EECE 6690: Statistical Learning for Biological and Information Systems

Final Project Report

Columbia University in the City of New York

By: Stefan Pancar and Ronan McNally
Submitted to: Professor Predrag Jelenkovic
December 18, 2024

Introduction

With the growing availability of low-cost and low-power devices such as accelerometers, gyroscopes, and cameras, Human Activity Recognition (HAR) has become an exciting field of study [1] [2]. HAR is the ability to accurately detect various human activities such as standing, walking, running, lying down, and other common motions that individuals perform in their daily lives. The data is typically collected using dedicated wearable sensors that attach to the user, or through video processing of frames and images [1] of a person performing the activities. After collecting that data from the sensors, real-time or after-the-fact processing takes place to detect the activity performed during a specified time window.

The recognition of HAR is useful in many fields where the study of movement or identification of patterns in a subject's movement is crucial. Such fields include healthcare, military, and security [3]. For instance, the exercise routine of a patient diagnosed with heart disease can be monitored using HAR techniques to ensure they are correctly performing the specified activities [3] [4]. HAR can also be used with intelligent surveillance systems in public spaces to classify people's movements and categorize them as routine or provide alerts if the movement becomes abnormal [5].

Smartphones have emerged as a prominent tool for HAR among wearable devices. This is because a smartphone is used by people of many different demographics and is carried without being perceived as an additional wearable sensor [2] [6]. Within the smartphone is typically an accelerometer and gyroscope that can gather data, which can be analyzed using statistical analysis and machine learning techniques to perform HAR.

A Support Vector Machine (SVM) is a widely used supervised machine-learning method that trains a classifier function using labeled data [7]. It works by identifying a hyperplane that acts as a decision boundary, separating data points into distinct classes or clusters. This separation is achieved by defining a hyperplane as the decision boundary between classes [7]. The SVM's behavior is influenced by hyperparameters such as the kernel, slackness, and hypermargin slope. A Gaussian kernel is typical for HAR because sensor data exhibits nonlinear behavior. Applying a Gaussian kernel, the data is mapped to a higher dimensional space and can then be separated using linear techniques [8].

This method of HAR is demonstrated in the study conducted by Anguita et al. [9] which through the use of data collected from a three-dimensional accelerometer and gyroscope in a Samsung smartphone, the activities of walking, walking upstairs, walking downstairs, sitting, standing, and laying down can be recognized with accuracy comparable to dedicated sensors. With the implementation of the supervised machine learning algorithm One-Versus-All Support Vector Machines, a margin can be made to separate a specific activity from the rest [9].

This report will replicate the experimental results performed and gathered by Anguita et al. [9] to prove that with an overall accuracy of 96%, One-Versus-All SVM is a machine-learning technique that can successfully distinguish between the specified human activities listed above. Different techniques to improve upon the SVM will be proposed along with other machine learning methods such as Decision Trees, Bagging, Random Forest Decision Trees, Boosting, and XGBoost, which will be implemented and tested to determine if they have an accuracy comparable to most dedicated sensors, which typically has an overall accuracy range of 90-96%.

Summary of Dataset and Paper

Within the paper [9], predicted human activity was constricted to six behaviors: walking, walking upstairs, walking downstairs, sitting, standing, and lying down (labeled one through six in the dataset respectively). Thirty volunteers were instructed to follow a protocol directing them to perform the set of behaviors twice; once with a smartphone mounted at the waist and once with the smartphone positioned wherever the subject preferred. The phone's accelerometer and gyroscope were used to record triaxial linear acceleration and angular velocity at a sampling rate of 50 Hz [9]. To minimize noise, the raw data was processed through a median filter followed by a third-order low-pass Butterworth filter with a 20 Hz cutoff frequency [9]. The reason for this is that the majority of human body motion is captured at frequencies below 15 Hz [9]. Then, by deriving the linear acceleration and angular velocity respectively, jerk and angular acceleration time signals are obtained. By applying a Fast Fourier Transform, certain time-domain signals were transformed into the frequency domain. This processing generated a multitude of signals which are detailed in Table 1 below, showing their respective domains.

Table 1: Signals obtained from smartphone sensors in the time and frequency domain. [9]

Name	**Time**	**Frequency**
Body Acc	✓	✓
Gravity Acc	✓	X
Body Acc Jerk	✓	✓
Body Angular Speed	✓	✓
Body Angular Acc	✓	X
Body Acc Magnitude	✓	✓
Gravity Acc Mag	✓	X
Body Body Acc Jerk Mag	✓	✓
Body Angular Speed Mag	✓	✓
Body Angular Acc Mag	✓	✓

The dataset observations were created by dividing each subject's performance of the six activities into 2.56-second time windows with 50% overlap [9]. Across all 30 subjects, this resulted in a total of 10,299 observations. For each 2.56-second observation window, the statistical values outlined in Table 2 were calculated to ensure a robust dataset was created. Each statistical function was calculated with respect to the x-, y-, and z-axis of each time signal.

Table 2: List of statistical values used for computing features. [9]

Function	Description
mean	Mean value
std	Standard deviation
mad	Median absolute value
max	Largest values in array
min	Smallest value in array
sma	Signal magnitude area
energy	Average sum of the squares
iqr	Interquartile range
entropy	Signal Entropy
arCoeff	Autoregression coefficients
correlation	Correlation coefficient
maxFreqInd	Largest frequency component
meanFreq	Frequency signal weighted average
skewness	Frequency signal Skewness
kurtosis	Frequency signal Kurtosis
energyBand	Energy of a frequency interval
angle	Angle between two vectors

The three-axial statistical functions, along with the magnitude of each time signal and frequency bands from 1-48 Hz in 8 Hz increments of each frequency signal, resulted in 561 features [9]. Including the subject number brings the total number of features to 562. The data was then split, with 70% used for training and 30% dedicated to testing the model.

The research team [9] developed a model using the SVM binary classifier with the One-Versus-All approach for their multiclass target variable. The SVM hyperparameters were obtained through a 10-fold cross-validation with a Gaussian kernel. The resulting model achieved an overall test accuracy of 96%, proving that smartphones can be highly effective as HAR devices. Illustrated in the confusion matrix on the next page (Table 3) are the results of their work [9].

Table 3: Confusion Matrix obtained by Anguita et al. using One-Versus-All SVM

	Wk	WU	WD	ST	SD	LD	**Recall**
Walking	**492**	1	3	0	0	0	99%
W. Upstairs	18	**451**	2	0	0	0	96%
W. Downstairs	4	6	**410**	0	0	0	98%
Sitting	0	2	0	**432**	57	0	88%
Standing	0	0	0	14	**518**	0	97%
Laying Down	0	0	0	0	0	**537**	100%
Precision	96%	98%	99%	97%	90%	100%	**96%**

Reproduce the Results of the Paper

This section provides a detailed explanation of the process used to replicate the results of the study conducted by Anguita et al. [9] using the popular statistical programming language R. It covers the steps for importing the data into RStudio, tuning the hyperparameters gamma and cost, constructing the One-Versus-All SVM, and generating the confusion matrix to evaluate the overall accuracy. Each step will be discussed in detail in its respective section below.

