MultiLayerPerceptrop_3_train_10_test

December 13, 2017

In [5]:
```
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
```

In [6]:
```
multi_layer_dup_train = pd.read_csv('../FeaturesCsvFile/featuresfile.csv')
multi_layer_dup_test = pd.read_csv('../FeaturesCsvFile/featuresfile_10.csv')
multi_layer_train = multi_layer_dup_train.drop_duplicates(subset=['User', 'Timestamp']
multi_layer_unique_test = multi_layer_dup_test.drop_duplicates(subset=['User', 'Timesta
multi_layer_test = multi_layer_unique_test.iloc[sample(range(len(multi_layer_unique_tes

print ('(#row,#column) of train dataset' , multi_layer_train.shape)
print ('(#row,#column) of test dataset' , multi_layer_test.shape)
```

```
('(#row,#column) of train dataset', (406, 46))
('(#row,#column) of test dataset', (40, 46))
```

In [7]:
```
X_train = multi_layer_train.values[:, 2:45]
y_train = multi_layer_train.values[:, 45]
X_test = multi_layer_test.values[:, 2:45]
y_test = multi_layer_test.values[:, 45]
```

In [8]:
```
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)
```

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

```
In [132]: mlp = MLPClassifier(hidden_layer_sizes=(15,),max_iter=60)
          mlp_pred=mlp.fit(X_train,y_train)
          y_pred = mlp.predict(X_test)
          print(confusion_matrix(y_test,y_pred))
          print(classification_report(y_test,y_pred))
          print('\nAccuracy of Multi-layer Perceptron Score: %.2f' % mlp.score(X_test,y_test))
          print('\nAccuracy of Accuracy Score : %.2f' % accuracy_score(y_test,y_pred))
```

```
['running' 'running' 'running' 'running' 'running' 'running' 'walking'
 'walking' 'walking' 'running' 'running' 'walking' 'walking' 'running'
 'walking' 'walking' 'walking' 'walking' 'walking' 'walking' 'walking'
 'walking' 'running' 'running' 'walking' 'walking' 'running' 'walking'
 'running' 'running' 'walking' 'walking' 'walking' 'running' 'running'
 'running' 'running' 'walking' 'walking' 'walking']
[[13  7]
 [ 5 15]]
             precision    recall  f1-score   support

    running       0.72      0.65      0.68        20
    walking       0.68      0.75      0.71        20

avg / total       0.70      0.70      0.70        40


Accuracy of Multi-layer Perceptron Score: 0.70

Accuracy of Accuracy Score : 0.70
```

```
In [123]: for i in range(0,len(mlp.coefs_[0])):
              print mlp.coefs_[0][i]
```

```
[-0.14832588 -0.0016869   0.19640928 -0.09960565  0.17380905 -0.16242179
  0.13349543 -0.02374858  0.16727954 -0.07438662 -0.16040824  0.11369525
  0.18569266 -0.10706482  0.05736786]
[ 0.19308784  0.34991407 -0.01503363 -0.08581065  0.00811677  0.25941156
