ROC_Using_Predict_Proba

December 11, 2017

```
In [41]: %matplotlib inline
```

```
In [42]: from pathlib import Path
         from pandas import DataFrame,Series
         from pandas.plotting import scatter_matrix
         from sklearn.model_selection import train_test_split
         from sklearn import tree
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import accuracy_score
         import pandas as pd
         from matplotlib.colors import ListedColormap
         import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix
         import numpy as np
         import scipy.stats as stats
         import pylab as pl
         from random import sample
```

```
In [43]: #Description of features
         #Average[3]: Average acceleration (for each axis)
         #Standard Deviation[3]: Standard deviation (for each axis)
         #Average Absolute Difference[3]: Average absolute
         #difference between the value of each of the 200 readings
         #within the ED and the mean value over those 200 values
         #(for each axis)
         #Average Resultant Acceleration[1]: Average of the square
         #roots of the sum of the values of each axis squared
         #over the ED
         #Time Between Peaks[3]: Time in milliseconds between
         #peaks in the sinusoidal waves associated with most
         #activities (for each axis)
         #Binned Distribution[30]: We determine the range of values
         #for each axis (maximum  minimum), divide this range into
         #10 equal sized bins, and then record what fraction of the
         #200 values fell within each of the bins.
```

```
In [44]: my_file = Path("/Users/bharu/CS690-PROJECTS/ActivityAnalyzer/activity_analyzer/Decisio
         df = pd.read_csv(my_file)
         df.head()
         df.shape#(no of rows, no of columns)

Out[44]: (417, 46)

In [45]: df['color'] = Series([(0 if x == "walking" else 1) for x in df['Label']])
         my_color_map = ListedColormap(['skyblue','coral'],'mycolormap')
         #0,red,walking
         #1,green,running

         df_unique = df.drop_duplicates(subset=['User', 'Timestamp'])
         df_unique.head()
         df_unique.shape

Out[45]: (406, 47)

In [46]: X_train = df_unique.values[:,2:45]
         Y_train = df_unique.values[:,45]

