Decision-Tree-CLassifier-CHD

March 8, 2020

[46]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
%matplotlib inline
import plusmodules as pm
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
```

[47]:
```
df=pd.read_csv('US_Heart_Patients.csv')
```

[48]:
```
df=df.sample(frac=1, random_state=3)
```

[49]:
```
df.head()
```

[49]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	\
3546	0	54	1.0	0	0.0	0.0	
1127	0	42	3.0	1	10.0	0.0	
3088	0	58	1.0	0	0.0	1.0	
437	1	45	1.0	1	30.0	0.0	
3188	1	63	1.0	0	0.0	0.0	

	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	\
3546	0	0	0	241.0	106.0	77.0	27.64	
1127	0	0	0	253.0	109.0	74.0	24.38	
3088	1	1	0	274.0	159.0	90.0	28.40	
437	0	0	0	240.0	141.0	89.0	25.01	
3188	0	1	0	190.0	148.0	90.0	27.13	

	heartRate	glucose	TenYearCHD
3546	78.0	74.0	0
1127	88.0	60.0	0
3088	72.0	81.0	0
437	95.0	76.0	0

```
3188       72.0     86.0           0
```

[50]:
```
df
```

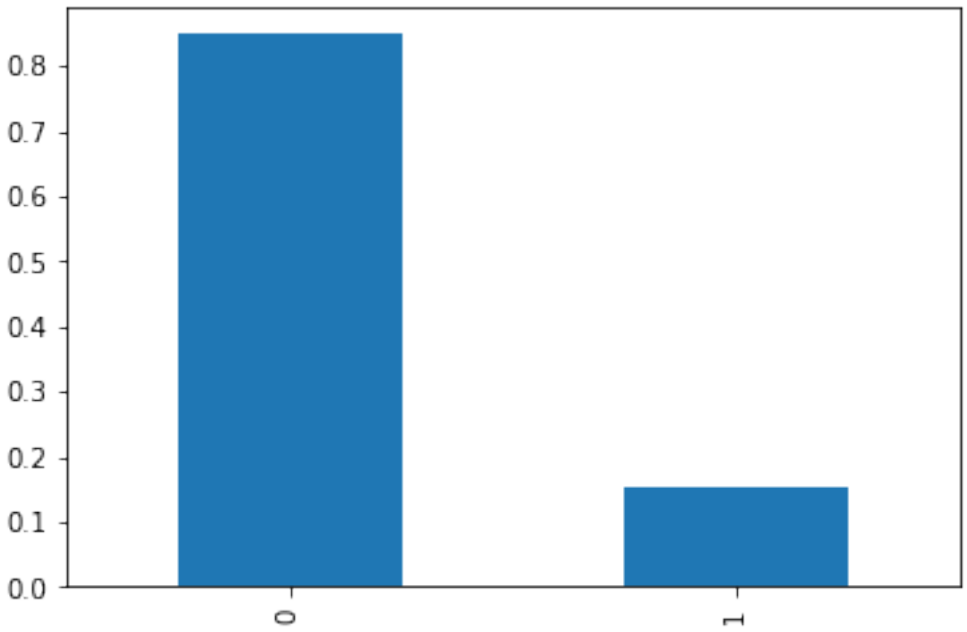
[50]:
```
      male  age  education  currentSmoker  cigsPerDay  BPMeds  \
3546     0   54        1.0              0         0.0     0.0
1127     0   42        3.0              1        10.0     0.0
3088     0   58        1.0              0         0.0     1.0
437      1   45        1.0              1        30.0     0.0
3188     1   63        1.0              0         0.0     0.0
…     …  …         …              …           …      …
789      1   63        1.0              0         0.0     0.0
968      0   54        1.0              0         0.0     0.0
1667     0   56        1.0              1         3.0     0.0
3321     0   58        2.0              0         0.0     0.0
1688     0   40        4.0              1        15.0     0.0