  0.24988267  0.00531556  0.0879106   0.05692786  0.3219229  -0.04735964
  0.35550873 -0.0079372   0.27870583]
[ 0.04463051 -0.21271666 -0.12848605  0.20441581  0.36220749 -0.11841668
 -0.04271731 -0.15981034 -0.19554312  0.18589542 -0.00870839 -0.05293841
  0.11212792  0.14585617 -0.31795423]
[-0.0419372  -0.10697277  0.21755814 -0.10296885 -0.12960097  0.07206938
 -0.31385372 -0.12143723  0.17091818 -0.05361283  0.19943426  0.10566402
  0.21958269  0.11879424  0.08046023]
[-0.19679028 -0.17616575 -0.04169148  0.27767236  0.02262695 -0.03973958
  0.1805154  -0.19325687  0.10979998  0.19590441 -0.14863489  0.0164746
  0.05916254 -0.09881695 -0.19507316]
[ 0.17538038 -0.3090144   0.20313079  0.40151156  0.07191118 -0.24502224
```

```
  0.06548774  0.23395856 -0.28431186  0.03575353 -0.41624276 -0.0456095
  0.08672698  0.17095521 -0.11885955]
[ 0.28406426  0.25250101  0.31559109  0.25632503  0.12791858 -0.05103644
 -0.24476778  0.14560913  0.08573068 -0.09146725  0.30370816 -0.00446255
  0.24737425  0.252935   -0.27254054]
[  5.77085738e-02  -1.72880106e-01  -3.59086890e-01   5.71809613e-02
   2.96493888e-01  -2.04328472e-01   3.77348622e-01   1.74747617e-01
   3.27600858e-01  -6.24062984e-02   1.06517826e-04  -8.29637375e-02
  -1.99410428e-01   3.09798290e-02   1.96633874e-01]
[-0.15477211 -0.24574909  0.00646108 -0.3829058   0.39291821 -0.00815564
  0.40982726 -0.22665871  0.39626689 -0.11317337  0.03012081 -0.32022111
 -0.00312841  0.21541448  0.22351709]
[ 0.04764688  0.11868709  0.24415005  0.06570779  0.12002237 -0.21578723
  0.25379073 -0.18001221  0.23217645 -0.18357123 -0.14493764 -0.12023188
 -0.01641436  0.10776536 -0.17114073]
[-0.11322951 -0.25717372  0.25664525 -0.22125817  0.24794967 -0.1070209
  0.15469011 -0.02379979 -0.05604391 -0.09690305  0.27268272 -0.28549774
 -0.00838848  0.27508241  0.07345669]
[ 0.09700731 -0.28922293  0.09654695 -0.11161879  0.29119028 -0.01067938
  0.39759234  0.09446626 -0.09310214  0.04548453  0.0847003  -0.38582301
  0.1820418  -0.09177828 -0.36074049]
[-0.20287545  0.07097811  0.15358541 -0.0682961   0.15985593 -0.20938828
  0.16423614  0.10922753  0.20065401 -0.31086398  0.20931865 -0.26436441
 -0.2323279   0.01660678 -0.01063379]
[-0.27523345  0.3070684   0.17030215  0.21378672 -0.03200652 -0.12174052
 -0.30964913  0.15552181 -0.0521165  -0.36605183 -0.20636639 -0.11493742
  0.26802007 -0.27906061 -0.11438685]
[ 0.1679838  -0.00097729  0.25669502  0.05435626  0.22258298  0.17730281
  0.06383851 -0.25192982 -0.10136595  0.10792481  0.1799989   0.22427637
  0.10688186  0.30953756  0.13795044]
