Data Mining Assignment

1.0 Introduction

The demand of recognizing human activities have grown basically in the domain of health, basically in taking care of senior assistances and cognitive disorders persons. We can save a lot of resources and time if we are analyzing the activity of any patients or his abnormal behavior through these mobile sensors. Instead of health domain there are so many other applications which are also performing by IoT devices like security issue. But these applications could be performed in surveillance camera. In this scenario we make a camera much advance so that it can define a restricted zone and trace the objects under its focus. These objects can be anything like an animal, human or box etc. and if any stranger or any box remains it's a fixed position for certain time then we will get informed or it will inform security guards. In many studies it is found that the wearable sensors have predicted activity at very low error rate. This project uses only easily available and low cost sensors to recognize the human activity. In today's teens mobile phone is ubiquitous device and its computational quality is much better which make it ideal for non-intrusive body attached sensors. Today about 95% mobile phones have in built sensors like accelerometer and gyroscope. In research found that gyroscope can help in activity recognition, but contribution in alone for it is not as good as accelerometer. Because any Smartphone can easily accessed but gyroscope can't. In our design Smartphone can be placed anywhere around waist such as jacket pocket or pant pocket. Whenever any new activity is added to the system we need to train entire system. Because of variance in sensors, if algorithms run on different device, the parameters of algorithms need to get trained. We propose active learning process to accelerate the training process because labeling a time series data is time consuming process and it is not possible to give label for all the training data. Given a classifier, active learning intelligently queries the unlabeled samples and learns parameters from the correct label. In this manner user do label only the samples that the algorithm asks for the total amount of required training samples reduced. The goal of this project is to make a model on Smartphone that can easily recognize the human activity. Moreover active learning models are developed in order to reduce labeling time and burden. Through testing and comparing with different learning algorithms, we find one best fit model for our system.

Data Summary

The Human Activity Recognition database was built from the recordings of 30 study participants performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors. The objective is to classify activities into one of the six activities performed.
Dataset consists of signals from a smartphone carried by
9 individuals performing 6 different activities. Activities
Performed are listed below with their corresponding codes.

- WALKING - 1
- CLIMBING UP THE STAIRS - 2
- CLIMBING DOWN THE STAIRS - 3
- SITTING - 4
- STANDING - 5
- LAYING – 6

The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

In data set we 102999 rows and 563 attributes we have split this data set into train and test, in training we have 7352 samples with 563 features and in testing we have 2947 rows and 563 columns. As we cannot explain about every feature here but we are explaining some of them

For each record in the dataset the following is provided:

- Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
- Triaxial Angular velocity from the gyroscope.
- A 563-feature vector with time and frequency domain variables.
- Its activity label.
- An identifier of the subject who carried out the experiment.
-

2.0 Features

The features selected for this database come from the accelerometer and gyroscope 3-axial raw signals tAcc-XYZ and tGyro-XYZ. These time domain
Signals (prefix’t’ to denote time) were captured at a constant rate of 50 Hz. Then they were filtered using a median filter and a 3rd order
Low pass Butterworth filter with a corner frequency of 20 Hz to remove noise. Similarly, the acceleration signal was then separated into body
And gravity acceleration signals (tBodyAcc-XYZ and tGravityAcc-XYZ) using another low pass Butterworth filter with a corner frequency of 0.3 Hz.

Subsequently, the body linear acceleration and angular velocity were derived in time to obtain Jerk signals (tBodyAccJerk-XYZ and tBodyGyroJerk-XYZ)
. Also the magnitude of these three-dimensional signals were calculated using the Euclidean norm (tBodyAccMag, tGravityAccMag, tBodyAccJerkMag, tBodyGyroMag, tBodyGyroJerkMag).

Finally a Fast Fourier Transform (FFT) was applied to some of these signals producing fBodyAcc-XYZ, fBodyAccJerk-XYZ, fBodyGyro-XYZ, fBodyAccJerkMag, fBodyGyroMag, fBodyGyroJerkMag. (Note the 'f' to indicate frequency domain signals).

These signals were used to estimate variables of the feature vector for each pattern: '-XYZ' is used to denote 3-axial signals in the X, Y and Z directions.

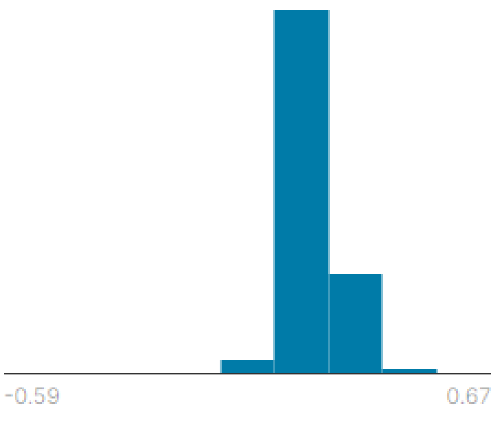
- tBodyAcc-XYZ
- tGravityAcc-XYZ
- tBodyAccJerk-XYZ
- tBodyGyro-XYZ
- tBodyGyroJerk-XYZ
- tBodyAccMag
- tGravityAccMag
- tBodyAccJerkMag
- tBodyGyroMag
- tBodyGyroJerkMag
- fBodyAcc-XYZ
- fBodyAccJerk-XYZ
- fBodyGyro-XYZ
- fBodyAccMag
- fBodyAccJerkMag
- fBodyGyroMag
- fBodyGyroJerkMag

Now I’m attaching some snaps related to features so we can get a better undersating of our data.

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-kurtosis()	angle
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.710304	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	-0.861499	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.760104	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	-0.482845	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	-0.699205	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7347	0.299665	-0.057193	-0.181233	-0.195387	0.039905	0.077078	-0.282301	0.043616	0.060410	0.210795	...	-0.880324	
7348	0.273853	-0.007749	-0.147468	-0.235309	0.004816	0.059280	-0.322552	-0.029456	0.080585	0.117440	...	-0.680744	
7349	0.273387	-0.017011	-0.045022	-0.218218	-0.103822	0.274533	-0.304515	-0.098913	0.332584	0.043999	...	-0.304029	
7350	0.289654	-0.018843	-0.158281	-0.219139	-0.111412	0.268893	-0.310487	-0.068200	0.319473	0.101702	...	-0.344314	
7351	0.351503	-0.012423	-0.203867	-0.269270	-0.087212	0.177404	-0.377404	-0.038678	0.229430	0.269013	...	-0.740738	

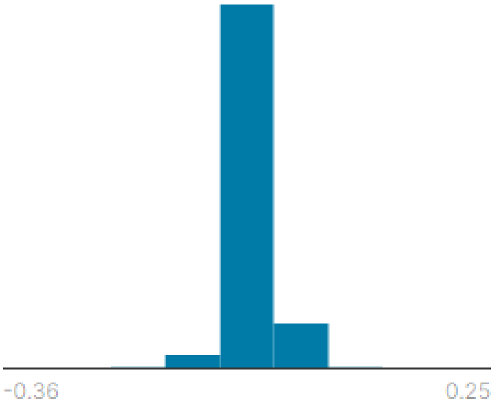
7352 rows × 563 columns

tBodyAcc-mean()-X



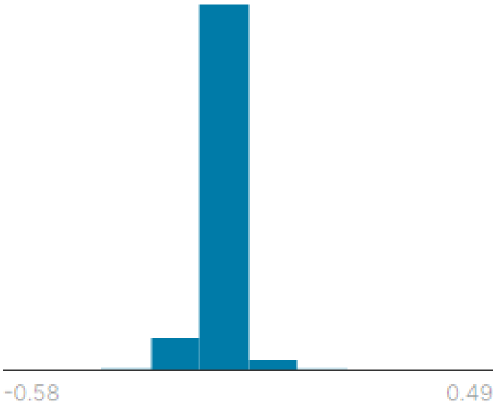
Valid	2947	100%
Mismatched	0	0%
Missing	0	0%
Mean	0.27	
Std. Deviation	0.06	
Quantiles	-0.59	Min
	0.26	25%
	0.28	50%
	0.29	75%
	0.67	Max