      prevalentStroke  prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  \
3546                0             0         0    241.0  106.0   77.0  27.64
1127                0             0         0    253.0  109.0   74.0  24.38
3088                1             1         0    274.0  159.0   90.0  28.40
437                 0             0         0    240.0  141.0   89.0  25.01
3188                0             1         0    190.0  148.0   90.0  27.13
…                 …             …       …       …    …     …     …
789                 0             1         0    260.0  159.5   91.0  27.01
968                 0             1         0    266.0  137.0   88.0  29.76
1667                0             1         0    285.0  145.0  100.0  30.14
3321                0             1         1    265.0  143.5   85.0  21.68
1688                0             0         0    155.0  121.0   86.0  23.16

      heartRate  glucose  TenYearCHD
3546       78.0     74.0           0
1127       88.0     60.0           0
3088       72.0     81.0           0
437        95.0     76.0           0
3188       72.0     86.0           0
…         …      …         …
789        68.0     66.0           0
968        80.0     80.0           0
1667       80.0     86.0           0
3321       91.0    107.0           0
1688       70.0     59.0           0

[4240 rows x 16 columns]
```

[51]:
```
df['TenYearCHD'].value_counts()
```

[51]:
```
0    3596
1     644
Name: TenYearCHD, dtype: int64
```

[52]:
```
df['TenYearCHD'].value_counts(normalize=True).plot.bar()
```

[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19fe778a128>
```



[53]:
```
mv=df.isnull().sum()
mv[mv>0]
```

[53]:
```
education     105
cigsPerDay     29
BPMeds         53
totChol        50
BMI            19
heartRate       1
glucose       388
dtype: int64
```

[54]:
```
df=df.fillna(method='ffill')
df.head()
# filling all the null values using forward filing method in order not to␣
 ↪change the distribution
```

[54]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	\
3546	0	54	1.0	0	0.0	0.0	
1127	0	42	3.0	1	10.0	0.0	
3088	0	58	1.0	0	0.0	1.0	
437	1	45	1.0	1	30.0	0.0	
3188	1	63	1.0	0	0.0	0.0	

	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	\
3546	0	0	0	241.0	106.0	77.0	27.64	
1127	0	0	0	253.0	109.0	74.0	24.38	
3088	1	1	0	274.0	159.0	90.0	28.40	
437	0	0	0	240.0	141.0	89.0	25.01	
3188	0	1	0	190.0	148.0	90.0	27.13	

	heartRate	glucose	TenYearCHD
3546	78.0	74.0	0
1127	88.0	60.0	0
3088	72.0	81.0	0
437	95.0	76.0	0
3188	72.0	86.0	0

[17]:
```
x=df.drop('TenYearCHD', axis=1)
y=df['TenYearCHD']
```

[18]:
```
from sklearn.model_selection import train_test_split
```

[19]:
```
x_train, x_test, y_train, y_test =train_test_split(x, y, test_size=0.3,␣
↪random_state=3)
```

[20]:
```
print(x_train.shape, x_test.shape)
```

```
(2968, 15) (1272, 15)
```

[29]:
```
from sklearn.metrics import confusion_matrix, roc_auc_score, accuracy_score,␣
↪roc_curve
```

