3500000	12	2.5	3	A	D
22	2	5000000	2500000	6	2.5	3	A	A
23	2	8000000	2500000	10	2.5	3	A	D
24	2	8000000	3700000	12	2.5	3	A	D
25	2	9000000	2500000	12	2.5	3	A	D
26	2	7000000	2500000	12	2.5	3	A	D
27	1	6000000	2200000	12	2.5	3	A	D
28	2	7500000	2700000	12	2.5	3	A	D
29	2	7000000	3500000	12	2.5	3	A	D
30	2	5000000	2500000	12	2.5	3	A	D
31	2	6000000	2500000	12	2.5	3	A	D
32	2	6000000	2500000	12	2.5	3	A	D
33	1	8000000	4000000	12	2.5	3	A	D

34	1	7000000	2500000	12	2.5	3	A	D
35	2	6000000	2100000	12	2.5	3	A	D
36	1	10000000	2400000	12	2.5	3	A	D
37	2	10000000	2500000	12	2.5	3	A	D
38	2	6000000	2500000	12	2.5	3	A	D
39	2	7500000	2500000	6	2.5	3	A	A
40	1	5000000	2400000	12	2.5	3	A	D
41	2	8000000	2500000	12	2.5	3	A	D
42	2	10000000	4500000	12	2.5	3	A	D
43	2	8000000	3500000	12	2.5	3	A	D
44	2	8000000	2500000	12	2.5	3	A	D
45	1	9000000	2400000	12	2.5	3	A	D
46	2	10000000	2500000	12	2.5	3	A	D
47	1	7000000	3600000	12	2.5	3	A	D
48	2	7000000	2500000	12	2.5	3	A	D
49	2	7000000	2500000	12	2.5	3	A	D
50	1	7000000	2600000	12	2.5	3	A	D
51	2	5000000	2500000	6	2.5	3	A	A
52	1	7500000	2750000	12	2.5	3	A	D
53	1	8000000	2500000	10	2.5	3	A	D
54	2	8000000	2500000	12	2.5	3	A	D
55	2	9000000	2500000	10	2.5	3	A	D
56	2	10000000	2500000	10	2.5	3	A	D
57	2	6000000	2500000	10	2.5	3	A	D
58	1	5000000	2500000	10	2.5	3	A	D
59	1	7000000	2500000	12	2.5	3	A	D
60	2	10000000	2500000	12	2.5	3	A	D
61	1	8000000	2500000	10	2.5	3	A	D
62	2	9000000	2500000	10	2.5	3	A	D
63	2	7000000	2500000	12	2.5	3	A	D
64	2	10000000	3790000	12	2.5	3	A	D
65	2	5000000	2500000	10	2.5	3	A	D
66	1	10000000	3800000	10	2.5	3	A	D
67	2	6000000	2500000	12	2.5	3	A	D
68	2	7000000	2500000	12	2.5	3	A	D
69	2	8000000	2500000	10	2.5	3	A	D
70	2	8000000	2500000	12	2.5	3	A	D
71	2	7000000	2500000	12	2.5	3	A	D
72	1	5000000	2500000	10	2.5	3	A	D
73	2	6000000	2500000	10	2.5	3	A	D
74	2	7000000	2500000	10	2.5	3	A	D
75	2	4000000	2500000	10	2.5	3	A	D
76	2	8000000	2500000	10	2.5	3	A	D
77	1	7000000	4000000	6	2.5	3	A	A

78	2	10000000	2500000	12	2.5	3	A	D
79	2	10000000	2500000	12	2.5	3	A	D
80	2	8000000	2500000	12	2.5	3	A	D
81	2	7500000	3000000	12	2.5	3	A	D
82	1	7500000	2500000	12	2.5	3	A	D
83	1	7500000	2500000	12	2.5	3	A	D
84	2	7000000	6500000	10	2.5	3	A	D
85	2	20000000	2310000	24	2.5	3	D	A
86	2	7000000	2500000	10	2.5	3	D	D
87	2	10000000	2500000	12	2.5	3	D	D
88	1	10000000	2500000	12	2.5	3	D	D
89	1	10000000	2500000	12	2.5	3	D	D
90	2	10000000	2500000	12	2.5	3	D	D
91	2	5000000	2500000	12	2.5	3	D	D
92	2	10000000	2500000	10	2.5	3	D	D
93	2	8000000	2500000	12	2.5	3	D	D
94	2	15000000	2500000	12	2.5	3	D	D

Based on testing using the 300 training data above, there are 15 data that can be classified correctly and 79 data is wrong.

