Assignment Week11

May 24, 2021

[374]:

```
import time
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

import pandas as pd

from sklearn.datasets import fetch_openml
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.utils import check_random_state
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix, plot_confusion_matrix, roc_curve, auc, precision_score,␣
 ↪recall_score
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

[21]:

```
df =pd.read_csv("Data_for_UCI_named.csv")
df
```

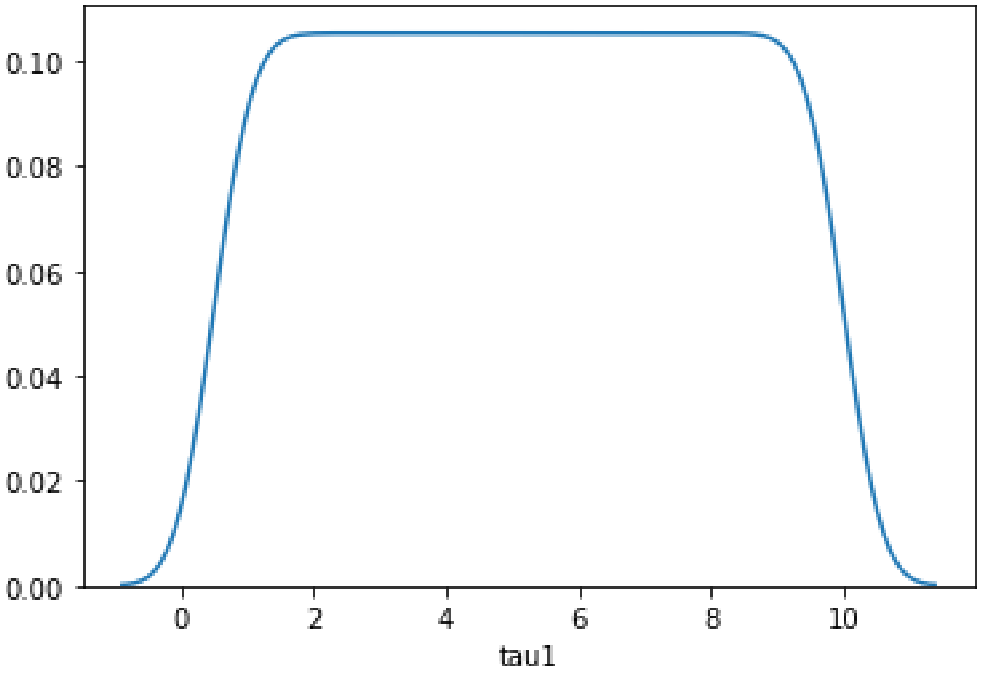
[21]:

```
          tau1      tau2      tau3      tau4        p1        p2        p3  \
0     2.959060  3.079885  8.381025  9.780754  3.763085 -0.782604 -1.257395
1     9.304097  4.902524  3.047541  1.369357  5.067812 -1.940058 -1.872742
2     8.971707  8.848428  3.046479  1.214518  3.405158 -1.207456 -1.277210
3     0.716415  7.669600  4.486641  2.340563  3.963791 -1.027473 -1.938944
4     3.134112  7.608772  4.943759  9.857573  3.525811 -1.125531 -1.845975
…          …         …         …         …         …         …         …
9995  2.930406  9.487627  2.376524  6.187797  3.343416 -0.658054 -1.449106
9996  3.392299  1.274827  2.954947  6.894759  4.349512 -1.663661 -0.952437
9997  2.364034  2.842030  8.776391  1.008906  4.299976 -1.380719 -0.943884
9998  9.631511  3.994398  2.757071  7.821347  2.514755 -0.966330 -0.649915
9999  6.530527  6.781790  4.349695  8.673138  3.492807 -1.390285 -1.532193

