15train15testLogisticRegression

December 13, 2017

```
In [1]: import pandas as pd
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import confusion_matrix
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import roc_auc_score
        import numpy as np
        from pandas import DataFrame,Series
        from matplotlib.colors import ListedColormap
        from random import sample
        import matplotlib.pyplot as plt
```

```
In [ ]: # 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 [17]: # Data of 15 people for training & testing the model, splitting the train-test set
         df_features = pd.read_csv("H:/mastersProject/activity_analyzer/LogisticRegression/Data
         df_features_3people = pd.read_csv("H:/mastersProject/activity_analyzer/LogisticRegress

         frames = [df_features, df_features_3people]
         df_15 = pd.concat(frames)

         #Drop duplicates
```

```
df_unique = df_15.drop_duplicates(subset=['User', 'Timestamp'])
df_unique.head()
df_unique.describe()
print("Shape of training and testing data", df_unique.shape)

X_data = df_unique.values[:, 2:45]
y_data = df_unique.values[:, 45]
usersList = set(df_unique.values[:,0])
print(len(usersList)+2) # Userid is for 3 people hence
```

```
Shape of training and testing data (821, 46)
15
```

In [18]:
```
# Splitting the training and testing set by 33%
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size = 0.3, :
# Fitting the logistic regression model
clf = LogisticRegression(C=0.01, random_state=1)
clf.fit(X_train, y_train)
```

Out[18]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=1, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [19]:
```
predict = clf.predict(X_test)
logisticRegScore = clf.score(X_test, y_test)

plt.figure(1, figsize=(4, 3))
plt.clf()
print("Logistic regression Score")
print(logisticRegScore*100)
print("Coefficients of the features")
print(clf.coef_)
print(X_train.shape)

# Convert all the values to float
float_array=np.array(X_train,dtype=np.float32)
feature_importance = np.std(float_array, 0)*np.absolute(clf.coef_)

np_column_list = np.array(df_unique.columns.tolist())
column_names = np_column_list[2:45,]

# featureimp_list = feature_importance.split(" ")
# print("List= ", featureimp_list)
print("Column Names=", column_names)
print("Feature importance=", feature_importance)
print(np.sort(feature_importance))
```