Following are the results of the second testing :

Table 5.2: Testing Table using 600 Data

No	Gender	Submission	Salary	Period	Interest	Guarantee	Class	Prediction
1	2	8000000	3500000	12	2.5	3	A	A
2	2	10000000	2400000	10	2.5	3	A	A
3	1	8000000	2500000	12	2.5	3	A	A
4	2	7000000	3000000	12	2.5	3	A	A
5	2	8000000	2500000	12	2.5	3	A	A
6	2	7500000	3000000	12	2.5	3	A	A
7	2	10000000	2500000	10	2.5	3	A	A
8	2	6000000	2500000	10	2.5	3	A	A
9	2	7000000	2500000	6	2.5	3	A	A
10	1	5000000	2700000	12	2.5	3	A	A
11	2	8000000	2500000	12	2.5	3	A	A
12	2	7000000	2500000	12	2.5	3	A	A
13	2	10000000	4000000	12	2.5	3	A	A
14	2	7500000	2500000	10	2.5	3	A	A
15	2	9000000	2600000	12	2.5	3	A	A
16	1	6000000	2500000	10	2.5	3	A	A
17	1	10000000	2500000	12	2.5	3	A	A
18	2	7000000	2500000	12	2.5	3	A	A

19	2	7000000	2500000	6	2.5	3	A	A
20	2	10000000	3800000	12	2.5	3	A	A
21	1	10000000	3500000	12	2.5	3	A	A
22	2	5000000	2500000	6	2.5	3	A	A
23	2	8000000	2500000	10	2.5	3	A	A
24	2	8000000	3700000	12	2.5	3	A	A
25	2	9000000	2500000	12	2.5	3	A	A
26	2	7000000	2500000	12	2.5	3	A	A
27	1	6000000	2200000	12	2.5	3	A	A
28	2	7500000	2700000	12	2.5	3	A	A
29	2	7000000	3500000	12	2.5	3	A	A
30	2	5000000	2500000	12	2.5	3	A	A
31	2	6000000	2500000	12	2.5	3	A	A
32	2	6000000	2500000	12	2.5	3	A	A
33	1	8000000	4000000	12	2.5	3	A	A
34	1	7000000	2500000	12	2.5	3	A	A
35	2	6000000	2100000	12	2.5	3	A	A
36	1	10000000	2400000	12	2.5	3	A	A
37	2	10000000	2500000	12	2.5	3	A	A
38	2	6000000	2500000	12	2.5	3	A	A
39	2	7500000	2500000	6	2.5	3	A	A
40	1	5000000	2400000	12	2.5	3	A	A
41	2	8000000	2500000	12	2.5	3	A	A
42	2	10000000	4500000	12	2.5	3	A	D
43	2	8000000	3500000	12	2.5	3	A	A
44	2	8000000	2500000	12	2.5	3	A	A
45	1	9000000	2400000	12	2.5	3	A	A
46	2	10000000	2500000	12	2.5	3	A	A
47	1	7000000	3600000	12	2.5	3	A	A
48	2	7000000	2500000	12	2.5	3	A	A
49	2	7000000	2500000	12	2.5	3	A	A
50	1	7000000	2600000	12	2.5	3	A	A
51	2	5000000	2500000	6	2.5	3	A	A
52	1	7500000	2750000	12	2.5	3	A	A
53	1	8000000	2500000	10	2.5	3	A	A
54	2	8000000	2500000	12	2.5	3	A	A
55	2	9000000	2500000	10	2.5	3	A	A
56	2	10000000	2500000	10	2.5	3	A	A
57	2	6000000	2500000	10	2.5	3	A	A
58	1	5000000	2500000	10	2.5	3	A	A
59	1	7000000	2500000	12	2.5	3	A	A
60	2	10000000	2500000	12	2.5	3	A	A
61	1	8000000	2500000	10	2.5	3	A	A
62	2	9000000	2500000	10	2.5	3	A	A

63	2	7000000	2500000	12	2.5	3	A	A
64	2	10000000	3790000	12	2.5	3	A	A
65	2	5000000	2500000	10	2.5	3	A	A
66	1	10000000	3800000	10	2.5	3	A	A
67	2	6000000	2500000	12	2.5	3	A	A
68	2	7000000	2500000	12	2.5	3	A	A
69	2	8000000	2500000	10	2.5	3	A	A
70	2	8000000	2500000	12	2.5	3	A	A
71	2	7000000	2500000	12	2.5	3	A	A
72	1	5000000	2500000	10	2.5	3	A	A
73	2	6000000	2500000	10	2.5	3	A	A
74	2	7000000	2500000	10	2.5	3	A	A
75	2	4000000	2500000	10	2.5	3	A	A
76	2	8000000	2500000	10	2.5	3	A	A
77	1	7000000	4000000	6	2.5	3	A	A
78	2	10000000	2500000	12	2.5	3	A	A
79	2	10000000	2500000	12	2.5	3	A	A
80	2	8000000	2500000	12	2.5	3	A	A
81	2	7500000	3000000	12	2.5	3	A	A
82	1	7500000	2500000	12	2.5	3	A	A
83	1	7500000	2500000	12	2.5	3	A	A
84	2	7000000	6500000	10	2.5	3	A	D
85	2	20000000	2310000	24	2.5	3	D	D
86	2	7000000	2500000	10	2.5	3	D	A
87	2	10000000	2500000	12	2.5	3	D	A
88	1	10000000	2500000	12	2.5	3	D	A
89	1	10000000	2500000	12	2.5	3	D	A
90	2	10000000	2500000	12	2.5	3	D	A
91	2	5000000	2500000	12	2.5	3	D	A
92	2	10000000	2500000	10	2.5	3	D	A
93	2	8000000	2500000	12	2.5	3	D	A
94	2	15000000	2500000	12	2.5	3	D	D

Based on testing using 600 training data above, there are 84 data that can be classified correctly and 10 data is wrong.

