

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

## 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)

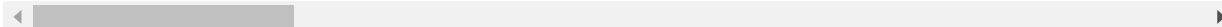
(7352, 564) (2947, 564)
```

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

Out[3]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tB
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.

3 rows × 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,))
```

## Let's model with our data

### Labels that are useful in plotting confusion matrix

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

### 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')

```

## 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=True, \
               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) - {}'.format(results['trainin
g_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) - {}'.format(results['testing_
time']))
results['predicted'] = y_pred

# calculate overall accuracy 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      {}'.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 confusion matrix
plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(cm, classes=class_labels, normalize=True, tit
le='Normalized confusion matrix', cmap = cm_cmap)
plt.show()

```

```

# get classification report
print('-----')
print('| Classification Report |')
print('-----')
classification_report = metrics.classification_report(y_test, y_pre
d)

# store report in results
results['classification_report'] = classification_report
print(classification_report)

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

return results

```

## 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.be
st_params_))

# number of cross validation splits

```

```

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

```

## 1. Logistic Regression with Grid Search

```

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

         from sklearn.model_selection import GridSearchCV

```

```

In [12]: # 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=-1)
         log_reg_grid_results = perform_model(log_reg_grid, X_train, y_train, X_test, y_test, class_labels=labels)

```

```
training the model..
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

```
[Parallel(n_jobs=-1)]: Done 36 out of 36 | elapsed: 4.3min finished
```

```
Done
```

```
training_time(HH:MM:SS.ms) - 0:04:45.228601
```

```
Predicting test data
```

```
Done
```

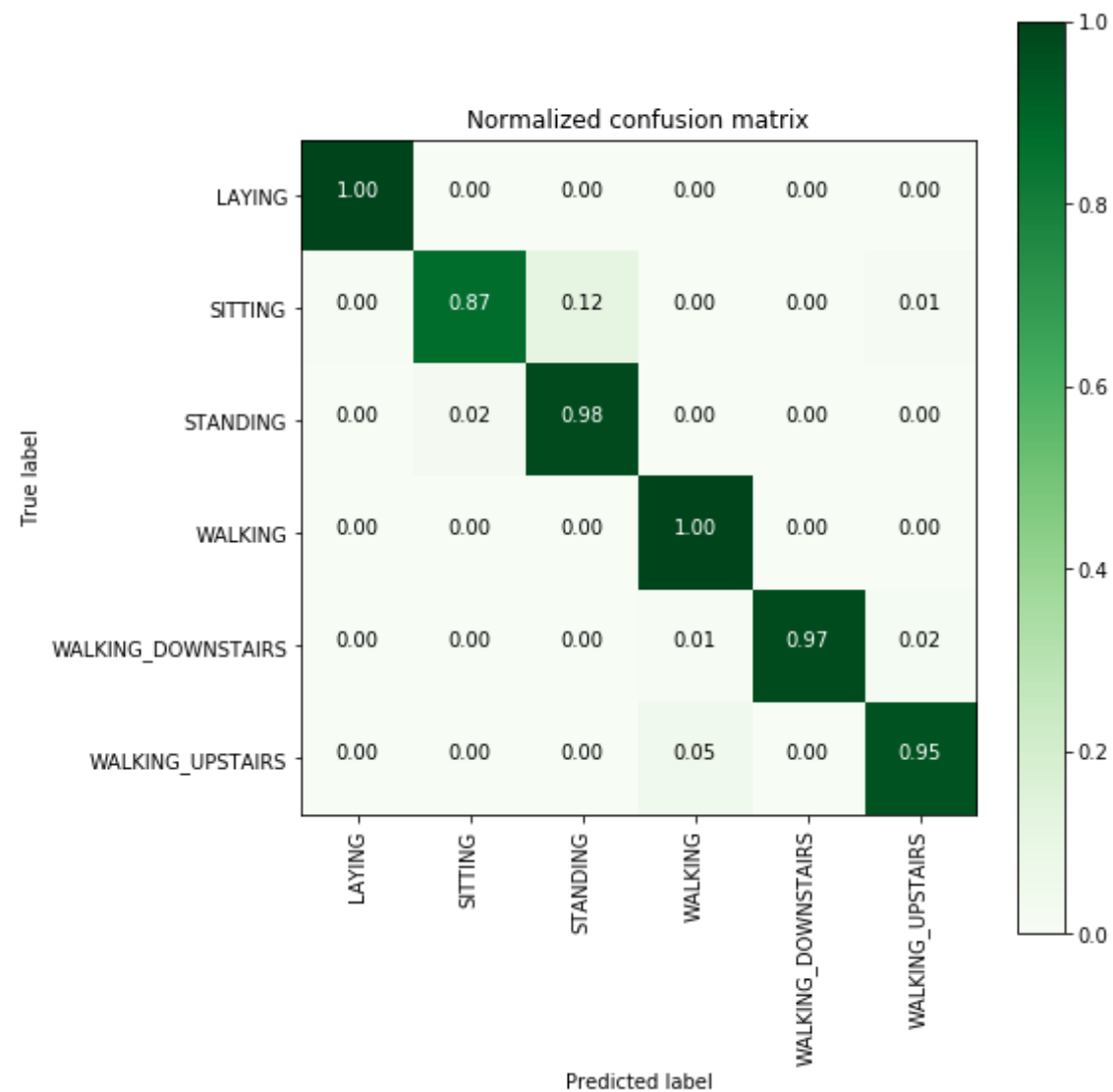
```
testing time(HH:MM:SS.ms) - 0:00:00.021958
```

```
-----  
|      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]]
```