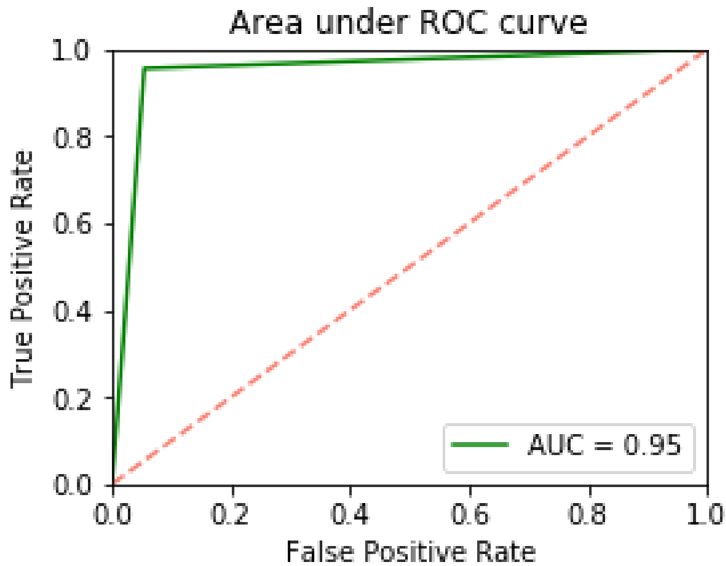
```
        # TimeDiff-X
        # StdDev-x
        # TimeDiffPeaks-y

Logistic regression Score
95.1417004049
Coefficients of the features
[[ -2.31054393e-03  -4.18999361e-03  -7.68049737e-03  -6.73008359e-04
    8.89051737e-03   1.58326400e-02   7.78248184e-03  -7.39253755e-03
   -1.05935509e-02  -4.50038446e-03  -4.11123837e-03  -8.00785339e-03
   -9.81673182e-03  -6.26217494e-03   2.63922862e-04  -2.93999692e-03
    4.03506675e-03   6.86358498e-03   7.37677689e-03   4.27038378e-03
   -3.68747089e-03  -3.85208413e-03  -1.13626247e-04   1.87845998e-03
    7.04841493e-04  -4.34782918e-04  -2.54994531e-04   1.14525297e-03
   -9.02447956e-04  -8.51511695e-04  -6.65044092e-02   6.11487841e-02
    7.48142953e-02  -3.58883969e-01  -2.53469402e-01  -3.42396804e-01
    4.99067902e-02   1.19460745e-01  -9.90639226e-02  -4.15955108e-01
   -3.01088118e-01  -4.07693080e-01   1.35578358e-02]]
(574, 43)
Column Names= ['Bin1,x' 'Bin2,x' 'Bin3,x' 'Bin4,x' 'Bin5,x' 'Bin6,x' 'Bin7,x' 'Bin8,x'
 'Bin9,x' 'Bin10,x' 'Bin1,y' 'Bin2,y' 'Bin3,y' 'Bin4,y' 'Bin5,y' 'Bin6,y'
 'Bin7,y' 'Bin8,y' 'Bin9,y' 'Bin10,y' 'Bin1,z' 'Bin2,z' 'Bin3,z' 'Bin4,z'
 'Bin5,z' 'Bin6,z' 'Bin7,z' 'Bin8,z' 'Bin9,z' 'Bin10,z' 'TimeDiffPeaks-x'
 'TimeDiffPeaks-y' 'TimeDiffPeaks-z' 'AvgAbsDiff-x' 'AvgAbsDiff-y'
 'AvgAbsDiff-z' 'AvgAcc-x' 'AvgAcc-y' 'AvgAcc-z' 'StdDev-x' 'StdDev-y'
 'StdDev-z' 'AvgResAcc']
Feature importance= [[  4.51133721e-05   2.04003588e-04   5.38006546e-04   4.37410940e-05
    5.58557566e-04   9.71777877e-04   5.04284542e-04   5.05617535e-04
    5.52573134e-04   1.18969050e-04   7.64959969e-05   2.73427300e-04
    4.45873933e-04   3.26179198e-04   1.31550617e-05   1.37763322e-04
    2.05725492e-04   4.08180708e-04   4.31305110e-04   1.39270193e-04
    8.01386747e-05   2.21450500e-04   8.07730704e-06   1.01285171e-04
    3.99829969e-05   2.27178746e-05   1.08738200e-05   4.19540988e-05
    2.53787510e-05   1.34049128e-05   8.55245602e-01   9.49327036e-01
    1.03148258e+00   8.63810978e-01   5.43848623e-01   7.15464718e-01
    1.23592204e-01   2.06835532e-01   1.23821188e-01   1.25888198e+00
    7.95675721e-01   1.08387371e+00   5.44202559e-03]]
[[  8.07730704e-06   1.08738200e-05   1.31550617e-05   1.34049128e-05
    2.27178746e-05   2.53787510e-05   3.99829969e-05   4.19540988e-05
    4.37410940e-05   4.51133721e-05   7.64959969e-05   8.01386747e-05
    1.01285171e-04   1.18969050e-04   1.37763322e-04   1.39270193e-04
    2.04003588e-04   2.05725492e-04   2.21450500e-04   2.73427300e-04
    3.26179198e-04   4.08180708e-04   4.31305110e-04   4.45873933e-04
    5.04284542e-04   5.05617535e-04   5.38006546e-04   5.52573134e-04
    5.58557566e-04   9.71777877e-04   5.44202559e-03   1.23592204e-01
    1.23821188e-01   2.06835532e-01   5.43848623e-01   7.15464718e-01
    7.95675721e-01   8.55245602e-01   8.63810978e-01   9.49327036e-01
    1.03148258e+00   1.08387371e+00   1.25888198e+00]]
```

```
In [20]: from sklearn.metrics import confusion_matrix

         cm = confusion_matrix(y_test, predict, labels=["walking", "running"])
         print(cm)
```

```
[[130   6]
 [  6 105]]
```

```
In [21]: #Area under ROC
         from sklearn.metrics import roc_curve
         from sklearn.metrics import auc
         from sklearn.preprocessing import LabelEncoder
         import matplotlib.pyplot as plt

         # # Encode the labels for ROC plot
         def encode_label(y_test):
             y_test_binary = []
             for y in y_test:
                 if y == "walking":
                     y_test_binary.append(1)
                 else:
                     y_test_binary.append(0)
             return y_test_binary

         y_test_binary = encode_label(y_test)
         y_predict_binary = encode_label(predict)

         # Compute fpr, tpr, thresholds and roc auc
         # fpr, tpr, thresholds = roc_curve(y_test_binary, probas_[:, 1])
         fpr, tpr, thresholds = roc_curve(y_test_binary, y_predict_binary)
         roc_auc = auc(fpr, tpr)
         print(roc_auc)

         # Plot ROC curve
         plt.plot(fpr, tpr, label='AUC = %0.2f' % roc_auc, color="green")
         plt.plot([0, 1], [0, 1], 'k--', color="salmon")  # random predictions curve, 50% accu
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.0])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('Area under ROC curve')
         plt.legend(loc="lower right")
         # plt.savefig('H:/mastersProject/activity_analyzer/LogisticRegression/roc_lr', dpi=20
         plt.show()
```