[ 0.12199559 -0.27545474 -0.07513353  0.18136846 -0.19653318 -0.16037755
  0.15625345 -0.05887303  0.23599811  0.09211018  0.06047696 -0.06996411
 -0.12891312  0.00565997  0.10133365]
[ 0.34974633  0.2050466   0.06272901 -0.13494607 -0.08649233  0.05997031
  0.29087194 -0.08577835 -0.11810794  0.15760276  0.2442759   0.17754149
  0.23028974  0.1894997   0.27275647]
[ 0.38276053  0.21964838  0.21726085 -0.12123721  0.20421511 -0.23229882
 -0.21500187  0.14366238  0.00711466 -0.16953708  0.09090952  0.02854524
  0.06549499 -0.0212512   0.32892267]
[-0.06976501  0.01492345 -0.15986645 -0.0357712  -0.36068285  0.22802
 -0.36718358 -0.10991929 -0.39002713  0.01598226 -0.02840341  0.28866986
 -0.17236885  0.0039396   0.10985078]
[ 0.3746858  -0.36653778  0.20539377  0.22928013 -0.25228579 -0.22011527
 -0.11939852 -0.24523653  0.02992883 -0.23171256  0.15266464  0.29253961
 -0.06889729 -0.14334933 -0.05170511]
[ 0.02215812  0.22057124  0.10669028 -0.18771659 -0.04996432 -0.04806994
 -0.07761321 -0.10914212  0.14932034 -0.20719492  0.10036791  0.06030728
  0.23955413  0.06050558 -0.04288482]
```

```
[-0.17409128 -0.16965298  0.00296333 -0.01924404  0.03379692  0.12229819
 -0.07476982  0.10412256 -0.18420695 -0.1872276   0.08493389 -0.34158042
  0.03907976  0.00799764 -0.0958614 ]
[-0.08214233  0.26044484 -0.02780715  0.14479735 -0.26508453  0.03810995
  0.24809806 -0.10335171  0.02915463 -0.10395192  0.32943794  0.24066859
  0.09990676 -0.12304669 -0.18808206]
[-0.2979393   0.07559592 -0.12633893 -0.19902505  0.34682852  0.04619203
 -0.20183824 -0.08315331  0.24900273 -0.1651726   0.32050999 -0.09617861
  0.29994015  0.1095376  -0.00272871]
[ 0.18775042 -0.0777644  -0.20763985  0.01221095 -0.12872941 -0.03235293
  0.37848178 -0.0941898   0.37092591 -0.13737457 -0.22532084 -0.17780159
  0.14681151 -0.28748703  0.12262934]
[ 0.03283072 -0.02993344 -0.15629266 -0.12234092  0.3648562  -0.24123603
 -0.18607444 -0.25498138 -0.07796693 -0.0606423   0.11696159 -0.02020966
  0.20219955  0.22547627  0.14470908]
[-0.19463684 -0.0657491  -0.14208487  0.09622469 -0.32969336 -0.28618395
 -0.33198198 -0.10350596  0.04841334 -0.03381644 -0.35390337 -0.0468073
  0.187496   -0.11302548  0.09947734]
[ 0.13981526 -0.07490286  0.09570873  0.23714873 -0.24390545 -0.23840668
  0.20516213  0.05023365  0.12325048  0.10855846 -0.23528621  0.29153197
 -0.33610657  0.08198491  0.16897469]
[ 0.38724218 -0.017476    0.31123329 -0.20666975 -0.03427847 -0.15662636
 -0.23997831 -0.0435458  -0.00355307 -0.21645226  0.18682175 -0.06156636
 -0.20786695 -0.06185011  0.18366068]
[-0.18255477  0.29114625  0.2434522  -0.19992921 -0.10962617  0.21468935