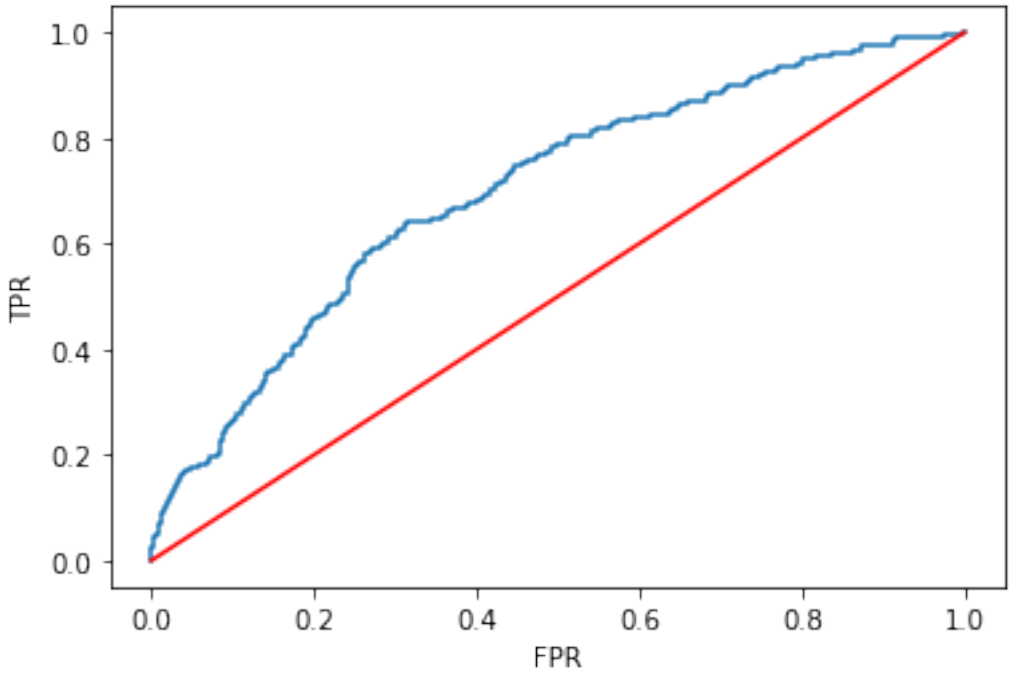
[43]:
```
from sklearn.linear_model import LogisticRegression

lr=LogisticRegression(fit_intercept= True, solver='liblinear')

lr.fit(x_train, y_train)

y_train_pred=lr.predict(x_train)

y_train_prob=lr.predict_proba(x_train)[:,1]
```

```
print('Confusion Matrix - Train: ', '\n' ,confusion_matrix(y_train,␣
 ↪y_train_pred))

print('Overall Accuracy - Train: ', accuracy_score(y_train, y_train_pred))

print('AUC- Train:' , roc_auc_score(y_train, y_train_prob))



y_test_pred= lr.predict(x_test)

y_test_prob=lr.predict_proba(x_test)[:,1]

print('Confusion Matrix - Test: ', '\n' ,confusion_matrix(y_test, y_test_pred))

print('Overall Accuracy - Test: ', accuracy_score(y_test, y_test_pred))

print('AUC- Test:' , roc_auc_score(y_test, y_test_prob))
```

```
Confusion Matrix - Train:
 [[2520   13]
 [ 400   35]]
Overall Accuracy - Train:  0.8608490566037735
AUC- Train: 0.7313212718551896
Confusion Matrix - Test:
 [[1060    3]
 [ 200    9]]
Overall Accuracy - Test:  0.8404088050314465
AUC- Test: 0.699491823718194
```

[44]:
```
fpr, tpr, thresholds =roc_curve(y_test, y_test_prob)
plt.plot(fpr, tpr)
plt.plot(fpr, fpr, 'red')
plt.xlabel('FPR')
plt.ylabel('TPR')
```

[44]:
```
Text(0, 0.5, 'TPR')
```



we can see that the auc score is 0.6 which says that the model is not good as the curve is not as exepected

[27]:
```
y_pred
# the probabilities are calculated
```

[27]:
```
array([0, 0, 0, …, 0, 1, 0], dtype=int64)
```

[28]:
```
y_train_prob
# here the two columns are the probabilities of 0 and 1
```

[28]:
```
array([[0.90618998, 0.09381002],
       [0.94311011, 0.05688989],
       [0.86481504, 0.13518496],
       …,
       [0.73037875, 0.26962125],
       [0.25581629, 0.74418371],
       [0.95469585, 0.04530415]])
```

[ ]:

1 decision tree classifier

[45]:
```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydotplus
import imblearn
import lightgbm
import hyperopt
```

```
Using TensorFlow backend.
Failed to load cloudpickle, try installing cloudpickle via "pip install
cloudpickle" for enhanced pickling support.
```