Next is to determine the precision, recall, and accuracy values for each test. The following is the formula used to determine the precision value in testing:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Illustration 5.1: Precision Formula

Information :

True Positive : the program can correctly classify a prospective customer in the Approved class

False Positive : the program classifies that a prospective customer is included in the Approved class but on the original data the prospective customer is included in the Disapproved class

Table 5.3: Precision table of the First Testing

No	Type	Value
1	True Positive	6
2	False Positive	1

First Testing Precision = 6 / (6 + 1)

= 6 / 7

= 0.85

Table 5.4: Precision table of the Second Testing

No	Type	Value
1	True Positive	82
2	False Positive	8

Second Testing Precision = 82 / (82 + 8)

= 82 / 90

= 0.91

The following is the formula used to determine the recall value:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Illustration 5.2: Recall Formula

Information :

True Positive : the program can correctly classify a prospective customer in the Approved class

False Negative : the program classifies that the prospective customer is included in the Disapproved class but on the original data the prospective customer is included in the Approved class.

Table 5.5: Recall table of the First Testing

No	Type	Value
1	True Positive	6
2	False Negative	78

First Testing Recall = 6 / (6 + 78)

= 6 / 84

= 0.07

Table 5.6: Recall table of the Second Testing

No	Type	Value
1	True Positive	82
2	False Negative	2

Second Testing Recall = 82 / (82 + 2)

= 82 / 84

= 0.97

Here is the formula used to determine accuracy value:

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

Illustration 5.3: Accuracy Formula

Information :

True Positive : the program can correctly classify a prospective customer in the Approved class

False Positive : the program classifies that a prospective customer is included in the Approved class but on the original data the prospective customer is included in the Disapproved class

False Negative : the program classifies that the prospective customer is included in the Disapproved class but on the original data the prospective customer is included in the Approved class.

True Negative : the program can correctly classify that the prospective customer is included in the Disapproved class.

Table 5.7: Accuracy table of the First Testing

No	Type	Value
1	True Positive	6
2	False Positive	1
3	False Negative	78
4	True Negative	9

First Testing Accuracy = (6 + 9) / (6 + 9 + 1 + 78)

= 15 / 94

= 0.15

Table 5.8: Accuracy table of the Second Testing

No	Type	Value
1	True Positive	82
2	False Positive	8
3	False Negative	2
4	True Negative	2

Second Testing Accuracy = (82 + 2) / (82 + 2 + 8 + 2)

= 84 / 94

= 0.89

The following is a summary table of the precision, recall, and accuracy values for each test:

Table 5.9: Summary Table

Testing	Precision	Recall	Accuracy
Testing 1	0.85	0.07	0.15
Testing 2	0.91	0.97	0.89

Based on the results of precision, recall, and accuracy on each test, it can be concluded that the first test gets a low recall and accuracy score, while in the second test a fairly high recall and accuracy score is obtained. Factors that influence these results are the amount of data in each class on the training data used, the mean value, the standard deviation value, and the probability value for each class.

The third testing uses the K-Fold Cross Validation method. K-Fold Cross-validation is a resampling procedure used to evaluate machine learning models. K-Fold Cross-validation has a single parameter called k that refers to the number of groups that a given data sample is to be split into. The following are the steps used in implementing the K-Fold Cross Validation method :

1. Shuffle the dataset randomly
2. Split the dataset into k groups
3. For each unique group:
    1. Take the group as a hold out or test data set
    2. Take the remaining groups as a training data set
    3. Fit a model on the training set and evaluate it on the test set
    4. Retain the evaluation score and discard the model
4. Summarize the accuracy level of the model using the sample of model evaluation scores

Following are the results of the third testing :

```
Scores: [91.30434782608695, 91.30434782608695, 91.30434782608695, 91.30434782608695, 82.6086956521739,
 95.65217391304348, 86.95652173913044, 89.85507246376811, 94.20289855072464, 33.33333333333333]
Mean Accuracy: 84.783%
```

Illustration 5.4: K-Fold Cross Validation Testing

Based on the results of testing using the K-Fold Cross Validation method with 10 fold, it can be concluded that the average accuracy of the 10 tests that have been carried out is 84,783%. This results indicate that the program can perform the classification process well using the Naive Bayes algorithm.