MultilayerPerceptron_3_test_3_train

December 17, 2017

In [4]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
import pandas as pd
from pandas import DataFrame,Series
from matplotlib.colors import ListedColormap
import numpy as np
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from random import sample
from sklearn.model_selection import train_test_split
```

In [2]:
```
df = pd.read_csv('../FeaturesCsvFile/featuresfile.csv')
df.shape
```

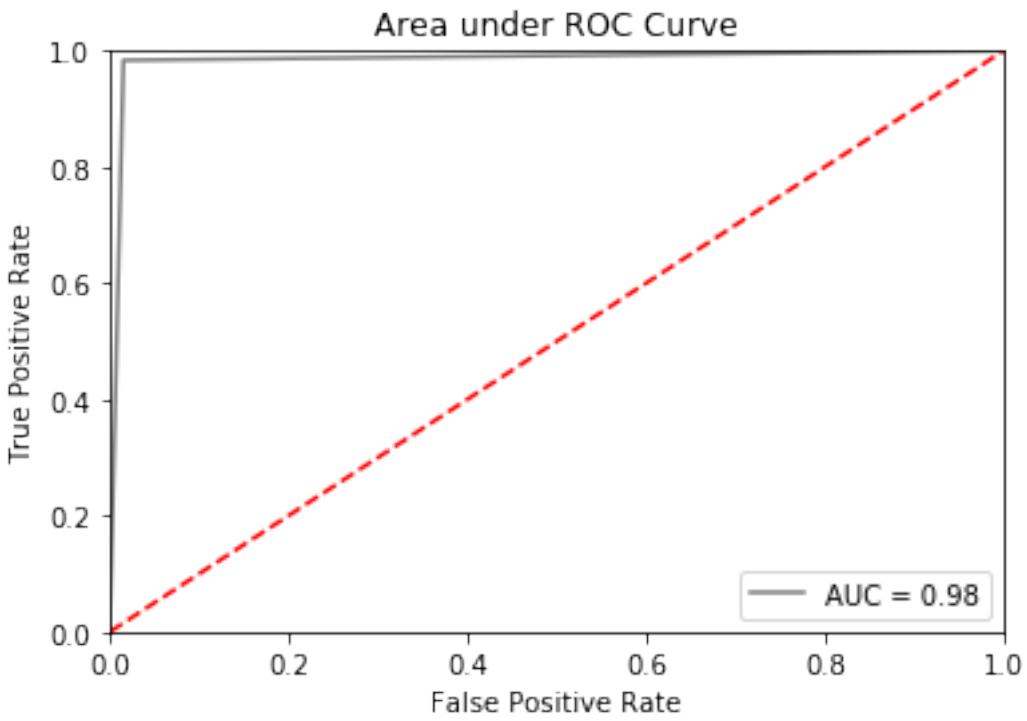
Out[2]:
```
(417, 46)
```

In [5]:
```
X = df.values[:, 2:45]
y = df.values[:, 45] #label : walking/runing
y_plot = np.where(y == 'walking', -1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y_plot, test_size=0.3)
scaler = StandardScaler()
scaler.fit(X_train)
StandardScaler(copy=True, with_mean=True, with_std=True)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
mlp = MLPClassifier(hidden_layer_sizes=(15,),max_iter=60)
mlp.fit(X_train,y_train)
y_pred = mlp.predict(X_test)
print('Accuracy of Accuracy Score : %.2f' % accuracy_score(y_test,y_pred))
print('Accuracy of Multi-Layer Perceptron Score: %.2f' % mlp.score(X_test,y_test))
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
/usr/local/lib/python2.7/site-packages/sklearn/utils/validation.py:475: DataConversionWarning:
  warnings.warn(msg, DataConversionWarning)
```

```
Accuracy of Accuracy Score : 0.98
Accuracy of Multi-Layer Perceptron Score: 0.98
[[64  1]
 [ 1 60]]
             precision    recall  f1-score   support

         -1       0.98      0.98      0.98        65
          1       0.98      0.98      0.98        61