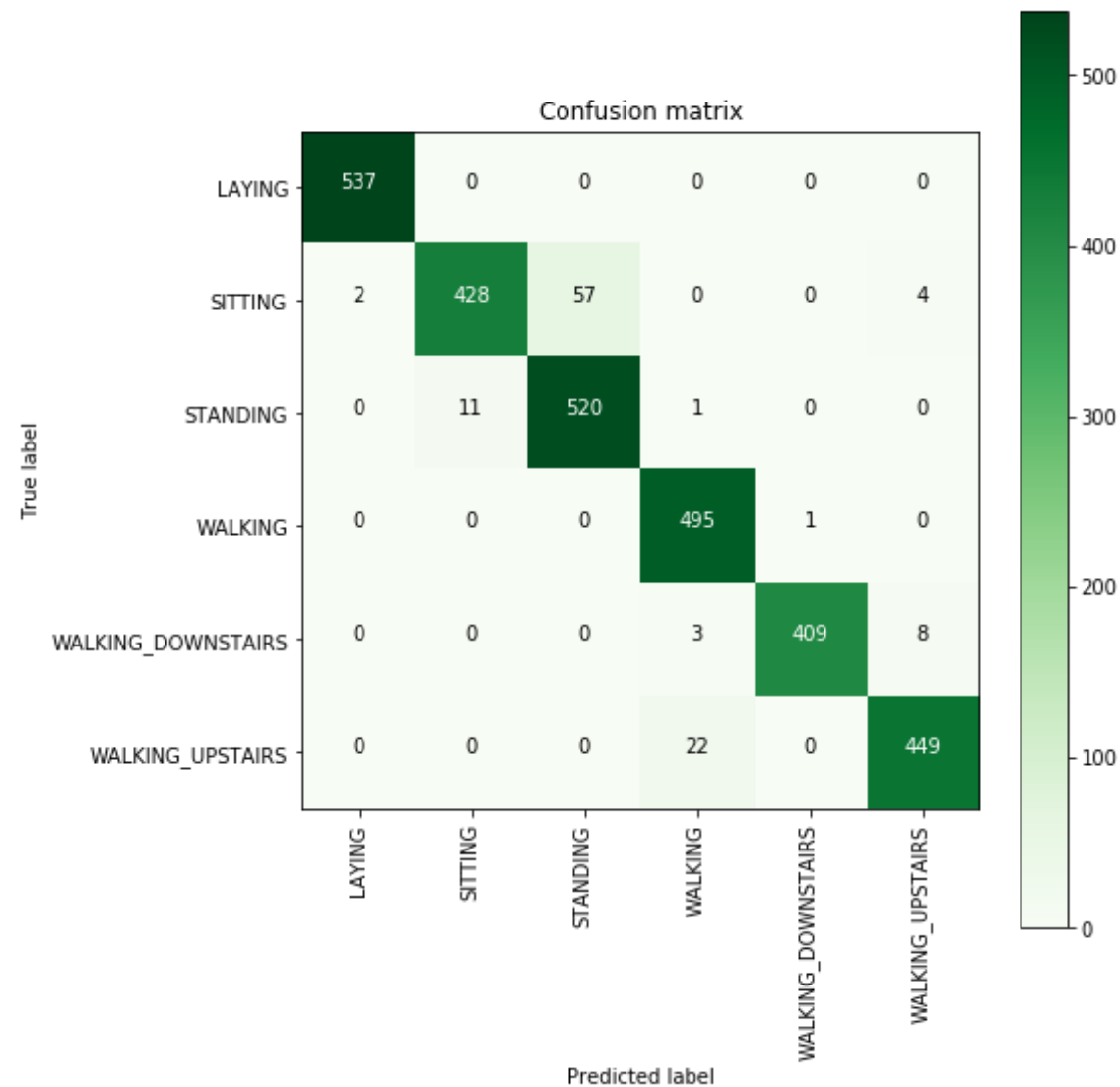
Classification Report

	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
avg / total	0.96	0.96	0.96	2947

```
In [13]: plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(log_reg_grid_results['confusion_matrix'], classes
=labels, cmap=plt.cm.Greens, )
plt.show()
```



```
In [14]: # 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_int
ercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
    penalty='l2', random_state=None, solver='liblinear', tol=0.00
01,
    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 :

