CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1 Implementation

```
1  df = pd.read_csv("invistico_airline.csv")
2  df['satisfaction'] = df['satisfaction'].replace({"satisfied":1, 'dissatisfied':0})
```

Line 1 of the program code contains a command to read a dataset file named invistico_airline. Line 2 contains the command to change the satisfaction variable to an integer in the form of satisfied to 1, and dissatisfied to 0.

```
3  def one_vs_all_cols(s):
4  m = list(set(s))
5  m.sort()
6  for i in range(len(s)):
7  new = [0]*len(m)
8  new[m.index(s[i])] = 1
9  s[i] = new
10 return m
```

Line 3 contains the command to create a function with the name one_vs_all_cols. Line 4 contains the variable m which is a list of set s. Line 5 contains the sorting of the variable m. Line 6 contains the loop function in length s. Line 7 contains the variable new which is where dataset [0] is multiplied by length m. Line 8 contains the new variable containing the variable m combined with index s equal to 1. Line 9 contains the index s equal to the new variable and returns the result on row 10.

```
11 def LinearSVM_cost0(z):
12 if(z < -1):
13 return 0
14 return z + 1
```

Line 11 contains the command to create a compute cost function. Line 12 contains the condition if z is less than -1. Lines 13-14 contain return the result.

```
15 def cost(theta,c,x,y):
16 cost = 0.0
17 for i in range(len(x)):
```

18 z = ThetaTX(theta[c], x[i])

19 cost += y[i]*LinearSVM_cost1(z) + (1 – y[i])*LinearSVM_cost0(z)

20 return cost

Line 15 contains the command to create a function to calculate the cost of SVM. Line 16 contains a variable declaration equal to 0,0. Row 17 contains repetitions in the range of length x. Line 18 contains the declaration of the variable z derived from the ThetaTX function. Line 19 contains the calculation of the function cost and returns the result on line 20.

21 def test_split(dataset, idx, val):

22 left, right = [ ], [ ]

23 for row in dataset:

24 if row[idx] < val:

25 left.append(row)

26 else:

27 right.append(row)

28 return left, right

Line 21 contains the command to create a data split function. Line 22 contains the left and right variable declarations. The core of the program code is in line 23 for looping rows in the dataset, line 24 contains the condition row in the index is less than the variable value. Lines 25 -27 are the contents of the if else line. Line 28 contains returning left and right variables.

29 def build_tree(train, max_depth, min_size, n_features):

30 root = get_split(train, n_features)

31 split(root, max_depth, min_size, n_features, 1)

32 return root

Line 29 is a command to create a function to build a tree randomly. Line 30 is a root variable declaration. Row 31 contains dividing depth, root, minimum size and number of features. Line 32 contains returns the root result.

33 def bagging_predict(trees, row):

34 predictions = [predict(tree, row) for tree in trees]

35 return max(set(predictions), key=predictions.count)

Line 33 is a command to create a function to store the prediction results. Line 34 is the contents of the prediction function and returns the maximum result. Line 35 is returning the label selected by most trees.

```
36 def predict(data,theta):
37 predictions = []
38 count = 1
39 for row in data:
40 hypothesis = []
41 multiclass_ans = [0]*len(theta)
42 for c in range(len(theta)):
43 z = ThetaTX(row,theta[c])
44 hypothesis.append(sigmoid(z))
45 index = hypothesis.index(max(hypothesis))
46 multiclass_ans[index] = 1
47 predictions.append(multiclass_ans)
48 count+=1
49 return predictions
```

Line 36 is the command for making predictions for the Support Vector Machine algorithm. Line 37 is creating an empty array named predictions. Line 38 is a count starting at one. Line 39 is the loop function. Line 40 creates an empty array called hypothesis. Line 41 is a multiclass calculation by multiplying the number of datasets by the theta length. Line 42 is looping c based on theta length. Line 43 is the calculation of the z value. Line 44 is a function adding array values in sigmoid order. Line 45 is the contents of the index. Line 46 is the content of multiclass is 1. Line 47 is adding array value to multiclass. Line 48 is count plus 1. Line 49 is returning predictions.

```
50 def compute_confusion_matrix (true,  pred):
51 K = len (np.unique(true))
52 result = np.zeros (K, K))
53 TP, TN, FP, FN = [], [], [], []
54 for i in range(len(true)):
55 if true[i] == 0 and pred[i] == 0:
56 TP.append(1)
57 TN.append(0)
58 FP.append(0)
59 FN.append(0)
```

```
60  elif true[i] == 1 and pred[i] == 1:
61  TP.append(0)
62  TN.append(1)
63  FP.append(0)
64  FN.append(0)
65  elif true[i] == 0 and pred[i] == 1:
66  TP.append(0)
67  TN.append(0)
68  FP.append(1)
69  FN.append(0)
70  elif true[i] == 1 and pred[i] == 0:
71  TP.append(0)
72  TN.append(0)
73  FP.append(0)
74  FN.append(1)
75  result[true[i]][pred[i]] += 1
76  return result
```

