Machine Learning Engineer Nanodegree

Capstone Project

BOUCHKARA Abdelilah

July 26th, 2018

I- Definition:

I-1. Project Overview:

Problem Domain:

Since 1870 a large growth in human life expectancy has been observed in Europe. This growth has been expanded in the whole world [1] principally due to the great achievements in health care field. As a result, the proportion of elderly people is rapidly increasing. Aging people in general lives in isolated conditions. In addition to that some of them are not capable to live normally and take advantages from health care facilities services. Building remote monitoring systems for elderly patients who live alone or without permanent caretaking will improve their quality of life. For better decision making these remote monitoring systems require regular and trustful information about patients.

Project Origins:

To fulfil remote monitoring systems' requirements **Jorge Luis Reyes Ortiz** [2] has developed a complete Human Activity Recognition (HAR [3]) System able to detect and recognize 12 different activities performed by humans in their daily living using smartphones. The recognition part of this system is an SVM [4] model already trained, capable of predicting activities performed by users. Necessary datasets of users' movements were collected from smartphone sensors (accelerometer [5] and gyroscope [6]). they will be processed and fed to the prediction model to recognize performed activities. (**see additional report for more information**).

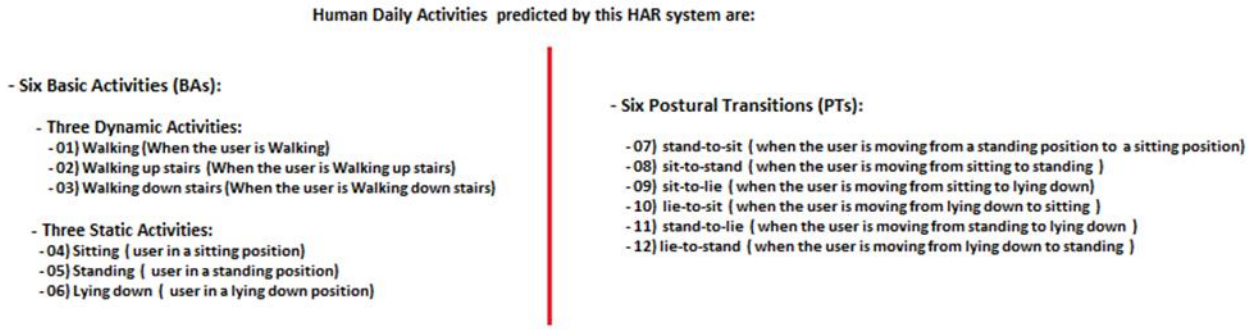


Figure: Daily activities performed by volunteers

Dataset and Inputs [License]:

In order to achieve high performance in predictions with this model, a data collection was done in collaboration with 30 volunteers were asked to perform a protocol of activities. All participants were wearing a smartphone on the waist during the experiments' execution. During each experiment **3-axial acceleration signals and 3-axial angular velocities signals** were captured using smartphone sensors **(see additional report for more info).** These raw inertial signals were stored in **RawData** (folder).

This folder includes 61 experiments each experiment has two logfiles:

- **'RawData/acc_expXX_userYY.txt':** The raw triaxial acceleration signal for the experiment number XX and associated to the user number YY. Every row is one acceleration sample (three axis) captured at a frequency of 50Hz. Units used for acceleration are 'g' s (gravity of earth =9.80665 m/sec^2)
- **'RawData/gyro_expXX_userYY.txt'**: The raw triaxial angular speed signal for the experiment number XX and associated to the user number YY. Every row is one angular velocity sample (three axis) captured at a frequency of 50Hz. Units used for angular velocities are radian per second (rad/s)

This folder includes also 1 additional file: labels file:

- **'RawData/labels.txt':** includes all activity labels available for the dataset (1 activity label per row). Each row contains the start-end row numbers related a group of log samples located in acc and gyro files. These files can be identified using the **exp Id** and the **user Id** existing in that row.

The goal of this capstone project is to build a machine learning model and a signal processing pipeline capable of processing signals collected using smart phone inertial sensors (accelerometer and gyroscope) stored in **RawData Folder** and producing useful datasets which will be used as inputs of a machine learning model capable of recognizing some of human daily activities **[figure above]** included in the dataset with a low error rate. The signal processing pipeline and the final model could be used in offline mode as a good source of information about patient's daily activities required by remote monitoring systems mentioned earlier.

I-2. Problem Statement:

Raw Inertial Signals mentioned above cannot be fed directly to machine learning models. A signal processing pipeline should be built to filter noise, extract useful signals. split them into windows. From each window a vector of features will be generated to obtain classical datasets **(Datasets 1and 2 figure below)**.

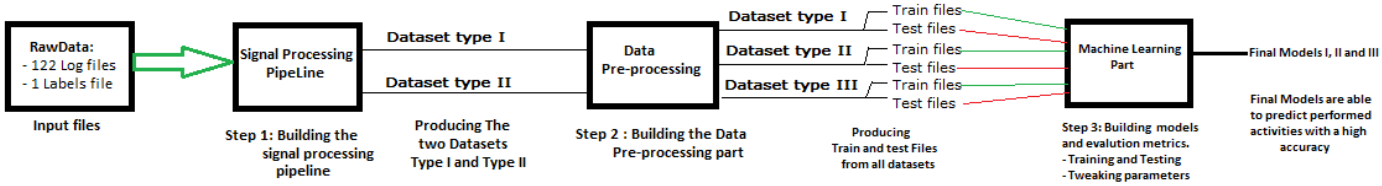


Figure: The full process

After exploring **raw inertial signals**, I will define a **3rd order median filter [7]** to reduce noise. Then a **Fast Fourier Transform [8]** will be applied to extract useful frequency components: **body components and gravity components.** An Inverse Fast Fourier Transform will be applied to useful components to obtain them in time domain. All Body components will be derived in time domain to obtain **jerk signals**. **The magnitude** of each triaxial signals will be generated using the **Euclidian Norm**.