  0.20617761 -0.17667671 -0.3250293  -0.10466001 -0.07956175 -0.23969737
 -0.26638306 -0.30576583 -0.23692377]
[ 0.26560717 -0.13005001  0.14742958 -0.03621046  0.10179015  0.28093779
 -0.29567661  0.08907186  0.02155883 -0.05436684 -0.2767174  -0.09144669
 -0.21123797  0.0583933  -0.23946749]
[ 0.01920103  0.13847251  0.08075203  0.37512252  0.10951623 -0.13916573
  0.15414107  0.19589714  0.07958008  0.19791812 -0.03673451  0.20201795
  0.18468862 -0.18944128  0.04818427]
[ 0.33392067  0.18646961  0.0583122   0.38001021 -0.09790946  0.40382632
  0.16721975  0.06414439 -0.1043317   0.10438914 -0.3510709   0.33581091
 -0.19866244  0.07080422  0.10419433]
[-0.3282666   0.03196397 -0.17150983  0.09565853  0.25981879 -0.19144601
  0.03784988 -0.02615715  0.49214744 -0.14412699  0.32161302 -0.35554824
  0.20094953 -0.06318327 -0.40280342]
[-0.39741919  0.18121998  0.04563106  0.01133312  0.16220052 -0.05094936
  0.0051839   0.18391864 -0.01144542 -0.17176319 -0.08297467 -0.1516798
  0.1101966   0.022635    0.03047028]
[ 0.1799002  -0.00440717 -0.33323745 -0.34487735  0.11418818  0.14183559
  0.05535726 -0.25319106  0.13655967  0.13833295 -0.18434571 -0.09844208
  0.36885025  0.17318861 -0.32709014]
[-0.23390966  0.29257852  0.2352722   0.08438576 -0.45473943 -0.13080274
 -0.17019313 -0.18058091  0.09634349 -0.17426977 -0.40900549 -0.10483176
 -0.09134652 -0.18890556 -0.12472898]
```

```
[-0.21310863 -0.09800851  0.25408033  0.31253059 -0.46212115  0.26923653
 -0.26178609 -0.02629252 -0.35413671 -0.02181341  0.18148354 -0.21324896
  0.08540564  0.02694339  0.19769011]
[ 0.12272466  0.40776267 -0.23006088  0.03282085  0.32961007 -0.29785991
  0.43756318 -0.09247498  0.02543837  0.2287696   0.15773692 -0.12985151
 -0.18502026  0.21312736  0.13145529]
[  3.81828118e-02   8.32672048e-02  -2.60405588e-01   1.86364271e-02
  -1.13027982e-01  -3.16363222e-01   2.72907311e-01  -3.47865831e-01
   4.08868703e-01   1.37540024e-01  -9.62297002e-02   3.63675252e-04
  -6.57174821e-02  -1.98172358e-01   9.99160518e-03]
[-0.21027805  0.01667815 -0.26185071 -0.38938506 -0.06519805 -0.00778855
  0.35330803 -0.22015157  0.04054785  0.00771493 -0.099369    0.08332159
  0.33427539  0.08750741 -0.11694933]
[-0.0370899   0.27912556 -0.4269452  -0.22701199  0.2674002  -0.10487541
  0.06311405  0.08569657  0.38162453  0.05502582  0.3126147   0.01854311
 -0.13695047  0.06219793 -0.17916562]
[-0.20775639 -0.01060788 -0.02628799 -0.06118866  0.07133874 -0.33688527
  0.00452911 -0.30719179 -0.16878182  0.12515594 -0.19485744  0.04968137
 -0.09882008  0.02909134  0.160281  ]
```

