Human Activity Recognition Using Smartphones Dataset – Comparative Analysis

Sanghamitra Muhuri | smuhur@uic.edu | LinkedIn | Code Repository

1.Abstract

Human Activity Recognition is important technology behind countless applications in domains of health and medical research, human survey system, virtual reality and it is an active research topic for over a decade. Smartphones equipped with three-dimensional built-in sensing platforms like accelerometer and gyroscope, can capture the state of the user and revolutionize the way humans interact with smartphones. The HAR system captures the raw sensor readings in time and frequency domain as the inputs and estimates a human motion activity using data mining and machine learning techniques. In this paper, we intended to design and analyze the performance of five different classification algorithms from a series of observations on human activities like walking, walking upstairs, walking downstairs, lying down, sitting and standing. We have also attempted to address the various problems associated the system, along with improving the accuracy and reducing dependency on domain knowledge.

2.Introduction

With smartphones becoming a major part of daily human life, extensive research has been going on to widen the capability of applications that can be processed using a smartphone. These include many important applications on smartphone such as health monitoring, fall detection, human survey systems and home automation, etc. The most important technology involved in all such applications is that of Smartphone-based Human Activity Recognition (HAR) systems. The evolution of HAR systems is increasingly creating demand in health-care domain, physiotherapist assistance, and cognitive impairment support, etc. The applications like human survey systems and location indicator have wider applications too. Extensive training process forms the crux of the system and is a necessary procedure whenever a new activity is added to the system for recognition. Fine tuning and training of algorithm parameters are required to be done to implement the system on different devices with various built-in sensing devices. However, the process of labeling a training dataset is a time-consuming procedure. Due to negligible installation cost and robustness, smart phones are increasingly becoming the main platform for human activity recognition. In this paper, we have focused on applying different machine learning algorithms along with appropriate feature selection to find the best fit for HAR systems in terms of efficiency and accuracy.

3.Related Work

Human activity recognition has been studied extensively over the past years and researchers have proposed different solutions for the problem. The existing approaches typically use vision sensor, inertial sensor, and the mixture of both with application of Machine learning and threshold-base algorithms. Machine learning has led to more accurate, reliable results, whereas threshold-based algorithms have been found to be faster/simpler. Cameras have been used to capture body posture along with multiple accelerometers and gyroscopes attached to different body positions have been used as common solutions. Often vision and inertial sensors have also been used for the purpose. Data processing is the next essential part of using these ML algorithms. The results products highly vary with the quality of the input features fed to the algorithm. There have been previous works which were focused on synthesizing

the most useful features from the time series data set. Analyzing the signals obtained in both time and frequency domain has been found as a commonly used approach. These machine learning problems, however, have often been found as extensively time-consuming and labor expensive.

4.Modelling

To construct a good predictive model, we check the performance of different classification models which have been explained in brief below.

4.1 Decision Tree as Baseline Model

Decision Tree model is very intuitive, fast to run, and easy to interpret and implement. Decision tree model includes internal nodes labels by features, branches labeled by tests on that feature and leaves labeled with classes. Since decision tree model is an intuitive model and easy and fast to implement without any assumption, decision tree model can be used as baseline model to predict the human activities from the dataset.

Decision tree is a recursive approach create a leaf node with label of the class that all data belong. If data does not belong to the same class, score for each feature is calculated to spit the data. Data will be partitioned based on the feature with the highest score and called recursively. Decision tree be overfit when the model is bias towards the training data; therefore, create prediction error on unseen data. To prevent overfitting problem, we can stop building the tree early when the data has the same feature values, particular depth in the tree has been reached, a certain number or fraction of example remain, or particular training error has been reached. Other method is to prune the tree- remove some lower parts of the tree after building the decision tree. When splitting the data in decision tree model, features that provides highest score will be picked to partition data. The score can be based on entropy- how much uncertainty there is in the distribution over labels after the split, Gini- sum of square of the label proportions after split, or training error- misclassification error. Following is the formula to calculate Gini index and Entropy:

Gini index:

$$G(A) = 1 - \sum_{k=1}^{C} p_k^2 \quad (1)$$

$$Where\ p_k\ is\ proportion\ of\ cases\ in\ node\ A\ that\ belong\ to\ class\ k$$

Entropy:

$$E(S) = \sum_{i=1}^{C} -p_i \log_2 p_i \quad (2)$$

The algorithm for decision tree model is to calculate score for features of dataset based on criterion such as Gini index or Entropy. Based on the feature that has highest score, the data will be partitioned based on that data value. The process is repeated until all data belong to the same class or depth of the tree has been reached.

4.2 Logistic Regression Model

Logistic model is used to model the probability of certain classes by mapping $[-\infty, \infty]\ to\ [0,1]$ by logistic function $g(z) = \frac{1}{1+\exp(-z)}$. Given the dataset $D = \{x^{(i)}, y^{(i)}\}$, the log-likelihood is:

$$L(\theta; D) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta) = \prod_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}} \quad (3)$$

$$l(\theta; D) = logL(\theta; D) = \sum_{i=1}^{m} (y^{(i)} log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)) \quad (4)$$

Best parameter can be learned by maximizing log likelihood of the data D

$$\hat{\theta} = argmax_{\theta \in R^N} L(\theta) = argmax_{\theta \epsilon R^N} \log\left(L(\theta)\right) \quad (5)$$

$$\frac{\partial l(\theta; x, y)}{\partial \theta_j} = \left(y - h_\theta(x)\right) x_j \quad (6)$$

The learning algorithm of logistic regression is to use negative log-likelihood:

$$-\frac{\partial l(\theta; x, y)}{\partial \theta_j} = (h_\theta(x) - y) x_j \quad (7)$$

The vanilla gradient descent algorithm will be:

$$\theta_j \coloneqq \theta_j - \alpha \frac{-\partial l(\theta; x, y)}{\partial \theta_j} = \theta_j - \alpha (h_\theta(x) - y) x_j \quad (8)$$

After initiate theta, we will update the data with all example $D = \{\left(x^{(i)}, y^{(i)}\right)\}$ as following:

$$\theta_j \coloneqq \theta_j - \alpha \frac{-\partial l(\theta; x, y)}{\partial \theta_j} = \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) {x_j}^{(i)} \quad (9)$$

4.3 k Nearest Neighbor (kNN) Model

The kNN method is a non-parametric method that assumes similarity based on close proximity. Based on the number of k neighbors that we choose, a point will be compared with k neighbors that are closest in distance with that point, will be assigned to the same class that are most common among k nearest neighbors. The distance between a point to its k nearest neighbors can be calculated using Euclidean distance:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \quad (10)$$

$$(with\ point\ p\ with\ coordinate\ (p_1, p_2, \dots p_n) and\ q\ with\ coordinate\ (q_1, q_2, \dots q_n))$$

Given a query x' to predict y', kNN model will be:

$$y' = h(x') = arg\max_{y \in Y}\{\sum_{i \in kNN(x')} 1_{[y^{(i)}=y]}\} \quad (11)$$

In the algorithm of kNN model, a k number of neighbors are chosen. For each example in the dataset, distant between the query example and the current example from the data are calculated and sorted from smallest to largest. The label of the query sample will be the same as the common label for the first k entries from the sorted examples.

4.4 Naïve Bayes Model

Naïve Bayes model is a conditional probability model. Naïve Bayes classification is based on an assumption that values of each features are independent to values of other features. Naïve Bayes classifier based on Bayes' theorem:

$$p(C_k|x) = \frac{p(C_k) * p(x|C_k)}{p(x)} \quad (12)$$

In Naïve Bayes model, we try to assign example to class with highest probability as following:

$$\hat{y} = arg\max_{C_k} p\big(C_k\big|X_1,..,X_p\big) = arg\max_{C_k} p(C_k)\prod_i p(X_i|C_k) \quad (13)$$

With a continuous data, a typical assumption for the continuous values for each class is Gaussian distribution. Then with $\mu_k \; and \; \sigma_k^2 \;\; are \; mean \; and \; variance \; of \; value \; x \; in \; class \; C_k$ we have:

$$p(x|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} * e^{\frac{-(x-\mu_k)^2}{2\sigma_k^2}} \quad (14)$$

The algorithm for Naïve Bayes classifier model is to find probability for each class C_k that each predictor value occurs in class C_k. The product of these probability of each predictor is multiplied by proportion of examples in class C. Using Bayes' theorem, probability for class C is estimated. Class with highest probability will be assigned to the example.

4.5 Support Vector Machine (SVM)

Support Vector Machine (SVM) uses maximum margin hyperplane to separate classes. Decision boundary of separating hyperplane can be :

$$\{x|w^T x + b = 0\} \quad (15)$$

Positive examples will be:

$$\big\{x^{(i)}, y^{(i)}\big|w^T x^{(i)} + b > 0\big\} \quad (16)$$

Negative example will be:

$$\big\{x^{(i)}, y^{(i)}\big|w^T x^{(i)} + b < 0\big\} \quad (17)$$

Functional margin of (w,b) with respect to an example $(x^{(i)}, y^{(i)})$ is

$$\gamma^{(i)} = y^{(i)} * \left(w^T x^{(i)} + b\right) (18)$$

In hard SVM, we are trying to maximize functional margin by:

$$\max_{w,b} \frac{1}{\|w\|} \quad (19)$$

$$subject\ to\ y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1\ \forall i$$

With that we can change the argument to:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad (20)$$

$$subject\ to\ y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1\ \forall i$$

In case of non-linearly separable, we have slack variables ξ_i. The argument becomes:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \quad (21)$$

$$subject\ to\ y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi_i, \qquad \xi_i \geq 0\ \forall i$$

Each example is allowed to have a margin less than 1. C will be trade off between maximizing the margin and minimizing the slack. Algorithm for SVM is to find the hyperplane to separate the class that can maximize the margin and minimize the slack variables.

5.Experimental Results

5.1 Data Description

The data set is used from the UCI Machine Learning Repository "Human Activity Recognition Using Smartphones Data Set". The data was collected from 30 participants with age ranging from 19 to 48 years old. These participants were wearing a waist-mounted smartphone that recorded their movements. These movements including walking, walking upstairs, walking downstairs, sitting, standing and laying was recorded in x, y, z accelerometer data (linear acceleration) and gyroscopic data (angular velocity) from the Samsung Galaxy SII smartphone. Participants performed activities twice: the first trial with the smartphone was fixed on the left side of the belt and the second trial with the smartphone placed according to the participants' preferences. The frequency to record the data is 50 data points per second. This data set contains 561 attributes and 10299 number of instances. The data set will be split to 70/30 for training/testing.

The attributes were measured from the accelerometer and gyroscope 3 axials- XYZ raw signals and were presented as tAcc-XYZ and tGyro-XYZ, respectively (t is prefix for time). From the acceleration signal, we can obtain body and gravity acceleration signals (tBodyAcc-XYZ and tGravityAcc-XYZ). From the body linear acceleration and angular velocity within time, we can obtain Jerk signals (tBodyAccJerk-XYZ and tBodyGyroJerk-XYZ). Using the Euclidean norm, we can obtain these magnitudes: tBodyAccMag, tGravityAccMag, tBodyAccJerkMag, tBodyGyroMag, tBodyGyroJerkMag. Using Fast Fourier Transform (FFT) on some of these signals produces fBodyAcc-XYZ, fBodyAccJerk-XYZ, fBodyGyro-XYZ,

fBodyAccJerkMag, fBodyGyroMag, fBodyGyroJerkMag with 'f' to indicate frequency domain signals. These signals were estimated with other set of variables.

mean()	Mean value
std()	Standard deviation
mad()	Median absolute deviation
max()	Largest value in array
min()	Smallest value in array
sma()	Signal magnitude area
energy()	Energy measure. Sum of the squares divided by the number of values.
iqr()	Interquartile range
entropy()	Signal entropy
arCoeff()	Autoregression coefficients with Burg order equal to 4
correlation()	correlation coefficient between two signals
maxInds()	index of the frequency component with largest magnitude
meanFreq()	Weighted average of the frequency components to obtain a mean frequency
skewness()	skewness of the frequency domain signal
kurtosis()	kurtosis of the frequency domain signal
bandsEnergy()	Energy of a frequency interval within the 64 bins of the FFT of each window.
angle()	Angle between to vectors.

Table 1: Set of variables used to estimate signals

Averaging the signals, we can obtain: gravityMean, tBodyAccMean, tBodyAccJerkMean, tBodyGyroMean, tBodyGyroJerkMean.

Many variables are highly correlated to each other, since a signal is transformed from different functions, and can reduce the performance of our data. Therefore, we eliminate any variables that have correlation coefficient higher than 0.7. After cleaning those attributes, we have 107 attributes to work on for building models to predict data. The correlation plots between 561 attributes, top 20 attributes that have highest correlation coefficient, 107 attributes that we use to work on are summarized in the appendix

The number of samples in each label in the training data set was summarized in the graph below. Based on the graph, the number of labels in each class was well distributed; therefore, we there was no need to resample the data.

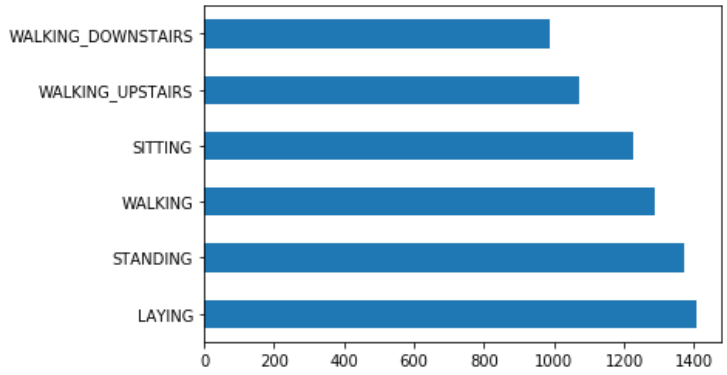

Figure 1: Summary of labels of activities

5.2 Modelling Implementation and its Challenges with Error Analysis and Outcome

5.2.1 Implementing Decision Tree

To find the best model for the decision tree, a grid search was performed with different parameters. Criterion was varied between Gini and Entropy method. Max depth was examined between 3 to 15. The grid search has shown that a decision tree with Gini method and max depth of 7 is the best model for the dataset. The decision tree model was performed on 107 attributes that produced lower than 0.7 for correlation coefficient. The results of the decision tree model are shown below:

Confusion Matrix

Training	Predict					
Actual	Laying	Sitting	Standing	Walking	Walking Downstairs	Walking Upstairs
Laying	1406	0	0	0	0	0
Sitting	0	1164	122	0	0	0
Standing	0	72	1302	0	0	0
Walking	0	0	0	1162	37	27
Walking Downstairs	0	0	0	14	939	33
Walking Upstairs	0	0	0	19	68	985

Table 2:Confusion Matrix of training set for decision tree

Testing	Predict					
Actual	Laying	Sitting	Standing	Walking	Walking Downstairs	Walking Upstairs
Laying	537	0	0	0	0	0
Sitting	0	367	124	0	0	0
Standing	0	65	467	0	0	0
Walking	0	0	0	424	58	14
Walking Downstairs	0	0	0	29	346	45
Walking Upstairs	0	0	0	49	53	369

Table 3:Confusion Matrix of testing set for decision tree

Summary Table

Training	precision	recall	f1-score	support	Accuracy	AUC (one vs rest)
LAYING	1	1	1	1407	0.95	1
SITTING	0.94	0.91	0.92	1286		0.95
STANDING	0.91	0.95	0.93	1374		0.96
WALKING	0.97	0.95	0.96	1226		0.97
WALKING_DOWNSTAIRS	0.9	0.95	0.92	986		0.97
WALKING_UPSTAIRS	0.94	0.92	0.93	1073		0.95

Table 4: Summary table for training set for decision tree

Testing	precision	recall	f1-score	support	Accuracy	AUC (one vs rest)
LAYING	1	1	1	537	0.85	1
SITTING	0.85	0.75	0.8	491		0.86
STANDING	0.79	0.88	0.83	532		0.91
WALKING	0.84	0.85	0.85	496		0.91
WALKING_DOWNSTAIRS	0.76	0.82	0.79	420		0.89

WALKING_UPSTAIRS	0.86	0.78	0.82	471		0.88

Table 5: Summary table for testing set for decision tree

For decision tree model, the accuracy for training set is 0.95 and for testing set is 0.85. The activity of Laying was predicted correctly for both training and testing set. Therefore, AUC and precision of predicting Laying is 1. There were some errors in distinguishing between sitting and standing, causing precision (on testing set) for sitting and standing labeling to be 0.85 and 0.79, respectively. AUC of ROC for sitting and standing label (on testing set) are 0.86 and 0.91, respectively. There were some false predictions between walking, walking downstairs and walking upstairs. The precisions for labeling Walking, Walking Downstairs and Walking Upstairs (on testing set) are 0.84, 0.76 and 0.86, respectively. AUC for Walking, Walking Downstairs and Walking Upstairs (on testing set) are 0.91, 0.89 and 0.88, respectively.

In order to examine the overall performance of the ROC curves, a micro average TPR and macro average TPR are calculated to draw ROC curves:

$$TPR_{macro} = \frac{1}{q} * \sum_{i=1}^{q} \left(\frac{TP}{TP + FN}\right)_i \quad (22)$$

$$TPR_{micro} = \frac{\sum_i^q TP_i}{\sum_i^q TP_i + FN_i} \quad (23)$$

The AUCs for micro average ROC curve and for macro average ROC curve are 0.98 in the training dataset. The AUCs for micro average ROC curve and for macro average ROC curve are 0.92 and 0.91, respectively, in the testing dataset.

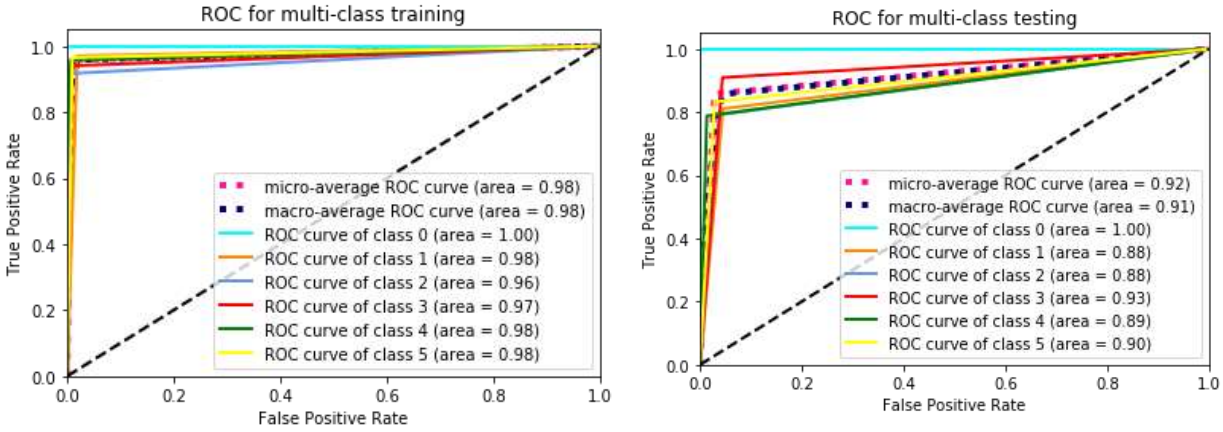

Figure 2: ROC curve for training and testing set for Decision tree model

5.2.1 Implementing Logistic Regression

For our baseline model we utilized sklearn to run a linear logistic regression algorithm. The parameter for multi-class was set to "multinomial" which uses the cross-entropy loss for model regularization. Random state of 0 was set to set seed used to shuffle the data. Any features with a correlation of more than 0.7 with any other attribute were removed from the dataset. The train and test set were standardized using

StandardScaler from sklearn.preprocessing so that independent feature wise scaling of the data is done such data is distributed with mean of 0 and standard deviation of 1.

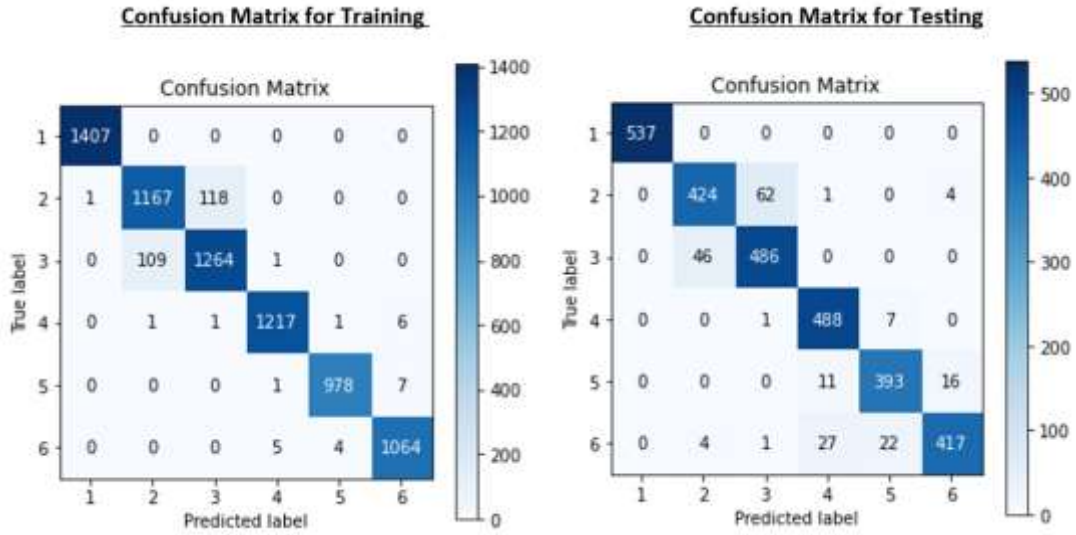

Figure 3: Confusion Matrix for training and testing set for Logistic Regression model

For Logistic Regression model, the accuracy for training set is 0.9711 and for testing set is 0.9243. The model performs very well but there were errors in distinguishing between the stationary postures of Sitting and Standing, resulting in lower testing precision of 0.89. Laying was perfectly predicted with precision of 1. Also, there were some false predictions between walking and walking downstairs with testing precisions of 0.90 and 0.92 respectively. This can be attributed to moderate slope of walk. Whereas, walking upstairs is predicted with precision of 0.95.

The overall performance can be seen from the below ROC curves

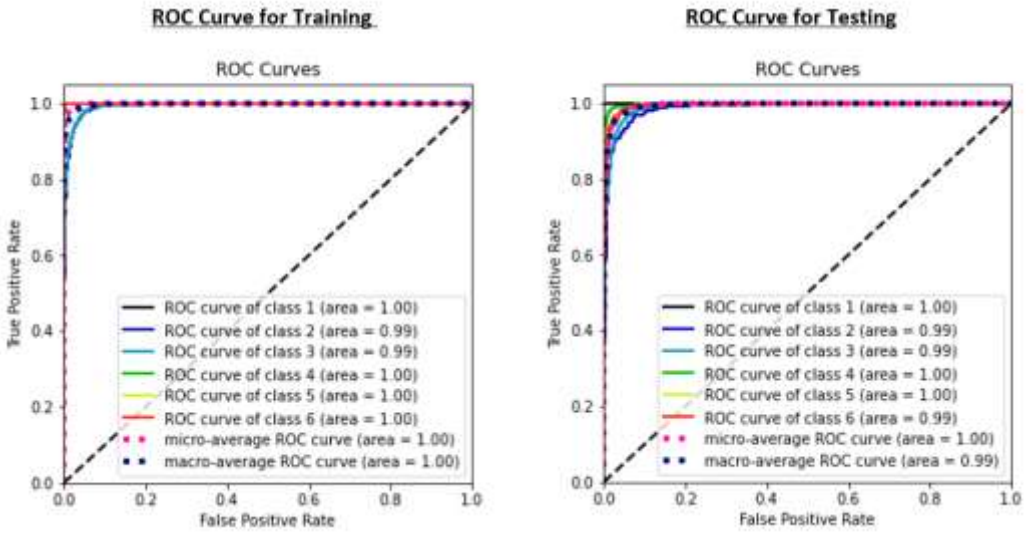

Figure 4: ROC curves for training and testing set for Logistic Regression

Training	precision	recall	f1-score	support	Accuracy	AUC (one vs rest)
LAYING	1	1	1	1407	0.97	1
SITTING	0.92	0.92	0.92	1286		0.99
STANDING	0.93	0.92	0.92	1374		0.99
WALKING	1	1	1	1226		1
WALKING_DOWNSTAIRS	1	1	1	986		1
WALKING_UPSTAIRS	1	1	1	1073		1

Table 6: Summary table for training set for Logistic Regression model

Testing	precision	recall	f1-score	support	Accuracy	AUC (one vs rest)
LAYING	1	1	1	537	0.92	1
SITTING	0.89	0.87	0.88	491		0.99
STANDING	0.89	0.91	0.9	532		0.99
WALKING	0.9	0.98	0.94	496		1
WALKING_DOWNSTAIRS	0.92	0.93	0.93	420		1
WALKING_UPSTAIRS	0.95	0.84	0.89	471		0.99

Table 7:Summary table for testing set for Logistic Regression model

Based on the accuracy and AUC for ROC curves, Logistic Regression model performed better than the baseline model (Decision tree model).

5.2.2 Implementing KNN

Among the most basic models that we use for classification on our data set is the KNN. The KNN is easy to implement but tends to get affected by the curse of dimensionality. Our data has a total of 559 attributes, a lot of which were highly correlated. So, we find out the attributes with a correlation of more than 0.7 with any other attribute and remove them from the data set. An important parameter that needs to be set for the KNN model is the value of K and it is always taken to be an odd number so that there is no chance of us getting a tie. We use the Euclidean distance and run our models over 7 different k values: 1 to 13 with all the odd numbers in between included. The training and testing accuracy along with the training and testing errors and plot of how the training and testing errors change for different values of K are as follows for the different k values:

K	Training Accuracy	Test Accuracy	Training_Error	Test_Error
1.0	1.000000	0.843909	0.000000	0.156091
3.0	0.983134	0.858839	0.016866	0.141161
5.0	0.976061	0.861215	0.023939	0.138785
7.0	0.970620	0.866983	0.029380	0.133017
9.0	0.964635	0.867662	0.035365	0.132338
11.0	0.960011	0.869019	0.039989	0.130981
13.0	0.956474	0.865287	0.043526	0.134713

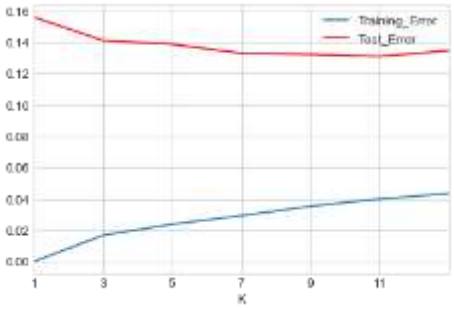

Figure 5: Table summarized training and testing accuracy and error (on the left) and plot of how the training and testing error changed for different values of K (on the right)

As we can see, the two best options to choose from are K=5 are K= 7. A higher K value leads to a higher training error but a smaller Test Error. We can use our discretion and select k=7 as the best parameter for our model. The resulting training and test confusion matrix for K=7 are:

Confusion Matrix

Training	Predict					
Actual	Laying	Sitting	Standing	Walking	Walking Downstairs	Walking Upstairs
Laying	1389	12	6	0	0	0
Sitting	9	1174	102	0	0	1
Standing	1	61	1309	2	0	1
Walking	0	0	0	1224	1	1
Walking Downstairs	0	0	0	11	972	3
Walking Upstairs	0	0	0	4	1	1068

Table 8: Confusion matrix for training set for Knn

Testing	Predict					
Actual	Laying	Sitting	Standing	Walking	Walking Downstairs	Walking Upstairs
Laying	511	17	8	0	0	1
Sitting	6	373	104	1	0	7
Standing	0	47	481	2	0	2
Walking	0	0	0	492	3	1
Walking Downstairs	0	0	0	29	346	45
Walking Upstairs	0	0	1	72	31	367

Table 9: Confusion matrix for testing set for Knn

Based on the accuracy, KNN model (with k=7) performed slightly better than the baseline model (Decision tree model).

5.2.1 Implementing Naïve Baiyes

The Naïve Bayes model was performed on 107 attributes that produced lower than 0.7 for correlation coefficient. The results of the Naïve Bayes model are shown below:

Confusion Matrix

Training	Predict					
Actual	Laying	Sitting	Standing	Walking	Walking Downstairs	Walking Upstairs
Laying	1405	0	0	0	0	2
Sitting	33	589	654	0	0	10
Standing	18	68	1281	6	0	1
Walking	0	0	0	1105	49	72
Walking Downstairs	0	0	0	18	878	90
Walking Upstairs	0	0	0	21	47	1005

Table 10: Confusion matrix for training set for Naive Bayes

Testing	Predict					
Actual	Laying	Sitting	Standing	Walking	Walking Downstairs	Walking Upstairs
Laying	537	0	0	0	0	0
Sitting	14	190	285	0	0	2
Standing	13	20	485	3	3	8
Walking	0	0	0	462	30	4
Walking Downstairs	0	0	0	27	348	45
Walking Upstairs	0	0	0	27	16	428

Table 11: Confusion matrix of testing set for Naive Bayes

Summary Table

Training	precision	recall	f1-score	support	Accuracy	AUC (one vs rest)
LAYING	0.96	1.00	0.98	1407	0.85	0.99
SITTING	0.90	0.46	0.61	1286		0.72
STANDING	0.66	0.93	0.77	1374		0.91
WALKING	0.96	0.90	0.93	1226		0.95
WALKING_DOWNSTAIRS	0.90	0.89	0.90	986		0.94
WALKING_UPSTAIRS	0.85	0.94	0.89	1073		0.95

Table 12: Summary table for training set for Naive Bayes model

Testing	precision	recall	f1-score	support	Accuracy	AUC (one vs rest)
LAYING	0.95	1.00	0.98	537	0.83	0.99
SITTING	0.90	0.39	0.54	491		0.69
STANDING	0.63	0.91	0.75	532		0.90
WALKING	0.89	0.93	0.91	496		0.95
WALKING_DOWNSTAIRS	0.88	0.83	0.85	420		0.90
WALKING_UPSTAIRS	0.88	0.91	0.89	471		0.94

Table 13: Summary table for testing set for Naive Bayes model

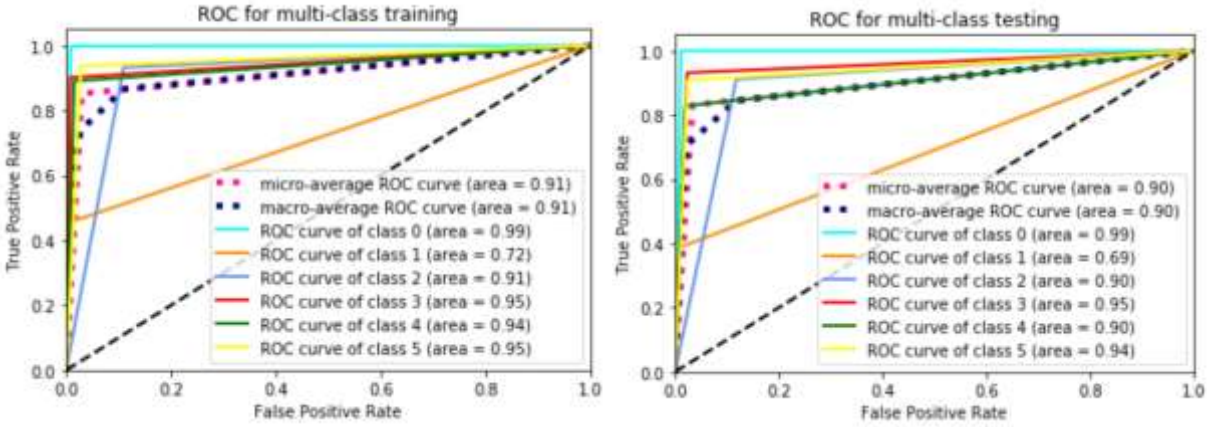

Figure 6: ROC curves for traing and testing set for Naive Bayes model

For Naïve Bayes model, the accuracy for training set is 0.85 and for testing set is 0.83. The activity of Laying was predicted almost correctly for both training and testing set with the precision to be 0.96 and 0.95 for training and testing, respectively. Therefore, AUC and precision of predicting Laying is 0.99. The prediction for Standing is poor with precision of 0.66 for training and 0.63 for testing. The precision to predict Sitting is high for training and testing ~0.9, but the recall for Sitting is really low- 0.46 for training and 0.39 for testing. AUC of ROC for sitting and standing label (on testing set) are 0.69 and 0.90, respectively. There were some false predictions between walking, walking downstairs and walking upstairs. The precisions for labeling Walking, Walking Downstairs and Walking Upstairs (on testing set) are 0.89, 0.88 and 0.88, respectively. AUC for Walking, Walking Downstairs and Walking Upstairs (on testing set) are 0.95, 0.90 and 0.94, respectively.

The AUCs for micro average ROC curve and for macro average ROC curve are 0.91 in the training dataset. The AUCs for micro average ROC curve and for macro average ROC curve are 0.90 in the testing dataset.

5.2.1 Implementing SVM

The Naïve Bayes model was performed on 107 attributes that produced lower than 0.7 for correlation coefficient. The results of the SVM model are shown below:

Confusion Matrix Training Set

Train	Precision	Recall	F1-score	Accuracy
1	1	1	1	96.98%
2	0.91	0.92	0.92	
3	0.93	0.92	0.92	
4	1	1	1	
5	1	1	1	
6	1	1	1	

Test	Precision	Recall	F1-score	Accuracy
1	1	1	1	92.65%
2	0.85	0.9	0.87	
3	0.91	0.87	0.89	
4	0.99	0.92	0.95	
5	0.94	0.91	0.93	
6	0.85	0.95	0.89	

Table 14: Summary for training (left) and testing (right) set for SVM model

Training	Predict					
Actual	Laying	Sitting	Standing	Walking	Walking Downstairs	Walking Upstairs
Laying	1407	0	0	0	0	0
Sitting	0	1169	117	0	0	0
Standing	0	96	1278	0	0	0
Walking	0	0	0	1224	1	1
Walking Downstairs	0	0	0	0	982	4
Walking Upstairs	0	0	0	1	2	1070

Table 15: Confusion matrix for training set for SVM model

Confusion Matrix Test Set

Test	Predict					
Actual	Laying	Sitting	Standing	Walking	Walking Downstairs	Walking Upstairs
Laying	537	0	0	0	0	0
Sitting	0	418	70	0	0	3
Standing	0	48	484	0	0	0
Walking	0	0	0	491	3	2
Walking Downstairs	0	0	0	0	395	18
Walking Upstairs	0	0	2	36	34	399

Table 16: Confusion matrix for testing set for SVM model

The accuracy of the SVM model for the training data is 96.98% and that on the test data is 92.65%. We use a Linear Kernel for this model. The Precision, recall and F1-Score for the model on the training and testing data are:

6.Conclusion or Discussion

The biggest problem that one faces while building a model is that of overfitting. In case of our dataset and the models we have implemented, we see that we did not face an aggravated issue of overfitting. The best results for the data set were provided by the Logistic Regression and SVM. The SVM algorithm can

be considered as the most sophisticated algorithm that we have implemented for our project and in most cases, one can expect it to provide decent results which we see happening in our project too. The logistic regression, which is most often the first classification algorithm that people learn comes in a close second on account of its test accuracy. SVM can be a hard model to implement when our data is not in the vectorized format. Luckily for us, we had vectorized data before-hand and the data was not of a size where it became costly to train the SVM model, thereby making the SVM was a viable option.

In the future, we would first of all try to gather enough domain knowledge so as to implement different kernels for the SVM and also try to implement an XGBoost model on out dataset to find out the most robust and best performing algorithm on this HAR dataset.

7.References

[1] A. Rasekh, C.-A. Chen and Y. Lu, "Human Activity Recognition using Smartphone," Texas A&M University, 2011.

Appendix

Correlation plot between 561 attributes is shown below:

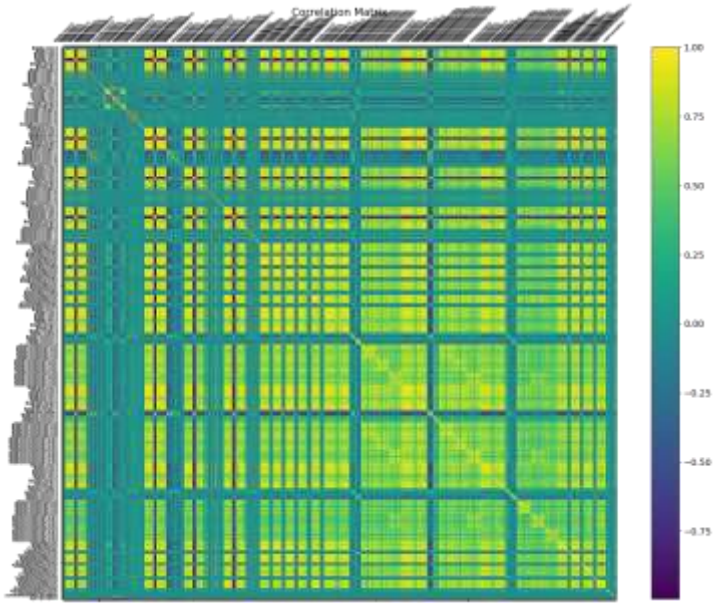

Figure 7: Correlation plot between 561 attributes

20 attributes that have highest correlation coeffiecients from the correlation plot were shown below:

fBodyAcc-energy()-X	fBodyAcc-bandsEnergy()-1,24	0.999878
fBodyAcc-bandsEnergy()-1,24	fBodyAcc-energy()-X	0.999878
fBodyGyro-bandsEnergy()-1,24	fBodyGyro-energy()-X	0.999767
fBodyGyro-energy()-X	fBodyGyro-bandsEnergy()-1,24	0.999767
fBodyAcc-energy()-Y	fBodyAcc-bandsEnergy()-1,24.1	0.999661
fBodyAcc-bandsEnergy()-1,24.1	fBodyAcc-energy()-Y	0.999661
tBodyAccJerkMag-mean()	tBodyAccJerk-sma()	0.999656
tBodyAccJerk-sma()	tBodyAccJerkMag-mean()	0.999656
tBodyAccJerkMag-mean()	tBodyAccJerkMag-sma()	0.999656
tBodyAccJerkMag-sma()	tBodyAccJerk-sma()	0.999656
tBodyAcc-energy()-X	fBodyAcc-energy()-X	0.999611
fBodyAcc-energy()-X	tBodyAcc-energy()-X	0.999611
fBodyGyro-energy()-Z	fBodyGyro-bandsEnergy()-1,24.2	0.999523
fBodyGyro-bandsEnergy()-1,24.2	fBodyGyro-energy()-Z	0.999523
tBodyAcc-energy()-X	fBodyAcc-bandsEnergy()-1,24	0.999507
fBodyAcc-bandsEnergy()-1,24	tBodyAcc-energy()-X	0.999507
tBodyGyro-energy()-Y	fBodyGyro-energy()-Y	0.999242
fBodyGyro-energy()-Y	tBodyGyro-energy()-Y	0.999242
tBodyGyroMag-sma()	tBodyGyro-sma()	0.999234
tBodyGyroMag-mean()	tBodyGyro-sma()	0.999234

Table 17: 20 attributes that have highest correlation coefficients from correlation plot

107 attributes that have lower than 0.7 correlation coefficients and are used to build predictive models:

tBodyAcc-mean()-X
tBodyAcc-mean()-Y
tBodyAcc-mean()-Z
tBodyAcc-std()-X
tBodyAcc-arCoeff()-X,4
tBodyAcc-arCoeff()-Y,1
tBodyAcc-arCoeff()-Y,4
tBodyAcc-arCoeff()-Z,1
tBodyAcc-correlation()-X,Y
tBodyAcc-correlation()-X,Z
tBodyAcc-correlation()-Y,Z
tGravityAcc-mean()-X
tGravityAcc-mean()-Z
tGravityAcc-std()-X
tGravityAcc-std()-Y
tGravityAcc-std()-Z
tGravityAcc-sma()
tGravityAcc-entropy()-X
tGravityAcc-entropy()-Y
tGravityAcc-entropy()-Z
tGravityAcc-arCoeff()-X,1

tGravityAcc-arCoeff()-Y,1
tGravityAcc-arCoeff()-Z,1
tGravityAcc-correlation()-X,Y
tGravityAcc-correlation()-X,Z
tGravityAcc-correlation()-Y,Z
tBodyAccJerk-mean()-X
tBodyAccJerk-mean()-Y
tBodyAccJerk-mean()-Z
tBodyAccJerk-arCoeff()-X,4
tBodyAccJerk-arCoeff()-Y,4
tBodyAccJerk-arCoeff()-Z,4
tBodyAccJerk-correlation()-X,Y
tBodyAccJerk-correlation()-X,Z
tBodyAccJerk-correlation()-Y,Z
tBodyGyro-mean()-X
tBodyGyro-mean()-Z
tBodyGyro-entropy()-X
tBodyGyro-entropy()-Y
tBodyGyro-arCoeff()-X,1
tBodyGyro-arCoeff()-X,4
tBodyGyro-arCoeff()-Y,1

tBodyGyro-correlation()-X,Y
tBodyGyro-correlation()-X,Z
tBodyGyro-correlation()-Y,Z
tBodyGyroJerk-mean()-X
tBodyGyroJerk-mean()-Y
tBodyGyroJerk-mean()-Z
tBodyGyroJerk-arCoeff()-X,3
tBodyGyroJerk-arCoeff()-X,4
tBodyGyroJerk-arCoeff()-Y,4
tBodyGyroJerk-arCoeff()-Z,4
tBodyGyroJerk-correlation()-X,Y
tBodyGyroJerk-correlation()-X,Z
tBodyGyroJerk-correlation()-Y,Z
tBodyAccMag-arCoeff()1
tBodyAccJerkMag-arCoeff()1
tBodyAccJerkMag-arCoeff()3
tBodyAccJerkMag-arCoeff()4
tBodyGyroMag-arCoeff()1
tBodyGyroJerkMag-arCoeff()1
tBodyGyroJerkMag-arCoeff()3
tBodyGyroJerkMag-arCoeff()4

fBodyAcc-min()-Y
fBodyAcc-min()-Z
fBodyAcc-maxInds-X
fBodyAcc-maxInds-Y
fBodyAcc-maxInds-Z
fBodyAcc-meanFreq()-Y
fBodyAcc-meanFreq()-Z
fBodyAcc-skewness()-X
fBodyAcc-skewness()-Y
fBodyAcc-skewness()-Z
fBodyAccJerk-min()-Y
fBodyAccJerk-min()-Z
fBodyAccJerk-maxInds-X
fBodyAccJerk-maxInds-Y
fBodyAccJerk-maxInds-Z
fBodyAccJerk-skewness()-X
fBodyAccJerk-skewness()-Y
fBodyAccJerk-skewness()-Z
fBodyAccJerk-bandsEnergy()-57,64
fBodyAccJerk-bandsEnergy()-57,64.1
fBodyAccJerk-bandsEnergy()-57,64.2
fBodyGyro-min()-X

fBodyGyro-min()-Y
fBodyGyro-min()-Z
fBodyGyro-maxInds-X
fBodyGyro-maxInds-Y
fBodyGyro-maxInds-Z
fBodyGyro-meanFreq()-X
fBodyGyro-meanFreq()-Y
fBodyGyro-meanFreq()-Z
fBodyGyro-skewness()-X
fBodyGyro-skewness()-Y
fBodyGyro-skewness()-Z
fBodyAccMag-min()
fBodyAccMag-maxInds
fBodyAccMag-skewness()
fBodyBodyAccJerkMag-maxInds
fBodyBodyAccJerkMag-skewness()
fBodyBodyGyroMag-maxInds
fBodyBodyGyroMag-skewness()
fBodyBodyGyroJerkMag-maxInds
fBodyBodyGyroJerkMag-skewness()
angle(tBodyAccMean,gravity)
angle(tBodyGyroJerkMean,gravityMean)

Figure 8: Attributes used to build models