modelling_activity_final

July 18, 2019

In [0]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")



import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

import math as m
```

In [2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-

Enter your authorization code:
ůůůůůůůůůů
Mounted at /content/gdrive
```

In [0]:
```
dataset_url = '/content/gdrive/My Drive/case_studies/UT_Data_Complex/new_df.csv'
data = pd.read_csv(dataset_url)
```

In [4]:
```
data.head()
```

Out[4]:
```
   Unnamed: 0     acc_X   acc_Y  ...    gjerky    gjerkz   mag_gjerk
0           0 -5.924900 -10.978  ... -115.4855 -119.7770  169.792790
1           1 -6.960000 -12.136  ...  -81.6415  -68.4160  117.277229
2           2 -3.963500 -15.568  ...  -23.1500  -98.4860  108.150660
3           3 -0.054481 -15.677  ...   16.9050  -92.4715   94.004156
4           4  0.354130 -13.048  ...  -77.1685  -53.4515  100.718867

[5 rows x 35 columns]
```

In [0]:
```
data.drop('Unnamed: 0', 1, inplace=True)
```

```
In [6]: data.head(2)

Out[6]:     acc_X   acc_Y    acc_Z  linearacc_X  ...  gjerkx    gjerky   gjerkz   mag_gjerk
        0 -5.9249 -10.978  1.00790      -5.3538  ...  33.855 -115.4855 -119.777  169.792790
        1 -6.9600 -12.136  0.28603      -5.4353  ...  49.070  -81.6415  -68.416  117.277229

        [2 rows x 34 columns]
```

0.1 Train Test Split

```
In [0]: y = data['labels'].values
        X = data.drop(['Activity_Name', 'labels'], 1)

In [10]: X.head()

Out[10]:       acc_X   acc_Y    acc_Z  ...    gjerky    gjerkz   mag_gjerk
         0 -5.924900 -10.978  1.00790  ... -115.4855 -119.7770  169.792790
         1 -6.960000 -12.136  0.28603  ...  -81.6415  -68.4160  117.277229
         2 -3.963500 -15.568 -3.37780  ...  -23.1500  -98.4860  108.150660
         3 -0.054481 -15.677 -4.44020  ...   16.9050  -92.4715   94.004156
         4  0.354130 -13.048 -2.57420  ...  -77.1685  -53.4515  100.718867

         [5 rows x 32 columns]

In [7]: len(data)

Out[7]: 1169999

In [0]: XT_train, XT_test, yT_train, yT_test = train_test_split(X, y, test_size=0.1, stratify=y

In [15]: print('X_train and y_train : ({},{})'.format(XT_train.shape, yT_train.shape))
         print('X_test  and y_test  : ({},{})'.format(XT_test.shape, yT_test.shape))

X_train and y_train : ((1052999, 32),(1052999,))
X_test  and y_test  : ((117000, 32),(117000,))


In [0]: # please write all the code with proper documentation, and proper titles for each subs
        # go through documentations and blogs before you start coding
        # first figure out what to do, and then think about how to do.
        # reading and understanding error messages will be very much helpfull in debugging you
        # when you plot any graph make sure you use
            # a. Title, that describes your plot, this will be very helpful to the reader
            # b. Legends if needed
            # c. X-axis label
            # d. Y-axis label


        # train test split
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(XT_test, yT_test, test_size=0.33, s
        #X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, str
```

```
In [19]: 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 : ((78390, 32),(78390,))
X_test  and y_test  : ((38610, 32),(38610,))
```

1 Modelling

```
In [0]: #X_train['Activity_Name'].value_counts()
```

```
In [0]: #labels=['walk', 'stand','jog','sit','bike', 'upstairs', 'downstairs', 'type', 'write'
```

```
In [0]: labels = data['labels'].unique()
```

```
In [29]: labels
```

```
Out[29]: array([11111, 11112, 11113, 11114, 11115, 11116, 11117, 11118, 11119,
                11120, 11121, 11122, 11123])
```

1.1 Function to print Confusion Matrix

```
In [0]: 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')
```

1.1.1 Generic function to run any model specified

```
In [0]: from datetime import datetime
        def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_normalize=T
                         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
```

In [0]:
```
#labels = data['Activity_Name'].unique()
```

In [24]:
```
data.columns
```

Out[24]:
```
Index(['acc_X', 'acc_Y', 'acc_Z', 'linearacc_X', 'linearacc_Y', 'linearacc_Z',
       'gyro_X', 'gyro_Y', 'gyro_Z', 'mag_X', 'mag_Y', 'mag_Z', 'labels',
       'Activity_Name', 'velx', 'vely', 'velz', 'distx', 'disty', 'distz',
       'mag_vel', 'mag_dist', 'mag_acc', 'mag_lacc', 'mag_gyro', 'mag_magnet',
       'jerkx', 'jerky', 'jerkz', 'mag_jerk', 'gjerkx', 'gjerky', 'gjerkz',
       'mag_gjerk'],
      dtype='object')
```

1.1.2 Method to print the gridsearch Attributes

In [0]:
```
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(moc
```

2 Random Forests

In [0]:
```
from sklearn import linear_model
from sklearn import metrics

from sklearn.model_selection import GridSearchCV
```

In [0]:
```
feature_names = X_train.columns
```

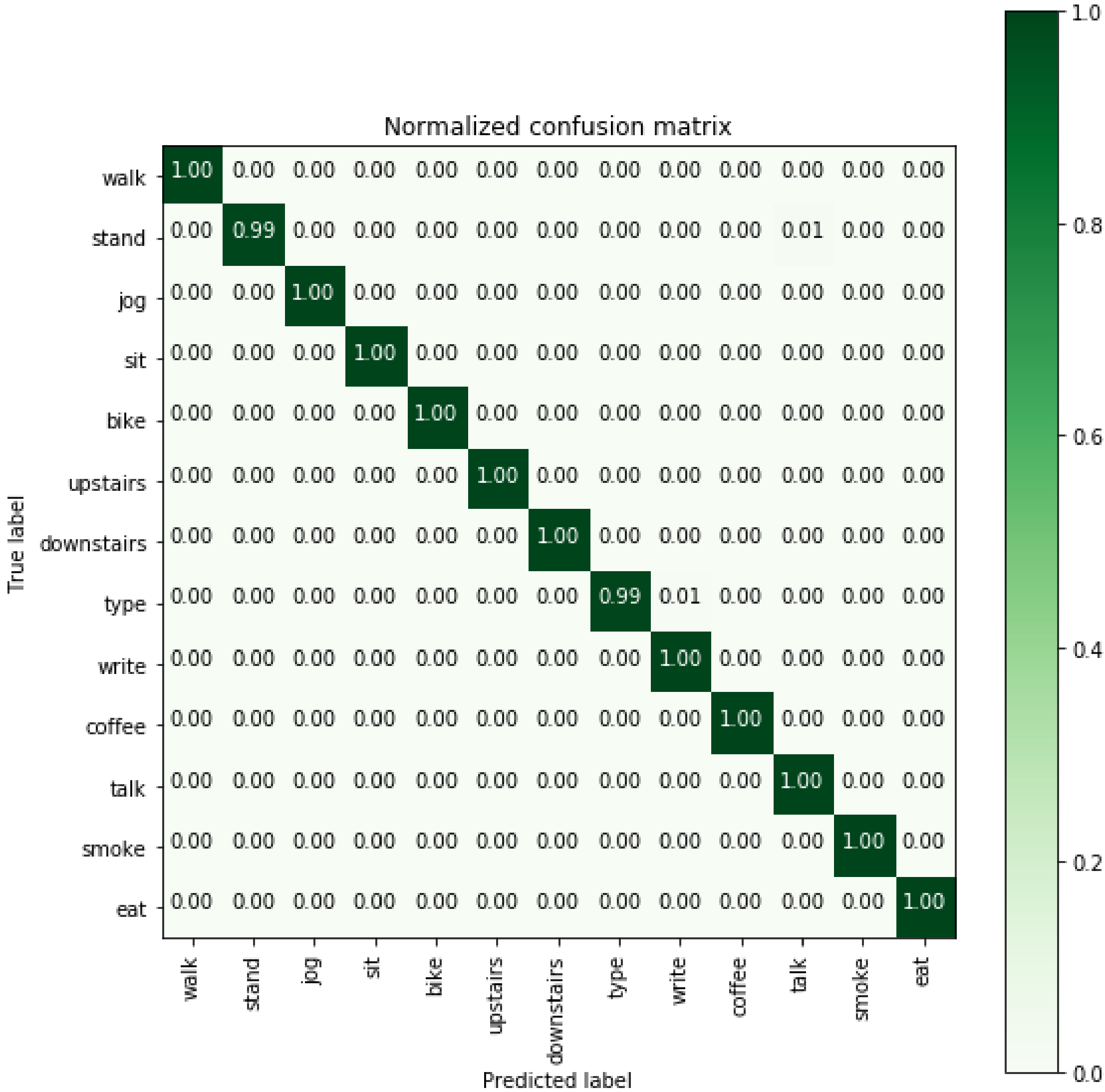
In [0]:
```
labels = data['Activity_Name'].unique()
```

In [38]:
```
labels
```

Out[38]:
```
array(['walk', 'stand', 'jog', 'sit', 'bike', 'upstairs', 'downstairs',
       'type', 'write', 'coffee', 'talk', 'smoke', 'eat'], dtype=object)
```

In [39]:
```
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..
Done


training_time(HH:MM:SS.ms) - 0:37:39.971188


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:01.371660


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

    0.9977466977466978


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

 [[2969    0    1    0    0    0    0    0    0    0    0    0    0]
 [   0 2942    0    0    0    0    0    0    0    0   28    0    0]
 [   0    0 2970    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0 2968    0    0    0    0    0    2    0    0    0]
 [   0    0    0    0 2967    3    0    0    0    0    0    0    0]
 [   0    0    0    0    9 2960    1    0    0    0    0    0    0]
 [   0    0    0    0    2    0 2968    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0 2948   18    4    0    0    0]
 [   0    0    0    0    0    0    0    7 2963    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0 2969    0    0    1]
 [   0   10    0    0    0    0    0    0    0    0 2960    0    0]
 [   0    0    1    0    0    0    0    0    0    0    0 2969    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0 2970]]
```

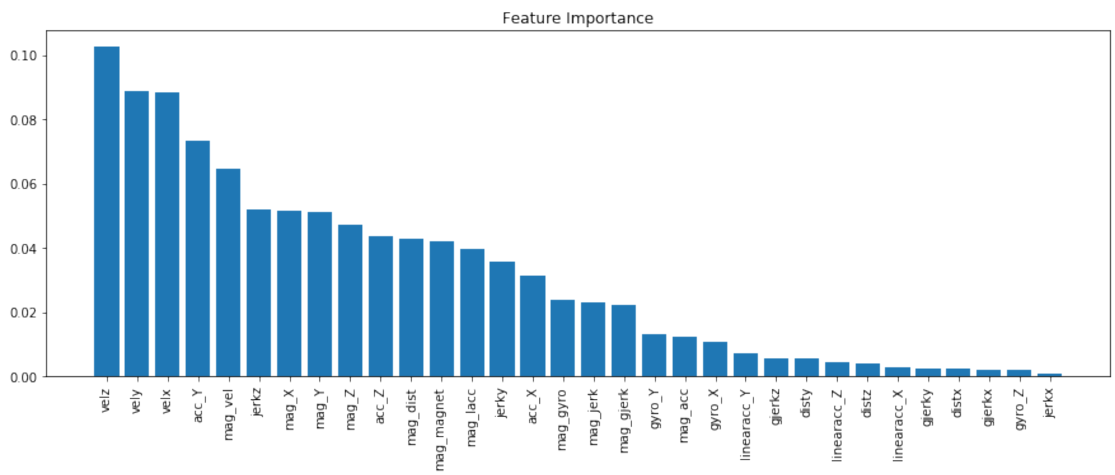



```
-------------------------
| Classifiction Report |
-------------------------
              precision    recall  f1-score   support

       11111       1.00      1.00      1.00      2970
       11112       1.00      0.99      0.99      2970
       11113       1.00      1.00      1.00      2970
       11114       1.00      1.00      1.00      2970
       11115       1.00      1.00      1.00      2970
       11116       1.00      1.00      1.00      2970
       11117       1.00      1.00      1.00      2970
       11118       1.00      0.99      1.00      2970
```

```
       11119       0.99      1.00      1.00      2970
       11120       1.00      1.00      1.00      2970
       11121       0.99      1.00      0.99      2970
       11122       1.00      1.00      1.00      2970
       11123       1.00      1.00      1.00      2970

    accuracy                           1.00     38610
   macro avg       1.00      1.00      1.00     38610
weighted avg       1.00      1.00      1.00     38610

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

        RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=13, 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=170,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

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

        {'max_depth': 13, 'n_estimators': 170}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.997767572394438
```

In [40]:
```
clf = RandomForestClassifier(n_estimators = 170, max_depth = 13, random_state=0, n_jol
model = clf.fit(X_train, y_train)
importances = model.feature_importances_
```

```
indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [feature_names[i] for i in indices]

# Create plot
plt.figure(figsize=(15,5))

# Create plot title
plt.title("Feature Importance")

# Add bars
plt.bar(range(X_train.shape[1]), importances[indices])

# Add feature names as x-axis labels
plt.xticks(range(X.shape[1]), names, rotation=90)

# Show plot
plt.show()
```



3 Conclusions

```
In [46]: # Please compare all your models using Prettytable library

         # Please compare all your models using Prettytable library

         from prettytable import PrettyTable
```

```
x = PrettyTable()
x.field_names = ["Model", "HyperParameter_1(n_estimators)", "HyperParameter_2(max_dep

x.add_row(["RandomForest", 170, 13, 0.9977])



print(x)
```

```
+--------------+--------------------------------+------------------------------+----------+
|    Model     | HyperParameter_1(n_estimators) | HyperParameter_2(max_depth) | Accuracy |
+--------------+--------------------------------+------------------------------+----------+
| RandomForest |              170               |              13              |  0.9977  |
+--------------+--------------------------------+------------------------------+----------+
```

In [47]:
```
len(data)
```

Out[47]: 1169999

3.1 How i approached this problem ?

- there are 1169999 data points in the dataset
- 10 healthy participants aged 25-35 - Seven activities were performed by all ten participants which are walking, jogging, biking, walking upstairs, walking downstairs, sitting and standing.
- These activities were performed for 3 min by each participant
- Seven out of these ten participants performed eating, typing, writing, drinking coffee and giving a talk.
- These activities were performed for 5–6 min. Smoking was performed by six out of these ten participants, where each of them smoked one cigarette. Only six participants were smokers among the ten participants.
- We used 30 min of data for each activity with an equal amount of data from each participant. This resulted in a dataset of 390 (13 Œ 30) min.
- so according to the above data, i divided 1169999 datapoints by 390 minutes which gives 0.02 seconds. so this is the calculated time which we have used in featurization.
- i performed eda, then featurization and the modelling as shown in the ipynb.