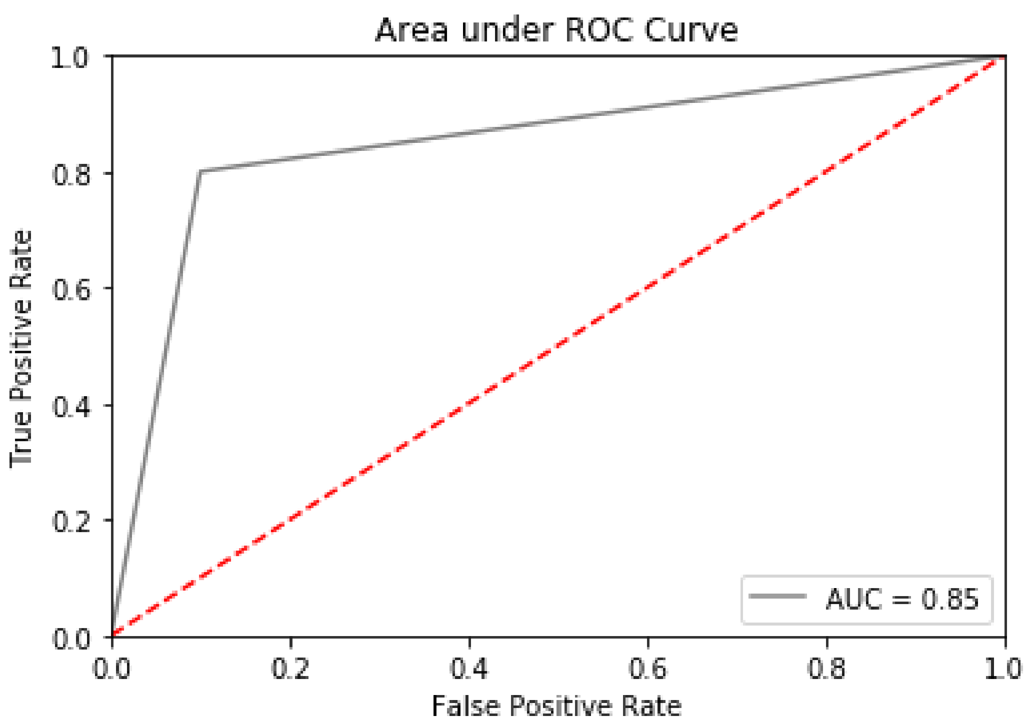
```
In [124]: avg_weight = []
          for i in range(0,len(mlp.coefs_[0])):
              avg_weight.append(np.mean(mlp.coefs_[0][i]))
          print ('Important features (featureName, weigh of important, #column)')
          header = list(multi_layer_train.head(1))
          important_feature = []
          for i in range(0,len(avg_weight)):
               important_feature.append((header[i+2],avg_weight[i],i+2))
          sorted_list = sorted(important_feature,key=lambda important_feature: important_featu
          for j in range(0,len(sorted_list)):
                  first_imp_fea = sorted_list[0]
                  second_imp_fea = sorted_list[1]
                  print sorted_list[j]
```

```
Important features (featureName, weigh of important, #column)
('Bin2,x', 0.13403755122428809, 3)
('Bin7,y', 0.12100037016790845, 18)
('Bin5,y', 0.11033708440750736, 16)
('Bin7,x', 0.10716550790342998, 8)
('TimeDiffPeaks-z', 0.097141816547573451, 34)
('TimeDiffPeaks-y', 0.09467666960425368, 33)
('AvgAcc-z', 0.076782762297476548, 40)
('Bin8,y', 0.061947211002028359, 19)
('Bin3,z', 0.033143447417199198, 24)
('Bin8,x', 0.029181653954342104, 9)
('StdDev-z', 0.02755359358421124, 43)
('Bin8,z', 0.024917417564454047, 29)
```

```
('Bin4,x', 0.020939838728566845, 5)
('Bin4,z', 0.018348813127264971, 25)
('Bin1,x', 0.01667337314830869, 2)
('Bin1,z', 0.015792597534347412, 22)
('Bin9,x', 0.014650772696619907, 10)
('Bin10,x', 0.010523429867257674, 11)
('Bin1,y', 0.0074127712160413596, 12)
('Bin6,x', 0.0017170414397350074, 7)
('Bin6,y', -0.00067019253068868008, 17)
('Bin2,y', -0.0035956829504882621, 13)
('Bin6,z', -0.0041762882708015255, 27)
('AvgAbsDiff-y', -0.0075628346317582829, 36)
('Bin5,z', -0.0099900346147206442, 26)
('Bin9,z', -0.012060369886151064, 30)
('Bin3,x', -0.012143857693701817, 4)
('Bin3,y', -0.014285823141720878, 14)
('Bin5,x', -0.015200846960264462, 6)
('AvgAbsDiff-z', -0.015825217621623505, 37)
('AvgAbsDiff-x', -0.016202690497325683, 35)
('AvgAcc-y', -0.021543056344720594, 39)
('TimeDiffPeaks-x', -0.02469231927366522, 32)
('Bin10,y', -0.027649692406183268, 21)
('StdDev-x', -0.028534960130635837, 41)
('StdDev-y', -0.029841131099100759, 42)
('Bin4,y', -0.050456671649889498, 15)
('Bin2,z', -0.05676281156994667, 23)
('AvgResAcc', -0.064819988571627171, 44)
('Bin9,y', -0.068840121303876997, 20)
('Bin10,z', -0.084756170118829452, 31)
('AvgAcc-x', -0.10364893250522102, 38)
('Bin7,z', -0.1046518181446817, 28)
```

```
In [125]: 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(labels, predns)
    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")
```

