EEE 485/585-Statistical Learning and Data Analytics

Final Project Report

Musteba Anil ONDER

21604090

19.12.2022

Table of Contents

MUSTEBA ANIL ONDER
21604090
EEE485 – Final Project Report
20.11.2022

1) Project Description

For the EEE-485 Statistical Learning and Data Analytics course project, we are planning to make classification by recognizing human activity. This process can be done using machine learning algorithms to analyze the given dataset Human Activity Recognition. We chose it since the dataset was compiled from actual sensor readings.

2) Dataset Description

The dataset, which was obtained via WISDM Lab, is made up of information gathered from 36 distinct users while they engaged in six various human activities for predetermined durations (ascending and descending stairs, sitting, walking, running, and standing). These statistics came from accelerometers, which can determine the orientation of the object being used to measure acceleration in all three dimensions. They were gathered at a sampling rate of 20 Hz, or 20 samples per second (1 sample every 50 millisecond). These time-series data can be used for a variety of tasks, including the detection of human activity.

Features:

- User**:** the user who acquired the data (integer from 1 to 36).
- Activity**:** the activity that the user was carrying out. It could be:
    - 1. Walking
    - 2. Jogging
    - 3. Sitting
    - 4. Standing
    - 5. Upstairs
    - 6. Downstairs.
- Timestamp**:** generally, the phone's uptime in nanoseconds.
- X**-axis:** The acceleration in the x direction as measured by the android phone's accelerometer.

  Floating-point values between -20 and 20. A value of 10 = 1g = 9.81 m/s^2, and 0 = no acceleration.

  The acceleration recorded includes gravitational acceleration toward the center of the Earth, so that when the phone is at rest on a flat surface the vertical axis will register +-10.
- Y**-axis:** same as x-axis, but along y axis.
- Z**-axis:** same as x-axis, but along z axis.

3) Work Done

The work done is transforming the smartphone accelerometer's raw signal data and extracting additional characteristics to identify six typical human activities. To analyze which values have changed, we create some statistics about our dataset.

a. Data preparation

According to results, we understand that instead of only x, y, z axes and time data, we have 94 distinct features. Since according to graphs data has some periodic features, we create windows containing 100 samples. The answer to how we choose these features is that we simply extract basic statistical data of each axis like mean, standard deviation etc. Additionally, due to rising doubts for frequency of axis components, we analyze the frequency domain of each axis.

Based on the observation, we carry out the subsequent steps: Drop the rows where the timestamp is 0, alter the datatype of the "z-axis" column to "float," and sort the data by user name and date in ascending order.

The raw time-series data cannot be simply used to standard categorization techniques. Instead, we must first use the "windowing" approach to modify the raw time-series data. In this method, we split the data into windows of 5 seconds, and then we aggregate the 100 raw samples that are present in each of these 5 second windows to produce new features. We select the action that occurs the most frequently in that window to apply a class label to the modified characteristics.

It will be divided into two rows after windowing and aggregation (with window size = 50). The most common action in each window will be the class-label applied to these 2 additional rows. Similarly, for a million rows, the transformed set will contain almost 20,000 rows.

We will create a total of 18 simple statistical features in stage 1 of feature engineering:

1. mean
2. standard deviation
3. average absolute deviation
4. minimum value
5. maximum value
6. difference of maximum and minimum values
7. median
8. median absolute deviation
9. interquartile range
10. negative values count
11. positive values count
12. number of values above mean
13. number of peaks
14. skewness
15. kurtosis
16. energy
17. average resultant acceleration
18. signal magnitude area