[55]:
```
df.head()
```

[55]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	\
3546	0	54	1.0	0	0.0	0.0	
1127	0	42	3.0	1	10.0	0.0	
3088	0	58	1.0	0	0.0	1.0	
437	1	45	1.0	1	30.0	0.0	
3188	1	63	1.0	0	0.0	0.0	

	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	\
3546	0	0	0	241.0	106.0	77.0	27.64	
1127	0	0	0	253.0	109.0	74.0	24.38	
3088	1	1	0	274.0	159.0	90.0	28.40	
437	0	0	0	240.0	141.0	89.0	25.01	
3188	0	1	0	190.0	148.0	90.0	27.13	

	heartRate	glucose	TenYearCHD
3546	78.0	74.0	0
1127	88.0	60.0	0
3088	72.0	81.0	0
437	95.0	76.0	0
3188	72.0	86.0	0

[67]:
```
y=df['TenYearCHD']
x=df.drop('TenYearCHD', axis=1)
```

[62]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=4)
```

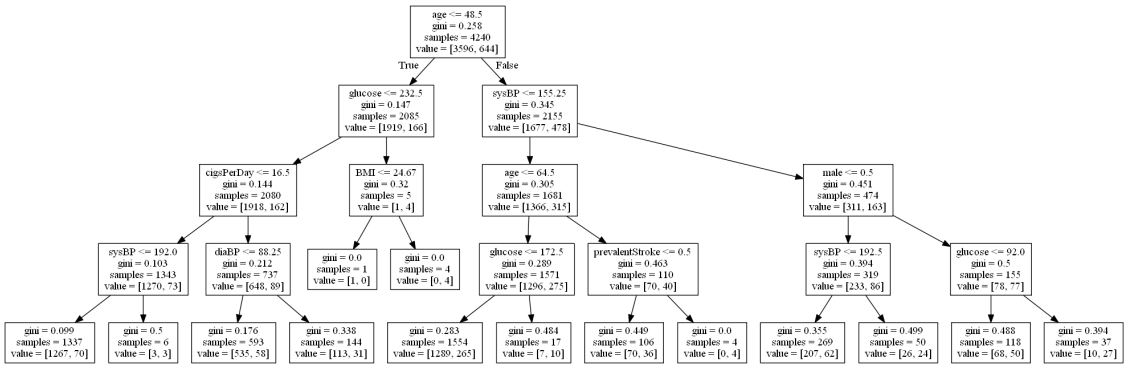
[63]:
```
dt.fit(x,y)
```

[63]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
```

```
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

[64]:
```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydotplus

features = X.columns
# Create DOT data
dot_data = export_graphviz(dt, out_file=None, feature_names=features)