All useful signals will be sampled in fixed-width sliding windows of 2.56 sec (128 readings/window) using two methods. The first windowing method concerns only signal captures related to basic activities (activity id from 1 to 6). This method assures that all rows in a window will have the same **activity id** which could be considered as the **window activity id (windows type I).** The second one which is more realistic it concerns all captures in **RawData (activity ids from 1 to 12).** Actually, this windowing method doesn't care if all rows in a window have even an activity id or the same activity id as a result, a **voting function** should be created to define the activity id of each window **(Windows type II).**

A Fast Fourier Transform will be applied to some columns of each window, to obtain frequency windows. From each tuple of time and frequency windows, a vector of features will be generated by calculating variables from the time and frequency domain signals. Each features vector is a row in the final dataset.

Each row of features obtained from time and frequency windows having the same windowing type will be arranged in one dataset. By the end of this signal processing pipeline two datasets were obtained.

Since all features from both datasets are real numbers a feature scaling is needed to standardize all columns. Rows considered as Outliers will be detected using a threshold which is an integer number: the number of features considered as outliers in a row. Outlier rows will be deleted before generating the train and test files from both datasets. Features scaling and outliers' detection will be discussed in detail in next sections. Both datasets will be randomly partitioned into two sets, where 70% of the volunteers were selected for generating the training data and 30% the test data.

Target columns of both datasets contain activity labels associated to each vector of features. Recognizing the activity associated to each vector is a multiclass classification problem. To solve it, I intend to use some supervised learning classifiers (since each vector has an activity label) and compare predictions.

To train models, I plan to choose 3 models one as benchmark model, the two others will be used to beat the benchmark results. For this problem, the benchmark model will be **gaussian Naive Bayes Classifier (GaussianNB**). I will try beat its performance with **logistic regression and decision tree classifiers**

Using a simplified cross-validation I can find which model performs the best and then use that one, tweak its relative parameters to get best accuracy.

To respect the project report template and the size proposed 15 pages, **RawData logfiles** and **Signal Processing steps** were explored and explained in details in **the additional report.** This capstone report concerns only **Dataset type I and II** obtained from the signal processing pipeline, it includes also **Data processing and Machine learning steps** followed to obtain the final model **(figure above).**

I-3. Metrics:

To Evaluate models' performances, I will use the Accuracy as a general evaluation metric which is a real number between 0 and 1:

$$\boldsymbol{Accuracy(model) = \frac{number\ of\ samples\ correctly\ predicted}{total\ number\ of\ samples}}$$

In order to have some detailed information about each model performance and types of errors I will use the confusion matrix adapted to multiclass problems as secondary evaluation metric (each activity type is considered as one class):

The actual class (***class_i***) versus all other classes ***class_j*** where ***j*** is different from ***i*** :

$\boldsymbol{True\ Positives}\ (\boldsymbol{TP_i})$: $\boldsymbol{number\ of}$ samples of a $\boldsymbol{class_i}$ correctly predicted as samples of the same $\boldsymbol{class_i}$.

$\boldsymbol{False\ Negatives}\ (\boldsymbol{FN_i})$: $\boldsymbol{number\ of}$ samples of a $\boldsymbol{class_i}$ incorrectly predicted as a samples of another $\boldsymbol{class_j}$

$\boldsymbol{False\ Positives}\ (\boldsymbol{FP_i})$: $\boldsymbol{number\ of}$ samples of other $\boldsymbol{classes_j}$ incorrectly predicted as samples of $\boldsymbol{class_i}$.

True negatives (TN): have an ambiguous meaning since we are going to deal all other classes.

$\boldsymbol{True\ negatives}(\boldsymbol{TN_i})$: $\boldsymbol{number\ of}$ samples belongs to other $classes_j$ not predicted as samples of the actual $class_i$

From these quantities mentioned above other metrics will be generated to evaluate model's performances on each class (activity type).

$\boldsymbol{i}$: an integer number between 1 and 6 for dataset type I and from 1 to 12 for dataset type II

Sensitivity (class i): it measures model's ability to correctly predict samples of a class i.

$$\boldsymbol{Sensitivity\ (model\ , class_i)\ = \frac{TP_i}{(TP_i\ +\ FN_i)}}$$

Specificity: it measures model's ability to reduce predictions errors of a class i samples compared to all other classes samples:

$$\boldsymbol{Specificity(model, class_i)} = \frac{\boldsymbol{TN_i}}{(\boldsymbol{TN_i} + \boldsymbol{FP_i})}$$

Precision: it measures model's ability to reduce predictions errors of other classes j's samples predicted as samples of class i.

$$\boldsymbol{Precision\ (model, class_i)} = \frac{\boldsymbol{TP_i}}{(\boldsymbol{TP_i} + \boldsymbol{FP_i})}$$

II- Analysis:

II-1. Data Exploration:

RawData Exploration:

RawData logfiles were explored in details see the additional report. Datasets explored below are the outputs of the signal processing pipeline.