            p4        g1        g2        g3        g4      stab     stabf
```

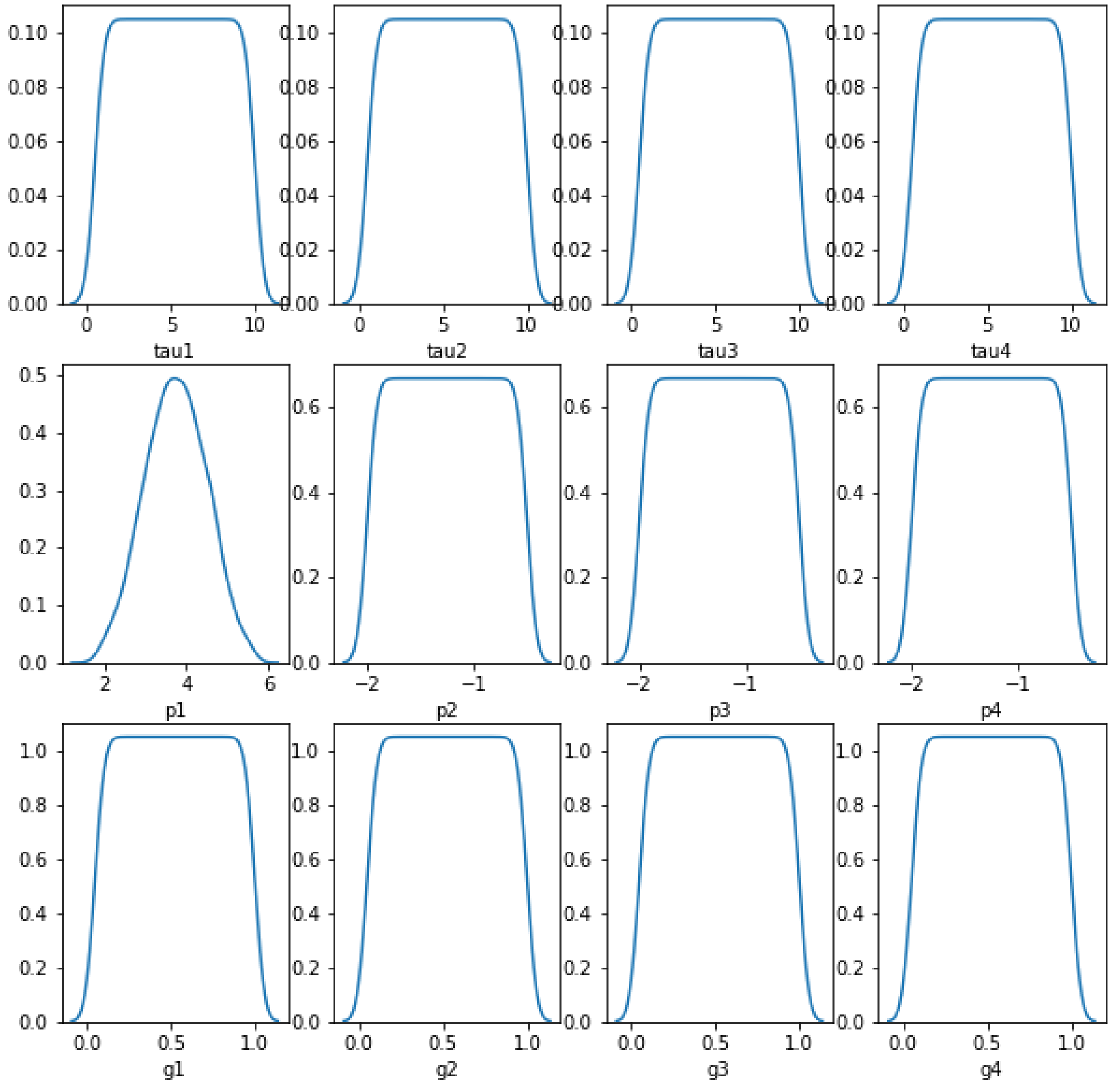
```
0    -1.723086  0.650456  0.859578  0.887445  0.958034  0.055347  unstable
1    -1.255012  0.413441  0.862414  0.562139  0.781760 -0.005957    stable
2    -0.920492  0.163041  0.766689  0.839444  0.109853  0.003471  unstable
3    -0.997374  0.446209  0.976744  0.929381  0.362718  0.028871  unstable
4    -0.554305  0.797110  0.455450  0.656947  0.820923  0.049860  unstable
…          …        …        …        …        …        …        …
9995 -1.236256  0.601709  0.779642  0.813512  0.608385  0.023892  unstable
9996 -1.733414  0.502079  0.567242  0.285880  0.366120 -0.025803    stable
9997 -1.975373  0.487838  0.986505  0.149286  0.145984 -0.031810    stable
9998 -0.898510  0.365246  0.587558  0.889118  0.818391  0.037789  unstable
9999 -0.570329  0.073056  0.505441  0.378761  0.942631  0.045263  unstable

[10000 rows x 14 columns]
```

[44]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.distplot(df.tau1, hist = False)
```