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png())
```

[64]:



the root node here is age and the gini index for root node is 0.258, the value shows the number of zeros and ones 3596 zeros and 644 ones, this tree says that age is the best feature as we can see the first split is based on age

[75]:
```
y=df['TenYearCHD']
x=df.drop('TenYearCHD', axis=1)
```

[76]:
```
x_train, x_test, y_train, y_test =train_test_split(x, y, test_size=0.3,␣
↪random_state=3)
```

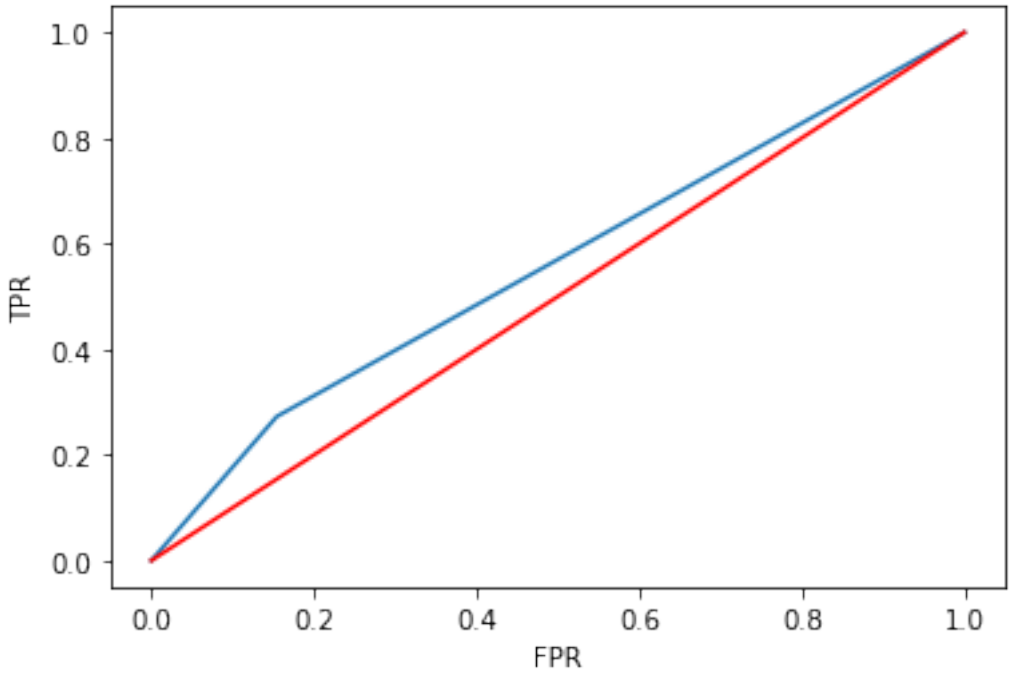
[78]:
```
from sklearn.metrics import confusion_matrix, roc_auc_score, accuracy_score,␣
↪roc_curve
```

[80]:
```
dt=DecisionTreeClassifier()

dt.fit(x_train, y_train)

y_train_pred=dt.predict(x_train)

y_train_prob=dt.predict_proba(x_train)[:,1]

print('Confusion Matrix - Train: ', '\n' ,confusion_matrix(y_train,␣
↪y_train_pred))

print('Overall Accuracy - Train: ', accuracy_score(y_train, y_train_pred))

print('AUC- Train:' , roc_auc_score(y_train, y_train_prob))



y_test_pred= dt.predict(x_test)

y_test_prob=dt.predict_proba(x_test)[:,1]

print('Confusion Matrix - Test: ', '\n' ,confusion_matrix(y_test, y_test_pred))

print('Overall Accuracy - Test: ', accuracy_score(y_test, y_test_pred))

print('AUC- Test:' , roc_auc_score(y_test, y_test_prob))
```

```
Confusion Matrix - Train:
 [[2533    0]
 [   0  435]]
Overall Accuracy - Train:  1.0
AUC- Train: 1.0
Confusion Matrix - Test:
 [[899 164]
 [152  57]]
Overall Accuracy - Test:  0.7515723270440252
AUC- Test: 0.5592234670315573
```

[84]:
```
fpr, tpr, thresholds =roc_curve(y_test, y_test_prob)
plt.plot(fpr, tpr)
plt.plot(fpr, fpr, 'red')
plt.xlabel('FPR')
plt.ylabel('TPR')
```

[84]:
```
Text(0, 0.5, 'TPR')
```



[ ]:

[ ]:

2 hyper parameter tuning using grid search

[83]:
```
import sklearn
sklearn.metrics.SCORERS.keys()
```

[83]:
```
dict_keys(['explained_variance', 'r2', 'max_error', 'neg_median_absolute_error',
'neg_mean_absolute_error', 'neg_mean_squared_error',
'neg_mean_squared_log_error', 'neg_root_mean_squared_error',
'neg_mean_poisson_deviance', 'neg_mean_gamma_deviance', 'accuracy', 'roc_auc',
'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted', 'roc_auc_ovo_weighted',
'balanced_accuracy', 'average_precision', 'neg_log_loss', 'neg_brier_score',
'adjusted_rand_score', 'homogeneity_score', 'completeness_score',
'v_measure_score', 'mutual_info_score', 'adjusted_mutual_info_score',
'normalized_mutual_info_score', 'fowlkes_mallows_score', 'precision',
'precision_macro', 'precision_micro', 'precision_samples', 'precision_weighted',
'recall', 'recall_macro', 'recall_micro', 'recall_samples', 'recall_weighted',
'f1', 'f1_macro', 'f1_micro', 'f1_samples', 'f1_weighted', 'jaccard',
'jaccard_macro', 'jaccard_micro', 'jaccard_samples', 'jaccard_weighted'])
```

[86]:
```
from sklearn.model_selection import GridSearchCV