Dataset type I and type II Exploration:

- **Dataset type I** includes 10399 rows and 642 columns**, no missing values, no categorical features.**
- **Dataset type II** includes 12637 rows and 642 columns, **no missing values, no categorical features.**

Both Datasets have the same features names obtained using the same functions.

The last two columns in both datasets include the activity and the user labels related to each row of features.

The first 640 columns are features obtained by calculating variables from time and frequency signals from each tuple of windows mentioned in the problematic.

Activity labels of the first dataset concerns basic activities only (activity id from 1 to 6). The second dataset concerns all 12 different activities.

Data Set type I

	t_body_acc_mean()_X	t_body_acc_mean()_Y	t_body_acc_mean()_Z
0	0.002012	0.000431	0.004441
1	-0.000713	-0.003098	0.000823
2	-0.000301	0.004025	-0.004280

3 rows × 642 columns

The first 3-rows , 3 features of both datasets

Dataset type II

	t_body_acc_mean()_X	t_body_acc_mean()_Y	t_body_acc_mean()_Z
0	0.002012	0.000431	0.004441
1	-0.000713	-0.003098	0.000823
2	-0.000301	0.004025	-0.004280

3 rows × 642 columns

	t_body_acc_mean()_X	t_body_acc_mean()_Y	t_body_acc_mean()_Z
500	-0.002612	0.001936	-0.001619
501	-0.004327	0.000869	-0.001459
502	0.000866	-0.002031	-0.002663

3 rows × 642 columns

The first 3 features of rows 500 to 502

	t_body_acc_mean()_X	t_body_acc_mean()_Y	t_body_acc_mean()_Z
500	-0.002838	0.003938	0.001229
501	0.000645	-0.000686	0.001023
502	0.002999	-0.003556	-0.002519

3 rows × 642 columns

	t_body_acc_mean()_X	t_body_acc_mean()_Y	t_body_acc_mean()_Z
count	10399.000000	10399.000000	10399.000000
mean	0.000181	-0.000253	0.000043
std	0.009100	0.006948	0.006990
min	-0.047488	-0.038424	-0.047545
25%	-0.003148	-0.003659	-0.003376
50%	0.000080	-0.000127	-0.000052
75%	0.003343	0.003268	0.003301
max	0.046679	0.036071	0.045893

Statistics of These 3 features for bothdatasets

	t_body_acc_mean()_X	t_body_acc_mean()_Y	t_body_acc_mean()_Z
count	12637.000000	12637.000000	12637.000000
mean	-0.000021	-0.000012	0.000025
std	0.009201	0.007817	0.007812
min	-0.046247	-0.053277	-0.042451
25%	-0.003742	-0.003890	-0.003715
50%	-0.000032	0.000007	-0.000092
75%	0.003711	0.003810	0.003636
max	0.045010	0.059873	0.054561

Figure: Data Exploration Results

From the first 2 tables above, it appears that both datasets have same values which means the first activity performed is common. The second two tables concern rows indexes 500,501 and 502 of both datasets. Here it appears the difference between Datasets. The third two tables show statistics of the first 3 features in both datasets. The min value of each column is negative. The difference between the third quartile and the maximum value of each column in both datasets is huge enough compared to the mean of each column. From this fast analysis two conclusions were made:

-All column values are real number they should be scaled to have values between -1 and 1

-The presence of top outliers is confirmed due to the huge difference between the 3rd quartile and the max value compared to the mean of each column in both datasets.

Features meaning: the first 3 features presented above are the mean of 3 axial signals [X, Y, Z]. features names in this case are:

[t_body_acc_mean()_X , t_body_acc_mean()_Y , t_body_acc_mean()_Z]

t: denotes time domain which means these signals belong the time domain.

t_Body_acc_X: is a column in the **original time domain window** related to this row of features. Same thing for the two other components.

Mean(): the mean function is applied to ***t_Body_acc_X*** column to obtain the mean value.

For more information about all features and signal names used to build these datasets see the additional report.

II.2. Exploratory Visualization:

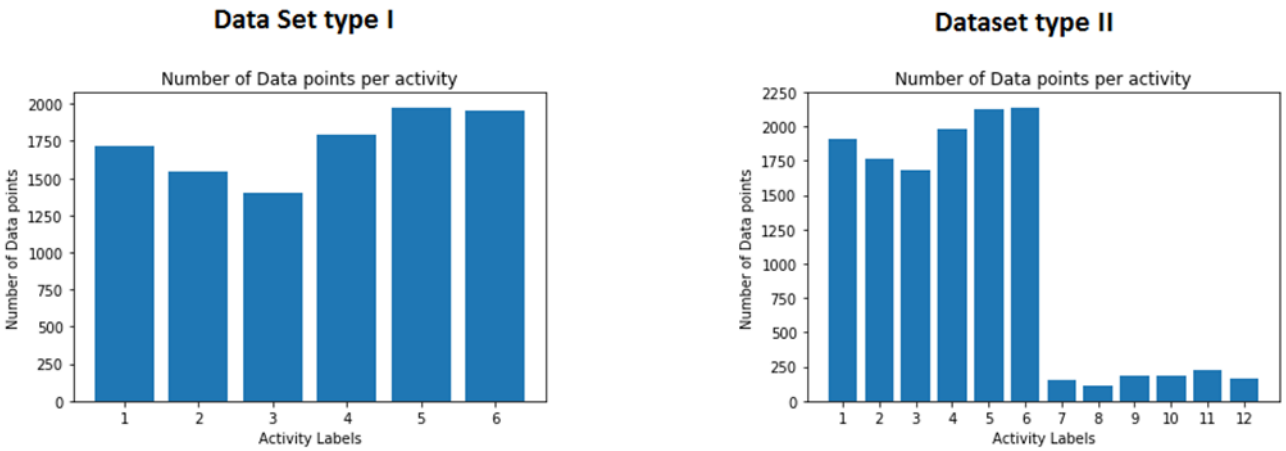


Figure: Activity Distribution for Both Datasets

The two figures above show the number of datapoints (rows) per activity for both datasets.

Dataset type I is little imbalanced no huge difference between bar's tops. All weights values are near to each other (figure below).

Dataset type II which concerns all 12 activities is heavily imbalanced.

Postural transitions (activity ids from 7 to 12) have low data points number less than 250 datapoints per each activity id. All postural transitions' datapoints represent less than 9% of the total size of this dataset. The dataset Distribution is approximately uniform considering only activities from 7 to 12 **(see weights table below).**

Basic Activities (activity ids from 1 to 6) have the majority of data points in this dataset more 91% of the total size. Basic activities dataset distribution is approximately uniform its weights are approximately near to each other **(figure below).**

Weights of each activity in both dataset

	1	2	3	4	5	6
Weights	0.165401	0.148283	0.135205	0.172805	0.190114	0.188191

	1	2	3	4	5	6
Weights	0.151223	0.139432	0.133418	0.156683	0.168632	0.169344

	7	8	9	10	11	12
Weights	0.012424	0.008863	0.014956	0.014323	0.017805	0.012899

Figure: Weights of each Activity in both Datasets

Tables above shows clearly activity distribution of each dataset. Heavily imbalanced datasets (dataset type II) may lead to errors in predictions if we don't adapt the machine learning model to this case or reshape this dataset to be approximately balanced before the training phase.

II.3. Algorithms and Techniques:

Since all features in both datasets have a target label the best machine learning approach to be applied is **Supervised Learning**. In this type of learning, data inputs are usually composed of a pair of elements, namely the input vector (features) together with its target (activity labels in this case)

Supervised Learning approach has variety of models to be used. In this case we are interested in classification models since the target variable takes output classes: **activity ids from 1 to 6 in dataset type I** or **from 1 to 12 in the second dataset.**

Classification models chosen in this study are:

Gaussian Naïve Bayes (gaussian NB) [9]: it belongs to the family of probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Decision Tree classifier [10]: is a predictive model based on decision trees which makes choices from a set of hierarchical rules related to the input data. It is a common approach for classification particularly because the resulting models are easily interpretable by humans.

Logistic Regression classifier [11]: is a probabilistic algorithm used for solving classification problems. It estimates the probability of a given sample belonging to a particular class. This algorithm uses the logistic function which is modelled by fitting the training data generally using maximum likelihood estimation.

GridSearchcv: A function that makes an Exhaustive search over specified parameter values for a model. It returns the parameters' combination having the best score depending on evaluation metric used.

II.4. Benchmark Model:

After applying data processing steps will be mentioned in **methodology section**, 3 types of datasets were obtained.

The Train test pipeline function will be applied to **gaussianNB ()** to report benchmark results.

This model was chosen as benchmark due to its simplicity. It assumes all columns are independent from each other.

Table: Benchmark accuracies on each dataset

	Dataset Type I	Dataset Type II	Dataset Type III
Accuracy in %	**72.080%**	**65.464%**	**71.210%**

III- Methodology:

III.0. Signal processing:

All signal processing steps and its implementations are explained in details in the additional report

III.1. Data Pre-processing:

Outliers Detection:

Assuming that rows having the same activity ids should have approximately same values per feature(theoretically). As a result outliers detection should be done separately per each group of rows having the same activity label.

First, we group all feature rows having the same activity id.

For each column in that group we define:

$$\boldsymbol{Low\ threshold(column)} = \boldsymbol{Q1(column)} - \boldsymbol{1.5} * (\boldsymbol{Q3(column)} - \boldsymbol{Q1(column)})$$

$$\boldsymbol{High\ threshold(column)} = \boldsymbol{Q3(column)} + \boldsymbol{1.5} * (\boldsymbol{Q3(column)} - \boldsymbol{Q1(column)})$$

$\boldsymbol{Q1(column)}$: $\boldsymbol{is\ the\ value\ of\ the\ first\ quartile}$

$\boldsymbol{Q3(column)}$: $\boldsymbol{is\ the\ value\ of\ the\ third\ quartile}$

A feature value is considered as ***outlier value*** if it is outside this band: ***[Low threshold, High threshold].***

For each row in that group if the number of its features considered as outliers using the low and high thresholds of each column exceeds **100** outliers per row, the row will be considered as ***«outlier row»*** and it will be dropped from the clean dataset.

Features Scaling:

Since both datasets values are floats and they have negative values for some features. A feature scaling is needed to standardize columns ranges.

features scaling formula:

$$\boldsymbol{new\ value} = \boldsymbol{2}\frac{\boldsymbol{old\ value} - \boldsymbol{min(column)}}{\boldsymbol{max(column)} - \boldsymbol{min(column)}}$$

Dataset type III generation:

Dataset type III will be generated from **cleaned and scaled rows of Dataset type II** to minimize effects of imbalanced columns. We applied a labels transformation to Postural transitions (labels from 7 to 12) of dataset type II so that have the number 7 as an activity id. This number 7 in Dataset type III represent all postural transitions.

Train test files generation:

All Datasets were separated in train and test parts in order to have 70% for training and 30% for testing. This separation was done by selecting rows having the same user id.

User unique labels list (integers from 1 to 30) was separated randomly into two other lists:

train_users = [1,3,5,6,7,8,10,11,14,15,27,17,21,29,30,16,19,20,22,23,25]

test_users = [2,4,9,12,13,26,18,28,24,]

train_users includes unique ids of volunteers chosen for training

test_users includes unique ids of volunteers chosen for testing

This separation was done in user's ids level to eliminate effects of learning **«users' hidden characteristics»** imbedded in features. Since each volunteer has his own way of performing activities, features calculated from these signals may contain some hidden correlations not related to performed activities but it's actually reflects the way each user was performing the protocol (e.g.: the way of moving legs when walking)

When the model fits the train part it will learn **activities' characteristics + user's characteristics**. If the model is tested on rows having different user ids from the training part then **training user's characteristics** won't have an effect on predictions since each user has his unique way to perform daily activities.

III.2 Implementation:

Signal Processing:

For signal processing steps all complicated functions were explained in detail in the additional report.

Data exploration:

After Importing the two datasets. A data exploration pipeline was created to generate Exploration reports for each dataset. This report contains some tables and statistics related to each dataset. Data Exploration pipeline call two other functions already defined. The first function calculates the number of datapoints per each tuple

(user, activity_type). The second function is a visualization function it plots activities' Distribution of each dataset.

Outliers Detection:

An outlier detection pipeline was defined to generate clean datasets with a report containing outliers info and the cleaned dataset info using the data exploration pipeline mentioned above. This pipeline selects and stores rows having the same activity label in separated data frame.

For each data frame:

It detects outlier values of each column based on its low and high thresholds.

It counts the number of outliers per row and stores these numbers in dictionary using the original rows' indexes as keys.

Using the threshold defined earlier which is 100.:

It selects rows considered as outliers in a data frame to generate outlier's info

It selects nonoutlier rows and group them in a clean data frame to generate clean data frame info

It applies Data exploration pipe line to both Data frames (outliers' and cleaned) to generate the final report.

Features scaling:

Features scaling part contains Two functions:

The first one it applies the scaling formula defined earlier to one column

The second function it applies the first function to features columns only from both **"cleaned datasets".**

A report is generated using Data Exploration pipeline.

Dataset type III generation:

After transforming the target labels and generating Dataset type III. This dataset was explored using the same data exploration function to verify if the new targets' distribution is **little imbalanced or not**.

Train and test data generation:

After defining train and test lists which includes respectively unique user ids used for training and testing.

A function is defined to cut the **Datasets type I, II, and III** in a train and test parts using the same user ids lists. The visualization function is applied again to each train and test data frame to verify if activities distribution of each data frame is **little imbalanced or not**.

Metrics:

A Confusion matrix **[12]** function is defined to adapt the binary confusion matrix to multiclass problems and to add all necessary scores (evaluation metrics mentioned in Definition's section). It returns a data frame containing contingency matrix and different scores.

Train test pipeline:

After defining the 3 models mentioned earlier. The first function train_predict is defined to train a model on one sample of a dataset. Possible Sample sizes are 10% 50% and 100% of train files sizes. It returns train-test results of each sample and the confusion matrix.

The second function is defined to apply the first function to **all Datasets (I, II, III) and samples' sizes**. The input model is reinitialized before each call of the first function. For each dataset the model will be trained on samples 10%, 50% and 100%. The train and test accuracies are reported at each phase.to verify if the model is overfitting. The confusion matrix which contains evaluation results is reported only when 100% of the training size is achieved. This function adapts confusion matrix inputs depending on each dataset type it will be dealing with. Finally, it generates a full report includes model's train and test results on each dataset.

Data Pre-processing Steps:

After importing and exploring datasets type I and II. I will delete outliers from both datasets and I will apply feature scaling function to **"cleaned datasets"**.

After generating Dataset type III, I will apply the func **create_training_testing_data** to each dataset (I, II and III)

Machine Learning Steps:

After generating train and test files, I will apply the train test pipeline to the benchmark model to define the benchmark results and thresholds. Then, I will train and test the two other models and report all results to choose the best model.

Decision tree and logistic regression models were implemented with a fixed ***random state=337*** to have the same results if jupyter notebook 's kernel was restarted.

III.3. Refinements:

	Dataset type I	Dataset type II	Dataset type III
Decision Tree accuracy	87.650%	82.761%	87.452%
Logistic Regression accuracy	**96.602%**	**92.553%**	**96.159%**
Selected Models:	**LR**	**LR**	**LR**
Path 1: tuned models 1,4,7 accuracies	**96.673%**	92.465%	**96.071%**
Path 2: tuned models 2,5,8 accuracies	96.426%	**92.582%**	**96.071%**
Path 3: tuned models 3,6,9 accuracies	96.178%	92.172%	95.866%
Selected Models:	**Model 1**	**Model 5**	**Models 7 and 8**
Best C value for each model:	**4.7**	**0.8**	**8.7** **21.6**
Final Accuracies results	**96.956%**	**92.670%**	**96.675%** **96.247%**
LR Final Models 1,2,3	**96.956%**	**92.670%**	**96.675%**

Table: Tuning Results

After applying the train test pipeline to **Decision Tree and Logistic Regression classifiers** it appears that **Logistic Regression (LR)** is the best model for all datasets **(two first rows in the table above)**. From now on Logistic regression classifier is chosen as a pillar base of any final solution.

Some **LR's** parameters' values could not be combined in one model. For example, **L1 norm** which is method to calculate errors cannot be used with **newton-cg solver**. **L2** is the only norm can work with this solver.

I decided to explore Three paths by defining 3 different dictionaries containing some possible parameters' values could be combined (see Ipython notebook "machine learning part").

After the exhaustive search throw each path for all datasets using ***gridsearchcv*** the best prediction's accuracies from each path was reported above (tuned models from 1 to 9).

For **Dataset type I: the *tuned model1*** obtained from the first path has the best accuracy (96.673%) compared to ***models' 2 and 3.***

For Dataset type II: the ***tuned model 5*** obtained from the second path has the highest accuracy (92.582%) compared to ***models 4 and 6.***

For **Dataset type III** both ***models 7 and 8*** obtained from **paths 1 and 2** respectively, have the highest accuracies compared to ***model 9*** obtained from the **3rd path.**

Models with highest accuracies for each dataset will be selected for the next tuning stage.

The last tuning step is to find the best C parameter value for each selected model. A look up function was created for each model to measure accuracies variations when C values varies from 0.1 to 20 with a step of 0.1. A visualization function was created also to plot accuracies variations of each model.

For dataset type 1: the best C value was 4.7 it was added to ***model 1*** parameters to build the **final_model_1.** The accuracy has been increased from 96.673% (model 1) to 96.956% **(final_model_1).**

For dataset type 2: the best C value is 0.8 it was added to ***model 5*** parameters to build the **final_model_2.** The accuracy has been increased from 92.582% (model 5) to 92.670% **(final_model_2).**

For dataset type 3: The **model 7** was selected due its high accuracy. the best C value of this model was 8.7 it was added to ***model 7*** parameters to build the **final_model_3.** The accuracy has been increased from 96.071% (**model 7)** to 96.675% **(final_model_3).**

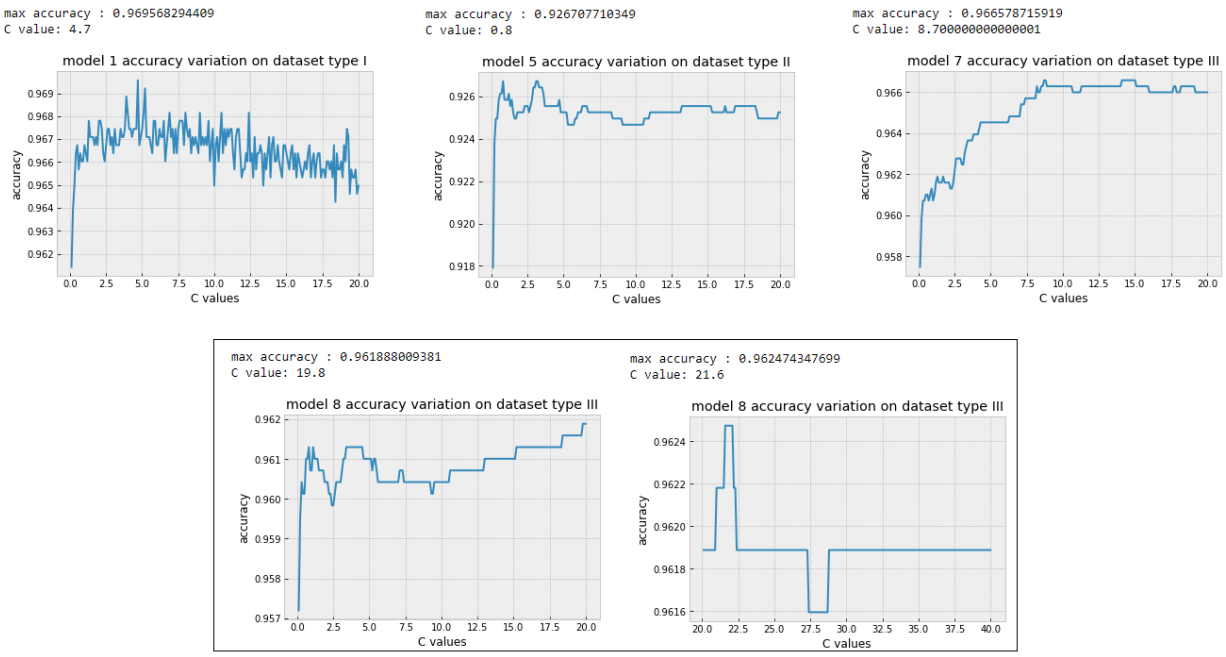


Figure: Hyperparameter C search results

IV- Results:

IV.1. Model Evaluation and Validation

The final models outperform the benchmarks results. The accuracy has been increased from **[72.080%,65.464%,71.210%]** to **[96.956%,92.670%,96.675%]** respectively for **Datasets I, II and III**.

Final models' parameters were carefully adjusted so that they will make better predictions. Accuracies scores' improvements reflect the fine-tuning process.

To test some samples, I will choose some rows from the test part of each dataset. All test parts were never learned by models used in this project:

Dataset type I

	Row index	real identifier	predicted identifier
STANDING	0	5	5
WALKING	500	1	1
SITTING	300	4	4
WALKING_UPSTAIRS	800	2	2
WALKING_DOWNSTAIRS	900	3	3
LIYING	1000	6	6

Dataset type III

	Row index	real identifier	predicted identifier
WALKING	92	1	1
WALKING_UPSTAIRS	135	2	2
WALKING_DOWNSTAIRS	125	3	3
SITTING	15	4	4
STANDING	1	5	5
LIYING	47	6	6
Postural Transition	190	7	7

Dataset type II

	Row index	real identifier	predicted identifier
WALKING	91	1	1
WALKING_UPSTAIRS	134	2	2
WALKING_DOWNSTAIRS	124	3	3
SITTING	14	4	4
STANDING	0	5	5
LIYING	46	6	6
STAND_TO_SIT	189	7	7
SIT_TO_STAND	27	8	8
SIT_TO_LIE	72	9	9
LIE_TO_SIT	56	10	10
STAND_TO_LIE	40	11	11
LIE_TO_STAND	89	12	12

Figure: Samples predictions

Dataset type j samples were predicted using ***final model type j***. ***[j values are 1,2,3]***

For all samples above the real activity and the predicted activity identifiers are equal.

IV.2. Justification:

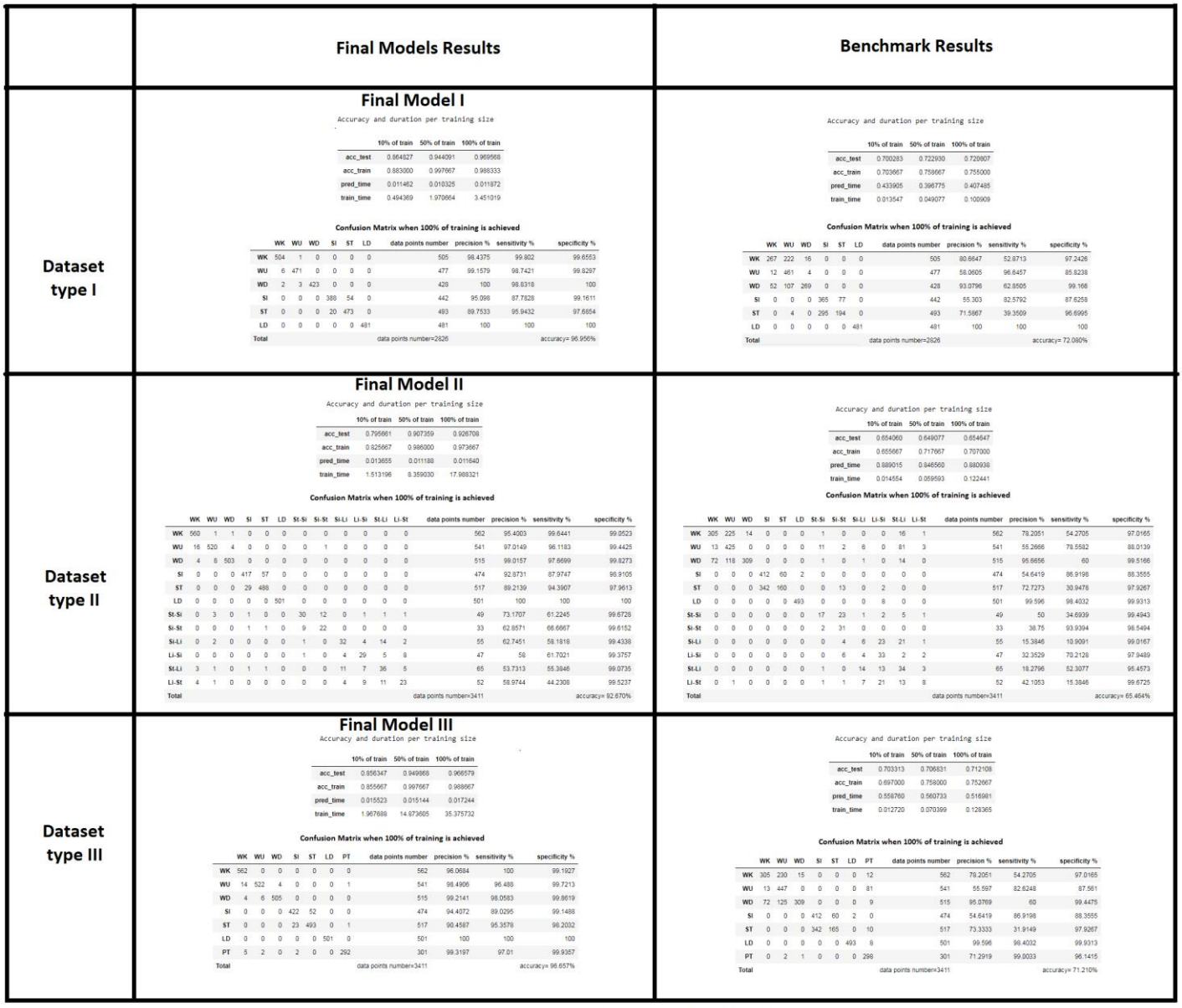
From the figure below, it appears clearly the huge difference between the benchmark performance and the final models' performance on each dataset.

All final models' general accuracies are better than Benchmarks' ones. The number of errors in final models' predictions is very low compared to benchmark errors.

Final models don't fit perfectly(overfit) all presented datasets, the test accuracy of each final model doesn't decrease when the training size is increased.

Final model II has the lowest accuracy in **final model's family** since it has to deal with all types of activities. The Final model I has the highest accuracy due to the low number of classes should be learned and predicted.

In my opinion the best model for this HAR system should be Final model III since it has a high accuracy and can make the difference between basic activities and postural transitions.



(Fig): Benchmark's and Final Models' Results

V- Conclusion

V.1. Free Form Visualization:

	Activity 1	Activity 2	Activity 3	Activity 4	Activity 5	Activity 6	Activity 7	Activity 8	Activity 9	Activity 10	Activity 11	Activity 12
count	30.000000	30.000000	30.000000	30.00000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000
mean	63.700000	58.733333	56.200000	66.00000	71.033333	71.333333	5.233333	3.733333	6.300000	6.033333	7.500000	5.433333
std	9.990168	5.906941	5.647093	10.76072	12.141361	11.544019	1.278019	1.574218	1.207734	1.496740	2.487902	1.454679
min	52.000000	48.000000	46.000000	50.00000	51.000000	53.000000	2.000000	0.000000	4.000000	0.000000	0.000000	2.000000
25%	58.000000	56.000000	54.000000	58.25000	61.000000	60.500000	4.250000	2.250000	6.000000	5.250000	6.000000	5.000000
50%	63.000000	58.500000	55.500000	67.00000	69.000000	74.000000	5.000000	4.000000	6.000000	6.000000	7.000000	5.000000
75%	65.000000	61.000000	57.750000	73.75000	80.750000	80.000000	6.000000	5.000000	7.000000	7.000000	9.000000	6.000000
max	108.000000	72.000000	71.000000	90.00000	99.000000	96.000000	8.000000	6.000000	9.000000	9.000000	12.000000	8.000000

Statistics of windows distribution per each tuple(user,activity)

The table above represents number of windows (or rows in dataset type II) per each tuple (user, activity) obtained from the second windowing method. Datapoints in each window doesn't necessarily have an activity id. Which means that the number of datapoints used to produce this dataset is bigger than number of points recorded while the users were actually performing an activity (useful datapoints). From the table above, we observe some activities (postural transitions) that were not performed by some users (**Min row of activities 8,10 and 11**) adding to that the low number of windows related to postural transitions this is more than enough to justify the low accuracy value obtained from the final model 2 (caused by postural transitions).

V.2 Reflection:

First, I built the signal processing pipeline from scratch to produce classical datasets (I and II) which I though it will be the only hard task in this project. Then I applied the **gaussianNB** model to verify if the signal processing steps were implemented correctly. After that I found a way to delete outliers, and scaling features, to improve the model's performance which will be considered as benchmark results.

When I proposed this project, I thought that tweaking models' parameters will be an easy task compared to signal processing steps but I was wrong. As I proceed through the machine learning part. I discovered the complexity of tweaking parameters and increase final accuracies even with a 0.01%. It is computationally expensive to include the majority of parameters in a grid search.

Now I have a better understanding of the supervised learning approach, signal processing and activity recognition, parameters tuning and definitely, I will look forward doing more projects in Activity Recognition using other approaches.

V.3 Improvements:

The project presented above (signal processing pipeline + final models) could be used as an online HAR tool. By modifying necessary parts in the signal processing pipeline related to targets. In real predictions inputs will be logfiles of acceleration and gyroscope without labels, the online signal processing part should be adapted to produce final features correctly and fastly.

To improve predictions' accuracy, I propose to use the final model 3 which can predict basic activities with a high accuracy. Postural transitions types could be deducted automatically from the previous and the next predictions around it. Since in online mode all obtained rows are chronologically ordered.

For example, if the previous state is sitting (row X) and the next state is standing (row Y) with an acceptable time difference between X and Y. All rows' features between them and predicted as postural transitions using final model 3 should have a postural transition type **"From Sit to Stand".**

References:

[1]: life expectancy _ James Riley for data 1990 and earlier: https://ourworldindata.org/grapher/life-expectancy-globally-since-1770

[2]: https://www.icephd.org/wiki/index.php/Jorge_Luis_Reyes_Ortiz

[3]: https://en.wikipedia.org/wiki/Activity_recognition

[4]: https://en.wikipedia.org/wiki/Support_vector_machine

[5]: https://en.wikipedia.org/wiki/Accelerometer

[6]: https://en.wikipedia.org/wiki/Gyroscope

[7]: https://en.wikipedia.org/wiki/Median_filter

[8]: https://en.wikipedia.org/wiki/Fast_Fourier_transform

[9]: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

[10]: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[11]: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[12]: https://en.wikipedia.org/wiki/Confusion_matrix

[License]: