A Capstone Project for Machine Learning Nanodegree at Udacity

by Glenn Mossy December 26, 2019

https://github.com/gmossy/human-activity-recognition

1 Definition

1.1 Project Overview

The goal of activity recognition is to recognize common human activities in real life settings. Human activity recognition or HAR for short is the problem of predicting what a person is doing based on a trace of their movement while doing a specific activity using sensors. This is a classification project, since the variable to be predicted is categorical. HAR plays an important role in people's daily life for its competence in learning profound high-level knowledge about human activity from raw sensor inputs.

Human Activity Recognition is appealing in applications ranging from security-related applications, logistics support, location-based services and exploitation of Ambient Intelligence by helping handicapped or elderly people to live more independently, rate health levels for healthcare, activate household controls, appliances and provide safety by monitoring. HAR is very multifaceted, useful many fields and may also be referred to as goal recognition, behavior recognition, location estimation, monitoring etc. With knowledge from the models, results can be used to perform Ambient Assisted Living. Using better capability to predict human behavior means we can proactively deliver specialized programs and targeted interventions to the populations that are most in need.

Human Activity Recognition aims to identify the actions carried out by a person during a set of observations from the person's movements and surrounding environment, and then ultimately used to build and use models for a wide range of human activities. Recognition can be accomplished by exploiting the information retrieved from various sources such as body-worn sensors. It is believed that by empowering a computer to monitor the behavior of agents, computers can become and suited to act on our behalf.

This project explores a readily available dataset from the UCI [1] that is based on the work "Human Activity Recognition on Smartphones with Awareness of Basic Activities and Postural Transitions", by Jorge-Luis Reyes-Ortiz, Luca Oneto, Alessandro Ghio, Albert SamÃ¡, Davide Anguita and Xavier Parra. This work expands on their earlier work done [2] "A Public Domain Dataset for Human Activity Recognition Using Smartphones." by adding a study of Postural Transitions. In this project I build several machine learning models to train and predict the human activities such as walking, walking upstairs, walking downstairs, sitting, standing, or laying along with postural transitions from one of standing, sitting or lying positions to their opposites. Accuracy scoring and analysis will be done on the performance of each of the models. One of the challenges with this dataset it will be an unbalanced dataset, with the postural activities do not have as many datapoints as the other activities.

The project in which this paper is based [1] uses sensors that are body mounted accelerometer and gyroscope but many additional sensors, such as microphones, motion video capture, human physiological signals or vital signs could be later added to this project in order to capture data to provide additional features to the model that would improve the accuracy, provide more data for the models, and increase the number of activities that can be recognized, with the ultimate goal of providing a real-time prediction of each and every activity that the person is doing.

My interest in this field stems from participation in AT&T Software Symposium Hackathons in which my team came in second place in 2018 by creating a project to provide assisted living support to wheelchair bound, severely disabled patients. In my part-time work as adjunct instructor at Frederick Community College I teach students how to program Arduino microcontrollers and sensors to interact with the physical environment. In my full-time position I'm am with AT&T Public sector where I am a Senior Solutions Architect providing technical leadership on Technology Innovation initiatives for our Nations Department of Defense, DISA, Army and Cyber programs. In this role I support multi-vendor, high risk/reward integration projects, including rapid software development, enterprise networking and automation that enables business processes, and provides operations automation.

1.2 Problem Statement

Human Activity Recognition or HAR for short, is the problem of predicting what a person is doing based on a time series recording of their movement using sensors. The idea is that once the subject's activity is recognized and known, it can be saved in a model that can then be used to recognize future those activities with high accuracy when applied, and then provide useful assistance with the result.

The objective is to build a classifier that can classify the different activity types, even when extended to new data (i.e.: new participants).

Movements that we will study in this project will be six normal indoor activities such as walking, standing, sitting, laying and walking up and downstairs, also six postural transitional activities will be included these are moving from standing to sit, or laying down to sit, or laying to standing. By using sensors in a smartphone, 3-axial linear acceleration from an accelerometer and 3-axis angular velocity from gyroscope data in three dimensions (x, y, z) is recorded.

It is a challenging problem because there is no clear analytical way to relate the sensor data to specific actions in a general way. It is technically challenging because of the large volume of sensor data collected (e.g. tens or hundreds of observations per second) and the classical use of hand-crafted features and heuristics from this data in developing predictive models.

In this project, we will examine data that will be an unbalanced data set, with the normal activities will have more data points than the postural activities, refer to table 1 for a list of the activities. Using confusion matrixes, and scoring that calculates precision, recall, f1, and accuracy scores, and r2, and the Matthews correlation coefficient for each model, evaluate and discuss the accuracy of how each of the models perform. It will also be noted how long it takes to train the models, which can provide insight on models that can be used in near real-time to learn new activities.

Classical approaches to the problem involve training machine learning models, such as ensembles of logistic regression or random forest methods, while in this project, the objective is build machine learning models to classify activities into one of the twelve activities performed and then score and analyze the results, and special models to evaluate it is there is improvement, using a stacking model and a basic artificial neural network model to see how well they do.

1.3 Metrics

Evaluation metrics are very important because they empower data scientists to intuitively explain results to the reader. The objective of this project is to predict the activity performed by the subject accurately, based on accelerometer and gyroscope readings, so misclassification rate is chosen with several assessment measures to compare the performance of different models. With the dataset ready, the HAR problem essentially becomes a classification problem with labeled data under supervised learning. This project addresses the HAR problem by applying different types of machine learning techniques: supervised learning using several classifiers (e.g. Logistic Regression, AdaboostClassifier, GradientBoostClassifier, K nearest Neighbors, Random Forest Classifier, Linear Support Vector Machines, and Stacking (a set of the classifiers, which creates a meta-classifier). And finally, a basic Artificial Neural Network (ANN) is used to corroborate the results from the other classifiers. For each of the proposed algorithms, accuracy and a set of metrics are used as the common metrics to compare performances.

The **classification Accuracy** is what we usually mean when we use the term accuracy. It is the ratio of number of correct predictions to the total number of activity input samples. We will record the training and the testing accuracy. The training accuracy tells us how well the training was performed on the training samples, while the testing accuracy tells us the ratio of number of correct predictions to the total number of activity input test samples. Most of the time we use classification accuracy to measure the performance of our model, however it is not enough to truly judge our model. Accuracy score coupled with a confusion matrix give a more robust assessment of the model performances.

Provided via the Classification report, there are five other metrics that will be used to measure performance of the models, the **confusion matrix, the Precision-Recall scores, F1 Score (weighted average and macro average).** The **Macro average** will be used to give us an average ratio of how well the postural activities are being recognized. Additionally, the R2 score, and the Matthews correlation coefficient (MCC) will be calculated for each. Also, we will record the training time to compare how fast the model performance is for training and provide comparison to our goal of achieving least training time performance, but accuracy.

The confusion matrix, which is an output table that summarizes the number of true positives, true negatives, false positives, and false negatives, and describes the complete performance of the model. The confusion matrix is a straightforward evaluation metric if we assume that the results have balanced classes, in the number of true positives to true negatives. A confusion matrix is particularly helpful to evaluate the models used. [6] Specifically, precision represents the number of correctly classified activities out of all the activities being classified. [11] In our case, we do have an imbalanced dataset.

So, the **Precision-Recall evaluation metrics** can be used if we find that the dataset is imbalanced. Precision represents the number of correctly classified activities out of all the activities being classified; while recall represents each type of activity in truth how many the machine correctly classified. Precision is the number of true positives over the number of total positive predictions. A high precision means that there will be many true positives and a low false positive rate. Recall is the number of true positives over the number of total actual positives in the dataset. A high recall means that the model has captured most of the true positives and has a low false negative rate. An optimal solution needs to have high precision and high recall, rejecting only those activities that don't match the class, i.e. high precision, and matching the correct activity to the class in the dataset (high recall).

F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise the classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

The **r2_score function** computes the coefficient of determination, usually denoted as R^2. . It represents the proportion of variance (of y) that has been explained by the independent variables in the model. It provides an indication of goodness of fit and therefore a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance. As such variance is dataset dependent, R^2 may not be meaningfully comparable across different datasets. The best possible R2 score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

The **Matthews Correlation Coefficient (MCC)** has a range of -1 to 1 where -1 indicates a completely wrong binary classifier while 1 indicates a completely correct binary classifier. Using the MCC allows one to gauge how well their classification model/function is performing. The Matthews correlation coefficient is used in machine learning as a measure of the quality of binary (two-class) classifications. It considers true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes.

We will look at what the R2 score and Matthews correlation coefficient tells us about the performance of the test data. Depending on the imbalance of the data, the mcc may or may not be a more positive result, while the R2 score which is a statistical measure will tell us how close the data are to the fitted regression line.

2 Analysis

2.1 Data Exploration

A standard human activity recognition dataset is the 'UCI [1] that is based on the work "Human Activity Recognition on Smartphones with Awareness of Basic Activities and Postural Transitions" that was made available in 2014. It was prepared and made available by Jorge-Luis Reyes-Ortiz, et al. from the University of Genova, Italy and is described in full in their 2014 paper. The dataset was made available and can be downloaded for free from the UCI Machine Learning Repository. The README.MD file in this project describes how to obtain and use the dataset. Here is a summary of the dataset files in Table 1, in addition, the original source dataset includes the raw data that was not used in this project, but can be used by the reader to expand on this work to manipulate the data based to improve on the work in this project.

Table 1 Summary of the components of the Dataset Used

X_train	The shape of the training set dataframe is (7767, 561) Figure 2 is an example
y_train	The shape of the training set labels dataframe is (7767, 1) Figure 3 is an example
X_test	The shape of the testing set dataframe is (3162, 561)
y_test	The shape of the test set labels dataframe is (3162, 1)
Activity_labels	Links the class labels with their activity name. See Table 2 for a list.
subject_train	Each row identifies the subject who performed the activity for each window sample.
subject_test	Each row identifies the subject who performed the activity for each window sample.
features	List of all features, See Table 3 for a list of the feature signal types.

The data was collected from 30 subjects aged between 19 and 48 years old performing 12 standard activities while wearing a waist-mounted smartphone that recorded the movement data. The movement data recorded was the x, y, and z accelerometer data (linear acceleration) and gyroscopic data (angular velocity) from a smartphone, specifically a Samsung Galaxy S II. Video was also recorded of each subject performing the activities and the movement data was labeled manually from these videos and can be used for reference to see how the activities were recorded. The HAR dataset was collected in laboratory conditions but volunteers were asked to perform freely the sequence of activities for a more naturalistic dataset. The set of experiments in which each person was instructed to follow was a protocol of activities while wearing a waist-mounted Samsung Galaxy S II smartphone.

Each subject performed the sequence of activities twice; once with the device on their left-hand-side and once with the device on their right-hand side. After a series of preprocessing, the final dataset has 561-feature vector per example derived from 17 action patterns and 17 functions over 12 activities labels. So, the basic activities plus the postural activities are as follows:

Table 2 The Activities and activity_labels

Standard Activities	0	WALKING	Moving
	1	WALKING_UPSTAIRS	
	2	WALKING_DOWNSTAIRS	
	3	SITTING	Stationary
	4	STANDING	
	5	LAYING	
Postural Activities	6	STAND_TO_SIT	Moving
	7	SIT_TO_STAND	
	8	SIT_TO_LIE	
	9	LIE_TO_SIT	
	10	STAND_TO_LIE	
	11	LIE_TO_STAND	

There is a datafile " subject_id_test.txt" that contains rows that identifies the subject who performed the activity for each window sample. Its range is from 1 to 30.

```
df=pd.DataFrame(subject_train)
df.T
```

	0	1	2	3	4	5	6	7	8	9	...	7757	7758	7759	7760	7761	7762	7763	7764	7765	7766
0	1	1	1	1	1	1	1	1	1	1	...	30	30	30	30	30	30	30	30	30	30

1 rows × 7767 columns

Figure 1 subject_train dataset example

Observations were recorded at 50 Hz (i.e. 50 data points per second), that was consistent across the dataset. The raw data is not available, but instead a pre-processed version of the dataset was made available. The dataset used in this project had already been pre-processed. In particular, the SBHAR data generated around 5-hours of experimental data, and was also pre-processed with noise filters Instead, The pre-processing steps included: Pre-processing accelerometer and gyroscope using noise filters and other data transformations were also applied including the calculation of Jerk signals from time, body linear acceleration and angular velocity information; magnitude using Euclidean norm and, the frequency domain signals using Fast Fourier Transform (FFT).

From this dataset we get a feature vector of 561 elements. The readings were split into training data (71.07%) and the remaining as testing data (28.93%) sets based on data for a total of 30 subjects, e.g. 21 subjects for train and nine for test. After checking for duplicates and null values, both datasets were found to be clean. The signal data points are captured on 3 axes and have been preprocessed and bounded between [-1,1].

The training dataset (X_train) has 7767 rows x 561 columns and the test dataset (X_test) has 3162 rows and 561 columns. Here's an example (only some of the 561 columns and 7767 rows) of what part of the training dataset looks like:

	tBodyAcc-Mean-1	tBodyAcc-Mean-2	tBodyAcc-Mean-3	tBodyAcc-STD-1	tBodyAcc-STD-2	tBodyAcc-STD-3	tBodyAcc-Mad-1	tBodyAcc-Mad-2	tBodyAcc-Mad-3	tBodyAcc-Max-1	...	fBodyGyroJerkMag-MeanFreq-1	fBodyGyroJerkM Skewne
0	0.043580	-0.005970	-0.035054	-0.995381	-0.988366	-0.937382	-0.995007	-0.988816	-0.953325	-0.794796	...	-0.012236	-0.31
1	0.039480	-0.002131	-0.029067	-0.998348	-0.982945	-0.971273	-0.998702	-0.983315	-0.974000	-0.802537	...	0.202804	-0.60
2	0.039978	-0.005153	-0.022651	-0.995482	-0.977314	-0.984760	-0.996415	-0.975835	-0.985973	-0.798477	...	0.440079	-0.40
3	0.039785	-0.011809	-0.028916	-0.996194	-0.988569	-0.993256	-0.996994	-0.988526	-0.993135	-0.798477	...	0.430891	-0.13
4	0.038758	-0.002289	-0.023863	-0.998241	-0.986774	-0.993115	-0.998216	-0.986479	-0.993825	-0.801982	...	0.137735	-0.36

5 rows × 561 columns

Figure 2 X_train data sample

The labeling dataset y_train and has 1 row and 7767 columns as shown below, and the y_test dataset has 3162 columns.

	0	1	2	3	4	5	6	7	8	9	...	7757	7758	7759	7760	7761	7762	7763	7764	7765	7766
Y	5	5	5	5	5	5	5	5	5	5	...	2	2	2	2	2	2	2	2	2	2

1 rows × 7767 columns

Figure 3 y_train dataset sample

The sensor signals are preprocessed by the authors of the original paper which involved applying noise filters and then sampled in fixed width windows (sliding windows) of 2.56 seconds each with 50% overlap. i.e.., each window has 128 readings. From each window, a feature vector was obtained by calculating variables from the time and frequency domain. So, in our dataset, each datapoint represents a window with different readings. The acceleration signal was separated into body and Gravity acceleration components using a low pass filter with a corner frequency of 0.3Hz. After that the body linear acceleration and angular velocity were derived in time to obtain jerk signals. Then the magnitude of the 3-dimensional signals was calculated using the Euclidian norm. Finally, the frequency domain signals were obtained by applying an FFT (Fast Fourier Transform). These signals are labeled as fBodyAcc-XYZ, and fBodyGyroMag, etc. Finally from the base signal readings there are calculations provided for mean value, max (largest value in the array), sma (Small signal magnitude area), arCoefficient (Autoregression coefficients, and correlation() coefficients between signals, meanFreq, skewness, kurtosis, energy bands measure, signal entropy and a few others

The acceleration signal from the smartphone accelerometer X axis in standard gravity unit's 'g'. The body acceleration signal obtained by subtracting the gravity from the total acceleration. The angular velocity vectors were measured by the gyroscope for each window sample. The units are radians/second. The input variables include time and frequency domain signals obtained from the smartphone sensors:

.

Table 1 feature input Inertial signals in the time and frequency domain

body_acceleration
gravity_acceleration
body_acceleration_jerk
body_angular_speed
body_angular_acceleration
body_acceleration_magnitude
gravity_acceleration_magnitude
body_acceleration_jerk_magnitude
body_angular_speed_magnitude
body_angular_acceleration_magnitude

An examination of the properties of the signal samples for the train dataset and the test dataset: Mean value, standard deviation, Quartile values, and Maximum values are close in range between the two datasets, so the test data is very similar to the training data.

Here is a list of the where to get the data and a list of the HAPT dataset files.

http://archive.ics.uci.edu/ml/machine-learning-databases/00341/

/HAPT Data Set/Train/subject_id_traintxt

/HAPT Data Set/Train/X_traintxt

/HAPT Data Set/Train/y_traintxt

/HAPT Data Set/activity_labels

/HAPT Data Set/features

/HAPT Data Set/activity_labels

/HAPT Data Set/features_info

2.1 Exploratory Visualization

The visualizations that were done include Number of Samples per Activity", “Samples of Activities provided by each Test Subject”, “Plots of 3 Axis Body Acceleration Mean Values with Respect to Frequency vs the Activity Type”, Visual exploration of the data in 3 dimensions with principal component analysis, and finally, “Visualization of Feature Distribution”. Below we explain why the visualization was chosen and how the chart is relevant to our analysis of the solution.

2.1.1 Number of Samples per Activity

Looking at the total samples count vs the 12 activities, we have an imbalance in the 12 classes of activities with the number of samples below 32 total datapoints each for the postural activities. The normal activities are the STANDING, SITTING, LAYING, WALKING, WALKING_DOWNSTAIRS, and WALKING_UPSTAIRS and the other are the postural activites.

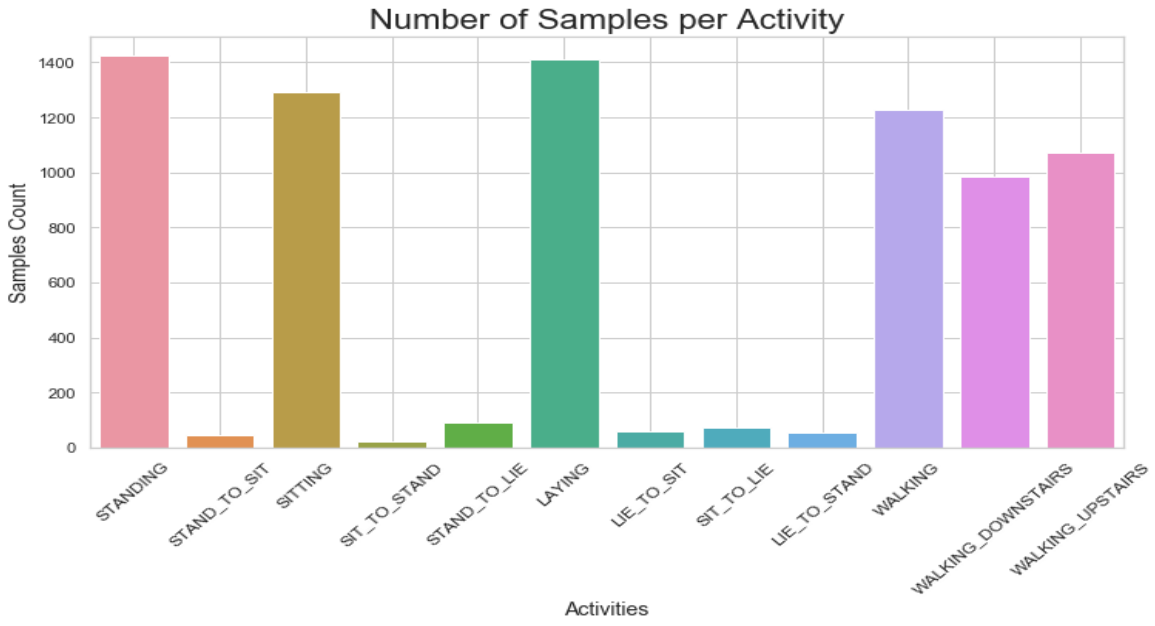


Figure 4 Number of Samples per Activity Chart

2.1.2 Samples of Activities provided by each Test Subject

All our test subject has performed the six normal activities between 37 and 85 samples. In future improvement samples I recommend that they should be taken more even between the subjects and would help balance out the test dataset. The six postural activities count is small, under 10 samples each subject, which is considerably lower than the normal activity samples. Increasing these samples closer to the count of the normal samples could significantly improve the outcome of our predictions. Below in Figure 5 is a Plot of Samples of Activities provided by each Test Subject.

Special note should be made of the “SIT_TO_STAND” count, which is the lowest of all, and ranges from activity ranges only from 0 to 10. This is not enough training data for proper analysis and prediction. Our results, as can be seen in the confusion matrixes, do poorly for this activity.

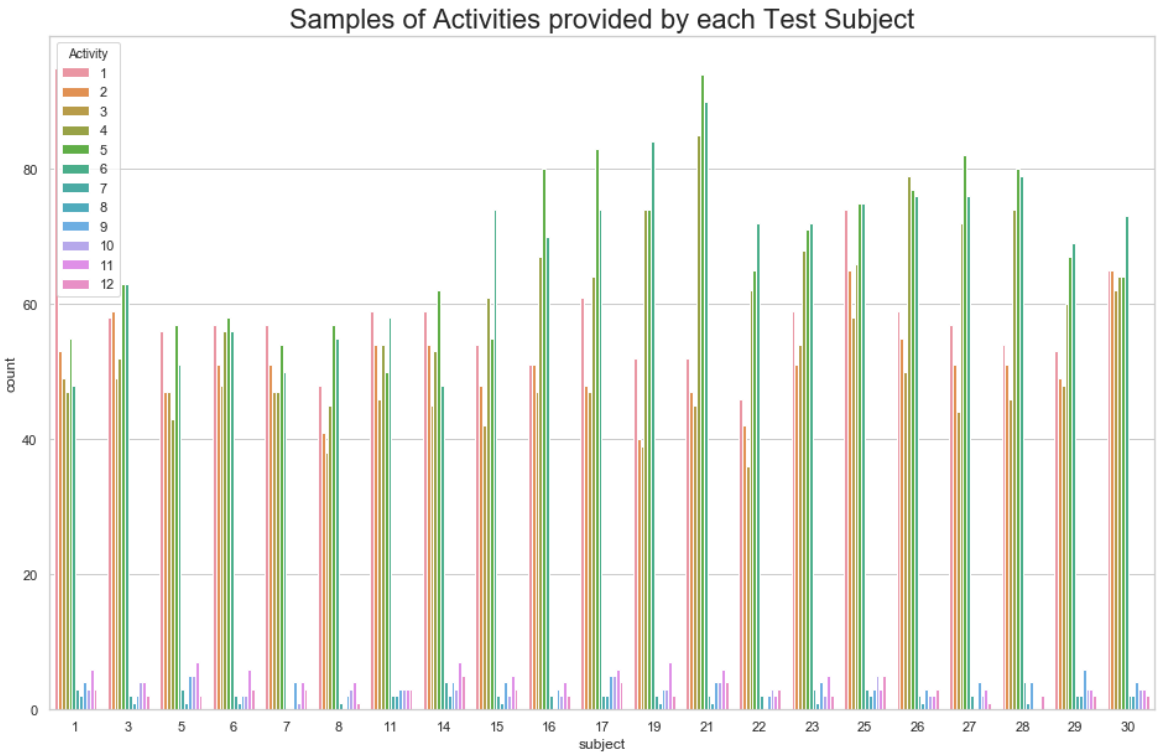


Figure 5 Samples of Activities provided by each Test Subject

2.1.3 Plots of 3 Axis Body Acceleration Mean Values with Respect to Frequency vs the Activity Type

Now we Plot the various features vs the activity type to examine the relationships visually and will note interesting observations. In the 3 Axis Body Acceleration plot, we identify there are three major groups of activities have the most similar frequency variations. The grouping can be seen as group 1, "walking, walking_upstairs, walking_downstairs, group 2, siting and standing, laying (these are the stationary activities we identified in the feature distribution visualization below, and group 3, the postural activities all are mostly similar. This means that we could possibly add some different scaling of the dataset value ranges to each of these 3 activity groups. This could reduce overfitting, if we then train the groups separately. For example, group 2 ranges from -1.0 to 0.0, the frequency plots have not ranged over 0.0. We could re-scale these types of activities together, whereas the other activities go above and below the 0.0 axis.