dtc=DecisionTreeClassifier()

params={'max_depth':[2,3,4,5,6],
       'min_samples_leaf':[1,2,3,4,5,6,7],
       'min_samples_split':[2,3,4,5,6,7,8,9,10],
       'criterion':['gini', 'entropy']}

gs=GridSearchCV(dtc, param_grid=params, cv=3, scoring='roc_auc')
gs.fit(x,y)
```

[86]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 3, 4, 5, 6],
                         'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7],
                         'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

[88]:
```
gs.best_params_
```

[88]:
```
{'criterion': 'gini',
 'max_depth': 4,
 'min_samples_leaf': 3,
 'min_samples_split': 8}
```

[90]:
```
gsr=pd.DataFrame(gs.cv_results_)
gsr.head(3)
```

[90]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
0	0.005945	0.000776	0.002022	0.000815	
1	0.007183	0.001326	0.006093	0.001452	
2	0.009139	0.001510	0.004038	0.000057	

```
  param_criterion param_max_depth param_min_samples_leaf  \
0            gini               2                      1
1            gini               2                      1
2            gini               2                      1

  param_min_samples_split                                              params  \
0                       2  {'criterion': 'gini', 'max_depth': 2, 'min_sam…
1                       3  {'criterion': 'gini', 'max_depth': 2, 'min_sam…
2                       4  {'criterion': 'gini', 'max_depth': 2, 'min_sam…

   split0_test_score  split1_test_score  split2_test_score  mean_test_score  \
0           0.668305            0.65094           0.670307         0.663184
1           0.668305            0.65094           0.670307         0.663184
2           0.668305            0.65094           0.670307         0.663184

   std_test_score  rank_test_score
0        0.008696              415
1        0.008696              415
2        0.008696              415
```

[108]:
```
dt = DecisionTreeClassifier(gs.best_params_)
```

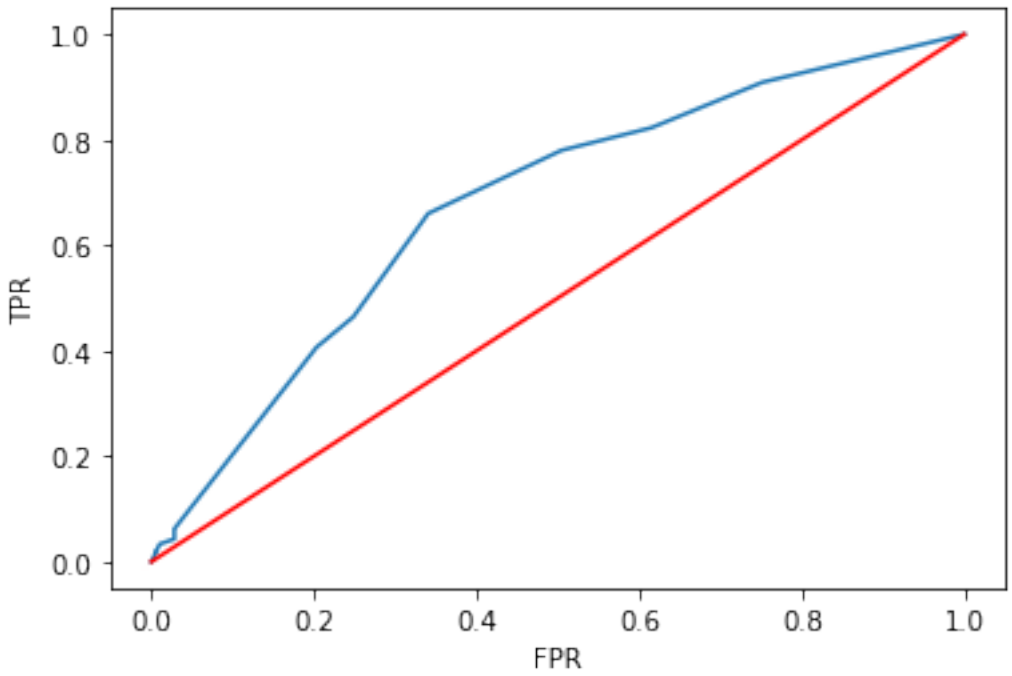
[109]:
```
dt=DecisionTreeClassifier(**gs.best_params_)

dt.fit(x_train, y_train)

y_train_pred=dt.predict(x_train)

y_train_prob=dt.predict_proba(x_train)[:,1]

print('Confusion Matrix - Train: ', '\n' ,confusion_matrix(y_train,␣
↪y_train_pred))

print('Overall Accuracy - Train: ', accuracy_score(y_train, y_train_pred))

print('AUC- Train:' , roc_auc_score(y_train, y_train_prob))



y_test_pred= dt.predict(x_test)

y_test_prob=dt.predict_proba(x_test)[:,1]

print('Confusion Matrix - Test: ', '\n' ,confusion_matrix(y_test, y_test_pred))

print('Overall Accuracy - Test: ', accuracy_score(y_test, y_test_pred))
```

```
print('AUC- Test:' , roc_auc_score(y_test, y_test_prob))
```

```
Confusion Matrix - Train:
 [[2520   13]
 [ 402   33]]
Overall Accuracy - Train:  0.8601752021563343
AUC- Train: 0.7284597338125252
Confusion Matrix - Test:
 [[1052   11]
 [ 202    7]]
Overall Accuracy - Test:  0.8325471698113207
AUC- Test: 0.6754063384751111
```

[110]:
```
fpr, tpr, thresholds =roc_curve(y_test, y_test_prob)
plt.plot(fpr, tpr)
plt.plot(fpr, fpr, 'red')
plt.xlabel('FPR')
plt.ylabel('TPR')
```

[110]: Text(0, 0.5, 'TPR')



[ ]:

[ ]:

#hyperparameter tuning using randomized search

[111]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
dtc= DecisionTreeClassifier()

params = {'max_depth': sp_randint(2,20),
          'min_samples_leaf': sp_randint(1, 20),
          'min_samples_split': sp_randint(2, 40),
          'criterion':['gini', 'entropy']}

rsearch=RandomizedSearchCV(dtc, param_distributions=params, cv=3,␣
 ↪scoring='roc_auc', n_iter=200)
rsearch.fit(x,y)
```

[111]:
```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features=None,
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
min_weight_fraction_leaf=0.0,
                                                    presort='deprecated',
                                                    random_state=None,
                                                    splitter='best'),
                   i…
                                         'max_depth':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x0000019FEFB3EC88>,
                                         'min_samples_leaf':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x0000019FEFB91320>,
                                         'min_samples_split':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x0000019FEFB91128>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='roc_auc', verbose=0)
```

[112]:
```
rsearch.best_params_
```

[112]:
```
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_leaf': 11,
 'min_samples_split': 5}
```

[117]:
```
rsr=pd.DataFrame(rsearch.cv_results_)
rsr.head(2)
```

[117]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_min_samples_leaf	param_min_samples_split	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score
0	0.035848	0.001561	0.004739	0.000520	entropy	14	7	8	{'criterion': 'entropy', 'max_depth': 14, 'min…	0.557459	0.580513	0.600881	0.579618	0.017738	196
1	0.016810	0.004090	0.003380	0.000788	gini	9	1	31	{'criterion': 'gini', 'max_depth': 9, 'min_sam…	0.637198	0.645082	0.633362	0.638547	0.004879	73