Import Dataset into RStudio

The HAR dataset [9] was downloaded from the UCI Machine Learning Repository and is provided as documents, with the input, output, and subject labels stored separately. The input data, as described in the previous section, consists of data points collected for each activity. The output data contains a column listing the activities performed, corresponding to the input data. Finally, the subject text file identified the volunteer associated with each data record.

A separate folder was made for the training and the testing data, providing a predefined 70/30 split ratio. This approach allows the model to be trained on 70% of the data and tested on the unseen 30%, ensuring more accurate results.

To prepare the data for subsequent steps, the input, output, and subject files for both the training and testing data were loaded and combined into respective variables as shown in Figure 1. Additionally, the subject and activity columns were labeled respectively to clarify the meaning of those two data columns.

```r
# Combine data
train_data <- cbind(subject_train, y_train, X_train)
test_data <- cbind(subject_test, y_test, X_test)

# Rename columns for clarity
colnames(train_data)[1:2] <- c("Subject", "Activity")
colnames(test_data)[1:2] <- c("Subject", "Activity")
```

Figure 1: Creation of training and testing data, and labeling of Subject and Activity columns.

To build the SVM models, the numbers in the activity column must be treated as categorical variables due to the classifying nature of support vector machines. This is done by converting the activity column into a factor, where each value corresponds to the six activities: (1) walking, (2) walking upstairs, (3) walking downstairs, (4) sitting, (5) standing, and (6) laying. By representing the activity column as a factor, the task is now a classification problem, a requirement for training support vector machines. The code that performs this task is simple (Figure 2), concluding this section of the replication process.

```r
# Extract labels from Activity columns to perform One Vs All SVM
train_data$Activity <- as.factor(train_data$Activity)
test_data$Activity <- as.factor(test_data$Activity)
```

Figure 2: Converting the Activity columns of training and testing data into factors, and storing them into appropriately labeled variables respectively

Support Vector Machine Hyperparameter Tuning

The performance of an SVM depends on the specified parameters as well as the kernel function [10]. The kernel function can be easily defined in R, and Anguita et al. [9] identified it as a Gaussian separation curve, corresponding to a radial kernel for SVM. The SVM hyperparameters are more difficult to find. Gamma is a hyperparameter needed for all kernels except a linear kernel, and it defines the slope of the kernel function. If the value of gamma is low, the slope of the decision boundary becomes low and there is less of a curve. If the value of gamma is high, the curve of the decision boundary becomes high and is more likely to create separating islands around data points [10]. Cost is another important hyperparameter that is a budget for slackness, or how many observations can be on the wrong side of the margin.

One can randomly select these hyperparameter values, but it is best to optimize the values for the given dataset. Tuning methods include Grid Search, Random Search, and Bayesian Optimization. Grid Search is a method that involves testing all possible combinations of hyperparameters in a specified bounds function. The combination of hyperparameters with the highest accuracy gets selected. The process is slow but with careful choosing of the bounds, and with multiple iterations, accurate hyperparameters can be achieved [11]. Random Search is similar to Grid Search but instead of testing all possible combinations it randomly selects values within bounds [11]. This method is less computationally heavy in traversing a wide range of bounds. However, when compared to Bayesian Optimization it is still not as efficient. Bayesian Optimization has two components, the probabilistic surrogate model and the acquisition function. The probabilistic surrogate model is fit to all observations in each iteration while the acquisition function identifies promising parameter combinations to improve the search and find

the best hyperparameters [1]. If set correctly, this method takes less time than Grid Search and Random Search. This is because the algorithm learns from previous evaluations, reducing the number of searches performed.

While all three methods were used to find the optimal cost and gamma values of the One-Versus-All SVM, it was found that the simple Grid Search resulted in the best overall accuracy of the model on both the testing and training data. Although Grid Search was the most time-consuming method, due to its superior accuracy with the HAR dataset, it was the preferred method. For this reason, its implementation will be discussed below.

The *train()* function, part of the R caret package, was selected to perform Grid Search due to its ability to enable parallel processing for faster computation. Another option could be the *tune()* function in the *CRAN* library, however, it can be more difficult to set up. The *train()* function was set to employ 10-fold cross-validation, configured with *trainControl()*, to define model evaluation settings to optimize the SVM. A grid of SVM hyperparameters, including cost and sigma (related to gamma and optimized by the train() function), is created using *expand.grid()*. This allows testing of all possible combinations of the defined values. Lastly, the *train()* function is used to train the SVM model on the training data to predict activities on all the other features of the dataset. By performing cross-validation, the model evaluates the accuracy for each combination of cost and gamma and selects the pair that achieves the highest accuracy on the training data. This ensures that after a few iterations and adjustments to the grid, the final model is well-tuned to seen and unseen data. The code that was made to perform this process is shown below in Figure 3.

```
# Start parallel processing
c1 <- makeCluster(detectCores() - 1)# Save 1 core for system processes
registerDoParallel(c1)

# Set up method of training as 10-fold cross-validation
train_control <- trainControl(
  method = "cv", # Method is cross-validation
  number = 10, # Number of folds is 10
  allowParallel = TRUE # Enable parallel processing
)
# Define search grid for hyperparameters
svm_grid <- expand.grid(
  C = c(0.1, 1, 10),
  sigma = c(0.01, 0.1, 1)
)

#train SVM model
svm_model <- train(
  Activity ~ ., # Predict Activities an all the other features in data
  data = train_data, # Train on the training data set (70%)
  method = "svmRadial", # Train on a SVM Radial model, which is mathematically similar to Gaussian
  metric = "Accuracy", # Optimize based on accuracy
  trControl = train_control, # Calling the trainControl function
  tuneGrid = svm_grid # Calling the grid to search from
)

stopCluster(c1) # Stop parallel processing
print(svm_model) # Print results to get best hyperparamters
```

Figure 3: Optimal hyperparameter training setup to find best cost and gamma values for SVM

A table has been constructed to demonstrate how the search grid was refined across iterations to identify the optimal cost and gamma values to maximize the SVM model's accuracy on the training data (Table 4). During the first iteration, the specified grid spans a wide range of

values to estimate the general magnitude of the hyperparameters. In the iterations to follow, the grid is progressively narrowed and fine-tuned until it converges to a single value of cost and gamma with little accuracy change. Using 10-fold cross-validation (Figure 3) ensures that the selected hyperparameters will be well-suited for the testing data. The final values for cost and gamma converged to 11.25, and 0.00125 after eight iterations. This completes the hyperparameter tuning and these values will be used in the One-Versus-All SVM.

Table 4: Grid Search iterations to find optimal cost and gamma hyperparameters using *train()*.