Special note should be made of the "SIT_TO_STAND" graph, because the signal has is missing to the right half of the graph. After reviewing the confusion matrix graphs, the true positives for this activity

ranges only from 0 to 10, and under 5 in most results. This lab capture should be redone.

Figure 6 Plots of 3 Axis Body Acceleration Mean Values with Respect to Frequency vs the Activity Type

2.1.4 Visual exploration of the data in 3 dimensions with principal component analysis

Using PCA chart show below, we can see 6 distinct color groupings within 3 main clusters - the 3 variations of walking are close together in the as one would expect. The static activities of Sitting/Standing/Laying are similarly close together, while the moving activities, Stand_to_Sit/Sit_to_Lie, etc are spread out across their associated components. In earlier work done [2] the authors did use these distinct six classes of activities of groupings. We will see how well we can still identify the activities, which are the postural activities, that are outliers from these six clusters. This identification could lend to redesigning the activities, for example standing_and_rotate instead of stand_to_lie, as that perhaps body turning rotation is an intermediate step when you transition from standing to laying down.

Special note can also be made of the “LAYING” graph, which looks to be abnormal for what should be a subject lying still in bed. However, the signal for it happens to be more unique than any of the others and may produce more true predictions.

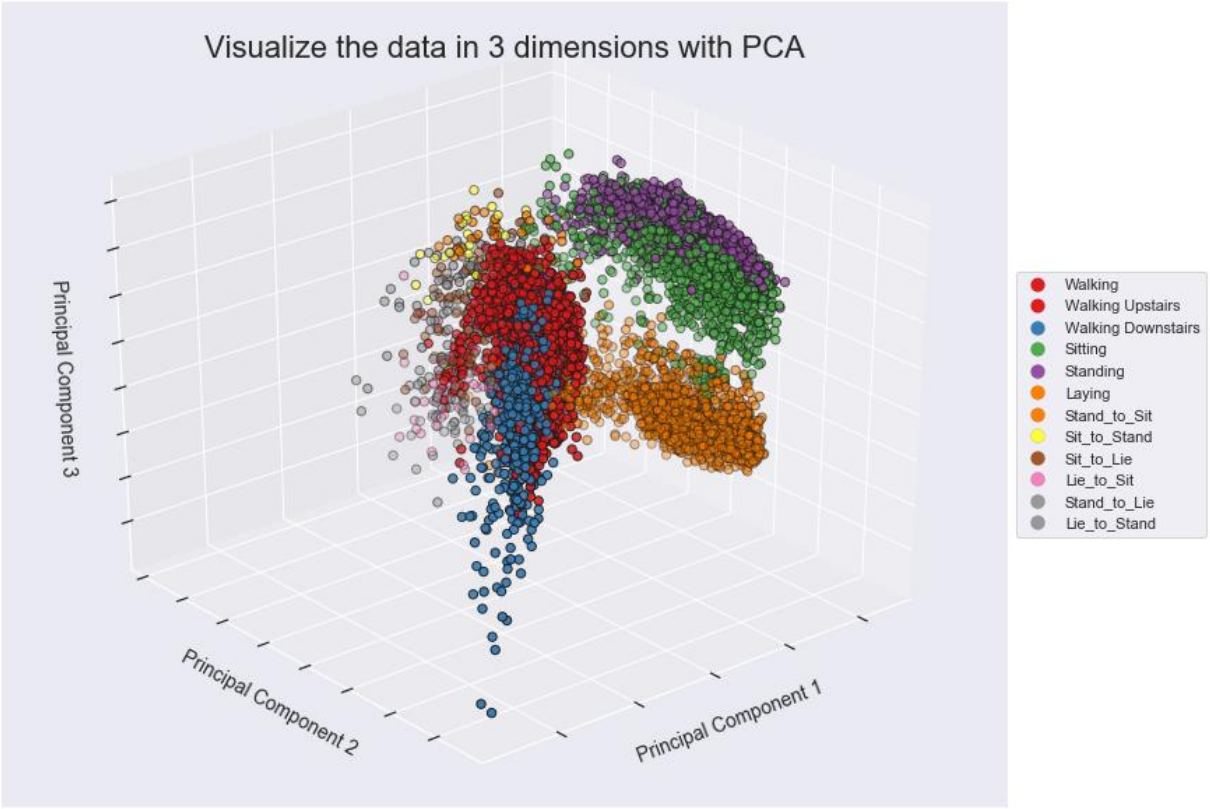
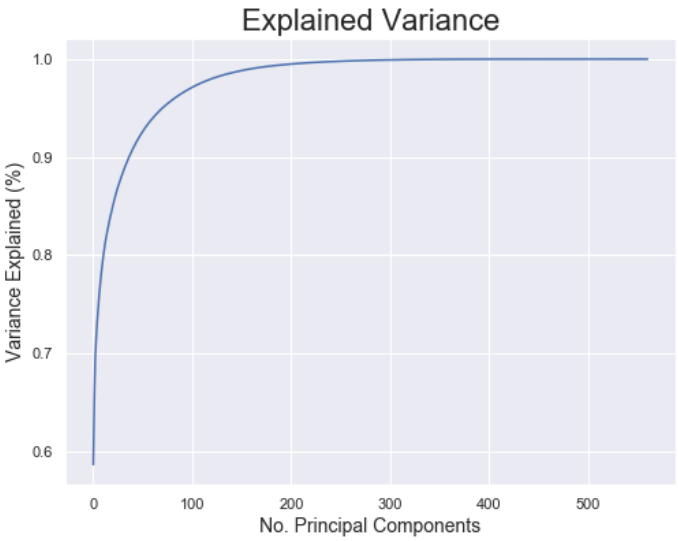


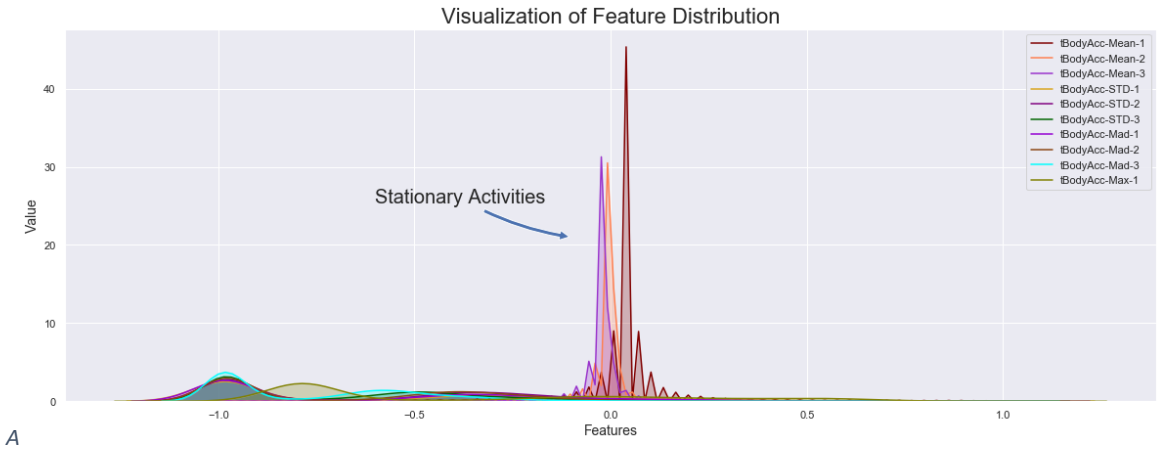
Figure 7 Principal Component Analysis Visualization in 3 dimensions



In this Plot PCA Explained Variance curve we evaluated if we need to use one set of components over the other, Using this Explained Variance Graph we can see about 100 components capture more than 95% of the variance in the X train data, while about 200 components capture nearly 100% of the variance in the training data. This variance is so close that in this project we will keep these components grouped together.

2.1.5 Visualization of Feature Distribution

In the following 3 charts, "Visualization of Feature Distribution plots", the features have been visualized to be certain of and identify which are the stationary, moving features and to look for anomalies. Three plots were required due to the number of features, 10 features per plot. If we have new features to engineer for example a new type of sensor, we need to be able to understand its properties. As an improvement to the models, the features that can be clustered can be separately modeled into a revised dataset into the model to prevent overfitting.



A

Figure 9 Visualization of Feature Distribution plot 1

Figure 10 Visualization of Feature Distribution plot 2

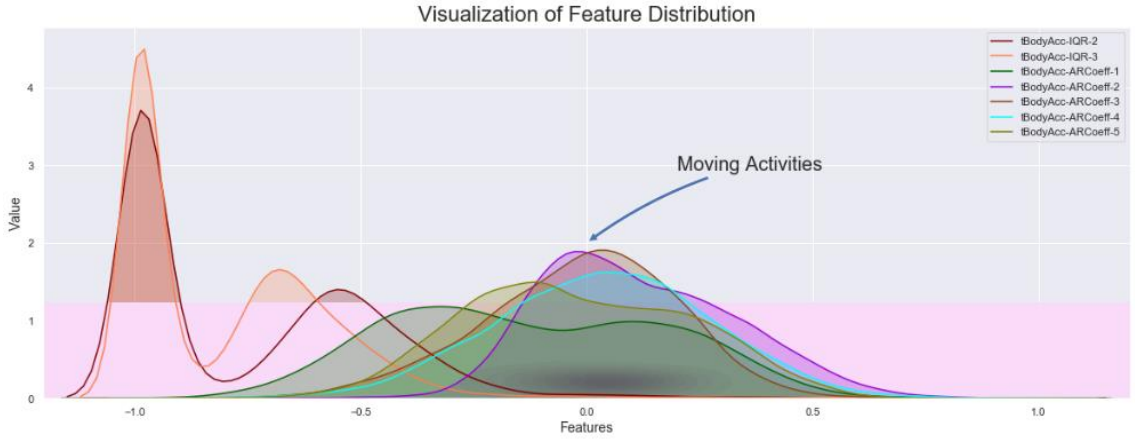


Figure 11 Visualization of Feature Distribution plot 3

2.2 Algorithms and Techniques

This stage of the activity recognition chain involves the training of a classification model that can discriminate the physical activity performed by a certain subject. Our objective is to eventually obtain a classifier that maximizes accuracy (i.e., the hit rate between the predicted and the actual class in a test set). The resulting classifier is intended to be used in a real-time cross-person prediction system that must be able to accurately predict the activity that a new user is performing.

This project addresses the HAR problem by applying different types of machine learning techniques: supervised learning using several classifiers (e.g. Logistic Regression, AdaboostClassifier, GradientBoostClassifier, K nearest Neighbors, Random Forest Classifier, Linear Support Vector Machines, and Stacking (a set of the classifiers). And finally, a basic Artificial Neural Network (ANN) is used to corroborate the results from the other classifiers. See **Table 2** below summary of the type of each model. For each of the proposed algorithms, accuracy and a set of metrics are used as the common metrics to compare performances.

First, to prototype a fast evaluation system, Python's machine learning library, using "scikit-learn" a benchmark in performance chosen is the robust method Logistic Regression.

The same data are input to different classifiers, and the results are stored for analysis. When selecting the classifiers, I used a wide set that includes classical and state-of-the-art techniques. I tested all the classifiers with various parameters, and the various parameters that were changed will be discussed in the sections below for each classifier's description. The classifiers with lower performances were discarded. The following classifiers were finally selected to be studied in more detail.

Table 2 Summary of Classifiers and Models Used in this solution

Classifier	Model Type
LogisticRegression	Benchmark
AdaboostClassifier	Ensemble Learner
GradientBoostClassifier	Ensemble Learner
kn-Neighbors	Distance Based Learner
RandomForestClassifier	Ensemble Learner/Distance Tree Based
LinearSVC	SVM
Linear GridSearchSVC	SVM with Gridsearch applied
Stacking Meta-Classifier	Stacking
ANN Neural Network Model	Basic Neural Network

2.2.1 **Logistic Regression Methods**

Logistic Regression is a statistical method for predicting binary classes. The outcome or target variable is binary in nature. It computes the probability of an event occurrence. Logistic Regression is very widely used on Financial forecasting, crime data mining, and binary classification problems. Logistic Regression is a variation of Linear Regression, useful when the observed dependent variable, *y*, is categorical. It produces a formula that predicts the probability of the class label as a function of the independent variables.