In [47]: test_file = Path("/Users/bharu/CS690-PROJECTS/ActivityAnalyzer/activity_analyzer/Decis
         df_test = pd.read_csv(test_file)
         df_test.head()
         df_test.shape#(no of rows, no of columns)

Out[47]: (518, 46)

In [48]: df_test['color'] = Series([(0 if x == "walking" else 1) for x in df_test['Label']])

In [49]: df_unique_test = df_test.drop_duplicates(subset=['User', 'Timestamp'])
         df_unique_test.head()
         df_unique_test.shape

Out[49]: (415, 47)

In [50]: #Predicting using test data
         #taking size of test data 10% of training data
         test_small = df_unique_test.iloc[sample(range(len(df_unique_test)), 40), :]
         X_test_small = test_small.values[:,2:45]
         Y_test_small = test_small.values[:,45]

In [56]: df_gini = DecisionTreeClassifier(criterion = 'gini')

In [57]: df_gini.fit(X_train, Y_train)

Out[57]: DecisionTreeClassifier(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=False, random_state=None,
                     splitter='best')
```

```
In [58]: #Predicting using test data
         Y_predict_gini = df_gini.predict(X_test_small)
```

```
In [59]: #Calculating accuracy score
         score = accuracy_score(Y_test_small,Y_predict_gini)
         score
```

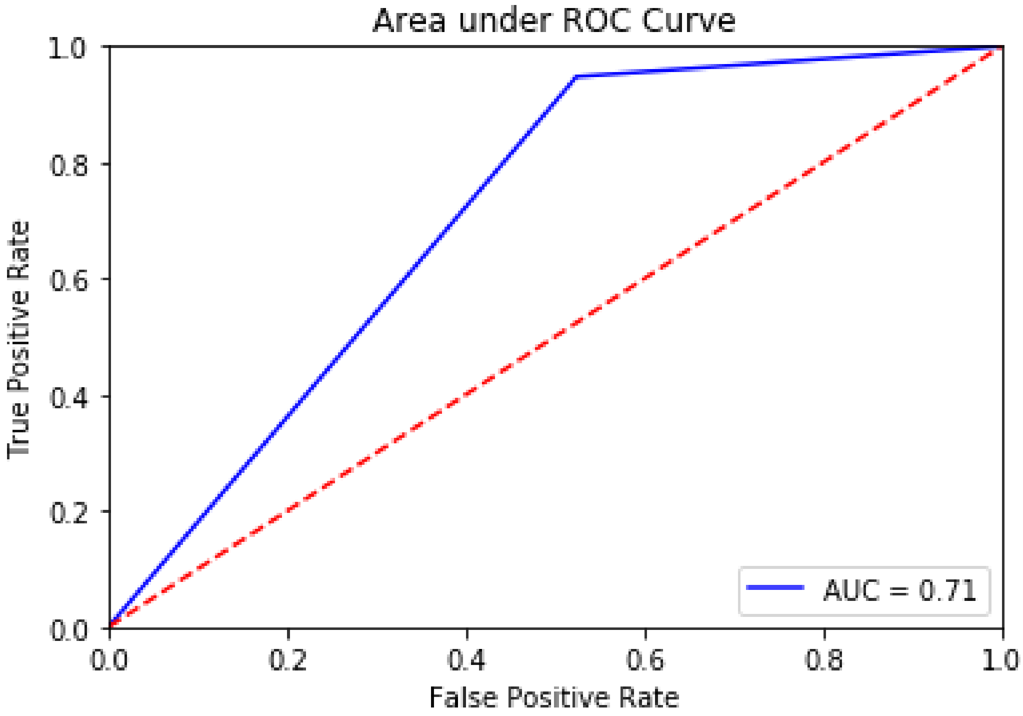
```
Out[59]: 0.69999999999999996
```

```
In [70]: #Predicting using test data
         Y_predict_gini_probas = df_gini.predict_proba(X_test_small)
         print (Y_predict_gini_probas[:,0])
         print (Y_predict_gini_probas[:,1])
         print(len(Y_predict_gini_probas))
```

```
[ 1.  0.  1.  0.  1.  1.  1.  0.  1.  1.  1.  0.  1.  1.  1.  0.  0.  1.
  1.  1.  0.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.  1.  0.
  1.  0.  1.  1.]
[ 0.  1.  0.  1.  0.  0.  0.  1.  0.  0.  0.  1.  0.  0.  0.  1.  1.  0.
  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  1.
  0.  1.  0.  0.]
40
```

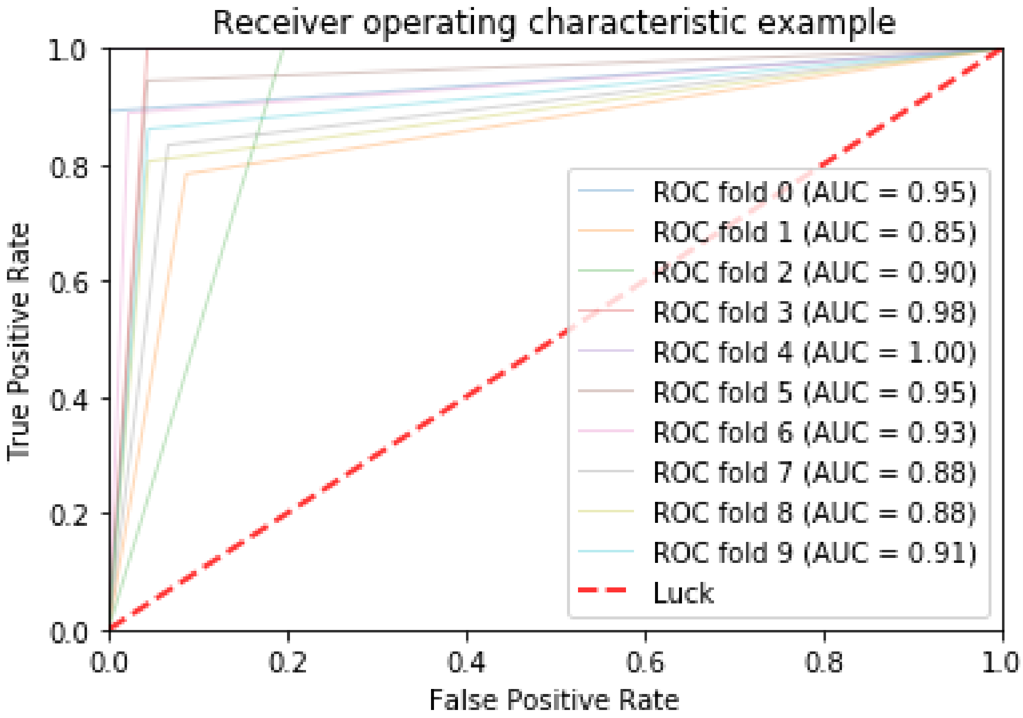
```
In [71]: import numpy as np
         from sklearn import metrics
         import matplotlib.pyplot as plt


         def plot_roc_curve(Y_predict_gini,Y_test,name_graph):
             num_labels = []
             for i in range(0,len(Y_test)):
                 if Y_test[i] == "walking":
                     num_labels.append(0)
                 else:
                     num_labels.append(1)

             labels = np.array(num_labels)
