HAR_PREDICTION_MODELS

March 16, 2019

In [1]:
```
import numpy as np
import pandas as pd
```

0.1 Obtain the train and test data

In [2]:
```
train = pd.read_csv('UCI_HAR_dataset/csv_files/train.csv')
test = pd.read_csv('UCI_HAR_dataset/csv_files/test.csv')
print(train.shape, test.shape)
```

```
(7352, 564) (2947, 564)
```

In [3]:
```
train.head(3)
```

Out[3]:
```
   tBodyAccmeanX  tBodyAccmeanY  tBodyAccmeanZ  tBodyAccstdX  tBodyAccstdY  \
0       0.288585      -0.020294      -0.132905     -0.995279     -0.983111
1       0.278419      -0.016411      -0.123520     -0.998245     -0.975300
2       0.279653      -0.019467      -0.113462     -0.995380     -0.967187

   tBodyAccstdZ  tBodyAccmadX  tBodyAccmadY  tBodyAccmadZ  tBodyAccmaxX  ...  \
0     -0.913526     -0.995112     -0.983185     -0.923527     -0.934724  ...
1     -0.960322     -0.998807     -0.974914     -0.957686     -0.943068  ...
2     -0.978944     -0.996520     -0.963668     -0.977469     -0.938692  ...

   angletBodyAccMeangravity  angletBodyAccJerkMeangravityMean  \
0                 -0.112754                          0.030400
1                  0.053477                         -0.007435
2                 -0.118559                          0.177899

   angletBodyGyroMeangravityMean  angletBodyGyroJerkMeangravityMean  \
0                      -0.464761                          -0.018446
1                      -0.732626                           0.703511
2                       0.100699                           0.808529

   angleXgravityMean  angleYgravityMean  angleZgravityMean  subject  Activity  \
0          -0.841247           0.179941          -0.058627        1         5
1          -0.844788           0.180289          -0.054317        1         5
2          -0.848933           0.180637          -0.049118        1         5
```

```
   ActivityName
0      STANDING
1      STANDING
2      STANDING

[3 rows x 564 columns]
```

```
In [4]: # get X_train and y_train from csv files
        X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
        y_train = train.ActivityName
```

```
In [5]: # get X_test and y_test from test csv file
        X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
        y_test = test.ActivityName
```

```
In [6]: print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
        print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
```

```
X_train and y_train : ((7352, 561),(7352,))
X_test  and y_test  : ((2947, 561),(2947,))
```

1 Let's model with our data

1.0.1 Labels that are useful in plotting confusion matrix

```
In [7]: labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_UPSTAIRS
```

1.0.2 Function to plot the confusion matrix

```
In [8]: import itertools
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.metrics import confusion_matrix
        plt.rcParams["font.family"] = 'DejaVu Sans'

        def plot_confusion_matrix(cm, classes,
                                  normalize=False,
                                  title='Confusion matrix',
                                  cmap=plt.cm.Blues):
            if normalize:
                cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

            plt.imshow(cm, interpolation='nearest', cmap=cmap)
            plt.title(title)
            plt.colorbar()
            tick_marks = np.arange(len(classes))
            plt.xticks(tick_marks, classes, rotation=90)
```

```
    plt.yticks(tick_marks, classes)

    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')
```

1.0.3 Generic function to run any model specified

```
In [9]: from datetime import datetime
        def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_normalize=
                         print_cm=True, cm_cmap=plt.cm.Greens):


            # to store results at various phases
            results = dict()

            # time at which model starts training
            train_start_time = datetime.now()
            print('training the model..')
            model.fit(X_train, y_train)
            print('Done \n \n')
            train_end_time = datetime.now()
            results['training_time'] =  train_end_time - train_start_time
            print('training_time(HH:MM:SS.ms) - {}\n\n'.format(results['training_time']))


            # predict test data
            print('Predicting test data')
            test_start_time = datetime.now()
            y_pred = model.predict(X_test)
            test_end_time = datetime.now()
            print('Done \n \n')
            results['testing_time'] = test_end_time - test_start_time
            print('testing time(HH:MM:SS:ms) - {}\n\n'.format(results['testing_time']))
            results['predicted'] = y_pred


            # calculate overall accuracty of the model
            accuracy = metrics.accuracy_score(y_true=y_test, y_pred=y_pred)
            # store accuracy in results
            results['accuracy'] = accuracy
```

```
print('---------------------')
print('|      Accuracy      |')
print('---------------------')
print('\n    {}\n\n'.format(accuracy))


# confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
results['confusion_matrix'] = cm
if print_cm:
    print('--------------------')
    print('| Confusion Matrix |')
    print('--------------------')
    print('\n {}'.format(cm))

# plot confusin matrix
plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(cm, classes=class_labels, normalize=True, title='Normalized c
plt.show()

# get classification report
print('-------------------------')
print('| Classifiction Report |')
print('-------------------------')
classification_report = metrics.classification_report(y_test, y_pred)
# store report in results
results['classification_report'] = classification_report
print(classification_report)

# add the trained  model to the results
results['model'] = model

return results
```

1.0.4 Method to print the gridsearch Attributes

```
In [10]: def print_grid_search_attributes(model):
             # Estimator that gave highest score among all the estimators formed in GridSearch
             print('--------------------------')
             print('|      Best Estimator     |')
             print('--------------------------')
             print('\n\t{}\n'.format(model.best_estimator_))


             # parameters that gave best results while performing grid search
```

```
        print('--------------------------')
        print('|      Best parameters     |')
        print('--------------------------')
        print('\tParameters of best estimator : \n\n\t{}\n'.format(model.best_params_))


        #  number of cross validation splits
        print('---------------------------------')
        print('|   No of CrossValidation sets   |')
        print('--------------------------------')
        print('\n\tTotal numbre of cross validation sets: {}\n'.format(model.n_splits_))


        # Average cross validated score of the best estimator, from the Grid Search
        print('--------------------------')
        print('|        Best Score        |')
        print('--------------------------')
        print('\n\tAverage Cross Validate scores of best estimator : \n\n\t{}\n'.format(mo
```