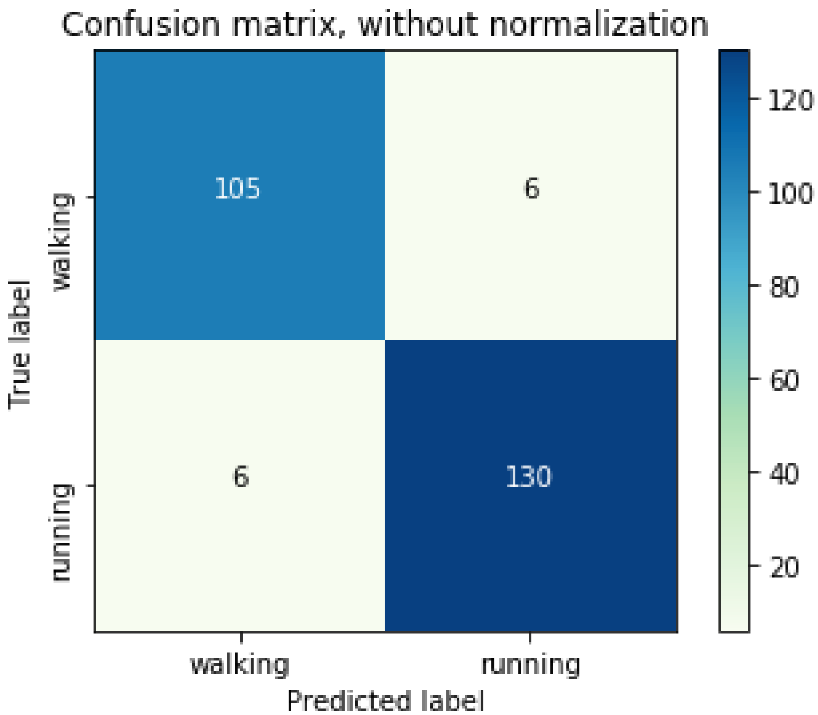
```
0.950914149444
```

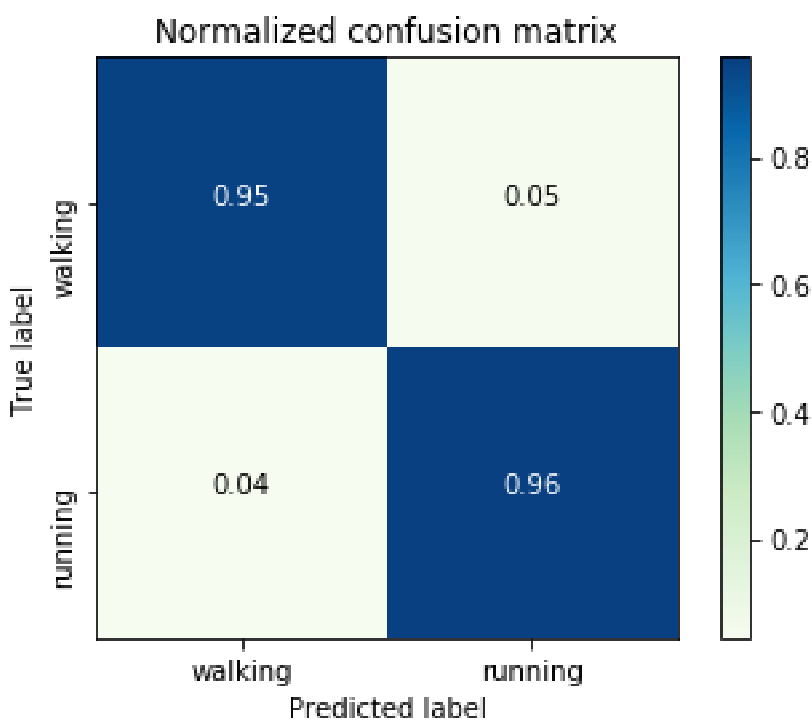



```
In [23]: #Confusion matrix plot
         import itertools
         import numpy as np
         import matplotlib.pyplot as plt

         def plot_confusion_matrix(cm, classes,
                                   normalize=False,
                                   title='Confusion matrix',
                                   cmap=plt.cm.GnBu):
             """
             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)
```

```python
    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.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, predict)
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=100

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
# plt.savefig('H:/mastersProject/activity_analyzer/LogisticRegression/cm_lr_normalize
plt.show()
```

```
Confusion matrix, without normalization
[[105   6]
 [  6 130]]
Normalized confusion matrix
[[ 0.95  0.05]
 [ 0.04  0.96]]
```


Confusion matrix, without normalization
105
6
6
130
walking
running
True label
walking
running
Predicted label
120
100
80
60
40
20


Normalized confusion matrix
0.95
0.05
0.04
0.96
walking
running
True label
walking
running
Predicted label
0.8
0.6
0.4
0.2