Iteration:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Cost Grid	0.1, 1, 10	5, 10, 15, 20	10, 11, 12, 13, 14, 15, 16, 17	10.5, 11, 11.5	10.8, 11, 11.2	11.1, 11.2, 11.3	11.2, 11.3, 11.4	11.25, 11.3, 11.35
Gamma Grid	0.01, 0.1, 1	0.005, 0.01, 0.02	0.004, 0.0045, 0.005, 0.0055	0.001, 0.002, 0.003, 0.004	0.0015, 0.002, 0.0025	0.0013, 0.0015, 0.0017	0.0012, 0.0013, 0.0014	0.00125, 0.0013, 0.00135
Values Selected (cost, gamma)	10, 0.01	15, 0.005	11, 0.004	11, 0.002	11.2, 0.0015	11.3, 0.0013	11.3, 0.0013	11.25, 0.00125
Accuracy	0.9447755	0.9778310	0.9829980	0.9885725	0.9892330	0.9891118	0.9892546	0.9892550

One-Versus-All Support Vector Machine

With the optimal hyperparameters, the One-Versus-All approach was constructed to imitate the methodology performed by Anguita et al. [9]. The code to create the SVM for activity classification is illustrated in Figure 4 below. Since there are six activities to traverse through, a *for()* loop is used with a counter that goes up to 6 total iterations. For each activity performed, the *Activity* column is transformed into binary labels, the current class is labeled as 1, while all other activities are labeled as 0, where a 1 indicates true and 0 indicates false. This transformation allows the SVM to distinguish one activity from the rest, essentially performing One-Versus-All separation techniques. The *svm()* function is contained in the *e1071* library and conveniently allows for the specification of data, kernel, hyperparameters, and probability calculation. Each SVM model is built with a radial kernel with tuned cost and gamma hyperparameters similar to the paper [9], and stored in a list. After training, the summary of each model's performance is printed. This method ensures that each model is specialized for one of the six activities and optimized for training and testing data accuracy.

```r
classes <- levels(train_data$Activity) # Extract numeric label for each activity (1->6)
td <- list() # Initialize list that will be filled with each one-versus-all SVM
for(i in classes){ # Loop from 1 to 6
  temp_data <- train_data
  temp_data$Activity <- as.integer(temp_data$Activity) # Ensure activity is an integer
  temp_data$Activity[temp_data$Activity != i ] <- 0 # Activity not equal to the current class is set to 0
  temp_data$Activity[temp_data$Activity == i] <- 1 # Activity equal to the current class is set to 1
  # Build One-Versus-All SVM
  td[[i]] <- svm(temp_data$Activity ~ .,
                 data = temp_data, kernel = "radial",
                 gamma = best_gamma, # Tuned gamma
                 cost = best_cost, # Tuned cost
                 probabilities = TRUE) # Calculate probabilities
  # Print for loop progress
  print(paste("Model for class", i, "trained successfully"))
}
```

Figure 4: One-Versus-All Support Vector Machine model training for all six activities with tuned hyperparameters.

Build of Confusion Matrix

Due to the classifying nature of the SVM, Anguita et al. [9] built a confusion matrix (Table 3) to analyze the performance of their model. A confusion matrix is a table that easily identifies in a model which classes have been confused with other classes during classification testing [12]. For instance, it not only shows the accuracy of the model but also the number of classes that were correctly predicted and the number of classes that were improperly placed into a different class. The topmost row typically represents the predicted classes and the leftmost row represents the actual classes. The principal diagonal values are the number of correctly predicted data points, the more data points that are correctly predicted, the higher the overall accuracy.

To ensure an accurate comparison with Anguita et al. [9], a similar confusion matrix was constructed in R. Additionally, the model's recall, precision, and overall accuracy were computed. The precision measures the percentage of correct predictions, while recall indicates the percentage of relevant data points correctly identified. The overall accuracy reflects the total performance of the SVM model. It is desirable to have these numbers as high as possible since a higher precision reduces the number of false positives, whereas higher recall focuses on maximizing the number of true positives. A higher overall accuracy is desired and is an easy way to see where the HAR SVM model falls within the typical dedicated sensor accuracy range.

The code to construct the confusion matrix is shown in Figure 5.a and Figure 5.b. The first step is to initialize an empty list that will store predicted probabilities for each activity class. A prediction loop was implemented to iterate over each activity class, using the One-Versus-All SVM model to predict probabilities for the test data. These predicted probabilities are combined to form a table using the *cbind()* function (Figure 5.a). A confusion matrix is created by comparing the true activity labels, *test_data$Activity* with the predicted labels in table *a* (Figure 5.b). This matrix summarizes the classification model performance, and precision, recall, and overall accuracy were also calculated to enhance the knowledge of performance metrics and to compare with Anguita et al. The confusion matrix resulted in an overall accuracy of 96.13%, which concludes that the SVM model accurately reflects the results obtained by Anguita et al. [9]. The full results of the confusion matrix are shown in Table 5, which illustrates results comparable to Anguita et al. [9].

```r
bind <- list() # Initialize empty list to store predicted probabilities
for (i in classes){
  # Return prediction based off of probability using SVM model and test data
  # Exclude Activity label column to predict on only the features
  predict_td <- predict(td[[i]], test_data[,-2], probability = TRUE)
  # Combine columns to form a table of predictions
  bind <- cbind(bind, predict_td)
}

# 1 WALKING
# 2 WALKING_UPSTAIRS
# 3 WALKING_DOWNSTAIRS
# 4 SITTING
# 5 STANDING
# 6 LAYING

# Create numerical labels for each activity
classnames <- c(1,2,3,4,5,6)
```

Figure 5.a: Build of a prediction table for the confusion matrix.

```r
# For each row, find the column matrix that corresponds to highest probability
a <- apply(bind, 1, function(row) classnames[which.max(row)])

# To visualize if needed, columns of class probabilities and final predictions
b <- cbind(bind, Predicted = a)

# Creates the confusion matrix of actual and predicted values
confusion_matrix <- table(True = test_data$Activity, Predicted = a)
print(confusion_matrix)

# Calculate the precision, recall, and overall accuracy
# Percentage of predictions that are correct
precision <- diag(confusion_matrix) / colSums(confusion_matrix)
# Percentage of relevant data points that were correctly identified
recall <- diag(confusion_matrix) / rowSums(confusion_matrix)
# Overall Accuracy
overall_accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)

# Print overall accuracy
print(data.frame(Precision = precision, Recall = recall))
cat("Overall Accuracy:", overall_accuracy)
```

Figure 5.b: Build of confusion matrix using a precision table, and calculation of precision, recall, and overall accuracy.

Table 5: Confusion Matrix obtained through One-Versus-All SVM demonstrated in this report.