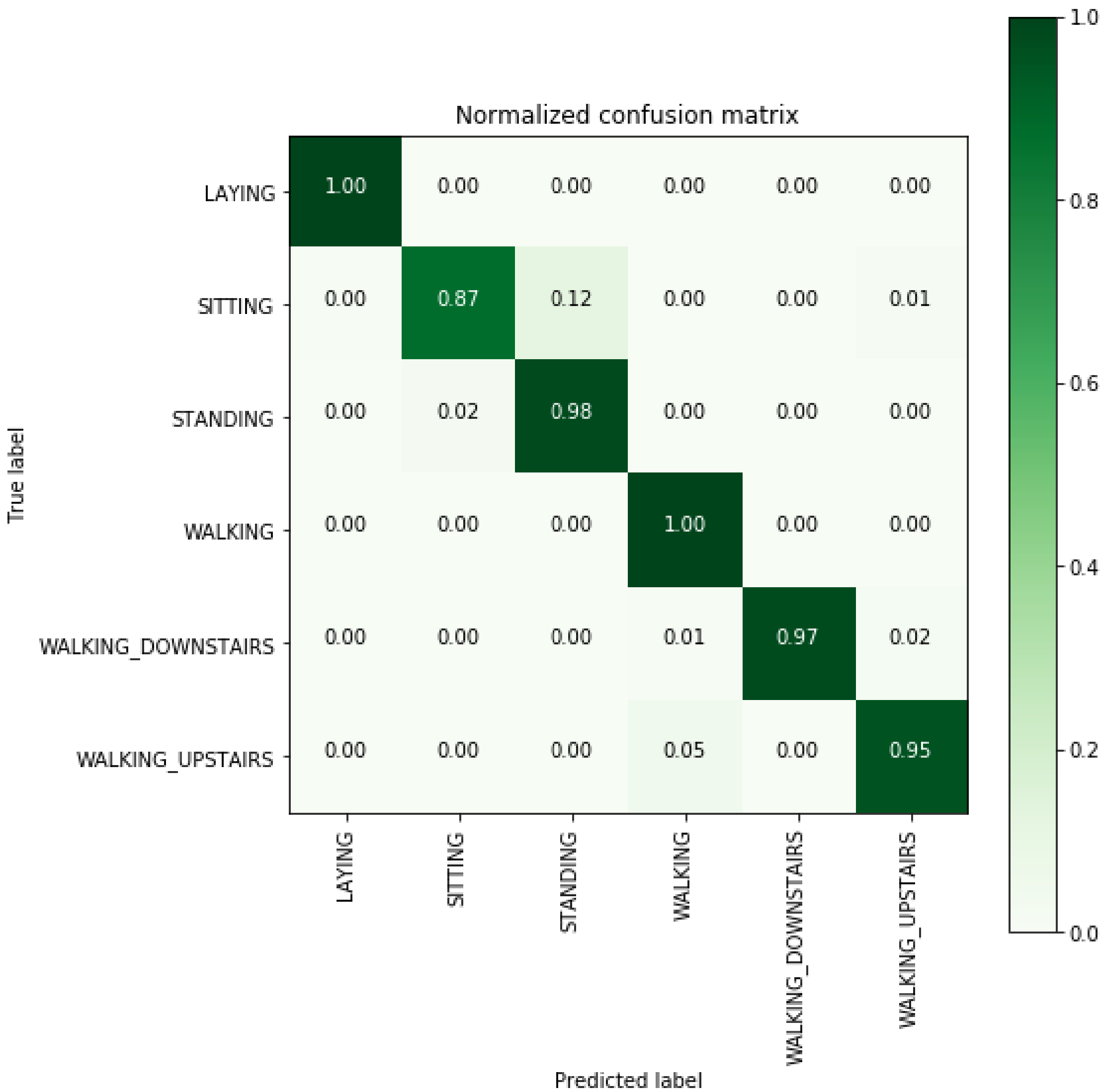
2 1. Logistic Regression with Grid Search

```
In [11]: from sklearn import linear_model
         from sklearn import metrics

         from sklearn.model_selection import GridSearchCV
```

```
In [13]: # start Grid search
         parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
         log_reg = linear_model.LogisticRegression()
         log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1, n_jobs=-
         log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X_test, y_test,
```

```
training the model..
Fitting 3 folds for each of 12 candidates, totalling 36 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  1.4min finished
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\sklearn\linear_model\lo
  FutureWarning)
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\sklearn\linear_model\lo
  "this warning.", FutureWarning)


Done
```