[44]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6d2a6af40>
```



[322]:
```python
figure = plt.figure(figsize = (10,10))
column_list =list(df.columns[:-3])
for i, v in enumerate(column_list):
    figure.add_subplot(3, 4, i+1)
    sns.distplot(df[v], hist = False)
```



[51]:
```
df.stabf.value_counts()
```

[51]:
```
unstable    6380
stable      3620
Name: stabf, dtype: int64
```

[119]:
```
df['Target'] = [1 if x== 'stable' else 0 for x in df.stabf]
```

[150]:
```
df.Target.value_counts()
```

[150]:
```
0    6380
1    3620
Name: Target, dtype: int64
```

[190]:
```
X = df.iloc[:,:-3]
y= df['Target']
```

[191]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size= 0.8)
```

[192]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[289]:
```
lr = LogisticRegression( solver = "saga")
```

[290]:
```
parameters = [{'C':[0.001, 0.01, 0.1,1], 'tol':[0.001, 0.01,0.05,0.1],␣
 ↪'penalty':['l1','l2']}]
```

[291]:
```
grid1 =GridSearchCV(lr, parameters, scoring = 'roc_auc', cv=4)
```

[292]:
```
grid1.fit(X, y)
```

[292]:
```
GridSearchCV(cv=4, estimator=LogisticRegression(solver='saga'),
             param_grid=[{'C': [0.001, 0.01, 0.1, 1], 'penalty': ['l1', 'l2'],
                          'tol': [0.001, 0.01, 0.05, 0.1]}],
             scoring='roc_auc')
```

[293]:
```
print("best parameter is: %s"%grid1.best_params_)
```

```
best parameter is: {'C': 1, 'penalty': 'l2', 'tol': 0.001}
```

[294]:
```
grid1.best_score_ # highest AUC value
```

[294]:
```
0.8906008070801366
```

[295]:
```
grid1.best_index_
```

[295]:
```
28
```

[296]:
```
grid1.cv_results_["mean_test_score"]
```

[296]:
```
array([0.78912382, 0.78913663, 0.78908537, 0.78895772, 0.81218137,
       0.81217461, 0.81172448, 0.81056184, 0.87472661, 0.87307799,
       0.86232737, 0.84927657, 0.87070178, 0.86938413, 0.8472102 ,
       0.83976047, 0.89054781, 0.88992951, 0.88083029, 0.8659594 ,
       0.89003776, 0.88908   , 0.87506157, 0.86329708, 0.89059561,
       0.88996467, 0.87920331, 0.87036925, 0.89060081, 0.88993852,
       0.87971787, 0.86771662])
```

```
[297]: grid1.cv_results_["std_test_score"][28]
```

```
[297]: 0.011710624735358218
```

```
[298]: print("average of auc across 4 folds for the best parameter: %.2f, with a std␣
        ↪dev of : %.2f"
            %(grid1.best_score_, grid1.cv_results_["std_test_score"][28]))
```

```
average of auc across 4 folds for the best parameter: 0.89, with a std dev of :
0.01
```

```
[299]: lr = LogisticRegression(C = 1, penalty = 'l2', solver ='saga', tol =0.001) #␣
        ↪Fitting a logistic Regression Model
```

```
[300]: lr.fit(X_train, y_train)
       y_predict = lr.predict(X_test)
```

1 Model 1 Logistic Regression Evaluation:

```
[301]: precision_score(y_test, y_predict, average ='weighted') # using weighted␣
        ↪because we have an imbalanced target data
```

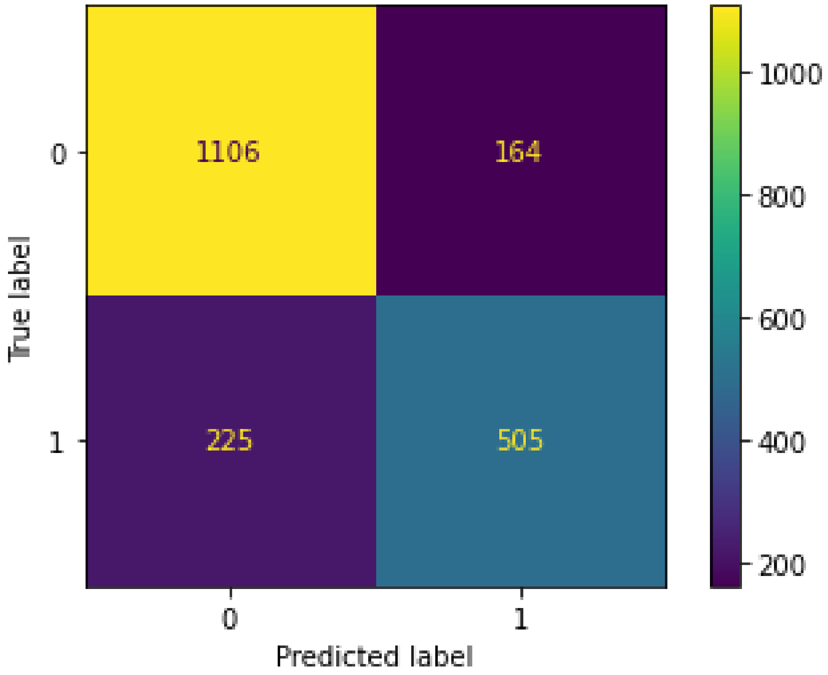
```
[301]: 0.8031790667300063
```

```
[302]: recall_score(y_test, y_predict, average = "weighted")
```

```
[302]: 0.8055
```

```
[303]: plot_confusion_matrix(lr, X_test, y_test)
```

```
[303]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
       0x7fb6d29c04f0>
```