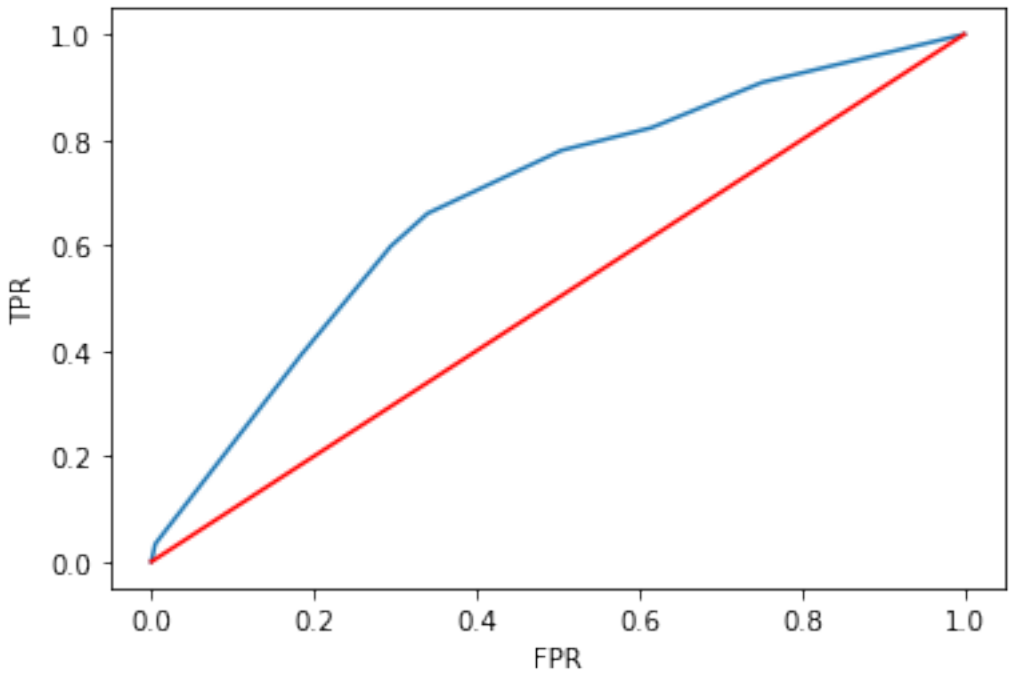
[118]:
```
dt=DecisionTreeClassifier(**rsearch.best_params_)

dt.fit(x_train, y_train)

y_train_pred=dt.predict(x_train)

y_train_prob=dt.predict_proba(x_train)[:,1]

print('Confusion Matrix - Train: ', '\n' ,confusion_matrix(y_train,␣
 ↪y_train_pred))

print('Overall Accuracy - Train: ', accuracy_score(y_train, y_train_pred))

print('AUC- Train:' , roc_auc_score(y_train, y_train_prob))



y_test_pred= dt.predict(x_test)

y_test_prob=dt.predict_proba(x_test)[:,1]

print('Confusion Matrix - Test: ', '\n' ,confusion_matrix(y_test, y_test_pred))

print('Overall Accuracy - Test: ', accuracy_score(y_test, y_test_pred))
```

```
print('AUC- Test:' , roc_auc_score(y_test, y_test_prob))
```

```
Confusion Matrix - Train:
 [[2524    9]
 [ 416   19]]
Overall Accuracy - Train:  0.8568059299191375
AUC- Train: 0.7151562592174107
Confusion Matrix - Test:
 [[1058    5]
 [ 202    7]]
Overall Accuracy - Test:  0.8372641509433962
AUC- Test: 0.683962964796752
```

[115]:
```
fpr, tpr, thresholds =roc_curve(y_test, y_test_prob)
plt.plot(fpr, tpr)
plt.plot(fpr, fpr, 'red')
plt.xlabel('FPR')
plt.ylabel('TPR')
```

[115]: Text(0, 0.5, 'TPR')



[ ]:

[ ]:

[ ]:

[ ]:

[ ]: