APPENDIX

IMPORT LIBRARY

```
1  from random import seed, randrange
2  from math import exp, floor
3  import copy
4  import numpy as np
5  from collections import Counter
6  import pandas as pd
7  from helpers import *
8  from joblib import load
9  import matplotlib.pyplot as plt
10 from warnings import filterwarnings
11 import matplotlib.patches as mpatches
12 filterwarnings("ignore")
```

LOAD DATASET

```
13 df = pd.read_csv("invistico_airline.csv")
```

REPLACE SATISFACTION TO 0 AND 1

```
14 df['satisfaction']= df['satisfaction'].replace({"satisfied":1,
   'dissatisfied':0})
```

SPLIT DATA INTO TRAIN AND TEST WITH SVM ALGORITHM

```
15 def cross_val_split(data_X, data_Y, test_size, seed_val):
16 data_x = data_X.tolist()
17 data_y = data_Y.tolist()
18 seed(seed_val)
19 train_size = floor((1 - test_size)*len(data_x))
20 train_x = []
21 train_y = []
22 while(len(train_x)<train_size):
23 index = randrange(len(data_x))
24 train_x.append(data_x.pop(index))
25 train_y.append(data_y.pop(index))
26 return train_x,train_y,data_x,data_y
```

NORMALIZE DATA

```
27 def statistics(x):
28 cols = list(zip(*x))
29 stats = []
30 for e in cols:
31 stats.append([min(e),max(e)])
32 return stats
```

SCALE THE FEATURES

```
33 def scale(x, stat):
34 for row in x:
35 for i in range(len(row)):
36 row[i] = (row[i] - stat[i][0])/(stat[i][1] - stat[i][0])
```

MULTICLASS CLASSIFICATION ONE VS ALL

```
37 def one_vs_all_cols(s):
38 m = list(set(s))
39 m.sort()
40 for i in range(len(s)):
41 new = [0]*len(m)
42 new[m.index(s[i])] = 1
43 s[i] = new
44 return m
```

COMPUTE THETA TRANSPOSE X FEATURE VECTOR

```
45 def ThetaTX(Q,X):
46 det = 0.0
47 for i in range(len(Q)):
48 det += X[i]*Q[i]
49 return det
```

COMPUTE COST FOR NEGATIVE CLASS

```
50 def LinearSVM_cost0(z):
51 if(z < -1):
52 return 0
53 return z + 1
```

COMPUTE COST FOR POSITIVE CLASS

```
54 def LinearSVM_cost1(z):
55 if(z > 1):
56 return 0
57 return -z + 1
```

CALCULATE SVM COST

```
58 def cost(theta,c,x,y):
59 cost = 0.0
60 for i in range(len(x)):
61 z = ThetaTX(theta[c], x[i])
62 cost += y[i]*LinearSVM_cost1(z) + (1 - y[i])*LinearSVM_cost0(z)
63 return cost
```

CALCULATE SIGMOID

```
64 def sigmoid(z):
65 return 1.0/(1.0 + exp(-z))
```

RETURN PREDICTIONS USING TRAINED WEIGHTS

```
66 def predict(data,theta):
67 predictions = []
68 count = 1
69 for row in data:
70 hypothesis = []
71 multiclass_ans = [0]*len(theta)
72 for c in range(len(theta)):
73 z = ThetaTX(row,theta[c])
74 hypothesis.append(sigmoid(z))
75 index = hypothesis.index(max(hypothesis))
```

```
multiclass_ans[index] = 1
predictions.append(multiclass_ans)
count+=1
return predictions
```

SPLIT DATASET INTO TRAIN N TEST WITH RASIO 70:30

```
def shuffle_split_data(X, y):
arr_rand = np.random.rand(X.shape[0])
split = arr_rand < np.percentile(arr_rand, 70)
X_train = X[split]
y_train = y[split]
X_test = X[~split]
y_test = y[~split]
print(len(X_train), len(y_train), len(X_test), len(y_test))
return X_train, y_train, X_test, y_test
```

GENERETE CONFUSSION METRICS BASE ON TRUE VS PREDICTED LABEL

```
def compute_confusion_matrix(true, pred):
K = len(np.unique(true))
result = np.zeros((K, K))
TP, TN, FP, FN = [], [], [], []
for i in range(len(true)):
if true[i] == 0 and pred[i] == 0:
TP.append(1)
TN.append(0)
FP.append(0)
FN.append(0)
elif true[i] == 1 and pred[i] == 1:
    TP.append(0)
    TN.append(1)
    FP.append(0)
    FN.append(0)
    elif true[i] == 0 and pred[i] == 1:
    TP.append(0)
    TN.append(0)
    FP.append(1)
    FN.append(0)
    elif true[i] == 1 and pred[i] == 0:
    TP.append(0)
    TN.append(0)
    FP.append(0)
    FN.append(1)
    result[true[i]][pred[i]] += 1
    print("Total TP is : ", result[0][0])
    print("Total TN is : ", result[1][1])
    print("Total FN is : ", result[0][1])
    print("Total FP is : ", result[1][0])
    return result
```