[304]:
```
from sklearn.model_selection import cross_val_score
score = cross_val_score(lr, X,y, scoring ='roc_auc', cv=4)
```

[307]:
```
score.mean() # with cross validation, auc score is 0.89
```

[307]:
```
0.8906020194322728
```

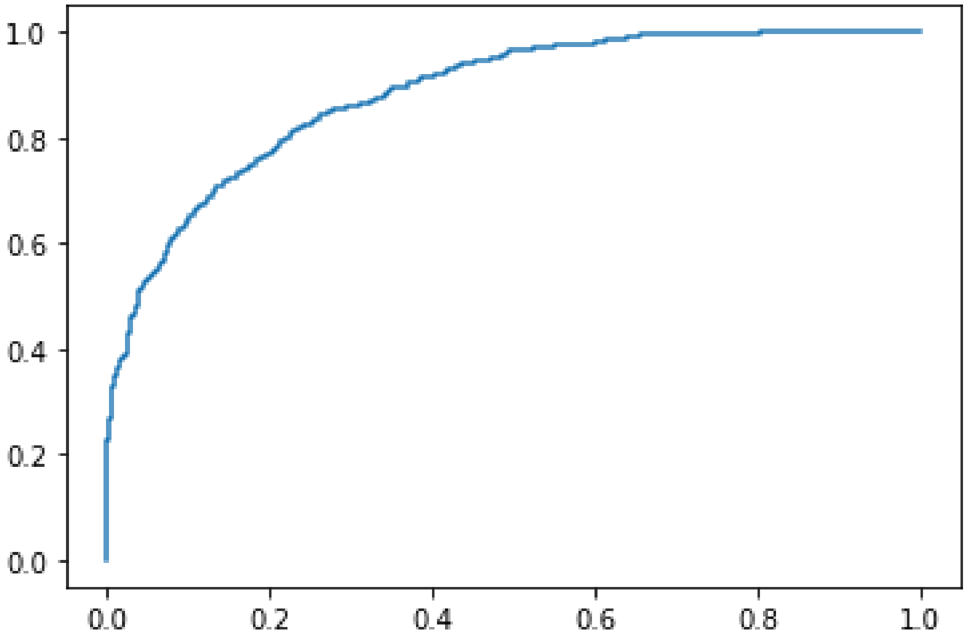
[308]:
```
roc_auc_score(y_test, lr.predict_proba(X_test)[:,1]) #with train, test split␣
 ↪method, auc score is 0.88
```

[308]:
```
0.8827246251752777
```

[309]:
```
fp_lr, tp_lr,_ = roc_curve(y_test, lr.predict_proba(X_test)[:,1],pos_label = 1)
```

[310]:
```
plt.plot(fp_lr, tp_lr)
```

[310]:
```
[<matplotlib.lines.Line2D at 0x7fb6d242dbb0>]
```



2 Model 2 Random Forest Classifier

```
[275]: rfc = RandomForestClassifier()
       rfc.fit(X_train, y_train)
       y_predict_rf = rfc.predict(X_test)
```

```
[276]: roc_auc_score(y_test, rfc.predict_proba(X_test)[:,1])
```

```
[276]: 0.9790211411929673
```

```
[278]: precision_score(y_test, y_predict_rf, average ='weighted') # using weighted␣
        ↪because we have an imbalanced target data
```