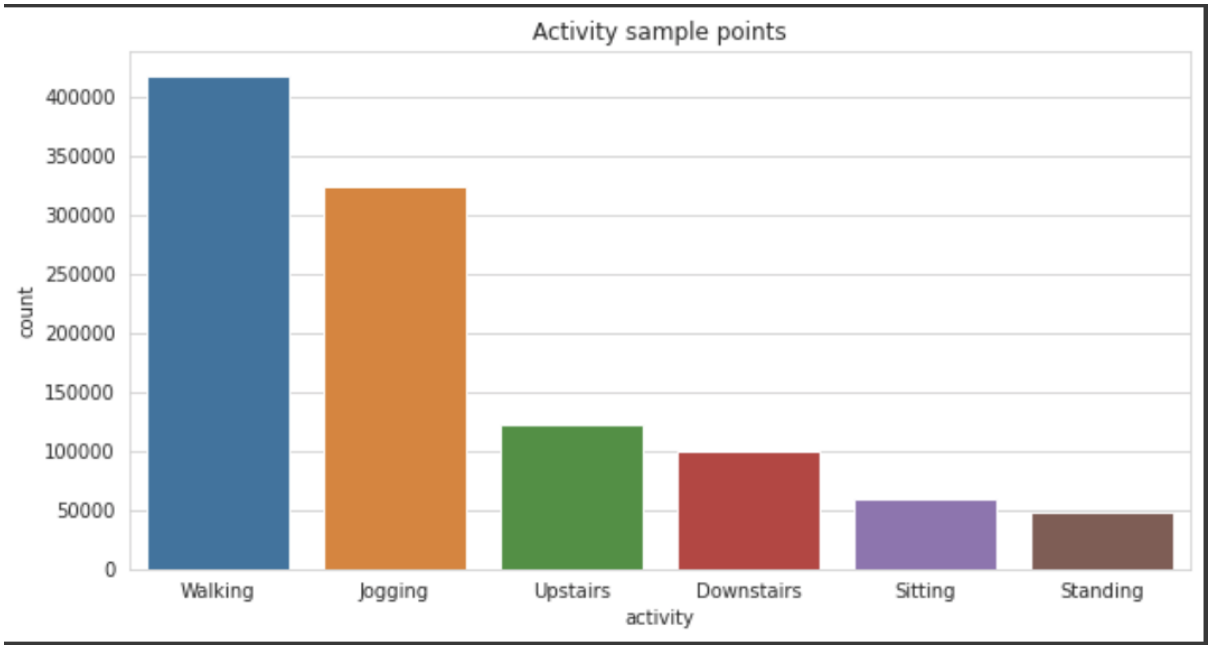

Figure 1: Activity sample counts

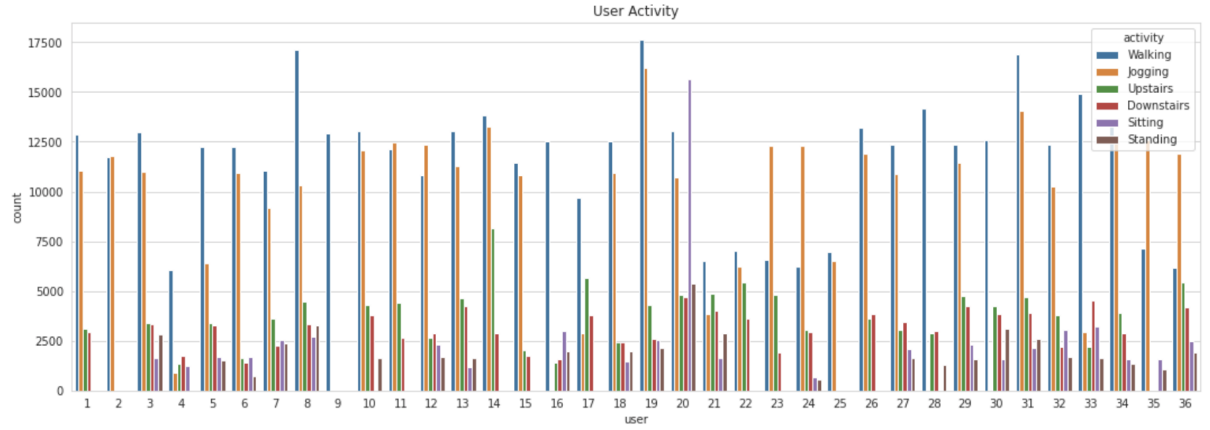

Figure 2: Activity sample count per user

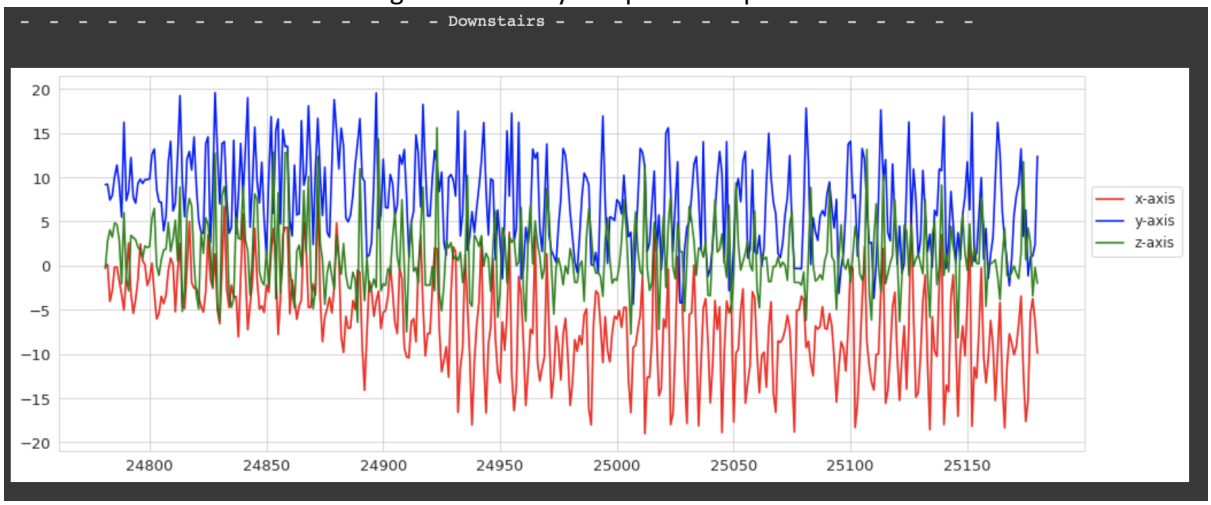

Figure 3: Sample analysis of downstairs samples

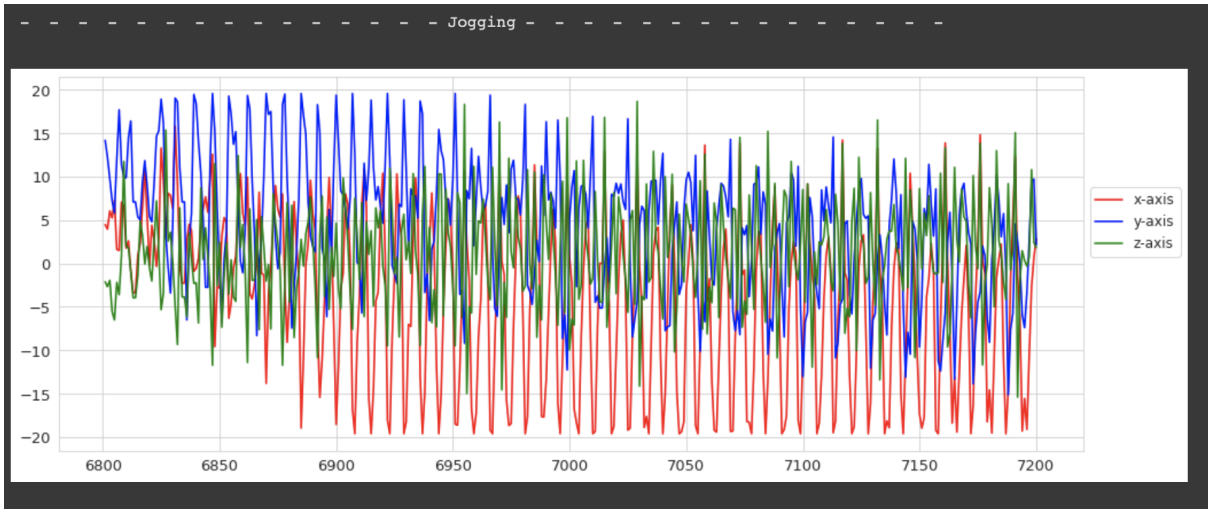

Figure 4: Sample analysis of jogging samples

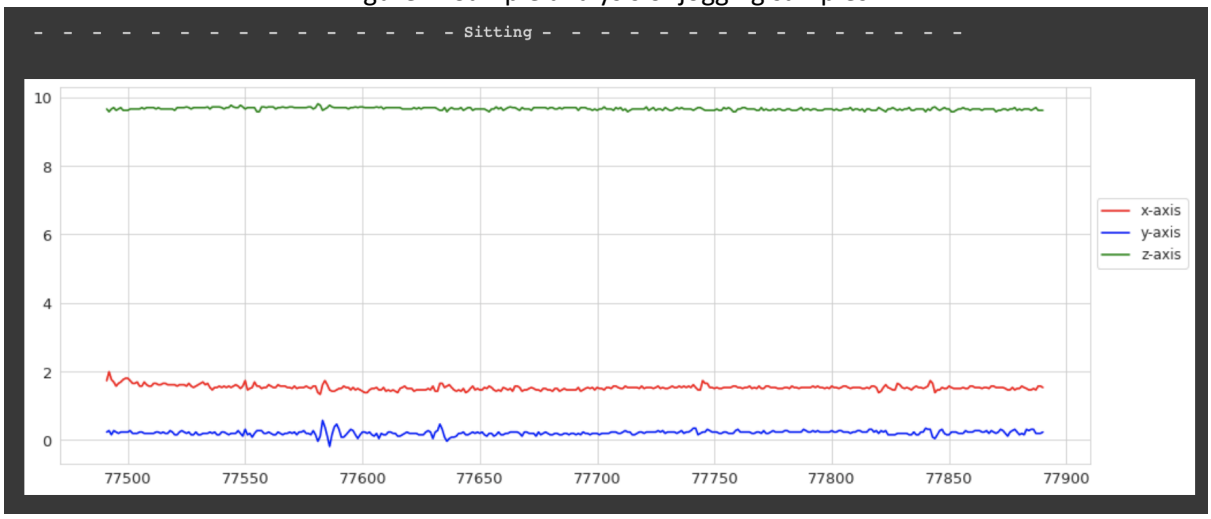

Figure 5: Sample analysis of sitting samples

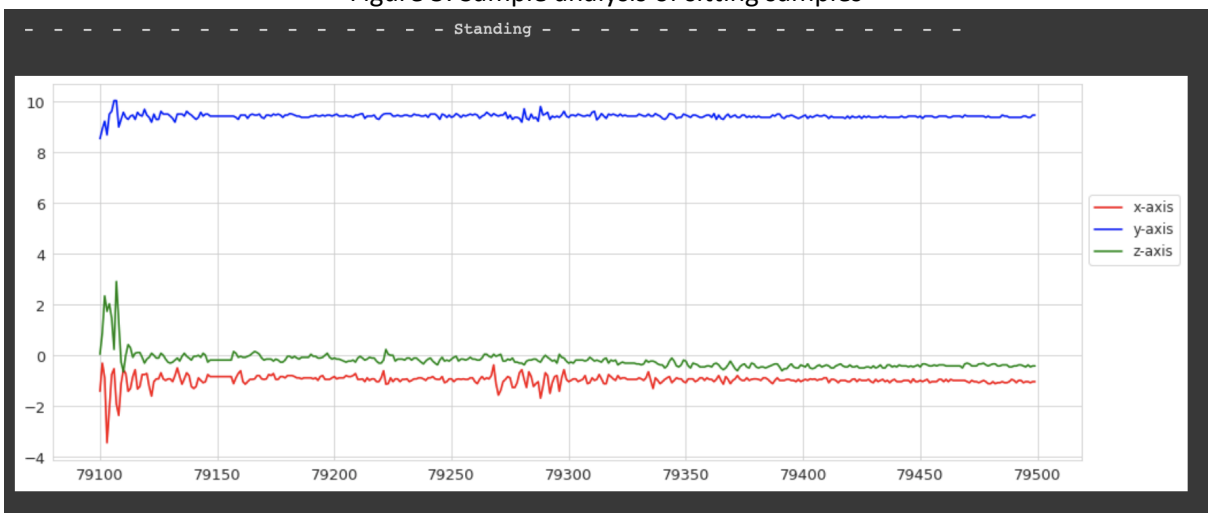

Figure 6: Sample analysis of standing samples

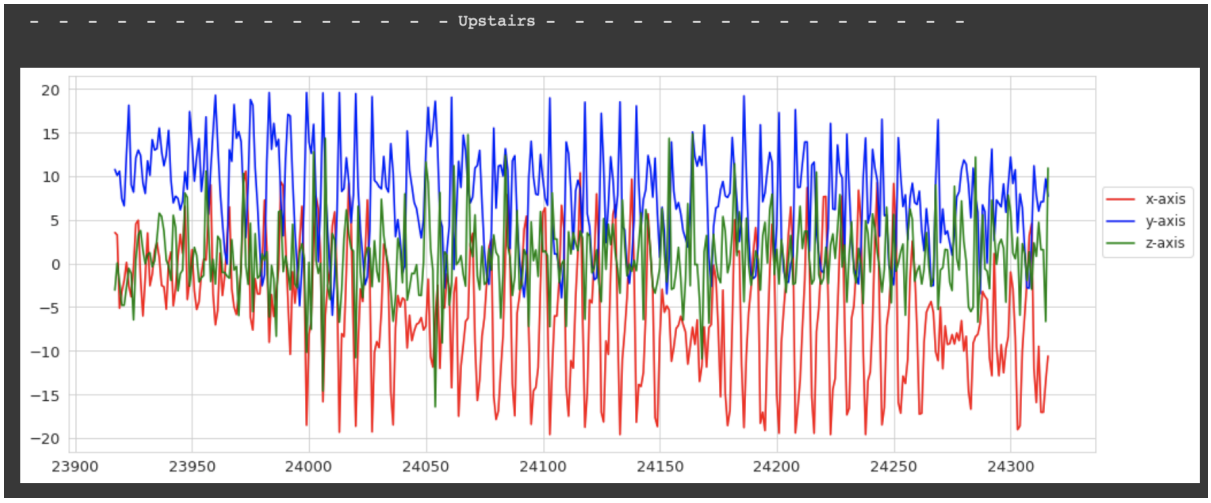

Figure 7: Sample analysis of upstairs

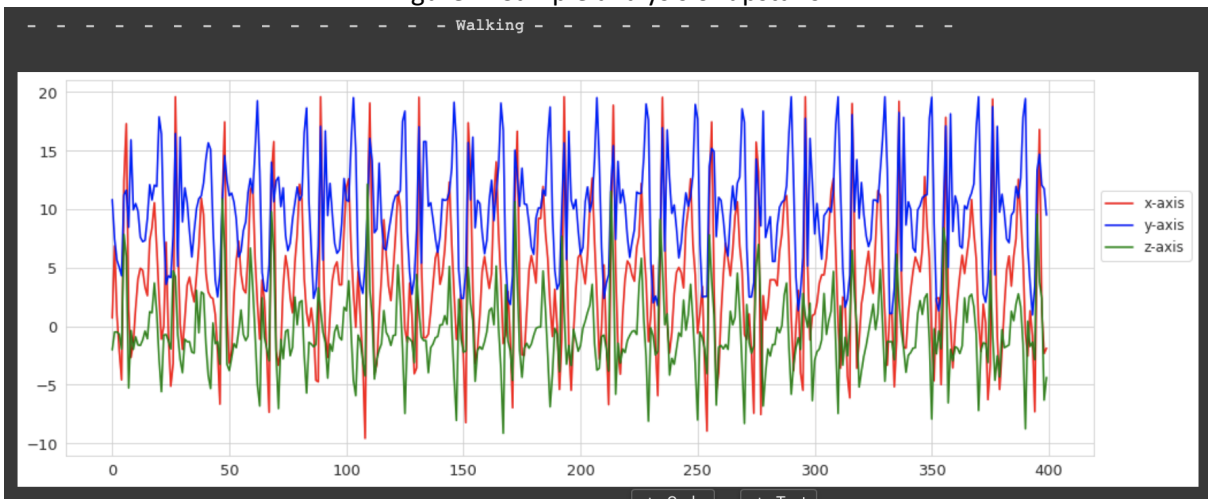

Figure 8: Sample analysis of walking samples

The raw time-series data cannot be simply used to standard categorization techniques. Instead, we must first use the "windowing" approach to modify the raw time-series data. In this method, we split the data into windows of 5 seconds, and then we aggregate the 100 raw samples that are present in each of these 5 second windows to produce new features. We select the action that occurs the most frequently in that window to apply a class label to the modified characteristics.

It will be divided into two rows after windowing and aggregation (with window size = 50). The most common action in each window will be the class-label applied to these 2 additional rows. Similarly, for a million rows, the transformed set will contain almost 20,000 rows.

