Comparative Study on ML Classification Techniques for Human Activity Recognition

ITCS 3156 Final Project Report

Ajay D. Sudhir

Table of Contents

1. Introduction

a. Problem Statement

Human Activity Recognition (HAR) through the use of smartphone sensors aims to determine the motion of users in real-time motion tracking applications. In this project, I aim to address the problem of recognizing six forms of human activity such as walking, walking upstairs, walking downstairs, sitting, standing, and laying down. Through the use of various Machine Learning classification models I have tried to predict human activities based on the extracted feature values from the University of California, Irvine's Machine Learning dataset. The goal of this project is to gauge the overall effectiveness of these classification models on this HAR dataset.

b. Motivation

The automation of human activities has various applications in multiple sectors of the world, such as healthcare, fitness support, and elderly care. Another reason why this problem is important is because the data has been collected from a regular smartphone, meaning that it is easy to replicate. I was also motivated by the opportunity to put my new knowledge learned in this class about classification models to the test through this highly complex dataset. HAR supports applications like fall detection, fitness tracking, and personalized elderly care which can enable safer living for such individuals.

c. Approach

This project applies Exploratory Data Analysis and Data Preprocessing to gain a better understanding of the dataset. I have also standardized the data during the preprocessing to make sure the values for all 561 features (excluding the target label and participant ID) are in the same range. This also allowed me to easily determine the training, validation, and test data I will be passing to my models. Next, I applied multiple machine learning models for classification such as k-Nearest Neighbors, Naive Bayes, and Logistic Regression. I finally compare results across the three models and analyze the trade-offs in terms of accuracy.