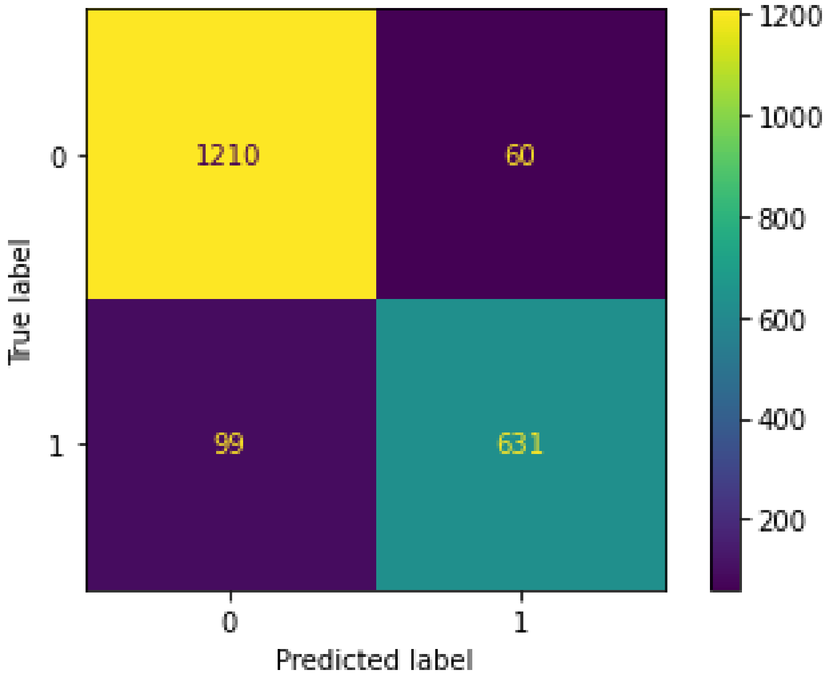
```
[278]: 0.92028159165258
```

```
[279]: recall_score(y_test, y_predict_rf, average = "weighted")
```

```
[279]: 0.9205
```

```
[281]: plot_confusion_matrix(rfc, X_test, y_test)
```

```
[281]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
       0x7fb6d263cd90>
```



```
[280]: fp_rf, tp_rf,_ = roc_curve(y_test, rfc.predict_proba(X_test)[:,1])
```

3 Model 3 KNN Classifier

```
[334]: from sklearn.neighbors import KNeighborsClassifier
       from sklearn import metrics
```

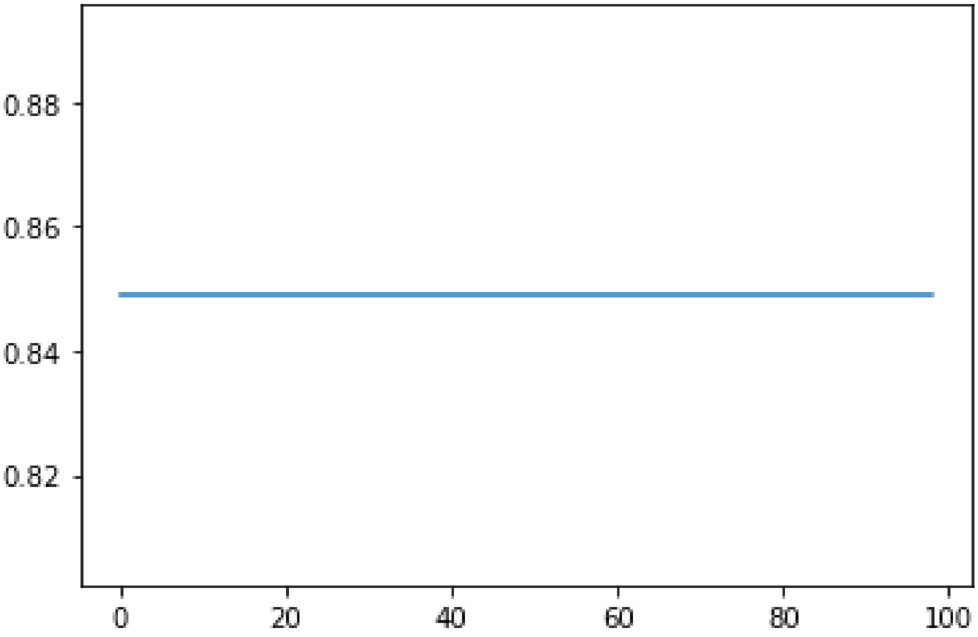
```
[329]: knn = KNeighborsClassifier()
```

```
[335]: knn = KNeighborsClassifier(n_neighbors = 3)
       knn.fit(X_train, y_train)
       yhat = knn.predict(X_test)
       metrics.accuracy_score(y_test, yhat)
```

```
[335]: 0.849
```

```
[336]: acc_score =[]
       for k in range(1,100):
           knn = KNeighborsClassifier(n_neighbors = 3)
           knn.fit(X_train, y_train)
           yhat = knn.predict(X_test)
           acc_score.append(metrics.accuracy_score(y_test, yhat))
```