avg / total       0.98      0.98      0.98       126



/usr/local/lib/python2.7/site-packages/sklearn/neural_network/multilayer_perceptron.py:564: Co
  % self.max_iter, ConvergenceWarning)
```

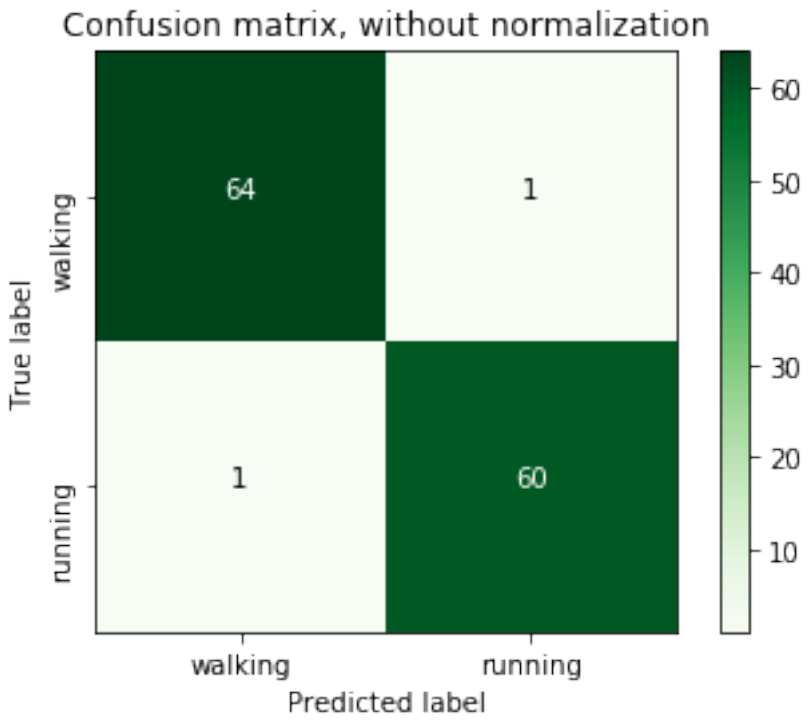
```
In [6]: from sklearn import metrics
        def plot_roc_curve(Y_predict,Y_test,name_graph):
            num_predns = []
            for i in range(0,len(Y_predict)):
                if Y_predict[i] == "walking":
                    num_predns.append(0)
                else:
                    num_predns.append(1)
            num_labels = []
            for i in range(0,len(Y_test)):
                if Y_test[i] == "walking":
                    num_labels.append(0)
                else:
                    num_labels.append(1)

            predns = np.array(num_predns)
            labels = np.array(num_labels)
            fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred)
            roc_auc = metrics.auc(fpr, tpr)
            plt.title('Area under ROC Curve')
            plt.plot(fpr, tpr, 'grey', 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.show()
        #     plt.savefig('./image/Area_under_roc_pc.png', dpi=1000)

        plot_roc_curve(y_pred,y_test,"Area_under_roc_pc")
```

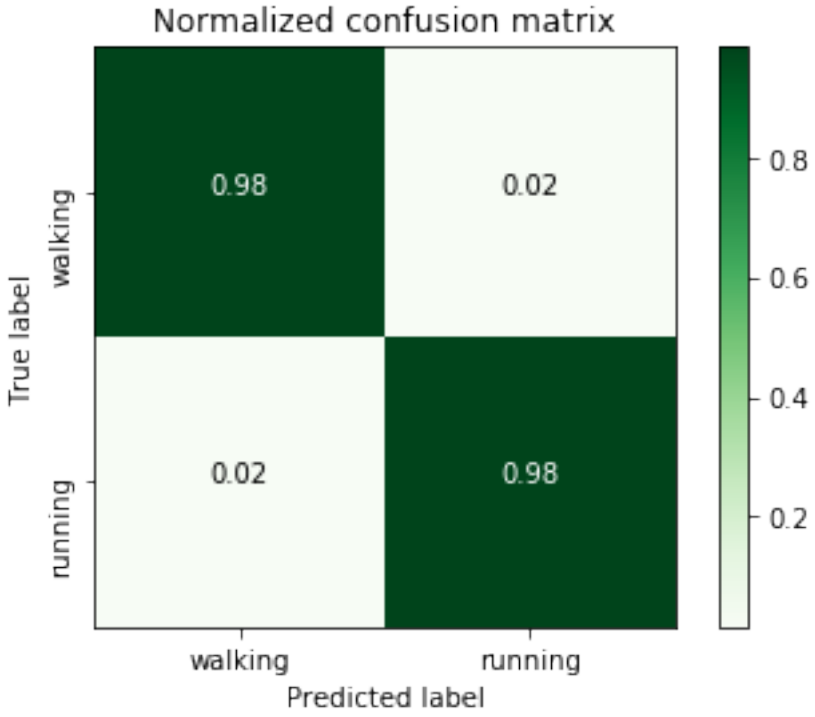


In [7]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Greens):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
```

```
            plt.xticks(tick_marks, classes)
            plt.yticks(tick_marks, classes, rotation=90)

            fmt = '.2f' if normalize else 'd'
            thresh = cm.max() / 2.
            for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                plt.text(j, i, format(cm[i, j], fmt),
                         horizontalalignment="center",
                         color="white" if cm[i, j] > thresh else "black")

            plt.ylabel('True label')
            plt.xlabel('Predicted label')

        cnf_matrix = confusion_matrix(y_test, y_pred)
        np.set_printoptions(precision=2)

        # Plot non-normalized confusion matrix
        plt.figure()
        class_names = ["walking", "running"]
        plot_confusion_matrix(cnf_matrix, classes=["walking", "running"],
                              title='Confusion matrix, without normalization')
        plt.figure()
        plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                              title='Normalized confusion matrix')
        plt.show()
```

```
Confusion matrix, without normalization
[[64  1]
 [ 1 60]]
Normalized confusion matrix
[[ 0.98  0.02]
 [ 0.02  0.98]]
```


Confusion matrix, without normalization
64
1
1
60
walking
running
True label
walking
running
Predicted label
60
50
40
30
20
10


Normalized confusion matrix
0.98
0.02
0.02
0.98
walking
running
True label
walking
running
Predicted label
0.8
0.6
0.4
0.2