	Wk	WU	WD	ST	SD	LD	**Recall**
Walking	**490**	3	3	0	0	0	98.79%
W. Up	11	**459**	1	0	0	0	97.45%
W. Down	1	16	**403**	0	0	0	95.95%
Sitting	0	1	0	**447**	41	2	91.039%
Standing	0	0	0	18	**514**	0	96.62%
Laying Down	0	0	0	0	17	**520**	96.83%
Precision	97.61%	95.8%	99.02%	96.14%	89.86%	99.62%	**96.13%**

Improvements made on Paper

Although the One-Versus-All SVM described above accurately reflects the results of Anguita et al. [9] as well as proving that this is an accurate method for HAR, there is room for potential improvement. For this reason, different techniques will be discussed and implemented in the following subsections. The overall accuracy will serve as a measurement of how much better or worse the proposed method is on the HAR dataset.

Improvements made to the SVM model

Before any new methods are proposed, it is essential to determine whether the accuracy of the SVM model can be improved to exceed an overall accuracy of 96%. One approach discussed in a previous section was fine-tuning the SVM hyperparameters using Bayesian Optimization instead of Grid Search. Although Bayesian Optimization proved to be faster, it was found that Grid Search achieved a slightly better overall accuracy.

Attention then shifted towards methods involving dataset manipulation. It is easy to prove that increasing the number of training data points generally improves the accuracy [13]. As previously mentioned a 70/30 data split is used, but an 80/20 split is also common. The 80/20 split gives the model 10% more data to train on while leaving sufficient data to test on. For this reason, an 80/20 split was implemented and evaluated to assess the impact of this data manipulation on the SVM model's accuracy as well as the accuracy of the new models proposed. The results of the models trained on the latest data split will be displayed in the discussion section of the report, and it was shown that the overall accuracy generally increased using the 80/20 data split. The preceding sections will dive into implementing the new machine learning techniques proposed with the intention to increase the overall accuracy.

Improvements made with Different Techniques: Decision Trees

As previously mentioned, a One-Versus-All SVM, with appropriately tuned hyperparameters, can achieve high accuracy in predicting ADL. However, with a multitude of algorithms capable of classification, it is worth exploring if there are alternative models for HAR. To this end, Decision Trees were employed on the dataset to evaluate it as a potential alternative. Decision Trees were explored for three reasons: the nonlinearity of the data points, the multiclass nature of the classification problem, and the intrinsic interpretability of Decision Trees. Their inherent nonlinearity, resulting from piecewise boundaries, makes them effective for issues like this. Given activities such as walking, walking upstairs, and walking downstairs produce oscillatory readings, while activities like standing, sitting, and lying down yield constant values along different axes, it is implied that this is a nonlinear problem. Moreover, the relationship between vector components, vector magnitude, and features spanning both time and frequency domains suggest a nonlinear problem. Additionally, while SVMs are capable of multiclass classification, a One-Versus-All (or One-Versus-One) approach may obscure correlations between the classes that could enhance the model. The hierarchical relationship, where walking, walking upstairs, and walking downstairs represent oscillatory movements, while sitting, standing, and lying down signifies stillness, may be overlooked by an SVM grouping movement and motionless ADLs together in a One-Versus-All approach. These are nonissues for Decision Trees. Additionally, while SVMs can perform well on moderately dimensional data

such as this dataset, they are black-box models, making it difficult to understand their decision-making process. However, Decision Trees, whether singular or ensemble, provide the opportunity to inspect their branching and garner insight into how their decisions are made. That said, not all Decision Tree algorithms perform equally well on the same problem, so it is essential to explore multiple options. While the following discussion's results are from the 70/30 data split, the test accuracy results from the 80/20 split can also be seen in Table 6.

Single Decision Tree

To establish baseline performance, a singular Decision Tree was trained on the training data. This tree is illustrated in Figure 6 below.

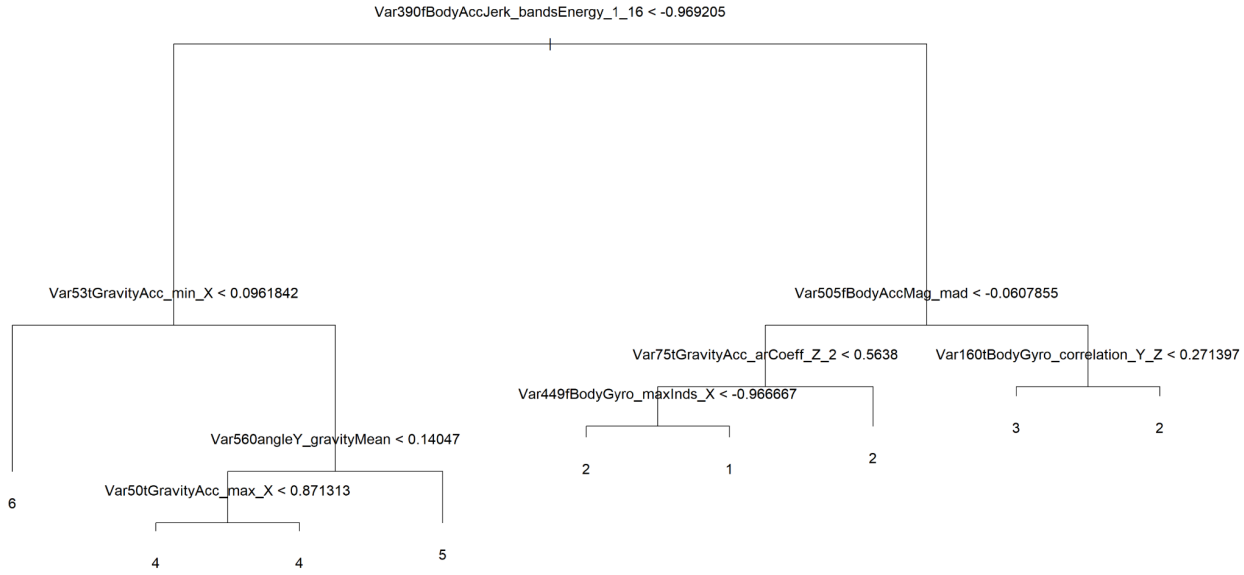

Figure 6: Plot of singular Decision Tree.

This model was found to have a testing accuracy of about 83.61% as shown by the confusion matrix below (Figure 7).

```
test.predict.tree_test    1   2   3   4   5   6
                      1 425  61  33   0   0   0
                      2  55 405 115   0   0   0
                      3  16   5 272   0   0   0
                      4   0   0   0 400 107   0
                      5   0   0   0  91 425   0
                      6   0   0   0   0   0 537

Test Accuracy of Initial Tree:
83.61045 %
```

Figure 7: Confusion matrix of singular Decision Tree.