Line 50 is the command to create the confusion matrix function. Line 51 defines the number of classes. Lines 52-53 are calculating each feature of the confusion matrix in the form of TP, TN, FP and FN. Line 54 is looping in each correct. Lines 55-59 are counting the number of True Positives (TP) in the confusion matrix. Lines 60-64 are counting the number of True Negatives (TN) in the confusion matrix. Lines 65-69 are counting the number of False Negatives (FN) in the confusion matrix. Lines 70-74 are counting the number of False Positives (FP) in the confusion matrix. Value 0 for satisfied and 1 for dissatisfied. Line 75 contains the result based on the correct prediction plus 1. Line 76 is returning the result.

```
77  def accuracy_score(predicted, actual):
78  n = len(predicted)
79  correct = 0
80  for i in range(n):
81  if (predicted[i] == actual[i]):
82  correct += 1
83  return correct / n
```

Line 77 is an accuracy score command. Row 78 is the value of n based on the number of predictions. Line 39 is correct starting from 0. Line 80 is a loop function. Line 81 is a looping function if the prediction matches or equals the actual. Line 82 is the correct number plus 1. Line 83 returns the correct number divided by the number of predictions.

```
84  PRE, RE = (TP/(TP+FP)), (TP/(TP+FN))
85  F1 = ((2 * PRE * RE)/(PRE + RE))
```

Line 84 is the calculation of precision and recall based on the confusion matrix. Line 85 is the calculation of the f1 score based on the confusion matrix.Line 84 is the calculation of precision and recall based on the confusion matrix. Line 85 is the calculation of the f1 score based on the confusion matrix.

```
86  b = mpatches.Patch(color='#0F52BA', label='Random Forest')
87  t = mpatches.Patch(color='#B61919', label='Linear SVM')
```

Line 86 is the function used to visualize the blue lines of the Random Forest algorithm. Line 87 is a function for line visualization of the Support Vector Machine algorithm.

```
88  fig, ax =  plt.subplots(2, 2, figsize=(20,10))
89  ax[0,0].plot(support, accuracy)
90  ax[0,0].plot(support, accuracy_)
91  ax[0,0].legend(handles=[b, t])
92  ax[0,0].title.set_text("Accuracy Score Comparison")
93  ax[0,1].plot(support, pre)
94  ax[0,1].plot(support, pre_)
95  ax[0,1].legend(handles=[b, t])
96  ax[0,1].title.set_text("Precision Score Comparison")
97  ax[1,0].plot(support, re)
98  ax[1,0].plot(support, re_)
99  ax[1,0].legend(handles=[b, t])
100 ax[1,0].title.set_text("Recall Score Comparison")
101 ax[1,1].plot(support, f1)
102 ax[1,1].plot(support, f1_)
103 ax[1,1].legend(handles=[b, t])
104 ax[1,1].title.set_text("F1-Score Score Comparison")
105 fig.savefig("result_save.png")
```

Line 88 describes the creation of figure axes or graphics. Line 89 is plotting area for random forest accuracy. Line 90 is plotting area for svm accuracy. Line 91 serves to determine the coordinates or angles of an image or accuracy axis. Line 92 to create the title text Comparison Score Accuracy. Line 93 is plotting area for random forest precision. Line 94 is plotting area for svm precision. Line 95 serves to determine the coordinates or angles of an image or precision axis. Line 96 to create the title text Precision Score Comparison. Line 97 is to plot the area for the random forest recall. Line 98 is creating a plot area for recall svm. Line 99

serves to determine the coordinates or angles of an image or recall axis. Line 100 to create title text Recall Score Comparison. Line 101 is to plot the area for the f1 score random forest. Line 102 is plotting the area for f1 svm scores. Line 103 serves to determine the coordinates or angles of an image or the f1 axis of the score. Line 104 to create title text F1 Score Comparison Score. Line 105 is a function to save the results of the graph in the form of an image.

