Mini project 3

IEE 520 Data Mining

Submitted by:
Sanket Bhale
Indraneel Phirke
Kiarsh Ghasemzadeh

1. Select a dataset of your own choice of sufficient size, at least 500 rows, (prefer over a thousand rows), and at least 10 predictor attributes (prefer at least 20), and a target attribute. The target should be categorical. Build a decision tree classifier for your data in Python, and select hyperparameters through the following steps:

• Hold back 30% of your data for testing.

• For the remaining data, use 5 fold cross validation to select hyperparameters.

• In each fold, use the training data to predict the test data for values of hyperparameters.

(a) Provide your code.

(b) Plot the test error rate over each fold (5 points) versus hyperparameters. Comment on the value of hyperparameters selected and why you selected these. Consider error rate and complexity.

(c) Estimate the generalization error rate for your final model.

SOLUTIONS:

(a) **Code for Question 1 and 2:**

```
# For compatibility with Python 2
from __future__ import print_function

# To load datasets
from sklearn import datasets

# To import the classifier (K-Nearest Neighbors Classifier and Regressor)
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

# To measure accuracy
from sklearn import metrics

from sklearn.model_selection import KFold

# To support plots
from ipywidgets import interact
import ipywidgets as widgets
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap


# To load datasets
from sklearn import datasets

# To import the classifier (K-Nearest Neighbors Classifier and Regressor)
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

# To measure accuracy
from sklearn import metrics

from sklearn.model_selection import KFold

```

```
# To support plots
from ipywidgets import interact
import ipywidgets as widgets
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import seaborn as sn
from sklearn.model_selection import GridSearchCV
import numpy as np

# To display all the plots inline
%matplotlib inline

# To splite the data
from sklearn.model_selection import train_test_split, cross_val_score
'''''
seed = 2357
#from google.colab import drive
drive.mount('/content/drive')
# To increase quality of figures
plt.rcParams["figure.figsize"] = (20, 10)
'''
#LOAD DATA
import pandas as pd

chess_data = pd.read_csv("chess.csv")
print(chess_data.shape)
```

```
(3196, 37)
```

```
chess_data.describe()
```

	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	...
count	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	...
mean	0.111702	0.070401	0.037547	0.100751	0.333855	0.461202	0.366083	0.217772	0.380476	0.303817	...
std	0.315049	0.255861	0.190128	0.301046	0.471662	0.498570	0.481808	0.412796	0.485580	0.459977	...
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	...
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...

	Column28	Column29	Column30	Column31	Column32	Column33	Column34	Column35	Column36	Column37
count	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000	3196.000000
mean	0.000313	0.014706	0.042553	0.176783	0.054756	0.379224	0.627972	0.733730	0.246871	0.522215
std	0.017689	0.120392	0.201879	0.381545	0.227539	0.485270	0.483421	0.442076	0.431259	0.499584
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	0.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 37 columns

```
from sklearn import preprocessing
#le = preprocessing.LabelEncoder()
data = chess_data.apply(le.fit_transform)
data = chess_data.to_numpy()
```

```
X = data[:, :-1]
y = data[:, -1]
print (len(X))
print (y)
print (X)
```

```
3196
[1 1 1 ... 0 0 0]
[[0 0 0 ... 1 1 0]
 [0 0 0 ... 1 1 0]
 [0 0 0 ... 1 1 0]
 ...
 [1 0 0 ... 1 0 0]
 [1 0 1 ... 0 0 0]
 [1 0 1 ... 0 0 0]]
```