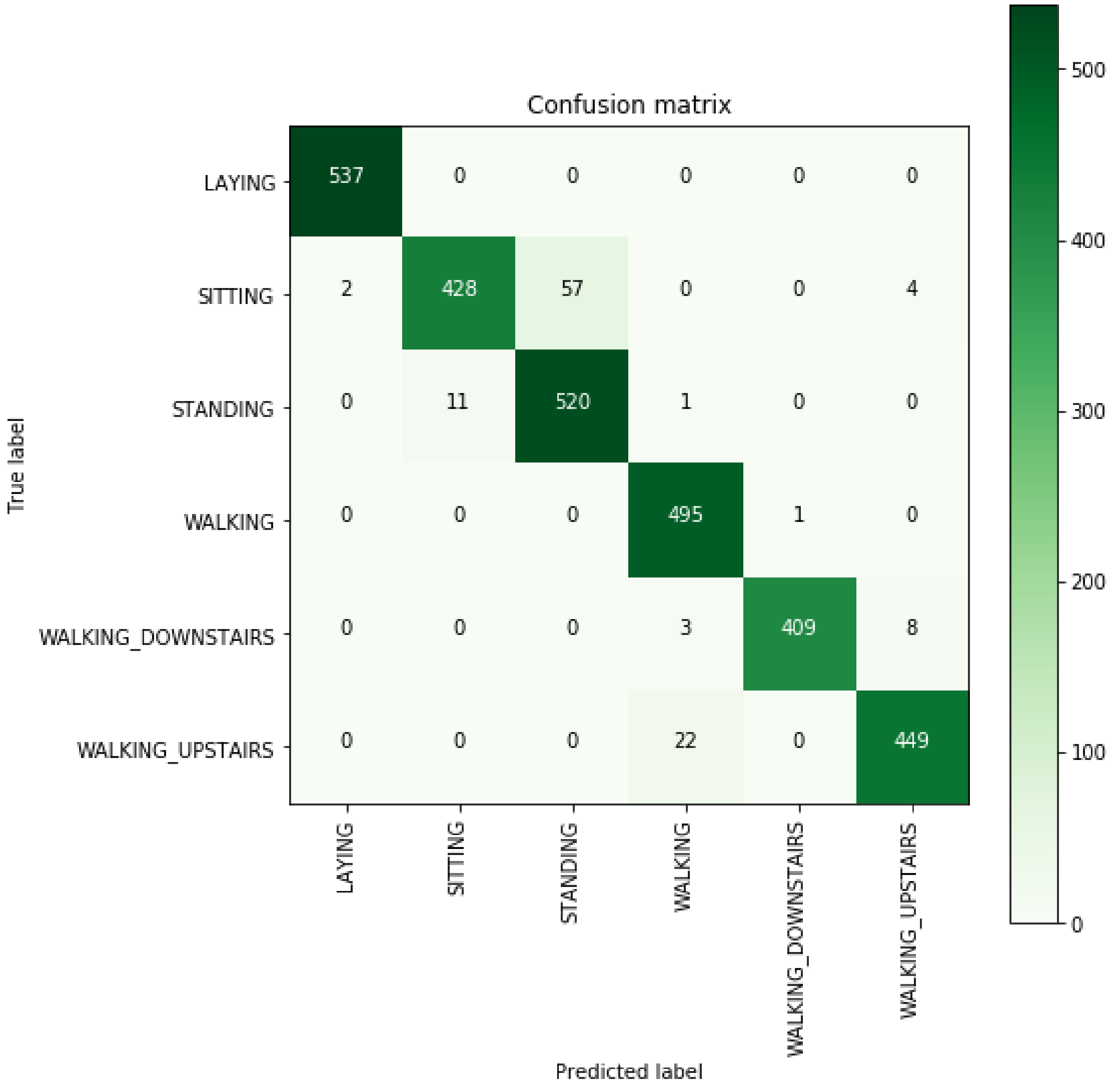
```
training_time(HH:MM:SS.ms) - 0:01:32.121277


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.005913


---------------------
|      Accuracy      |
---------------------

    0.9630132337970818


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  2 428  57   0   0   4]
 [  0  11 520   1   0   0]
 [  0   0   0 495   1   0]
 [  0   0   0   3 409   8]
 [  0   0   0  22   0 449]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.87      0.92       491
          STANDING       0.90      0.98      0.94       532
           WALKING       0.95      1.00      0.97       496
WALKING_DOWNSTAIRS       1.00      0.97      0.99       420
  WALKING_UPSTAIRS       0.97      0.95      0.96       471

         micro avg       0.96      0.96      0.96      2947
```

```
   macro avg       0.97      0.96      0.96      2947
weighted avg       0.96      0.96      0.96      2947
```

```
In [14]: plt.figure(figsize=(8,8))
         plt.grid(b=False)
         plot_confusion_matrix(log_reg_grid_results['confusion_matrix'], classes=labels, cmap=p
         plt.show()
```




```
In [15]: # observe the attributes of the model
         print_grid_search_attributes(log_reg_grid_results['model'])

--------------------------
|      Best Estimator     |
```

```
--------------------------

        LogisticRegression(C=30, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

--------------------------
|      Best parameters     |
--------------------------
        Parameters of best estimator :

        {'C': 30, 'penalty': 'l2'}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|         Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.9458650707290533
```

