

# PRML Bonus Project Report

## Human Activity Recognition with Smartphones using Machine Learning

Soumil Sinha B20CS071

### Problem Statement

**Human activity recognition** is the problem of classifying sequences of data recorded by specialized harnesses or smartphones into known well-defined human activities. The goal of this machine learning project is to build a classification model that can precisely identify human fitness activities. Working on this machine learning project will help you understand how to solve multiclass-classification problems.

### Introduction

This dataset has 562 columns along with 1 target column (namely Activity). At the first glance, I thought that I would apply some kind of **Dimensionality Reduction** technique to reduce the number of features while still retaining some properties of the original data.

I didn't have an idea about which model would work best but I knew it had to be some sort of **Tree based algorithm**. Nevertheless, I decided I would test different types of algorithms to decide the best one.

My idea of a **Pipeline** was of an object that stored all the progress I made while exploring this task. It should be able to classify the data along with the usual evaluation metrics in one go. So I decided I will first finalize my model then **build a pipeline around it**.

### Experiments

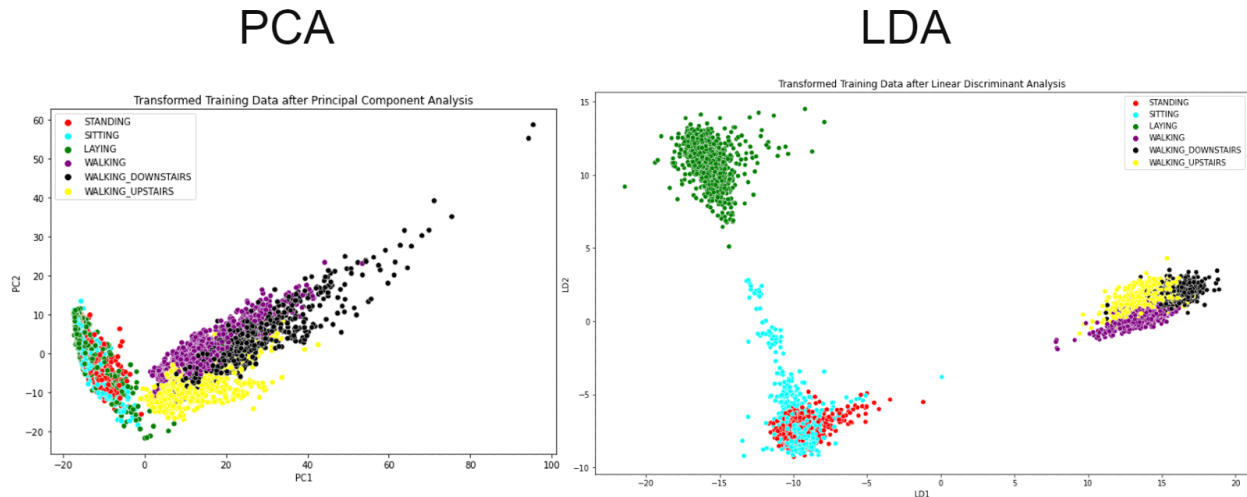
#### Exploratory Data Analysis

After importing the **Train** dataset, I first checked it for any **NULL** values which were none for all columns. I dropped the **subject** column since it was useless for classification and applied a **StandardScaler** to all features. Next up was **Label Encoding** which I applied on the target column with its 6 classes:

- WALKING
- WALKING\_UPSTAIRS
- WALKING\_DOWNSTAIRS
- SITTING
- STANDING

- LAYING

I applied PCA and LDA as a part of **Dimensionality Reduction** techniques. With the best 2 components in each technique, this was the **2D dataset visualisation**:



It is visible that even with 2 components LDA is able to distinguish between different clusters.

The **target column was well-balanced** so when I split the train dataset into train and validation datasets, I used the `stratify = TRAINY` option to preserve this distribution of the **Activity** column.

Next, I started training different models and **tuning them using the validation dataset**. I considered 7 models for training:

- LDA
- QDA
- RandomForestClassifier
- XGBoostClassifier
- LightGBMClassifier
- KNN
- MLPClassifier

I imported the `classification_report` library from `sklearn` which allows me to print various metrics for each class in a single table.