[347]:
```
plt.plot(acc_score); # n value does not matter!
```



[345]:
```
roc_auc_score(y_test, knn.predict_proba(X_test)[:,1])
```

[345]:
```
0.9017614065365117
```

[348]:
```
precision_score(y_test, yhat, average ='weighted') # using weighted because we␣
 ↪have an imbalanced target data
```

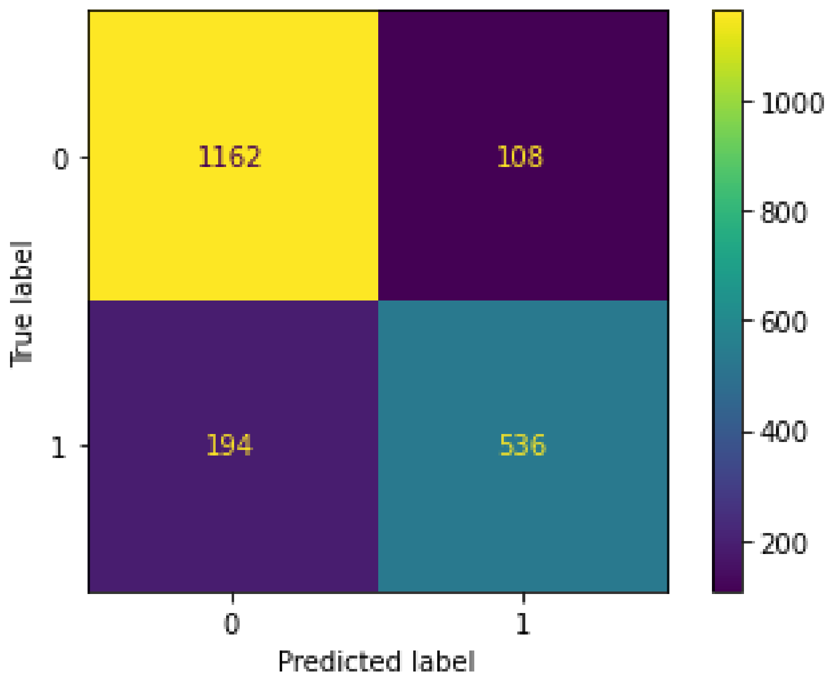
[348]:
```
0.8479407372799063
```

[351]:
```
recall_score(y_test, yhat, average ='weighted') # using weighted because we␣
 ↪have an imbalanced target data
```

[351]:
```
0.849
```

[353]:
```
plot_confusion_matrix(knn, X_test, y_test)
```

[353]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fb6d23df070>
```



[354]:
```
fp_knn, tp_knn,_ = roc_curve(y_test, knn.predict_proba(X_test)[:,1])
```

4 Model 4 Decision Tree Classifier

[358]:
```
DT = DecisionTreeClassifier()
```

[362]:
```
parameters = [{"criterion":["entropy","min_weight_fraction_leaf","poisson"],␣
 ↪"max_depth":list(np.arange(10))}]
grid4 =GridSearchCV(DT,parameters , scoring = 'roc_auc', cv=4)
```

[ ]:
```
grid4.fit(X,y)
```

[365]:
```
grid4.best_params_
```

[365]:
```
{'criterion': 'entropy', 'max_depth': 7}
```

[366]:
```
DT = DecisionTreeClassifier(criterion ="entropy", max_depth =7)
```

[367]:
```
DT.fit(X_train, y_train)
```

[367]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=7)
```

[368]:
```
y_predict_dt = DT.predict(X_test)
```

```
[369]: roc_auc_score(y_test, DT.predict_proba(X_test)[:,1])
```

```
[369]: 0.8937477079063747
```

```
[370]: precision_score(y_test, y_predict_dt, average ='weighted') # using weighted␣
        ↪because we have an imbalanced target data
```