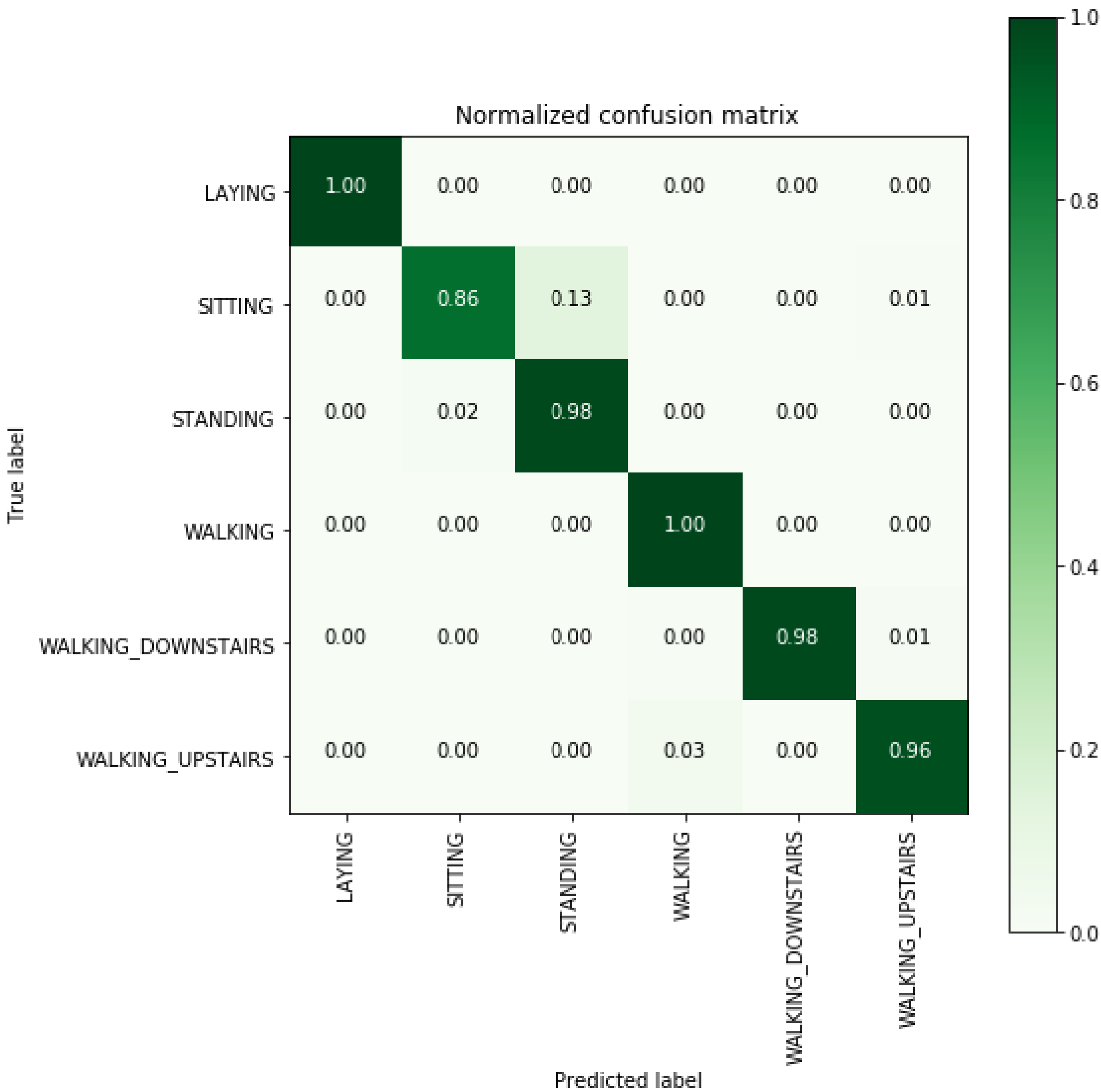
3 2. Linear SVC with GridSearch

```
In [16]: from sklearn.svm import LinearSVC
```

```
In [17]: parameters = {'C':[0.125, 0.5, 1, 2, 8, 16]}
         lr_svc = LinearSVC(tol=0.00005)
         lr_svc_grid = GridSearchCV(lr_svc, param_grid=parameters, n_jobs=-1, verbose=1)
         lr_svc_grid_results = perform_model(lr_svc_grid, X_train, y_train, X_test, y_test, cla
```

```
training the model..
Fitting 3 folds for each of 6 candidates, totalling 18 fits


c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\sklearn\model_selection\
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:   26.6s finished
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\sklearn\svm\base.py:922
```

```
  "the number of iterations.", ConvergenceWarning)


Done


training_time(HH:MM:SS.ms) - 0:00:30.736681


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.006907


---------------------
|      Accuracy      |
---------------------

    0.9657278588394977


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  1 423  64   0   0   3]
 [  0   9 523   0   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0   2 413   5]
 [  0   0   0  16   1 454]]
```




```
--------------------------
| Classifiction Report |
--------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.98      0.86      0.92       491
          STANDING       0.89      0.98      0.93       532
           WALKING       0.96      1.00      0.98       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.96      0.97       471

         micro avg       0.97      0.97      0.97      2947
```

```
   macro avg       0.97      0.97      0.97      2947
weighted avg       0.97      0.97      0.97      2947
```