tBodyAcc-mean()-Y



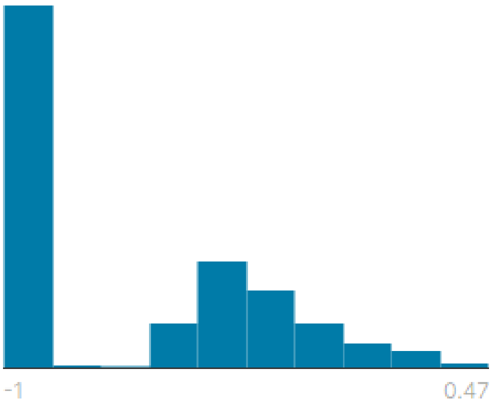
Valid	2947	100%
Mismatched	0	0%
Missing	0	0%
Mean	-0.02	
Std. Deviation	0.03	
Quantiles	-0.36	Min
	-0.02	25%
	-0.02	50%
	-0.01	75%
	0.25	Max

tBodyAcc-mean()-Z



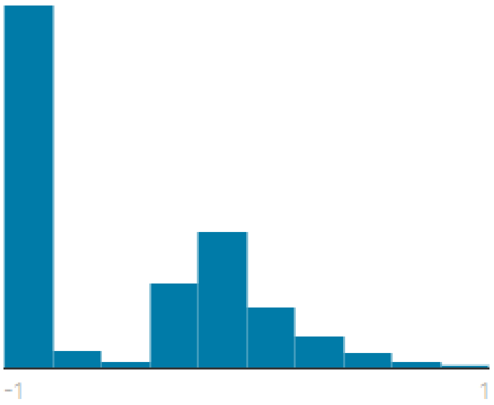
Valid	2947	100%
Mismatched	0	0%
Missing	0	0%
Mean	-0.11	
Std. Deviation	0.04	
Quantiles	-0.58	Min
	-0.12	25%
	-0.11	50%
	-0.1	75%
	0.49	Max

tBodyAcc-std()-X



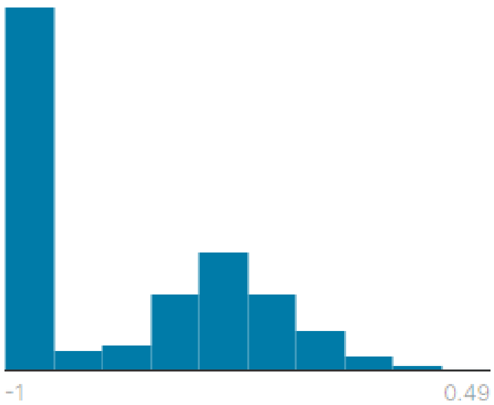
Valid	2947	100%
Mismatched	0	0%
Missing	0	0%
Mean	-0.61	
Std. Deviation	0.41	
Quantiles	-1	Min
	-0.99	25%
	-0.93	50%
	-0.27	75%
	0.47	Max

tBodyAcc-std()-Y



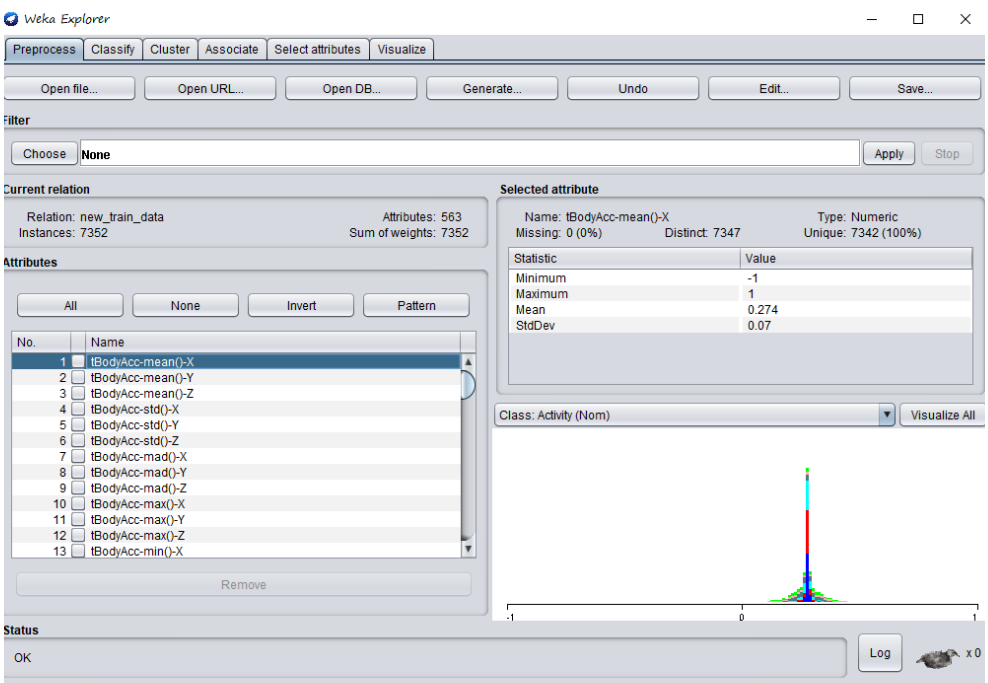
Valid	2947	100%
Mismatched	0	0%
Missing	0	0%
Mean	-0.51	
Std. Deviation	0.49	
Quantiles	-1	Min
	-0.97	25%
	-0.79	50%
	-0.11	75%
	1	Max

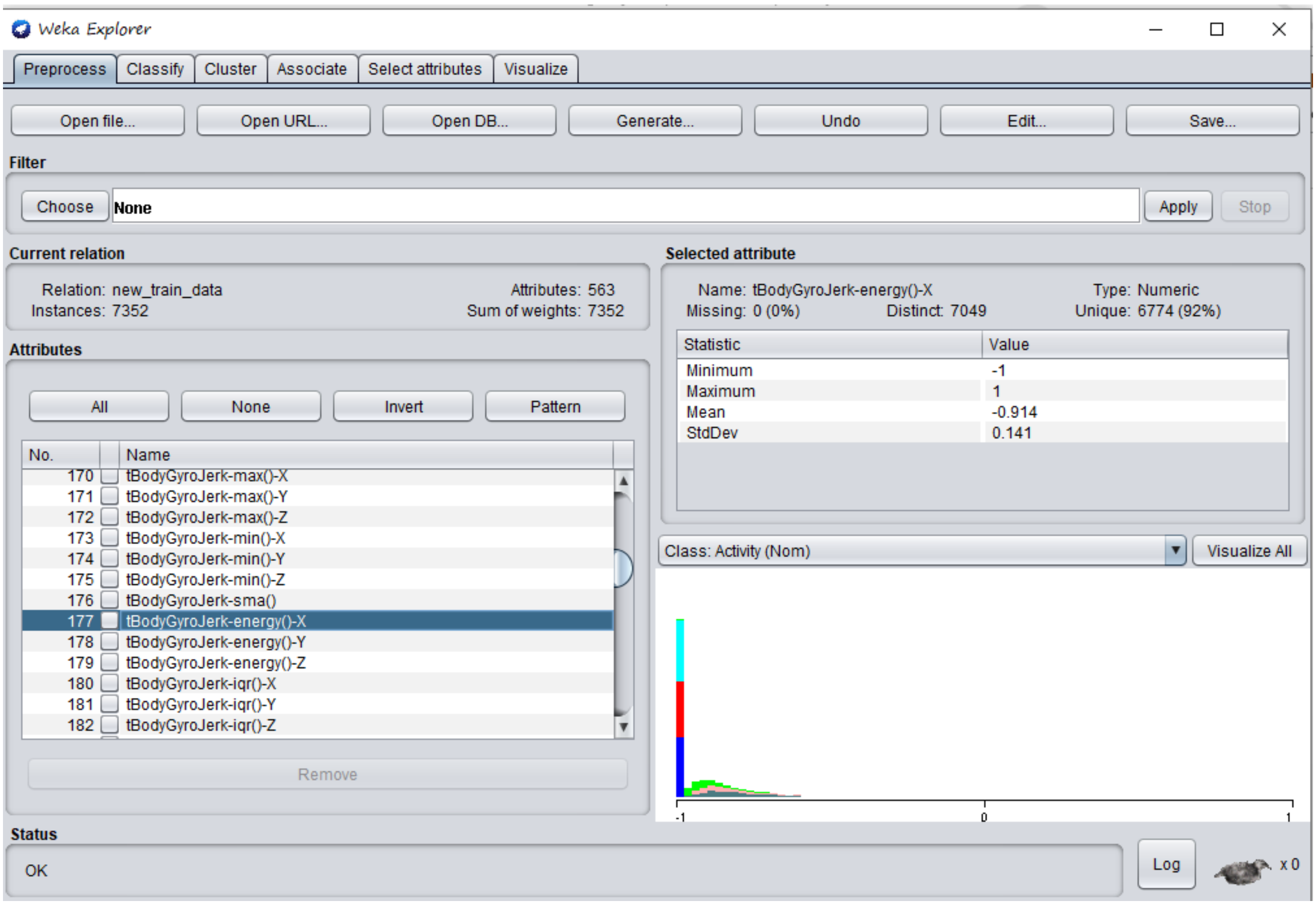
tBodyAcc-std()-Z



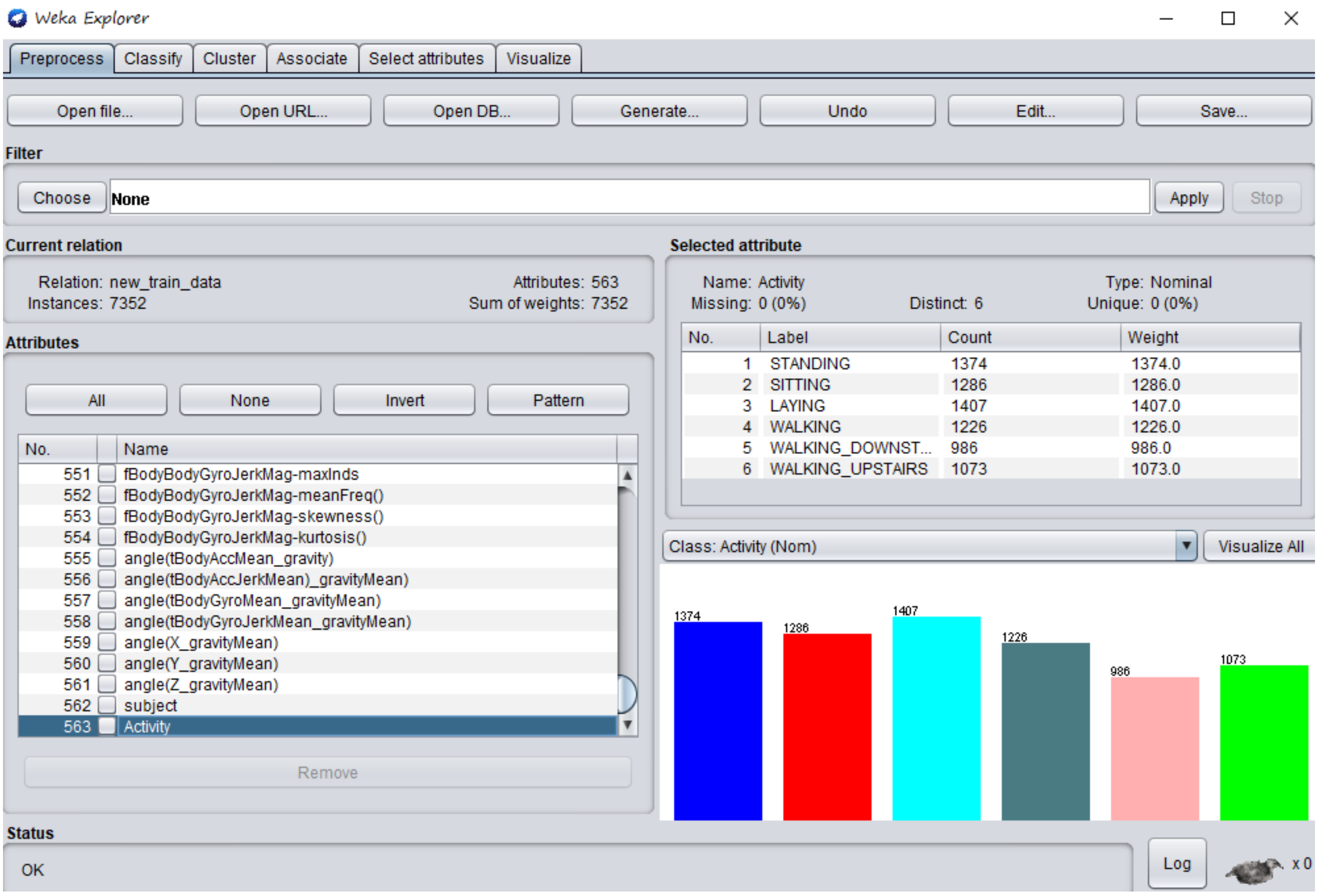
Valid	2947	100%
Mismatched	0	0%
Missing	0	0%
Mean	-0.63	
Std. Deviation	0.36	
Quantiles	-1	Min
	-0.98	25%
	-0.83	50%
	-0.31	75%
	0.49	Max

We have seen some data and statistics related to it lets load data in weka and see some snaps from there.





Let’s see our class column



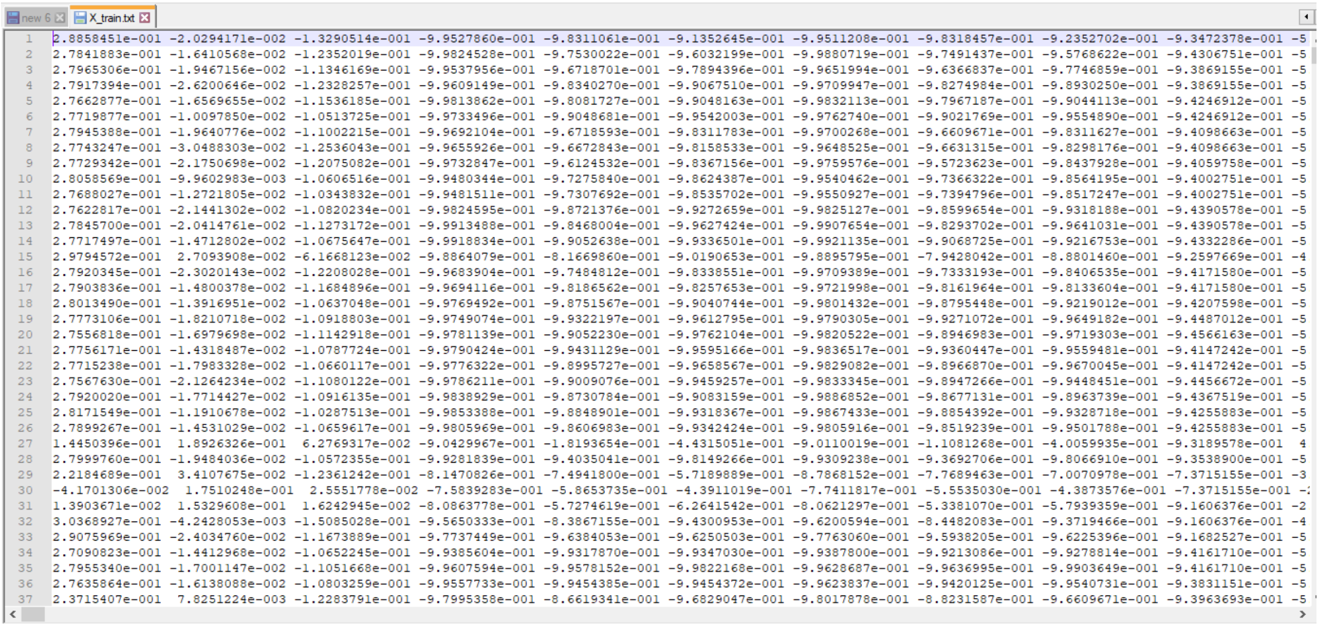
Now we have seen our data now it’s time for preprocessing

3.0 Data Pre Processing

Information preprocessing is a significant advance in the information mining process. The expression "trash in, trash out" is especially relevant to information mining and AI ventures. For our situation we have to do a few information preprocessing before we can prepare our model how about we see bit by bit how we set up our information.

Ok when we download data from UCI repository that data wasn't ready for training a model data was in text files we had a train.txt file test.txt file and in these both files columns names wasn't there columns was in another file feature.txt.

Train file was like this



new 6 | X_train.txt

```
2.8858451e-001 -2.0294171e-002 -1.3290514e-001 -9.9527860e-001 -9.8311061e-001 -9.1352645e-001 -9.9511208e-001 -9.8318457e-001 -9.2352702e-001 -9.3472378e-001 -5
2.7841883e-001 -1.6410568e-002 -1.2352019e-001 -9.9824528e-001 -9.7530022e-001 -9.6032199e-001 -9.9880719e-001 -9.7491437e-001 -9.5768622e-001 -9.4306751e-001 -5
2.7965306e-001 -1.9467156e-002 -1.1346169e-001 -9.9537956e-001 -9.6718701e-001 -9.7894396e-001 -9.9651994e-001 -9.6366837e-001 -9.7746859e-001 -9.3869155e-001 -5
2.7917394e-001 -2.6200646e-002 -1.2328257e-001 -9.9609149e-001 -9.8340270e-001 -9.9067510e-001 -9.9709947e-001 -9.8274984e-001 -9.8930250e-001 -9.3869155e-001 -5
2.7662877e-001 -1.6569655e-002 -1.1536185e-001 -9.9813862e-001 -9.8081727e-001 -9.9048163e-001 -9.9832113e-001 -9.7967187e-001 -9.9044113e-001 -9.4246912e-001 -5
2.7719877e-001 -1.0097850e-002 -1.0513725e-001 -9.9733496e-001 -9.9048681e-001 -9.9542003e-001 -9.9762740e-001 -9.9021769e-001 -9.9554890e-001 -9.4246912e-001 -5
2.7945388e-001 -1.9640776e-002 -1.1002215e-001 -9.9692104e-001 -9.6718593e-001 -9.8311783e-001 -9.9700268e-001 -9.6609671e-001 -9.8311627e-001 -9.4098663e-001 -5
2.7743247e-001 -3.0488303e-002 -1.2536043e-001 -9.9655926e-001 -9.6672843e-001 -9.8158533e-001 -9.9648525e-001 -9.6631315e-001 -9.8298176e-001 -9.4098663e-001 -5
2.7729342e-001 -2.1750698e-002 -1.2075082e-001 -9.9732847e-001 -9.6124532e-001 -9.8367156e-001 -9.9759576e-001 -9.5723623e-001 -9.8437928e-001 -9.4059758e-001 -5
2.8058569e-001 -9.9602983e-003 -1.0606516e-001 -9.9480344e-001 -9.7275840e-001 -9.8624387e-001 -9.9540462e-001 -9.7366322e-001 -9.8564195e-001 -9.4002751e-001 -5
2.7688027e-001 -1.2721805e-002 -1.0343832e-001 -9.9481511e-001 -9.7307692e-001 -9.8535702e-001 -9.9550927e-001 -9.7394796e-001 -9.8517247e-001 -9.4002751e-001 -5
2.7622817e-001 -2.1441302e-002 -1.0820234e-001 -9.9824595e-001 -9.8721376e-001 -9.9272659e-001 -9.9825127e-001 -9.8599654e-001 -9.9318188e-001 -9.4390578e-001 -5
2.7845700e-001 -2.0414761e-002 -1.1273172e-001 -9.9913488e-001 -9.8468004e-001 -9.9627424e-001 -9.9907654e-001 -9.8293702e-001 -9.9641031e-001 -9.4390578e-001 -5
2.7717497e-001 -1.4712802e-002 -1.0675647e-001 -9.9918834e-001 -9.9052638e-001 -9.9336501e-001 -9.9921135e-001 -9.9068725e-001 -9.9216753e-001 -9.4332286e-001 -5
2.9794572e-001  2.7093908e-002 -6.1668123e-002 -9.8864079e-001 -8.1669860e-001 -9.0190653e-001 -9.8895795e-001 -7.9428042e-001 -8.8801460e-001 -9.2597669e-001 -4
2.7920345e-001 -2.3020143e-002 -1.2208028e-001 -9.9683904e-001 -9.7484812e-001 -9.8338551e-001 -9.9709389e-001 -9.7333193e-001 -9.8406535e-001 -9.4171580e-001 -5
2.7903836e-001 -1.4800378e-002 -1.1684896e-001 -9.9694116e-001 -9.8186562e-001 -9.8257653e-001 -9.9721998e-001 -9.8161964e-001 -9.8133604e-001 -9.4171580e-001 -5
2.8013490e-001 -1.3916951e-002 -1.0637048e-001 -9.9769492e-001 -9.8751567e-001 -9.9040744e-001 -9.9801432e-001 -9.8795448e-001 -9.9219012e-001 -9.4207598e-001 -5
2.7773106e-001 -1.8210718e-002 -1.0918803e-001 -9.9749074e-001 -9.9322197e-001 -9.9612795e-001 -9.9790305e-001 -9.9271072e-001 -9.9649182e-001 -9.4487012e-001 -5
2.7556818e-001 -1.6979698e-002 -1.1142918e-001 -9.9781139e-001 -9.9052230e-001 -9.9762104e-001 -9.9820522e-001 -9.8946983e-001 -9.9719303e-001 -9.4566163e-001 -5
2.7756171e-001 -1.4318487e-002 -1.0787724e-001 -9.9790424e-001 -9.9431129e-001 -9.9595166e-001 -9.9836517e-001 -9.9360447e-001 -9.9559481e-001 -9.4147242e-001 -5
2.7715238e-001 -1.7983328e-002 -1.0660117e-001 -9.9776322e-001 -9.8995727e-001 -9.9658567e-001 -9.9829082e-001 -9.8966870e-001 -9.9670045e-001 -9.4147242e-001 -5
2.7567630e-001 -2.1264234e-002 -1.1080122e-001 -9.9786211e-001 -9.9009076e-001 -9.9459257e-001 -9.9833345e-001 -9.8947266e-001 -9.9448451e-001 -9.4456672e-001 -5
2.7920020e-001 -1.7714427e-002 -1.0916135e-001 -9.9838929e-001 -9.8730784e-001 -9.9083159e-001 -9.9886852e-001 -9.8677131e-001 -9.8963739e-001 -9.4367519e-001 -5
2.8171549e-001 -1.1910678e-002 -1.0287513e-001 -9.9853388e-001 -9.8848901e-001 -9.9318367e-001 -9.9867433e-001 -9.8854392e-001 -9.9328718e-001 -9.4255883e-001 -5
2.7899267e-001 -1.4531029e-002 -1.0659617e-001 -9.9805969e-001 -9.8606983e-001 -9.9342424e-001 -9.9805916e-001 -9.8519239e-001 -9.9501788e-001 -9.4255883e-001 -5
1.4450396e-001  1.8926326e-001  6.2769317e-002 -9.0429967e-001 -1.8193654e-001 -4.4315051e-001 -9.0110019e-001 -1.1081268e-001 -4.0059935e-001 -9.3189578e-001  4
2.7999760e-001 -1.9484036e-002 -1.0572355e-001 -9.9281839e-001 -9.4035041e-001 -9.8149266e-001 -9.9309238e-001 -9.3692706e-001 -9.8066910e-001 -9.3538900e-001 -5
2.2184689e-001  3.4107675e-002 -1.2361242e-001 -8.1470826e-001 -7.4941800e-001 -5.7189889e-001 -8.7868152e-001 -7.7689463e-001 -7.0070978e-001 -7.3715155e-001 -3
-4.1701306e-002  1.7510248e-001  2.5551778e-002 -7.5839283e-001 -5.8653735e-001 -4.3911019e-001 -7.7411817e-001 -5.5535030e-001 -4.3873576e-001 -7.3715155e-001 -
1.3903671e-002  1.5329608e-001  1.6242945e-002 -8.0863778e-001 -5.7274619e-001 -6.2641542e-001 -8.0621297e-001 -5.3381070e-001 -5.7939359e-001 -9.1606376e-001 -2
3.0368927e-001 -4.2428053e-003 -1.5085028e-001 -9.5650333e-001 -8.3867155e-001 -9.4300953e-001 -9.6200594e-001 -8.4482083e-001 -9.3719466e-001 -9.1606376e-001 -4
2.9075969e-001 -2.4034760e-002 -1.1673889e-001 -9.7737449e-001 -9.6384053e-001 -9.6250503e-001 -9.7763060e-001 -9.5938205e-001 -9.6225396e-001 -9.1682527e-001 -5
2.7090823e-001 -1.4412968e-002 -1.0652245e-001 -9.9385604e-001 -9.9317870e-001 -9.9347030e-001 -9.9387800e-001 -9.9213086e-001 -9.9278814e-001 -9.4161710e-001 -5
2.7955340e-001 -1.7001147e-002 -1.1051668e-001 -9.9607594e-001 -9.9578152e-001 -9.9822168e-001 -9.9628687e-001 -9.9636995e-001 -9.9903649e-001 -9.4161710e-001 -5
2.7635864e-001 -1.6138088e-002 -1.0803259e-001 -9.9557733e-001 -9.9454385e-001 -9.9454372e-001 -9.9623837e-001 -9.9420125e-001 -9.9540731e-001 -9.3831151e-001 -5
2.3715407e-001  7.8251224e-003 -1.2283791e-001 -9.7995358e-001 -8.6619341e-001 -9.6829047e-001 -9.8017878e-001 -8.8231587e-001 -9.6609671e-001 -9.3963693e-001 -5
```

Same as test file and feature file was like this.

new 6 | X_train.txt | features.txt

```
1 tBodyAcc-mean()-X
2 tBodyAcc-mean()-Y
3 tBodyAcc-mean()-Z
4 tBodyAcc-std()-X
5 tBodyAcc-std()-Y
6 tBodyAcc-std()-Z
7 tBodyAcc-mad()-X
8 tBodyAcc-mad()-Y
9 tBodyAcc-mad()-Z
10 tBodyAcc-max()-X
11 tBodyAcc-max()-Y
12 tBodyAcc-max()-Z
13 tBodyAcc-min()-X
14 tBodyAcc-min()-Y
15 tBodyAcc-min()-Z
16 tBodyAcc-sma()
17 tBodyAcc-energy()-X
18 tBodyAcc-energy()-Y
19 tBodyAcc-energy()-Z
20 tBodyAcc-iqr()-X
21 tBodyAcc-iqr()-Y
22 tBodyAcc-iqr()-Z
23 tBodyAcc-entropy()-X
24 tBodyAcc-entropy()-Y
25 tBodyAcc-entropy()-Z
26 tBodyAcc-arCoeff()-X,1
27 tBodyAcc-arCoeff()-X,2
28 tBodyAcc-arCoeff()-X,3
29 tBodyAcc-arCoeff()-X,4
30 tBodyAcc-arCoeff()-Y,1
31 tBodyAcc-arCoeff()-Y,2
32 tBodyAcc-arCoeff()-Y,3
33 tBodyAcc-arCoeff()-Y,4
34 tBodyAcc-arCoeff()-Z,1
35 tBodyAcc-arCoeff()-Z,2
36 tBodyAcc-arCoeff()-Z,3
37 tBodyAcc-arCoeff()-Z,4
38 tBodyAcc-correlation()-X,Y
```

So to get arrange this data we load these files as a data frame and then merge them and apply each column name on its column and then save it as csv file which looks like this.

new 6 X_train.txt features.txt new_train_data.csv

```
tBodyAcc-mean()-X,tBodyAcc-mean()-Y,tBodyAcc-mean()-Z,tBodyAcc-std()-X,tBodyAcc-std()-Y,tBodyAcc-std()-Z,tBodyAcc-mad()-X,tBodyAcc-mad()-Y,tBodyAcc-mad()-Z,tBody
0.28858451,-0.020294171,-0.13290514,-0.9952786,-0.98311061,-0.91352645,-0.99511208,-0.98318457,-0.92352702,-0.93472378,-0.56737807,-0.74441253,0.85294738,0.68584
0.27841883,-0.016410568,-0.12352019,-0.99824528,-0.97530022,-0.96032199,-0.99880719,-0.97491437,-0.95768622,-0.94306751,-0.55785126,-0.81840869,0.84930787,0.6858
0.27965306,-0.019467156,-0.11346169,-0.99537956,-0.96718701,-0.97894396,-0.99651994,-0.96366837,-0.97746859,-0.93869155,-0.55785126,-0.81840869,0.84360895,0.6824
0.27917394,-0.026200646,-0.12328257,-0.99609149,-0.9834027,-0.9906751,-0.99709947,-0.98274984,-0.9893025,-0.93869155,-0.57615889,-0.82971145,0.84360895,0.6824009
0.27662877,-0.016569655,-0.11536185,-0.99813862,-0.98081727,-0.99048163,-0.99832113,-0.97967187,-0.99044113,-0.94246912,-0.56917385,-0.82470529,0.84909512,0.6832
0.27719877,-0.01009785,-0.10513725,-0.99733496,-0.99048681,-0.99542003,-0.997627399999999,-0.99021769,-0.9955489,-0.94246912,-0.56568389,-0.82276614,0.84909512,
0.27945388,-0.019640776000000002,-0.11002215,-0.99692104,-0.96718593,-0.98311783,-0.99700268,-0.96609671,-0.98311627,-0.94098663,-0.56568389,-0.81718902,0.851040
0.27743247,-0.030488302999999998,-0.12536043,-0.99655926,-0.96672843,-0.98158533,-0.99648525,-0.96631315,-0.98298176,-0.94098663,-0.57263824,-0.81718902,0.850327
0.27729342,-0.021750698,-0.12075082,-0.99732847,-0.96124532,-0.98367156,-0.99759576,-0.95723623,-0.98437928,-0.94059758,-0.56417505,-0.82352693,0.85032778,0.6704
0.28058569,-0.0099602983,-0.10606516,-0.99480344,-0.9727584,-0.98624387,-0.99540462,-0.97366322,-0.98564195,-0.94002751,-0.55459369,-0.81585037,0.84544241,0.6847
0.27688027,-0.012721805,-0.10343832,-0.99481511,-0.97307692,-0.98535702,-0.99550927,-0.97394796,-0.98517247,-0.94002751,-0.55459369,-0.81585037,0.84544241,0.6847
0.27622817,-0.021441302000000002,-0.10820234,-0.99824595,-0.98721376,-0.99272659,-0.99825127,-0.98599654,-0.99318188,-0.94390578,-0.57142214,-0.82069493,0.850532
0.278457,-0.020414761,-0.11273172,-0.99913488,-0.98468004,-0.99627424,-0.99907654,-0.98293702,-0.99641031,-0.94390578,-0.56970419,-0.82503369,0.85206858,0.686527
0.27717497,-0.014712801999999999,-0.10675647,-0.99918834,-0.99052638,-0.99336501,-0.99921135,-0.99068725,-0.99216753,-0.94332286,-0.56935747,-0.822514699999999,
0.29794572,0.027093907999999996,-0.061668123,-0.98864079,-0.816698599999999,-0.90190653,-0.98895795,-0.79428042,-0.888014600000001,-0.92597669,-0.44846574,-0.7
0.27920345,-0.023020143,-0.12208028,-0.99683904,-0.97484812,-0.98338551,-0.99709389,-0.97333193,-0.98406535,-0.941715800000001,-0.57093916,-0.81763455,0.848666
0.27903836,-0.014800378000000001,-0.11684896,-0.99694116,-0.98186562,-0.98257653,-0.99721998,-0.98161964,-0.98133604,-0.941715800000001,-0.56349958,-0.8242149,0
0.2801349,-0.013916951,-0.10637048,-0.99769492,-0.98751567,-0.99040744,-0.99801432,-0.98795448,-0.99219012,-0.94207598,-0.56349958,-0.81589176,0.84985145,0.69387
0.27773106,-0.018210718,-0.10918803,-0.99749074,-0.99322197,-0.99612795,-0.99790305,-0.99271072,-0.99649182,-0.94487012,-0.57509103,-0.81589176,0.84715408,0.6925
0.27556818,-0.016979698,-0.11142918,-0.99781139,-0.9905223,-0.99762104,-0.99820522,-0.98946983,-0.99719303,-0.94566163,-0.57334183,-0.82658225,0.84715408,0.69259
0.27756171,-0.014318487,-0.10787724,-0.99790424,-0.99431129,-0.99595166,-0.99836517,-0.99360447,-0.99559481,-0.94147242,-0.57295816,-0.82002259,0.85161294,0.6947
0.27715238,-0.017983328,-0.10660117,-0.99776322,-0.98995727,-0.99658567,-0.99829082,-0.9896687,-0.99670045,-0.94147242,-0.57295816,-0.81751965,0.84968627,0.68945
0.2756763,-0.021264234,-0.11080122,-0.99786211,-0.99009076,-0.99459257,-0.99833345,-0.98947266,-0.99448451,-0.94456672,-0.57528783,-0.81751965,0.84968627,0.68945
0.2792002,-0.017714426999999998,-0.10916135,-0.99838929,-0.98730784,-0.99083159,-0.99886852,-0.98677131,-0.98963739,-0.94367519,-0.56780337,-0.82543702,0.8500151
0.28171549,-0.011910678000000001,-0.10287513,-0.99853388,-0.98848901,-0.99318367,-0.99867433,-0.98854392,-0.99328718,-0.94255883,-0.56386922,-0.81539879,0.852528
0.27899267,-0.014531028999999999,-0.10659617,-0.99805969,-0.98606983,-0.99342424,-0.99805916,-0.98519239,-0.99501788,-0.94255883,-0.56386922,-0.81539879,0.850440
0.27573444,-0.01801884,-0.10677578,-0.99925496,-0.99366888,-0.994188800000001,-0.99940654,-0.99362014,-0.99358295,-0.94288961,-0.57586911,-0.82266472,0.85044026,
0.14450396,0.18926326,0.062769317,-0.90429967,-0.18193654,-0.44315051,-0.90110019,-0.11081268,-0.40059935,-0.93189578,0.042098681,-0.3365258,0.71647623,0.6558130
0.28725164,-0.037455063999999996,-0.14597431,-0.98291504,-0.89160489,-0.94143811,-0.98441781,-0.89137296,-0.93336072,-0.93189578,-0.49953681,-0.82469972,0.826365
0.27999759999999996,-0.019484036,-0.10572355,-0.99281839,-0.94035041,-0.98149266,-0.99309238,-0.93692706,-0.980669099999998,-0.935388999999999,-0.56525476,-0.8
0.22184689,0.034107675,-0.12361242,-0.81470826,-0.749418,-0.57189889,-0.87868152,-0.77689463,-0.70070978,-0.73715155,-0.31840187,-0.47333399,0.50346078,0.6607428
-0.041701306,0.17510248,0.025551777999999997,-0.75839283,-0.58653735,-0.43911019,-0.77411817,-0.555350300000001,-0.43873576,-0.73715155,-0.21299426,-0.47333399,
0.013903671000000001,0.15329608,0.016242945,-0.80863778,-0.57274619,-0.62641542,-0.80621297,-0.5338107,-0.57939359,-0.91606376,-0.21299426,-0.4804685,0.56631576,
0.30368927,-0.0042428053,-0.15085028,-0.95650333,-0.83867155,-0.94300953,-0.96200594,-0.84482083,-0.93719466,-0.91606376,-0.43888293,-0.81719544,0.80103579,0.676
0.29075969,-0.02403476,-0.11673889,-0.97737449,-0.96384053,-0.96250503,-0.9776306,-0.95938205,-0.96225396,-0.91682527,-0.57174232,-0.82009798,0.84280757,0.676809
0.27090823,-0.014412968,-0.10652245,-0.99385604,-0.993178700000001,-0.993470300000001,-0.993878,-0.99213086,-0.99278814,-0.9416171,-0.57174232,-0.82009798,0.84
```

yGyroMean_gravityMean)	angle(tBodyGyroJerkMean_gravityMean)	angle(X_gravityMean)	angle(Y_gravityMean)	angle(Z_gravityMean)	subject	Activity
-0.464761	-0.018446	-0.841247	0.179941	-0.058627	1	STANDING
-0.732626	0.703511	-0.844788	0.180289	-0.054317	1	STANDING
0.100699	0.808529	-0.848933	0.180637	-0.049118	1	STANDING
0.640011	-0.485366	-0.848649	0.181935	-0.047663	1	STANDING
0.693578	-0.615971	-0.847865	0.185151	-0.043892	1	STANDING
...	...	...	...	...	...	...
0.206972	-0.425619	-0.791883	0.238604	0.049819	30	WALKING_UPSTAIRS
-0.879033	0.400219	-0.771840	0.252676	0.050053	30	WALKING_UPSTAIRS
0.864404	0.701169	-0.779133	0.249145	0.040811	30	WALKING_UPSTAIRS
0.936674	-0.589479	-0.785181	0.246432	0.025339	30	WALKING_UPSTAIRS
-0.056088	-0.616956	-0.783267	0.246809	0.036695	30	WALKING_UPSTAIRS

Same process was done for test data, so now our data is ready let's move on towards out model which we need to train using this training data.

3.1 Data Mining Algorithms

Decision Tree

First article I've had the choice to find that develops a " Decision tree" approach dates to 1959 and a British expert, William Belson,whose hypothetical depicts his philosophy as one of planning people tests and making rules for doing thusly: In this article Dr Belson portrays a technique for organizing masses tests. This depends upon the mix of tentatively made markers to give the best available perceptive, or organizing, composite. The fundamental standard is entirely indisputable from that characteristic in the various relationship methodology.

it is an avaricious calculation. In this calculation Tree is developed in a top down recursive and separate and-vanquish way. At start, all the preparation models are the root. At that point most significant qualities are chosen on the dependent on entropy, info gain.

We have two technical terms in decision tree, Entropy and info gain. Entropy is a measure of the number of possible arrangements the data points in a system can have.

And info gain is based on the decrease in entropy after a data-set is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain. We get information grain using gini index formula of both are given below.

Gini Index

$$I_G = 1 - \sum_{j=1}^{c} p_j^2$$

p_j: proportion of the samples that belongs to class c for a particular node

Entropy

$$I_H = - \sum_{j=1}^{c} p_j log_2(p_j)$$

p_j: proportion of the samples that belongs to class c for a particular node.

*This is the the definition of entropy for all non-empty classes ($p \neq 0$). The entropy is 0 if all samples at a node belong to the same class.

Advantages and Disadvantages of Decision Tree:

Advantages

- Easy to traverse or understand for small-sized trees
- Decision trees perform classification without requiring much computation.

Disadvantages

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Not good with large set of attributes)

Random Forest

Random Forest was created by Breiman, the technique consolidates Breiman's sacking testing approach and the arbitrary choice of highlights, presented autonomously by Ho.

The RF merges hundreds or thousands of decision trees, plans everybody on a fairly extraordinary game plan of the observations, separating center points in each tree contemplating a foreordained number of the features. The last desires for the sporadic woods are made by averaging the figures of each decision tree.

Random Forest use the concept of Bagging: Bagging Bootstrap Aggregation is utilized to diminish the difference of a D.T. Assume a set D of d tuples, at every cycle I, a preparation set Di of d tuples is examined with substitution from D (i.e., bootstrap). At that point a classifier model Mi is found out for each preparation set D < I. Every classifier Mi restores its class expectation. The sacked classifier M* checks the votes and doles out the class with the most votes to X (obscure example).

Random Forest is an augmentation over bagging. Every classifier in the gathering is a decision tree classifier and is created utilizing a random determination of ascribes at every hub to decide the split. During grouping, each tree votes and the most well-known class is returned.

Advantages and Dis advantages of Random Forest:

Advantages

- The prescient execution can contend with the best directed learning calculations.
- One of the biggest advantages of random forest is its versatility. It can be used for both regression and classification tasks, and it's also easy to view the relative importance it assigns to the input features.

Disadvantages

- The main disadvantage of Random forests is their complexity. They are much harder and time-consuming to construct than decision trees
- It is difficult to understand an ensemble of classifiers

4 Modeling

Experiment one (DecisionTree-J48)

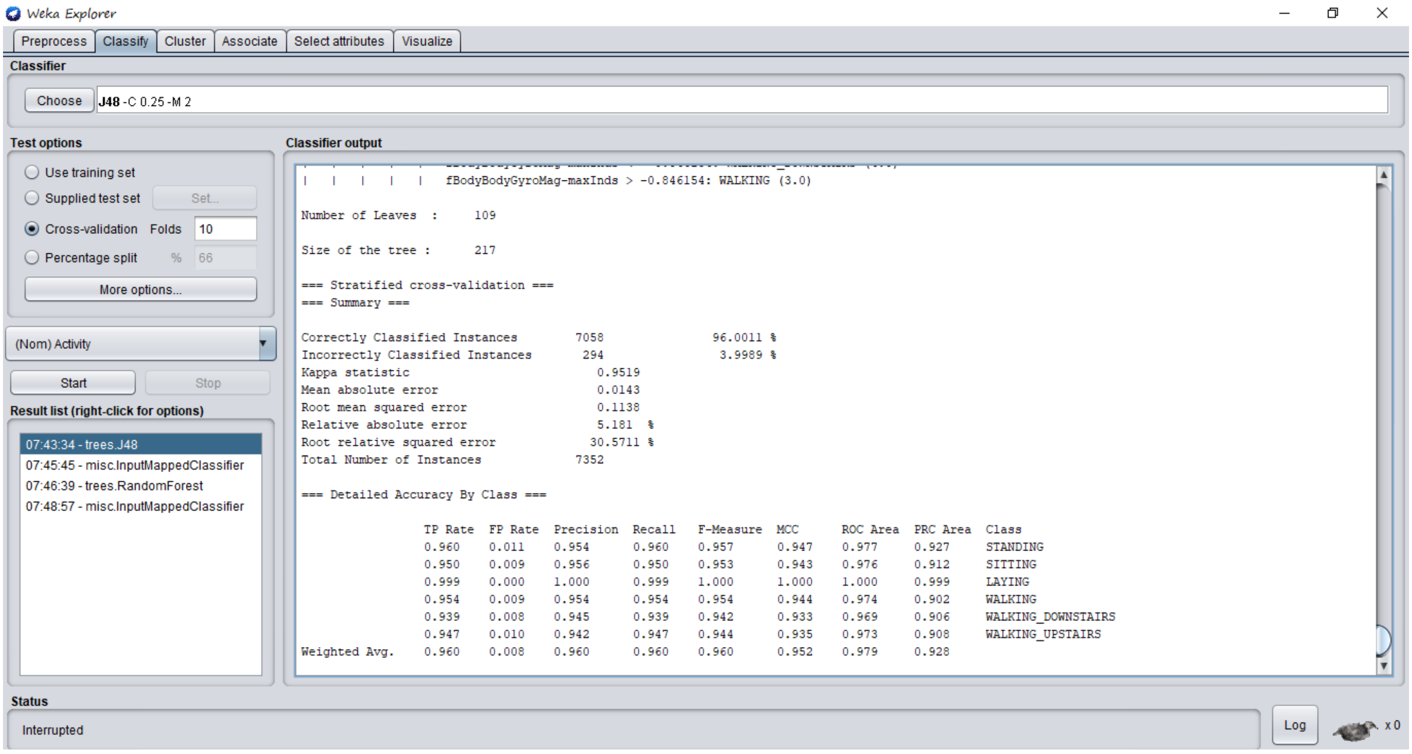
With Cross-Validation, 10 Folds

Build Decision Tree model and predict Human Action

Aim: to predict a Human Actions

Methodologies

We are using decision tree using J48-C 0.25 –M 2 and for validation we are using cross validation with 10 Folds.



We can see the result 96% actions are correctly classified.

Experiment Two (DecisionTree-J48)

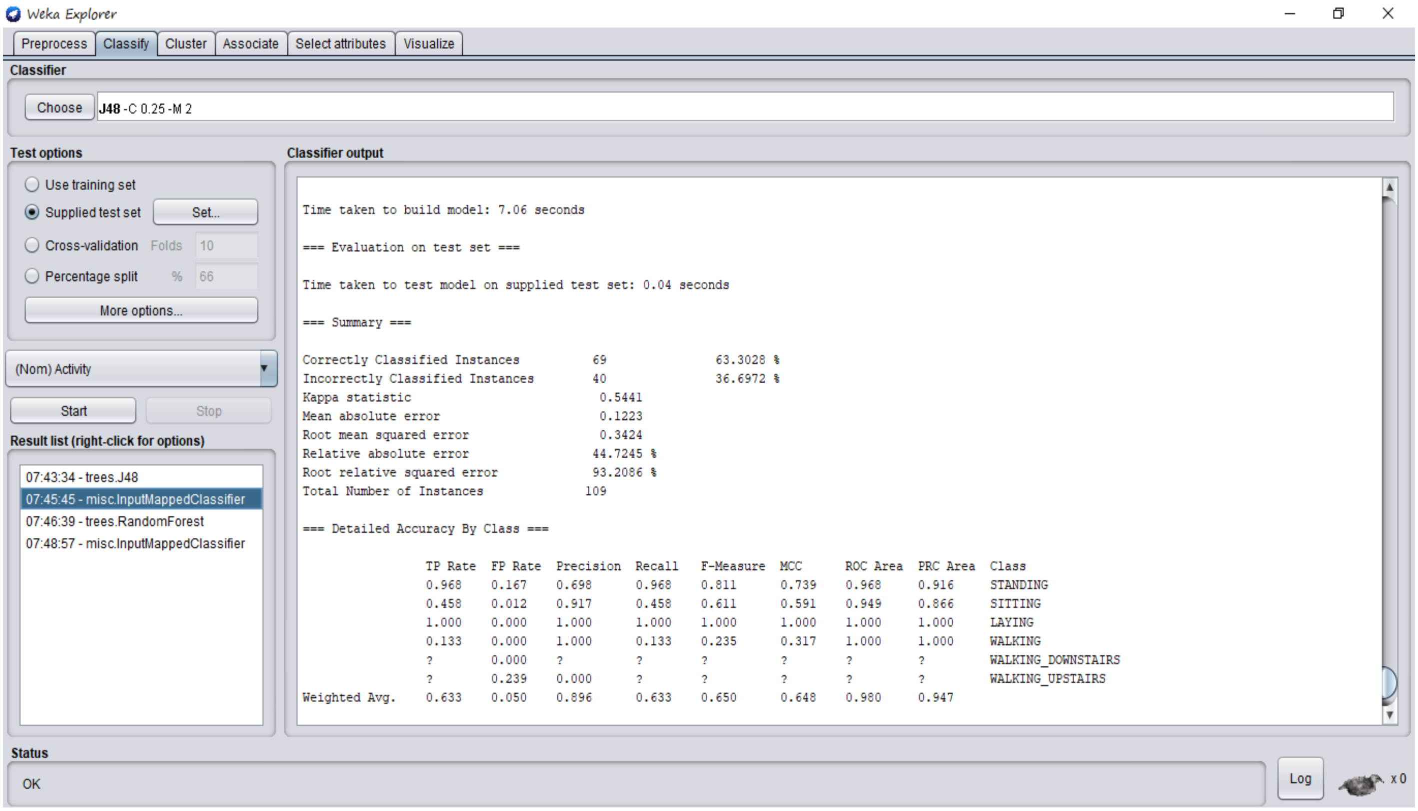
Now we are using testing data for predictions

Build Decision-Tree model and predict Human Actions.

Aim: to predict a Human actions.

Methodologies

In this experiment we are using decision tree-J48 for classifying human actions and we are using test data for predictions



Conclusion

We can see accuracy of decision tree drops down to 63% when we use test data for testing.

Visualization

```
=== Confusion Matrix ===

  a  b  c  d  e  f   <-- classified as
 30  1  0  0  0  0 |  a = STANDING
 13 11  0  0  0  0 |  b = SITTING
  0  0 24  0  0  0 |  c = LAYING
  0  0  0  4  0 26 |  d = WALKING
  0  0  0  0  0  0 |  e = WALKING_DOWNSTAIRS
  0  0  0  0  0  0 |  f = WALKING_UPSTAIRS
```

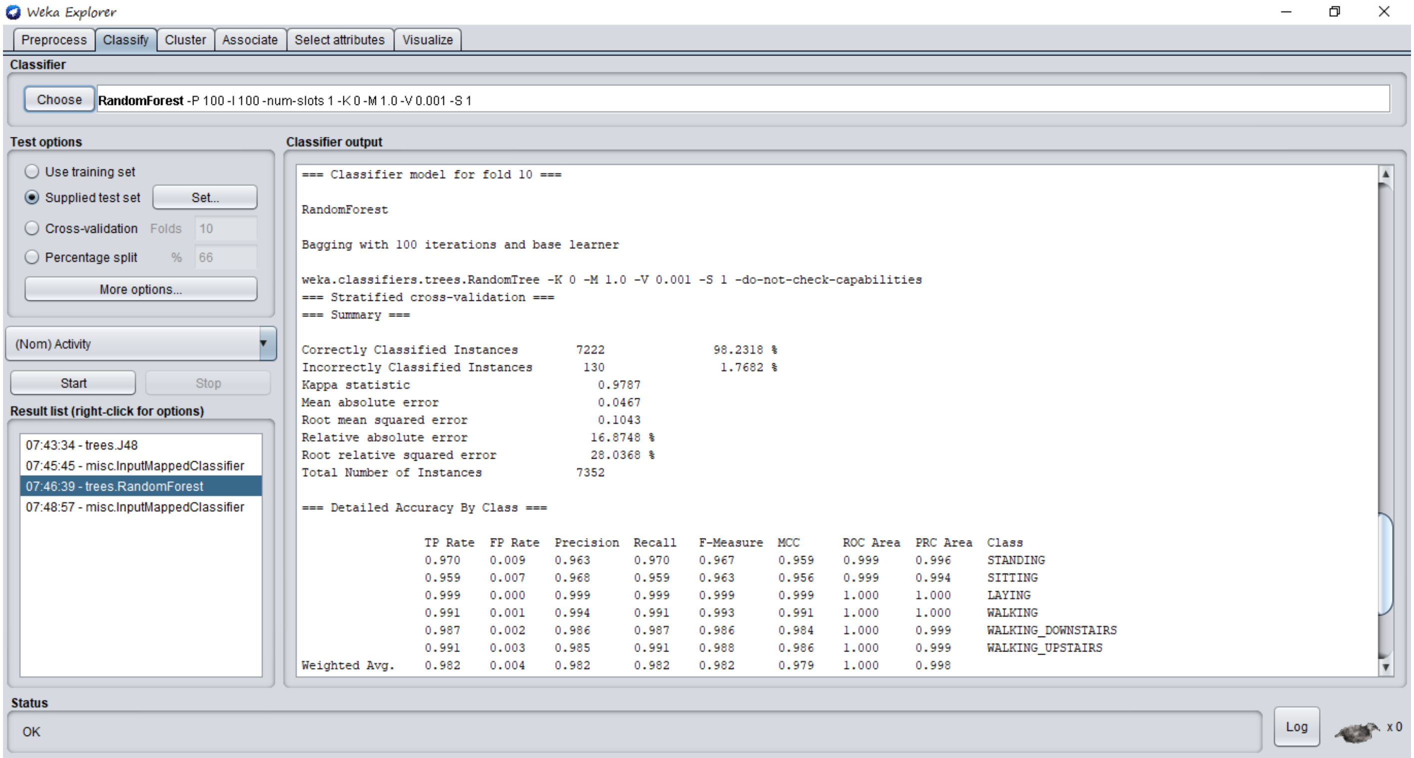
Experiment Three (Random Forest) With cross validation 10 Folds

Build Random Forest model and predict Human Action

Aim: to predict a Human action belongs to category

Methodologies

Now we are using RF for classifying human actions and for validation we are using cross Validation with 10 folds.



Conclusion

In this experiment results we have 98% accuracy. If we compare the results of Random Forest and decision tree, of course Random Forest is better than decision tree.

Visualization

```
=== Confusion Matrix ===

    a    b    c    d    e    f   <-- classified as
 1333   41    0    0    0    0 |    a = STANDING
   51 1233    1    0    0    1 |    b = SITTING
    0    0 1405    0    0    2 |    c = LAYING
    0    0    0 1215    6    5 |    d = WALKING
    0    0    0    5  973    8 |    e = WALKING_DOWNSTAIRS
    0    0    0    2    8 1063 |    f = WALKING_UPSTAIRS
```

Experiment Four (Random Forest)

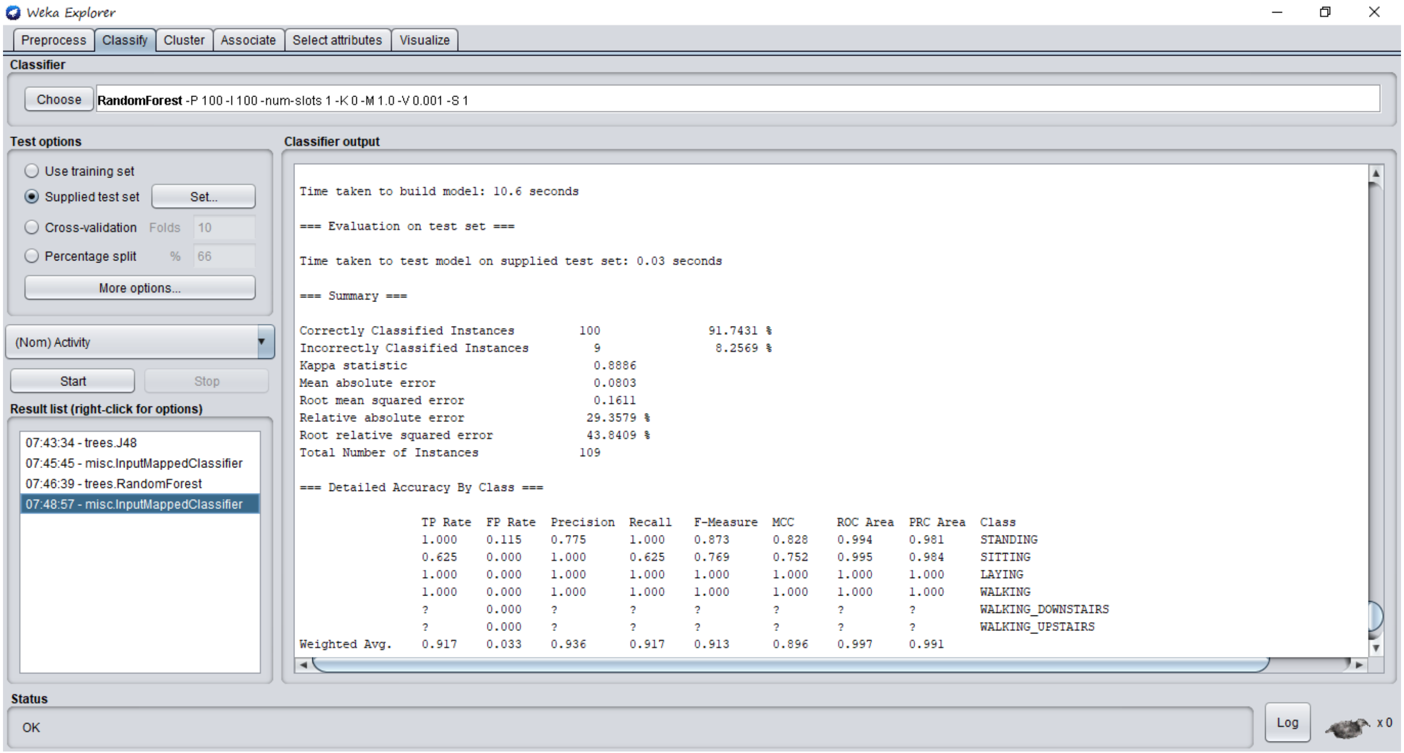
With Test data for predictions.

Build RF model and predict Human action category

Aim: to predict a Human Actions using RF classifier

Methodologies

In this experiment we are using training samples for training our random forest model and then predict the results using test data.



Conclusion

We can got 91% accuracy on test data which is much better then decision tree but if we look at build time it takes 10.6 sec and we look the time of decision tree it took 6 sec so, It is clear that RF takes more time to build and train as compare to DT but it gives more accurate results. Time it's against accuracy.

Visualization

```
=== Confusion Matrix ===

  a  b  c  d  e  f   <-- classified as
 31  0  0  0  0  0 |  a = STANDING
  9 15  0  0  0  0 |  b = SITTING
  0  0 24  0  0  0 |  c = LAYING
  0  0  0 30  0  0 |  d = WALKING
  0  0  0  0  0  0 |  e = WALKING_DOWNSTAIRS
  0  0  0  0  0  0 |  f = WALKING_UPSTAIRS
```

Conclusion

So after performing multiple experiments we have concluded that cross-validation with folds gives much accurate results then train model on training data and then test it on testing data and 2nd conclusion is decision tree is faster than Random forest and if we see at accuracy wise it is opposite RF is much better then DT

6.0 Final Results

Experiments No	Accuracy
1: DecisionTree-J48(cross-vald)	96%
2: DecisionTree-J48(test data)	63%
3: Random Forest (cross-vald)	98%
4: Random Forest (test data)	96%

7.0 References

[Online] Available from: http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones
[Accessed 25 May 2020]

[Online] Available from: https://www.youtube.com/watch?v=yEN-1iJNBmw
[Accessed 26 May 2020]

[Online] Available from: https://www.shirin-glander.de/2018/10/ml_basics_rf/
[Accessed 26 May 2020]

[Online] Available from: https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/ml-decision-tree/tutorial/
[Accessed 25 May 2020]

[Online] Available from: https://www.cs.waikato.ac.nz/ml/weka/
[Accessed 25 May 2020]