GENERATE ACCURACY SCORE

```
    def accuracy_score(predicted, actual):
    n = len(predicted)
    correct = 0
    for i in range(n):
    if (predicted[i] == actual[i]):
    correct += 1
    return correct / n
```

SPLIT DATASET BASE ON ATTRIBUTE VALUE WITH RANDOM FOREST

```
127      def test_split(dataset, idx, val):
128      left, right = [], []
129      for row in dataset:
130      if row[idx] < val:
131      left.append(row)
132      else:
133      right.append(row)
134      return left, right
```

CALCULATE GINI INDEX FOR SPLITTED DATASET

```
135      def gini_index(groups_classes, class_values):
136      gini = 0.0
137      for class_value in class_values:
138      for groups in groups_classes:
139      size = len(groups)
140      if size == 0:
141      continue
142      proportion = [row[-1] for row in groups].count(class_value) /
   float(size)
143      gini += (proportion * (1.0 - proportion))
144      return gini
```

APPLY BEST SPLIT POINT FOR DATASET

```
145      def get_split(dataset, n_features):
146      features = []
147      while len(features) < n_features:
148      index = randrange(len(dataset[0])-1)
149      if index not in features:
150      features.append(index)
151      class_values = list(set(row[-1] for row in dataset))
152      b_score  = 9999
153      b_index  = None
154      b_value  = None
155      b_groups = None
156      for col_index in features:
157      for row in dataset:
158      col_value = row[col_index]
159      groups = test_split(dataset, col_index, col_value)
160      gini = gini_index(groups, class_values)
161      if gini < b_score:
162      b_index  = col_index
163      b_value  = col_value
164      b_score  = gini
165      b_groups = groups
166      return {'index':b_index, 'value':b_value, 'groups':b_groups}
```

RETURN LABEL WITH THE HIGHEST FREQUENCY

```
167      def to_terminal(group):
168      labels = [row[-1] for row in group]
169      return max(set(labels), key=labels.count)
170      def split(node, max_depth, min_size, n_features, depth):
171      left, right = node['groups']
172      del(node['groups'])
173      if not left or not right:
174      node['left'] = node['right'] = to_terminal(left + right)
```

```
175      return
176      if depth >= max_depth:
177      node['left']  = to_terminal(left)
178      node['right'] = to_terminal(right)
179      return
180      if len(left) <= min_size:
181      node['left'] = to_terminal(left)
182      else:
183      node['left'] = get_split(left, n_features)
184      split(node['left'], max_depth, min_size, n_features, depth+1)
185      if len(right) <= min_size:
186      node['right'] = to_terminal(right)
187      else:
188      node['right'] = get_split(right, n_features)
189      split(node['right'], max_depth, min_size, n_features, depth+1)
```

BUILT OUR RANDOM FOREST TREES

```
190      def build_tree(train, max_depth, min_size, n_features):
191      root = get_split(train, n_features)
192      split(root, max_depth, min_size, n_features, 1)
193      return root
```

RANDOM SAMPLING WITH REPLACEMENT

```
194      def subsample(dataset, ratio_value):
195      sample_data = []
196      n_sample = round(len(dataset) * ratio_value)
197      while len(sample_data) < n_sample:
198      idx = randrange(len(dataset))
199      sample_data.append(dataset[idx])
200      return sample_data
```

PREDICTION BASE ON NODE & ROW

```
201      def predict(node, row):
202      if row[node['index']] < node['value']:
203      if isinstance(node['left'], dict):
204      return predict(node['left'], row)
205      else:
206      return node['left']
207      else:
208      if isinstance(node['right'], dict):
209      return predict(node['right'], row)
210      else:
211      return node['right']
```

PREDICTION WITH A LIST OF BAGGED TREES

```
212      def bagging_predict(trees, row):
213      predictions = [predict(tree, row) for tree in trees]
214      return max(set(predictions), key=predictions.count)
```

MAIN FUNCTION OF RANDOM FOREST

```
215      def
   random_forest(train,test,max_depth,min_size,sample_size,n_trees,n_fea
   tures:
216      trees = []
217      for i in range(n_trees):
```

```
218       sample = subsample(train, sample_size)
219       tree = build_tree(sample, max_depth, min_size, n_features)
220       trees.append(tree)
221       predictions = [bagging_predict(trees, row) for row in test]
222       return(predictions)
```

EXTERNAL FUNCTION FOR CROSS_VALIDATION

```
223       def cross_validation_split(dataset, n_folds):
224       dataset_copy = copy.copy(dataset)
225       fold_size = int(len(dataset_copy)/n_folds)
226       folds = []
227       for i in range(n_folds):
228       fold = []
229       while len(fold) < fold_size:
230       index = randrange(len(dataset_copy))
231       fold.append(dataset_copy.pop(index))
232       folds.append(fold)
233       return folds
```

GET ACCURACY SCORE

```
234       def accuracy_metric(actual, predicted):
235       correct = 0
236       for i in range(len(actual)):
237       if actual[i] == predicted[i]:
238       correct += 1
239       return correct / float(len(actual)) * 100.0
```

EVALUATION ALGORITHM

```
240       def evaluate_algorithm(dataset,algorithm,params,n_folds=5):
241       folds = cross_validation_split(dataset, n_folds)
242       scores = []
243       for fold in folds:
244       trainset = copy.copy(folds)
245       trainset.remove(fold)
246       trainset = sum(trainset, [])
247       testset = copy.copy(fold)
248       predicted = algorithm(trainset, testset, **params)
249       actual = [row[-1] for row in fold]
250       accuracy = accuracy_metric(actual, predicted)
251       scores.append(accuracy)
252       return scores
```

DROP NAN VALUES

```
253       df = df.dropna()
254       df = df.iloc[:50000, :]
```

ITERATION (Accuracy, Precision, Recall and F1 Score)

```
255       for x in range(10000, df.shape[0]+10000, 10000):
256       X, Y = df.iloc[:x, 1:], df.iloc[:x, 0]
257       X_train, y_train, X_test, y_test = shuffle_split_data(X, Y)
258       model = load("random_forest.py")
259       label = model.predict(X_test)
260       res = compute_confusion_matrix(y_test.values, label)
261       TP, FP, FN, TN = res[0][0], res[1][0], res[0][1], res[1][1]
262       PRE, RE = (TP/(TP+FP)), (TP/(TP+FN))
```

```
263       F1 = ((2 * PRE * RE)/(PRE + RE))
264       support.append(x)
265       accuracy.append(accuracy_score(y_test.values, label))
266       pre.append(PRE)
267       re.append(RE)
268       f1.append(F1)
269       print("Running Random Forest and SVM for Comparison Customer
    Satisfication")
270       print("Random Forest")
271       print("Accuracy\t", accuracy_score(y_test.values, label))
272       print("Precision\t", PRE)
273       print("Recall\t\t", RE)
274       print("F1-Score\t", F1)
275       print()
```

APPLY SUPPORT VECTOR MACHINE

```
276       model_1 = load("svm.py")
277       label = model_1.predict(X_test)
278       res = compute_confusion_matrix(y_test.values, label)
279       TP, FP, FN, _ = res[0][0], res[1][0], res[0][1], res[1][1]
280       PRE, RE = (TP/(TP+FP)), (TP/(TP+FN))
281       F1 = ((2 * PRE * RE)/(PRE + RE))
282       accuracy_.append(accuracy_score(y_test.values, label))
283       pre_.append(PRE)
284       re_.append(RE)
285       f1_.append(F1)
286       print("Support Vector Machine")
287       print("Accuracy\t", accuracy_score(y_test.values, label))
288       print("Precision\t", PRE)
289       print("Recall\t\t", RE)
290       print("F1-Score\t", F1)
291       print()
```

PLOT GRAPH

```
292       b = mpatches.Patch(color='#0F52BA', label='Random Forest')
293       t = mpatches.Patch(color='#B61919', label='Linear SVM')
294       fig, ax =  plt.subplots(2, 2, figsize=(20,10))
295       ax[0,0].plot(support, accuracy)
296       ax[0,0].plot(support, accuracy_)
297       ax[0,0].legend(handles=[b, t])
298       ax[0,0].title.set_text("Accuracy Score Comparison")
299       ax[0,1].plot(support, pre)
300       ax[0,1].plot(support, pre_)
301       ax[0,1].legend(handles=[b, t])
302       ax[0,1].title.set_text("Precision Score Comparison")
303       ax[1,0].plot(support, re)
304       ax[1,0].plot(support, re_)
305       ax[1,0].legend(handles=[b, t])
306       ax[1,0].title.set_text("Recall Score Comparison")
307       ax[1,1].plot(support, f1)
308       ax[1,1].plot(support, f1_)
309       ax[1,1].legend(handles=[b, t])
310       ax[1,1].title.set_text("F1-Score Score Comparison")
```

PLOT SHOW

```
311       fig.savefig("result_save.png")
```

PAPER NAME
18.K1.0045_Wahono Soesantio

AUTHOR
Wahono Soesantio

WORD COUNT
11958 Words

CHARACTER COUNT
55923 Characters

PAGE COUNT
46 Pages

FILE SIZE
140.9KB

SUBMISSION DATE
Apr 28, 2022 2:58 PM GMT+7

REPORT DATE
Apr 28, 2022 3:00 PM GMT+7

16% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 11% Internet database
- 7% Publications database
- Crossref database
- Crossref Posted Content database
- 10% Submitted Works database

Excluded from Similarity Report

- Bibliographic material
- Quoted material
- Cited material
- Small Matches (Less then 10 words)