It's evident from the confusion matrix that this tree is less accurate than the SVM developed by Anguita et al. [9]. Despite this, its interpretability still enables insights to be gleaned. For

instance, the first split made by the decision tree divides the classes according to their hierarchical relationship to movement; walking, walking upstairs, and walking downstairs are all ADL with movement while standing, sitting, and lying down are all ADL without movement. The first metric to divide them is the amount of energy within the 1 Hz to 16 Hz frequency band of the jerk signal. Given the average walking speed of a person ranges from 1.8 Hz to 2.1 Hz, it makes sense to form a decision based on the amount of energy used within this frequency band for the jerk signal when determining whether a person is moving, thus distinguishing between moving and non-moving ADLs [14]. If the amount of energy is sufficiently high, it is a movement ADL; if not, it is a motionless ADL. Regardless of this insight, a more accurate model is preferred. A conventional source of error in single decision trees is that they are typically grown with more splits than necessary and overfit the training data, thereby increasing test error. Performing tree pruning with cross-validation to evaluate the model performance after each cut resulted in Figures 8 and 9 below.

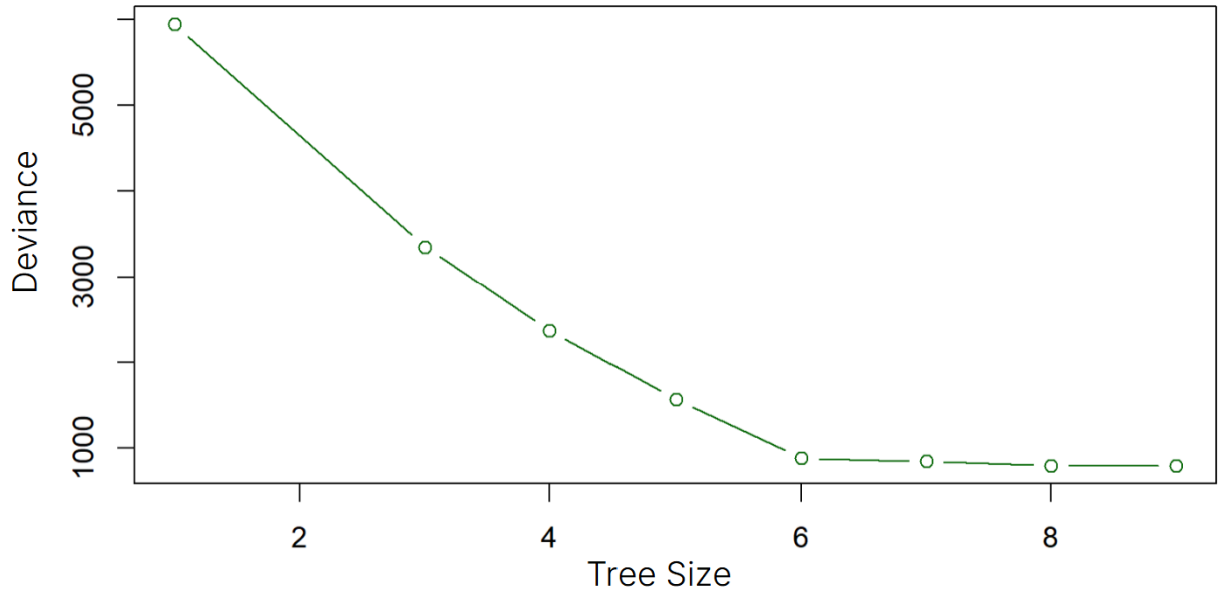

Figure 8: Deviance of singular Decision Tree concerning size as it is pruned.

```
[1] "size"   "dev"    "k"      "method"
$size
[1] 9 8 7 6 5 4 3 1

$dev
[1]  790  790  842  881 1562 2369 3351 5945

$k
[1]   -Inf    0.0   59.0   75.0  682.0  826.0  989.0 1299.5

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

Figure 9: Terminal output from singular Decision Tree pruning process.

From the figures, it is apparent that the error does not reside in overfitting as any amount of pruning results in an increase in deviance. Thus, it is more likely that the single tree is struggling to encapture the entire feature space given the moderately dimensional dataset. Across all 562 features, it is difficult to choose the optimal split. Bagging will be explored in the subsequent section to resolve this.

Bagging

An ensemble of 500 decision trees was constructed using the *randomForest* library, where every feature was considered at each possible split. These trees collectively traverse the feature space and aggregate into the final predictive model. The resulting model produced the confusion matrix shown in Figure 10.

```
bagging.test.predict   1   2   3   4   5   6
                   1 477  56   8   0   0   0
                   2   6 407  48   0   0   0
                   3  13   8 364   0   0   0
                   4   0   0   0 398  61   0
                   5   0   0   0  93 471   0
                   6   0   0   0   0   0 537

Bagging Test Accuracy:
90.05769 %
```

Figure 10: Confusion matrix from the Bagging method.

With an accuracy of around 90.06%, it is evident from the Bagging confusion matrix that an ensemble method is a marked improvement over a single decision tree. This accuracy places this model within the accuracy range of specialized HAR sensors of 90-96% described by Anguita et al. [9].

To further improve the test accuracy, one might examine the relationship between the out-of-bag (OOB) error and the number of decision trees generated (Figure 11). While increasing the number of trees within the Bagging ensemble could enhance generalizability and reduce the test error, the plot in Figure 11 indicates that any additional gains would be minimal. The computational cost of adding more trees would not justify the minimal improvements.

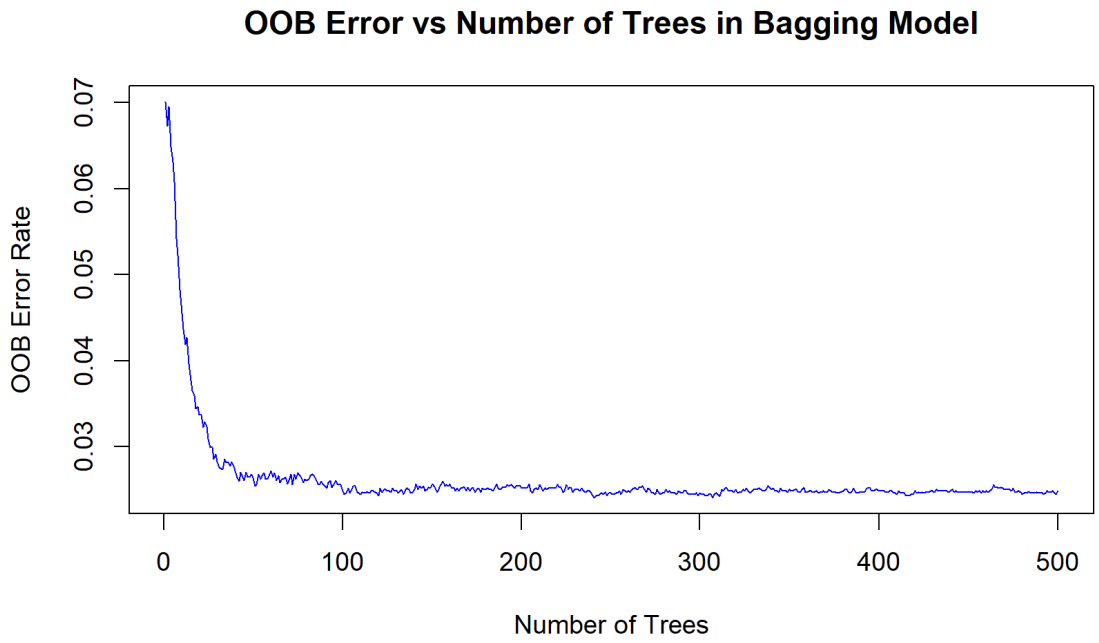

Figure 11: Out-of-bag error for Bagging with respect to the number of trees in the ensemble.

With that said, one notable weakness of Bagging is that it is susceptible to feature dominance, where features with strong predictive power can disproportionately dominate the ensemble's decision-making. This can be seen in Figure 12, where a small set of features causes a steep decrease in accuracy in their absence. While the labels of said features are not included for visual simplicity, the shape of the discrete curve remains telling.

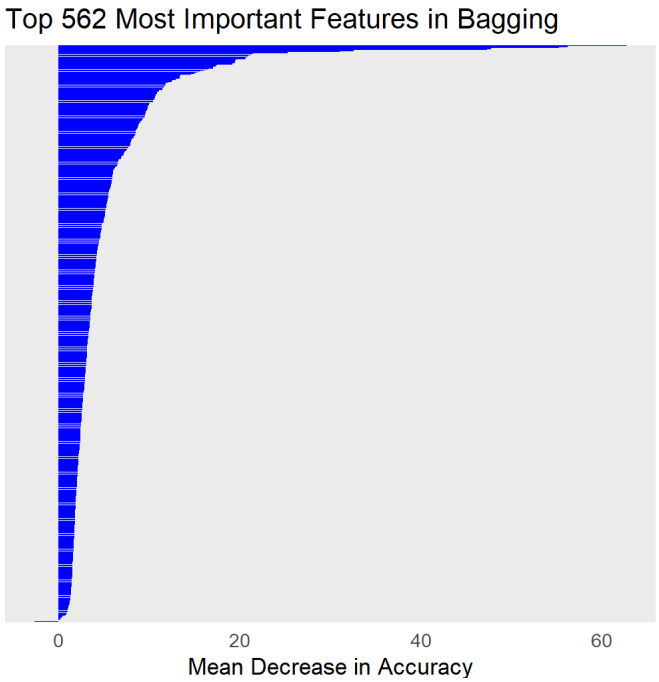

Figure 12: Importance ranking of features for Bagging.

When this occurs, the ensemble's ability to reduce overfitting through averaging is diminished due to increased correlation between the trees. Random Forest Decision Trees address this issue by randomly selecting a subset of the features to consider at each split, effectively reducing the feature dominance.

Random Forest Decision Trees

This section describes the implementation of Random Forest Decision Trees. When using the *randomForest()* function and setting the number of variables to randomly sample equal to the total number of available features, the function defaults to randomly sampling the (rounded) square root of the number of features, as this is a classification problem. Training and evaluating the model with this approach produces the confusion matrix shown in Figure 13.

```
rf.test.predict   1   2   3   4   5   6
              1 481  32  19   0   0   0
              2   8 433  46   0   0   0
              3   7   6 355   0   0   0
              4   0   0   0 438  45   0
              5   0   0   0  53 487   0
              6   0   0   0   0   0 537

RF Test Accuracy:
92.67051 %
```

Figure 13: Confusion matrix of the Random Forest method.

Results show that transitioning from Bagging to Random Forest improves test accuracy. By tuning the maximum number of terminal nodes within the tree ensemble to reduce overfitting, a marginally higher test accuracy of 93.05% was found. It is important to note that the same number of trees as in Bagging was used, as evidenced by the similar OOB error rate versus the tree number plot shown in Figure 14.

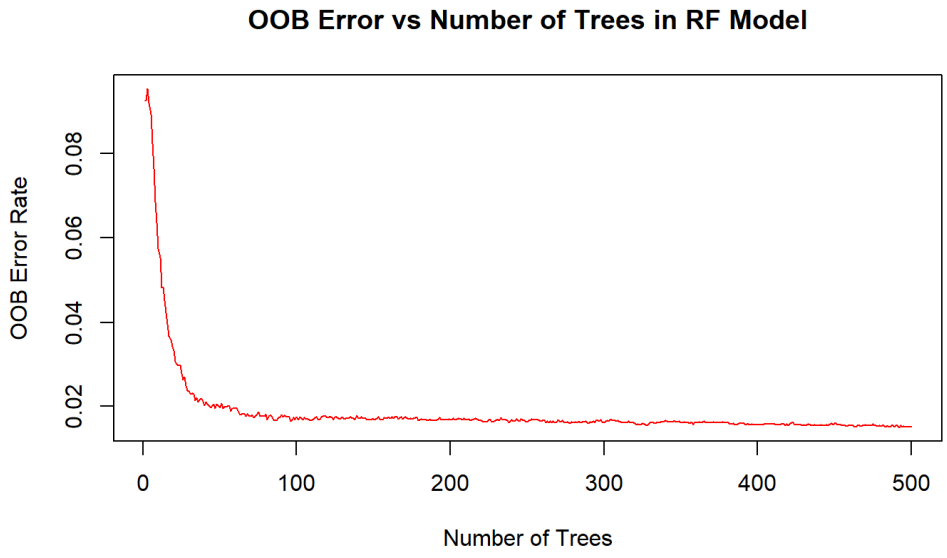

Figure 14. Out-of-bag error for Random Forest with respect to the number of trees in the ensemble.

From the Random Forest model, a ranking of the features in terms of predictive importance can also be generated, as shown in Figures 15 and 16.

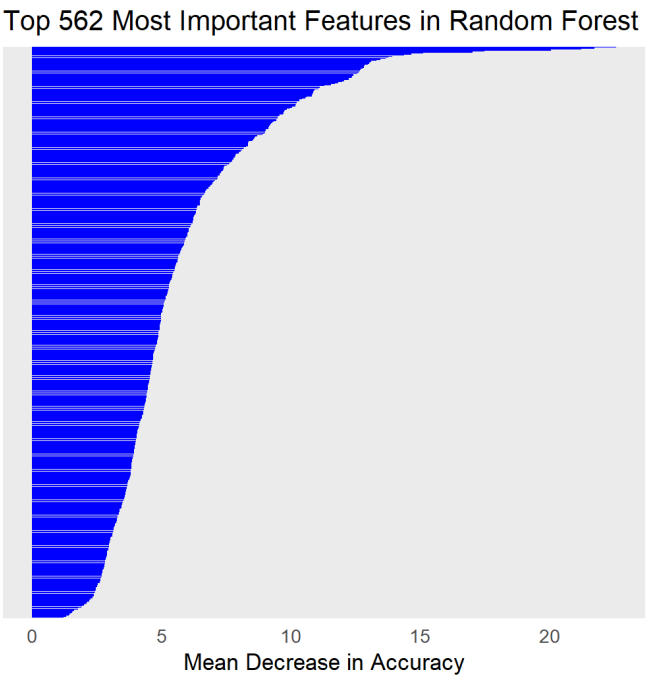

Figure 15: Importance ranking of features for Random Forest.

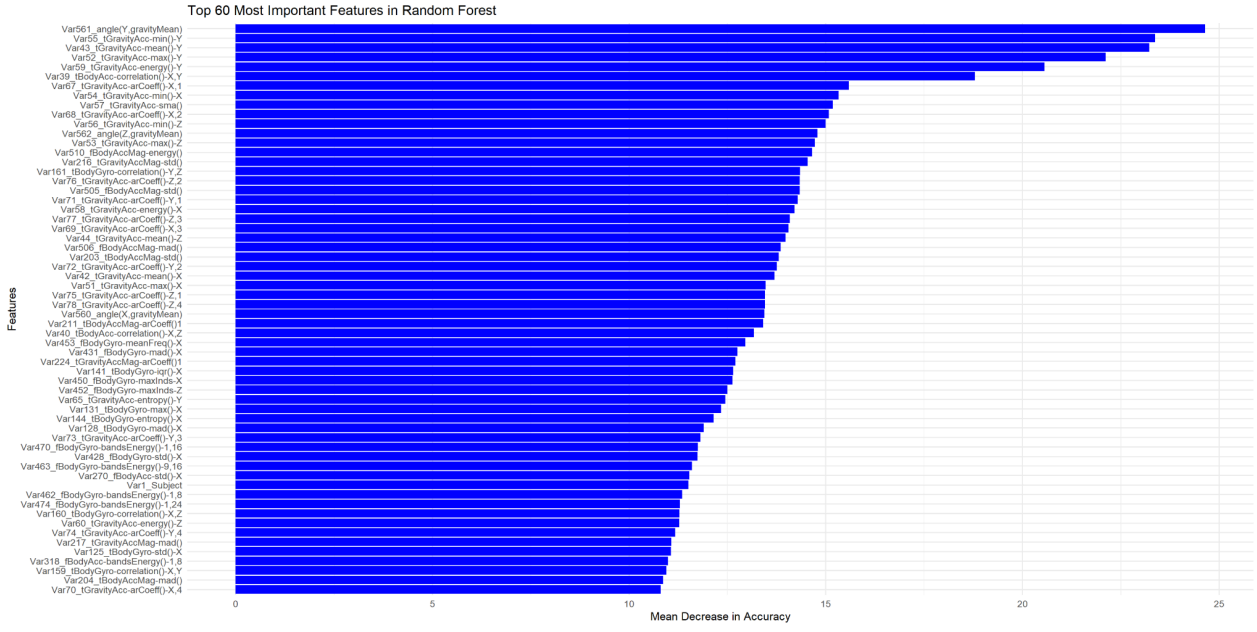

Figure 16: Top sixty most important features from the Random Forest method.

Comparing Figure 12 with Figure 15, it is evident that randomly sampling the feature space at each decision split prevents strong feature dominance, as shown by the narrower spread of mean decrease in accuracy in Figure 12 compared to the greater variability in Figure 15. Additionally, Figure 16, which zooms in on the rankings to display the top 60 most important features, highlights the importance of ensemble methods. The first decision made by the singular decision tree, deemed the "optimal choice", is not present within this list underlining the challenges single trees face in optimizing their splits with higher dimensional datasets. However, while the Random Forest model markedly improves upon the singular decision tree, its accuracy still falls behind the SVM developed by Anguita et al [9]. For this reason, Boosting methods were the next approach explored.

Generative Boosting

The first Boosting method that will be explored is Generative Boosting, which like most Boosting methods, has an advantage over Random Forest in that instead of building an ensemble of independent trees, the forest is built sequentially, with each one learning from and correcting the errors of its predecessor. This weak learning process prevents overlearning and overfitting assuming the shrinkage parameter is appropriately tuned. The tuning process for the shrinkage parameter in the Generative Boosting model was informed by the Generative Boosting section of *An Introduction to Statistical Learning with Applications in R Version* by Gareth James et al. [15], alongside the use of *gbm()* function from the *gbm* library. In this section, it is recommended to start with a shrinkage value of 0.01, which resulted in a testing accuracy of 88.53% on the 70/30 split. Following the recommended procedure, increasing the shrinkage parameter from 0.01 to 0.2 results in an improved test accuracy of 95.11% on the HAR Generative Boosting model. Next, a grid search between 0.2 and 0.4 for the optimal shrinkage revealed a value of 0.335 resulting in a test accuracy of 95.21%. Traversing this shrinkage-to-test-error topology with a resolution of 0.005 for the shrinkage parameter yielded test accuracies ranging from 94.80% to 95.22%, suggesting no optimal minima have been missed. With this optimal shrinkage parameter, the confusion matrix of Figure 17 was found.

```
max_values_list   1   2   3   4   5   6
              1 489  23  10   0   0   0
              2   6 443   7   3   0   0
              3   1   3 394   0   0   0
              4   0   0   0 439  28   0
              5   0   2   9  49 504   0
              6   0   0   0   0   0 537

Generative Boost Test Accuracy:
95.21547 %
```

Figure 17: Confusion matrix of Generative Boosting method.

The number of trees within this model was set to 500 to ensure comparability with the Bagging and Random Forest ensembles, prevent overfitting, and because plotting the cross-validation against the number of trees grown (Figure 18), suggests minimal meaningful improvement beyond this point. However, this is not the only boosting method one could use, and the relatively recent method of XGBoosting presents the opportunity to improve further as is explored in the next section.

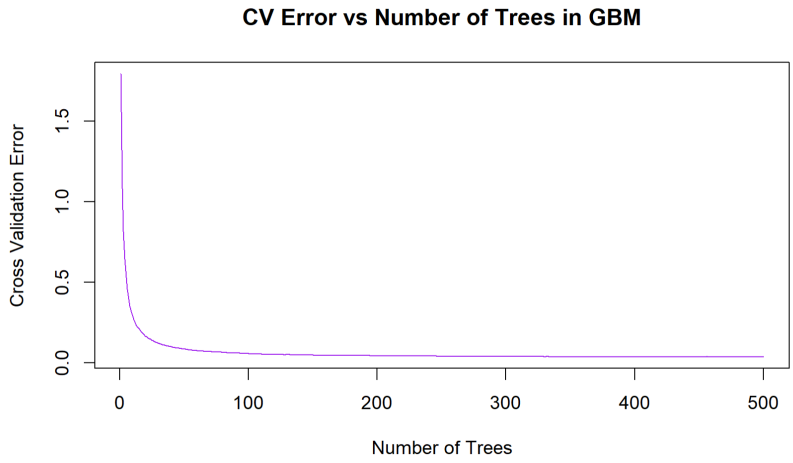

Figure 18: Cross-validation error vs number of trees from Generative Boosting method.

XGBoost

Extreme Gradient Boosting, better known as XGBoost, is a machine learning algorithm developed to increase model performance and computational speed [16] [17] when compared to other tree-boosting techniques. XGBoost combines boosted tree algorithms with gradient descent to maximize model performance [17]. For this reason, this algorithm was selected to follow the hierarchical pattern to increase the overall accuracy of the model.

The library *xgboost* includes a function titled *xgboost()* that performs the XGBoost algorithm using a multitude of hyperparameters such as eta, maximum depth, gamma columns sampled by tree, minimum child weight, subsamples, and number of rounds. Due to the large number of hyperparameters, a stepwise hyperparameter training method was integrated. This was performed similarly to how the hyperparameters cost and gamma were tuned using the *train()* function, however now with XGBoost, only a small group of hyperparameters were tuned at once while the subsequent groups were set as default. This reduced the dimensions of the search space and made it more manageable to find the optimized hyperparameters. In a similar method shown with the above decision tree methods, a confusion matrix for the XGBoost method was built (Figure 19), showing an overall test accuracy of 95.83% on the 70/30 split.

	Reference					
Prediction	1	2	3	4	5	6
1	489	18	4	0	0	0
2	6	447	22	2	0	0
3	1	5	394	0	0	0
4	0	0	0	442	17	0
5	0	1	0	47	515	0
6	0	0	0	0	0	537

Figure 19: Confusion matrix of XGBoost method.

Discussion and Conclusion

This report has demonstrated the successful replication of the study produced by Anguita et al. [9] who proved that the accelerator and gyroscope on a typical smartphone are sufficient to perform HAR with an accuracy range that falls within a dedicated HAR sensor range. Additional methods were employed to increase the model's accuracy, including a new data split ratio of 80/20, along with additional machine learning algorithms. The additional techniques used were Single Decision Tree, Bagging, Random Forest, Generative Boosting, and XGBoost. The accuracy of each new model improves sequentially, proving that with this dataset, in addition to SVM, the XGBoost is an effective method for HAR. The table below (Table 6) summarizes the overall accuracies obtained for the methods performed in this report as well as to illustrate their effectiveness. Notably, the XGBoost model outperforms the paper's model within the 80/20 split.

Table 6: Summary of overall test accuracy for all methods performed in this report.

	SVM Model	**Single Decision Tree**	**Bagging**	**Random Forest**	**Generative Boosting**	**XGBoost**
Test Accuracy (70/30 Split)	96.13%	83.61%	90.06%	92.67%	95.22%	95.83%
Test Accuracy (80/20 Split)	99.17%	86.41%	97.67%	98.20%	99.03%	99.37%

References

[1]C. Jobanputra, J. Bavishi, and N. Doshi, "Human Activity Recognition: A Survey," *Procedia Computer Science*, vol. 155, pp. 698–703, Jan. 2019, doi: https://doi.org/10.1016/j.procs.2019.08.100.

[2]A. Ferrari, D. Micucci, M. Mobilio, and P. Napoletano, "Trends in human activity recognition using smartphones," *Journal of Reliable Intelligent Environments*, vol. 7, no. 3, pp. 189–213, Jul. 2021, doi: https://doi.org/10.1007/s40860-021-00147-0.

[3]O. D. Lara and M. A. Labrador, "A Survey on Human Activity Recognition using Wearable Sensors," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1192–1209, 2013, doi: https://doi.org/10.1109/surv.2012.110112.00192.

[4]F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, and Y. Amirat, "Physical Human Activity Recognition Using Wearable Sensors," *Sensors*, vol. 15, no. 12, pp. 31314–31338, Dec. 2015, doi: https://doi.org/10.3390/s151229858.

[5]A. Rehman, T. Saba, Muhammad Zeeshan Khan, Robertas Damaševičius, and Saeed Ali Bahaj, "Internet-of-Things-Based Suspicious Activity Recognition Using Multimodalities of

Computer Vision for Smart City Security," vol. 2022, pp. 1–12, Oct. 2022, doi: https://doi.org/10.1155/2022/8383461.

[6]W. Sousa Lima, E. Souto, K. El-Khatib, R. Jalali, and J. Gama, "Human Activity Recognition Using Inertial Sensors in a Smartphone: An Overview," *Sensors*, vol. 19, no. 14, p. 3213, Jul. 2019, doi: https://doi.org/10.3390/s19143213.

[7]B. Ghaddar and J. Naoum-Sawaya, "High dimensional data classification and feature selection using support vector machines," *European Journal of Operational Research*, vol. 265, no. 3, pp. 993–1004, Mar. 2018, doi: https://doi.org/10.1016/j.ejor.2017.08.040.

[8]Abdullah Elen, Selçuk Baş, and Cemil Közkurt, "An Adaptive Gaussian Kernel for Support Vector Machine," *Arabian Journal for Science and Engineering*, vol. 47, no. 8, pp. 10579–10588, Mar. 2022, doi: https://doi.org/10.1007/s13369-022-06654-3.

[9]D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A Public Domain Dataset for Human Activity Recognition Using Smartphones," in *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2013)*, i6doc.com publ., pp. 437–442. Accessed: Nov. 16, 2024. [Online]. Available: http://www.i6doc.com/en/livre/?GCOI=28001100131010

[10]I. S. Al-Mejibli, J. K. Alwan, and D. H. Abd, "The effect of gamma value on support vector machine performance with different kernels," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 5, p. 5497, Oct. 2020, doi: https://doi.org/10.11591/ijece.v10i5.pp5497-5506.

[11]H. Alibrahim and S. A. Ludwig, "Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization," *2021 IEEE Congress on Evolutionary Computation (CEC)*, Jun. 2021, doi: https://doi.org/10.1109/cec45853.2021.9504761.

[12]R. Susmaga, "Confusion Matrix Visualization," *Intelligent Information Processing and Web Mining*, pp. 107–116, 2004, doi: https://doi.org/10.1007/978-3-540-39985-8_12.

[13]A. Gholamy, V. Kreinovich, and O. Kosheleva, "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation," *Departmental Technical Reports (CS)*, Feb. 2018, Available: https://scholarworks.utep.edu/cs_techrep/1209

[14]A. Pachi, "Frequency and velocity of people walking," *The Structural Engineer*, Feb. 2005, Available: https://api.semanticscholar.org/CorpusID:109115117

[15]G. James, D. Witten, T. Hastie, and R. Tibshirani, *INTRODUCTION TO STATISTICAL LEARNING : with applications in r.* S.L.: Springer-Verlag New York, 2021.

[16]A. Ogunleye and Q.-G. Wang, "XGBoost Model for Chronic Kidney Disease Diagnosis," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 17, no. 6, pp. 2131–2140, Nov. 2020, doi: https://doi.org/10.1109/tcbb.2019.2911071.

[17]T. Chen, “XGBoost: A Scalable Tree Boosting System,” *Cornell University*, Jun. 2016, doi: https://doi.org/10.48550/arXiv.1603.02754.