We will create a total of 18 simple statistical features in stage 1 of feature engineering:

1. mean
2. standard deviation
3. average absolute deviation
4. minimum value
5. maximum value
6. difference of maximum and minimum values
7. median
8. median absolute deviation
9. interquartile range

10. negative values count
11. positive values count
12. number of values above mean
13. number of peaks
14. skewness
15. kurtosis
16. energy
17. average resultant acceleration
18. signal magnitude area

After extracting simple features, we moved to the next step which is Fourier Transform. A time domain signal is converted into a frequency domain signal using the Fourier transform function. The function takes a temporal signal as input and outputs the signal's frequency representation. In the actual world, every signal is a temporal signal made up of several sinusoids with various frequencies.

The signal is not changed by the Fourier transform. Because some characteristics and elements of the signal may be completely studied in the frequency domain, it just offers a fresh perspective for examining your time signal.

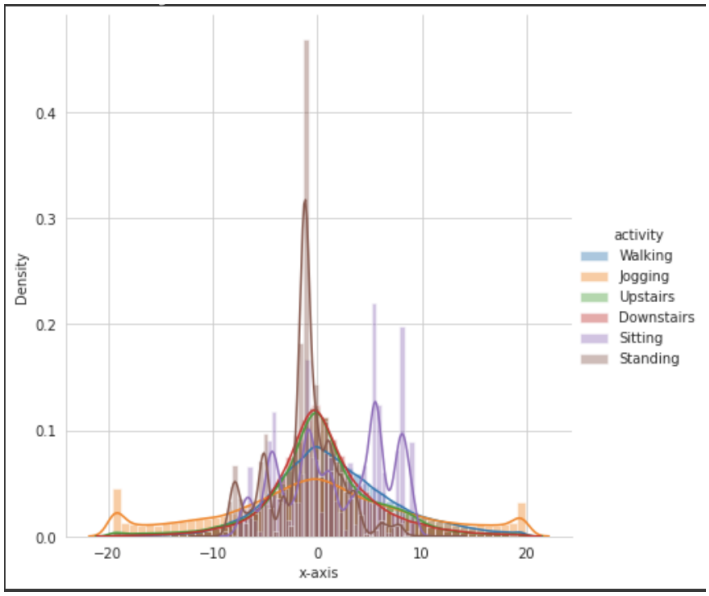

Figure 9: FFT of x-axis

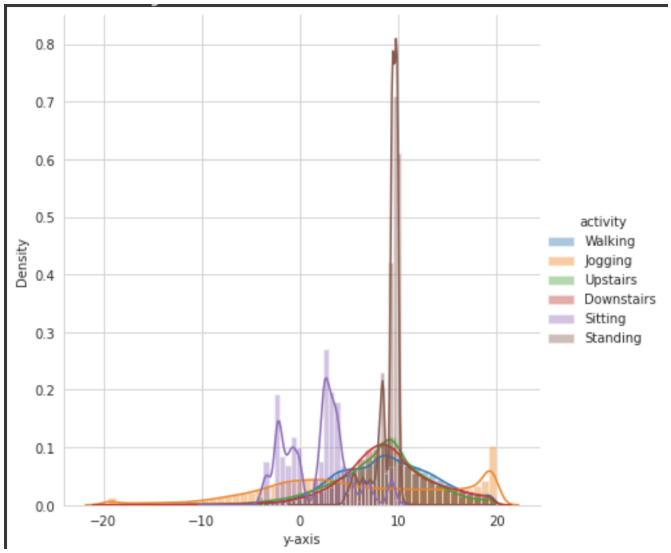

Figure 10: FFT of y-axis

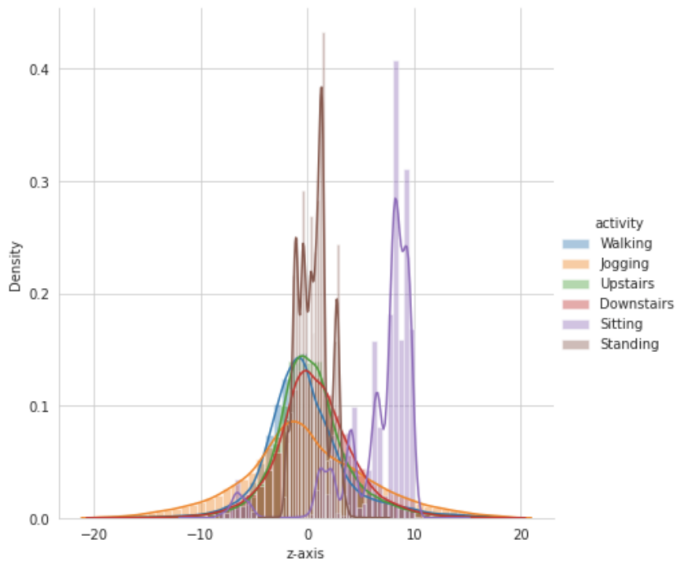

Figure 11: FFT of z-axis

At the end of all the executions we had four datasets which are X_train, X_test, y_train and y_test. The data preparation is done here. Then we integrated these datasets to our machine learning models.

b. Selected Algorithms

Multinomial classification is a machine learning technique used to classify instances into one of multiple classes. In multinomial classification, there can be three or more classes. To solve a multinomial classification problem, we can use a machine learning algorithm that is capable of handling multiple classes. Some examples of algorithms that can be used for multinomial classification include logistic regression, support vector machines, and decision trees. To train a multinomial classifier, we need a dataset that contains a set of labeled examples, where each example belongs to one of the multiple classes. We can then use the labeled examples to train the classifier, which will learn to predict the class of a new instance based on its features.

In this project, I also applied multinomial classification with three algorithms which are these algorithms, KNN, Decision Tree and Logistic regression. K-nearest neighbors (KNN) is a simple and effective method for multinomial classification. It can be used for classification problems with multiple classes. KNN is a suitable method for multinomial classification because it can handle multiple classes and it is not sensitive to the scale of the features. It is also easy to implement and does not require a lot of computation, making it efficient for large datasets. However, KNN can be slow when making predictions because it requires calculating the distance between the new instance and all the instances in the training set. It can also be affected by the choice of the distance measure and the value of "k." Decision trees are a popular method for multinomial classification because they are easy to understand and interpret, and they can handle multiple classes without the need for transformation. Decision trees are suitable for multinomial classification because they can handle multiple classes directly. They are also easy to understand and interpret, making them useful for explaining the decision-making process to non-technical audiences. However, decision trees can be prone to overfitting, especially when they are not pruned, and they may not be the most accurate method for certain types of data. Finally, Logistic Regression is a popular method for multinomial classification because it is a simple and efficient method that can handle multiple classes. It is a type of regression analysis that is used to predict the probability of an event occurring based on the values of a set of features. Logistic regression is suitable for multinomial classification because it can handle multiple classes, and it is a simple and efficient method that is easy to implement. It can also provide probabilistic predictions, which can be useful in certain applications. However, logistic regression may not perform well on complex and non-linear datasets, and it can be sensitive to the scale of the features.

I also used cross validation for all these algorithms to improve my results. Cross-validation is a resampling procedure used to evaluate the performance of a machine learning model. It is a way to estimate the skill of the model on new data. In cross-validation, the data is split into two or more subsets, and the model is trained and evaluated multiple times using different combinations of the subsets. here are several types of cross-validation techniques, including k-fold cross-validation and stratified k-fold cross-validation. In k-fold cross-validation, the data is randomly split into "k" folds, and the model is trained and evaluated "k" times, using a different fold as the test set in each iteration. Stratified k-fold cross-validation is similar, but it ensures that the proportions of the classes in the folds are approximately the same as in the original dataset.

c. KNN Algorithm

K-nearest neighbors (KNN) is a non-parametric, instance-based learning algorithm that can be used for classification or regression tasks. In the KNN algorithm, a set of labeled training examples is used to make predictions for a new, unlabeled example. The prediction is made based on the label of the nearest neighbors to the new example in the training set, as determined by a distance metric such as Euclidean distance.

Euclidean distance is a measure of the straight-line distance between two points in Euclidean space. It is calculated as the square root of the sum of the squares of the differences between the coordinates of the points. For example, the Euclidean distance between points A and B in two-dimensional space is calculated as follows:

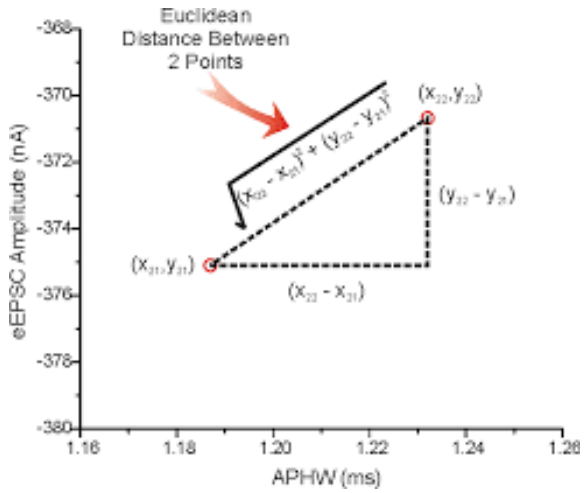

Figure 1: The Euclidian Distance

Steps of the KNN Algorithm that I applied is as the following list:

1)Split the data into a training set and a test set. The training set will be used to build the model, and the test set will be used to evaluate the model's performance.
2)Use cross-validation to determine the best value of k for the training set.
3)For each data point in the training set, calculate the distances to all other data points.
4)Select the k data points in the training set that are closest to the data point in question. These are the "nearest neighbors" of the data point.

5) Repeat steps 3-4 for each data point in the training set.
6)Use the trained model to classify the data points in the test set.
7)Calculate the model's performance by comparing the predicted classes to the true classes of the data points in the test set.

The results are as following:

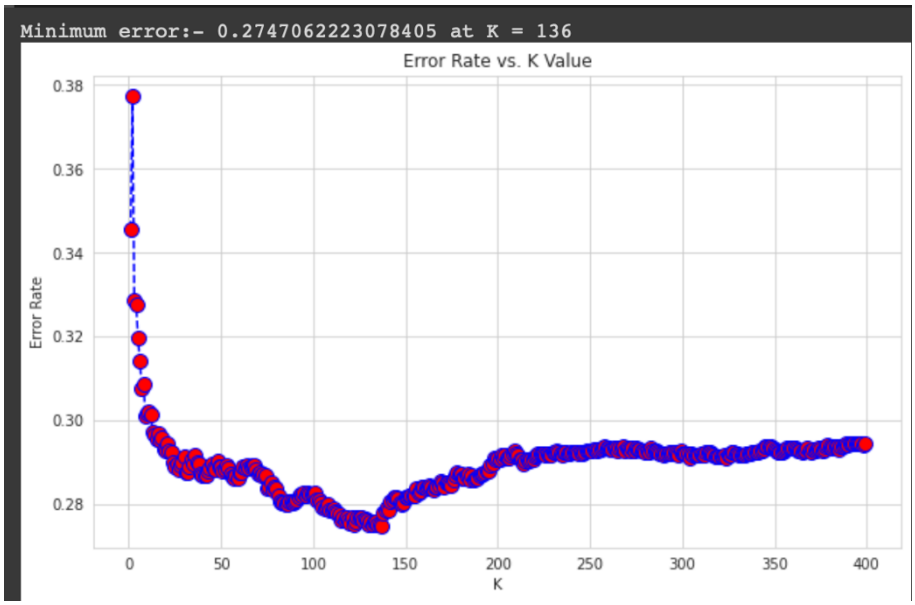

Figure 2: Finding best hyper-parameter K value

```
The Sklearn Library KNN Accuracy Result = 0.7249084954729339
The KNN class that I created Accuracy Result = 0.724715854363211
The error according to sklearn library = % 0.026574541604353297
```

Figure 3:The results of KNN

d. Decision Tree

A decision tree is a machine learning algorithm that is used for classification and regression tasks. It works by creating a tree-like model of decisions and their possible consequences, with the goal of predicting the outcome of a new data point based on its features. In a decision tree, the root node represents the root of the tree, and each branch represents a decision that splits the data into two or more subgroups. The leaves of the tree represent the final classification or prediction.

$$H = -\sum_{i=1}^{N} p_i \log(p_i)$$

Entropy is a measure of the disorder or randomness in a system. In the context of decision trees, entropy is used to measure the impurity of a set of data. When building a decision tree, the goal is to maximize the information gain at each split, which means choosing the split that results in the purest subgroups. This is done by minimizing the entropy of the subgroups.

The implementation of a decision tree using entropy as the measure of impurity for determining the best split at each step in the tree. The build_tree function is a recursive function that builds the decision tree by continually splitting the data into subgroups until certain stopping conditions are met. The stopping conditions in this implementation are specified by the min_samples_split and max_depth parameters, which specify the minimum number of samples required to split a node and the maximum depth of the tree, respectively. The get_best_split function is used to find the best split at each step in the tree building process. It does this by looping over all the features and possible thresholds for each feature, and selecting the split that results in the greatest information gain, which is a measure of how much the purity of the subgroups is increased by the split. The split function is used to split the data into two subgroups based on a given feature index and threshold. The information_gain function is used to calculate the information gain for a given split, using either entropy or the Gini index as the measure of impurity. Finally, the calculate_leaf_value function is used to compute the final classification or prediction at the leaf nodes of the tree. This is usually done by taking the majority class or the mean value of the data at the leaf node.

I could not train all the train set since it lasts more than 12 hours and Google Colab only allows 12 hours unless purchasing a premium account. So, I just got a result with 1000 samples which corresponds to 6% of the total train data. The comparison of the results that I get from Sklearn and class I created is as the following:

```
X_52 <= 32.042262515648964 ? 0.862330963692662
 left:X_84 <= 1.9140239430829686 ? 0.1344405830573696
  left:X_2 <= 0.2316 ? 0.6232802965716294
    left:X_0 <= 1.9185999999999996 ? 0.205592508185083
        left:Walking
        right:Downstairs
    right:X_62 <= 205.91416017320796 ? 0.6896763215021658
        left:Downstairs
        right:Upstairs
  right:X_3 <= 7.0998812975992776 ? 0.04097202425663312
    left:Walking
    right:Upstairs
 right:X_6 <= 4.81953 ? 0.3436175218385429
  left:X_11 <= -17.92 ? 0.6320683403479106
    left:X_5 <= 4.851601256286423 ? 0.9927744539878083
        left:Walking
        right:Jogging
    right:X_13 <= 16.17 ? 0.1792560669283215
        left:Upstairs
        right:Downstairs
  right:X_55 <= 30.627762586694715 ? 0.17799939160533207
    left:X_54 <= 49.3703125126166 ? 0.4691703417723029
        left:Upstairs
        right:Jogging
    right:X_20 <= -0.30500000000000005 ? 0.05846103801228965
        left:Jogging
        right:Jogging
```

Figure 4:The decision tree of my implementation with 6% of the train set

```
0.42997495665575036
```

Figure 5:Accuracy of my implementation with 6% of the train set

```
Accuracy: 0.7640146407243306
```

Figure 6:The Accuracy by Sklearn Library for Decision Tree

e. Logistic Regression

Logistic regression with softmax is a type of classification algorithm that is used to predict the class of an input sample in a multiclass classification problem. It is a generalization of logistic regression, which is used for binary classification, to the case where there are more than two classes. In logistic regression with softmax, the goal is to learn a set of weights and biases that can be used to predict the class of an input sample by computing the probability of each class given the input features. The predicted class is the one with the highest probability.

To compute the probability of each class, logistic regression with softmax applies the softmax function to the linear combination of the input features and the weights. The softmax function is defined as follows:

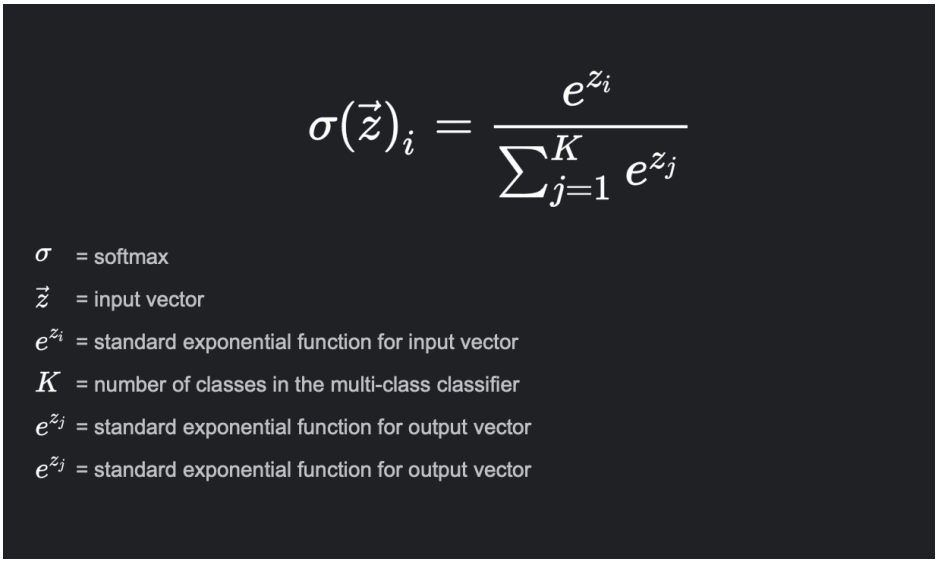

Figure 7:Softmax Equation

The softmax function maps the linear combination of the input features and the weights to a probability distribution over the classes. The predicted class is the one with the highest probability. Logistic regression with softmax is often used in machine learning and deep learning applications, such as image classification, natural language processing, and speech recognition.

In this custom implementation of a logistic regression classifier in Python using without the scikit-learn API. The MyLogisticReg class extends the BaseEstimator and ClassifierMixin classes from without scikit-learn to implement a classifier. It has a constructor __init__ that initializes several attributes such as the model parameters, the number of features and classes, an OneHotEncoder object, and a verbosity flag. The init_params method is used to randomly initialize the model parameters, which include the slope coefficients and the y-intercept. The get_logits method computes the logits, which are the input to the sigmoid function in the case of binary classification, or the softmax function in the case of multi-class classification. The logits are computed as the dot product of the feature matrix and the coefficient matrix, plus the intercept matrix. The predict_proba method takes in a feature matrix X and returns the predicted class probabilities. It first computes the logits using the get_logits method, and then applies the sigmoid function for binary classification, or the softmax function for multi-class classification. The fit method is used to fit the logistic regression model to the training data. It takes in the feature matrix X and target labels y, along with optional hyperparameters such as the learning rate and the number of iterations. It first one-hot encodes the target labels if it is a multi-class classification problem, and then initializes the model parameters using the init_params method. It then uses stochastic gradient descent to optimize the model parameters and minimize the cross-entropy loss. The score method takes in a feature matrix X and target labels y, and returns the mean accuracy on the given test data and labels. It first calls the predict method to get the predicted labels, and then compares them to the true labels to compute the accuracy.
The following result is the accuracy that I reached by using Sklearn Library and the Logistic Regression model which I created:

```
Accuracy: 0.8431901367751878

 -------------Classification Report-------------

              precision    recall  f1-score   support

  Downstairs       0.67      0.74      0.71       514
     Jogging       0.82      0.94      0.88      1520
     Sitting       0.92      0.99      0.95       361
    Standing       0.99      0.82      0.90       299
     Upstairs      0.70      0.63      0.66       582
      Walking      0.93      0.83      0.88      1915

    accuracy                           0.84      5191
   macro avg       0.84      0.83      0.83      5191
weighted avg       0.85      0.84      0.84      5191
```

Figure 8:Logistic Regression Results by Sklearn

```
My implementation       :0.8425856784576482
Sklearn implementation  : 0.8431901367751878
```

Figure 9: Logistic Regression Model Accuracy

4) Division Of Work

Musteba Anil Onder

Through the term project, I completed almost every part of this project. At first, we had prepared a proposal but when TA asked to revise, I completed the project proposal. After that, we made a work division which is I would do the data preparation and my teammate would implement the machine learning algorithms. However, he did not contribute anything. In interim report we could not report any methods, but I implemented KNN and Decision Tree in two days between the interim report deadline and presentation time. After presentation, I encouraged him to implement the last algorithm which is Logistic Regression but he did not contribute anything again even if I tried to communicate with him. So, in total, I wrote project proposal and the first half of the interim report, I implemented the data preparation, cross-validation, KNN, Decision Tree and Logistic Regression algorithms which is the all of the programming about the project. And finally, I wrote final project report. I was always enthusiastic for group work but my teammate did not help me at all. So that, I was unable to create a Gant Chart because of these reasons since until December,18 I was thinking that he completed the Logistic Regression part but he did not. So, I completed these parts and report just in 2 days.

Ceyhun Yilmaz

He helped while searching dataset at the beginning of the semester and wrote the second half of the interim report. He accepts that he only contributed these.

Reference

[1] “Human Activity Recognition”, “https://www.kaggle.com/datasets/die9origephit/human-activity-recognition”, 27.10.2022 .

[2]”Feature Engineering on Time-Series Data for Human Activity Recognition”,Pratik NABRIYA, “https://towardsdatascience.com/feature-engineering-on-time-series-data-transforming-signal-data-of-a-smartphone-accelerometer-for-72cbe34b8a60”, 20.11.2022

[3] “Definition logistic regression” “https://www.techtarget.com/searchbusinessanalytics/definition/logistic-regression” 20.11.2022

[4] “Decision Trees” “https://scikit-learn.org/stable/modules/tree.html” 20.11.2022

[5] “Decision Tree” https://www.geeksforgeeks.org/decision-tree/ 20.11.2022

[6] “K-Nearest Neighbors Algorithm” https://www.ibm.com/topics/knn 20.11.2022

Appendix

```
# -*- coding: utf-8 -*-
"""TermProjectEEE485.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1YOVmJIvzBbnPk_Fo-fs8lH4kownSosJT
"""

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.signal import find_peaks
import warnings
warnings.filterwarnings('ignore')

"""Following code line drops the null values"""

dataframe = pd.read_csv('time_series_data_human_activities.csv', sep=',')
dataframe.shape

human_activity = dataframe.dropna()
human_activity.shape

print(human_activity['activity'].dtype)
#print(har_df['timestamp'].dytpe)
print(human_activity['user'].dtype)
print(human_activity['x-axis'].dtype)
print(human_activity['y-axis'].dtype)
print(human_activity['z-axis'].dtype)

# drop the rows where timestamp is 0
dataframe = human_activity[human_activity['timestamp'] != 0]
human_activity.shape

# now arrange data in ascending order of the user and timestamp
human_activity = human_activity.sort_values(by = ['user', 'timestamp'], ignore_index=True)

sns.set_style("whitegrid")
```

```
plt.figure(figsize = (10, 5))
sns.countplot(x = 'activity', data = dataframe)
plt.title("Activity sample points")
plt.show()

plt.figure(figsize = (18, 6))
sns.countplot(x = 'user', hue = 'activity', data = dataframe)
plt.title('User Activity')
plt.show()

def plot_activity(activity):
  data = human_activity[human_activity['activity'] == activity][['x-axis', 'y-axis', 'z-axis']][:400]
  axis = data["x-axis"].plot(subplots=True, color="r", fontsize = 13)
  axis = data["y-axis"].plot(subplots=True, color="b", fontsize = 13)
  axis = data["z-axis"].plot(subplots=True, color="g", fontsize = 13)
  for ax in axis:
     ax.legend(loc='lower left', bbox_to_anchor=(1.0, 0.5), fontsize = 12)
for i in np.unique(human_activity['activity'].values):
  plt.figure(figsize = (16, 6))
  print("\n")
  print(" - "*15 + str(i) + " - "*15)
  print("\n")
  plot_activity(i)
  plt.show()

sns.FacetGrid(dataframe, hue = 'activity', size = 6).map(sns.distplot, 'x-axis').add_legend()

sns.FacetGrid(dataframe, hue = 'activity', size = 6).map(sns.distplot, 'y-axis').add_legend()

sns.FacetGrid(dataframe, hue = 'activity', size = 6).map(sns.distplot, 'z-axis').add_legend()

# train data -> Users upto User ID = 28 (i.e. 24 users)
train = dataframe[dataframe['user'] <= 28]
# test data -> Users from User ID = 28 to 36 (i.e. 12 users)
test = dataframe[dataframe['user'] > 28]

x_set = []
y_set = []
z_set = []
train_labels = []

window_size = 100
step_size = 50
```

```
for i in range(0, train.shape[0] - window_size, step_size):
    xval = train['x-axis'].values[i: i + 100]
    yval = train['y-axis'].values[i: i + 100]
    zval = train['z-axis'].values[i: i + 100]
    label_train = stats.mode(train['activity'][i: i + 100])[0][0]

    x_set.append(xval)
    y_set.append(yval)
    z_set.append(zval)
    train_labels.append(label_train)

# Statistical Features on raw x, y and z in time domain
X_train = pd.DataFrame()

# mean
X_train['x_mean'] = pd.Series(x_set).apply(lambda x: x.mean())
X_train['y_mean'] = pd.Series(y_set).apply(lambda x: x.mean())
X_train['z_mean'] = pd.Series(z_set).apply(lambda x: x.mean())

# std dev
X_train['x_std'] = pd.Series(x_set).apply(lambda x: x.std())
X_train['y_std'] = pd.Series(y_set).apply(lambda x: x.std())
X_train['z_std'] = pd.Series(z_set).apply(lambda x: x.std())

# avg absolute diff
X_train['x_aad'] = pd.Series(x_set).apply(lambda x: np.mean(np.absolute(x - np.mean(x))))
X_train['y_aad'] = pd.Series(y_set).apply(lambda x: np.mean(np.absolute(x - np.mean(x))))
X_train['z_aad'] = pd.Series(z_set).apply(lambda x: np.mean(np.absolute(x - np.mean(x))))

# min
X_train['x_min'] = pd.Series(x_set).apply(lambda x: x.min())
X_train['y_min'] = pd.Series(y_set).apply(lambda x: x.min())
X_train['z_min'] = pd.Series(z_set).apply(lambda x: x.min())

# max
X_train['x_max'] = pd.Series(x_set).apply(lambda x: x.max())
X_train['y_max'] = pd.Series(y_set).apply(lambda x: x.max())
X_train['z_max'] = pd.Series(z_set).apply(lambda x: x.max())

# max-min diff
X_train['x_maxmin_diff'] = X_train['x_max'] - X_train['x_min']
X_train['y_maxmin_diff'] = X_train['y_max'] - X_train['y_min']
X_train['z_maxmin_diff'] = X_train['z_max'] - X_train['z_min']
```

```
# median
X_train['x_median'] = pd.Series(x_set).apply(lambda x: np.median(x))
X_train['y_median'] = pd.Series(y_set).apply(lambda x: np.median(x))
X_train['z_median'] = pd.Series(z_set).apply(lambda x: np.median(x))

# median abs dev
X_train['x_mad'] = pd.Series(x_set).apply(lambda x: np.median(np.absolute(x - np.median(x))))
X_train['y_mad'] = pd.Series(y_set).apply(lambda x: np.median(np.absolute(x - np.median(x))))
X_train['z_mad'] = pd.Series(z_set).apply(lambda x: np.median(np.absolute(x - np.median(x))))

# interquartile range
X_train['x_IQR'] = pd.Series(x_set).apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
X_train['y_IQR'] = pd.Series(y_set).apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))
X_train['z_IQR'] = pd.Series(z_set).apply(lambda x: np.percentile(x, 75) - np.percentile(x, 25))

# negtive count
X_train['x_neg_count'] = pd.Series(x_set).apply(lambda x: np.sum(x < 0))
X_train['y_neg_count'] = pd.Series(y_set).apply(lambda x: np.sum(x < 0))
X_train['z_neg_count'] = pd.Series(z_set).apply(lambda x: np.sum(x < 0))

# positive count
X_train['x_pos_count'] = pd.Series(x_set).apply(lambda x: np.sum(x > 0))
X_train['y_pos_count'] = pd.Series(y_set).apply(lambda x: np.sum(x > 0))
X_train['z_pos_count'] = pd.Series(z_set).apply(lambda x: np.sum(x > 0))

# values above mean
X_train['x_above_mean'] = pd.Series(x_set).apply(lambda x: np.sum(x > x.mean()))
X_train['y_above_mean'] = pd.Series(y_set).apply(lambda x: np.sum(x > x.mean()))
X_train['z_above_mean'] = pd.Series(z_set).apply(lambda x: np.sum(x > x.mean()))

# number of peaks
X_train['x_peak_count'] = pd.Series(x_set).apply(lambda x: len(find_peaks(x)[0]))
X_train['y_peak_count'] = pd.Series(y_set).apply(lambda x: len(find_peaks(x)[0]))
X_train['z_peak_count'] = pd.Series(z_set).apply(lambda x: len(find_peaks(x)[0]))

# skewness
X_train['x_skewness'] = pd.Series(x_set).apply(lambda x: stats.skew(x))
X_train['y_skewness'] = pd.Series(y_set).apply(lambda x: stats.skew(x))
X_train['z_skewness'] = pd.Series(z_set).apply(lambda x: stats.skew(x))

# kurtosis
X_train['x_kurtosis'] = pd.Series(x_set).apply(lambda x: stats.kurtosis(x))
X_train['y_kurtosis'] = pd.Series(y_set).apply(lambda x: stats.kurtosis(x))
X_train['z_kurtosis'] = pd.Series(z_set).apply(lambda x: stats.kurtosis(x))
```

```
# energy
X_train['x_energy'] = pd.Series(x_set).apply(lambda x: np.sum(x**2)/100)
X_train['y_energy'] = pd.Series(y_set).apply(lambda x: np.sum(x**2)/100)
X_train['z_energy'] = pd.Series(z_set).apply(lambda x: np.sum(x**2/100))

# avg resultant
X_train['avg_result_accl'] = [i.mean() for i in ((pd.Series(x_set)**2 + pd.Series(y_set)**2 +
pd.Series(z_set)**2)**0.5)]

# signal magnitude area
X_train['sma'] =    pd.Series(x_set).apply(lambda x: np.sum(abs(x)/100)) +
pd.Series(y_set).apply(lambda x: np.sum(abs(x)/100)) \
            + pd.Series(z_set).apply(lambda x: np.sum(abs(x)/100))

# converting the signals from time domain to frequency domain using FFT
x_set_fft = pd.Series(x_set).apply(lambda x: np.abs(np.fft.fft(x))[1:51])
y_set_fft = pd.Series(y_set).apply(lambda x: np.abs(np.fft.fft(x))[1:51])
z_set_fft = pd.Series(z_set).apply(lambda x: np.abs(np.fft.fft(x))[1:51])

# Statistical Features on raw x, y and z in frequency domain
# FFT mean
X_train['x_mean_fft'] = pd.Series(x_set_fft).apply(lambda x: x.mean())
X_train['y_mean_fft'] = pd.Series(y_set_fft).apply(lambda x: x.mean())
X_train['z_mean_fft'] = pd.Series(z_set_fft).apply(lambda x: x.mean())

# FFT std dev
X_train['x_std_fft'] = pd.Series(x_set_fft).apply(lambda x: x.std())
X_train['y_std_fft'] = pd.Series(y_set_fft).apply(lambda x: x.std())
X_train['z_std_fft'] = pd.Series(z_set_fft).apply(lambda x: x.std())

# FFT avg absolute diff
X_train['x_aad_fft'] = pd.Series(x_set_fft).apply(lambda x: np.mean(np.absolute(x -
np.mean(x))))
X_train['y_aad_fft'] = pd.Series(y_set_fft).apply(lambda x: np.mean(np.absolute(x -
np.mean(x))))
X_train['z_aad_fft'] = pd.Series(z_set_fft).apply(lambda x: np.mean(np.absolute(x -
np.mean(x))))

# FFT min
X_train['x_min_fft'] = pd.Series(x_set_fft).apply(lambda x: x.min())
X_train['y_min_fft'] = pd.Series(y_set_fft).apply(lambda x: x.min())
X_train['z_min_fft'] = pd.Series(z_set_fft).apply(lambda x: x.min())
```

```
# FFT max
X_train['x_max_fft'] = pd.Series(x_set_fft).apply(lambda x: x.max())