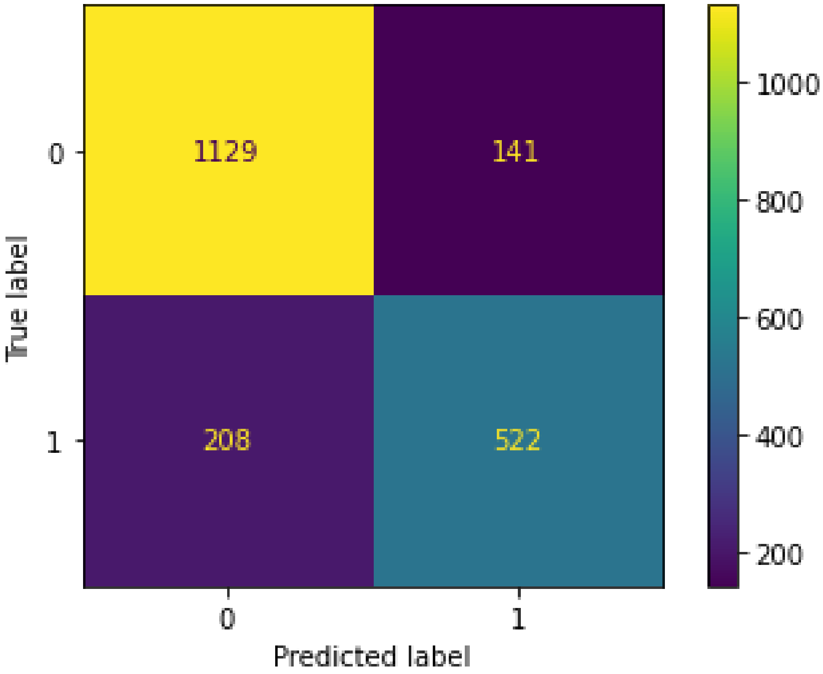
```
[370]: 0.8235872335240983
```

```
[371]: recall_score(y_test, y_predict_dt, average ='weighted') # using weighted␣
        ↪because we have an imbalanced target data
```

```
[371]: 0.8255
```

```
[372]: plot_confusion_matrix(DT, X_test, y_test)
```

```
[372]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
       0x7fb6d2291a00>
```



```
[373]: fp_DT, tp_DT,_ = roc_curve(y_test, DT.predict_proba(X_test)[:,1])
```

5 Model 5 Gradient Boosting Classifier

[375]:
```
gb = GradientBoostingClassifier(n_estimators=100, max_depth=1)
params = [{'n_estimators':[50, 75,100,125,150], "max_depth":[1,2,3,4,5],␣
 ↪"max_features": ['auto', 'sqrt', 'log2']}]
```

[376]:
```
grid4 =GridSearchCV(gb,params , scoring = 'roc_auc', cv=4)
```

[377]:
```
grid4.fit(X,y)
```

[377]:
```
GridSearchCV(cv=4, estimator=GradientBoostingClassifier(max_depth=1),
             param_grid=[{'max_depth': [1, 2, 3, 4, 5],
                          'max_features': ['auto', 'sqrt', 'log2'],
                          'n_estimators': [50, 75, 100, 125, 150]}],
             scoring='roc_auc')
```

[378]:
```
grid4.best_params_
```

[378]:
```
{'max_depth': 5, 'max_features': 'auto', 'n_estimators': 150}
```

[383]:
```
grid4.best_score_ # The highest ROC_AUC score yet!
```

[383]:
```
0.9866899322814735
```

[379]:
```
gb = GradientBoostingClassifier(n_estimators=150, max_depth=5, max_features␣
 ↪="auto")
```

[380]:
```
gb.fit(X_train, y_train)
```

[380]:
```
GradientBoostingClassifier(max_depth=5, max_features='auto', n_estimators=150)
```

[381]:
```
y_predict_gb = gb.predict(X_test)
```

[384]:
```
precision_score(y_test, y_predict_gb , average ='weighted') # using weighted␣
 ↪because we have an imbalanced target data
```