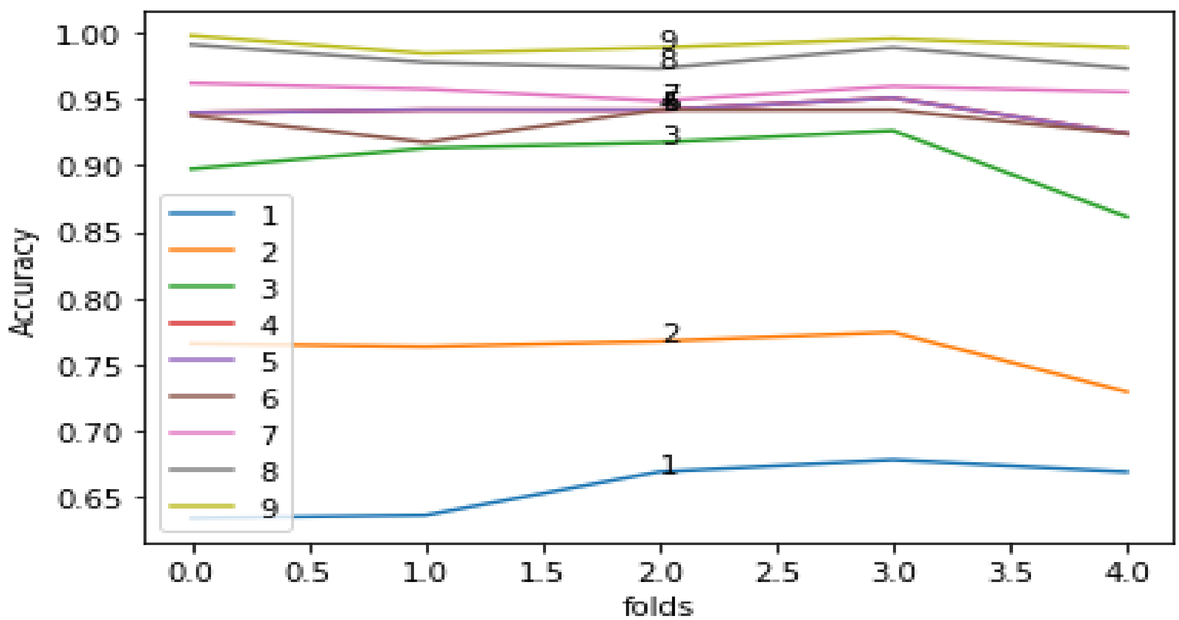
#Hold back 30% of data for testing

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=seed)
# # For the remaining data, using 5 fold cross validation to select hyperparameters.
from sklearn.model_selection import GridSearchCV, cross_validate
kfold = KFold(n_splits= 5, shuffle = True, random_state =520)
yhat = np.zeros((X_train.shape[0], ))
max_depth = list()
List = list()
for i in range(1, 10):
    folds = list()
    for train, test in kfold.split(X_train, y_train):
        model = DecisionTreeClassifier(max_depth = i,random_state=520, class_weight='balanced')
        model.fit(X_train[train], y_train[train])
        yhat[test] = model.predict(X_train[test])
        value = (metrics.accuracy_score(y_train[test], yhat[test]))
        folds.append(value)
    List.append(folds)
    max_depth.append(i)
print(max_depth)
print(List)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[[0.6339285714285714, 0.6361607142857143, 0.668903803131991, 0.6778523489932886, 0.668903803131991],
[0.765625, 0.7633928571428571, 0.767337807606264, 0.7740492170022372, 0.7293064876957495],
[0.8973214285714286, 0.9129464285714286, 0.9172259507829977, 0.9261744966442953, 0.8612975391498882],
[0.9397321428571429, 0.9419642857142857, 0.941834451901566, 0.9507829977628636, 0.9239373601789709],
[0.9397321428571429, 0.9419642857142857, 0.941834451901566, 0.9507829977628636, 0.9239373601789709],
[0.9375, 0.9174107142857143, 0.941834451901566, 0.941834451901566, 0.9239373601789709],
[0.9620535714285714, 0.9575892857142857, 0.9485458612975392, 0.959731543624161, 0.9552572706935123],
```

```
[0.9910714285714286, 0.9776785714285714, 0.9731543624161074, 0.9888143176733781,
0.9731543624161074],
[0.9977678571428571, 0.984375, 0.9888143176733781, 0.9955257270693513, 0.9888143176733781]]
```

```
for index, value in enumerate(List):
    plt.plot(value, label=index+1)
    plt.text(2.0, value[2], index+1)
    plt.legend()
plt.xlabel('folds')
plt.ylabel('Accuracy')
plt.savefig('plot2.png')
plt.show()
```

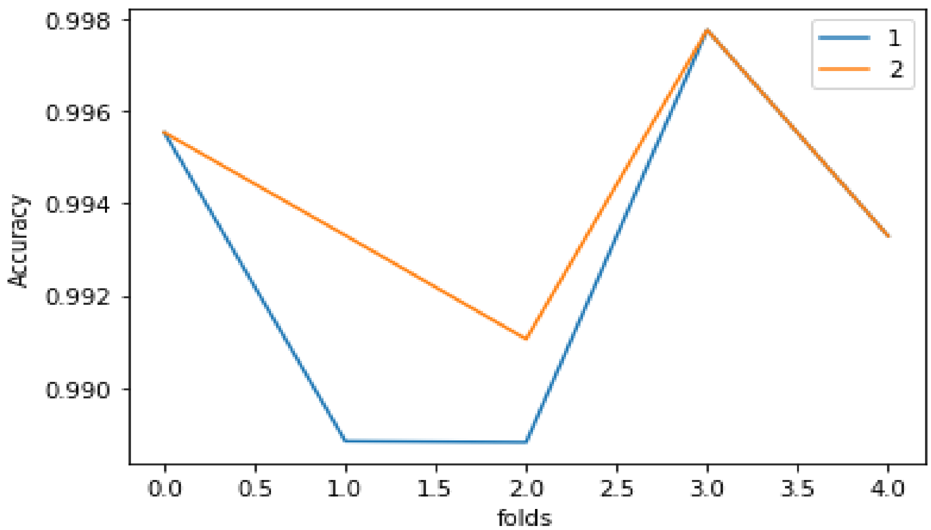