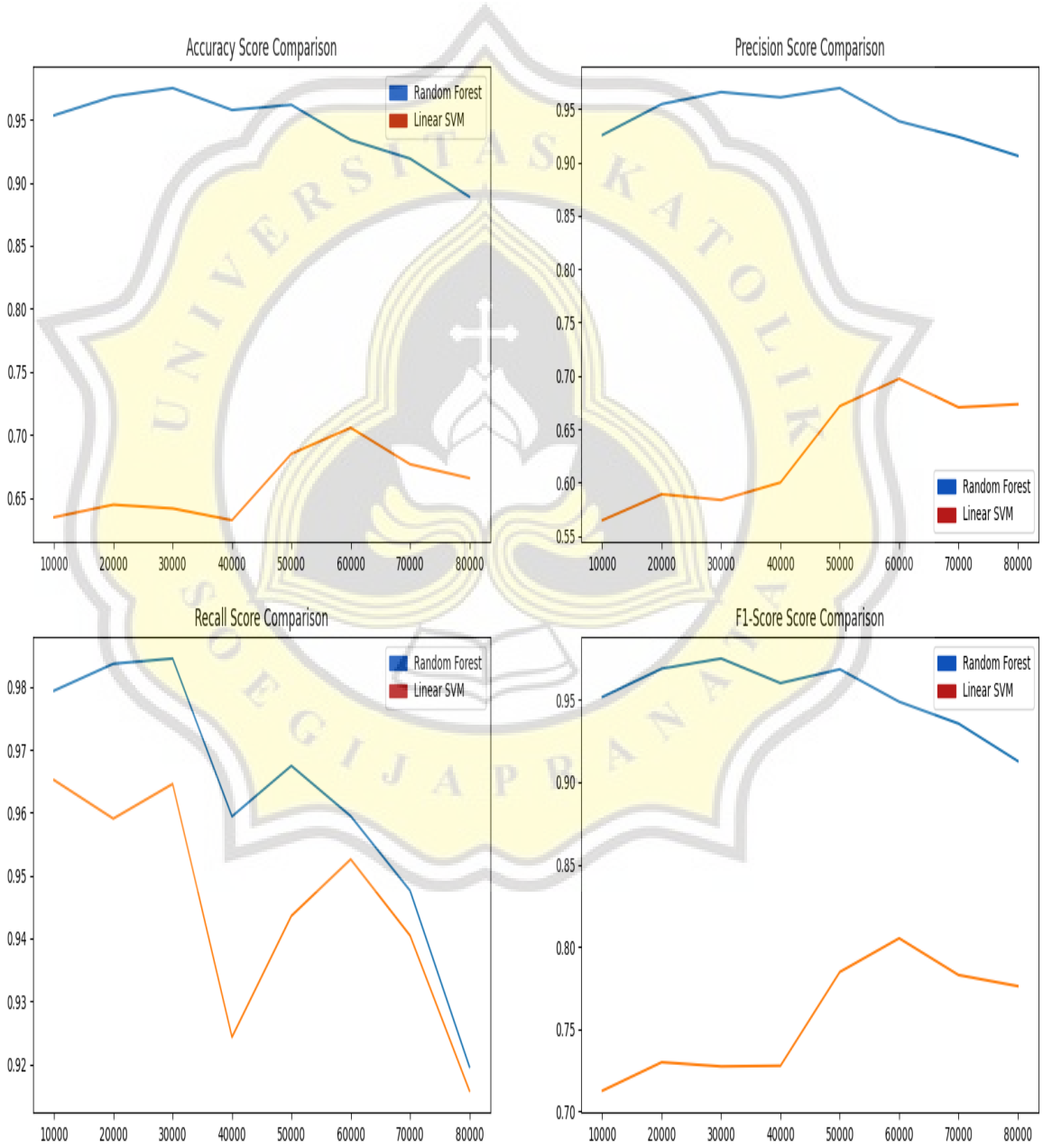
```
106 def shuffle_split_data(X, y):
107 arr_rand = np.random.rand(X.shape[0])
108 split = arr_rand < np.percentile(arr_rand, 70)
109 X_train = X[split]
110 y_train = y[split]
111 X_test = X[~split]
112 y_test = y[~split]
113 return X_train, y_train, X_test, y_test
```

Line 106 is the command to split the training and testing dataset with a ratio of 70: 30. Line 107 is to create a random array. Line 108 is a split based on a random array less than a percentage of 70. Line 109 describes the splitting of the dataset in the form of X trains to accommodate the source data to be trained. Line 110 describes the separation of the dataset in the form of a y train to accommodate the source data that will be used for testing. Line 111 describes the dataset separation in the form of an X test to accommodate the target data to be trained. Line 112 describes the separation of the dataset in the form of a y test to accommodate the target data that will be used for testing. Line 113 returns the values X_train, y_train, X_test and y_test.