0.9461371055495104

```

## 2. Linear SVC with GridSearch

```
In [15]: from sklearn.svm import LinearSVC
```

```
In [16]: 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, class_labels=labels)
```

training the model..

Fitting 3 folds for each of 6 candidates, totalling 18 fits

[Parallel(n\_jobs=-1)]: Done 18 out of 18 | elapsed: 1.9min finished

Done

training\_time(HH:MM:SS.ms) - 0:02:08.947784

Predicting test data

Done

testing time(HH:MM:SS.ms) - 0:00:00.014473

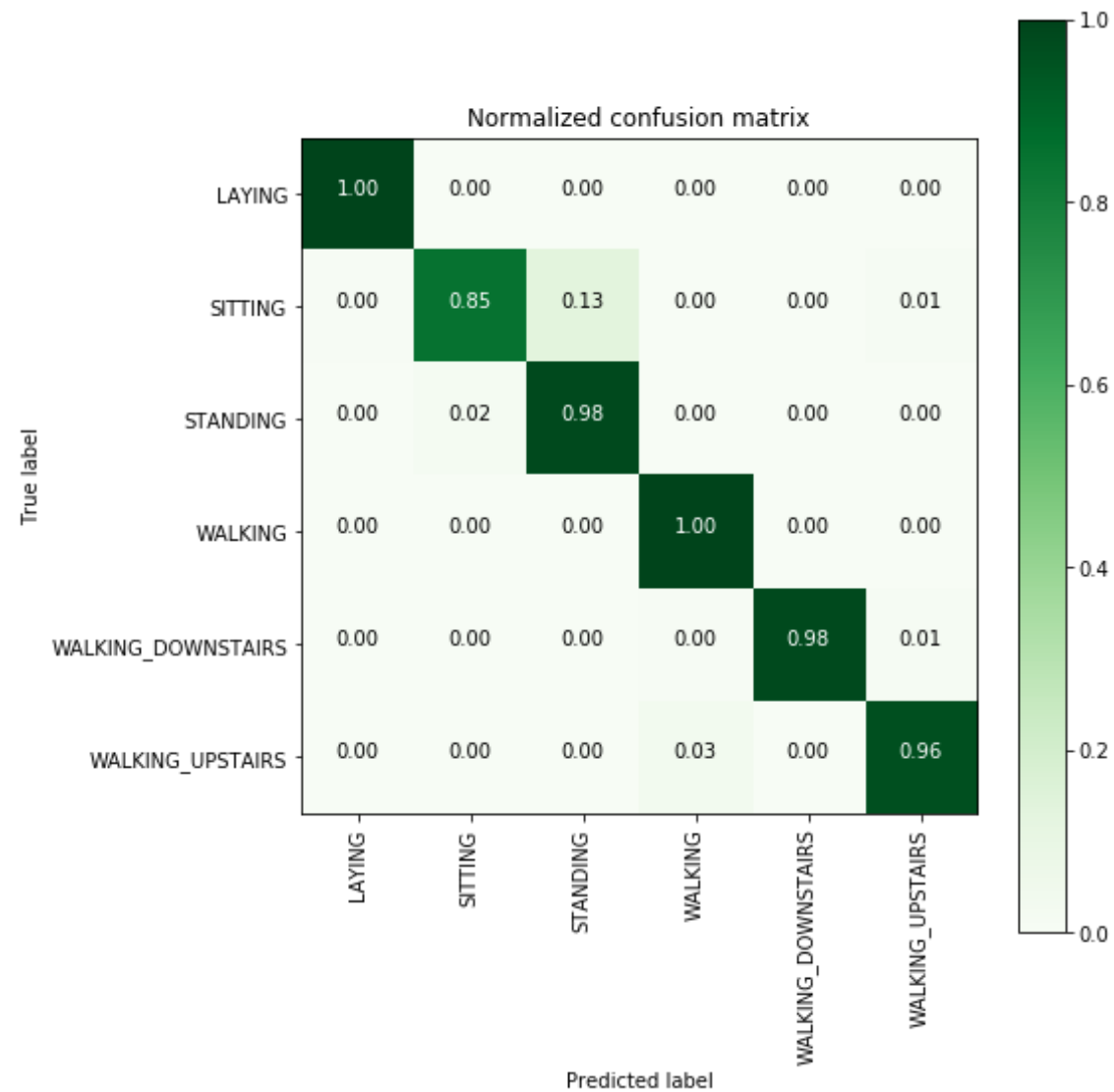
```
-----
|      Accuracy      |
|-----|
```