             fpr, tpr, thresholds = metrics.roc_curve(labels,Y_predict_gini)
             roc_auc = metrics.auc(fpr, tpr)
             plt.title('Area under ROC Curve')
             plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
             plt.legend(loc = 'lower right')
             plt.plot([0, 1], [0, 1],'r--')
             plt.xlim([0, 1])
             plt.ylim([0, 1])
             plt.ylabel('True Positive Rate')
             plt.xlabel('False Positive Rate')
             plt.savefig('./../Data-Visualization/images/' + name_graph +'.png',dpi=1000)
```

```
In [72]: plot_roc_curve(Y_predict_gini_probas[:,0],Y_test_small,"DecisionTree_ROC_using_predic
```



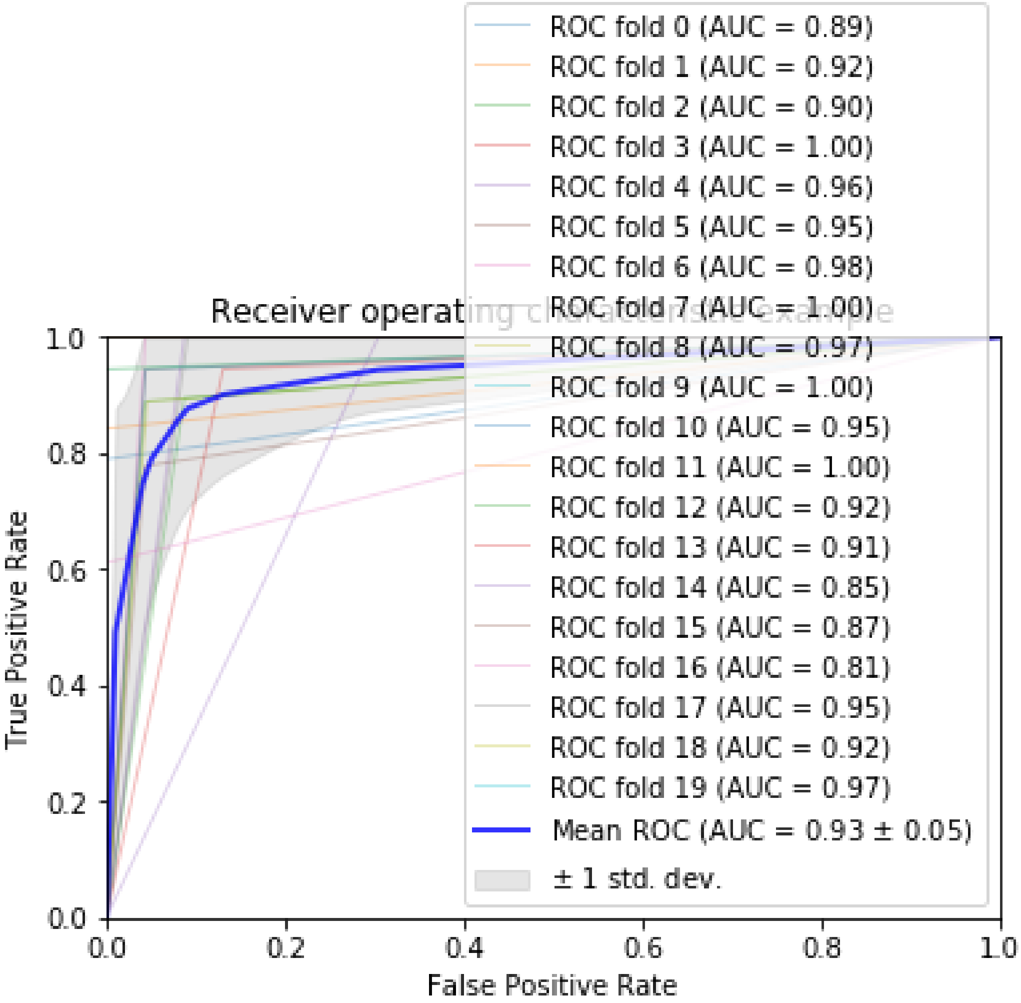
```
In [74]: df_3_10 = pd.concat([df_unique,df_unique_test])
         df_3_10.shape

Out[74]: (821, 47)

In [103]: X = df_3_10.values[:,2:45]
          y = df_3_10.values[:,45]

In [108]: X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size=0.5)

In [109]: df_gini.fit(X_train, Y_train)

Out[109]: DecisionTreeClassifier(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=False, random_state=None,
                      splitter='best')

In [110]: #Predicting using test data
          Y_predict_gini_3_10 = df_gini.predict(X_test)
```

```
In [111]: #Calculating accuracy score
          score = accuracy_score(Y_test,Y_predict_gini_3_10)
          score

Out[111]: 0.93187347931873477

In [122]: from sklearn.model_selection import StratifiedKFold

          cv = StratifiedKFold(n_splits=10)

          j = 0
          for train, test in cv.split(X, y):
              probas_ = df_gini.fit(X[train], y[train]).predict_proba(X[test])

              num_labels = []
              for i in range(0,len(y[test])):
                  if y[test][i] == "walking":
                      num_labels.append(0)
                  else:
                      num_labels.append(1)

              labels = np.array(num_labels)

              # Compute ROC curve and area the curve
              fpr, tpr, thresholds = metrics.roc_curve(labels, probas_[:, 0])
              roc_auc = metrics.auc(fpr, tpr)
              plt.plot(fpr, tpr, lw=1, alpha=0.3,
                       label='ROC fold %d (AUC = %0.2f)' % (j, roc_auc))
              j += 1

          plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',label='Luck', alpha=.8)
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.xlabel('False Positive Rate')
          plt.ylabel('True Positive Rate')
          plt.title('Receiver operating characteristic example')
          plt.legend(loc="lower right")
          plt.show()
```



```
In [121]: from sklearn.model_selection import StratifiedKFold

          cv = StratifiedKFold(n_splits=20)

          tprs = []
          aucs = []
          mean_fpr = np.linspace(0, 1, 100)

          j = 0
          for train, test in cv.split(X, y):
              probas_ = df_gini.fit(X[train], y[train]).predict_proba(X[test])

              num_labels = []
              for i in range(0,len(y[test])):
                  if y[test][i] == "walking":
                      num_labels.append(0)
                  else:
                      num_labels.append(1)

              labels = np.array(num_labels)

              # Compute ROC curve and area the curve
              fpr, tpr, thresholds = metrics.roc_curve(labels, probas_[:, 0])
              tprs.append(np.interp(mean_fpr, fpr, tpr))
```

```
    tprs[-1][0] = 0.0
    roc_auc = metrics.auc(fpr, tpr)
    aucs.append(roc_auc)
    plt.plot(fpr, tpr, lw=1, alpha=0.3,
             label='ROC fold %d (AUC = %0.2f)' % (j, roc_auc))
    j += 1



mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = metrics.auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
plt.plot(mean_fpr, mean_tpr, color='b',
         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
         lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                 label=r'$\pm$ 1 std. dev.')

plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```


Receiver operating characteristic example
ROC fold 0 (AUC = 0.89)
ROC fold 1 (AUC = 0.92)
ROC fold 2 (AUC = 0.90)
ROC fold 3 (AUC = 1.00)
ROC fold 4 (AUC = 0.96)
ROC fold 5 (AUC = 0.95)
ROC fold 6 (AUC = 0.98)
ROC fold 7 (AUC = 1.00)
ROC fold 8 (AUC = 0.97)
ROC fold 9 (AUC = 1.00)
ROC fold 10 (AUC = 0.95)
ROC fold 11 (AUC = 1.00)
ROC fold 12 (AUC = 0.92)
ROC fold 13 (AUC = 0.91)
ROC fold 14 (AUC = 0.85)
ROC fold 15 (AUC = 0.87)
ROC fold 16 (AUC = 0.81)
ROC fold 17 (AUC = 0.95)
ROC fold 18 (AUC = 0.92)
ROC fold 19 (AUC = 0.97)
Mean ROC (AUC = 0.93 ± 0.05)
± 1 std. dev.
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0