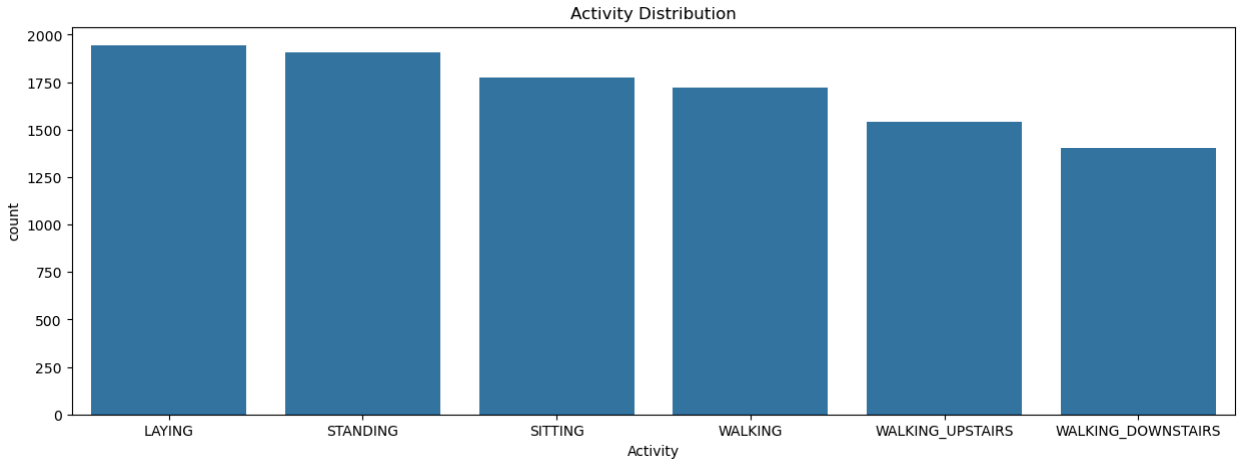
2. Data

a. Introducing the Data

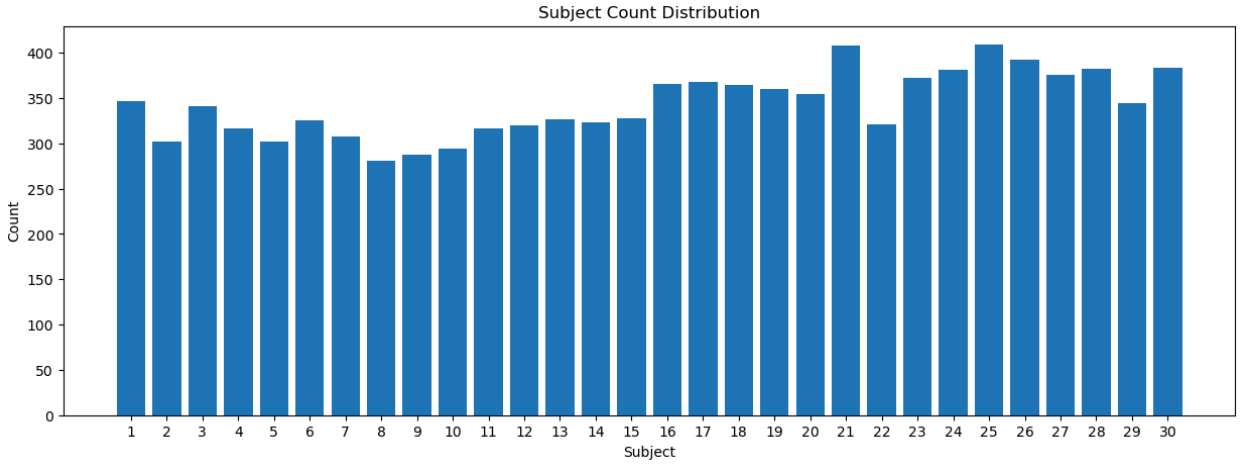
The Human Activity Recognition (HAR) database was taken from the recording of 30 participants who performed six basic daily human activities while wearing a waist-mounted smartphone (Samsung Galaxy S II) embedded with inertial sensors. The original pre-processed structured dataset can be found in the UC Irvine's Machine Learning Repository, but I found the same dataset in comma-separated value format from their kaggle portal. This dataset consists of 10229 samples (rows) which has 563 features (columns) after combining both training and test dataset. These datasets were preprocessed using noise filters and sampled in fixed-width sliding windows. The 563 features also include one feature for the unique participant ID and another for the target label value, representing the six classification labels.

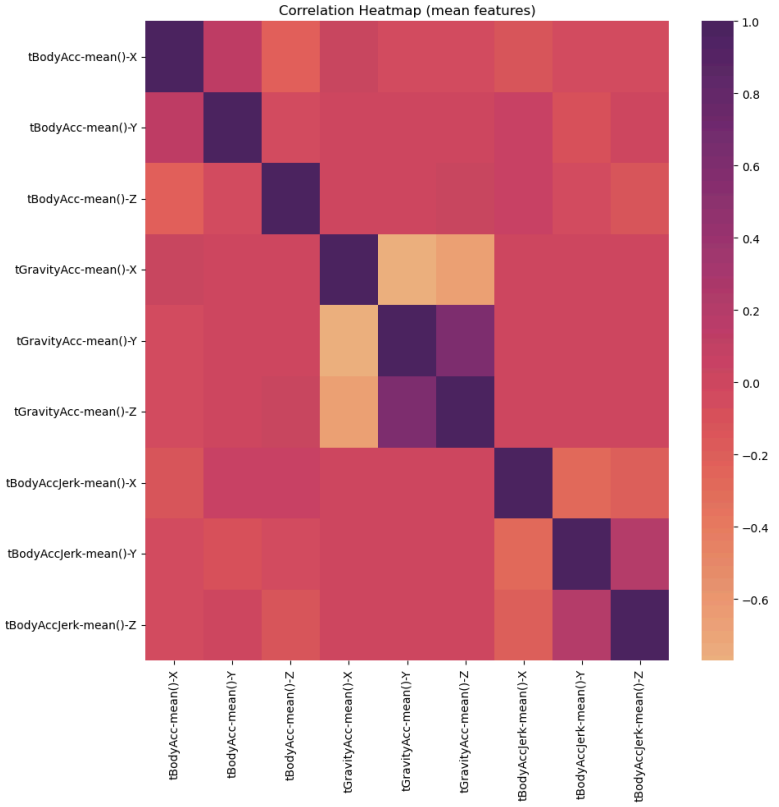
b. Visual Analysis of Data

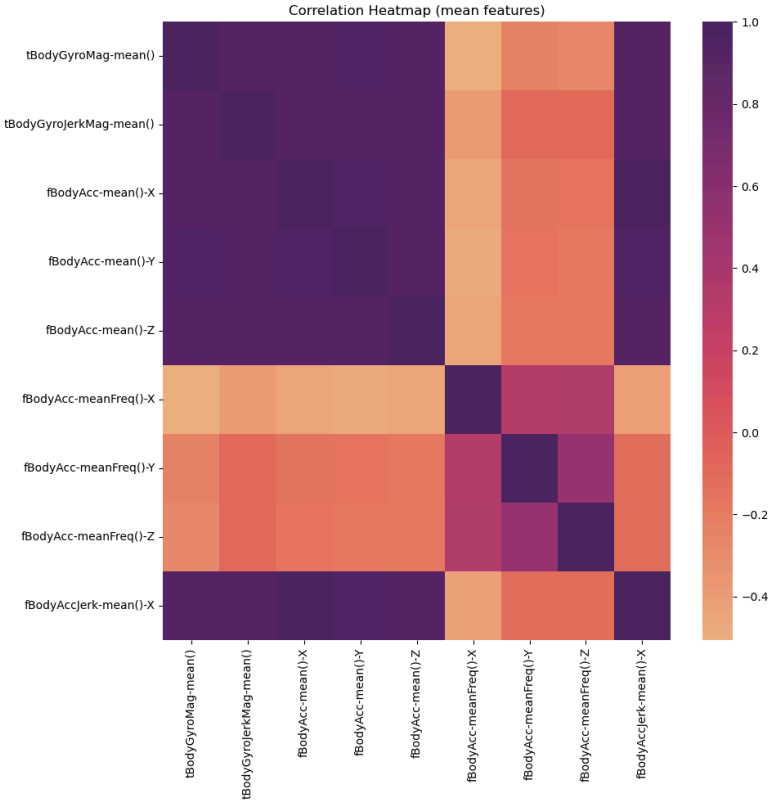
Exploratory Data Analysis was really helpful in gaining a better understanding of the dataset I was working with. I used the Seaborn and Matplotlib libraries to help me visualize the data clearly. First, I wanted to know how the data was distributed across the six target classes, as I assumed that having a certain label with more samples may lead to bias in the classification. Fortunately all the target classes had a similar distribution of values (1500 - 2000) as seen in the graph below, taken from the 'EDA_Preprocessing.ipynb'.



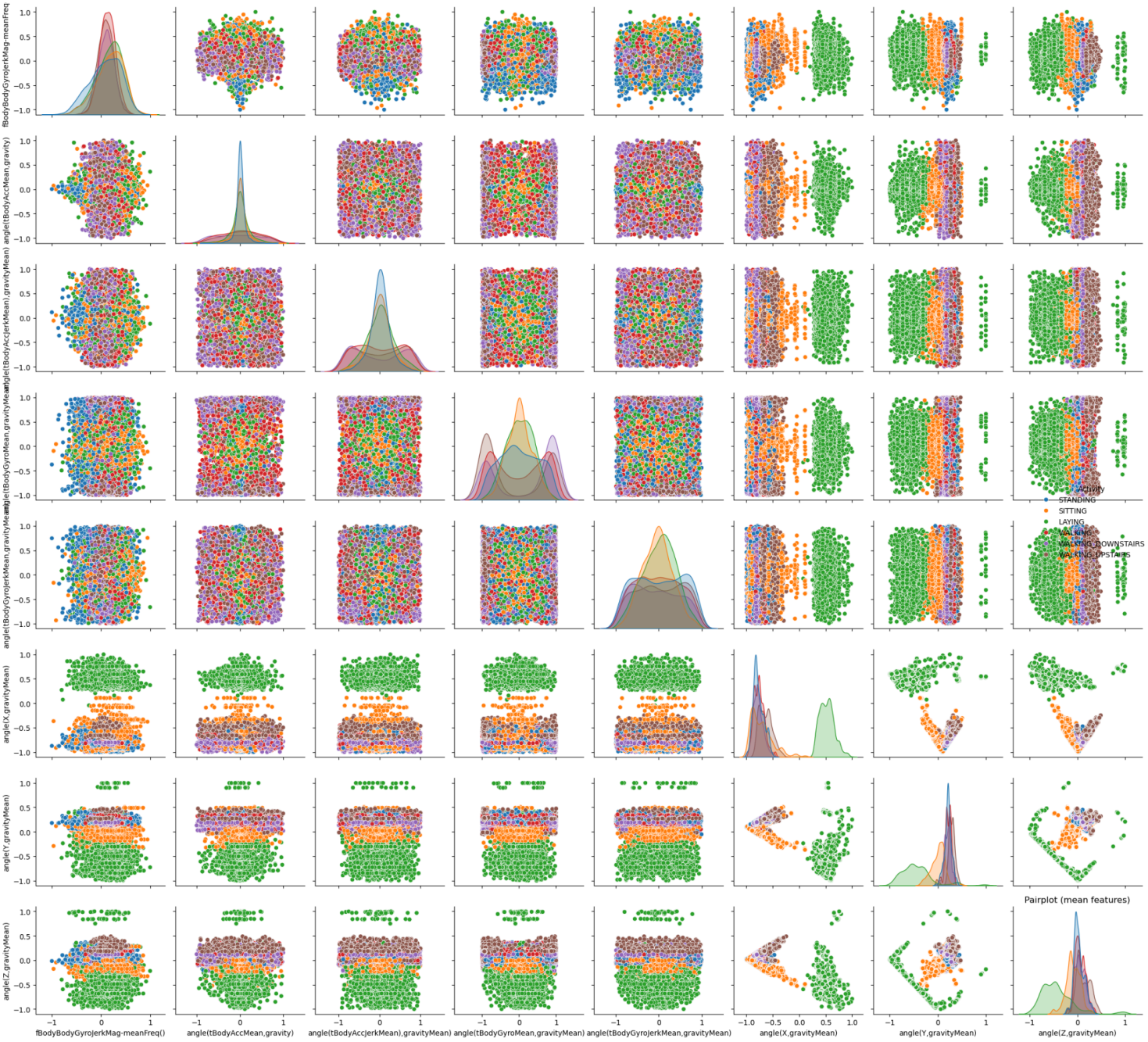
Next, I wanted to check if different participants had huge differences in the number of trials performed for the dataset. This was to determine if certain participants who may act in a particular way and perform certain activities more frequently than others would influence my model later on. As shown below, there was no significant difference between the number of trials for different participants in the dataset, with all values falling between 300 to 400.







Then, I tried to create correlation heatmaps to see which features are related to each other. I choose to do this for the mean() attribute features only, as it would have been repetitive to apply it to all attributes of a single measure such as std(), max(), min(), etc. This reduced the number of plots created as well as helped me to see the high correlations more easily as can be seen in the images above which show that fBodyAcc is highly correlated to tBodyGyroMag and so on.



Finally, I created a pairplot to visualize the distribution of the target classification values in relation to the various features in the dataset. Like the heatmap, I only used the mean attribute values of the features which resulted in nine feature comparisons between 54 mean features out of the total 561 features which is excluding the participant ID and target label. From this plot, it is clear that 'laying' has completely different values compared to the other activities. This separation makes it possible to distinguish between activities, as trails with similar human activities are grouped closer together.

c. Data Preprocessing

Data preprocessing for any dataset involves cleaning the data, handling missing values, removing outliers, encoding categorical features, scaling numerical features, and addressing potential class imbalances. In this case, the dataset collected from the Kaggle portal has already been preprocessed.

The data preprocessing for this project required me to create training, validation, and testing data from the combined data frame, which I had previously created by merging my training and test files from the original data files. I had three separate preprocessing workflows for each of my three machine learning classification algorithms.

The k-NN preprocessing consisted of splitting the combined data into training, validation, and testing datasets in ratios of 80 percent to 20 percent, like in the class assignments. I also applied the standardization function from the Sklearn library that often improves performance in distance-based models, even though it wasn't strictly required for the k-NN algorithm.

The Naive Bayes preprocessing consisted of similar splitting of the combined data into training, validation, and test dataset in an 80:20 ratio. I believed to use this ratio because it would provide sufficient data for training purposes as well as validation and testing. Like before I have done standardization through the Sklearn library for the feature scaling. I also ran into an error initially as the categorical target label needed to be converted to numerical values which has been accomplished through the use of the 'LabelEncoder' of the Sklearn library.

The Logistic Regression preprocessing consisted of a similar splitting of the combined data into training, validation, and test datasets in an 80:20 ratio. In this work, I used the 'OneHotEncoder' from the Sklearn library to convert categorical data into a binary format, as done in the logistic regression assignment. I also standardized the data and manually added the bias column during preprocessing, since the logistic regression model was hardcoded.

The preprocessing works done for this project are meant to be tailored towards each machine learning algorithm in their attempt to classify the human activities from smartphone sensor data.

3. Machine Learning Methods

a. k-Nearest Neighbors

The k-Nearest Neighbors algorithm is a machine learning classification algorithm that is non-parametric in nature. It doesn't require training like other algorithms, as it works based on a distance metric. In essence, the k-NN algorithm classifies data points by using a distance metric to group samples that are closer together into the same class. The 'k' value represents the number of closest neighbors among which the target labels are compared to get the highest vote as the label of the required data sample.

The distance metric used in this project is 'Euclidean Distance'. This is the difference between the x and y values of two data points.

$$d(\mathbf{x}, \mathbf{Y}) = \sqrt{(\mathbf{x} - \mathbf{y}_i)(\mathbf{x} - \mathbf{y}_i)^\top}$$

The formula above performs the euclidean distance calculation and has been implemented in the code for my project as a function, which is called each time distance must be calculated. This is used for the majority voting of the data samples.

The k-NN algorithm works well in low-dimensional datasets but can be computationally expensive for large datasets as it has to compute the distance to every sample at prediction time. The performance is highly impacted by the value of k, so normalization is necessary. As k-NN is very simple and an intuitive algorithm it can be applied to many classification tasks.

b. Naive Bayes

The Naive Bayes algorithm is a probabilistic classification algorithm based on *Bayes' Theorem*. The "naive" assumption that all features are independent given a class label. The Gaussian Naive Bayes also assumes that the features follow a normal distribution for each class. These assumptions enable the algorithm to calculate the probabilities used for classification of the data samples.

- *Bayes' Theorem*:

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)}$$

- *Log Prior Calculation*:

$$P(C_k) = \log\left(\frac{\text{Number of samples for class } k}{\text{Total number of samples}}\right)$$

- *Log Likelihood Calculation*:

$$P(\mathbf{X} \mid C_k) = -\frac{1}{2}\log(\vec{\sigma}_k^2 2\pi) - \frac{1}{2}\left(\frac{\mathbf{X} - \vec{\mu}_k}{\vec{\sigma}_k}\right)^2$$

In *Gaussian Naive Bayes*, the model doesn't truly learn the training data by learning weights like other models. Instead, it calculates the log priors, which represent how likely each class is to appear in the data and stores them. It also computes the mean and standard deviation of each feature for every class, which tells how the features behave in each class under a normal distribution. During testing these statistics are used to compute the log likelihoods which measure how probable it is to observe the features in a given class. These log likelihoods are summed with the priors for the classes to produce the joint log probability. The model then chooses the class with the highest joint log probability as its prediction.

Gaussian Naive Bayes is effective when the independence assumption holds. It is computationally efficient compared to the k-NN and works well on higher dimensional datasets. It is a very popular baseline model for classification and other probabilistic tasks.

c. Logistic Regression

Logistic Regression is a linear model that, unlike the linear regression algorithm, is used to calculate the probability that a given data sample belongs to a particular class for classification purposes. The *sigmoid* function is used for two classes and for multiple classes the *softmax* function is used.

- *Sigmoid* Function:

$$g(\mathbf{z}) = \frac{1}{1+e^{-\mathbf{z}}} = \frac{e^{\mathbf{z}}}{1+e^{\mathbf{z}}}$$

- *Softmax* Function:

$$g(\mathbf{z})_i = \frac{e^{z_i - z_{max}}}{\sum_{k=1}^{K} e^{z_k - z_{max}}}$$

- *Negative Log Likelihood*:

$$NLL(\mathbf{W}) = -\frac{1}{N}\sum_{n=1}^{N} \mathbf{y}_n * \log[f(\mathbf{x}_n; \mathbf{W})]$$

The *sigmoid* function maps any input into a range between 0 and 1. For multi-class classification, the *softmax* function generalizes the sigmoid by transforming the values into a probability relative to the classes. During training, the model uses negative log likelihood as the loss function, which tells it how well the predicted probabilities match the actual class labels. Batch gradient descent is used to minimize this loss between the prediction and true value.

As a result, logistic regression is a great machine learning algorithm that can be used for multiclass classification problems and is faster than k-NN, based on the number of batches and epochs in its hyperparameters.

4. Results

a. Experimental Setup

i. k-Nearest Neighbors

To implement the k-NN algorithm, I started by creating a class with a *fit()* method that simply stores the training data. In the *predict()* method each test sample is taken to compute its distance to all other training samples using a given distance function. Then find the indices of the k closest samples by sorting those distances and selecting the most common one as the predicted label. This process is repeated for each test sample, and all predictions are returned as an array.

ii. Naive Bayes

To implement the Naive Bayes algorithm, I started by creating a class with a *fit()* method that calculates and stores the log priors, means, and standard deviations for each class using the training data. In the *predict()* method each test sample is taken to compute the log-likelihoods of the data given each class using the Gaussian probability formula. Then I add the log priors to get the joint log-likelihoods. The predicted class is the one with the highest joint log-likelihood which is found using an *argmax* function. The final prediction is a column vector of predicted labels for all test samples.

iii. Logistic Regression

To implement the Logistic Regression algorithm using the softmax regression with the mini-batch gradient descent, I started by creating the *SoftmaxRegression* class which initializes the hyperparameters like learning rate, batch size, number of epochs, and a random seed. The *fit()* method trains the model through random initialization of weights. Then during each epoch it takes the training data from mini-batches using the *get_batches()* function to which it computes predictions using the *softmax* function. The gradient of the negative log-likelihood loss is calculated and the weights are updated to minimize the loss function. The *predict()* method makes predictions by using the learned weights on input data to compute softmax probabilities and select the class with the highest probability using the *argmax* function.

b. Algorithm Accuracies

i. k-Nearest Neighbors

The accuracy metric used is based on the number of correct classification divided by the total number of samples. Using this metric and different values for k I got the following accuracies:

- **k = 1 : 0.96**
- **k = 25 : 0.94**
- **k = 50 : 0.93**
- **k = 100 : 0.90**

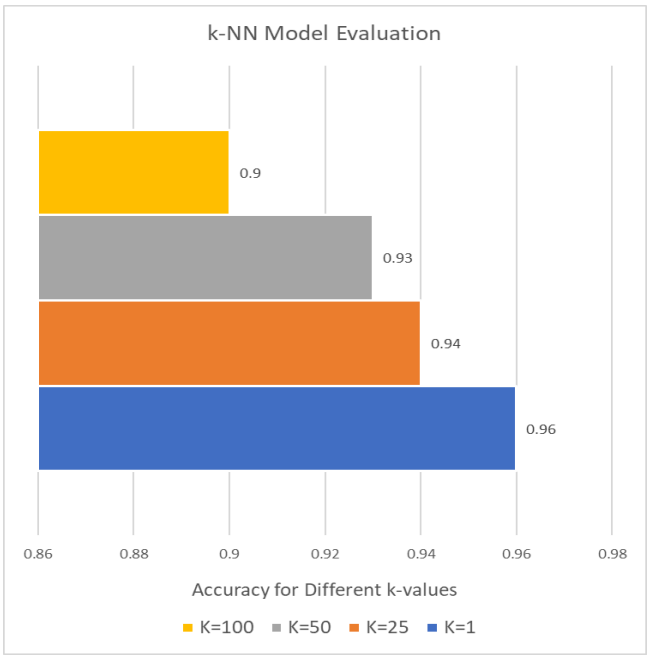
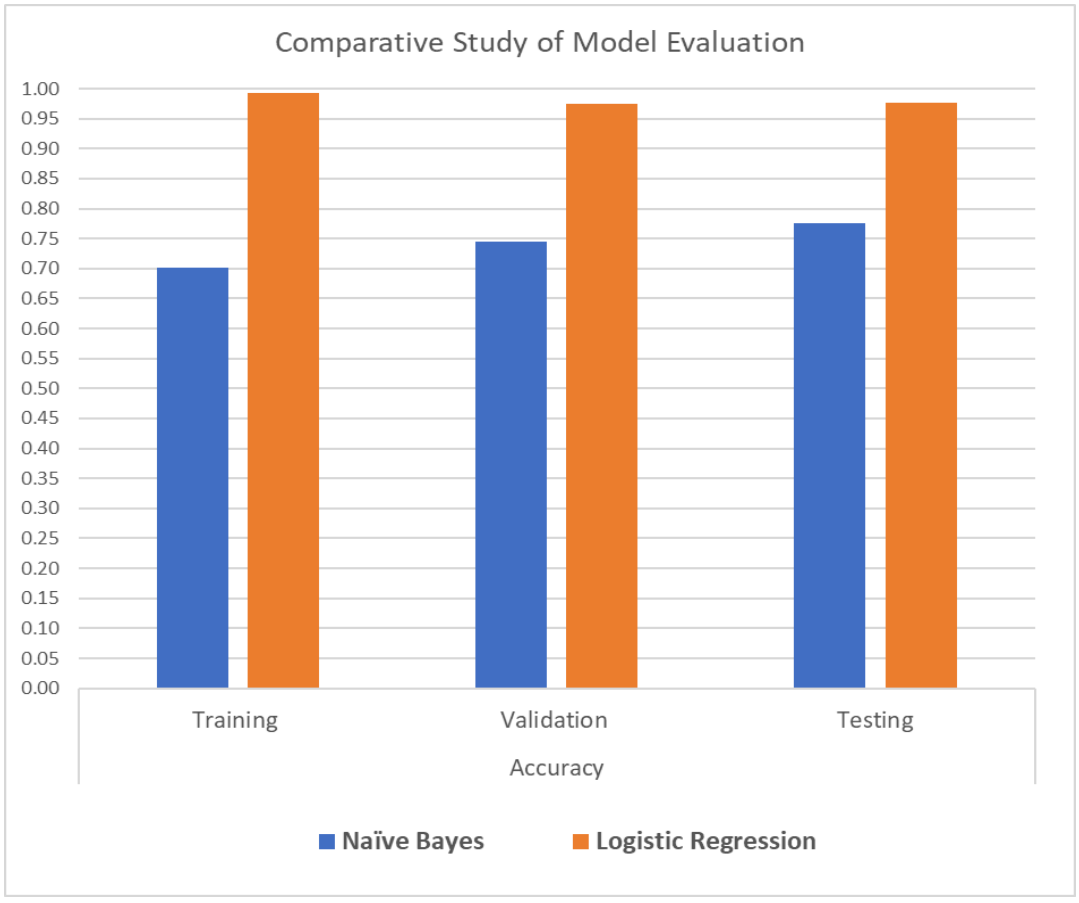


Chart describing the accuracy of k-NN Model for different k-values



Model Evaluation Analysis for Naive Bayes and Logistic Regression

ii. Naive Bayes

The accuracy metric used is from the Sklearn library. The accuracies for Naive Bayes algorithm are the following:

- **Training : 0.7008041268396298**
- **Validation : 0.7445388349514563**
- **Testing : 0.7747572815533981**

iii. Logistic Regression

The accuracy metric used is from the Sklearn library. The accuracies for Logistic Regression algorithm with the hyperparameters *alpha = 0.1*, *batch_size = 64*, *epochs = 100*, and *seed = 42* are the following:

- **Training : 0.9921104536489151**
- **Validation : 0.9745145631067961**
- **Testing : 0.9766990291262136**

c. Model Analysis

i. k-Nearest Neighbors

It performed better at lower values of k as seen when the model achieved its highest test accuracy of 0.96 at k=1. But as k increased the accuracy decreased. This is probably because lower k values result in overfitting caused by low bias and high variance which causes higher accuracy on this dataset as the data points near each other in HAR are mostly of the same class. However as k increases the model generalizes and reduces in accuracy until 0.9 for k = 100. The k-NN does better than Naive Bayes but worse than Logistic Regression.

ii. Naive Bayes

It performed lower compared to the other models with training accuracy around 0.70 and testing accuracy of 0.77. This may be due to the independence assumption, where it assumes that all features are conditionally independent given the class label. Sensor data from accelerometers and gyroscopes in the HAR dataset are highly correlated as seen from the heatmap from the EDA which doesn't mean that the features are truly independent. Naive Bayes does not account for these dependencies therefore has the lowest accuracies among the three models tested as it usually misclassified 'walking upstairs' and 'walking downstairs'.

iii. Logistic Regression

It performed the best among all three models with training accuracy of 0.99 and testing accuracy of 0.97. The use of a small learning rate combined with its batch size may have led to this performance as the gradient descent in the algorithm may have greatly reduced the loss function during each epoch. Logistic regression better handles feature correlations present in the HAR dataset very well and the drop in accuracy from training to validation to test data shows that the model generalizes well without overfitting.

5. Conclusion

a. Closure

To conclude this project demonstrates that through exploratory data analysis and data preprocessing we can gain a better understanding and control of the data we pass into our machine learning models. For Human Activity Recognition, Logistic Regression provided the most efficient model along with the best accuracy. The k-NN model performs well too, but may not generalize well for new external data sources. Naive Bayes performed the worst, as its initial assumption may not be entirely true in human activities where gyroscopic data may be dependent on body angle feature values.

Throughout this project I was able to apply the concepts I have learned throughout the semester to use and apply them to a real world problem. I have learned that the easiest model such as k-NN, may work well depending on the dataset but to always try other models before choosing the best model for a use case. I also learned the importance of understanding the data before constructing a machine learning model as knowledge about your dataset is crucial in understanding the results produced by the model and tuning the parameters of the model to increase prediction accuracy.

b. Challenges

I didn't face many challenges while making the machine learning models as most of my models are derived from the class assignments throughout the semester. The place I initially faced issues was extracting the data from text files in UC Irvine's ML Repository which I later gave up on and found the same data on Kaggle in comma-separated values format which I had experience using.

c. Future Work

This project can definitely be improved through Principal Component Analysis (PCA), which may be helpful in selecting the most crucial features among the 561 features of this dataset. Recursive Feature Elimination (RFE) is another improvement which may be made which will help in removing the features which are highly correlated and will greatly reduce the number of features the models will need to use for training increasing the efficiency of the model. Decision Trees and Deep Neural Networks may also help in refining the model's accuracy and generalize well at the same time. Only through experimentation with models can we improve the classification of such datasets which may greatly impact healthcare, fitness, and elderly care industries.

6. References/Acknowledgements

Dataset:

i. “Human Activity Recognition with Smartphones.” Www.kaggle.com, www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones.

ii. Reyes-Ortiz, Jorge, et al. "Human Activity Recognition Using Smartphones." UCI Machine Learning Repository, 2013, https://doi.org/10.24432/C54S4K.

Resources:

iii. morrisb. “What Does Your Smartphone Know about You?” Kaggle.com, Kaggle, 30 June 2018, www.kaggle.com/code/morrisb/what-does-your-smartphone-know-about-you/notebook. Accessed 6 May 2025.

- This is a Kaggle Notebook which was under code for the dataset which helped me during my EDA.

iv. “Pandas.DataFrame.select_dtypes — Pandas 2.2.3 Documentation.” Pydata.org, 2024, pandas.pydata.org/docs/reference/api/pandas.DataFrame.select_dtypes.html#pandas.DataFrame.select_dtypes. Accessed 6 May 2025.

- I needed only numerical data for visualization after I made a combined dataframe so to do that I googled, “how to select only numbers from a dataframe.” This led me to the above documentation which I used to understand how to use .select_dtypes.

v. seaborn. “Choosing Color Palettes — Seaborn 0.9.0 Documentation.” Pydata.org, 2013, seaborn.pydata.org/tutorial/color_palettes.html.

- The default color for my heatmap was not good so to find better color schemes I googled, “cmaps for seaborn plots.” This led me to the above documentation which I used to find better colors for the heatmap.

vi. Scikit-learn. “Sklearn.preprocessing.LabelEncoder — Scikit-Learn 0.22.1 Documentation.” Scikit-Learn.org, 2019, scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

- Originally my code raised an error when using the combined data frame for preprocessing with a hint that there is a categorical label present. I googled, “Label encoding in machine learning” which led me to the above documentation which I used to understand how to implement .LabelEncoder from the Sklearn library.

Acknowledgement:

I have completely used all my old code from the homeworks for Module 3 - KNN, Module 8 - Naive Bayes, and Module 9 - Logistic Regression. Apart from that I hereby declare that I have not used any other resource other than the ones listed above in the references.

7. Source Code : https://github.com/ajaydsudhir/ITCS_3156_FinalProject.git