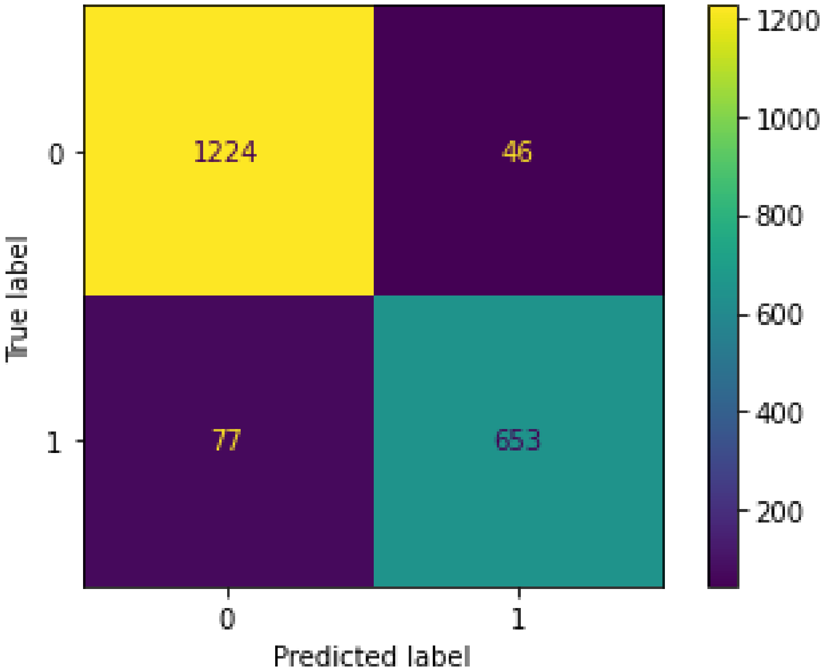
[384]:
```
0.9383973426405792
```

[385]:
```
recall_score(y_test, y_predict_gb , average ='weighted') # using weighted␣
 ↪because we have an imbalanced target data
```

[385]:
```
0.9385
```

[386]:
```
plot_confusion_matrix(gb, X_test, y_test)
```

[386]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fb6d2351ac0>
```

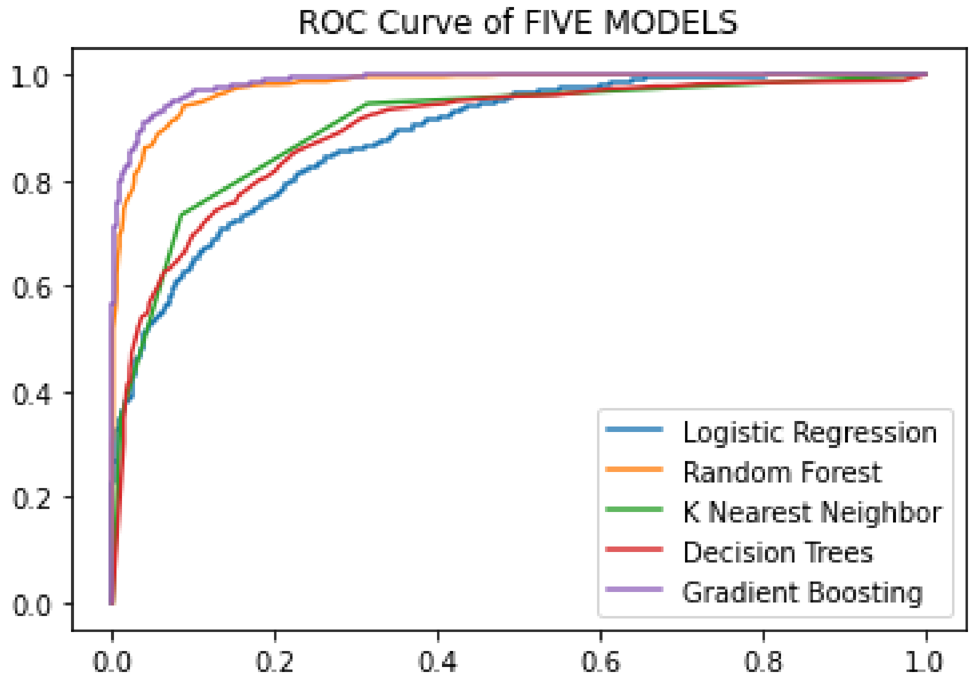


[388]:

```
fp_gb, tp_gb,_ = roc_curve(y_test, gb.predict_proba(X_test)[:,1])
```

[393]:

```
plt.plot(fp_lr, tp_lr, label ="Logistic Regression")
plt.plot(fp_rf, tp_rf, label ="Random Forest")
plt.plot(fp_knn, tp_knn, label ="K Nearest Neighbor")
plt.plot(fp_DT, tp_DT, label ="Decision Trees")
plt.plot(fp_gb, tp_gb, label ="Gradient Boosting")
plt.legend()
plt.title("ROC Curve of FIVE MODELS");
```



6 Gradient Boosting outperforms all others!

```
[ ]:
```