```
min_length = list()
List = list()
for i in range(2,5,2):
    kfolds = list()
    for train, test in kfold.split(X_train, y_train):
        model = DecisionTreeClassifier(min_samples_split=
i,class_weight="balanced" ,random_state=520)
        model.fit(X_train[train], y_train[train])
        yhat[test] = model.predict(X_train[test])
        score = (metrics.accuracy_score(y_train[test], yhat[test]))
        kfolds.append(score)
    List.append(kfolds)
    min_length.append(i)
print(min_length)
print(List)
```

```
[2, 4]
[[0.9955357142857143, 0.9888392857142857, 0.9888143176733781,
0.9977628635346756, 0.9932885906040269], [0.9955357142857143,
```

```
0.9933035714285714, 0.9910514541387024, 0.9977628635346756,
0.9932885906040269]]
```

```python
for index, value in enumerate(List):
    plt.plot(value, label=index+1)
    plt.legend()
plt.xlabel('folds')
plt.ylabel('Accuracy')
#plt.savefig('plot2.png')
plt.show()
```

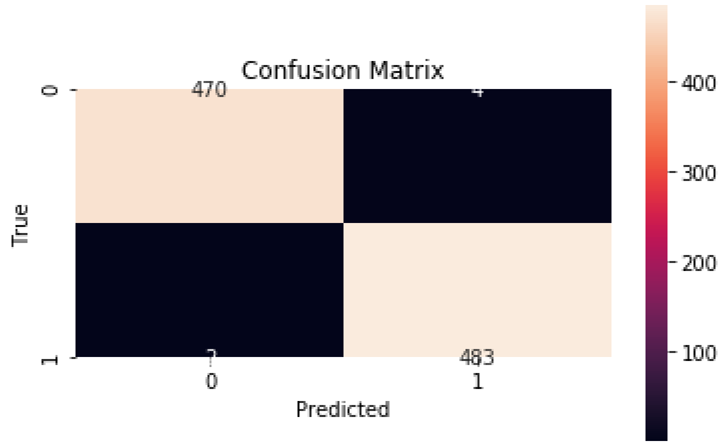



```python
model_tree = GridSearchCV(DecisionTreeClassifier(random_state=seed),
                          cv=5,
                          param_grid={
                              "max_depth": list(range(1, 10, 1)),
                              "min_samples_split": list(range(2, 5, 2))
                          })
model_tree.fit(X_train, y_train)
print('The parameters found by CV search:')
print(model_tree.best_params_)
y_test_hat = model_tree.predict(X_test)

print('Accuracy:', metrics.accuracy_score(y_test, y_test_hat))

cm = metrics.confusion_matrix(y_test, y_test_hat)
ax = sn.heatmap(cm, annot=True, fmt='g', square=True)
ax.set_xlabel('Predicted')
ax.set_ylabel('True')
ax.set_title('Confusion Matrix')
plt.show()
print(cm)
```

```
The parameters found by CV search:
{'max_depth': 9, 'min_samples_split': 2}
Accuracy: 0.9937434827945777
```




```
[[470   4]
 [  2 483]]
```

```
### <center> Question 2 code </center>
  # Random Forest Classifier
model_tree = GridSearchCV(RandomForestClassifier(random_state=520,
class_weight='balanced'),
                          cv=5,return_train_score = True,
                          param_grid={
                              "max_depth": list(range(1, 10, 2)),
                              "min_samples_split": list(range(2, 5, 2))
                          })
model_tree.fit(X_train, y_train)
print('The parameters found by CV search:')
print(model_tree.best_params_)
```

```
The parameters found by CV search:
{'max_depth': 9, 'min_samples_split': 4}
```

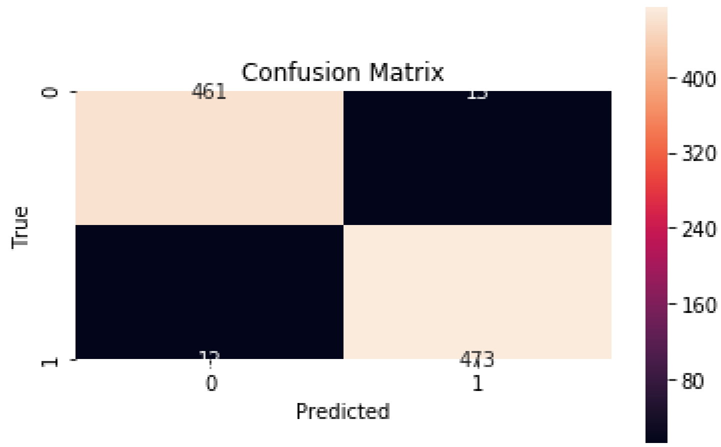
```
model_forest = RandomForestClassifier(n_estimators=100, random_state =520,
                                      max_depth=9,
                                      min_samples_split=4,
                                      n_jobs=-1, class_weight="balanced")

model_forest.fit(X_train, y_train)
ytest_hat = model_forest.predict(X_test)

print('Accuracy:', metrics.accuracy_score(y_test, ytest_hat))

cm = metrics.confusion_matrix(y_test, ytest_hat)
ax = sn.heatmap(cm, annot=True, fmt='g', square=True)
ax.set_xlabel('Predicted')
ax.set_ylabel('True')
ax.set_title('Confusion Matrix')
print(cm)
plt.show()
```

Accuracy: 0.9739311783107404
[[461 13]
 [12 473]]



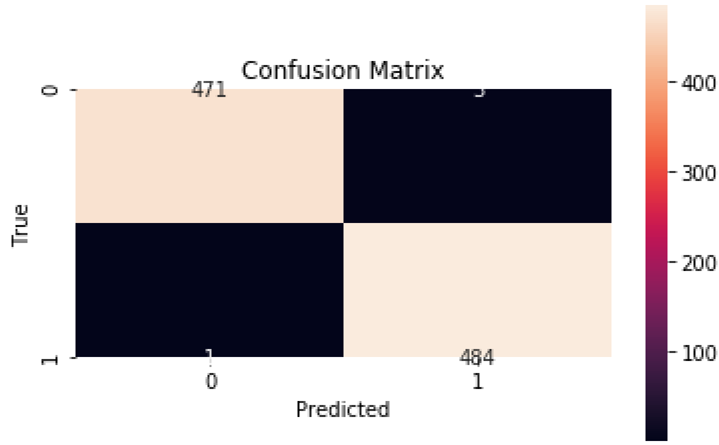

```python
#AdaBoost Classifier
model_adaboost = AdaBoostClassifier(DecisionTreeClassifier(random_state=520,
                                                           max_depth=9,

min_samples_split=4))
model_adaboost.fit(X_train, y_train)
y_test_hat = model_adaboost.predict(X_test)

print('Accuracy:', metrics.accuracy_score(y_test, y_test_hat))

cm = metrics.confusion_matrix(y_test, y_test_hat)
ax = sn.heatmap(cm, annot=True, fmt='g', square=True)
ax.set_xlabel('Predicted')
ax.set_ylabel('True')
ax.set_title('Confusion Matrix')
print(cm)
plt.show()
```

Accuracy: 0.9958289885297185
[[471   3]
 [1 484]]



(b) Plot the test error rate over each fold (5 points) versus hyperparameters. Comment on the value of hyperparameters selected and why you selected these. Consider error rate and complexity.

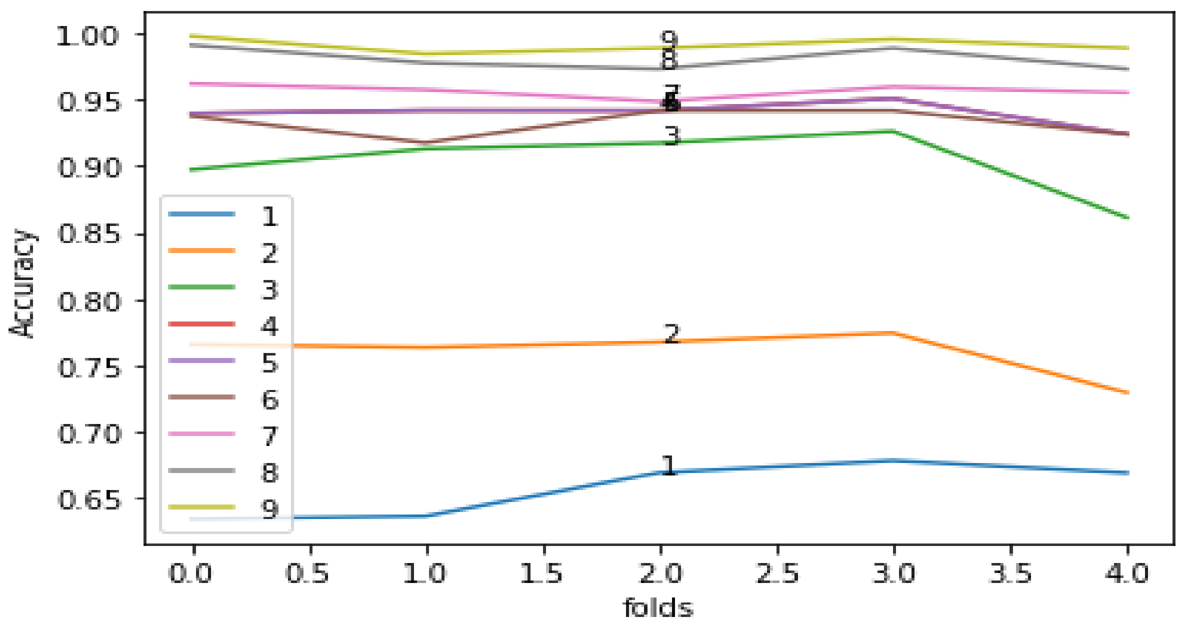


Figure: Plot of accuracy vs Number of folds for hyperparameter (Max depth)

As seen from the above figure, the accuracy increases with increase in number of depths. After a certain increase in the number of depths, the accuracy tends to stay constant. For our data set, this was 9 (Max. number of depths).

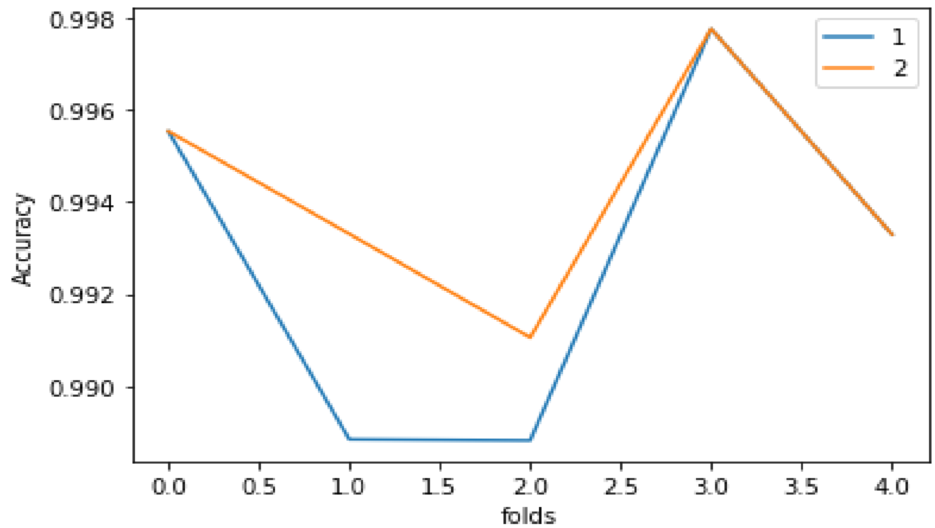


Figure: Plot of accuracy vs Number of folds for hyperparameter (Min. samples split)

As seen accuracy increases with increase in minimum samples splits. (In the above diagram the blue line denotes sample splits as 2 and orange denotes sample splits 4)

However to choose the optimum hyperparameters, we used Gridsearch.**Using grid search we found the optimal max depth= 9 and min sample split to be equal to 2**. The reason is as we increase max depth the model becomes more accurate but more complex. Which is not necessarily a good thing (over fits the data). Therefore, the accuracy of prediction for test data would decrease (error rate will increase). The same result goes for the min sample split. If we choose min sample split to be a higher value then we will end up with overfitting and lower than 2 could result in underfitting the data (in both cases the error rate on test data increases)

(c) Estimate the generalization error rate for your final model.

```
Accuracy: 0.9937434827945777
```

Generalization error for final model (with max. depth= 9, min samples split= 2) =1-Accuracy= **0.0062565172**

The final model is obtained by using the hyperparameters generated by the GridSearch which gave Max. depth as 9 and min. samples split as 2.

2. For the data you selected for the first question, use Python to construct an ensemble classifier. You may choose any ensemble classifier.
• Hold back the same 30% of test data as in the first question.
• Select parameters for your ensemble based on the remaining data and attempt to build the best possible model.
(a) When your model is complete, compare the error rate on the 30% test data from the ensemble and the best decision tree classifier you obtained in the first question. Compute confusion matrices for the test data from each model and comments on any differences in performance.
(b) Which model do you prefer and why? Be brief and clear.

Solutions:

The code is continued in code provided in question 1 itself. We used Random Forest Ensemble and AdaBoost Ensemble Classifiers.

Classifier	Test Error	Accuracy	Confusion Matrix
Decision Tree	0.0062565172	0.9937434827945777	[[470 4] [2 483]]
Random Forest	0.02606882168	0.9739311783107404	[[461 13] [12 473]]
AdaBoost	0.00417101147	0.9958289885297185	[[471 3] [1 484]]

As seen from the above comparison table, model obtained from decision tree and AdaBoost Ensemble classifier perform very close and both have accuracy values close to each other. They perform better than the random forest for our data set. The number of wrongly predicted values for the sample is also close (For Decision tree=4+2=6, For AdaBoost= 1+3=4).

Based on the above comparison we would recommend to use AdaBoost Ensemble Classifier for our dataset. The hyperparameters selected are max depth=9 and Min samples splits=4. These optimal values of parameters are obtained after performing Gridsearch.

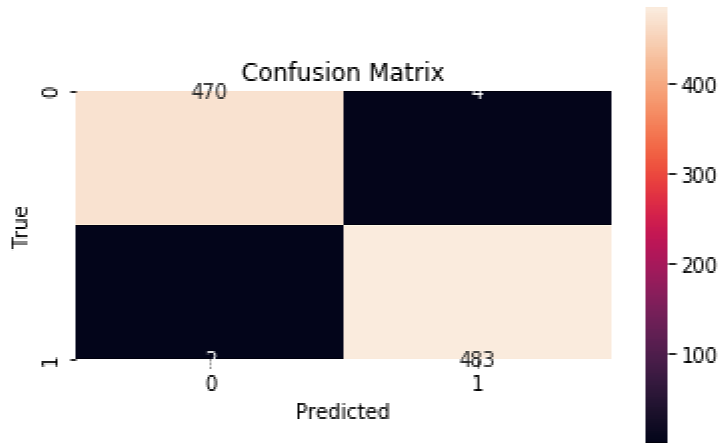


Figure: Confusion Matrix Decision tree

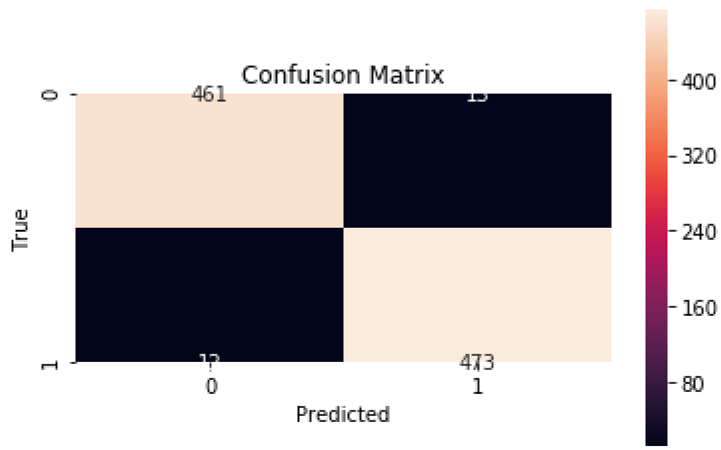


Figure: Confusion Matrix RandomForrest classifier

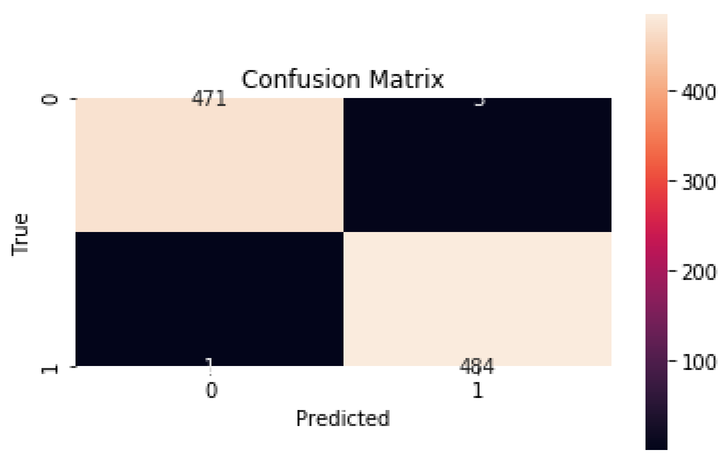


Figure: Confusion Matrix AdaBoost Classifier

3. Read the notes on Class Imbalance. For the following data build a decision tree classifier. Hold back 30% of data for testing. Use the default parameter settings, but you may limit the depth of the tree. Use the same depth for all the questions below.

(a) Use the 30% test data to construct a confusion matrix and calculate the FPR, FNR, and balanced error rate.

(b) Adjust the class weights to put equal total weight on each class. What weight did you use for each class? Build a decision tree classifier with the default parameter settings. Use the 30% test data to construct a confusion matrix and calculate the FPR, FNR, and balanced error rate. Comment on any differences from the unweighted case.

Solutions:a)

```
# To load datasets
from sklearn import datasets

# To import the models (Decision Tree Classifier and Regressor)
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

# To display a tree
from sklearn.tree import plot_tree

# To measure accuracy
from sklearn import metrics

from sklearn.model_selection import cross_validate, KFold

# To support plots
from ipywidgets import interact
import ipywidgets as widgets
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

import numpy as np
import pandas as pd

import math

# To display all the plots inline
%matplotlib inline
```

```
# To increase quality of figures
plt.rcParams["figure.figsize"] = (30, 10)
```

```
#import data
import pandas as pd
path = "Miniproject 3 2019 data.csv"
data = pd.read_csv(path)
```

```
print(data.shape)
```

```
(3679, 16)
```

```
data.describe()
```

Out[22]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
count	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679
mean	0.515516	0.304702	0.123947	0.013319	0.011688	0.039413	0.014406	0.014134	0.014950	0.060614	0.062245	0
std	0.189588	0.460344	0.329566	0.114652	0.107492	0.194602	0.119174	0.118061	0.121368	0.238654	0.241633	0
min	0.010000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.360000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
50%	0.540000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
75%	0.670000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
max	0.940000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
data = data.apply(le.fit_transform)
data = data.to_numpy()
X = data[:, 0:-1]
y = data[:, -1]
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

```
# To split the data
from sklearn.model_selection import train_test_split, cross_val_score

seed = 2357
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=seed)
```

```
model = DecisionTreeClassifier(max_depth=24,random_state=520)
                                               # class_weight='balanced')
model.fit(X_train, y_train)
yhat = model.predict(X_test)
test_error = 1- metrics.accuracy_score(yhat, y_test)
print(test_error)
```

```
0.0625
```

```
print ('accuracy=',metrics.accuracy_score(yhat, y_test))
```

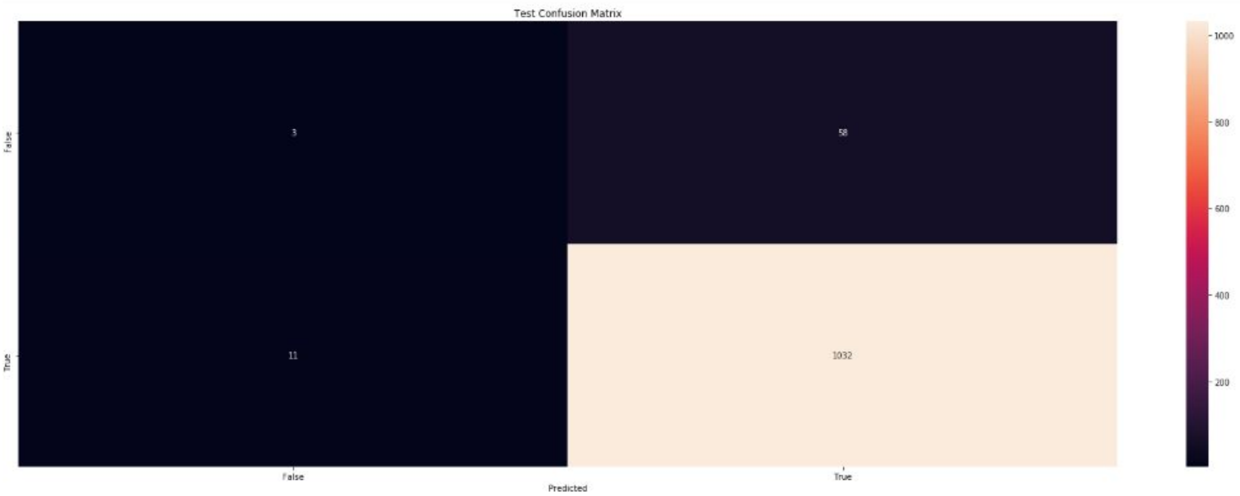
```
accuracy= 0.9375
```

```
from pandas_ml import ConfusionMatrix
cm = ConfusionMatrix(y_test, yhat)
print(cm)

cm.print_stats()
```

```
ax = cm.plot(backend='seaborn', annot=True, fmt='g')
ax.set_title('Test Confusion Matrix')
plt.show()
```

```
Predicted  False  True  __all__
Actual
False          3    58       61
True          11  1032     1043
__all__       14  1090     1104
population: 1104
P: 1043
N: 61
PositiveTest: 1090
NegativeTest: 14
TP: 1032
TN: 3
FP: 58
FN: 11
TPR: 0.9894534995206136
TNR: 0.04918032786885246
PPV: 0.9467889908256881
NPV: 0.21428571428571427
FPR: 0.9508196721311475
FDR: 0.05321100917431193
FNR: 0.010546500479386385
ACC: 0.9375
F1_score: 0.9676511954992968
MCC: 0.078885564928465
informedness: 0.03863382738946597
markedness: 0.16107470511140232
prevalence: 0.9447463768115942
LRP: 1.0406321288061626
LRN: 0.21444550974752316
DOR: 4.852664576802508
FOR: 0.7857142857142857
```



Confusion Matrix

b)

```
# To load datasets
from sklearn import datasets

# To import the models (Decision Tree Classifier and Regressor)
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

# To display a tree
from sklearn.tree import plot_tree

# To measure accuracy
from sklearn import metrics

from sklearn.model_selection import cross_validate, KFold

# To support plots
from ipywidgets import interact
import ipywidgets as widgets
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

import numpy as np
import pandas as pd

import math

# To display all the plots inline
%matplotlib inline
```

```
# To increase quality of figures
```

```
plt.rcParams["figure.figsize"] = (30, 10)
```

```
#import data
import pandas as pd
path = "Miniproject 3 2019 data.csv"
data = pd.read_csv(path)
```

```
print(data.shape)
```

```
(3679, 16)
```

```
data.describe()
```

Out[5]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
count	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679.000000	3679
mean	0.515516	0.304702	0.123947	0.013319	0.011688	0.039413	0.014406	0.014134	0.014950	0.060614	0.062245	0
std	0.189588	0.460344	0.329566	0.114652	0.107492	0.194602	0.119174	0.118061	0.121368	0.238654	0.241633	0
min	0.010000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.360000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
50%	0.540000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
75%	0.670000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
max	0.940000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
data = data.apply(le.fit_transform)
data = data.to_numpy()
X = data[:, 0:-1]
y = data[:, -1]
#from sklearn.preprocessing import StandardScaler
#scaler = StandardScaler()
#scaler.fit(X)
#X_scaled = scaler.transform(X)
```

```
# To splite the data
from sklearn.model_selection import train_test_split, cross_val_score

seed = 2357
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=seed)
```

```
model = DecisionTreeClassifier(max_depth=24,random_state=520
                                        ,class_weight='balanced')
model.fit(X_train, y_train)
yhat = model.predict(X_test)
test_error = 1- metrics.accuracy_score(yhat, y_test)
print(test_error)
```

```
0.3432971014492754
```

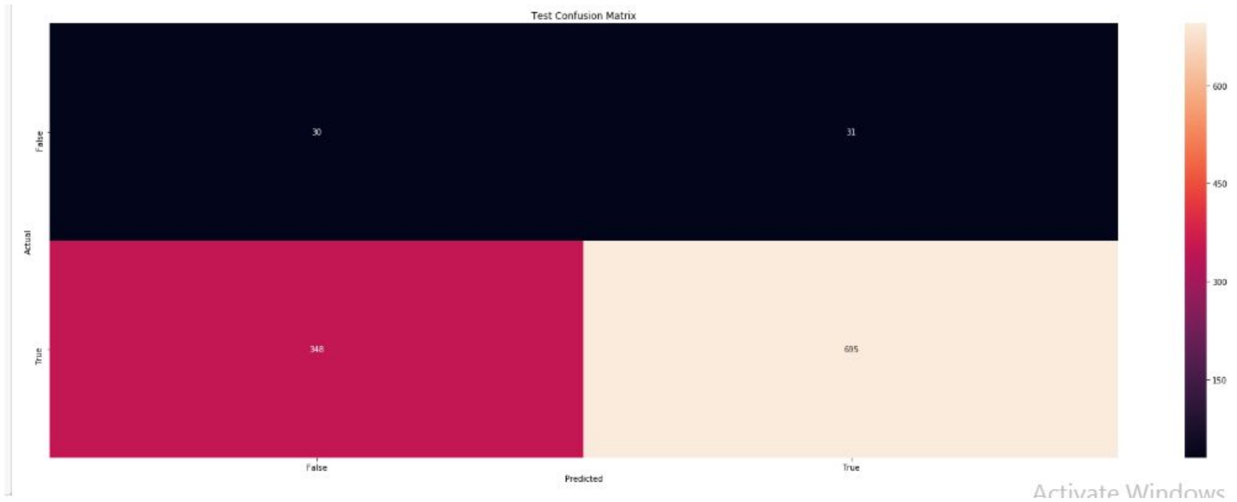
```
print ('accuracy=',metrics.accuracy_score(yhat, y_test))
```

```
accuracy= 0.6567028985507246
```

```python
from pandas_ml import ConfusionMatrix
cm = ConfusionMatrix(y_test, yhat)
print(cm)

cm.print_stats()
ax = cm.plot(backend='seaborn', annot=True, fmt='g')
ax.set_title('Test Confusion Matrix')
plt.show()
```

```
Predicted  False  True  __all__
Actual
False         30    31       61
True         348   695     1043
__all__      378   726     1104
population: 1104
P: 1043
N: 61
PositiveTest: 726
NegativeTest: 378
TP: 695
TN: 30
FP: 31
FN: 348
TPR: 0.6663470757430489
TNR: 0.4918032786885246
PPV: 0.9573002754820936
NPV: 0.07936507936507936
FPR: 0.5081967213114754
FDR: 0.04269972451790634
FNR: 0.3336529242569511
ACC: 0.6567028985507246
F1_score: 0.7857546636517807
MCC: 0.07614879424153626
informedness: 0.15815035443157344
markedness: 0.036665354847172926
prevalence: 0.9447463768115942
LRP: 1.3111990845266446
LRN: 0.6784276126558005
DOR: 1.9327030033370411
FOR: 0.9206349206349206
```



Confusion Matrix

	Non-balanced	Balanced
FPR	0.9508196721311475	0.5081967213114754
FNR	0.010546500479386385	0.3336529242569511
Accuracy	0.9375	0.6567028985507246

The balanced Decision tree classifier performs better than the unweighted. As seen from the above comparison table, the accuracy is less for the balanced decision tree classifier than the unbalanced.

Although accuracy obtained by balanced model is less ,the value of accuracy obtained for non-balanced model can be misleading as the data set has high variation in count of instances for the 2 classes in response variable.

Calculating weights:

We used the following formula to calculate weights on values of y= 0 and y=1:

Balanced weight for y'= Total number of samples/(number of classes * count of y')

Total number of samples=3679
Numbers (count) of 1 (in y)= 3488
Numbers(count) of 0 (in y)=191
Number of classes=2

To calculate weights for 0:
W_0 = 3679/(2*191) =9.6308
To calculate weights for 1
W_1 = 3679/(2*3488) = 0.52738