0.9636918900576857

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

```
[[537  0  0  0  0  0]
```

```
[ 2 418 66 0 0 5]
[ 0 9 522 1 0 0]
[ 0 0 0 496 0 0]
[ 0 0 0 2 413 5]
[ 0 0 0 16 1 454]]
```



```

-----
| Classification Report |
-----
              precision    recall  f1-score   support

    LAYING                1.00      1.00      1.00        537
    SITTING               0.98      0.85      0.91        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

 avg / total              0.97      0.96      0.96       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,
e,
      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 number of cross validation sets: 3

```

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

Average Cross Validate scores of best estimator :

0.9465451577801959

### 3. Kernel SVM with GridSearch

```
In [20]: 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, class_labels=labels)
```

training the model..  
Done

training\_time(HH:MM:SS.ms) - 0:12:52.622461

Predicting test data  
Done

testing time(HH:MM:SS.ms) - 0:00:05.581884

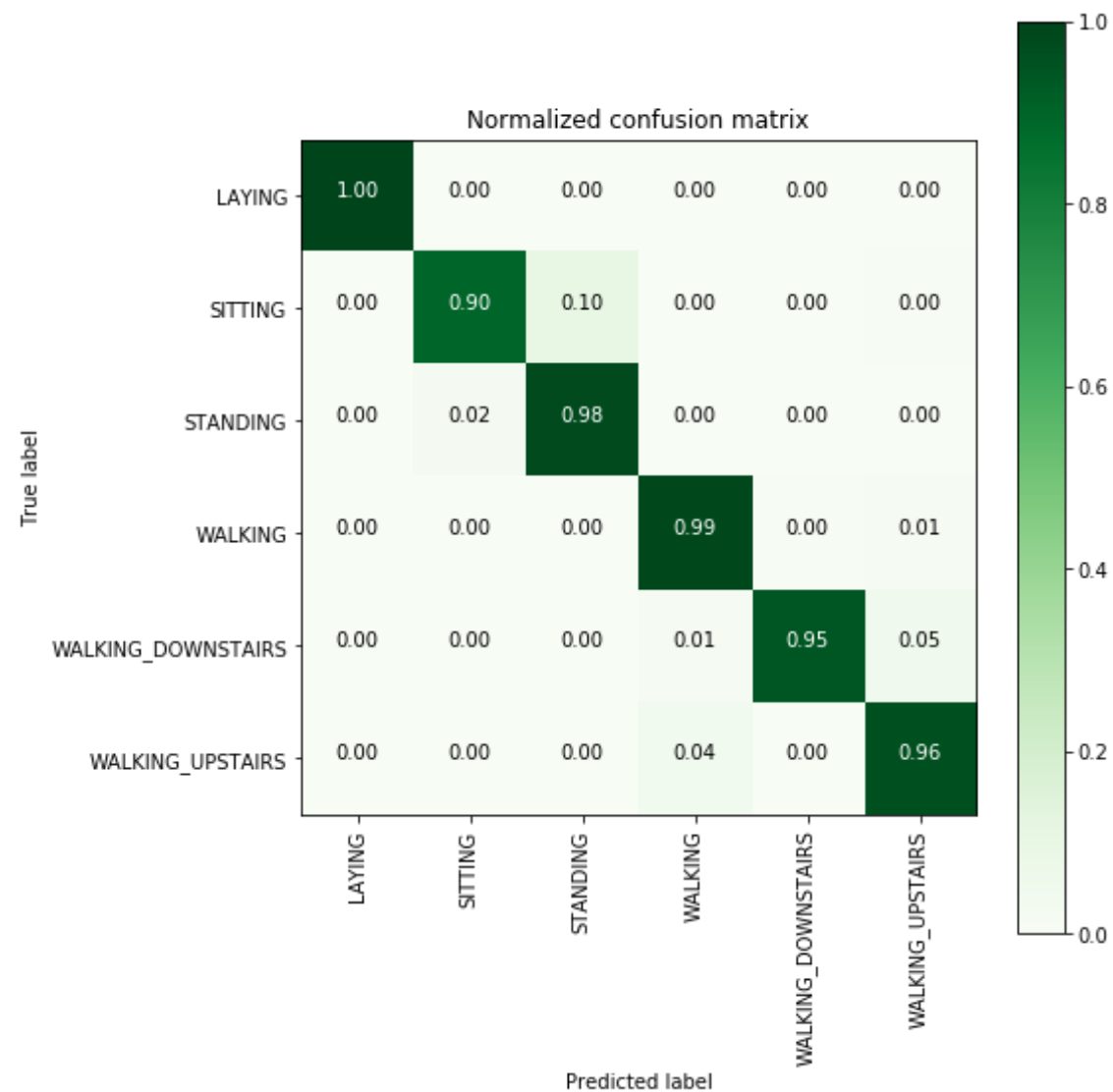
```
-----  
|           Accuracy           |  
-----
```