While Linear Regression is suited for estimating continuous values (e.g. estimating house price), it is not the best tool for predicting the class of an observed data point. In order to estimate the class of a data point, we need

some sort of guidance on what the most probable class for that data point would be. For this, we use **Logistic Regression.**

Logistic regression fits a special s-shaped curve by taking the linear regression and transforming the numeric estimate into a probability with the following function, which is called sigmoid function.

The version of Logistic Regression in Scikit-learn, supports regularization. Regularization is a technique used to solve the overfitting problem in machine learning models.

The is the final parameters used:

Logistic Regression Classifier	`solver='lbfgs'`
	Regularization = 1.0 the default. "C"
	k-fold cross-validation was not used in this project.

The LR classifier was used as the benchmark for this solution and performed very well at 94.66% testing accuracy, Macro average of 0.88 (from the classification report), and R2 of 0.939, and MCC of 0.937. From the confusion matrix below, we achieved fairly good results across all the classes, including the postural classes. Remarkable performance on the 3 LAYING, STAND_TO_SIT, and SIT_TO_STAND classes.

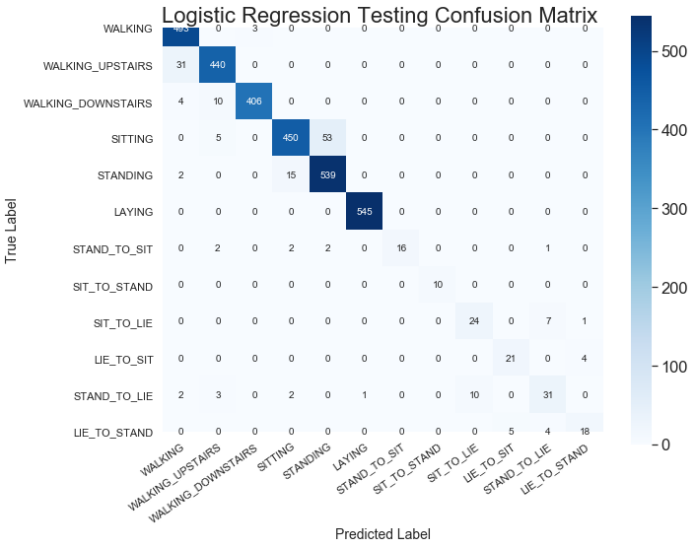


Figure 12 Confusion Matrix

Table 3 LR Classification Report (The Benchmark)

Classification_report	precision	recall	f1-score	support
WALKING	0.93	0.99	0.96	496
WALKING_UPSTAIRS	0.96	0.93	0.95	471
WALKING_DOWNSTAIRS	0.99	0.97	0.98	420
SITTING	0.96	0.89	0.92	508
STANDING	0.91	0.97	0.94	556
LAYING	1.00	1.00	1.00	545
STAND_TO_SIT	1.00	0.70	0.82	23
SIT_TO_STAND	1.00	1.00	1.00	10
SIT_TO_LIE	0.71	0.75	0.73	32
LIE_TO_SIT	0.81	0.84	0.82	25
STAND_TO_LIE	0.72	0.63	0.67	49
LIE_TO_STAND	0.78	0.67	0.72	27
accuracy			0.95	3162
macro avg	0.90	0.86	0.88	3162
weighted avg	0.95	0.95	0.95	3162

2.2.2 **Support Vector Machines (SVM) Methods**

These techniques build a model that predicts a target variable by learning interpretable decision rules inferred from the training data. Using a decision tree as a predictive model is widely used as a decision support tool because they are easily interpreted (i.e., they provide a sequence of decisions to obtain the final classification result).

LinearSVC - As indicated by PCA there are 3 different sets activities are like each other and could be grouped together. Maximizing margins and separating the different activities may lead to a stronger performance. SVM Linear could be a good model to implement.

The is the final parameters used and note the variations in hyperparameters tried in trail runs:

LinearSVC Classifier	Used default parameters. no k-fold cross validation used.

The LinearSVC Model training time was somewhat better than the LR Benchmark! And achieved about the same accuracy as the LR Benchmark at of LSVC at 94.62% vs. 94.66% LR. So LinearSVC was an excellent model.

From the confusion matrix below, we achieved the great results with LinearSVC across all the classes, including the postural classes:

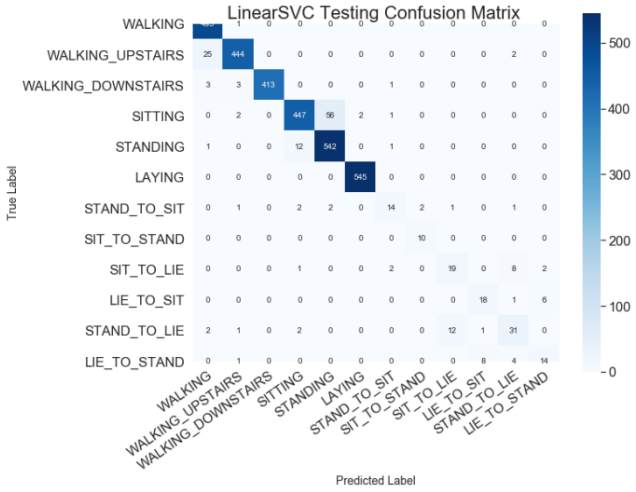


Figure 13 LinearSVC Testing Confusion Matrix

LinearSVC with GridSearch

Since the LinearSVC was an excellent model it was chosen to do an improvement of it by add "Grid Search" method. Our LinearSVC with Grid search with a linear kernel performed the best with 95.2% test accuracy using with linear kernel parameter, on second try by replacing the 'poly' kernel with linear. A gridsearch did lead to a significant improvement to the SVC model. The R2 score shows the highest so far of 0.971, quite a strong performance. Result is hard to beat. The LinearSVC with Grid search Model training time was 11x slower than the LR Benchmark.

LinearSVC with GridSearch
 	
Classifier
	`parameters = {'kernel': ['poly', 'rbf', 'sigmoid']}`
	`# 1st Try`
	
	`parameters = {'kernel': ['linear', 'rbf', 'sigmoid']} # 2nd try`

	Other parameters default. no k-fold cross validation used.

From the confusion matrix below, we achieved the best results across all the classes, including the postural classes and the best overall results of all the classifiers/models.

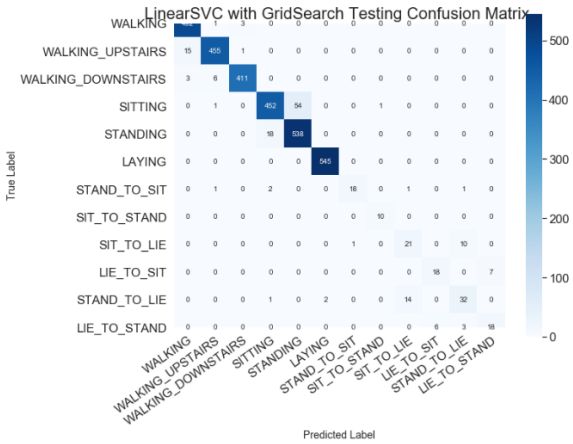


Figure 14 linearSVC with Gridsearch kernel': ['linear', 'rbf', 'sigmoid'] confusion matrix

2.2.3 **Decision Tree-Based Methods**

These techniques build a model that predicts a target variable by learning interpretable decision rules inferred from the training data. Using a decision tree as a predictive model is widely used as a decision support tool because they are easily interpreted (i.e., they provide a sequence of decisions to obtain the final classification result).

A decision tree base classifier was tried for the adaboost classifier but did not perform as well as using the Random Forest as the base classifier.

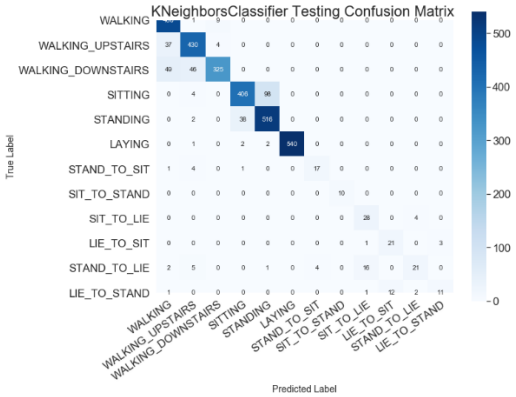
```
# Setting max_features=4 in the decision tree classifier used as the base classifier
# for AdaBoost will increase the convergence rate # base =
DecisionTreeClassifier(max_features=5, max_depth=6)
```

2.2.4 **Distance-Based Methods**

These algorithms assume that the data have some type of similarity and relations based on geometric properties and can be grouped according to these patterns. The k-nearest neighbors' algorithm (k-NN) is one of the simplest and most effective nonparametric machine learning algorithms for classification and regression. The K-nearest neighbors' method is a simple model but requires that the same activity types mostly generate similar signal data. In our tests, the k-NN classifier obtained better results for online recognition when compared to a decision tree but was the worst overall performer of the classifiers that were used in this project. After testing several parameters, we eventually used the k-NN algorithm with the following scikit-learn parameters.

The K-nearest neighbors (Knn) model performed the worst of all the models at 89% and found that attempts were unable to improve the performance.

k-nearest neighbors' Classifier	Optimal No. Of Neighbors: 7 Accuracy Score: 89.247 % Attempted up to 50 Neighbors, but accuracy score decreased, and optimal calculations reported either 7 or 13 without any improvement over 89%.



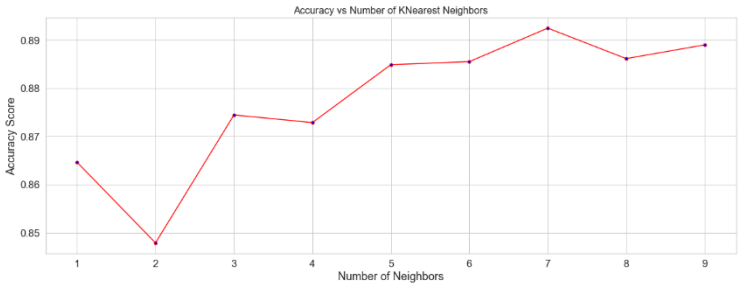


Figure 15 K-Neighbors Accuracy Score vs. Number of Knn Graph

Figure 16 K-nearest neighbors (Knn) confusion matrix

2.2.5 **Kernel Methods**

These models perform pattern analysis based on a kernel function, which is a similarity function over pairs of data points.

2.2.6 **Ensemble Learners**

These models combine the predictions of several base estimators built with a given learning algorithm to improve the results and robustness over a single estimator. Instead of picking just one of the machine learning solutions we have developed, we evaluate whether an ensemble of the models leans to an improved classification rate. This is because each of the standalone solutions have different strengths and weaknesses, and the compensation for the strengths and weaknesses between the models will generally lead to a better result of them all together in an ensemble. We will use a couple ensemble models with boosting. Boosting is a method of converting weak learners into strong learners. In boosting, each new tree is a fit on a modified version of the original data set.

Random Forest Classifier: The random forest did not perform as well most of the other classifiers that were tried, but better than KNN at 90.9% accuracy as our data is highly dimensional, and the randomness from bootstrap sampling so this classifier works better on data that is structured closer in types, such as just the six normal activities. Using just a subset of features may help correctly classify more activities. Could not achieve over 90.9% testing accuracy, and training time was 102 seconds. The **Random Forest** model training time was 10x slower than the LR Benchmark. From the confusion matrix below, we achieved fairly good results across the normal classes.

Random Forest Classifier	n_estimators – ranged in a loop from 15 to 400
	Default values used; no k-fold cross validation used.