5.2 Result

5.2.1 Comparison of Test Results of both Algorithms with 10000, 20000, 30000, 40000, 50000, 60000, 70000,80000 data.

Figure 5: Chart



Trial Random Forest	Accuracy	Precision	Recall	F-1 Score
10.000 data.	0.955	0.9310344827586207	0.9765957446808511	0.9532710280373832
20.000 data.	0.9678333333333	0.9500166611129623	0.9851416724257084	0.9672603901611535
30.000 data.	0.97622222222	0.9692240052758848	0.9692240052758848	0.9763064658990258
40.000 data.	0.95466666666666	0.956096020214782	0.9579113924050633	0.9570028453999369
50.000 data.	0.962466666666666	0.9697898423817863	0.968623592434678	0.9692063665700377
60.000 data.	0.9362777777777778	0.9421270835096032	0.960079324021383	0.9510184908399881
70.000 data.	0.9155238095238095	0.920199315781645	0.9463821324766712	0.9331070889894418
80.000 data.	0.8900833333333333	0.907634429525848	0.9201683435260077	0.9138584117032392

Trial Support Vector Machine	Accuracy	Precision	Recall	F-1 Score
10.000 data.	0.616333333333333	0.5526636844245628	0.9638297872340426	0.702507107779788
20.000 data.	0.6385	0.5742675681212867	0.9685556323427782	0.7210289389067525
30.000 data.	0.647	0.588362652232747	0.9698862368949365	0.7324180914680367
40.000 data.	0.63275	0.5975522692503824	0.9270569620253165	0.7266976744186047
50.000 data.	0.6895333333333	0.675472877911521	0.9447906417404613	0.7877489631283897
60.000 data.	0.71044444444444	0.7035054174633525	0.7035054174633525	0.8090002931691586
70.000 data.	0.6766666666666	0.6722776620243448	0.9378155117026159	0.7831502299437915
80.000 data.	0.668	0.6754058638236007	0.9164858288945881	0.777690977066012

Figure 5 describes the results of accuracy, precision, recall and f1 scores with a total of 80000 data in the form of graphs which were tested 8 times. The Random Forest algorithm's accuracy score increases and then decreases. However, the accuracy of the SVM algorithm increases and does not decrease and the result score is not better than the Random Forest algorithm. For precision values, both algorithms have increased and not decreased. For the value of F1, the score of the Random Forest algorithm then increases and decreases, but the SVM algorithm does not decrease. For recall results both algorithms decreased and then increased. Comparison of the two algorithms based on 80000 data will definitely affect the results that appear which are affected by the pattern search to detect the test data.

5.3 Analysis

Based on the results of research on Comparing the Support Vector Machine and Random Forest Algorithms in predicting the level of flight satisfaction, it can be Analysis

The way of classifying the SVM Algorithm is by separating the training data and testing data starting from the first to the eighth column as training data and the ninth column to the tenth column being the test data. Then look for the weight value of each attribute, and then carry out a trial by entering it into the test data. While the classification of the Random Forest Algorithm is done by separating the dataset based on the attribute values which sort each attribute starting from the smallest to the largest. The next step is to calculate the Gini index value from the previous split dataset and find the best split point for the dataset. Next is to create a decision tree and create a new random sample with the same attributes by repeating from the first step to making a decision tree that aims to compare the results formed from the original data set with the new random value data set.

The comparison between the SVM Algorithm and the Random Forest Algorithm to predict the level of passenger satisfaction is carried out by testing the data 8 times. Data testing was carried out with 10000 data, 20000 data, 30000 data, 40000 data, 50000 data 60000 data 70000 data and 80000 data. From the results of testing the data 8 times, it shows that the Random Forest Algorithm is better than the SVM algorithm because the Random Forest algorithm studies data patterns until the detection of test data is taken based on the number of voting results from the formed tree.

How to calculate the accuracy results is done by using a confusion matrix. The confusion matrix aims to measure performance for machine learning classification problems where the output is in the form of two or more classes. To get the value of accuracy is done by

looking for the value of TP (True Positive), FP (False Positive), TN (True Negative) and FN (False Negative). The accuracy value is obtained from the ratio of correct predictions to the entire data or can be described by (TP+TN) / (TP+TN+FP+FN).

For the results of the accuracy value, it shows that the Random Forest Algorithm is better than the SVM algorithm because from the graph of testing the data as much as 5x in finding the accuracy value, it shows that the accuracy value of the Random Forest Algorithm is higher than the SVM algorithm.

The analysis obtained based on the result is that the amount of training data will affect each machine learning model, because the small data volume causes the potential recall value to be higher (no peak to lower accuracy). And the random forest and SVM algorithms have far comparisons in terms of accuracy due to the basic differences in how the two models start from studying data patterns to detecting data testing.