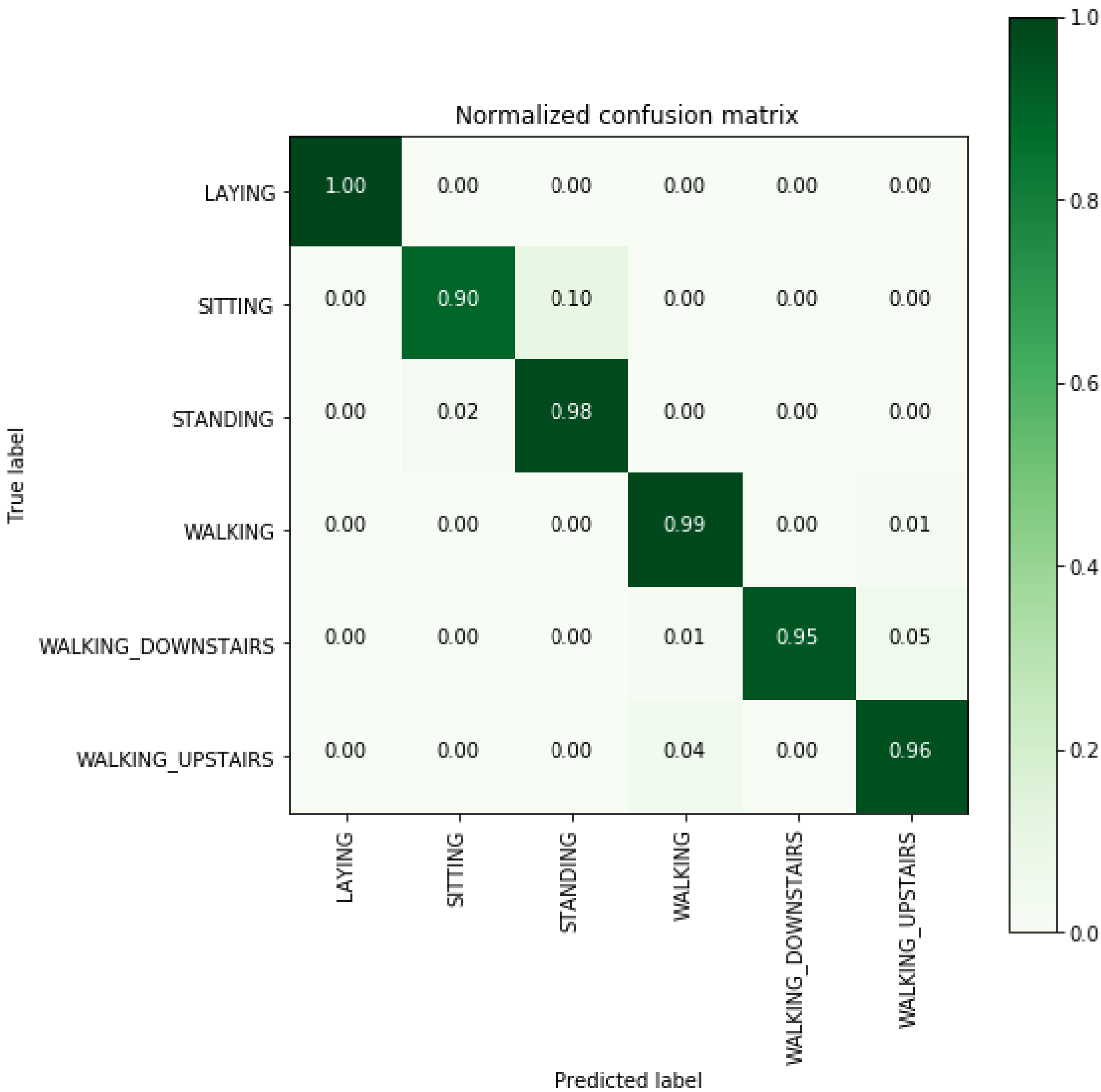
In [18]:
```
print_grid_search_attributes(lr_svc_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

        LinearSVC(C=2, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=5e-05,
     verbose=0)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'C': 2}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.9469532100108814
```

4 3. Kernel SVM with GridSearch

In [19]:
```
from sklearn.svm import SVC
parameters = {'C':[2,8,16],\
              'gamma': [ 0.0078125, 0.125, 2]}
rbf_svm = SVC(kernel='rbf')
rbf_svm_grid = GridSearchCV(rbf_svm,param_grid=parameters, n_jobs=-1)
rbf_svm_grid_results = perform_model(rbf_svm_grid, X_train, y_train, X_test, y_test, c
```

```
training the model..


c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\sklearn\model_selection\
  warnings.warn(CV_WARNING, FutureWarning)


Done


training_time(HH:MM:SS.ms) - 0:05:48.772455


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:02.371631


---------------------
|      Accuracy      |
---------------------

    0.9626739056667798


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 441  48   0   0   2]
 [  0  12 520   0   0   0]
 [  0   0   0 489   2   5]
 [  0   0   0   4 397  19]
 [  0   0   0  17   1 453]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.90      0.93       491
          STANDING       0.92      0.98      0.95       532
           WALKING       0.96      0.99      0.97       496
WALKING_DOWNSTAIRS       0.99      0.95      0.97       420
  WALKING_UPSTAIRS       0.95      0.96      0.95       471

         micro avg       0.96      0.96      0.96      2947
```

```
   macro avg       0.96      0.96      0.96      2947
weighted avg       0.96      0.96      0.96      2947
```

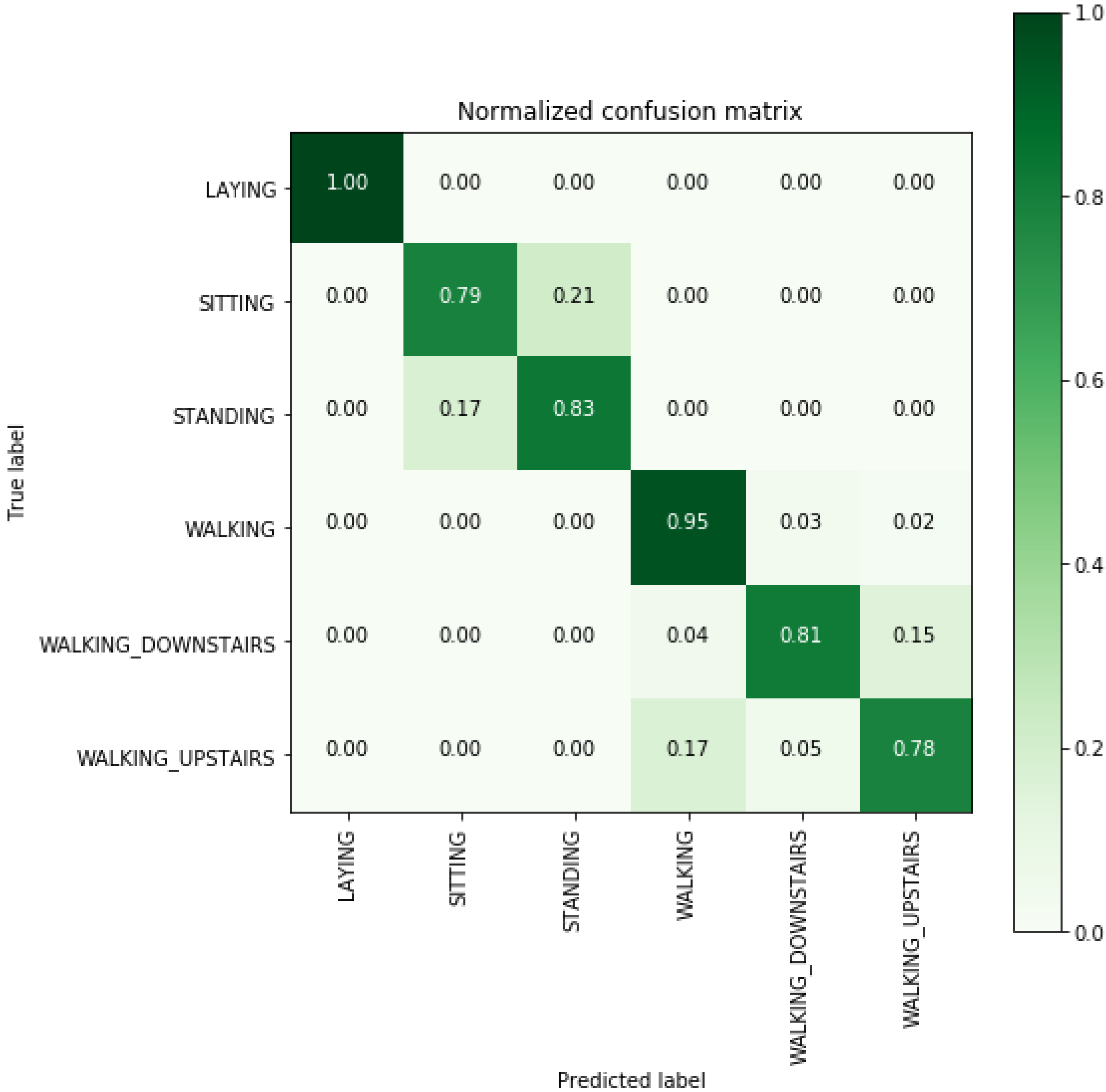
```
In [20]: print_grid_search_attributes(rbf_svm_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

        SVC(C=16, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.0078125, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'C': 16, 'gamma': 0.0078125}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.9440968443960827
```

5 4. Decision Trees with GridSearchCV

```
In [21]: from sklearn.tree import DecisionTreeClassifier
         parameters = {'max_depth':np.arange(3,10,2)}
         dt = DecisionTreeClassifier()
         dt_grid = GridSearchCV(dt,param_grid=parameters, n_jobs=-1)
         dt_grid_results = perform_model(dt_grid, X_train, y_train, X_test, y_test, class_label
         print_grid_search_attributes(dt_grid_results['model'])
```

```
training the model..

c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\sklearn\model_selection\
  warnings.warn(CV_WARNING, FutureWarning)

Done

training_time(HH:MM:SS.ms) - 0:00:08.387619

Predicting test data
Done

testing time(HH:MM:SS:ms) - 0:00:00.005944

---------------------
|      Accuracy      |
---------------------

    0.8635900916185952

--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 386 105   0   0   0]
 [  0  93 439   0   0   0]
 [  0   0   0 472  16   8]
 [  0   0   0  17 342  61]
 [  0   0   0  78  24 369]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.81      0.79      0.80       491
          STANDING       0.81      0.83      0.82       532
           WALKING       0.83      0.95      0.89       496
WALKING_DOWNSTAIRS       0.90      0.81      0.85       420
  WALKING_UPSTAIRS       0.84      0.78      0.81       471

         micro avg       0.86      0.86      0.86      2947
```

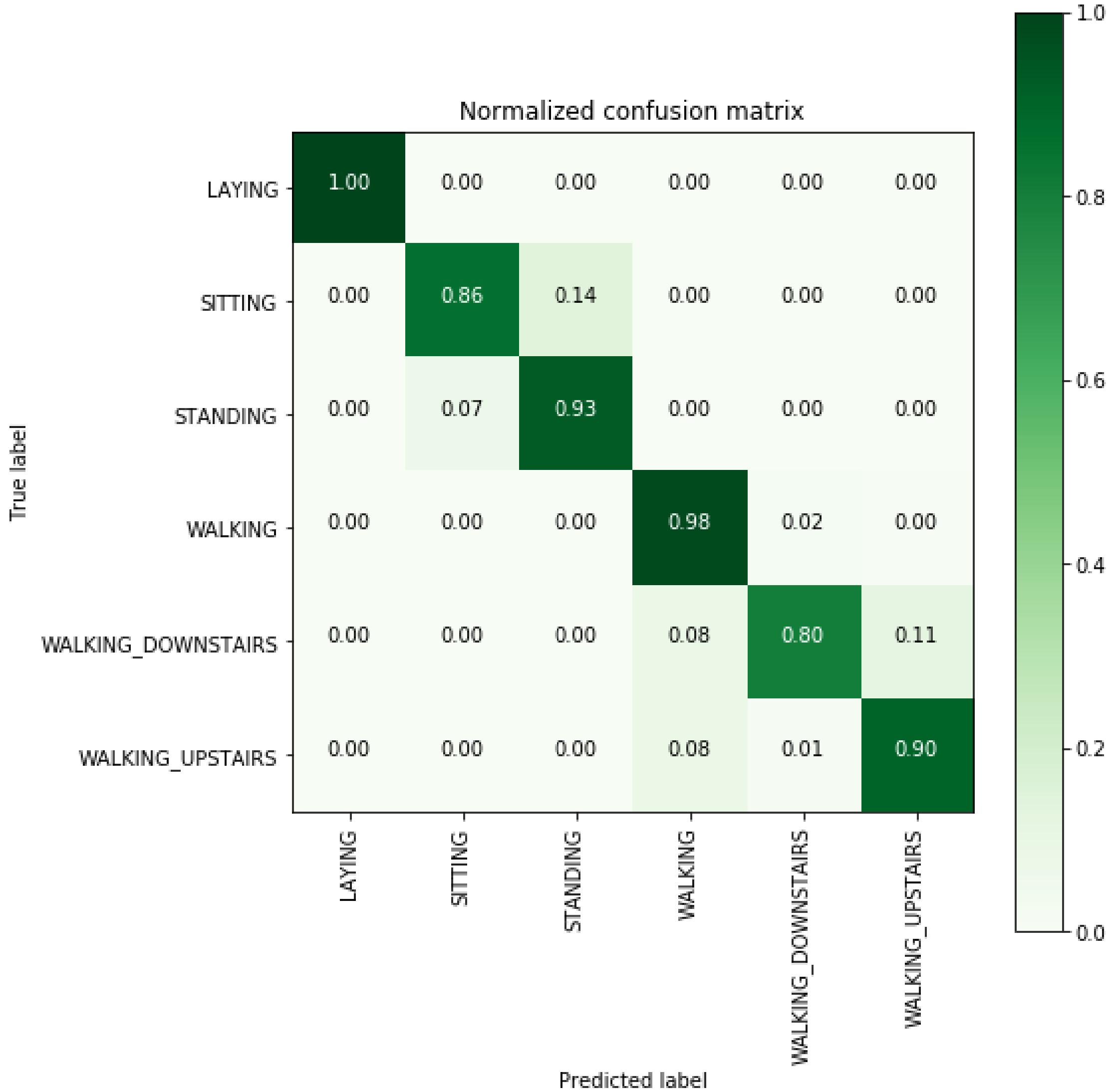
```
   macro avg       0.86      0.86      0.86      2947
weighted avg       0.86      0.86      0.86      2947

--------------------------
|      Best Estimator     |
--------------------------

        DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=7,
            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')

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'max_depth': 7}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.8392274211099021
```

6 5. Random Forest Classifier with GridSearch

In [22]:
```
from sklearn.ensemble import RandomForestClassifier
params = {'n_estimators': np.arange(10,201,20), 'max_depth':np.arange(3,15,2)}
rfc = RandomForestClassifier()
rfc_grid = GridSearchCV(rfc, param_grid=params, n_jobs=-1)
rfc_grid_results = perform_model(rfc_grid, X_train, y_train, X_test, y_test, class_lal
print_grid_search_attributes(rfc_grid_results['model'])
```

```
training the model..
```

```
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\sklearn\model_selection\
  warnings.warn(CV_WARNING, FutureWarning)


Done


training_time(HH:MM:SS.ms) - 0:03:15.851561


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.046874


---------------------
|      Accuracy      |
---------------------

    0.9158466236851035


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 421  70   0   0   0]
 [  0  38 494   0   0   0]
 [  0   0   0 485   9   2]
 [  0   0   0  35 337  48]
 [  0   0   0  40   6 425]]
```



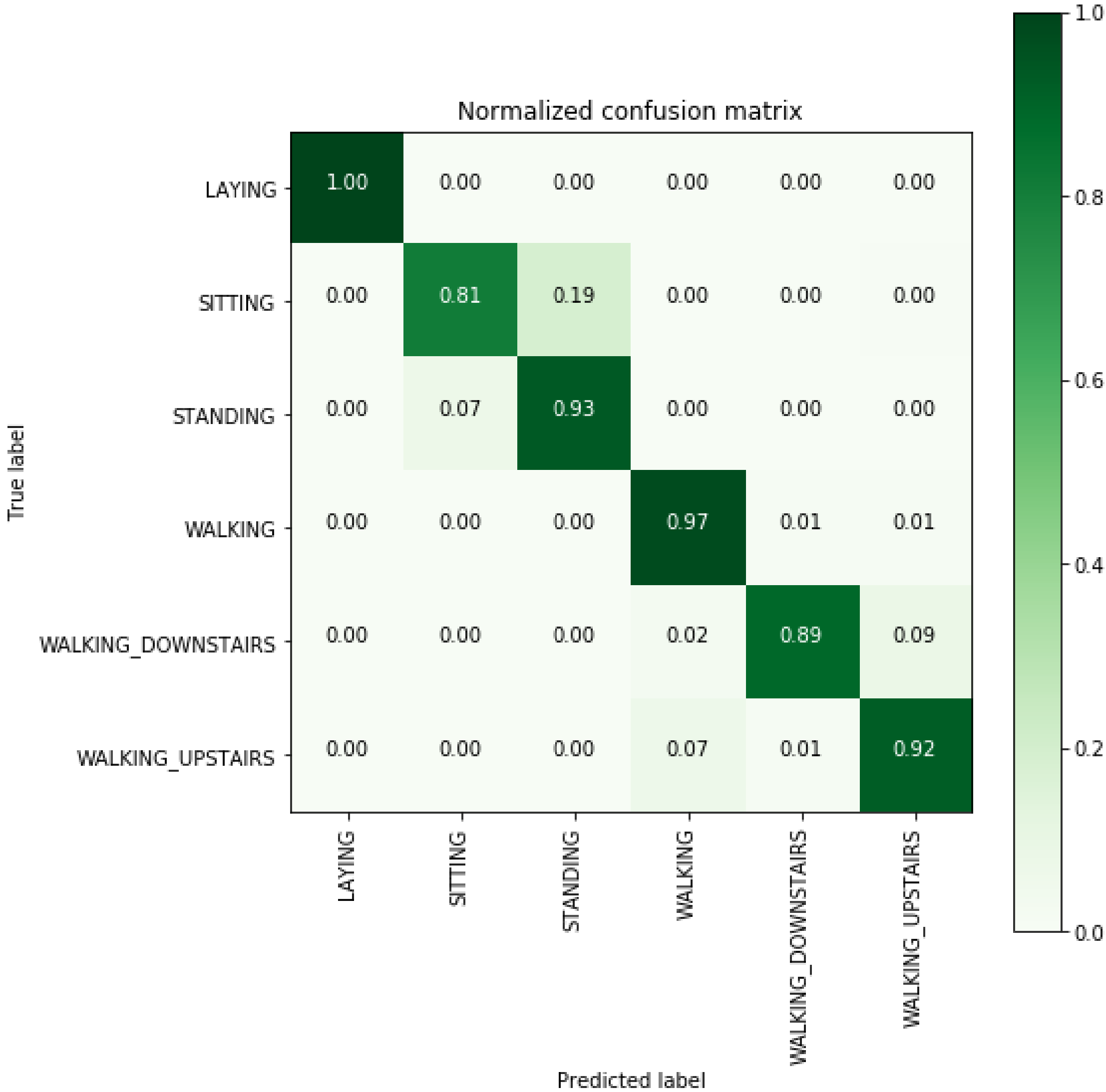

```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.92      0.86      0.89       491
          STANDING       0.88      0.93      0.90       532
           WALKING       0.87      0.98      0.92       496
WALKING_DOWNSTAIRS       0.96      0.80      0.87       420
  WALKING_UPSTAIRS       0.89      0.90      0.90       471

         micro avg       0.92      0.92      0.92      2947
```

```
     macro avg       0.92      0.91      0.91      2947
  weighted avg       0.92      0.92      0.92      2947

--------------------------
|      Best Estimator     |
--------------------------

	RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=7, max_features='auto', 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, n_estimators=130, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
	Parameters of best estimator :

	{'max_depth': 7, 'n_estimators': 130}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

	Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

	Average Cross Validate scores of best estimator :

	0.9151251360174102
```

7 6. Gradient Boosted Decision Trees With GridSearch

```
In [23]: from sklearn.ensemble import GradientBoostingClassifier
         param_grid = {'max_depth': np.arange(5,8,1), \
                      'n_estimators':np.arange(130,170,10)}
         gbdt = GradientBoostingClassifier()
         gbdt_grid = GridSearchCV(gbdt, param_grid=param_grid, n_jobs=-1)
         gbdt_grid_results = perform_model(gbdt_grid, X_train, y_train, X_test, y_test, class_l
         print_grid_search_attributes(gbdt_grid_results['model'])
```

```
training the model..

c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\sklearn\model_selection\
  warnings.warn(CV_WARNING, FutureWarning)

Done

training_time(HH:MM:SS.ms) - 0:28:30.458286

Predicting test data
Done

testing time(HH:MM:SS:ms) - 0:00:00.077851

---------------------
|      Accuracy      |
---------------------

    0.9229725144214456

--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 398  91   0   0   2]
 [  0  37 495   0   0   0]
 [  0   0   0 483   7   6]
 [  0   0   0  10 374  36]
 [  0   1   0  31   6 433]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.91      0.81      0.86       491
          STANDING       0.84      0.93      0.89       532
           WALKING       0.92      0.97      0.95       496
WALKING_DOWNSTAIRS       0.97      0.89      0.93       420
  WALKING_UPSTAIRS       0.91      0.92      0.91       471

         micro avg       0.92      0.92      0.92      2947
```

```
   macro avg       0.93      0.92      0.92      2947
weighted avg       0.92      0.92      0.92      2947

--------------------------
|      Best Estimator     |
--------------------------

        GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=5,
              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, n_estimators=160,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'max_depth': 5, 'n_estimators': 160}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.904923830250272
```

8 7. Comparing all models

```
In [24]: print('\n                     Accuracy     Error')
         print('                     ----------   --------')
         print('Logistic Regression : {:.04}%      {:.04}%'.format(log_reg_grid_results['accu
                                                          100-(log_reg_grid_results['accuracy
```

```
print('Linear SVC          : {:.04}%       {:.04}% '.format(lr_svc_grid_results['accu
                                                             100-(lr_svc_grid_results['accu

print('rbf SVM classifier  : {:.04}%      {:.04}% '.format(rbf_svm_grid_results['accu
                                                               100-(rbf_svm_grid_results['a

print('DecisionTree        : {:.04}%      {:.04}% '.format(dt_grid_results['accuracy']
                                                             100-(dt_grid_results['accuracy

print('Random Forest       : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy
                                                                100-(rfc_grid_results['accu
print('GradientBoosting DT : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy
                                                             100-(rfc_grid_results['accura
```

```
                      Accuracy     Error
                      ----------   --------
Logistic Regression : 96.3%        3.699%
Linear SVC          : 96.57%        3.427%
rbf SVM classifier  : 96.27%       3.733%
DecisionTree        : 86.36%       13.64%
Random Forest       : 91.58%       8.415%
GradientBoosting DT : 91.58%       8.415%
```

We can choose ***Logistic regression*** or ***Linear SVC*** or ***rbf SVM***.

9 Conclusion :

In the real world, domain-knowledge, EDA and feature-engineering matter most.