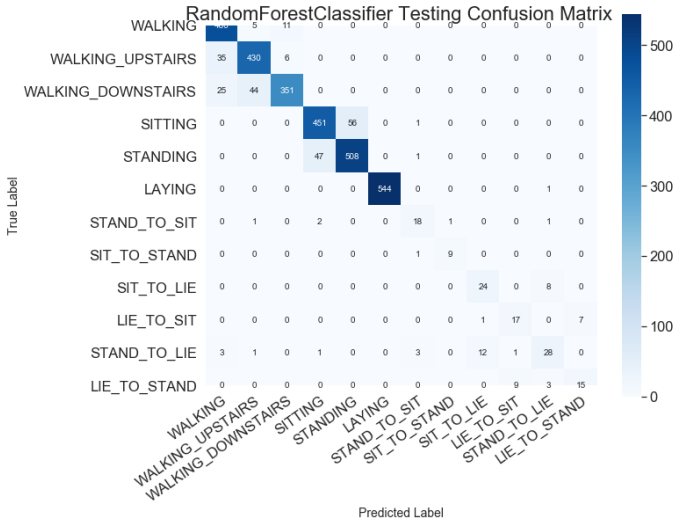


Figure 17 RF Confusion Matrix

GradientBoost classifier

Gradient Boosting trains models in a gradual, additive and sequential manner. The gradient boosting algorithm (gbm) Algorithm begins by training a decision tree in which each observation is assigned an equal weight. After evaluating the first tree, we increase the weights of those observations that are difficult to classify and lower the weights for those that are easy to classify. The second tree is therefore grown on this weighted data. Here, the idea is to improve upon the predictions of the first tree. Good real-world applications of GB are anomaly detection in supervised learning settings where data is often highly unbalanced, in which our activities are different with static and moving classes. It was chosen to use GradientBoosting model for this project because of its strengths in that it performs very good for large datasets, make use of its stronger ability on unbalanced datasets, reduces bias and variance, combines multiple weak predictors to a build strong predictor. **Weaknesses:** Some of the weaknesses of the GradientBoosting model is that it typically requires a relatively high training time, relative to Logistic Regression, and some over-fitting if the data sample is too small or if the data is noisy. Training generally takes longer because trees are built sequentially. Finally, the GB model does require complex tuning. There are typically three parameters: number of trees, depth of trees and learning rate, and each tree built is generally shallow.

The parameters that we used that were the final parameters use were:

GradientBoosting Classifier	n_estimators – ranged in a loop from 15 to 500
	max_features=5
	learning_rate=0.1 (performed runs with values of .01, .2 and accuracy went down from 93.2% to 91.1%, and 92.6%
	max_depth=3 (default)
	max_leaf_nodes=None,
	no k-fold cross validation used.

The GradientBoosting classifier performed very well after refinement noted by adjusting the learning rate at 93.23% accuracy on the testing data. From the confusion matrix below, we achieved fairly good results across

the normal classes, it did not do well on the anomaly of the SIT_TO_STAND activity signal data. The GradientBoosting model training time was 7x slower than the LR Benchmark at 79 seconds.

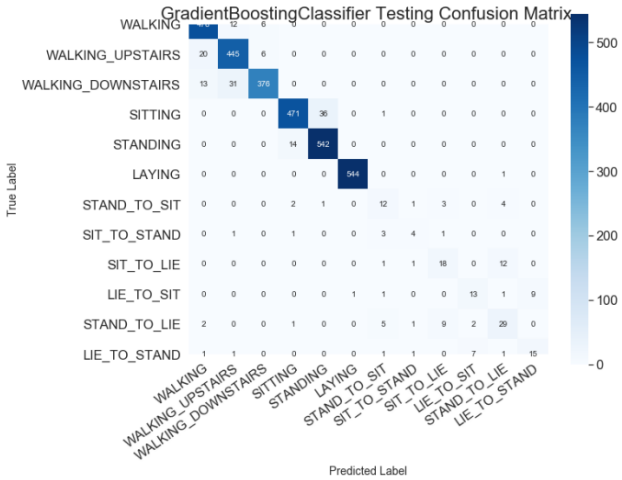


Figure 18 GBC Confusion Matrix

AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. [9]

AdaBoost Classifier	`algorithm="SAMME".`
	The default base-estimator used initially was Decision Tree, but the result used was Random Forest because it produced a better result by 2% increase. `base = RandomForestClassifier(max_features=5, max_depth=6)`
	learning_rate=0.01 changes did not make a significant difference.
	`AdaBoostClassifier(base_estimator=RF, n_estimators=n_trees, learning_rate=0.01, random_state=42, algorithm="SAMME"`
	no k-fold cross validation used.

The **AdaBoost** classifier was used as the benchmark for this solution and performed only slightly better than the GradientBoosting model at 93.71% accuracy on the testing data. The Adaboost model training time was 1.7x slower than GBC at 138 seconds and was 13x slower than the LR Benchmark. From the confusion matrix below, we achieved fairly good results across the normal classes, it did better than GradientBoosting on the anomaly of the SIT_TO_STAND activity signal data.

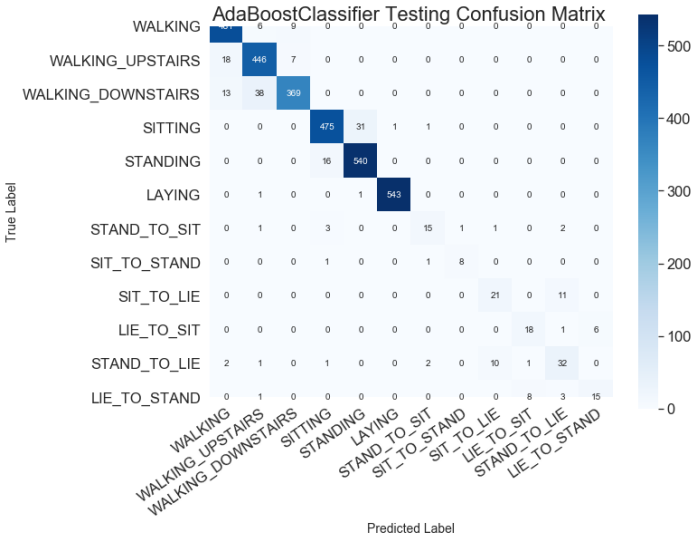


Figure 19 ABC Confusion Matrix

Stacking

We use stacking to determine whether we can get an improvement in performance compared to the standalone models. In stacking, we first input the training data into several classifiers, i.e. C1, C2, C3, etc. Next we perform Level one predictions, i.e. P1, P2, P3), then we could extend to a Level 2 prediction, by taking the Level 1 predictions from the k-fold cross-validation from each of the standalone models (level one predictions, i.e. P1, P2, P3) and append them to the original training dataset. We then train on this original feature plus layer one predictions dataset using k-fold cross-validation, resulting in layer two (Level 1 and Level 2) predictions in this "Meta-Classifier". And from this it will be evaluated to see if we have an improvement over any of the standalone models. This is a graphical depiction of the simple model of stacking using only a Level 1 prediction. In this project, we will use this simple version of the stacking model, and did not use k-fold validation in this project, so it would be a great refinement to implement k-fold cross-validation and it's likely improvement over the LR benchmark could be obtained, but will cost lot more than 200 seconds.

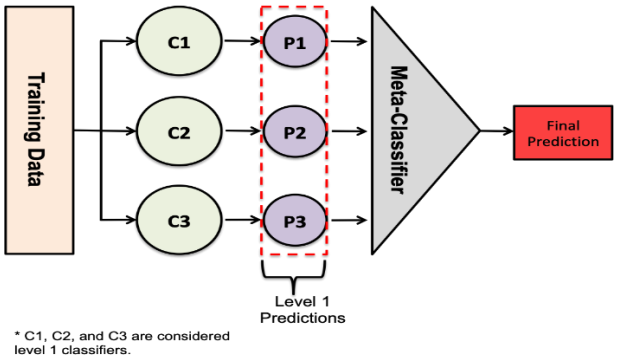


Figure 20 Stacking Model Overview

Stacking model Meta-Classifier	`estimators = [('LR', LRclassifier), ('GBC', GBclassifier), ('rf', rf), ('ABclassifier', ABclassifier) ]`
	`VotingClassifier(estimators, voting='soft', n_jobs=2)`

	no k-fold cross validation used.

The **Stacking model Meta-Classifier *performed 2nd best overall, and better than the LR benchmark*** at 94.8% accuracy on the testing data, and the R2 at 0.95 and the MCC of 0.9.39 was slightly better than the LR benchmark results. From the confusion matrix below, we achieved fairly good results across the normal classes. The training time was 20x slower than the LR Benchmark. From the confusion matrix below, we achieved fairly good results across the normal classes, it did poor on the SIT_TO_STAND activity signal data.

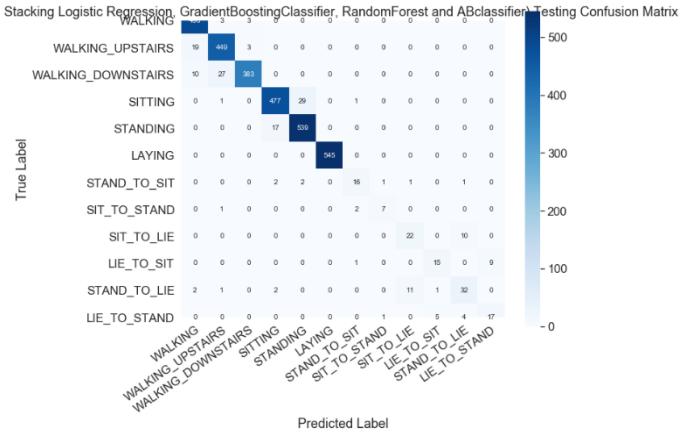


Figure 21 Stacking Confusion Matrix

Table 4 Stacking Classification report

classification_report	precision	recall	f1-score	support
WALKING	0.94	0.99	0.96	496
WALKING_UPSTAIRS	0.94	0.95	0.95	471
WALKING_DOWNSTAIRS	0.98	0.92	0.95	420
SITTING	0.96	0.94	0.95	508
STANDING	0.95	0.97	0.96	556
LAYING	1.00	1.00	1.00	545
STAND_TO_SIT	0.80	0.70	0.74	23
SIT_TO_STAND	0.78	0.70	0.74	10
SIT_TO_LIE	0.67	0.69	0.68	32
LIE_TO_SIT	0.71	0.60	0.65	25
STAND_TO_LIE	0.69	0.67	0.68	49
LIE_TO_STAND	0.65	0.63	0.64	27
accuracy			0.95	3162
macro avg	0.84	0.81	0.82	3162
weighted avg	0.95	0.95	0.95	3162

2.2.7 **Neural Networks**

Finally, a basic artificial neural network (ANN) approach specifically adjusted for the best performance for this classification problem was approached. The model summary is listed and described below.

Artificial neural networks are based on the idea of modeling problems with high-level abstractions in data by using multiple processing layers that basically perform multiple non-linear transformations. [13]

Given the accuracy on testing data is much lower than that on the training data, the algorithms appeared to suffer from overfitting. Applying deep neural network could also help reduce overfitting. Alternatively, some form of regularization, we use a layer called DENSE with "relu" activation, to penalize overfitting which may be helpful to alleviate overfitting.

```
# Using the Sequential Model as a sequence of layers
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(768, input_dim= 561, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(.03),    # dp made improvement in accuracy
  tf.keras.layers.Dense(196, activation='relu'),
  #tf.keras.layers.Dropout(.03),    # dp made improvement in accuracy
  tf.keras.layers.Dense(32, activation='relu'),
  tf.keras.layers.Dense(num_classes, activation='sigmoid')
  # tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

Table 5 Final ANN Model Summary (Sequential Layers)

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 768)               431616
_________________________________________________________________
dense_7 (Dense)              (None, 128)               98432
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
dense_8 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 196)               25284
_________________________________________________________________
dropout_3 (Dropout)          (None, 196)               0
_________________________________________________________________
dense_10 (Dense)             (None, 32)                6304
_________________________________________________________________
dense_11 (Dense)             (None, 12)                396
=================================================================
Total params: 579,056
Trainable params: 578,800
Non-trainable params: 256
```

Table 6 All changes made from these made the accuracy worst.

ANN Model 1	Optimizer: Adam with LR of 0.0005 (tried rmsprop, but did not improve)
	Batch Size: 16 / Epochs 100
	Network Depth: Note above Table 5
	Activation: Sigmoid (tried softmax, but did not improve)
	Dropout: One layer at 0.03 (second layer made acc go down)
	Validation with Test dataset used.

The ANN Model 1 was the middle ranking in performance of accuracy with a score fo 92.995 %. The training time was 65x slower than the LR Benchmark. The Macro average of 0.74 was poor., activities STAND_TO_LIE, and LIE_TO_STAND had low precision scores at .53 and .49.

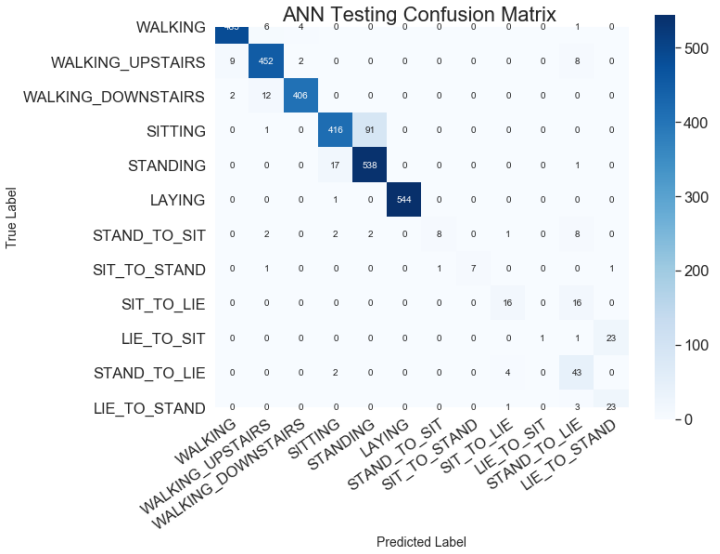


Figure 22 ANN Confusion Matrix

Table 7 ANN Classification Report

classification_report	precision	recall	f1-score	support
WALKING	0.98	0.98	0.98	496
WALKING_UPSTAIRS	0.95	0.96	0.96	471
WALKING_DOWNSTAIRS	0.99	0.97	0.98	420
SITTING	0.95	0.82	0.88	508
STANDING	0.85	0.97	0.91	556
LAYING	1.00	1.00	1.00	545
STAND_TO_SIT	0.89	0.35	0.50	23
SIT_TO_STAND	1.00	0.70	0.82	10
SIT_TO_LIE	0.73	0.50	0.59	32
LIE_TO_SIT	1.00	0.04	0.08	25
STAND_TO_LIE	0.53	0.88	0.66	49
LIE_TO_STAND	0.49	0.85	0.62	27
accuracy			0.93	3162
macro avg	0.86	0.75	0.75	3162
weighted avg	0.94	0.93	0.93	3162

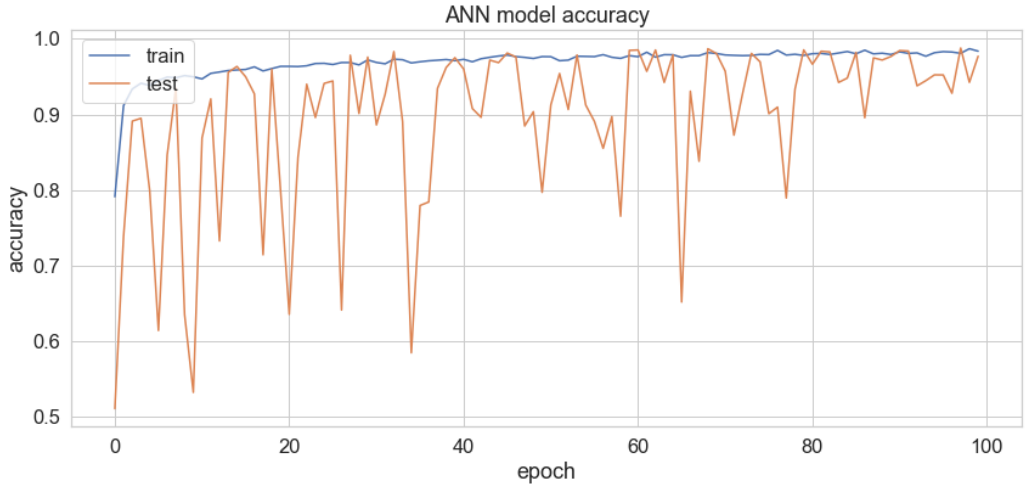


Figure 23 ANN Model Accuracy Graph

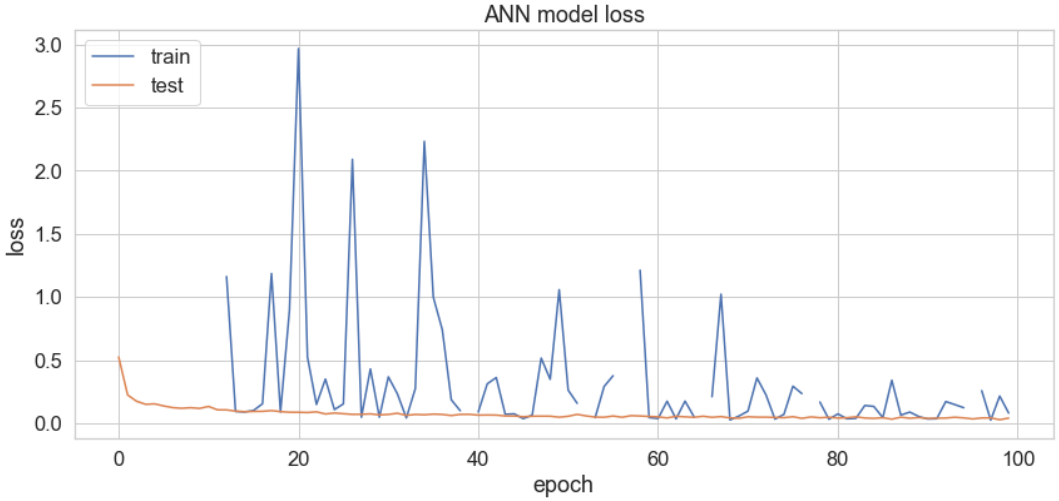


Figure 24 ANN Model Loss Graph

2.3 Benchmark

The benchmark model that will be used for measuring performance will be a logistic regression model for a classification to develop a baseline activity recognition model. The reasoning behind using this benchmark because LR is a good candidate because our problem consists of doing binary classification with clean data, which makes for good conditions for the logistic regression model. *Also, an important reason to start with it for this project and to use the most basic model and without k-fold validation is training time, in that our ultimate objective is to classify any human activity in near real-time.* LR is the most basic classification algorithm, so it's a good ground floor to start with. It's fast, so we can evaluate the best options, i.e. changes to features, or thresholds quickly. Logistic regression would provide great flexibility should we get additional data or decide to experiment with different thresholds, i.e., different type of sensors providing.

The benchmark model will then be compared to the different standalone classifiers/models, and then to the ensemble, then the stacking, and then the ANN model.

We will build our benchmark model using Logistic Regression from Scikit-learn package. This function implements logistic regression and can use different numerical optimizers to find parameters, including 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' solvers. The version of Logistic Regression in Scikit-learn, supports regularization. Regularization is a technique used to solve the overfitting problem in machine learning models. Regularization helps address overfitting by penalizing complexity. The regularization constant is known as "C" a positive floating-point number, is the regularization strength, and for our benchmark we will stay with the default C of 1.0.

It was found that the `Best results obtained with 'lbfgs' solver`

```
LRclassifier = LogisticRegression(random_state = 0, solver= 'lbfgs'
```

3 Methodology

3.1 Data Preprocessing

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as preprocessing. Fortunately, for this dataset, there are no invalid or missing entries we must deal with, however, this is some formatting of the dataset required for some of the models and there are some qualities about certain features that must be adjusted. Reformatting that data based on refinement optional ideas presented for future work would help tremendously with the outcome and predictive power of nearly all learning algorithms. The preprocessing steps were as follows:

Data Preparation and Exploration

Explored and established a basic understanding of the dataset, performed basic cleaning and processing, transform the data into examples and explore the data to show the distribution information of the dataset.

a) Loading the data, transformations of the data, data exploration and data reformatting.
b) Perform Data Cleaning.

c) Split the dataset into a Training and Testing set.
d) Collect Statistics about the dataset.
e) Display the dataset to view for correctness and format.

Visualization of Feature Distribution

Sets of plots were created to examine the data

f) Plot Number of Samples per Activity
g) Plot Samples of Activities provided by each Test Subject
h) Plot of 3 Axis Body Acceleration Mean Values with Respect to Frequency vs the Activity Type
i) Plot Visual exploration of the data in 3 dimensions with principal component analysis
j) Plot Explained Variance by No. of Principal Components.

Features Engineering

k) Develop concepts about new features could be developed based on the given features.
l) Determine Approaches to Modeling the Problem

ANN Model Preprocessing Steps Required:

m) Removed Subject and Activities column from the dataset, subject info not needed for basic ANN training, and Activities get converted to one-hot encoded.
n) Categorical labels encoded using by using a technique called One-hot Encoding
o) Convert the Training and Test set data to NumPy arrays.
p) Set the set the input shape to a one-dimensional array.

3.2 Implementation

The goal is to implement models that demonstrate a comparison between the traditional classification models to a basic ANN and understanding any improvements in terms of computational costs while maintaining similar accuracy. We implement these models using primary using the following commonly available libraries that we import and include NumPy's, pandas, mathplotlib, and sklean.

3.2.1 We use the following libraries and examples to implement the data manipulation, plotting, the model fitting and training, and evaluation.
NumPy and pandas: NumPy enables us to work with arrays with great efficiency. Pandas is used to read the dataset file and import it as a dataframe, which is like a table with rows and columns.

```
# Panda csv function to import The Training dataset

X_train = pd.read_csv('./data/HAPT Data Set/Train/X_train.txt', sep='\s+', header=None)

#converting training data into numpy array

from numpy import array
```

```
aX_train=X_train.values
```

matplotlib: matplotlib is a highly customizable package which has the sub package pyplot that enables us to draw plots, bar charts, pie charts and more. We get options to add legends, axis titles, change thickness of lines etc. The cm package (colormap) allows us to get colors for our charts.

sklearn: This machine learning library includes numerous machine learning algorithms already built-in with certain parameters set as default parameters, so they work right out of the box.

For example:

```
# Fitting Logistic Regression to the Training set

from sklearn.linear_model import LogisticRegression

# print classification report of Precision-Recall, f1-score accuracy scores

from sklearn.metrics import confusion_matrix, classification_report

from sklearn.metrics import classification_report, accuracy_score
```

3.2.2 Creating a Training and Predicting Pipeline for the learning algorithms

To properly evaluate the performance each model chosen, we created a training and predicting pipeline that allows to quick and effective training models using various sizes of training data and perform predictions on the testing data. The basics steps to implement the learning classifiers are as follows:

a. **INPUT Data for Training:** The input data for all sci-learn learning algorithms except for the ANN Neural Network is the **X_train** data as shown in the example in figure 2. Which is the csv panda import of the data from the HAR dataset. No manipulations were done on the final training runs. Scaling was not used, it is left in the jypter notebook and used for earlier trial runs, but not used in the final.

b. **INPUT Data for Testing Predictions**: Likewise, to the training data, the input data for all sci-learn learning algorithms except for the ANN Neural Network is the **X_test** data as shown in the example in figure 2.

c. Import the libraries like this

```
from sklearn.linear_model import LogisticRegression
```

d. configure the algorithm for the learning classifier

```
LogisticRegression(random_state = 0, solver= 'lbfgs')
```

e. Fit the learner to the sampled training data and record the training time.

```
LRresults = LRclassifier.fit(X_train, y_train)
```

f. Perform predictions on the test data X_test, and on the training points X_train.

```
# Predicting the Training set results
```

```
LR_pred_train = LRclassifier.predict(X_train)

# Predicting the Test set results

y_pred = LRclassifier.predict(X_test)
```

In this way we validate the training was performed adequately and is 100% or 99% on all except k-neighbors training.

g. Record the total prediction time.

h. Calculate the accuracy score for both the training subset and testing set.

i. **Calculating the Confusion Matrix, the classification** report was written as a function builds a text report showing the main classification metrics. Precision is the ability of the classifier not to label as positive a sample that is negative and recall is the ability of the classifier to find all the positive sample. [6] Matplotlib and sns.heatmap is used to layout the confusion matrix.

A Normalization of the confusion matrix is also performed, like this:
`normalized_confusion_matrix = np.array(cm, dtype=np.float32)/np.sum(cm)*100`

Calculate the Precision, Recall, F1 score for the testing set. For each individual run of all the experiments conducted, accuracy, precision, recall and F1, R2 and MCC scores are collected. The accuracy metric is simply the percentage of correctly classified instances. Although the dataset used for this work is not perfectly balanced, because we are working with twelve classes, we consider that average accuracy metric to be both a simple and appropriate metric for representing the classifiers' overall performances. For this reason, and to aid readability, we summarized only the accuracy metric plus variance for the preliminary evaluation. Of course, we also analyzed the confusion matrix, precision, recall, F1. Because this is a multiclass problem, these metrics are obtained by weighted averaging.

j. Calculate the R2 Score

```
# The r2_score function

from sklearn.metrics import r2_score LRr2 = r2_score(y_test, y_pred) print
("r2 score: %.4f" %(LRr2))
```

k. Calculate the Mathews correlation coefficient.

```
# "The Matthews correlation coefficient

from sklearn.metrics import matthews_corrcoef LR_mc = matthews_corrcoef(y_test,
y_pred) print('matthews_corrcoef: ', LR_mc) print ("matthews_corrcoef: %.4f"
%(LR_mc))
```

3.2.3 Creating a Training and Predicting Pipeline for the artificial neural network

For the ANN, the following differences was used to pre-process the input data, One-hot-encode, convert to Numpy arrays, and set one dimensional.

l. **ONE-HOT-ENCODE**: For the y_train and y_test labeling data. From the table in Exploring the Data for the activities, we can see there are several features for each record that are non-numeric. Typically, learning algorithms expect input to be numeric, which requires that non-

numeric features (called categorical variables) be converted. One popular way to convert categorical variables is by using the one-hot encoding scheme. One-hot encoding creates a "dummy" variable for each possible category of each non-numeric feature. For example, assume someFeature has three possible entries: A, B, or C. We then encode this feature into someFeature_A, someFeature_B and someFeature_C.

| | someFeature | | someFeature_A | someFeature_B | someFeature_C | | :-: | :-: | | :-: | :-: | :-: | | 0 | B | | 0 | 1 | 0 | | 1 | C | ---> one-hot encode ----> | 0 | 0 | 1 | | 2 | A | | 1 | 0 | 0 |

```
# one hot encode implementation do for both Training and Testing label data
# reference: https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/
from sklearn.preprocessing import OneHotEncoder

onehotencoder = OneHotEncoder()

one_hot_enc_y_train = onehotencoder.fit_transform(y_train).toarray()

one_hot_enc_y_test = onehotencoder.fit_transform(y_test).toarray()
```

m. convert to Numpy arrays

```
from numpy import array

aX_test = X_test.values #converting test data into array
```

n. Set INPUT SHAPE to one dimensional.

```
# set the input shape

input_dim = 561,
```

3.3 Refinement

The solution to our problem will be summarized in the results section, but here we discuss the process of improvements made upon the algorithms and techniques used in the implementation.

Scaling of the datasets was looked at earlier in the project in the LR benchmark algorithm, but not included in the final training results because a brief trial of scaling in the benchmark did not improve the results. It can be recommended to implement scaling, now looking back on the visualization and analysis of the results, scaling implemented taking into some of the considerations noted in the data exploration and results section of this report. ***Logistic Regression refinement – the best solver= 'lbfgs' made a percentage point improvements.*** LR implements logistic regression and can use different numerical optimizers to find parameters, including 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' solvers.

The ***initial highest performing model*** used was LinearSVC Model because it achieved about the same accuracy as the LR Benchmark at of LSVC at 94.62% vs. 94.66% LR. So LinearSVC was an excellent model initially, and since refinements can be made to the all the advanced models, this became the new model to improve upon that was likely to be capable of achieving better results than the LR benchmark.

LinearSVC with Gridsearch refinement: Since the LinearSVC was an excellent model it was chosen to do *an improvement of it by add "Grid Search" method*. Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Grid-searching can be applied across any machine learning model to calculate the best parameters to use for any given model. Our LinearSVC with Grid search with a linear kernel performed the best with 95.2% test accuracy using with linear kernel parameter, on second try by replacing the 'poly' kernel with linear. A gridsearch did lead to a significant improvement to the SVC model. The R2 score shows the highest so far of 0.971, quite a strong performance. Result is hard to beat. The LinearSVC with Grid search Model training time was 11x slower than the LR Benchmark.

***AdaBoost Classifier**, Using LR=0.01, and algorithm change from the default SAMME to SAMMR, with the base-estimator the Decision Tree Classifier.* The SAMME.R real boosting algorithm typically converges faster than SAMME, achieving a lower test error with fewer boosting iterations.

The base-estimator Classifiers

base = DecisionTreeClassifier(max_features=5, max_depth=6)

base = RandomForestClassifier(max_features=5, max_depth=6)

Two algorithms can be used here: SAMME and SAMME.R

```
ABclassifier = AdaBoostClassifier(base_estimator=base, n_estimators=n_trees,

                    learning_rate=0.01, random_state=42, algorithm="SAMME")
```

The **GradientBoosting classifier** performed very well after refinement noted by adjusting the learning rate at 93.23% accuracy on the testing data. learning_rate=0.1 (performed runs with values of .01, .2 and accuracy went down from 93.2% to 91.1%, and 92.6%

Adding k-fold cross validation would make improvements over the models used but will add training time.

ANN improvements

All the parameters listed in Table 6 were adjusted but arrived at the final values by adjusting and running values slightly higher and lower than the final. The accuracy varied as low as 82%.

So, the basic ANN, improperly set with hyperparameters lead to very wild swings of the results, such as when you run the training under 60 epochs, or when the weights get randomly set wrong, you will occasionally get unpredictable results came out below 80% as you can see in the "Artificial Neural Network Model Accuracy= vs. epoch results" chart. For the data method i.e. class imbalance used on the input to this basic ANN model, you need to use at least 90 epochs, however, I did get unpredictable results at 120 and over epochs, so found it was best at 100 epochs.

4 Results

4.1 Model Evaluation and Validation

The winner with best results of the **Linear with GridSearch SVC** test accuracy was **95.2%,** so this model does very well with unseen data, and the best R2 and MCC scores of any model by a fair margin, it is considered as appropriate solution to address the HAR problem adequately. The weighted average scores were: Precision 95%, Recall 95%, F1-Score 95%, and its training time was the very lowest of all models at 7.4 seconds. The training accuracy final score was 99.3. The macro avg, in the classification report below in figure was 0.86 which was the best score of all the models, except the benchmark LR did slightly better at 0.88. The improvement of the macro average over its non-refined model LinearSVC was a difference from 0.82 vs. 0.86, *so the grid search improved the ability to recognize the postural activities!*

Table 8 Test Accuracy Results Sorted for all 8 Models.

Classifier	Test Accuracy
Linear GridSearchSCV	95.193%
Stacking	94.782%
LogisticRegression	94.655%
LinearSVC	94.624%
AdaboostClassifier	93.707%
GradientBoostClassifier	93.201%
ANN Model	92.948%
RandomForestClassifier	90.923%
kn-Neighbors	88.899%
Average Test Accuracy %	***93.215%***

Table 9 Results of the Classifiers/Models by Training, Accuracy, R2, MCC

Classifier	Training Time (Seconds)	Test Accuracy	R2 Score	MCC score
LogisticRegression	10.84	0.94655	0.939	0.937
AdaboostClassifier	137.88	0.93707	0.94	0.926
GradientBoostClassifier	78.82	0.93201	0.926	0.92
kn-Neighbors	229.98	0.88899	0.895	0.87
RandomForestClassifier	101.75	0.90923	0.932	0.893
LinearSVC	7.41	0.94624	0.931	0.937
Linear GridSearchSVC	**110.64**	**0.95193**	**0.971**	**0.943**
Stacking	206.11	0.94782	0.95	0.939
ANN Neural Network Model	656.05	0.92948	0.905	0.918
Averages of each:	171	93.21 %	0.932	0.920

A gridsearch refinement to the model lead to a significant improvement to the model, but the training time was 11x slower than our LR benchmark, but also half the time at 110 seconds of the second best performing model the Stacking meta-classifier took 206 seconds to train. Note Table 9 for a list of all the training times for all models. So, at 110 seconds, that's equal to the average time out of all the learning in the results except the Neural Network model (656 seconds).

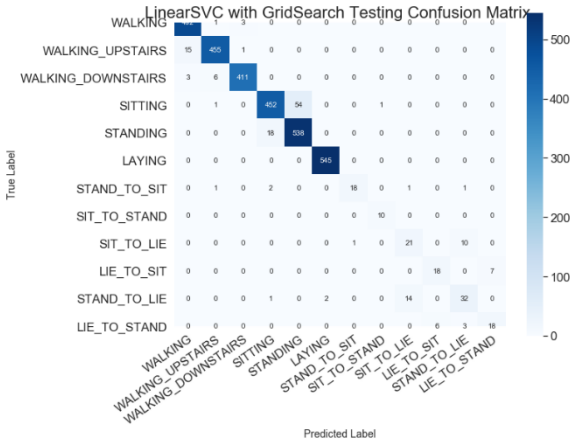


Figure 25 Best Results: linearSVC with Gridsearch kernel': ['linear', 'rbf', 'sigmoid'] confusion matrix

Classification_report				
	precision	recall	f1-score	support
WALKING	0.96	0.99	0.98	496
WALKING_UPSTAIRS	0.98	0.97	0.97	471
WALKING_DOWNSTAIRS	0.99	0.98	0.98	420
SITTING	0.96	0.89	0.92	508
STANDING	0.91	0.97	0.94	556
LAYING	1.00	1.00	1.00	545
STAND_TO_SIT	0.95	0.78	0.86	23
SIT_TO_STAND	0.91	1.00	0.95	10
SIT_TO_LIE	0.58	0.66	0.62	32
LIE_TO_SIT	0.75	0.72	0.73	25
STAND_TO_LIE	0.70	0.65	0.67	49
LIE_TO_STAND	0.72	0.67	0.69	27
accuracy			0.95	3162
macro avg	0.87	0.86	0.86	3162
weighted avg	0.95	0.95	0.95	3162

Table 10 Classification Report for LinearSVC with Gridsearch

Changes made to the LR, SVM, and ensemble learner classifiers typically only made small % changes to the accuracy results, such as a difference between 94% to 93%, or 90%. For adjustment tests on the distance learning classifiers, results above the 89% were not obtainable. For Stacking, the trials of the different models varied down to worst case in the 85%. For the basic ANN, improperly set with hyperparameters lead to very wild swings of the results, such as when you run the training under 60 epochs, or when the weights get randomly set wrong, you will occasionally get unpredictable results came out below 80% as you can see in the "Artificial Neural Network Model Accuracy= vs. epoch results" chart. For the data method i.e. class imbalance used on the input to this basic ANN model, you need to use at least 90 epochs, however, I did get unpredictable results at 120 and over epochs.

Due to class imbalance presented in the data, a robust methodology is required to examine accuracy together with a confusion matrix to understand where the errors were coming from.

We have success in finding optimizing a model in which is higher than our benchmark with, the final highest accuracy is of 95% using the Linear SVC with grid search, and with an average training time of all the 8 models. We can see that the ANN can peak to values above the training average at some moments during the training, depending on how the neural network's weights got initialized at the start of the training epoch, randomly.

4.2 Justification

The final results of the Linear with GridSearch SVC test accuracy was 95.2%, while the LR benchmark was 94.65% than 1% better, but the Gridsearch R2 score at 0.971 and MCC score at 0.943 both showed an increase over the benchmark and are measures of closeness of true positives and accuracy. So, I believe this justifies LGSearch as a best performing learner. The training time however is a factor of 10X for Gridsearch vs. the LogisticRegression, so the solution does depend on how fast the training is required.

5 Conclusion

5.1 Free-Form Visualization

As a summary analysis of the performance of all the models here is a visualization of the Classifiers performance shown by Test Accuracy, vs. R2 Score, vs. MCC Score in Figure 26. Test Accuracy displayed at 0-1 instead of Percentages (%) to show comparison trace line to R2 and MCC scores. Our highest performing L GridsearchSVC can be seen to come in nearly on the average for the R2 and MCC against the other models, and on figure 27 Training Time shows it the lowest, so it's an excellent fast model. I have overlayed the two visualizations in figure 28 to illustrate the performance vs. training time. Our chosen benchmark does hold its own on the left being midway in performance and second to lowest in training time. So, this visualization provides a way to suggest next steps for refinement.

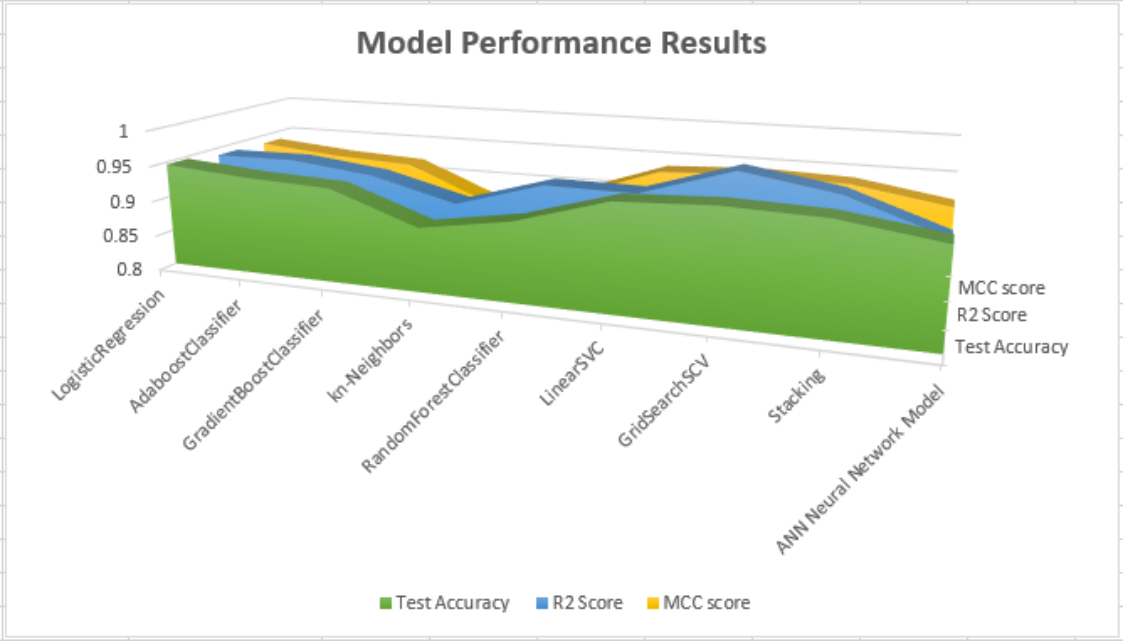


Figure 26 Classifier Performance Visualization

Figure 27 Classifier Training Time (Seconds)

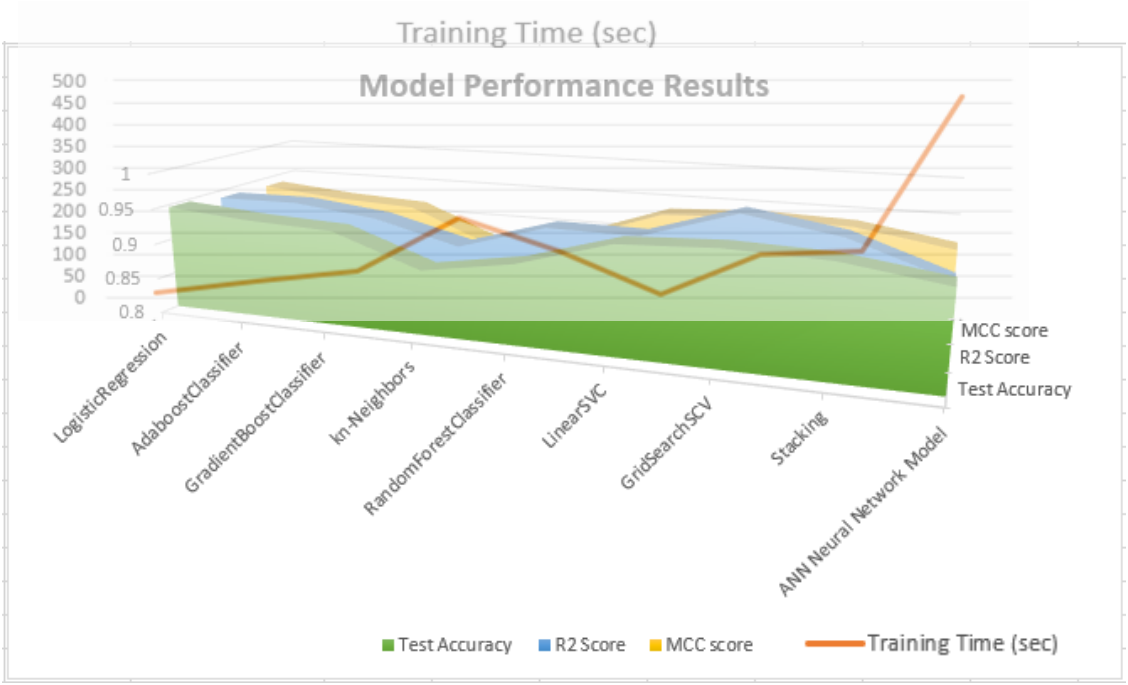


Figure 28 A Visual Overlay of Performance vs. Training Time Results of the Final models

5.2 Reflection

This project helped give me a great understanding of the learning algorithms and the basic artificial neural network, and the coding required for it. There is quite a bit of further coding optimization that can be done to optimize the pipeline to get the models displaying information that provides guidance on ways to tune for better performance.

The most difficult aspect of the project was the determination and implementation of the preprocessing steps required for the Neural Network. But with much research and experimentation, a solution was found for the basic implementation. I found that Quora, StackOverflow, and other StackExchange sites, and of course the Udacity Mentor are good places for asking questions and finding answers.

Much work was put in by the researchers to preprocess the dataset and by providing labeled data, enabling the work for the study for this project to focus purely on applying machine learning techniques and focusing on improving the model performance. However, such well treated dataset is rare in practice.

In designing a suitable neural network structure, there are multiple combinations of hyperparameters, and different types of neural networks layers can be further applied for evaluation. But for this project, several different types of neural networks can be developed to make much better models to handle this type of data to reach toward the goal of being able to recognize a wide variety of human activities. This include Convolutional Neural Networks (CNN's), and Long-Term-Short-Term-Memory (LSTM) which are Recurrent Neural Networks. See [14] for example.

5.3 Improvement

Since Linear GridsearchSVC performed overall the best in this project, I would recommend starting further work with this model and further fine-tune the grid search function to further improve the model performance. Suggest breaking the input dataset according to methods in the data exploration process determined could provide for more balance among input groups. Can it be optimized to run faster, but with the same results? Scaling was not used in these results, but evaluation of the best way to do scaling of the data should be done next. Use of the raw dataset which can be found at [1] could also be explored to see whether the preprocessing procedures have significant impact on model performance.

In addition, given the errors were predominantly present in transitioning activities, partitioning out the static and non-static (postural transitions) activities may be helpful to eliminating the class imbalances during classification training. If no significant improvement were found after the separating out the static and non-static activities, it might mean further sensor devices or data points maybe required to delineate the different transitioning activities from each other.

Improvements can be made to each model, for example for our benchmark, Logistic Regression, k-fold cross-validation can be applied which will split training dataset up and evaluate the performance on slides one at a time. It will take a longer time to train. See reference [6] for an example how to implement.

Improvements over Gradient Boosting Classifier would be applying the popular XGBoost and LightGBM algorithms and implementation examples can also be found in reference [6]. These two models are much stronger than the others, random forest, and logistic regression [6], and either one could be used in stacking with other models, and then also add k-fold cross validation. However, if the goal is training time, this will increase the training time.

Next, I would suggest CNNs, and LSTM RNN's to evaluated for be use for the Artificial Neural Network, because I do think you can see improvement over the basic ANN if special pre-processing is performed on the input data signals. The types of signals from the activities data as shown in figure 6 plot of body axis acceleration are similar in a way with time and frequency domain to speech, etc., and LSTM's are used in state of the art deep learning applications like speech recognition, speech synthesis, natural language understanding, etc. So, the applying their techniques to HAR could prove useful. [14]

References

[1] Jorge-Luis -Ortiz, Luca Oneto, Alessandro Ghio, Albert SamÃ¡, Davide Anguita and Xavier Parra. Human Activity Recognition on Smartphones with Awareness of Basic Activities and Postural Transitions. http://archive.ics.uci.edu/ml/datasets/Smartphone-Based+Recognition+of+Human+Activities+and+Postural+Transitions, Artificial Neural Networks and Machine Learning â€" ICANN 2014. Lecture Notes in Computer Science. Springer. 2014.

[2] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition Using Smartphones. http://archive.ics.uci.edu/ml/datasets/Smartphone-Based+Recognition+of+Human+Activities+and+Postural+Transitions, 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Reyes Learning, ESANN 2013. Bruges, Belgium 24-26 April 2013.

[2] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, Lisha Hu, Deep Learning for Sensor-based Activity Recognition: A Survey, https://arxiv.org/abs/1707.03502, arXiv:1707.03502v2 [cs.CV], 14 Dec 2017

[3] Mohammad Abu Alsheikh and Ahmed Selim and Dusit Niyato Linda Doyle and Shaowei Lin and Hwee-Pink Tan, Deep Activity Recognition Models with Triaxial Accelerometers, https://arxiv.org/abs/1511.04664, arXiv:1511.04664v2 [cs.LG], 25 Oct 2016 https://arxiv.org/pdf/1511.04664.pdf

[4] Kwapisz J., Weiss G., Samuel A. Moore, S., Activity Recognition using Cell Phone Accelerometers, http://www.cis.fordham.edu/wisdm/includes/files/sensorKDD-2010.pdf, 2010

[5] Calatroni, Alberto; Roggen, Daniel; Tröster, Gerhard, A methodology to use unknown new sensors for activity recognition by leveraging sporadic interactions with primitive sensors and behavioral assumptions, https://doi.org/10.3929/ethz-a-006286309, 2010

Books

[6] Patel, A., "Hands-On Unsupervised Learning Using Python" How to Build Applied Machine Learning Solutions from Unlabeled Data, O'Reilly Media Inc., 3rd Release, May 03, 2019.

Web Resources

[7] "1.1. Generalized Linear Models", 1.1.11. Logistic regression - scikit-learn v0.21.3 Documentation. https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression, N.p., n.d., Web., 11 Nov 2019

[8] "1.11. Ensemble methods", 1.11.2.1. Random Forests - scikit-learn v0.21.3 Documentation. https://scikit-learn.org/stable/modules/ensemble.html, N.p., n.d., Web., 11 Nov 2019

[9] "1.11. Ensemble methods", 1.11.3. AdaBoost - scikit-learn v0.21.3 Documentation. https://scikit-learn.org/stable/modules/ensemble.html, N.p., n.d., Web., 11 Nov 2019

[10] "3.3. Model evaluation: quantifying the quality of predictions" - scikit-learn v0.21.3 Documentation. https://scikit-learn.org/stable/modules/model_evaluation.html, N.p., n.d., Web., 11 Nov 2019

[11] "3.3. Model evaluation: quantifying the quality of predictions", 3.3.2.5. Confusion matrix - scikit-learn v0.21.3 Documentation. https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py, N.p., n.d., Web., 11 Nov 2019

[12] "1.17. Neural network models (supervised) - scikit-learn v0.21.3 Documentation. https://scikit-learn.org/stable/modules/neural_networks_supervised.html, N.p., n.d., Web., 11 Nov 2019

[13] https://www.neuraldesigner.com/learning/examples/activity-recognition https://www.youtube.com/watch?v=XOEN9W05_4A, N.p., n.d., Web., 11 Nov 2019

[14] https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21, Sept 24, 2018

[15] https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab, Nov 3, 2018