DATA ANALYTICS FOR HUMAN ACTIVITY RECOGNITION USING SMARTPHONE SENSORS

ISEN 613: ENGINEERING DATA ANALYSIS PROJECT

INSTRUCTOR: DR. NA ZOU

TEXAS A&M UNIVERSITY, COLLEGE STATION
DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

PROJECT REPORT BY - GROUP NUMBER 2

TEAM MEMBERS:

NAME	UIN	CONTRIBUTION
BHAVESH HASTIMAL JAIN	426009682	20% - QSVM, Random Forest, PCA, Ensembles, Bagging , LDA, QDA
ABHISHEK NITIN KSHIRSAGAR	226007820	20%- Linear SVM, PCA, Neural Network, Report Writing
NIMISH VIDYADHAR OJALE	926008327	20% - LDA, QDA, Multinomial Logistic Regression
SUKHADA SHANTANU SAOJI	326005803	20%- LASSO, Ridge, Neural Network, Literature Review, Report Writing
SANTOSH SHYAMALA RAMASUBRAMANIAN	525005188	20%- EDA, KNN, PCA, Hierarchical Clustering, Data Preprocessing

Table of Contents

Executive Summary:

With the increasing vested interest in human health, it is highly essential to obtain precise and opportune patient information to deliver the best healthcare support. Hence, the real time patient's activities and behaviors monitoring data holds paramount importance in innumerable applications in medical science. In the past decade, there has been an exceptional growth of sensing technologies, microelectronics, and smart mobile devices. Although the wearable sensors are being extensively used for health status monitoring, they lack the convenience of usage. Smartphones have become pervasive and hence prove to provide a viable solution for regular activity monitoring as everyone carries a mobile device having various smart sensors like- accelerometers, gyro sensors, and GPS sensors. These sensors can be used to capture the human movements and the raw sensor data could be processed further to derive useful conclusions about health of the user.

The project focuses on the implementation of various statistical models and techniques for accurately classifying the response in one of the six activities being monitored namely: Laying, Sitting, Standing, Walking, walking downstairs, walking upstairs based on the available sensor data acquired from smartphones. Various supervised and unsupervised techniques have been tried by several researchers on the HAR dataset for building up an accurate predictive model. This project brings up a novel approach of identifying the most important features from the large number of available predictors and identifying the most accurate classification model for human activity recognition. Exploratory data analysis - K means clustering, Hierarchical clustering has been performed to understand the data distribution in depth and come up with useful models. Further, feature selection is approached by implementation of PCA, Random Forest, Multinomial LASSO and Ridge Regression. The project discusses in depth the implementation of LDA, QDA, Multinomial Logistic Regression, Quadratic SVM using Radial Kernel, Linear SVM, KNN, and Neural Network for classification. The results have been compared for all the models by performing analysis on the original data and the feature reduced data obtained after PCA.

Introduction:

Problem Statement:

Human Activity Recognition - HAR - using low cost sensor inputs is an emerging field of research in recent years having applications across different industries. An IDC study suggests that there are going to be more than 212 billion sensor enabled objects by the year 2020. Based on a research by IoT Analytics, there has been an increasing trend in popularity of smart wearable devices and connected health applications. However, fast and accurate prediction of outputs using sensor data as an input still posits a wide gap in research. Mobile phones have a reach of more than 1 per 2 people on planet earth. Making use of the continuous stream of data received from accelerometer and gyroscopic sensors available from these devices, we can develop systems that can positively impact the lives of millions. In healthcare industry, HAR finds many potential applications, viz.: elderly monitoring, life log systems for monitoring energy expenditure and for supporting weight-loss programs, and digital assistants for weight lifting exercises
(Ref: http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises).

Importance of the Problem:

Human activity recognition plays a vital role in real time patient health monitoring and assistive feedback for treatment. It is highly essential for patients suffering from heart diseases, obesity, and diabetes to follow a regular exercise regime. Thus, the recognition of human activities like standing, walking, and running helps the caretaker to give a constructive feedback to the patient. Similarly, activity recognition is important to monitor the real time positive or negative responses of patients admitted in the Intensive Care Unit (ICU). Human activity recognition also plays a vital role in detecting abnormalities in patients suffering from mental illness like dementia.

Objective:

Data acquired through accelerometers and gyroscopes in the phones attached to users can be used for several impactful applications in the field of healthcare. By properly analyzing the data our objective is to develop an accurate predictive model which can classify the type of motion based on the accelerometer and gyroscope readings. Doing this will enable us to use the model for monitoring and predicting activities a human is undergoing, which can later be used to calculate level of activity as well as for other applications as stated above.

Literature Review:

Paper Reviews

1. *"A comparative predictive analysis of neural networks (NNs), nonlinear regression and classification and regression tree (CART) models" by Muhammad A. Razi*, Kuriakose Athappilly*

This paper discusses the comparison of nonlinear regression, Neural Networks and Classification and Regression Tree (CART) models for a large dataset on smoking habits of people. Neural Network was chosen for analysis as it handles the non-linearities in the data very well. CART is a non-parametric approach for predicting continuous dependent variable with categorical predictor variables, which was a perfect fit for the given dataset. Model goodness of fit measure is not a good measure for measuring accuracy in prediction models. Measures like Mean Absolute Percentage Error (MAPE), Mean Squared Error (MSE), and Large Prediction Error (LPE) were used to compare the model performance. It was observed that the Neural Network method yields the lowest MAE, MAPE and MSE values. However, Regression seems to provide consistently higher values of errors) compared to other two models.
Conclusion: The study shows that NNs and CART models provide better prediction compared to regression models when we have binary or categorical predictor variables and the dependent variable continuous over regression.

2. *Transition-Aware Human Activity Recognition Using Smartphones by Jorge-L.Reyes-Ortiz a,b,n, Luca Oneto c, Albert Samà b, XavierParra b, Davide Anguita*

The paper discusses how activity is acquired using various sensors like accelerometers. It describes the data acquisition and sensor processing system for human activity data using smartphones. Further, it discusses the application of a multiclass linear SVM. It comprises of different One-vs-All (OVA) binary SVMs which characterize each of the one studied activities. In the one vs all approach the probability output of each SVM class is compared against others to find out the class c with the maximum a posteriori (MAP) probability. The PrSVM model achieved an error rate of 6.96% which was further improved using advanced filtering techniques.

3. *Unsupervised learning for human activity recognition using smartphone sensors by Yongjin Kwon, Kyuchang Kang, Changseok Bae*

In this paper unsupervised learning approach is discussed considering the number of activities k is unknown. Feature extraction was performed using Fast Fourier Transform and sample space was divided into 24 features. After feature extraction data was normalized within [0,1] for analysis. Four clustering algorithms are implemented and measured against each other- k-means clustering, mixture of Gaussian (GMM), average-linkage hierarchical agglomerative clustering (HIER), and DBSCAN. GMM gives the highest accuracy when k is assumed to be known. Even when k is unknown, unsupervised learning methods the high recognition accuracy can be achieved by selecting k based on CH index.

Conclusion: The unsupervised learning approach reduces the problem usually faced in the implementation of supervised algorithm, which is the difficulty of dividing the huge train data in two parts flawlessly for testing purposes.

Scope of Work:

The data has been acquired in raw form. The scope of the project will be all stages including data cleaning, dimension reduction, feature selection, model building, and model selection. Using validation methods, we aim to produce an accurate predictive model.

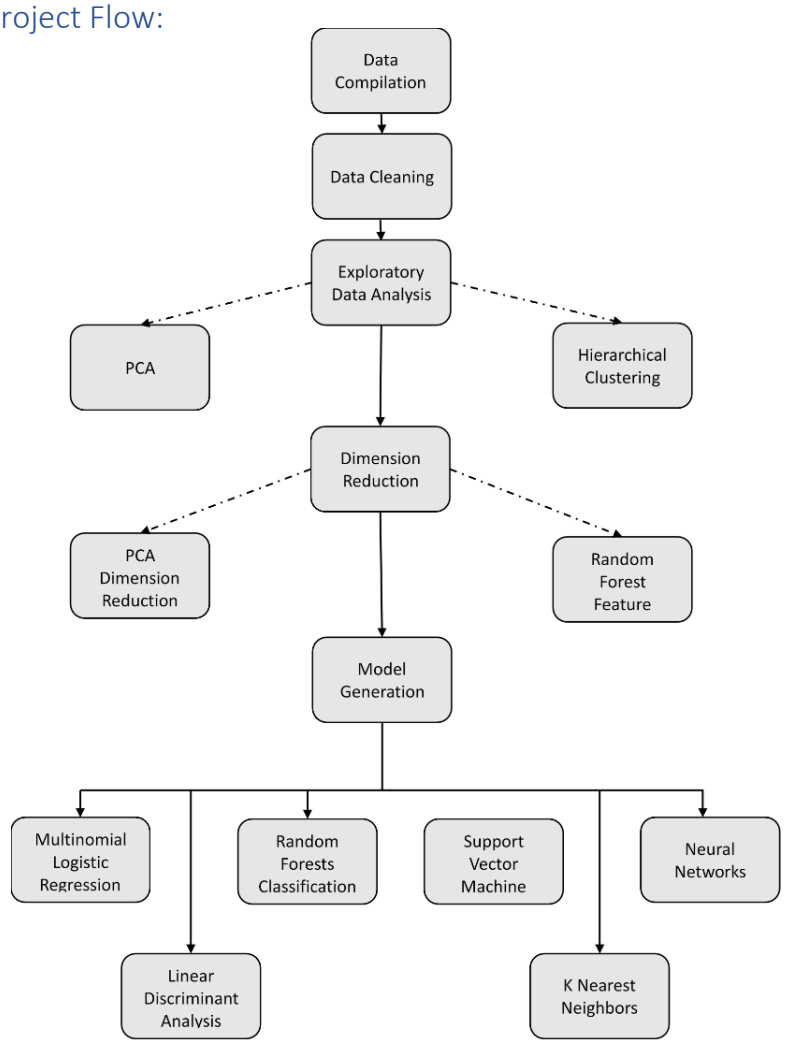
Project Approach:

Data Description:

The dataset that we are planning to use is obtained from 30 subjects doing Activities of Daily Living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors, which is released to public domain on a well-known on-line repository(UIC). The subjects are within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. The signal that was collected form the accelerometer and gyroscope in the phone, the DAQ system captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The activity was recorded using cameras and the dataset was then manually labelled with activities by the creators of the data. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers were selected for generating the training data and 30% the test data.

We dived into how the dataset was created to get a better understanding of the predictors being used in our model. We found out that the dataset obtained from the accelerometer and gyro sensors was a time-series data which was denoised using multiple noise filters. Using a time-domain analysis or a frequency domain analysis (obtained using Fourier transformations) would have restricted us to a lesser number of predictors. A data transformation approach was used using advanced signal processing techniques. The data was transformed using windowing function with sliding window size of 2.56 sec and a 50 % overlap which lead to generation of 128 readings/window. The transformed data thus produced a spatio-temporal data matrix of size 561 predictors. With an additional column of a classifier and subject number added manually, the data dimension available for modelling is 563.

Project Flow:



In data analysis, data compilation and cleaning play a major role in ensuring that the data is consistent and can yield proper results on applying analysis techniques. We perform unsupervised methods like PCA and Clustering to study the data and get a feel of its behavior. Further we perform Random Forest and PCA to simultaneously reduce dimensions and study the importance of different predictors against each other.

When the data has been reduced to lower dimensions and consists only of important predictors, we start building models for classification. Using computationally easy validation methods like k-fold CV and Validation set, we will obtain the best model out of all the ones we generated.

Implementation & Methodology:

Preprocessing:

Test and train datasets were available separately. We used a correlation factor of 0.8 to remove uncorrelated predictors. There were no missing/ NA values present in the data. When required activities were binary encoded (0 to 5) in R.