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]]
```





Classification 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
avg / total	0.96	0.96	0.96	2947

```
In [19]: 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='rb
f',
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 numbere of cross validation sets: 3

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

Average Cross Validate scores of best estimator :
```

0.9440968443960827

## 4. Decision Trees with GridSearchCV

```
In [20]: 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_labels=labels)
print_grid_search_attributes(dt_grid_results['model'])
```

training the model..  
Done

training\_time(HH:MM:SS.ms) - 0:00:19.476858

Predicting test data  
Done

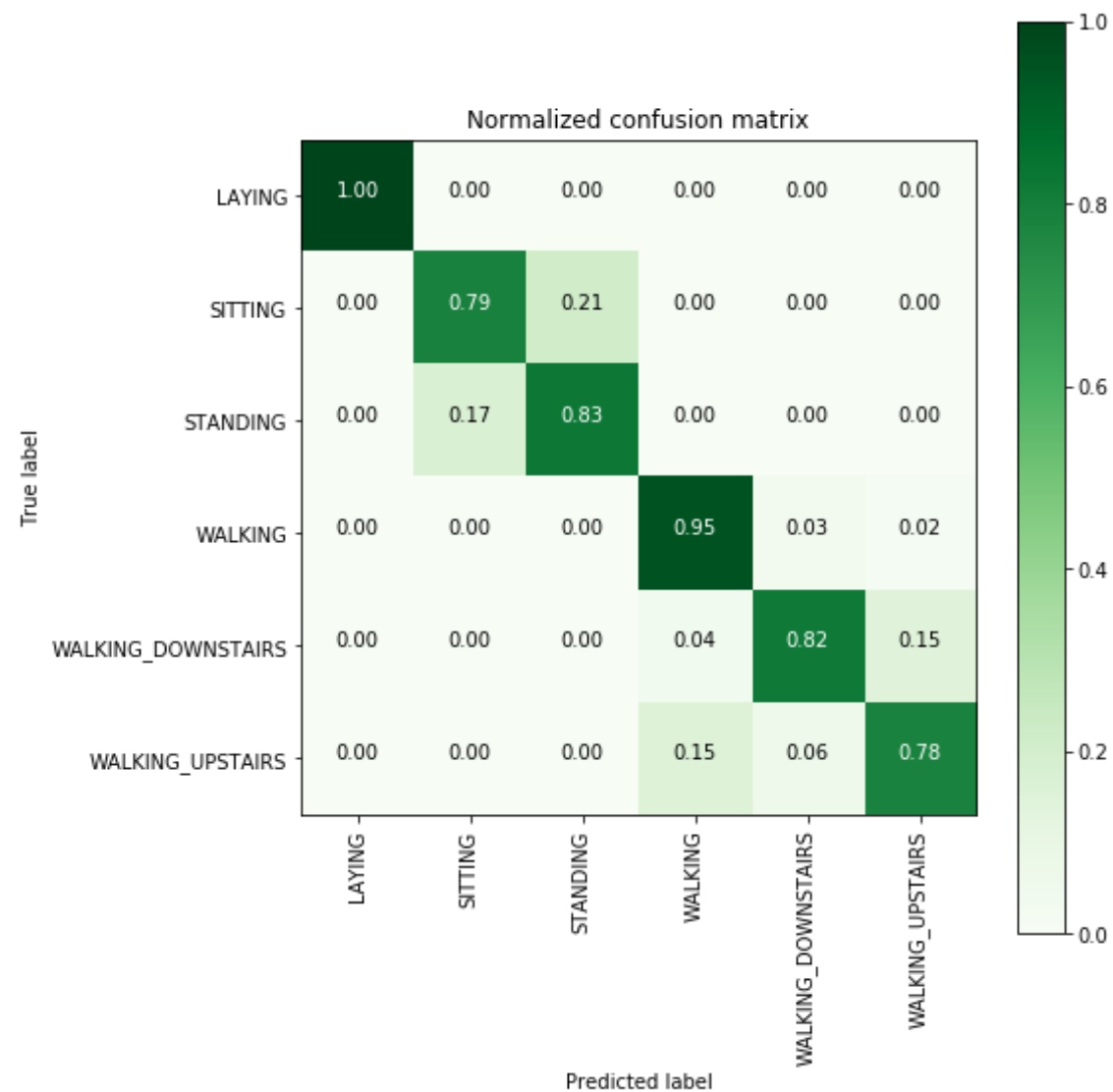
testing time(HH:MM:SS.ms) - 0:00:00.012858