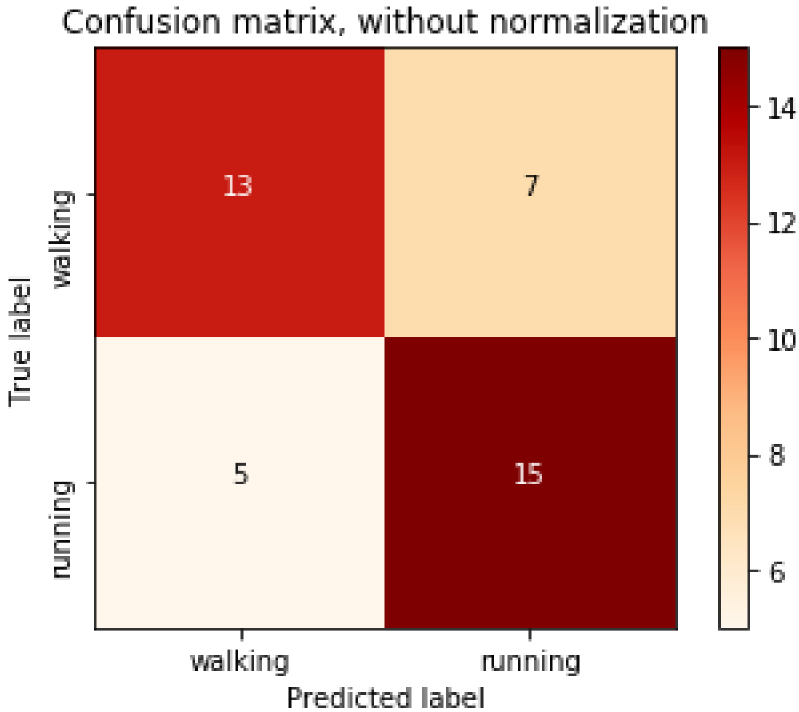


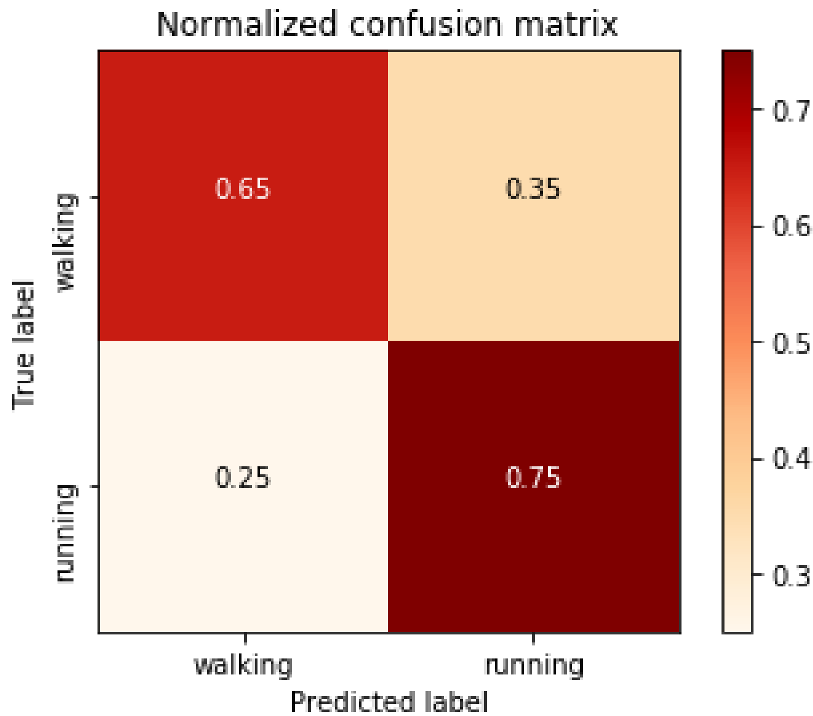
In [138]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
```

```python
                              cmap=plt.cm.OrRd):
    """
    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.savefig('H:/mastersProject/activity_analyzer/LogisticRegression/cm_lr', dpi=10

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
plt.show()
```

```
Confusion matrix, without normalization
[[13  7]
```

```
 [ 5 15]]
Normalized confusion matrix
[[ 0.65  0.35]
 [ 0.25  0.75]]
```





Normalized confusion matrix
0.65
0.35
0.25
0.75
walking
running
True label
walking
running
Predicted label
0.7
0.6
0.5
0.4
0.3