Exploratory Analysis:

Exploratory data analysis (EDA) is the method to analyze datasets to identify the important characteristics by visual representation as well as quantitative methods. Basically, EDA is an approach to look at the data and identify trends, patterns and clusters.

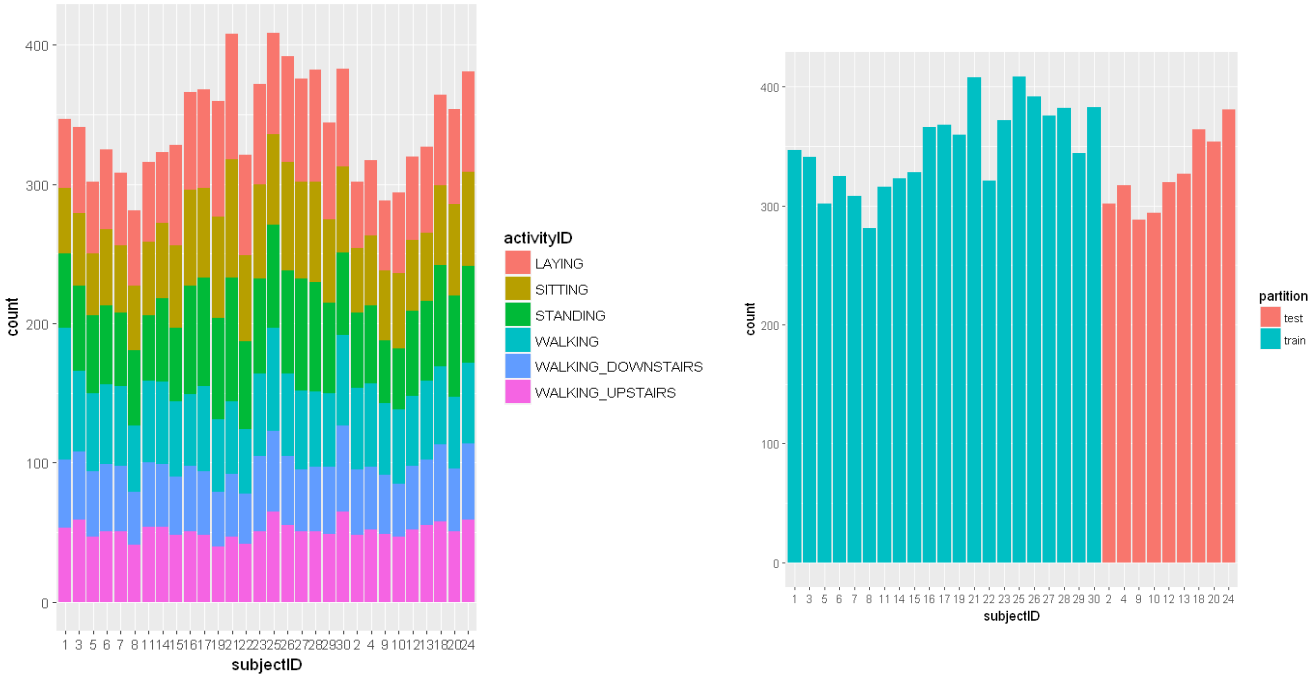


Figure 1. Exploratory Data Analysis on Train and Test Data

Hierarchical Clustering

The following is the result of Hierarchical Clustering using Euclidean distance and complete distance from Subject 1's data:

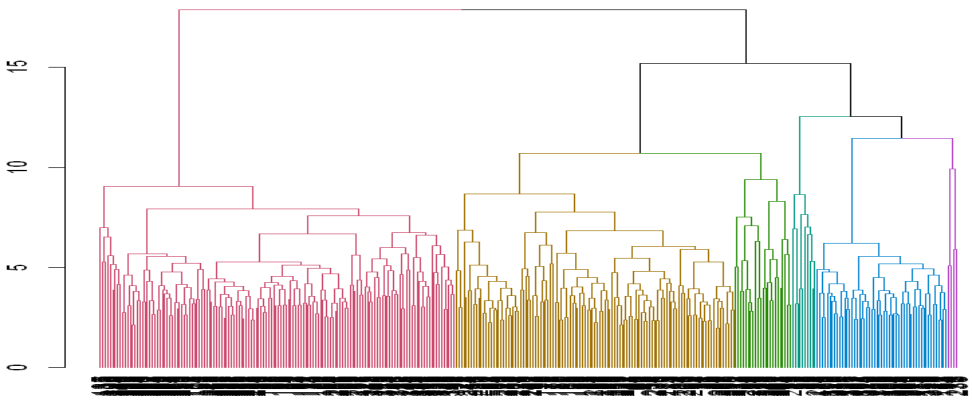


Figure 2: Hierarchical Clustering of Sub 1

k-Means Clustering:

The following is the result of K-Means clustering:

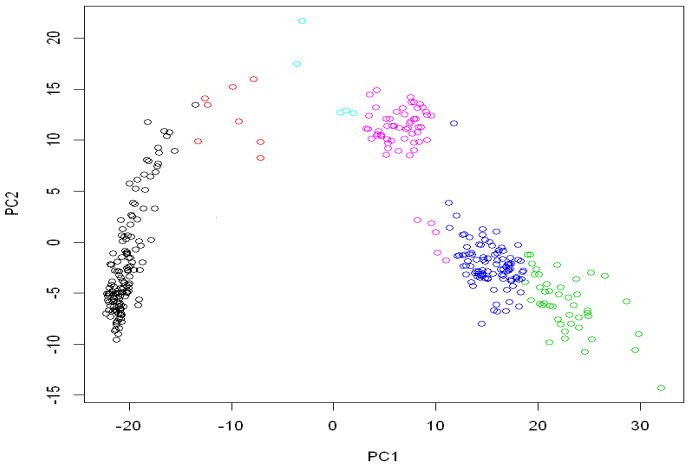


Figure 3: k Means Cluster Scatterplot

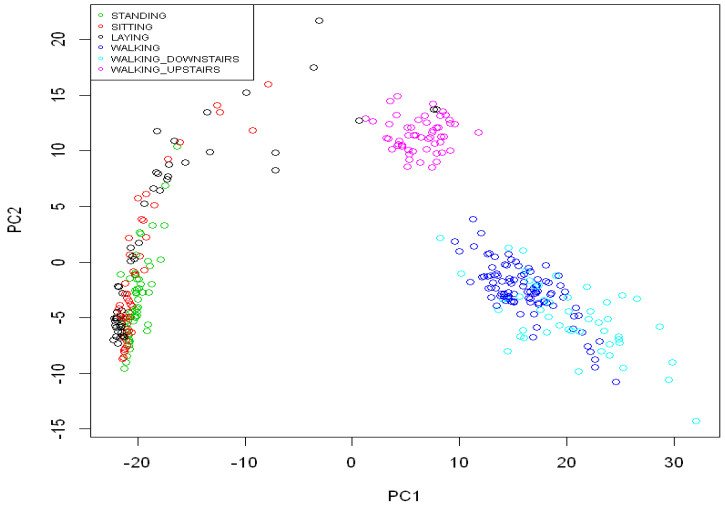


Figure 4: Actual Clustering of activity

PCA

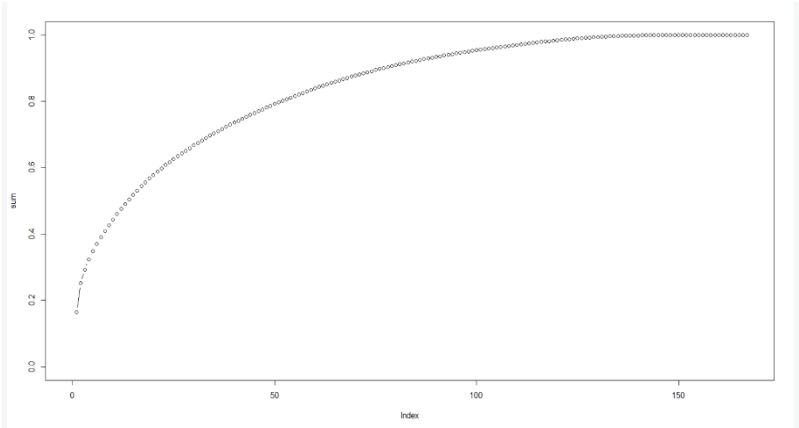


Figure 5. PVE vs Predictors

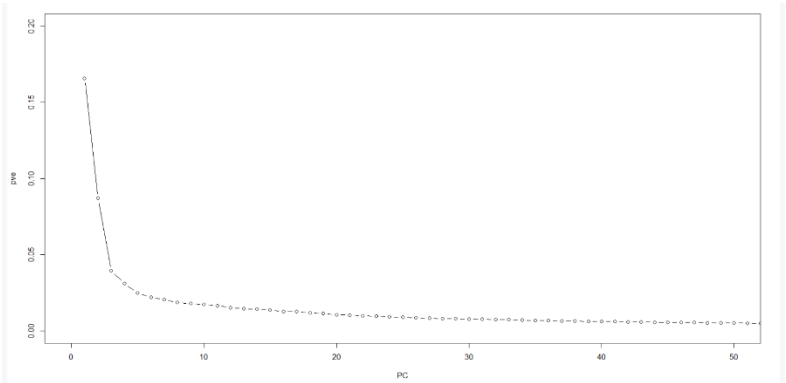


Figure 6. PVE vs PCs

We performed PCA as an unsupervised learning approach to identify important predictors and the distribution of variance amongst the predictors. With the help of an R package, we could identify the top 52 components explaining ~80 percent of the variance of the predictors (uncorrelated data of 168 predictors). The rotation, scale and other parameters of these components were then stored which was then used for training and testing the data. By using PCA we could identify ~30% data that explained ~80% of the variance, thus achieving a reduction of ~70% in the number of parameters. This aided us in developing models that were comparatively less complex and provided comparable results with relatively lower computational time.

Random forest feature selection:

We performed a random forest training using R on the data points. The algorithm helps us to identify the key predictors. We use the meandecreasegini plot to identify the key predictors. The GINI importance is a measure of the average gain of purity obtained by splits of a given variable. If a predictor is useful, it tends to split mixed label splits into pure nodes having single class thus aiding classification. If a split is performed with a variable that is not that important, It does not contribute towards improving the purity of a node. Thus, permuting an important predictor leads to a relatively larger decrease in mean gini-gain. Thus a gini index is can be easily computed with a tree-based method like the random forest. We use the inbuilt R functions to predict and plot the meandecreasegini.

Looking at these values and plots we can identify the most important predictors. In our data, these predictors might point us towards the key frequency bands of the data. This data can later be used for selecting appropriate sensor selection and design. Knowing the important or the active frequency bands corresponding to the activities that we intend to classify can be used to apply appropriate low-pass and high-pass filters aiding us in a significant reduction of frequency bandwidth. This would also eliminate the noise in the data and help us improve prediction. Unfortunately, we do not have access to the method used for generating the spatiotemporal features used as predictors in our data, hence we could not perform this analysis. This can be outlined and added as a good candidate for further unsupervised analysis of this dataset if the necessary details are available.