```
-----
|      Accuracy      |
-----
```

0.8642687478791992

```
-----
| 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 15 344 61]  
[  0  0  0 73 29 369]]
```



Classification 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	0.84	0.95	0.89	496
WALKING_DOWNSTAIRS	0.88	0.82	0.85	420
WALKING_UPSTAIRS	0.84	0.78	0.81	471
avg / total	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=N
one,
                        splitter='best')
```

```
-----
| Best parameters |
-----
```

Parameters of best estimator :

```
{'max_depth': 7}
```

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

Total numbere of cross validation sets: 3

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

Average Cross Validate scores of best estimator :

0.8369151251360174

## 5. Random Forest Classifier with GridSearch

```
In [21]: 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_labels=labels)
print_grid_search_attributes(rfc_grid_results['model'])
```

training the model..  
Done

training\_time(HH:MM:SS.ms) - 0:06:22.775270

Predicting test data  
Done

testing time(HH:MM:SS.ms) - 0:00:00.025937

-----  
Accuracy

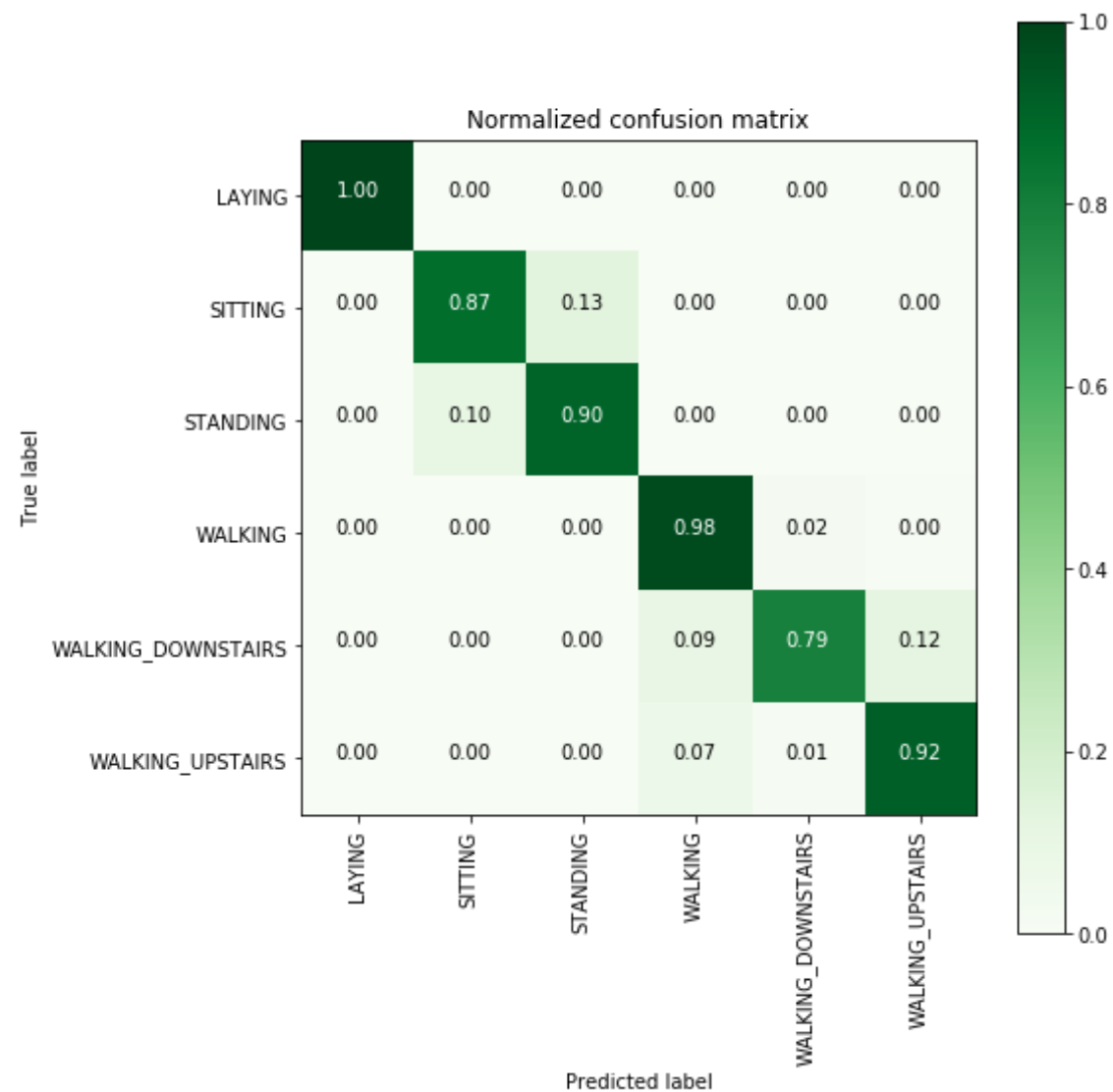
0.9131319986426875

-----

# | Confusion Matrix |

```
-----  
[[537  0  0  0  0  0]  
[  0 427 64  0  0  0]  
[  0 52 480  0  0  0]  
[  0  0  0 484 10  2]  
[  0  0  0 38 332 50]  
[  0  0  0 34  6 431]]
```





Classification Report

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	537

SITTING	0.89	0.87	0.88	491
STANDING	0.88	0.90	0.89	532
WALKING	0.87	0.98	0.92	496
WALKING_DOWNSTAIRS	0.95	0.79	0.86	420
WALKING_UPSTAIRS	0.89	0.92	0.90	471
avg / total	0.92	0.91	0.91	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=70, n_jobs=1,
                        oob_score=False, random_state=None, verbose=0,
                        warm_start=False)
```

```
-----
| Best parameters |
-----
```

Parameters of best estimator :

```
{'max_depth': 7, 'n_estimators': 70}
```

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

Total number of cross validation sets: 3

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

Average Cross Validate scores of best estimator :

0.9141730141458106

## 6. Gradient Boosted Decision Trees With GridSearch