X_train['y_max_fft'] = pd.Series(y_set_fft).apply(lambda x: x.max())
X_train['z_max_fft'] = pd.Series(z_set_fft).apply(lambda x: x.max())

# FFT max-min diff
X_train['x_maxmin_diff_fft'] = X_train['x_max_fft'] - X_train['x_min_fft']
X_train['y_maxmin_diff_fft'] = X_train['y_max_fft'] - X_train['y_min_fft']
X_train['z_maxmin_diff_fft'] = X_train['z_max_fft'] - X_train['z_min_fft']

# FFT median
X_train['x_median_fft'] = pd.Series(x_set_fft).apply(lambda x: np.median(x))
X_train['y_median_fft'] = pd.Series(y_set_fft).apply(lambda x: np.median(x))
X_train['z_median_fft'] = pd.Series(z_set_fft).apply(lambda x: np.median(x))

# FFT median abs dev
X_train['x_mad_fft'] = pd.Series(x_set_fft).apply(lambda x: np.median(np.absolute(x -
np.median(x))))
X_train['y_mad_fft'] = pd.Series(y_set_fft).apply(lambda x: np.median(np.absolute(x -
np.median(x))))
X_train['z_mad_fft'] = pd.Series(z_set_fft).apply(lambda x: np.median(np.absolute(x -
np.median(x))))

# FFT Interquartile range
X_train['x_IQR_fft'] = pd.Series(x_set_fft).apply(lambda x: np.percentile(x, 75) - np.percentile(x,
25))
X_train['y_IQR_fft'] = pd.Series(y_set_fft).apply(lambda x: np.percentile(x, 75) - np.percentile(x,
25))
X_train['z_IQR_fft'] = pd.Series(z_set_fft).apply(lambda x: np.percentile(x, 75) - np.percentile(x,
25))

# FFT values above mean
X_train['x_above_mean_fft'] = pd.Series(x_set_fft).apply(lambda x: np.sum(x > x.mean()))
X_train['y_above_mean_fft'] = pd.Series(y_set_fft).apply(lambda x: np.sum(x > x.mean()))
X_train['z_above_mean_fft'] = pd.Series(z_set_fft).apply(lambda x: np.sum(x > x.mean()))

# FFT number of peaks
X_train['x_peak_count_fft'] = pd.Series(x_set_fft).apply(lambda x: len(find_peaks(x)[0]))
X_train['y_peak_count_fft'] = pd.Series(y_set_fft).apply(lambda x: len(find_peaks(x)[0]))
X_train['z_peak_count_fft'] = pd.Series(z_set_fft).apply(lambda x: len(find_peaks(x)[0]))

# FFT skewness
X_train['x_skewness_fft'] = pd.Series(x_set_fft).apply(lambda x: stats.skew(x))
X_train['y_skewness_fft'] = pd.Series(y_set_fft).apply(lambda x: stats.skew(x))
```

```
X_train['z_skewness_fft'] = pd.Series(z_set_fft).apply(lambda x: stats.skew(x))

# FFT kurtosis
X_train['x_kurtosis_fft'] = pd.Series(x_set_fft).apply(lambda x: stats.kurtosis(x))
X_train['y_kurtosis_fft'] = pd.Series(y_set_fft).apply(lambda x: stats.kurtosis(x))
X_train['z_kurtosis_fft'] = pd.Series(z_set_fft).apply(lambda x: stats.kurtosis(x))

# FFT energy
X_train['x_energy_fft'] = pd.Series(x_set_fft).apply(lambda x: np.sum(x**2)/50)
X_train['y_energy_fft'] = pd.Series(y_set_fft).apply(lambda x: np.sum(x**2)/50)
X_train['z_energy_fft'] = pd.Series(z_set_fft).apply(lambda x: np.sum(x**2/50))

# FFT avg resultant
X_train['avg_result_accl_fft'] = [i.mean() for i in ((pd.Series(x_set_fft)**2 +
pd.Series(y_set_fft)**2 + pd.Series(z_set_fft)**2)**0.5)]

# FFT Signal magnitude area
X_train['sma_fft'] = pd.Series(x_set_fft).apply(lambda x: np.sum(abs(x)/50)) +
pd.Series(y_set_fft).apply(lambda x: np.sum(abs(x)/50)) \
                + pd.Series(z_set_fft).apply(lambda x: np.sum(abs(x)/50))

x_set_test = []
y_set_test = []
z_set_test = []
test_labels = []

for i in range(0, test.shape[0] - window_size, step_size):
    xval = test['x-axis'].values[i: i + 100]
    yval = test['y-axis'].values[i: i + 100]
    zval = test['z-axis'].values[i: i + 100]
    label_test = stats.mode(test['activity'][i: i + 100])[0][0]

    x_set_test.append(xval)
    y_set_test.append(yval)
    z_set_test.append(zval)
    test_labels.append(label_test)

X_test = pd.DataFrame()

# mean
X_test['x_mean'] = pd.Series(x_set_test).apply(lambda x: x.mean())
X_test['y_mean'] = pd.Series(y_set_test).apply(lambda x: x.mean())
X_test['z_mean'] = pd.Series(z_set_test).apply(lambda x: x.mean())
```

```
# std dev
X_test['x_std'] = pd.Series(x_set_test).apply(lambda x: x.std())
X_test['y_std'] = pd.Series(y_set_test).apply(lambda x: x.std())
X_test['z_std'] = pd.Series(z_set_test).apply(lambda x: x.std())

# avg absolute diff
X_test['x_aad'] = pd.Series(x_set_test).apply(lambda x: np.mean(np.absolute(x - np.mean(x))))
X_test['y_aad'] = pd.Series(y_set_test).apply(lambda x: np.mean(np.absolute(x - np.mean(x))))
X_test['z_aad'] = pd.Series(z_set_test).apply(lambda x: np.mean(np.absolute(x - np.mean(x))))

# min
X_test['x_min'] = pd.Series(x_set_test).apply(lambda x: x.min())
X_test['y_min'] = pd.Series(y_set_test).apply(lambda x: x.min())
X_test['z_min'] = pd.Series(z_set_test).apply(lambda x: x.min())

# max
X_test['x_max'] = pd.Series(x_set_test).apply(lambda x: x.max())
X_test['y_max'] = pd.Series(y_set_test).apply(lambda x: x.max())
X_test['z_max'] = pd.Series(z_set_test).apply(lambda x: x.max())

# max-min diff
X_test['x_maxmin_diff'] = X_test['x_max'] - X_test['x_min']
X_test['y_maxmin_diff'] = X_test['y_max'] - X_test['y_min']
X_test['z_maxmin_diff'] = X_test['z_max'] - X_test['z_min']

# median
X_test['x_median'] = pd.Series(x_set_test).apply(lambda x: np.median(x))
X_test['y_median'] = pd.Series(y_set_test).apply(lambda x: np.median(x))
X_test['z_median'] = pd.Series(z_set_test).apply(lambda x: np.median(x))

# median abs dev
X_test['x_mad'] = pd.Series(x_set_test).apply(lambda x: np.median(np.absolute(x -
np.median(x))))
X_test['y_mad'] = pd.Series(y_set_test).apply(lambda x: np.median(np.absolute(x -
np.median(x))))
X_test['z_mad'] = pd.Series(z_set_test).apply(lambda x: np.median(np.absolute(x -
np.median(x))))

# interquartile range
X_test['x_IQR'] = pd.Series(x_set_test).apply(lambda x: np.percentile(x, 75) - np.percentile(x,
25))
X_test['y_IQR'] = pd.Series(y_set_test).apply(lambda x: np.percentile(x, 75) - np.percentile(x,
25))
```

```
X_test['z_IQR'] = pd.Series(z_set_test).apply(lambda x: np.percentile(x, 75) - np.percentile(x,
25))

# negtive count
X_test['x_neg_count'] = pd.Series(x_set_test).apply(lambda x: np.sum(x < 0))
X_test['y_neg_count'] = pd.Series(y_set_test).apply(lambda x: np.sum(x < 0))
X_test['z_neg_count'] = pd.Series(z_set_test).apply(lambda x: np.sum(x < 0))

# positive count
X_test['x_pos_count'] = pd.Series(x_set_test).apply(lambda x: np.sum(x > 0))
X_test['y_pos_count'] = pd.Series(y_set_test).apply(lambda x: np.sum(x > 0))
X_test['z_pos_count'] = pd.Series(z_set_test).apply(lambda x: np.sum(x > 0))

# values above mean
X_test['x_above_mean'] = pd.Series(x_set_test).apply(lambda x: np.sum(x > x.mean()))
X_test['y_above_mean'] = pd.Series(y_set_test).apply(lambda x: np.sum(x > x.mean()))
X_test['z_above_mean'] = pd.Series(z_set_test).apply(lambda x: np.sum(x > x.mean()))

# number of peaks
X_test['x_peak_count'] = pd.Series(x_set_test).apply(lambda x: len(find_peaks(x)[0]))
X_test['y_peak_count'] = pd.Series(y_set_test).apply(lambda x: len(find_peaks(x)[0]))
X_test['z_peak_count'] = pd.Series(z_set_test).apply(lambda x: len(find_peaks(x)[0]))

# skewness
X_test['x_skewness'] = pd.Series(x_set_test).apply(lambda x: stats.skew(x))
X_test['y_skewness'] = pd.Series(y_set_test).apply(lambda x: stats.skew(x))
X_test['z_skewness'] = pd.Series(z_set_test).apply(lambda x: stats.skew(x))

# kurtosis
X_test['x_kurtosis'] = pd.Series(x_set_test).apply(lambda x: stats.kurtosis(x))
X_test['y_kurtosis'] = pd.Series(y_set_test).apply(lambda x: stats.kurtosis(x))
X_test['z_kurtosis'] = pd.Series(z_set_test).apply(lambda x: stats.kurtosis(x))

# energy
X_test['x_energy'] = pd.Series(x_set_test).apply(lambda x: np.sum(x**2)/100)
X_test['y_energy'] = pd.Series(y_set_test).apply(lambda x: np.sum(x**2)/100)
X_test['z_energy'] = pd.Series(z_set_test).apply(lambda x: np.sum(x**2/100))

# avg resultant
X_test['avg_result_accl'] = [i.mean() for i in ((pd.Series(x_set_test)**2 +
pd.Series(y_set_test)**2 + pd.Series(z_set_test)**2)**0.5)]

# signal magnitude area
```

```
X_test['sma'] =    pd.Series(x_set_test).apply(lambda x: np.sum(abs(x)/100)) +
pd.Series(y_set_test).apply(lambda x: np.sum(abs(x)/100)) \
            + pd.Series(z_set_test).apply(lambda x: np.sum(abs(x)/100))

# converting the signals from time domain to frequency domain using FFT
x_set_fft_test = pd.Series(x_set_test).apply(lambda x: np.abs(np.fft.fft(x))[1:51])
y_set_fft_test = pd.Series(y_set_test).apply(lambda x: np.abs(np.fft.fft(x))[1:51])
z_set_fft_test = pd.Series(z_set_test).apply(lambda x: np.abs(np.fft.fft(x))[1:51])

# Statistical Features on raw x, y and z in frequency domain
# FFT mean
X_test['x_mean_fft'] = pd.Series(x_set_fft_test).apply(lambda x: x.mean())
X_test['y_mean_fft'] = pd.Series(y_set_fft_test).apply(lambda x: x.mean())
X_test['z_mean_fft'] = pd.Series(z_set_fft_test).apply(lambda x: x.mean())

# FFT std dev
X_test['x_std_fft'] = pd.Series(x_set_fft_test).apply(lambda x: x.std())
X_test['y_std_fft'] = pd.Series(y_set_fft_test).apply(lambda x: x.std())
X_test['z_std_fft'] = pd.Series(z_set_fft_test).apply(lambda x: x.std())

# FFT avg absolute diff
X_test['x_aad_fft'] = pd.Series(x_set_fft_test).apply(lambda x: np.mean(np.absolute(x -
np.mean(x))))
X_test['y_aad_fft'] = pd.Series(y_set_fft_test).apply(lambda x: np.mean(np.absolute(x -
np.mean(x))))
X_test['z_aad_fft'] = pd.Series(z_set_fft_test).apply(lambda x: np.mean(np.absolute(x -
np.mean(x))))

# FFT min
X_test['x_min_fft'] = pd.Series(x_set_fft_test).apply(lambda x: x.min())
X_test['y_min_fft'] = pd.Series(y_set_fft_test).apply(lambda x: x.min())
X_test['z_min_fft'] = pd.Series(z_set_fft_test).apply(lambda x: x.min())

# FFT max
X_test['x_max_fft'] = pd.Series(x_set_fft_test).apply(lambda x: x.max())
X_test['y_max_fft'] = pd.Series(y_set_fft_test).apply(lambda x: x.max())
X_test['z_max_fft'] = pd.Series(z_set_fft_test).apply(lambda x: x.max())

# FFT max-min diff
X_test['x_maxmin_diff_fft'] = X_test['x_max_fft'] - X_test['x_min_fft']
X_test['y_maxmin_diff_fft'] = X_test['y_max_fft'] - X_test['y_min_fft']
X_test['z_maxmin_diff_fft'] = X_test['z_max_fft'] - X_test['z_min_fft']

# FFT median
```

```
X_test['x_median_fft'] = pd.Series(x_set_fft_test).apply(lambda x: np.median(x))
X_test['y_median_fft'] = pd.Series(y_set_fft_test).apply(lambda x: np.median(x))
X_test['z_median_fft'] = pd.Series(z_set_fft_test).apply(lambda x: np.median(x))

# FFT median abs dev
X_test['x_mad_fft'] = pd.Series(x_set_fft_test).apply(lambda x: np.median(np.absolute(x -
np.median(x))))
X_test['y_mad_fft'] = pd.Series(y_set_fft_test).apply(lambda x: np.median(np.absolute(x -
np.median(x))))
X_test['z_mad_fft'] = pd.Series(z_set_fft_test).apply(lambda x: np.median(np.absolute(x -
np.median(x))))

# FFT Interquartile range
X_test['x_IQR_fft'] = pd.Series(x_set_fft_test).apply(lambda x: np.percentile(x, 75) -
np.percentile(x, 25))
X_test['y_IQR_fft'] = pd.Series(y_set_fft_test).apply(lambda x: np.percentile(x, 75) -
np.percentile(x, 25))
X_test['z_IQR_fft'] = pd.Series(z_set_fft_test).apply(lambda x: np.percentile(x, 75) -
np.percentile(x, 25))

# FFT values above mean
X_test['x_above_mean_fft'] = pd.Series(x_set_fft_test).apply(lambda x: np.sum(x > x.mean()))
X_test['y_above_mean_fft'] = pd.Series(y_set_fft_test).apply(lambda x: np.sum(x > x.mean()))
X_test['z_above_mean_fft'] = pd.Series(z_set_fft_test).apply(lambda x: np.sum(x > x.mean()))

# FFT number of peaks
X_test['x_peak_count_fft'] = pd.Series(x_set_fft_test).apply(lambda x: len(find_peaks(x)[0]))
X_test['y_peak_count_fft'] = pd.Series(y_set_fft_test).apply(lambda x: len(find_peaks(x)[0]))
X_test['z_peak_count_fft'] = pd.Series(z_set_fft_test).apply(lambda x: len(find_peaks(x)[0]))

# FFT skewness
X_test['x_skewness_fft'] = pd.Series(x_set_fft_test).apply(lambda x: stats.skew(x))
X_test['y_skewness_fft'] = pd.Series(y_set_fft_test).apply(lambda x: stats.skew(x))
X_test['z_skewness_fft'] = pd.Series(z_set_fft_test).apply(lambda x: stats.skew(x))

# FFT kurtosis
X_test['x_kurtosis_fft'] = pd.Series(x_set_fft_test).apply(lambda x: stats.kurtosis(x))
X_test['y_kurtosis_fft'] = pd.Series(y_set_fft_test).apply(lambda x: stats.kurtosis(x))
X_test['z_kurtosis_fft'] = pd.Series(z_set_fft_test).apply(lambda x: stats.kurtosis(x))

# FFT energy
X_test['x_energy_fft'] = pd.Series(x_set_fft_test).apply(lambda x: np.sum(x**2)/50)
X_test['y_energy_fft'] = pd.Series(y_set_fft_test).apply(lambda x: np.sum(x**2)/50)
X_test['z_energy_fft'] = pd.Series(z_set_fft_test).apply(lambda x: np.sum(x**2/50))
```

```
# FFT avg resultant
X_test['avg_result_accl_fft'] = [i.mean() for i in ((pd.Series(x_set_fft_test)**2 +
pd.Series(y_set_fft_test)**2 + pd.Series(z_set_fft_test)**2)**0.5)]

# FFT Signal magnitude area
X_test['sma_fft'] = pd.Series(x_set_fft_test).apply(lambda x: np.sum(abs(x)/50)) +
pd.Series(y_set_fft_test).apply(lambda x: np.sum(abs(x)/50)) \
                + pd.Series(z_set_fft_test).apply(lambda x: np.sum(abs(x)/50))

y_train = np.array(train_labels)
y_test = np.array(test_labels)

"""----------------------------------------------------



DATA PREPRAPERATION IS DONE HERE!




----------------------------------------------------
"""

X_train.head()
type(X_train)
print(X_train.dtypes)

X_train_list = X_train.values.tolist()
X_test_list = X_test.values.tolist()
y_test_list = y_test.tolist()

y_train_list = y_train.tolist()

print(type(X_test.head()))

print(y_test)
```

```
print(y_train)

#--------------------KNN---------------------------


import scipy.spatial
from collections import Counter
from sklearn import datasets
from sklearn.model_selection import train_test_split

class KNN:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def distance(self, X1, X2):
        distance = scipy.spatial.distance.euclidean(X1, X2)

    def predict(self, X_test):
        final_output = []

        for i in range(len(X_test)):
            d = []
            votes = []

            for j in range(len(self.X_train)):
                #print(j)
                # print(self.X_train[j+1])
                dist = scipy.spatial.distance.euclidean(self.X_train[j] , X_test[i])
                d.append([dist, j])
            d.sort()
            d = d[0:self.k]
            for _d, j in d:
                votes.append(self.y_train[j])
            ans = Counter(votes).most_common(1)[0][0]
            final_output.append(ans)

        return final_output

    def score(self, X_test, y_test):
```

```
        predictions = self.predict(X_test)
        # print(predictions)
        # print(predictions == y_test)

        """for i in predictions:
          print(i, end= ' ')
        predictions == y_test"""
        temp = 0
        for i in range(len(y_test)):
            if predictions[i]== y_test[i]:
                temp = temp+1
        result = temp/len(y_test)

        return result

clf = KNN(136)
clf.fit(X_train_list, y_train_list)
print("KNN Score =")
clf.score(X_test_list, y_test_list)

from sklearn.neighbors import KNeighborsClassifier
temp2 = 0
model = KNeighborsClassifier(n_neighbors=136)

# Train the model using the training sets
model.fit(X_train_list,y_train_list)

#Predict Output
predicted= model.predict(X_test_list) # 0:Overcast, 2:Mild
#print(predicted)
for i in range(len(y_test_list)):
    if predicted[i]== y_test_list[i]:
        temp2 = temp2+1
result = temp2/len(y_test_list)
print(result)

error_rate = []
for i in range(1,400):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
```

```
plt.plot(range(1,400),error_rate,color='blue', linestyle='dashed',
       marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K =",error_rate.index(min(error_rate)))

### ---------------------------------------- Decision Tree ----------------------------

# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train_list,y_train_list)

#Predict the response for test dataset
y_pred = clf.predict(X_test_list)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test_list, y_pred))

"""The above code is Decision Tree classification with scikit library.After that I am modeliing
from stracth.

"""

## ---------- node class ---------
class Node():
   def __init__(self, feature_index=None, threshold=None, left=None, right=None,
info_gain=None, value=None):
      ''' constructor '''

      # for decision node
      self.feature_index = feature_index
      self.threshold = threshold
      self.left = left
      self.right = right
      self.info_gain = info_gain

      # for leaf node
```

```
        self.value = value

class DecisionTreeClassifier():
    def __init__(self, min_samples_split=2, max_depth=2):
        ''' constructor '''

        # initialize the root of the tree
        self.root = None

        # stopping conditions
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth

    def build_tree(self, dataset, curr_depth=0):
        ''' recursive function to build the tree '''

        X, Y = dataset[:,:-1], dataset[:,-1]
        num_samples, num_features = np.shape(X)
        # print(num_samples)
        # print(num_features)
        # split until stopping conditions are met
        if num_samples>=self.min_samples_split and curr_depth<=self.max_depth:
            # find the best split
            best_split = self.get_best_split(dataset, num_samples, num_features)
            # check if information gain is positive
            if best_split["info_gain"]>0:
                # recur left
                left_subtree = self.build_tree(best_split["dataset_left"], curr_depth+1)
                # recur right
                right_subtree = self.build_tree(best_split["dataset_right"], curr_depth+1)
                # return decision node
                return Node(best_split["feature_index"], best_split["threshold"],
                            left_subtree, right_subtree, best_split["info_gain"])

        # compute leaf node
        leaf_value = self.calculate_leaf_value(Y)
        # return leaf node
        return Node(value=leaf_value)

    def get_best_split(self, dataset, num_samples, num_features):
        ''' function to find the best split '''

        # dictionary to store the best split
        best_split = {}
```

```
        max_info_gain = -float("inf")

        # loop over all the features
        for feature_index in range(num_features):
            print(feature_index)
            feature_values = dataset[:, feature_index]
            possible_thresholds = np.unique(feature_values)
            # loop over all the feature values present in the data
            for threshold in possible_thresholds:
                # get current split
                dataset_left, dataset_right = self.split(dataset, feature_index, threshold)
                # check if childs are not null
                if len(dataset_left)>0 and len(dataset_right)>0:
                    y, left_y, right_y = dataset[:, -1], dataset_left[:, -1], dataset_right[:, -1]
                    # compute information gain
                    curr_info_gain = self.information_gain(y, left_y, right_y, "entropy")
                    # update the best split if needed
                    if curr_info_gain>max_info_gain:
                        best_split["feature_index"] = feature_index
                        best_split["threshold"] = threshold
                        best_split["dataset_left"] = dataset_left
                        best_split["dataset_right"] = dataset_right
                        best_split["info_gain"] = curr_info_gain
                        max_info_gain = curr_info_gain

        # return best split
        return best_split

    def split(self, dataset, feature_index, threshold):
        ''' function to split the data '''

        dataset_left = np.array([row for row in dataset if row[feature_index]<=threshold])
        dataset_right = np.array([row for row in dataset if row[feature_index]>threshold])
        return dataset_left, dataset_right

    def information_gain(self, parent, l_child, r_child, mode="entropy"):
        ''' function to compute information gain '''

        weight_l = len(l_child) / len(parent)
        weight_r = len(r_child) / len(parent)
        if mode=="gini":
            gain = self.gini_index(parent) - (weight_l*self.gini_index(l_child) +
weight_r*self.gini_index(r_child))
        else:
```

```
        gain = self.entropy(parent) - (weight_l*self.entropy(l_child) +
weight_r*self.entropy(r_child))
        return gain

    def entropy(self, y):
        ''' function to compute entropy '''

        class_labels = np.unique(y)
        entropy = 0
        for cls in class_labels:
            p_cls = len(y[y == cls]) / len(y)
            entropy += -p_cls * np.log2(p_cls)
        return entropy

    def gini_index(self, y):

        return "warning gini"

    def calculate_leaf_value(self, Y):
        ''' function to compute leaf node '''

        Y = list(Y)
        return max(Y, key=Y.count)

    def print_tree(self, tree=None, indent=" "):
        ''' function to print the tree '''

        if not tree:
            tree = self.root

        if tree.value is not None:
            print(tree.value)

        else:
            print("X_"+str(tree.feature_index), "<=", tree.threshold, "?", tree.info_gain)
            print("%sleft:" % (indent), end="")
            self.print_tree(tree.left, indent + indent)
            print("%sright:" % (indent), end="")
            self.print_tree(tree.right, indent + indent)

    def fit(self, X, Y):
        ''' function to train the tree '''

        dataset = np.concatenate((X, Y), axis=1)
```

```
        self.root = self.build_tree(dataset)

    def predict(self, X):
        ''' function to predict new dataset '''

        preditions = [self.make_prediction(x, self.root) for x in X]
        return preditions

    def make_prediction(self, x, tree):
        ''' function to predict a single data point '''

        if tree.value!=None: return tree.value
        feature_val = x[tree.feature_index]
        # print(type(feature_val))
        # print(type(tree.threshold))
        new_threshold = tree.threshold.astype(np.float)
        if feature_val<=new_threshold:
            return self.make_prediction(x, tree.left)
        else:
            return self.make_prediction(x, tree.right)

X_train_numpy = X_train.to_numpy()

classifier = DecisionTreeClassifier(min_samples_split=3, max_depth=3)
classifier.fit(X_train_numpy[:1000],np.array([y_train[:1000]]).T)
classifier.print_tree()
X_test_numpy = X_test.to_numpy()
Y_pred = classifier.predict(X_test_numpy)
from sklearn.metrics import accuracy_score
accuracy_score(y_test, Y_pred)

"""## Logistic Regression"""

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# standardization
scaler = StandardScaler()
scaler.fit(X_train)
X_train_data_lr = scaler.transform(X_train)
X_test_data_lr = scaler.transform(X_test)
# logistic regression model
lr = LogisticRegression(random_state = 21)
```

```
lr.fit(X_train_data_lr, y_train)
y_pred = lr.predict(X_test_data_lr)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\n ----------Classification Report----------\n")
print(classification_report(y_test, y_pred))

from sklearn.base import BaseEstimator, ClassifierMixin
from scipy.special import softmax
from sklearn.preprocessing import OneHotEncoder

class Logistic_Regression(BaseEstimator, ClassifierMixin):
    def __init__(self):
        self.params = {}
        self.n_features = None
        self.n_class = None
        self.encoder = OneHotEncoder(sparse=False)
        self.verbose = 0

    def init_coeff(self, X, y):

        self.n_features = X.shape[1]

        self.params['coef'] = np.random.randn(self.n_features, self.n_class)

        self.params['intercept'] = np.random.randn(1, self.n_class)
        if self.verbose:
            print("[INFO] Initialized parameters.")
            print(f"Shape of coefficient matrix: {self.params['coef'].shape}")
            print(f"Shape of intercept matrix: {self.params['intercept'].shape}")

    def logistic_coeff(self, X, y=None):

        if 'coef' not in self.params and y is None:

            raise Exception("This LogisticRegression instance is not fitted yet." +
                            "Call 'fit' with appropriate arguments before using this estimator.")
        elif 'coef' not in self.params and y is not None:
            print("[INFO] The model is not fitted yet. Using random parameters.")
            self.init_coeff(X, y)
        return X @ self.params['coef'] + self.params['intercept']

    def prob_prediction(self, X, y=None):

        if y is not None:
```

```
        logits = self.logistic_coeff(X, y)
    else:
        logits = self.logistic_coeff(X)

    if self.n_class == 1:

        return 1 / (1 + np.exp(-logits))

    mx = np.max(logits, axis=-1, keepdims=True)
    numerator = np.exp(logits - mx)
    denominator = np.sum(numerator, axis=-1, keepdims=True)
    return numerator / denominator

def fit(self, X, y, learning_rate=0.05, iterations=1000, verbose=0):
    self.verbose = verbose

    if isinstance(X, pd.DataFrame):

        X = X.values
        print("X=",X)
    else :
        print(X)
    if isinstance(y, pd.DataFrame):
        y = y.values
        print("Y=",y)
     else :
        print(y)


    numberOfclasses = len(np.unique(y.flatten()))
    print("n classes=",numberOfclasses)

    self.n_class = numberOfclasses if numberOfclasses > 2 else 1
    if self.n_class > 2:

        y = self.encoder.fit_transform(y)


    self.init_coeff(X, y)
    m = X.shape[0]

    if self.verbose:
        print("[INFO] Training ...")
```

```
    for i in range(1, iterations + 1):

        if self.n_class == 1:

            y_proba = self.prob_prediction(X)

            loss = - (1 / m) * np.sum(y * np.log(y_proba) \
                    + (1 - y) * np.log(1 - y_proba))


            dW = (1 / m) * (X.T @ (y_proba - y))
            db = (1 / m) * np.sum(y_proba - y)


        else:

            y_proba = self.prob_prediction(X, y)

            loss = - (1 / m) * np.sum(y * np.log(y_proba))

            dW = (1 / m) * (X.T @ (y_proba - y))
            db = (1 / m) * np.sum((y_proba - y), axis=0, keepdims=True)

        self.params['coef'] -= (learning_rate * dW)
        self.params['intercept'] -= (learning_rate * db)

        if self.verbose and (i == 1 or i % 100 == 0):
            print(f"\nIteration {i}/{iterations}")
            print("--" * 12)
            print(f"Loss: {loss}")
            print(f"Coefficient:\n{self.params['coef']}")
            print(f"Intercept:\n{self.params['intercept']}")

def predict(self, X, threshold=0.5):
    y_proba = self.prob_prediction(X)
    if self.n_class == 1:
        y_pred = np.where(y_proba > threshold, 1, 0)
    else:

        y_pred = np.argmax(y_proba, axis=1)
    return y_pred

def predict_score(self, X, y):
    from sklearn.metrics import accuracy_score
```

```
        y_pred = self.predict(X)

        return accuracy_score(y.reshape(-1), y_pred.reshape(-1))

def softmax(x):

    mx = np.max(x, axis=-1, keepdims=True)
    print(f"{mx = }")
    print(f"{x - mx = }")
    numerator = np.exp(x - mx)
    denominator = np.sum(numerator, axis=-1, keepdims=True)
    print(f"{denominator = }")
    return numerator / denominator

result = softmax(X_train_numpy)

print(f"{result = }")

print("Sum of probabilities =", np.sum(result, axis=1, keepdims=True))

model = Logistic_Regression()


model.fit(X_train_numpy,(np.array([y_train]).T), learning_rate=0.05, iterations=1500,
verbose=0)

log_reg = LogisticRegression(penalty='none', solver='sag', multi_class='multinomial',
random_state=42)
log_reg.fit(X_train_numpy,(np.array([y_train]).T))
print("Result in accuracy:")
print(f"My implementation\t: {model.predict_score(X_train_numpy,(np.array([y_train]).T))}")
print(f"Sklearn implementation\t: {log_reg.score(X_train_numpy,(np.array([y_train]).T))}")
```