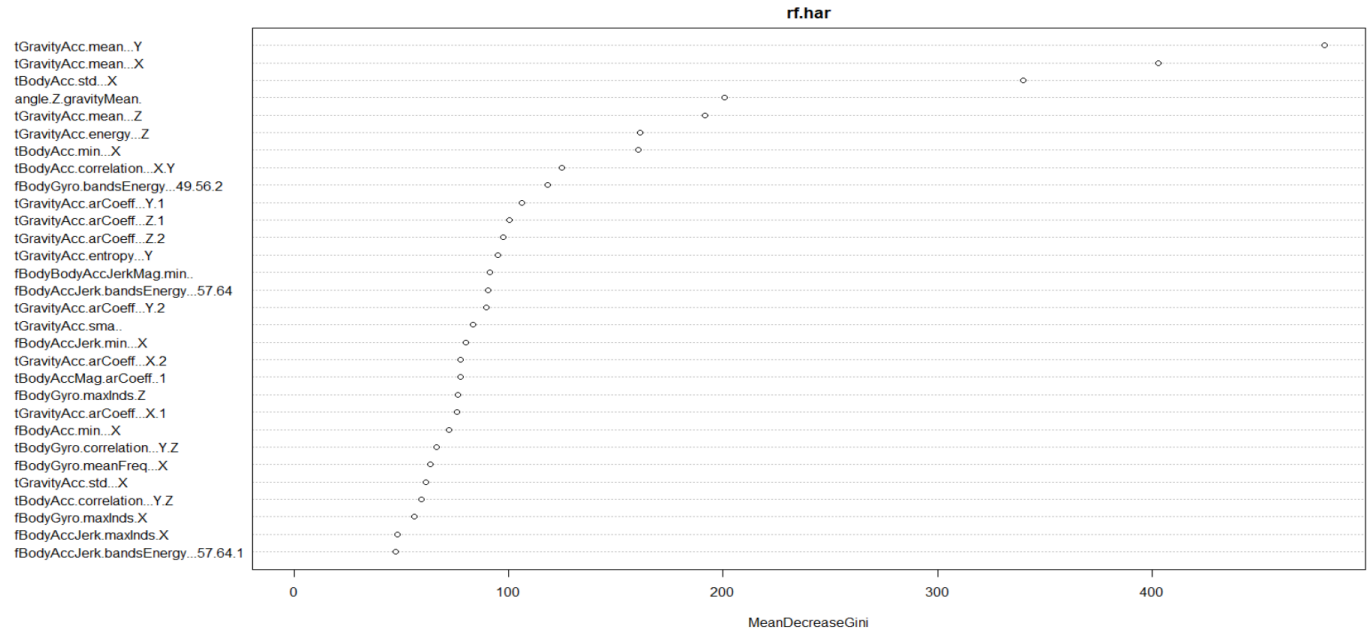


Figure 7. meandecreasegini plot

Correlation

Columns with correlation greater than 0.8 were removed. This was done considering the nature of the data.

Lasso & Ridge

LASSO is a regression method that performs variable selection by shrinking the coefficients of insignificant independent variables to exactly zero. LASSO can be applied to a classification problem to obtain the most important predictors for a given class. For implementing Multinomial LASSO for the HAR dataset, classes should be coded as binary variable. Using the 'glmnet' package and selecting 'multinomial' family significant predictors related to each activity can be found out. R outputs a list which contains 6 sparse matrices, each comprising of the important predictors for the 6 levels of activities which are coded as binary numbers from 0 to 5. Similarly, ridge multinomial regression is also performed to identify important features.

The following plots show the magnitude of coefficients for all the 166 predictors plotted against each activity. Activity Laying has the most number of uninfluential predictors. From the plots we can see that LASSO efficiently shrinks many of the predictors to zero while Ridge regression fails to do so. It is difficult to extract a subset of all the important predictors making sense for the overall classification model. Thus, we have not used LASSO/Ridge for feature selection of our classification model.

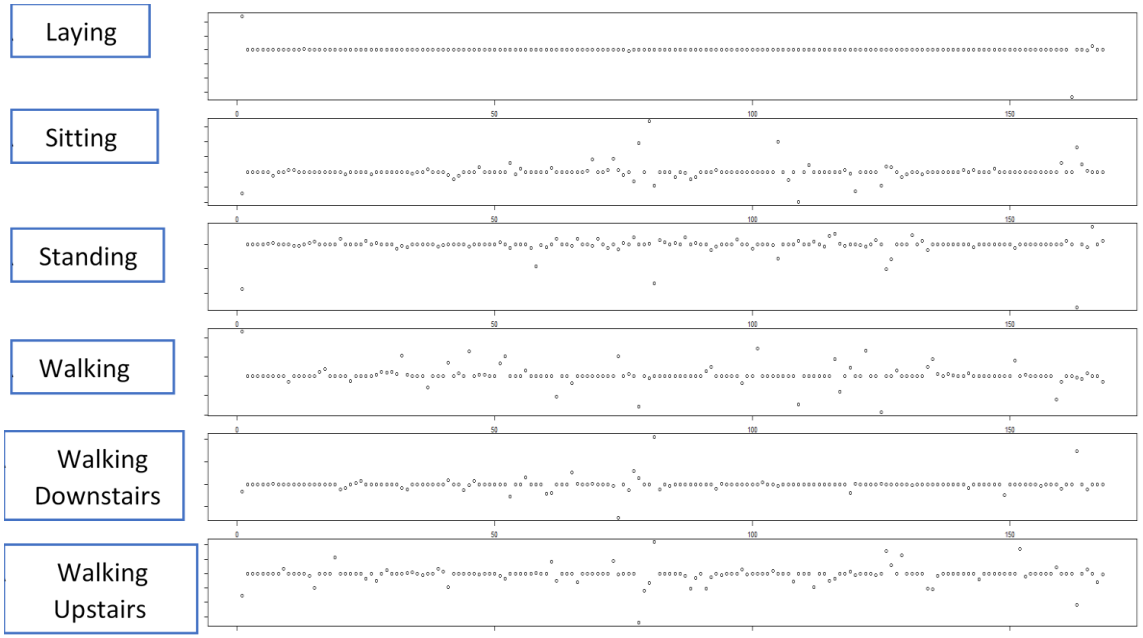


Figure 8. Magnitude of Coefficient obtained from LASSO Multinomial Regression vs Predictors

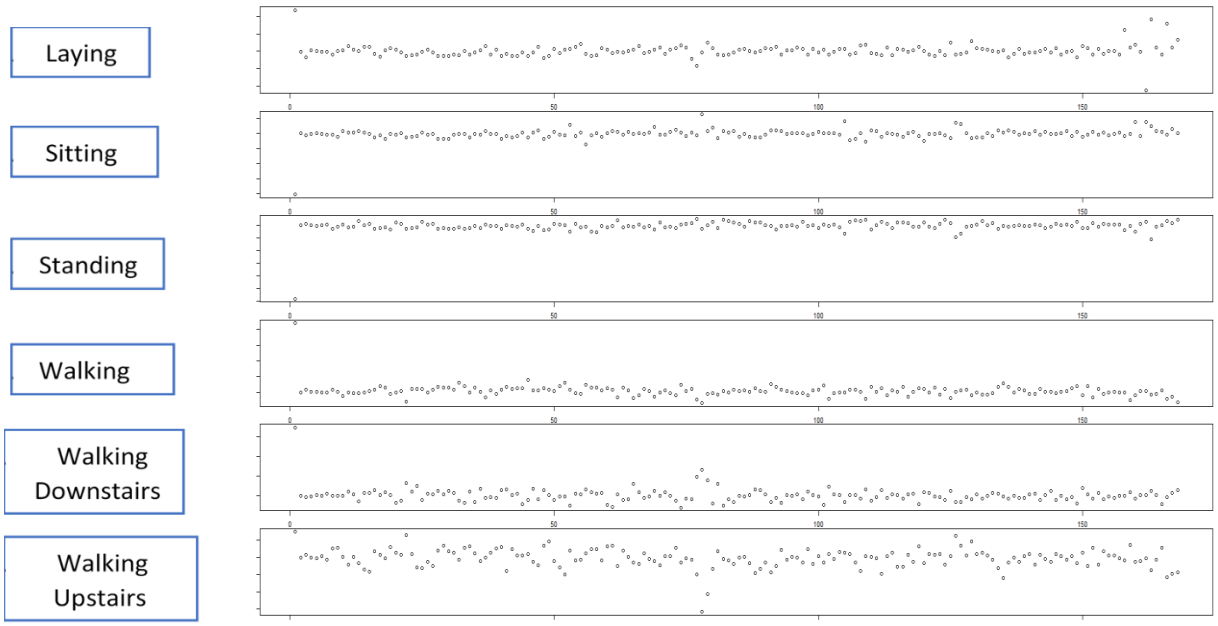


Figure 9. Magnitude of obtained Coefficients from Ridge Regression vs Predictors

Models:

The models used on the data are as follows:

No.	Method	PCA (uncor)	Uncor	PCA (total)	Total
1	MLR	■	■		
2	LDA	■	■		
3	QDA	■	■		
4	KNN		■	■	■
5	Hierarchical Clustering				■
6	K-means Clustering				■
7	Bagging/ Random Forest				■
8	SVM Radial	■	■		
9	SVM Linear	■	■		
10	Neural Networks				■

PCA (uncor): Principle Components that explain 80% of the variance, data with less than 80% correlation (52)
Uncor: Data with less than 80% correlation (166 Predictors)
PCA (total): Principle Components that explains 80% of the variance, complete dataset (27)
Total: Complete dataset (561 Predictors)

In case of bagging & random forest and SVM, we use 10-fold cross validation and 3 iterations to identify the test accuracy of the model. The model having the best train accuracy or the lowest training error rate is chosen for predicting values from the validation set (test set kept aside). These predicted values are then compared with the original observed values to generate a confusion matrix and obtain a test accuracy, which can be later compared with outputs of other models to evaluate performance.

The implementation for each class is given in respective sections.

Multinomial Logistic Regression

In the dataset provided, we have 6 classes in the response variable. For fitting a classification model to this data, we can use multinomial logistic regression (mlogit), which is an extension of the binary classifier. For an mlogit model, we use the multinom() function from the 'nnet' library.

Discriminant Analysis (LDA and QDA)

We perform Linear and Quadratic Discriminant Analysis on the uncorrelated dataset with and without the Principal Components data and plot the confusion matrices for both.

KNN Classification

k Nearest Neighbors is an unparameterized supervised learning algorithm capable of classification and regression. The output depends on whether *k*-NN is used for classification or regression:

- In *k-NN classification*, the output is predicting the class. A vector in a p dimensional space is classified by the calculating the number of vectors belonging to each class in its k nearest neighbors. If *k* = 1, then the vector is classified as the class of its nearest neighbor.
- In *k-NN regression*, the output is a value. This value is the average of the values of its k nearest neighbors. The training examples are vectors in a multidimensional feature space, each with a class label. During training, each of the vectors are observed in the multidimensional phase along with its class label.

During classification, k is a given by the user, and the nearest k neighbors are considered to predict the class of the each given vector. Euclidean distance can be used as a measure to find the nearest neighbors. The bias-variance tradeoff is achieved changing the value of k. When the value of k is small, a small change in training data an affect the classifier, hence the classifier tends to have high variance with a low bias. Similarly, if the value of K is large, the classifier remains tolerant to change in data, hence the classifier tends to have high bias and low variance.

The optimum value of k-NN (k Nearest Neighbors) can be chosen by using k-fold Cross Validation, where the training observation is divided into k groups, k-1 groups used for training and accuracy is calculated by testing the classifier on the k^{th} group. This approach by testing on all possible k groups for a given k, and the accuracy is the average of all the accuracies. This algorithm is iterated over the required range of k-NN, and the best k is chosen with best CV accuracy. This approach is used with our current dataset to predict the activity of the subjects. Three different set of predictors are used with this model: 1. Entire predictor set, 2. Predictors with less than 0.8 correlation, 3. Principle Components that explains 80% of the variance. To prevent getting an overfitted model from k-fold CV, we divide the training subjects into k groups, instead of dividing the entire observations into k groups. This enables calculating test accuracy on subjects which has not been used to train the model. The following are the result of the approach discussed above:

Hierarchical Clustering

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

- The bottom up approach. Each observation is a cluster of its own, and clusters are formed as we go up.
- The Top down approach. All observations start as one cluster and split as we go down.

A greedy approach is used to split/merge the clusters. The clusters are represented in a dendogram to explain the clusters. Distance is the vertical axis in dendogram while the observations are the horizontal axis.
Some commonly used methods to calculate distance for hierarchical clustering are:

- Euclidean Distance: $\sqrt{\sum(a_i - b_i)^2}$
- Squared Euclidean Distance: $\sum(a_i - b_i)^2$
- Manhattan Distance: $\sum|a_i - b_i|$
- Maximum Distance: $\max_i|a_i - b_i|$

The linkage criterion determines the distance between sets of observations as a function of the pairwise distances between observations. Some commonly used linkage criteria between two sets of observations A and B are:

- Complete Linkage: $\max\{d(a,b): a \in A, b \in B\}$
- Single Linkage: $\min\{d(a,b): a \in A, b \in B\}$
- Average Linkage: $\frac{1}{|A||B|}\sum\sum d(a,b)$

- Centroid Linkage: $\|c_s - c_t\|$, where c_s and c_t are centroid of clusters s and t

k-Means Clustering:

k-means clustering is a way to classify n observations into k clusters such that each observation belongs to the cluster with the nearest mean. The algorithm uses an iterative technique, to find the clusters. Given an initial set of k, the algorithm proceeds by alternating between two steps:

- **Assignment step**: Assign each observation to the cluster whose mean has the least squared Euclidean distance, this is intuitively the "nearest" mean.

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \forall j, 1 \leq j \leq k \right\}$$

Where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

- **Update step**: Calculate the new means to be the centroids of the observations in the new clusters.

$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j$$

The algorithm has converged when the assignments no longer change. There is no guarantee that the optimum is found using this algorithm. K-Means has been used to cluster the data of Subject 1 into 4 different clusters. Since, the dimensions are very high, it cannot be visualized. Using, principal component analysis, the dimensions of the data can be reduced to 2, which can be visualized using scatterplot. This can be compared to true distribution of activities of Subject 1 in the principle component space.

Bagging ensembles and random forest classification

We apply multiple loops of bagging iterations varying the selection of mtry (number of variables randomly sampled as candidates at each split) and number of trees to identify the best value of the tuning parameters. The different models are compared using the test accuracy as a performance measure.

Step 1: Random search

We have used a random search method to randomly pick the mtry value from the pool and perform a bagging training on the model. We used 10 fold CV and 3 iterations to evaluate model performance using accuracy as a parameter.

Step 2: Grid search

We now performed a gridsearch for mtry =10 to mtry = 20 to further tune our choice of mtry. The same approach of 10 fold CV and 3 iterations was used to evaluate performance.

Step 3: Holdout validation

We now use the 2 models to make predictions on the test data (holdout set).

SVM ensemble using radial kernel

1. Using PCA on uncorrelated dataset
2. On entire uncorrelated dataset

SVM using linear kernel

Support vector machines are classifiers that use a set of support points to define the boundaries. In case of linear SVM, the boundaries are linear.

1. Using PCA on uncorrelated dataset: Here we use PCA to select principle components and then predict the classes.
2. On entire uncorrelated dataset: Here we predict the classes for the entire uncorrelated dataset. A grid of different 'Costs' is used to find out the best 'Cost'. Cost is basically a penalty put on the SVM for tuning. Higher the cost parameter, better the classification but lesser is the smoothness of the boundary.

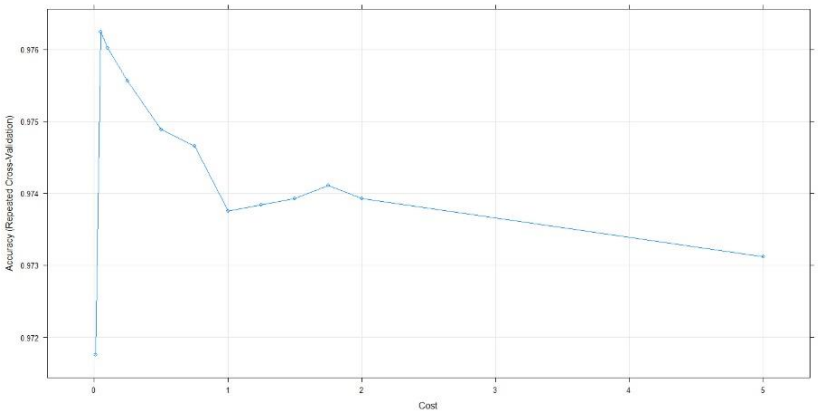


Figure 10. Accuracy Vs Cost using SVM Linear Kernel

Quadratic SVM:

1. On entire Uncorrelated Data: We performed quadratic SVM using the classyfire package in R which has inbuilt functions for creation and application of highly optimized and robustly evaluated ensembles of support vector machines (SVMs). We performed bootstrap resampling of the data to identify the best training model selected on based of training accuracy obtained by the bootstrap iterations. We performed 10 bootstrap iterations running parallel on 4 cpus which enabled to reduce our computation times which are relatively high for this type of a classification method. The cfBuild function helped us to create a highly optimized ensemble of radial basis function (RBF) support vector machines (SVMs). The cfBuild function takes care of all aspects of the SVM optimization, internally splitting the supplied data into separate training and testing subsets using a bootstrapping approach coupled with a heuristic optimization algorithm and parallel processing to minimize computation time. The ensemble object can then be used to classify newly acquired data using the cfPredict function (source: ??cfBuild). Highest test accuracy obtained on training data was around 98%. The model was then used to predict values from the holdout set which returned an accuracy of ~94%. This result is very good and very close to the output of the random forest algorithm.

The results for multiple bootstrap iterations have been illustrated below.

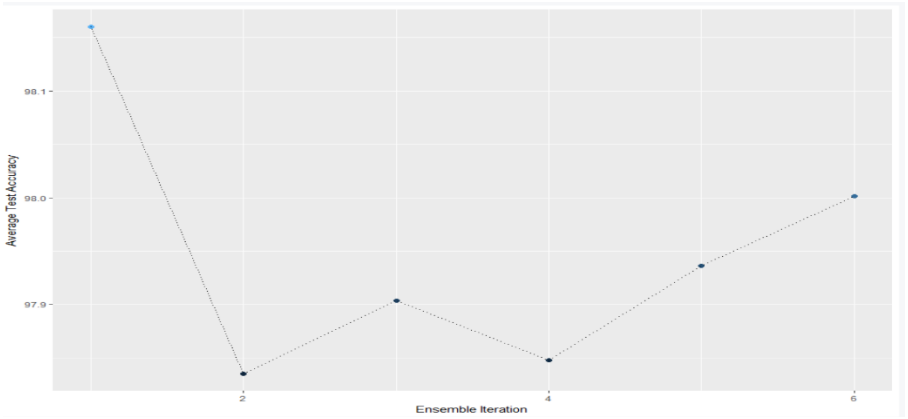


Figure 11. QVSM train accuracy obtained from 6 bootstraps

2. QSVM after PCA dimension reduction: We also performed SVM with radial kernel on the dataset obtained after using PCA for dimension reduction. Using PCA we rotated and scaled the input data matrix to obtain a new data matrix with modified score and loadings of top 52 principle components. A model was trained on these components and was used for prediction on the holdout set. The holdout set was rotated and scaled with respect to the top 52 components to obtain a test PC matrix. The trained model was then applied to this matrix to evaluate model performance. We obtained a test classification accuracy of around 90% for the holdout set. This is imperative as we compromised to reduce the model complexity by losing 20% of the variance.

Summary:

Multinomial Logistic Regression

1. Using PCA on uncorrelated data:

```
pred.mlr   0   1   2   3   4   5
        0 500   3   1   0   0   2
        1  31 403  58   0   0   2
        2   3  81 472   0   0   0
        3   2   0   0 486  38  58
        4   1   0   1   5 344  48
        5   0   4   0   5  38 361

pred.mlr     0     1     2     3     4     5
        0 93.11  0.61  0.19  0.00  0.00  0.42
        1  5.77 82.08 10.90  0.00  0.00  0.42
        2  0.56 16.50 88.72  0.00  0.00  0.00
        3  0.37  0.00  0.00 97.98  9.05 12.31
        4  0.19  0.00  0.19  1.01 81.90 10.19
        5  0.00  0.81  0.00  1.01  9.05 76.65
[1] 87.0716
```

2. Using correlation for feature selection

```
pred.mlr2   0   1   2   3   4   5
        0 532   0   0   2   0   1
        1   0 407  28   1   1   3
        2   2  78 503   0   2   5
        3   0   2   0 460  13  29
        4   3   0   1  11 370  10
        5   0   4   0  22  34 423

pred.mlr2     0     1     2     3     4     5
        0 99.07  0.00  0.00  0.40  0.00  0.21
        1  0.00 82.89  5.26  0.20  0.24  0.64
        2  0.37 15.89 94.55  0.00  0.48  1.06
        3  0.00  0.41  0.00 92.74  3.10  6.16
        4  0.56  0.00  0.19  2.22 88.10  2.12
        5  0.00  0.81  0.00  4.44  8.10 89.81
[1] 91.44893
```

Discriminant Analysis (LDA and QDA)

1. LDA using PCA on uncorrelated data

```
    0   1   2   3   4   5
0 490   1   0   0   0   0
1  39 395  59   0   0   0
2   7  92 473   0   0   0
3   1   1   0 476  36  56
4   0   0   0  12 343  23
5   0   2   0   8  41 392

      0     1     2     3     4     5
0 91.25  0.20  0.00  0.00  0.00  0.00
1  7.26 80.45 11.09  0.00  0.00  0.00
2  1.30 18.74 88.91  0.00  0.00  0.00
3  0.19  0.20  0.00 95.97  8.57 11.89
4  0.00  0.00  0.00  2.42 81.67  4.88
5  0.00  0.41  0.00  1.61  9.76 83.23
[1] 87.1734
```

2. LDA using correlation for feature selection

```
    0   1   2   3   4   5
0 537   0   0   0   0   0
1   0 426  49   0   0   0
2   0  64 483   0   0   0
3   0   0   0 485  14  29
4   0   0   0   2 374   1
5   0   1   0   9  32 441

       0      1      2      3      4      5
0 100.00   0.00   0.00   0.00   0.00   0.00
1   0.00  86.76   9.21   0.00   0.00   0.00
2   0.00  13.03  90.79   0.00   0.00   0.00
3   0.00   0.00   0.00  97.78   3.33   6.16
4   0.00   0.00   0.00   0.40  89.05   0.21
5   0.00   0.20   0.00   1.81   7.62  93.63
[1] 93.1795
```

Using the uncorrelated dataset, we build a more accurate model. From the confusion matrix below we can see that the accuracy has increased to 93.18%.

3. QDA using PCA on uncorrelated data

Using qda() function, we obtain an accuracy of 87.61%, which is a marginal improvement over both the linear models.

```
     0   1   2   3   4   5
0  506   9   4   0   0   0
1   31 359  39   0   0   0
2    0 120 487   0   0   0
3    0   0   1 480  12  48
4    0   0   1  15 382  55
5    0   3   0   1  26 368

       0     1     2     3     4     5
0  94.23  1.83  0.75  0.00  0.00  0.00
1   5.77 73.12  7.33  0.00  0.00  0.00
2   0.00 24.44 91.54  0.00  0.00  0.00
3   0.00  0.00  0.19 96.77  2.86 10.19
4   0.00  0.00  0.19  3.02 90.95 11.68
5   0.00  0.61  0.00  0.20  6.19 78.13
[1] 87.61452
```

4. QDA on entire uncorrelated dataset

Using the uncorrelated predictors for our QDA model we get an improved accuracy of 90.81%, which is lower than the accuracy figures obtained using the linear models.

```
     0   1   2   3   4   5
0  537   9   9   0   0   0
1    0 375  32   0   0   0
2    0 106 486   0   0   0
3    0   0   0 412   0   4
4    0   1   3  79 410  11
5    0   0   2   5  10 456

       0      1      2      3      4      5
0 100.00   1.83   1.69   0.00   0.00   0.00
1   0.00  76.37   6.02   0.00   0.00   0.00
2   0.00  21.59  91.35   0.00   0.00   0.00
3   0.00   0.00   0.00  83.06   0.00   0.85
4   0.00   0.20   0.56  15.93  97.62   2.34
5   0.00   0.00   0.38   1.01   2.38  96.82
[1] 90.80421
```

KNN Classification

The following are the result of the approach discussed above:

1. Entire Predictor Set

The accuracy of this model was found to be 92.02%

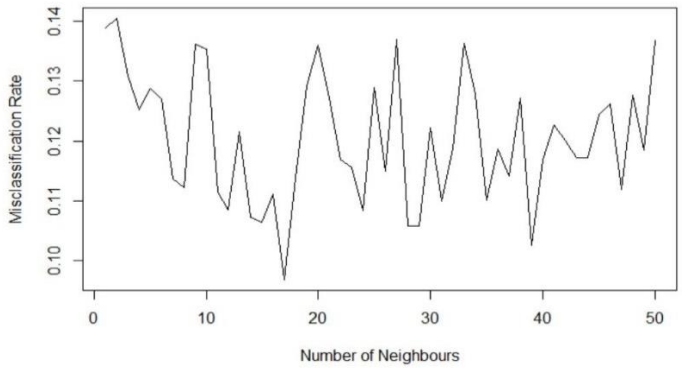


Figure 12: Misclassification rate plotted against k nearest neighbors obtained through k fold CV

	test.y					
knn.testpred	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
LAYING	654	2	0	0	0	0
SITTING	4	542	105	0	0	0
STANDING	0	61	526	0	0	0
WALKING	0	0	0	508	15	11
WALKING_DOWNSTAIRS	0	0	0	6	394	9
WALKING_UPSTAIRS	0	0	0	11	44	468

Figure 13: Confusion Matrix yielding an accuracy of 92.02% after choosing k= 17

2. Predictors with less than 0.8 correlation

The accuracy of this model was found to be 88.39%

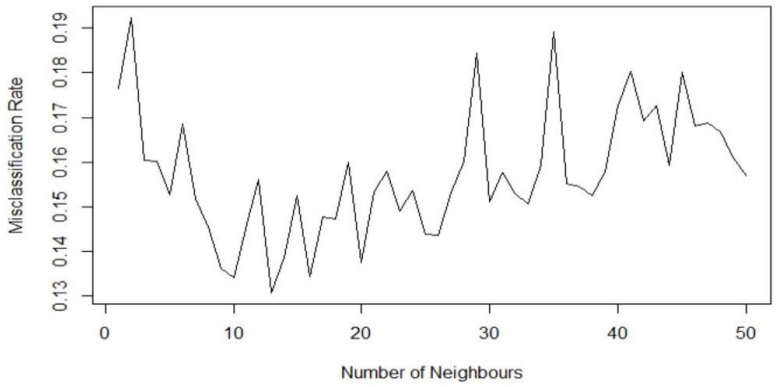


Figure 14: Misclassification rate plotted against k nearest neighbors obtained through k fold CV

	test.y					
knn.testpred	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
LAYING	550	9	0	0	0	0
SITTING	18	462	114	0	0	0
STANDING	16	71	454	0	0	0
WALKING	1	0	0	462	72	12
WALKING_DOWNSTAIRS	0	0	0	4	315	6
WALKING_UPSTAIRS	1	1	0	13	30	428

Figure 15: Confusion Matrix yielding an accuracy of 88.39% after choosing k= 12

3. Principle Components that explains 80% of the variance on the complete dataset

The accuracy of this model was found to be 88.39%

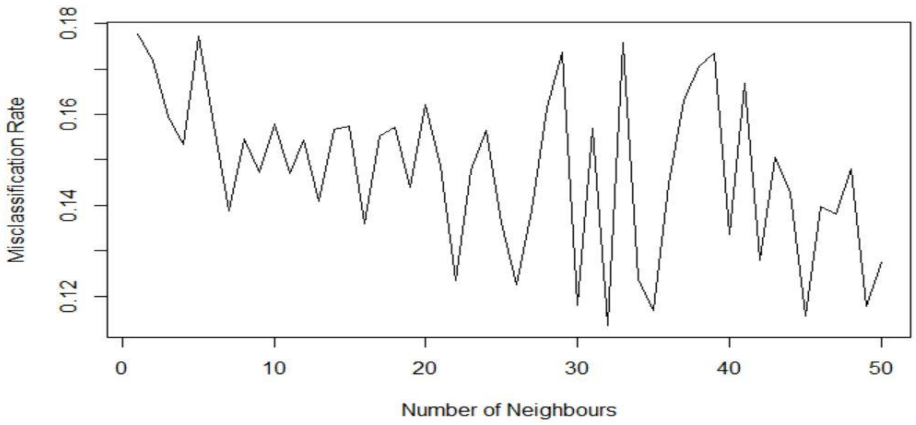


Figure 16: Misclassification rate plotted against k nearest neighbors obtained through k fold CV

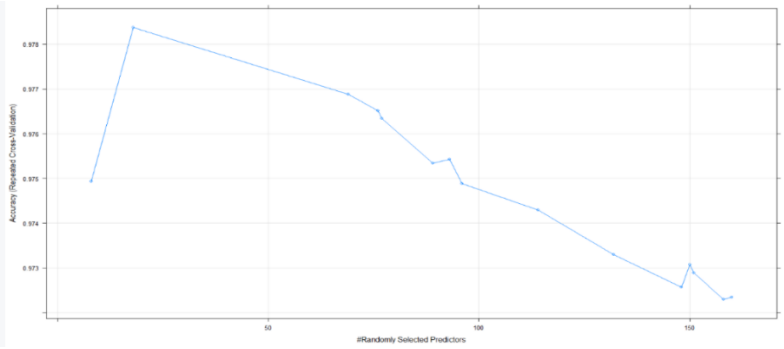
knn.testpred	1	2	3	4	5	6
1	600	4	0	0	0	0
2	21	495	75	0	0	0
3	37	106	556	0	0	0
4	0	0	0	503	40	19
5	0	0	0	3	343	22
6	0	0	0	19	70	447

Figure 17: Confusion Matrix yielding an accuracy of 87.62% after choosing k= 32

Bagging ensembles and random forest classification

Step 1: Random search

The train accuracy for these models with different mtry are plotted in the graph below:



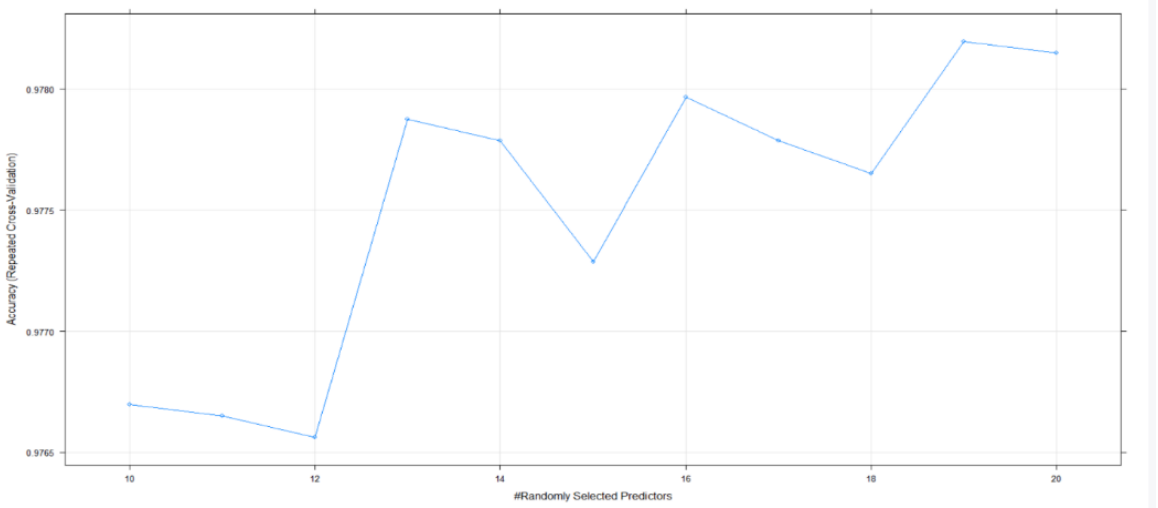
The results have also been tabulated as below:

```
Random Forest

7352 samples
 167 predictor
   6 classes: 'LAYING', 'SITTING', 'STANDING', 'WALKING', 'WALKING_DOWNSTAIRS', 'WALKING_UPSTAIRS'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 6619, 6618, 6619, 6616, 6614, 6617, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
    8   0.9749308  0.9698242
   18   0.9783758  0.9739712
   69   0.9768799  0.9721703
   76   0.9765182  0.9717342
   77   0.9763364  0.9715157
   89   0.9753392  0.9703151
   93   0.9754295  0.9704242
   96   0.9748863  0.9697704
  114   0.9742969  0.9690612
  132   0.9732981  0.9678586
  148   0.9725717  0.9669839
  150   0.9730709  0.9675854
  151   0.9728886  0.9673656
  158   0.9722994  0.9666565
  160   0.9723456  0.9667123

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 18.
```

It was observed that the model with mtry = 18 in the grid had the best performance. This is very close to the mtry used for a randomForest model, i.e. sqrt(no. of predictors) ~ 12 in our case (floor(sqrt(168))).

Step 2: Grid search

We now performed a gridsearch for mtry =10 to mtry = 20 to further tune our choice of mtry. The same approach of 10 fold CV and 3 iterations was used to evaluate performance. The following graph and tabulated accuracy values were obtained for the train data:



```
Random Forest

7352 samples
 167 predictor
   6 classes: 'LAYING', 'SITTING', 'STANDING', 'WALKING', 'WALKING_DOWNSTAIRS', 'WALKING_UPSTAIRS'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 6619, 6618, 6619, 6616, 6614, 6617, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  10    0.9766979  0.9719516
  11    0.9766519  0.9718959
  12    0.9765624  0.9717886
  13    0.9778769  0.9733711
  14    0.9777870  0.9732627
  15    0.9772881  0.9726621
  16    0.9779671  0.9734797
  17    0.9777873  0.9732632
  18    0.9776519  0.9731003
  19    0.9781964  0.9737554
  20    0.9781504  0.9737003

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 19.
```

The model with mtry = 19 provided the best train accuracy of 97.82% which is marginally better than the accuracy for randomForest model with mtry = 12 which provides an accuracy of 97.66%.

Step 3 : Holdout validation

We now use the 2 models to make predictions on the test data (holdout set). Confusion matrix is tabulated in absolute values and percentages below:

i. Bagging, mtry = 19 ntree=1000

```
7352 samples
 167 predictor
   6 classes: 'LAYING', 'SITTING', 'STANDING', 'WALKING', 'WALKING_DOWNSTAIRS', 'WALKING_UPSTAIRS'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 7352, 7352, 7352, 7352, 7352, 7352, ...
Resampling results:

  Accuracy  Kappa
  0.976075  0.9711995

Tuning parameter 'mtry' was held constant at a value of 19

test.rf4             LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS
  LAYING             100.00    0.00     0.00    0.00              0.00             0.00
  SITTING              0.00   87.58    11.28    0.00              0.00             0.00
  STANDING             0.00   12.22    88.72    0.00              0.00             0.00
  WALKING              0.00    0.00     0.00   99.40              0.24             6.79
  WALKING_DOWNSTAIRS   0.00    0.00     0.00    0.40             91.19             2.97
  WALKING_UPSTAIRS     0.00    0.20     0.00    0.20              8.57            90.23
[1] 92.97591
```

The bagging model with mtry = 19 returns an overall accuracy of 92.98%

ii. Random Forest, mtry = 12 ntree =1000

Confusion matrix is tabulated in absolute values and percentages below:

```
test.rf              LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS
  LAYING                537       1        0       0                 0               0
  SITTING                 0     437       47       0                 0               0
  STANDING                0      52      485       0                 0               0
  WALKING                 0       0        0     494                 3              30
  WALKING_DOWNSTAIRS      0       0        0       1               382              12
  WALKING_UPSTAIRS        0       1        0       1                35             429

test.rf              LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS
  LAYING             100.00    0.20     0.00    0.00              0.00             0.00
  SITTING              0.00   89.00     8.83    0.00              0.00             0.00
  STANDING             0.00   10.59    91.17    0.00              0.00             0.00
  WALKING              0.00    0.00     0.00   99.60              0.71             6.37
  WALKING_DOWNSTAIRS   0.00    0.00     0.00    0.20             90.95             2.55
  WALKING_UPSTAIRS     0.00    0.20     0.00    0.20              8.33            91.08
```

The random forest model returns an overall accuracy of 93.82 % on the validation set which is a very good result.

SVM ensemble using radial kernel

1. Using PCA
2. On entire dataset

SVM using linear kernel

1. Using PCA: Here we use PCA to select principle components and then predict the classes.

```
#Confusion matrix in percentages

predsvm LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS
      1  93.11    0.41     0.00    0.00              0.00            0.00
      2   6.70   84.11    12.41    0.00              0.00            0.42
      3   0.19   14.66    87.41    0.00              0.00            0.00
      4   0.00    0.00     0.00   98.19              6.67           16.14
      5   0.00    0.00     0.00    0.81             85.24           10.19
      6   0.00    0.81     0.19    1.01              8.10           73.25

[1] 87.13946
```

```
#Confusion matrix

predsvm LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS
      1    500       2        0       0                 0               0
      2     36     413       66       0                 0               2
      3      1      72      465       0                 0               0
      4      0       0        0     487                28              76
      5      0       0        0       4               358              48
      6      0       4        1       5                34             345
```

2. On entire uncorrelated dataset:

```
Confusion Matrix and Statistics

                    Reference
Prediction           LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS WALKING_UPSTAIRS
  LAYING                512       0        0       0                 0               0
  SITTING                 0     427       37       0                 0               0
  STANDING               25      61      495       0                 0               0
  WALKING                 0       0        0     490                11              38
  WALKING_DOWNSTAIRS      0       0        0       2               386              28
  WALKING_UPSTAIRS        0       3        0       4                23             405

Overall Statistics

               Accuracy : 0.9213
                 95% CI : (0.911, 0.9307)
    No Information Rate : 0.1822
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9054
 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: LAYING Class: SITTING Class: STANDING Class: WALKING Class: WALKING_DOWNSTAIRS
Sensitivity                 0.9534         0.8697          0.9305         0.9879                      0.9190
Specificity                 1.0000         0.9849          0.9644         0.9800                      0.9881
Pos Pred Value              1.0000         0.9203          0.8520         0.9091                      0.9279
Neg Pred Value              0.9897         0.9742          0.9844         0.9975                      0.9866
Prevalence                  0.1822         0.1666          0.1805         0.1683                      0.1425
Detection Rate              0.1737         0.1449          0.1680         0.1663                      0.1310
Detection Prevalence        0.1737         0.1574          0.1971         0.1829                      0.1412
Balanced Accuracy           0.9767         0.9273          0.9474         0.9840                      0.9536
                     Class: WALKING_UPSTAIRS
Sensitivity                           0.8599
Specificity                           0.9879
Pos Pred Value                        0.9310
Neg Pred Value                        0.9737
Prevalence                            0.1598
Detection Rate                        0.1374
Detection Prevalence                  0.1476
Balanced Accuracy                     0.9239
```

SVM Using Radial Kernel

1. Using PCA:

```
> qsvm.cfpc #Confusion matrix in percentages

        0     1     2     3     4     5
  0 93.67  0.41  0.38  0.00  0.00  0.00
  1  4.47 88.59 11.65  0.00  0.00  1.70
  2  1.12 11.00 87.97  0.20  0.00  0.21
  3  0.56  0.00  0.00 98.39  5.00 10.62
  4  0.00  0.00  0.00  0.81 88.81  7.22
  5  0.19  0.00  0.00  0.60  6.19 80.25
> acc3=k/m*100
> print(acc3)
[1] 89.75229
>
```

2. On entire uncorrelated dataset:

```
          Test_Class
Predicted   0   1   2   3   4   5
        0 100   0   0   0   0   0
        1   0  95   5   0   0   0
        2   0   4  96   0   0   0
        3   0   0   0 100   0   0
        4   0   0   0   0  99   1
        5   0   0   0   0   0 100
     Opt Gamma  Opt Cost
[1,] -8.154986  2.333508
[2,] -8.684171  4.962969
[3,] -8.155800  9.850139
[4,] -8.819178  4.480813
[5,] -8.139534  3.394100
[6,] -9.037524 11.408474

        0     1     2     3     4     5
  0 95.16  0.41  0.00  0.00  0.00  0.00
  1  0.74 88.59  4.70  0.00  0.00  0.00
  2  4.10 10.59 95.30  0.00  0.00  0.00
  3  0.00  0.00  0.00 99.40  2.38  6.37
  4  0.00  0.00  0.00  0.60 94.76  4.67
  5  0.00  0.41  0.00  0.00  2.86 88.96
[1] 93.75636
```

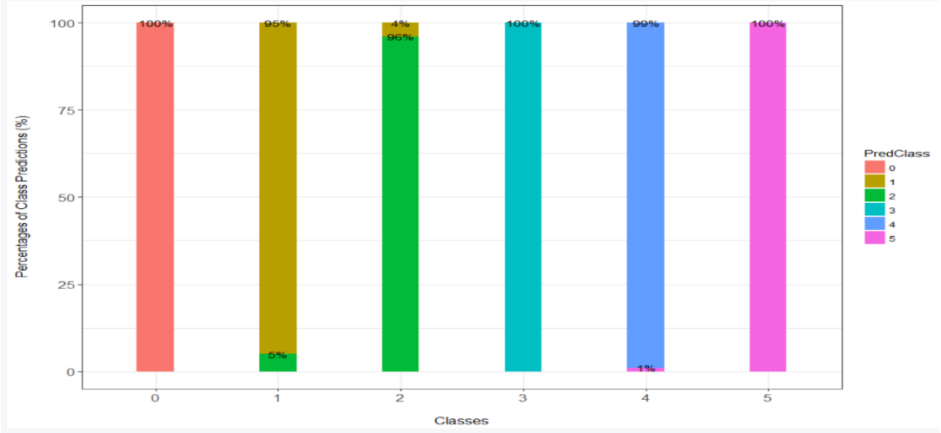


Figure 18. QVSM bootstrap classification results

Model Comparison:

Models:

Models implemented on the Entire dataset (561 predictors)

Sr. No.	Method	Accuracy
1	K-Nearest Neighbors	92.02
2	Bagging & Random Forest,19	92.98
3	Bagging & Random Forest,12	93.82

Models implemented on the Uncorrelated dataset (166 predictors)

Sr. No.	Method	Accuracy
1	Multinomial Logistic Regression	91.45
2	Linear Discriminant Analysis	93.18
3	Quadratic Discriminant Analysis	90.80
4	K-Nearest Neighbors	88.39
5	SVM Radial Kernel	93.76
6	SVM Linear Kernel	92.13

Models implemented on the feature reduced data after PCA

Sr. No.	Method	Accuracy
1	Multinomial Logistic Regression	87.07
2	Linear Discriminant Analysis	87.17
3	Quadratic Discriminant Analysis	87.62
4	K-Nearest Neighbors	87.62
5	SVM Radial Kernel	89.75
6	SVM Linear Kernel	87.14

Inference:

It is evident from the results that the models trained with the uncorrelated predictors to classify the given data, outperforms the models trained with the Principal Components. This can be accounted to the fact that we had a threshold of 80% (Variance Explained by the Principle Components), to reduce the number of dimensions. Accuracy can be improved, by increasing this threshold, leading to increase in the number of Principal Components and thereby retaining more information from the actual set of predictors, but at the cost of increase in the complexity of the model. The uncorrelated predictors give higher accuracy but it is fairly complex, since this uses 168 dimensions to explain the data.

Conclusions:

The best accuracy among all the 18 implemented models for the classification model with the hold out set, was 93.82% from Random Forests using 12 predictors. As an ensemble outperforms models like LDA, QDA, Logistic regression we can conclude that the actual classification predictive model might not be of a polynomial form. Due to 20% loss of variance in PCA, the original Uncorrelated data fits all the models well and gives better accuracy. But it is a trade off between computational complexity and accuracy. By the loss of 4% in accuracy we were able to build a significantly efficient model with just 30% of the data.

Future Scope:

We have tried to implement Neural Network for HAR dataset. We have achieved an accuracy of 80% with basic feedforward neural network. We can further apply other deep learning algorithms like RNN, CNN which perform well to handle the non-linearities in the data. We will also try to analyze the dataset with the additional approaches as mentioned above.

References:

1. http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises
2. https://rpubs.com/mlavanya92/ucihar1
3. https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition
4. Method for CV and comparing the results
   https://machinelearningmastery.com/compare-models-and-select-the-best-using-the-caret-r-package/
5. SVM
   http://dataaspirant.com/2017/01/19/support-vector-machine-classifier-implementation-r-caret-package/
6. Quadratic SVM
   https://github.com/topepo/caret/blob/master/RegressionTests/Code/treebag.R
7. Different training approaches
   http://topepo.github.io/caret/train-models-by-tag.html
8. Random Forest loops
   https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/
9. Predictor Selection:
   http://rstatistics.net/variable-importance-of-predictors-that-contribute-most-significantly-to-a-response-variable-in-r/
10. Feature Selection:
    https://machinelearningmastery.com/feature-selection-with-the-caret-r-package/
11. http://dataaspirant.com/2018/01/15/feature-selection-techniques-r/
12. Genetic search algorithm for feature selection
    http://topepo.github.io/caret/feature-selection-using-genetic-algorithms.html
13. ISLR
14. https://github.io/

Appendix:

Codes:

```
#EDA: Hierarchical, K-means, Frequency Distribution
library(ggplot2)
library(dendextend)
library(plyr)
library(reshape2)
library(lattice)
library(caret)
library(class)
library(MASS)
library(psych)
library(glmnet)
library(mlbench)
library(randomForest)

nsampleSize = -1L
train<-read.csv("hartrain.csv",header= TRUE, sep=",")
test<-read.csv("hartest.csv",header= TRUE, sep=",")
train <- transform(train, subjectID = factor(subject), activityID =
factor(activity))
test <- transform(test, subjectID = factor(subject), activityID =
factor(activity))
train$partition = "train"
test$partition = "test"
data <- rbind(train,test)
par(mfrow=c(1,2))
qplot(data= data,x= subjectID,fill= activityID) + theme(plot.margin=
unit(c(1,1,1,1),"cm"))
qplot(data = data, x = subjectID, fill = partition) +
theme(plot.margin= unit(c(1,1,1,1),"cm"))
par(mfrow=c(1,1))
sub1 <- subset(data, subjectID == 1)
distanceMatrix <- dist(sub1[, -c(562:564)])
hclustering <- hclust(distanceMatrix,, method='complete')
dend <- as.dendrogram(hclustering)
dend <- color_branches(dend, k = 6)
plot(dend)
numPredictors = ncol(data) - 5
dataSd = colwise(sd)(data[, 1:numPredictors])
dataSd$stat = "Predictor Variable Standard Deviation"
dataMean = colwise(mean)(data[, 1:numPredictors])
dataMean$stat = "Predictor Variable Mean"
temp = melt(rbind(dataMean, dataSd), c("stat"))
qplot(data = temp, x = value, binwidth = 0.025) + facet_wrap(~stat,
ncol = 1)
numPredictors= ncol(train)-5
zScaleTrain = preProcess(train[, 1:numPredictors])
scaledX = predict(zScaleTrain, train[, 1:numPredictors])
correlatedPredictors = findCorrelation(cor(scaledX), cutoff = 0.8)
uncorData= scaledX[, -correlatedPredictors]
pcaTrain = preProcess(uncorData, method = "pca", thresh = 0.8)
pcaTrain
names(sub1)
n= ncol(sub1)-5
pr.out=prcomp(sub1[1:n], scale=TRUE)
pr.out$rotation

set.seed(45)
sub1Cluster <- kmeans(c(pr.out$x[,1],pr.out$x[,2]), 6, nstart = 5)
sub1clus <- sub1Cluster$cluster
par(mfrow=c(1,1))
plot(pr.out$x[,1],pr.out$x[,2], col= sub1clus,xlab='PC1',ylab='PC2')
plot(pr.out$x[,1],pr.out$x[,2], col=
data$activityID,,xlab='PC1',ylab='PC2')
legend("topleft",legend=
unique(sub1$activityID),col=unique(sub1$activityID),cex=0.6,pch=1)

#PCA
pr.out=prcomp(scaledXTrain, scale=TRUE)
names(pr.out)
pr.out$center
pr.out$scale
pr.out$rotation
biplot(pr.out,scale=0)
pr.out$sdev
pr.var=pr.out$sdev^2
pr.var
pve=pr.var/sum(pr.var)
sum=0
pve
#cumulative sum of variance ratios (PVE)
for (i in 1:n)
{
 if(i==1)
 {sum[1]=pve[1]}
 else
 {sum[i]=sum[i-1]+pve[i]}
}
sum[1:n]
#computing PVE indices for different % of variances
varlim=0.8#80% fraction of total variance to be explained
j=0
for (i in 2:n-1)
{
 if(sum[i]>=varlim && sum[i-1]<varlim)
 {j=i}#index of the PC till which 80% of the variance is explained
 else
 {j=j}
}
j

# MULTINOMIAL LASSO
x <- model.matrix(Activity ~ .,data=train)[, -1]
y=as.factor(train$Activity)
grid=10^seq(10,-2,length=100)
cv.glmnet(x,y,type.measure="class",alpha=1,family="multinomial",
lambda=grid, type.multinomial="grouped")
coef=coef(cvfit, s = "lambda.min")
coef
par(mar=c(1,1,1,1))
par(mfrow=c(6,1))
plot(coef$'0', xlabel="Laying", ylabel= "Predictors")
plot(coef$'1', xlabel="Sitting,, ylabel="Predictors")
plot(coef$'2', xlabel= "=Standing",  ylabel= "Predictors")
plot(coef$'3', xlabel= "Walking",  ylabel= "Predictors")
```

```
plot(coef$'4', xlabel="Walking_Downstairs", ylabel= "Predictors")
plot(coef$'5', xlabel=" Walking_Upstairs", ylabel="Predictors")
# use alpha=0 for ridge regression

#KNN
data= rbind(train,test)
compute.misclass<-function(conf){
  x= ncol(conf)
  y= nrow(conf)
  mis= 0
  clas= 0
  for (i in (1:x)){
    for (j in (1:y)){
      if(i==j){
        clas= clas+conf[i,j]
      }
      else{
        mis= mis+conf[i,j]
      }
    }
  }
  mis/(mis+clas)
}
set.seed(1)
tr <- data.frame(matrix(ncol = length(names(data)), nrow = 0))
colnames(tr) <- names(data)
te<- data.frame(matrix(ncol = length(names(data)), nrow = 0))
colnames(te) <- names(data)
kerr <- data.frame(matrix(ncol = 1, nrow = 0))
kfold= 5
totsize=21
te.size= totsize/kfold
for (k in (1:50)){
  mis=0
  for (i in (1:kfold)){
    tr <- NULL
    te <- NULL
    te.idx= sample((1:totsize),te.size, replace=F)
    for (j in (1: totsize)){
      if (j %in% te.idx){
        te <- rbind(te,data[(data$subject== j),])
      }
      else{
        tr <- rbind(tr,data[(data$subject== j),])
      }
    }
    tr.x= tr[,-c(562:563)]
    tr.y= tr[,563]
    te.x= te[,-c(562:563)]
    te.y= te[,563]
    knn.testpred=knn(tr.x,te.x,tr.y,k=k)
    conf= table(knn.testpred,te.y)
    mis= mis + compute.misclass(conf)
  }
  mis=mis/kfold
  kerr <- rbind(kerr,mis)
}
plot(c(1:50),kerr[,1],type='l',xlab='Number of
Neighbours',ylab='Misclassification Rate')
traindata <- data.frame(matrix(ncol = length(names(data)), nrow =
0))
testdata <- data.frame(matrix(ncol = length(names(data)), nrow = 0))
for (i in 1:nrow(data)){
 if(data$subject[i]<22){
  traindata= rbind(traindata,data[i,])
 }
 else{
  testdata= rbind(testdata,data[i,])
 }
}
train.x= traindata[,-c(562,563)]
train.y= traindata[,563]
test.x= testdata[,-c(562,563)]
test.y= testdata[,563]
knn.testpred=knn(train.x,test.x,train.y,k=17)
knn.testconf= table(knn.testpred,test.y)
knn.testconf
accuracy= 1- compute.misclass(knn.testconf)
cat('The Accuracy of this model is: ', accuracy)
cor = findCorrelation(cor(data[,-c(562,563)]), cutoff = 0.8)
uncor.data= data[,-cor]
set.seed(1)
tr <- data.frame(matrix(ncol = length(names(uncor.data)), nrow = 0))
colnames(tr) <- names(uncor.data)
te<- data.frame(matrix(ncol = length(names(uncor.data)), nrow = 0))
colnames(te) <- names(uncor.data)
kerr <- data.frame(matrix(ncol = 1, nrow = 0))
kfold= 5
totsize=21
te.size= totsize/kfold
for (k in (1:50)){
  mis=0
  for (i in (1:kfold)){
    tr <- NULL
    te <- NULL
    te.idx= sample((1:totsize),te.size, replace=F)
    for (j in (1: totsize)){
      if (j %in% te.idx){
        te <- rbind(te,uncor.data[(uncor.data$subject== j),])
      }
      else{
        tr <- rbind(tr,uncor.data[(uncor.data$subject== j),])
      }
    }
    tr.x= tr[,-c(173:174)]
    tr.y= tr[,174]
    te.x= te[,-c(173:174)]
    te.y= te[,174]
    knn.testpred=knn(tr.x,te.x,tr.y,k=k)
    conf= table(knn.testpred,te.y)
    mis= mis + compute.misclass(conf)
  }
  mis=mis/kfold
  kerr <- rbind(kerr,mis)
```

```
}
plot(c(1:50),kerr[,1],type='l',xlab='Number of
Neighbours',ylab='Misclassification Rate')
train.x= train.x[,-cor]
test.x= test.x[-cor]
knn.testpred=knn(train.x,test.x,train.y,k=12)
knn.testconf= table(knn.testpred,test.y)
knn.testconf
accuracy= 1- compute.misclass(knn.testconf)
cat('The Accuracy of this model is: ', accuracy)
```

#LDA after PCA

```
zScaleTrain = preProcess(trainnum[,3:ncol(trainnum)])
scaledX = predict(zScaleTrain, trainnum[, 3:ncol(trainnum)])
pcaTrain = preProcess(scaledX, method = "pca", thresh = 0.8)
pcaTrain #This gives you the number of components required to get
80% variance
pr.out=prcomp(scaledX, scale=TRUE) #Get all the PCs
pr.train = predict(pr.out,trainnum[,3:ncol(trainnum)])[,1:52]
pr.train.mat <- cbind(pr.train, trainnum$Activity)
write.csv(pr.train.mat, file = 'prtrain.csv')
pr.tr=read.csv('prtrain.csv', header = TRUE, sep = "," )
pr.tr=pr.tr[,-1]
fit.lda = lda(X.1~.,data = pr.tr)

#LDA after pca, prediction
tst = testnum[,-1]
pr.test = predict(pr.out,tst)[,1:52]
write.csv(pr.test, file = 'prtest.csv')
pr.test=read.csv('prtest.csv', header = TRUE, sep = "," )
pr.test=pr.test[,-1]
pred.lda = predict(fit.lda,pr.test)

res.lda = table(pred.lda$class,testognum$Activity)
res.lda #Confusion matrix
res.ldapc = res.lda
k=0; m=0
for (i in 1:6) {
 for (j in 1:6)
 {
  res.ldapc[i,j] = round(res.ldapc[i,j]/sum(res.lda[,j])*100,2)
  if (i==j)
  {k=k+res.lda[i,j]}
  m=m+res.lda[i,j]
 }
}
res.ldapc #Confusion matrix in percentages
acclda=k/m*100
print(acclda)
```

#QDA

```
zScaleTrain = preProcess(trainnum[,3:ncol(trainnum)])
scaledX = predict(zScaleTrain, trainnum[, 3:ncol(trainnum)])
pcaTrain = preProcess(scaledX, method = "pca", thresh = 0.8)
pcaTrain #This gives you the number of components required to get
80% variance
pr.out=prcomp(scaledX, scale=TRUE) #Get all the PCs
pr.train = predict(pr.out,trainnum[,3:ncol(trainnum)])[,1:52]
pr.train.mat <- cbind(pr.train, trainnum$Activity)
write.csv(pr.train.mat, file = 'prtrain.csv')
pr.tr=read.csv('prtrain.csv', header = TRUE, sep = "," )
pr.tr=pr.tr[,-1]
fit.qda = qda(X.1~.,data = pr.tr)
#QDA after PCA, prediction
tst = testnum[,-1]
pr.test = predict(pr.out,tst)[,1:52]
write.csv(pr.test, file = 'prtest.csv')
pr.test=read.csv('prtest.csv', header = TRUE, sep = "," )
pr.test=pr.test[,-1]
pred.qda = predict(fit.qda,pr.test)
res.qda = table(pred.qda$class,testognum$Activity)
res.qda #Confusion matrix
res.qdapc = res.qda
k=0; m=0
for (i in 1:6) {
 for (j in 1:6)
 {
  res.qdapc[i,j] = round(res.qdapc[i,j]/sum(res.qda[,j])*100,2)
  if (i==j)
  {k=k+res.qda[i,j]}
  m=m+res.qda[i,j]
 }
}
res.qdapc #Confusion matrix in percentages
accqda=k/m*100
print(accqda)
```

#Multinomial logistic regression

```
library(nnet)
fit.mlr = multinom(X.1~.,data = pr.tr)
pred.mlr = predict(fit.mlr,pr.test)
res.mlr = table(pred.mlr,testognum$Activity)
res.mlr #Confusion matrix
res.mlrpc = res.mlr
k=0; m=0
for (i in 1:6) {
 for (j in 1:6)
 {
  res.mlrpc[i,j] = round(res.mlrpc[i,j]/sum(res.mlr[,j])*100,2)
  if (i==j)
  {k=k+res.mlr[i,j]}
  m=m+res.mlr[i,j]
 }
}
res.mlrpc #Confusion matrix in percentages
accmlr=k/m*100
print(accmlr)
```

#Bagging

```
testog=read.csv('hartest.csv', header = TRUE, sep = "," )
train=read.csv('hartrain.csv', header = TRUE, sep = "," )
test=read.csv('hartestfit.csv', header = TRUE, sep = "," )
bag.har = randomForest(Activity~., train)
test.bag = predict(bag.har, test)
varImpPlot(bag.har) #GiniIndex
res.bag = table(test.bag,testog$Activity)
res.bag #Confusion matrix
```

```
res.bagpc = res.bag
for (i in 1:6) {
 for (j in 1:6)
 {
 res.bagpc[i,j] = round(res.bag[i,j]/sum(res.bag[,j])*100,2)
 }
}
res.bagpc #Confusion matrix in percentages
```

#RandomForest

```
rf.har = randomForest(Activity~., train, mtry=24, ntree=1000)
test.rf = predict(rf.har, test)
varImpPlot(rf.har) #GiniIndex
res.rf = table(test.rf,testog$Activity)
res.rf #Confusion matrix
res.rfpc = res.rf
for (i in 1:6) {
 for (j in 1:6)
 {
  res.rfpc[i,j] = round(res.rf[i,j]/sum(res.rf[,j])*100,2)
 }
}
res.rfpc #Confusion matrix in percentages
```

#Quadratic SVM - CV 10 fold

```
control <- trainControl(method="repeatedcv", number=10,
repeats=3)
# train the SVM model
set.seed(7)
modelSvm <- train(Activity~., data=train, method="svmRadial",
trControl=control)
modelSvm #Print the results, 3 iterations, 10 fold CV
test.Svm = predict(modelSvm,test);
res.Svm = table(test.Svm,testog$Activity)
res.Svm
res.rf #Confusion matrix
res.Svmpc = res.Svm
for (i in 1:6) {
 for (j in 1:6)
 {
  res.Svmpc[i,j] = round(res.Svm[i,j]/sum(res.Svm[,j])*100,2)
 }
}
res.Svmpc #Confusion matrix in percentages
```

#Bagging Loops and SVM ensemble

```
trainnum=read.csv('HartrainUncorNum.csv', header = TRUE, sep =
"," )
testnum=read.csv('HartestUncorNum.csv', header = TRUE, sep = ","
)
testognum=read.csv('HartestUncorOGNum.csv', header = TRUE, sep
= "," )
#RandomForest
# Direct from the help page for the randomForest() function in R:
# mtry: Number of variables randomly sampled as candidates at
each split.
# ntree: Number of trees to grow.
# Creating a baseline for comparison by using the recommend
defaults for each parameter and mtry=floor(sqrt(p))and ntree=500.
#Using 10 fold CV with 3 repititions
control <- trainControl(method="repeatedcv", number=10,
repeats=3)
metric <- "Accuracy"
set.seed(123)
mtry <- floor(sqrt(ncol(train))) #Selecting mtry = 12
tunegrid <- expand.grid(.mtry=mtry)
rf_default <- train(Activity~., data=train, method="rf",
metric=metric, tuneGrid=tunegrid, trControl=control)
print(rf_default)
varImp(rf_default)
test.rf2 = predict(rf_default,test);
rf2.cf = table(test.rf2,testog$Activity)
rf2.cfpc = rf2.cf
for (i in 1:6) {
 for (j in 1:6)
 {
  rf2.cfpc[i,j] = round(rf2.cf[i,j]/sum(rf2.cf[,j])*100,2)
   if (i==j)
  {k=k+rf2.cf[i,j]}
  m=m+rf2.cf[i,j]
 }
}
rf2.cfpc #Confusion matrix in percentages
acc1=k/m*100
print(acc1)
# Random Search
control1 <- trainControl(method="repeatedcv", number=10,
repeats=3, search="random")
set.seed(123)
mtry <- sqrt(ncol(train))
rf_random <- train(Activity~., data=train, method="rf",
metric="Accuracy", tuneLength=15, trControl=control1)
print(rf_random)
plot(rf_random)
test.rf3 = predict(rf_random,test);
rf3.cf = table(test.rf3,testog$Activity)
rf3.cfpc = rf3.cf
k=0; m=0
for (i in 1:6) {
 for (j in 1:6)
 {
  rf3.cfpc[i,j] = round(rf3.cf[i,j]/sum(rf3.cf[,j])*100,2)
  if (i==j)
  {k=k+rf3.cf[i,j]}
  m=m+rf3.cf[i,j]
 }
}
rf3.cfpc #Confusion matrix in percentages
acc=k/m*100
print(acc)
control2 <- trainControl(method="repeatedcv", number=10,
repeats=3, search="grid")
set.seed(123)
tunegrid <- expand.grid(.mtry=c(10:20))
```

```
rf_gridsearch <- train(Activity~., data=train, method="rf",
metric=metric, tuneGrid=tunegrid, trControl=control2)
print(rf_gridsearch)
plot(rf_gridsearch)
test.rf4 = predict(rf_gridsearch,test);
rf4.cf = table(test.rf4,testog$Activity)
rf4.cfpc = rf4.cf
k=0; m=0
for (i in 1:6) {
 for (j in 1:6)
 {
  rf4.cfpc[i,j] = round(rf4.cf[i,j]/sum(rf4.cf[,j])*100,2)
  if (i==j)
  {k=k+rf4.cf[i,j]}
  m=m+rf4.cf[i,j]
 }
}
rf4.cfpc #Confusion matrix in percentages
acc2=k/m*100
print(acc2)
#RandomForest
library(randomForest)
rf.har = randomForest(Activity~., train, mtry=12, ntree=1000)
test.rf = predict(rf.har, test)
varImpPlot(rf.har) #GiniIndex
varImp(rf.har)

res.rf = table(test.rf,testog$Activity)
res.rf #Confusion matrix
res.rfpc = res.rf
for (i in 1:6) {
 for (j in 1:6)
 {
  res.rfpc[i,j] = round(res.rf[i,j]/sum(res.rf[,j])*100,2)
 }
}
res.rfpc #Confusion matrix in percentages
```

#SVM ensemble

```
#Renaming classifier with numbers 0 = Laying, 1 = Sitting, 2 =
Standing, 3 = Walking, 4 = Walking_Downstairs, 5 =
Walking_Upstairs
library(classyfire)
ens <- cfBuild(inputData = trainnum[,-1], inputClass =
trainnum$Activity, bootNum = 10, ensNum = 6,parallel = TRUE, cpus
= 4, type = "SOCK") #10 bootstrap iterations, 6 classifiers, running
parallel on 4 cpus
attributes(ens)
getAvgAcc(ens)$Test #Overall avg test accuracy
getAvgAcc(ens)$Train #Overall avg train accuracy
predRes  <- cfPredict(ens, testnum) #Predicting on holdout set
getAcc(ens)
getAcc(ens)$Test
getAcc(ens)$Trai
getConfMatr(ens)
optParam <- getOptParam(ens)
optParam
# Show the percentages of correctly classified samples in
# a barplot with or without text respectively
# Show the percentages of classified and missclassified samples
# in a barplot simultaneously with and without text

ggClassPred(ens, displayAll = TRUE)
ggClassPred(ens, position = "stack", displayAll = TRUE)
ggClassPred(ens, position = "stack", displayAll = TRUE, showText =
TRUE)
# Alernatively, using a dodge position
ggClassPred(ens, position = "dodge", displayAll = TRUE)
ggClassPred(ens, position = "dodge", displayAll = TRUE, showText =
TRUE)
ggEnsTrend(ens)
# Plot with text
ggEnsTrend(ens, showText  = TRUE)
# Plot with text; set different limits on y axis
ggEnsTrend(ens, showText  = TRUE, ylims=c(90, 100))
ggEnsHist(ens)
# Density plot of the test accuracies in the ensemble
ggEnsHist(ens, density = TRUE)
# Density plot that highlights additional descriptive statistics
ggEnsHist(ens, density = TRUE, percentiles=TRUE)
ggEnsHist(ens, density = TRUE, percentiles=TRUE, mean=TRUE)
ggEnsHist(ens, density = TRUE, percentiles=TRUE, median=TRUE)

#Comparing holdout set results
qsvm.cf = table(predRes[,1], testognum$Activity )
qsvm.cfpc =qsvm.cf
k=0; m=0
for (i in 1:6) {
 for (j in 1:6)
 {
  qsvm.cfpc[i,j] = round(qsvm.cf[i,j]/sum(qsvm.cf[,j])*100,2)
  if (i==j)
  {k=k+qsvm.cf[i,j]}
  m=m+qsvm.cf[i,j]
 }
}
qsvm.cfpc #Confusion matrix in percentages
acc3=k/m*100
print(acc3)
```

#SVM with radial kernel after PCA with standardization

```
zScaleTrain = preProcess(trainnum[,3:ncol(trainnum)])
scaledX = predict(zScaleTrain, trainnum[, 3:ncol(trainnum)])
pcaTrain = preProcess(scaledX, method = "pca", thresh = 0.8)
pcaTrain #This gives you the number of components required to get
80% variance
pr.out=prcomp(scaledX, scale=TRUE) #Get all the PCs
pr.train = predict(pr.out,trainnum[,3:ncol(trainnum)])[,1:52]
pr.train.mat <- cbind(pr.train, trainnum$Activity)
write.csv(pr.train.mat, file = 'prtrain.csv')
pr.tr=read.csv('prtrain.csv', header = TRUE, sep = ","  )
pr.tr=pr.tr[,-1]
library(classyfire)
```

```
ens <- cfBuild(pr.tr[,-ncol(pr.tr)], inputClass = pr.tr$X.1, bootNum =
5, ensNum = 6,parallel = TRUE, cpus = 4, type = "SOCK") #5 bootstrap
iterations, 6 classifiers, running parallel on 4 cpus
attributes(ens)
getAvgAcc(ens)$Test #Overall avg test accuracy
getAvgAcc(ens)$Train #Overall avg train accuracy
tst = testnum[,-1]
pr.test = predict(pr.out,tst)[,1:52]
predRes  <- cfPredict(ens, pr.test) #Predicting on holdout set
getAcc(ens)
getAcc(ens)$Test
getAcc(ens)$Trai
getConfMatr(ens)
optParam <- getOptParam(ens)
optParam
# Show the percentages of correctly classified samples in
# a barplot with or without text respectively
# Show the percentages of classified and missclassified samples
# in a barplot simultaneously with and without text
ggClassPred(ens, displayAll = TRUE)
ggClassPred(ens, position = "stack", displayAll = TRUE)
ggClassPred(ens, position = "stack", displayAll = TRUE, showText =
TRUE)
# Alernatively, using a dodge position
ggClassPred(ens, position = "dodge", displayAll = TRUE)
ggClassPred(ens, position = "dodge", displayAll = TRUE, showText =
TRUE)
ggEnsTrend(ens)
# Plot with text
ggEnsTrend(ens, showText  = TRUE)
# Plot with text; set different limits on y axis
ggEnsTrend(ens, showText  = TRUE, ylims=c(90, 100))
ggEnsHist(ens)

# Density plot of the test accuracies in the ensemble
ggEnsHist(ens, density = TRUE)

# Density plot that highlights additional descriptive statistics
ggEnsHist(ens, density = TRUE, percentiles=TRUE)
ggEnsHist(ens, density = TRUE, percentiles=TRUE, mean=TRUE)
ggEnsHist(ens, density = TRUE, percentiles=TRUE, median=TRUE)

#Comparing holdout set results
qsvm.cf = table(predRes[,1], testognum$Activity )
qsvm.cfpc =qsvm.cf
k=0; m=0
for (i in 1:6) {
  for (j in 1:6)
  {
    qsvm.cfpc[i,j] = round(qsvm.cf[i,j]/sum(qsvm.cf[,j])*100,2)
    if (i==j)
    {k=k+qsvm.cf[i,j]}
    m=m+qsvm.cf[i,j]
  }
}
qsvm.cfpc #Confusion matrix in percentages
acc3=k/m*100
print(acc3)

#SVM with linear kernel
numPredictors=ncol(HARdata)#number of columns
numPredictors #number of predictors
str(HARdata)#for checking the structure of the dataframe
head(HARdata)#for checking top 5 rows of the dataframe
library(caret)
#10 fold cross validation repeated 3 times
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats =
3)
#Grid Cost parameter(C)= Penalty term. Higher the cost parameter,
#better the classification but lesser is the smoothness of the
boundary
grid <- expand.grid(C = c(0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25,
1.5, 1.75, 2,5))
set.seed(3233)
#training
##Second column is subject id, which cannot be a predictor
##The value of header 'Activity' changes to 'ï..Activity' while
importing
svm_Linear_Grid <- train(ï..Activity ~., data = train[,-2], method =
"svmLinear",trControl=trctrl,
                    preProcess = c("center", "scale"),tuneGrid =
grid,tuneLength = 10)

svm_Linear_Grid
plot(svm_Linear_Grid)
#Testing
test_pred_grid <- predict(svm_Linear_Grid, newdata =
test[,3:numPredictors])
##The value of header 'Activity' changes to 'ï..Activity' while
importing
confusionMatrix(test_pred_grid, test$ï..Activity )
```