```
In [22]: 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_labels=labels)
print_grid_search_attributes(gbdt_grid_results['model'])
```

training the model..  
Done

training\_time(HH:MM:SS.ms) - 0:28:03.653432

Predicting test data  
Done

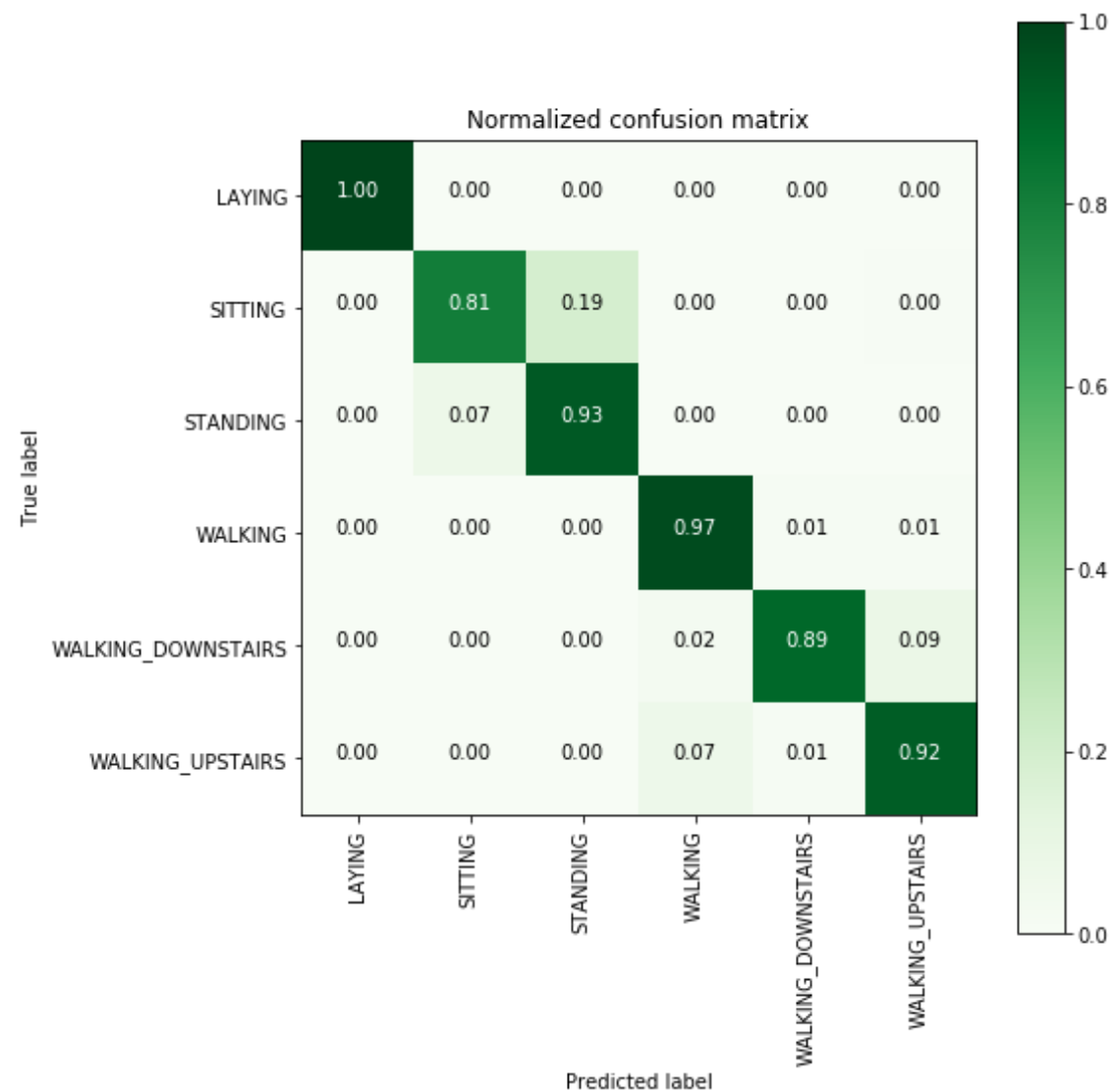
testing time(HH:MM:SS.ms) - 0:00:00.058843

-----  
Accuracy

0.9222938581608415

-----  
Confusion Matrix

```
[[537    0    0    0    0    0]
 [   0 396  93    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]]
```



Classification Report

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	537
SITTING	0.81	0.81	0.81	421
STANDING	0.93	0.93	0.93	421
WALKING	0.97	0.97	0.97	421
WALKING_DOWNSTAIRS	0.89	0.89	0.89	421
WALKING_UPSTAIRS	0.92	0.92	0.92	421

SITTING	0.91	0.81	0.86	491
STANDING	0.84	0.93	0.88	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
avg / total	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=140,
    presort='auto', random_state=None, subsample=1.0, verbose
=0,
    warm_start=False)
```

```
-----
| Best parameters |
-----
```

Parameters of best estimator :

```
{'max_depth': 5, 'n_estimators': 140}
```

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

Total numbere of cross validation sets: 3

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

Average Cross Validate scores of best estimator :

0.904379760609358

## 7. Comparing all models

```
In [23]: print('\n
print('
print('-----'
print('-----')
print('Logistic Regression : {:.04}%
_results['accuracy'] * 100,\
100-(log_reg_grid_res
ults['accuracy'] * 100)))

print('Linear SVC : {:.04}%
_results['accuracy'] * 100,\
100-(lr_svc_gri
d_results['accuracy'] * 100)))

print('rbf SVM classifier : {:.04}%
_results['accuracy'] * 100,\
100-(rbf_svm_
grid_results['accuracy'] * 100)))

print('DecisionTree : {:.04}%
lts['accuracy'] * 100,\
100-(dt_grid_re
sults['accuracy'] * 100)))

print('Random Forest : {:.04}%
ults['accuracy'] * 100,\
100-(rfc_gri
d_results['accuracy'] * 100)))
print('GradientBoosting DT : {:.04}%
ults['accuracy'] * 100,\
100-(rfc_grid_r
esults['accuracy'] * 100)))
```

	Accuracy	Error
	-----	-----
Logistic Regression :	96.27%	3.733%
Linear SVC :	96.61%	3.393%
rbf SVM classifier :	96.27%	3.733%
DecisionTree :	86.43%	13.57%
Random Forest :	91.31%	8.687%
GradientBoosting DT :	91.31%	8.687%

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

## Conclusion :

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