I **tuned a few hyperparameters independently** for some models like `RandomForestClassifier`, `XGBoostClassifier`, `LightGBMClassifier`, `KNN` and `MLPClassifier`. I do not guarantee that these are the best hyperparameters for this dataset (and the corresponding model), I have settled with a **suboptimal solution** since it would take a long time to go through all possible combinations of even a few hyperparameters.

After training and tuning the models I decided to test all these models with their **final hyperparameters** on the **Test** dataset. First, I applied the same preprocessing on this dataset as I had applied on the **Train** dataset. Then I trained these models on the **entire training dataset** and evaluated them on the **Test** dataset.

## Results

I concluded that **LDA** and **XGBoostClassifier** are the **best models** for this dataset.

Next, I implemented an **end-to-end pipeline** which consisted of:

- Data Import
- Preprocessing
- Model Training
- Evaluation

The pipeline is a **class** with **many private methods** and **one public method evaluate**. Once, you initialize the Pipeline object with your choice of model (either XGBC or LDA) and scaler (StandardScaler/MinMaxScaler/MaxAbsScaler) along with the **Train** and **Test** dataset, it stores all these inputs. Then when the **evaluate** function is called through the object, then all the private methods are called sequentially and finally the predictions along with a few classification metrics are outputted. If any error is encountered while executing the pipeline, an error message is printed at the step where the error occurred.

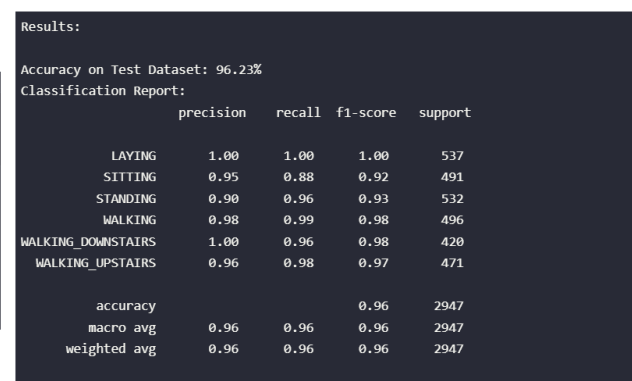


```
PL = Pipeline("LDA", "train.csv", "test.csv", "Activity", "MaxAbsScaler")
PL.evaluate()

72%

Initializing Pipeline... ✓
Setting up the Pipeline... ✓
Importing Data... ✓
Preprocessing Data... ✓
Training Model... ✓
Evaluating Model... ✓
```

Running the pipeline



	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	537
SITTING	0.95	0.88	0.92	491
STANDING	0.90	0.96	0.93	532
WALKING	0.98	0.99	0.98	496
WALKING_DOWNSTAIRS	1.00	0.96	0.98	420
WALKING_UPSTAIRS	0.96	0.98	0.97	471
accuracy			0.96	2947
macro avg	0.96	0.96	0.96	2947
weighted avg	0.96	0.96	0.96	2947

Obtaining results from the pipeline

## Instructions for running the notebook

- Use `git clone https://github.com/sawmill811/Human-Activity-Recognition.git` in your terminal to **clone the repository**. If you wish to run the file on **Google Colaboratory**, then here is the [link to the notebook](#).
- To run the pipeline, simply run the first cell in the `HAR.ipynb` notebook - this will import the necessary libraries.

- Next, skip all cells under the heading "*Exploring the Task*" and run the cells under the heading "*Creating a Pipeline*".
- The predictions for the test data will be stored in the working directory under the filename `predictions.csv`.

## Pipeline Performance

- **LDA**: Predictions are obtained in less than 10 seconds on my PC as well as Google Colaboratory.
- **XGBoostClassifier**: Predictions are obtained in about 30 seconds on my PC (using GPU) and 1.5 minutes (using CPU). It takes about 4 minutes to run the pipeline on Google Colaboratory.

## Troubleshooting

If you are getting an error message then try removing the `tree_method = "gpu_hist"` parameter from the `XGBClassifier` model. This is used to speed up the training process using GPU resources. Other than that, the code should run fine on most systems.

## Abbreviations



**HAR** — Human Activity Recognition  
**PCA** — Principal Component Analysis  
**LDA** — Linear Discriminant Analysis  
**QDA** — Quadratic Discriminant Analysis  
**XGBC** — XGBoostClassifier  
**KNN** — K-Nearest Neighbours  
**MLPClassifier** — MultiLayerPerceptronClassifier