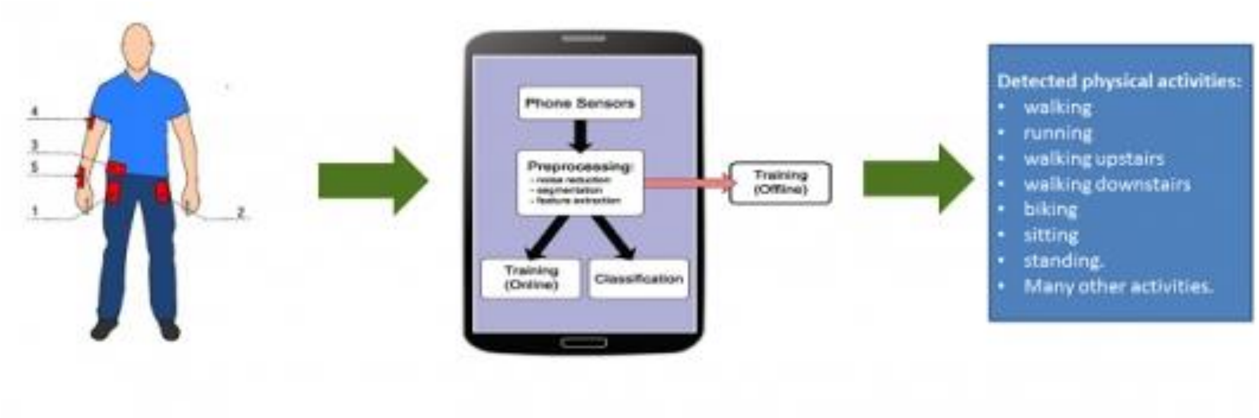


RUTGERS

HUMAN ACTIVITY RECOGNITION WITH SMARTPHONE DATA

Submitted By : Abhilash Basaru Yethesh Kumar, Adarsh Vijayaraghavan & Gokulramanan Soundararajan
Submitted To : Farid Alizadeh

Course: Algorithmic Learning Theory

CONTENT

Data Collection Methodology

Exploring the data

- Dimensions
- Correlation
- Distribution
- Variance

Classification # 1 – Naïve Bayes

- Naïve Bayes
- Naïve Bayes with Laplace Smoothening

Classification # 2 – Logistic Regression

- Logistic regression
- Lasso regression – A primer
- Lasso regression – Tuning parameters
- Lasso regression – Confusion table

Classification # 3 – Neural Networks

Future Work

Conclusions

DATA COLLECTION METHODOLOGY

- Identify the activity carried out by a person from a set of observations about them.
- Use data from accelerometer and gyrometer sensors which can be collected from most modern smartphones
- Data collected from 30 volunteers and released to public domain by researchers.
- Available in UCI Machine Learning repository
- 6 activities : Stand, Sit, Walk, Lay Down, Walk Upwards, Walk Downwards
- Noise reduction using techniques like median filter, Butterworth filter. Signals mapped into frequency domain using FFT
- 561 features were extracted in total

EXPLORING THE DATA

- Data pre-divided into training and test sets
- 7352 records in training set, 2947 in test set
- 563 attributes in total

```
df1.shape
```

```
(7352, 563)
```

```
df2.shape
```

```
(2947, 563)
```

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7352 entries, 0 to 7351
Columns: 563 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), int64(1), object(1)
```

- As per 5-number statistic, all values have been normalized to be between -1 and 1
- No null values, clean dataset

```
df1.describe()
```

	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
count	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000
mean	0.274488	-0.017695	-0.109141	-0.605438	-0.510938	-0.604754	-0.630512	-0.526907	-0.606150	-0.468604
std	0.070261	0.040811	0.056635	0.448734	0.502645	0.418687	0.424073	0.485942	0.414122	0.544547
min	-1.000000	-1.000000	-1.000000	-1.000000	-0.999873	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.262975	-0.024863	-0.120993	-0.992754	-0.978129	-0.980233	-0.993591	-0.978162	-0.980251	-0.936219
50%	0.277193	-0.017219	-0.108676	-0.946196	-0.851897	-0.859365	-0.950709	-0.857328	-0.857143	-0.881637
75%	0.288461	-0.010783	-0.097794	-0.242813	-0.034231	-0.262415	-0.292680	-0.066701	-0.265671	-0.017129
max	1.000000	1.000000	1.000000	1.000000	0.916238	1.000000	1.000000	0.967664	1.000000	1.000000

8 rows × 562 columns

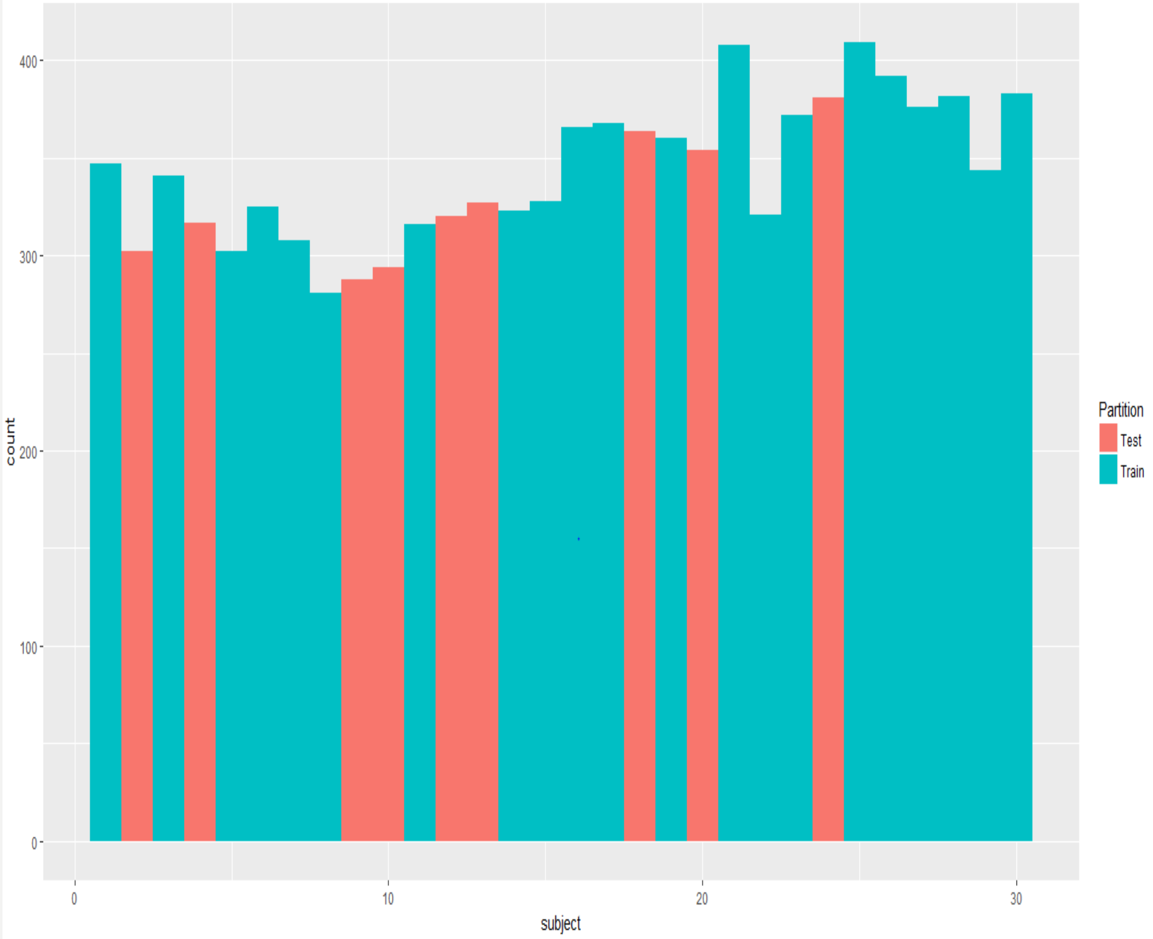
EXPLORING THE DATA - CORRELATION

```
corr = df1[df1.columns].corr()
corr
```

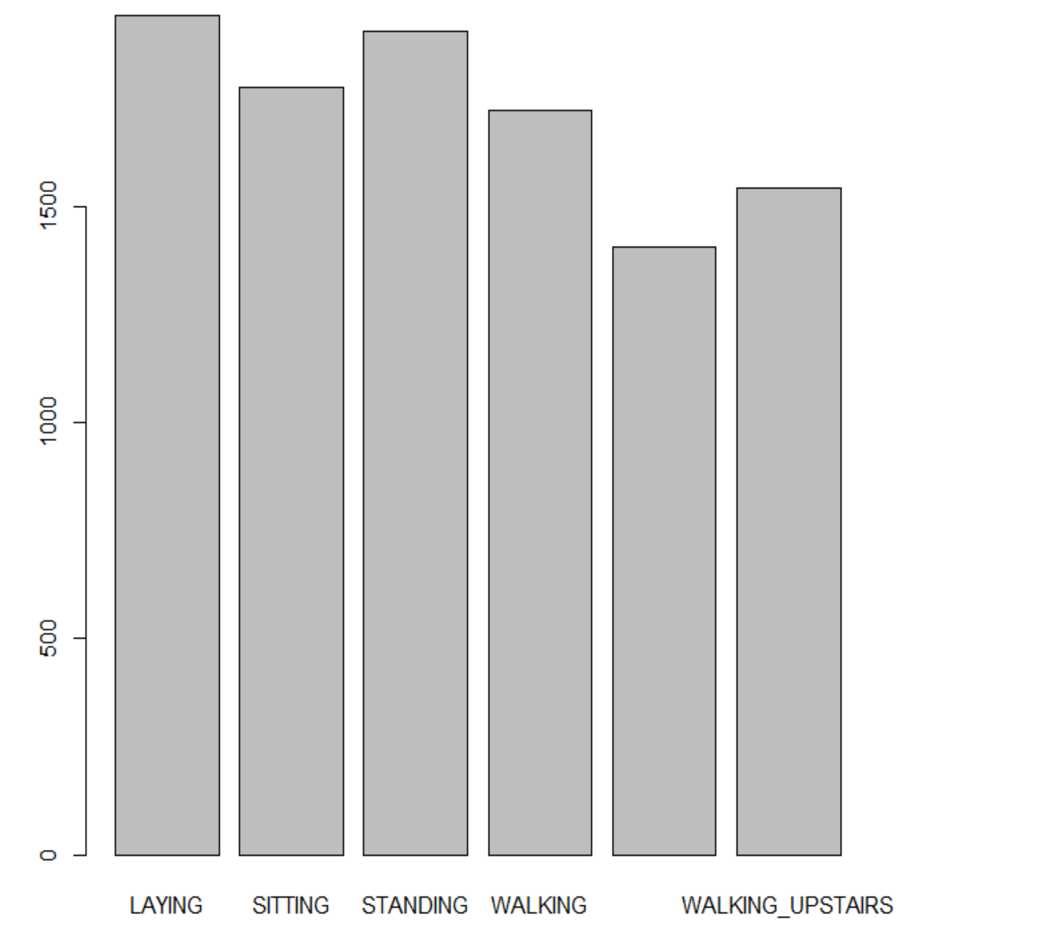
	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBody
tBodyAcc-mean()-X	1.000000	0.148061	-0.256952	0.000619	-0.021903	-0.044617	0.006290	-0.022754	-0.047558	0.044062	...	
tBodyAcc-mean()-Y	0.148061	1.000000	-0.078769	-0.045160	-0.044920	-0.049746	-0.044180	-0.045049	-0.050402	-0.038108	...	
tBodyAcc-mean()-Z	-0.256952	-0.078769	1.000000	-0.020217	-0.016641	-0.008410	-0.018747	-0.015203	-0.001988	-0.037197	...	
tBodyAcc-std()-X	0.000619	-0.045160	-0.020217	1.000000	0.927461	0.851668	0.998632	0.920888	0.846392	0.980844	...	
tBodyAcc-std()-Y	-0.021903	-0.044920	-0.016641	0.927461	1.000000	0.895510	0.922803	0.997347	0.894509	0.917366	...	
tBodyAcc-std()-Z	-0.044617	-0.049746	-0.008410	0.851668	0.895510	1.000000	0.844469	0.891441	0.997418	0.853884	...	
tBodyAcc-mad()-X	0.006290	-0.044180	-0.018747	0.998632	0.922803	0.844469	1.000000	0.916106	0.839267	0.973216	...	
tBodyAcc-mad()-Y	-0.022754	-0.045049	-0.015203	0.920888	0.997347	0.891441	0.916106	1.000000	0.891178	0.910411	...	
tBodyAcc-mad()-Z	-0.047558	-0.050402	-0.001988	0.846392	0.894509	0.997418	0.839267	0.891178	1.000000	0.847870	...	
tBodyAcc-max()-X	0.044062	-0.038108	-0.037197	0.980844	0.917366	0.853884	0.973216	0.910411	0.847870	1.000000	...	
tBodyAcc-max()-Y	-0.007875	0.090189	-0.027803	0.895217	0.953573	0.866820	0.889934	0.949550	0.865312	0.885533	...	
tBodyAcc-max()-Z	-0.075881	-0.057029	0.110455	0.844993	0.884490	0.937802	0.838920	0.879898	0.931937	0.839990	...	
tBodyAcc-min()-X	0.078354	0.058568	0.006544	-0.966500	-0.937918	-0.860691	-0.962235	-0.933135	-0.856964	-0.941451	...	
tBodyAcc-min()-Y	0.021214	0.132042	0.013678	-0.904539	-0.957736	-0.853346	-0.900336	-0.941377	-0.848485	-0.898652	...	
tBodyAcc-min()-Z	-0.003283	0.037539	0.119078	-0.828170	-0.838818	-0.939072	-0.821987	-0.830013	-0.921870	-0.837620	...	
tBodyAcc-sma()	-0.029204	-0.046390	-0.008180	0.973155	0.971500	0.928042	0.970683	0.968444	0.926489	0.956887	...	
fBodyBodyGyroJerkMag-meanFreq()	0.030681	-0.022395	-0.020481	-0.065987	-0.105621	-0.097978	-0.059972	-0.102908	-0.101864	-0.076599	...	
fBodyBodyGyroJerkMag-skewness()	-0.017557	-0.001587	0.020091	0.148034	0.206227	0.157792	0.149257	0.200890	0.157937	0.154220	...	
fBodyBodyGyroJerkMag-kurtosis()	-0.015613	-0.004459	0.019127	0.115565	0.176946	0.126701	0.117804	0.172809	0.127359	0.120023	...	
angle(tBodyAccMean,gravity)	-0.544320	0.070559	0.052841	-0.035011	-0.020379	-0.006769	-0.042713	-0.023722	-0.008768	-0.033048	...	
angle(tBodyAccJerkMean),gravityMean)	0.012173	-0.013541	-0.039836	-0.021633	-0.012505	-0.020036	-0.021537	-0.012310	-0.020508	-0.021895	...	
angle(tBodyGyroMean,gravityMean)	0.037444	0.017967	-0.063609	0.018985	-0.008507	-0.018429	0.019389	-0.012546	-0.023525	0.025066	...	
angle(tBodyGyroJerkMean,gravityMean)	0.028844	0.075679	-0.034037	-0.024810	-0.014592	-0.006471	-0.024951	-0.012341	-0.007231	-0.028871	...	
angle(X,gravityMean)	-0.035257	-0.005309	0.008587	-0.371653	-0.380531	-0.345011	-0.368191	-0.377025	-0.347389	-0.384192	...	
angle(Y,gravityMean)	0.034371	0.001053	-0.015288	0.471065	0.523600	0.476006	0.466424	0.525081	0.477607	0.480229	...	
angle(Z,gravityMean)	0.028242	-0.013903	-0.022643	0.394825	0.433169	0.482828	0.390922	0.431459	0.479751	0.405023	...	
subject	0.024181	-0.003144	-0.000637	-0.064345	-0.115524	-0.050123	-0.063440	-0.114753	-0.055457	-0.055633	...	

562 rows × 562 columns

EXPLORING THE DATA - DISTRIBUTION



30 subjects randomly distributed in test and training sets

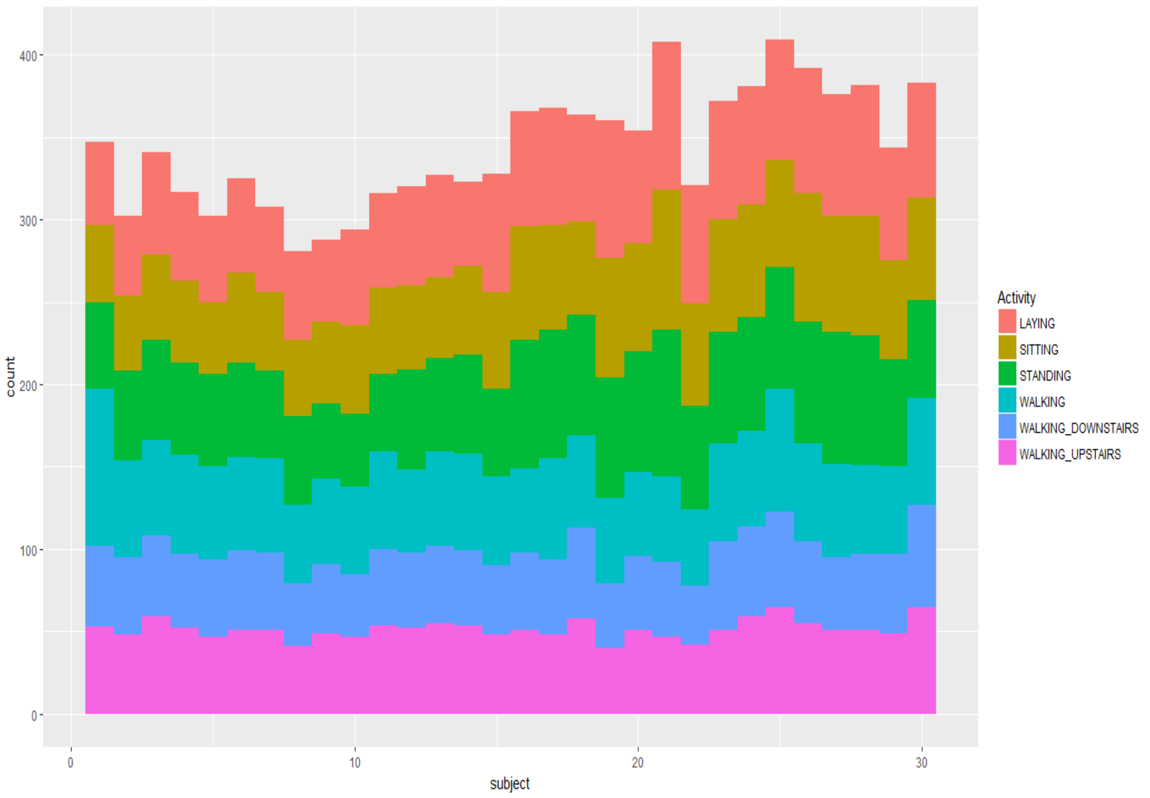


Activities approximately uniformly distributed

EXPLORING THE DATA - DISTRIBUTION

```
pd.crosstab(df1.subject, df1.Activity)
```

Activity subject	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
1	50	47	53	95	49	53
3	62	52	61	58	49	59
5	52	44	56	56	47	47
6	57	55	57	57	48	51
7	52	48	53	57	47	51
8	54	46	54	48	38	41
11	57	53	47	59	46	54
14	51	54	60	59	45	54
15	72	59	53	54	42	48
16	70	69	78	51	47	51
17	71	64	78	61	46	48
19	83	73	73	52	39	40
21	90	85	89	52	45	47
22	72	62	63	46	36	42
23	72	68	68	59	54	51
25	73	65	74	74	58	65
26	76	78	74	59	50	55
27	74	70	80	57	44	51
28	80	72	79	54	46	51
29	69	60	65	53	48	49
30	70	62	59	65	62	65

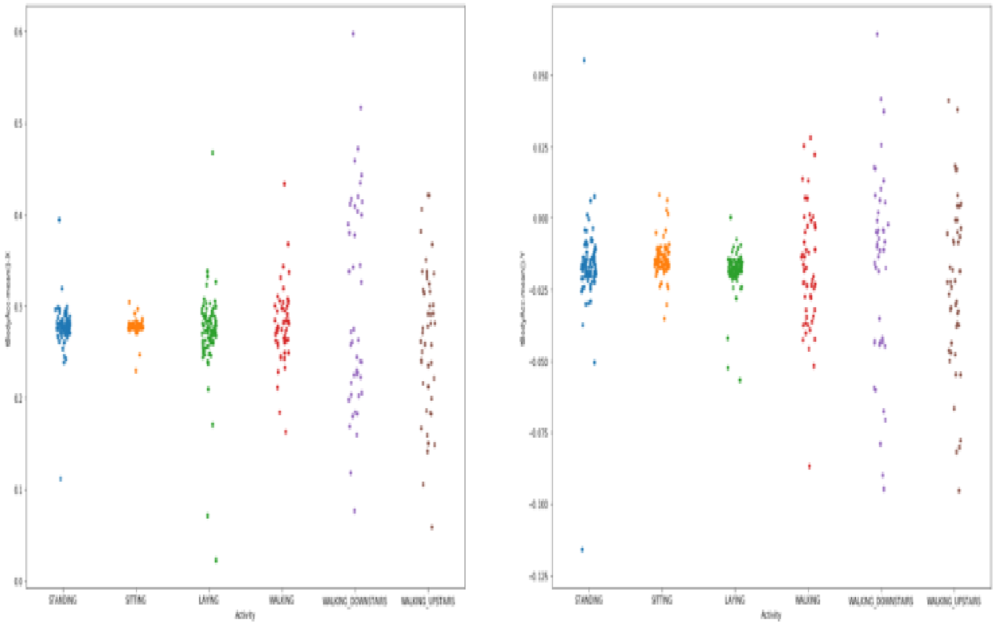


Activities distributed by subject

EXPLORING THE DATA - VARIANCE

```
sub21 = df1.loc[df1['subject']==21]
```

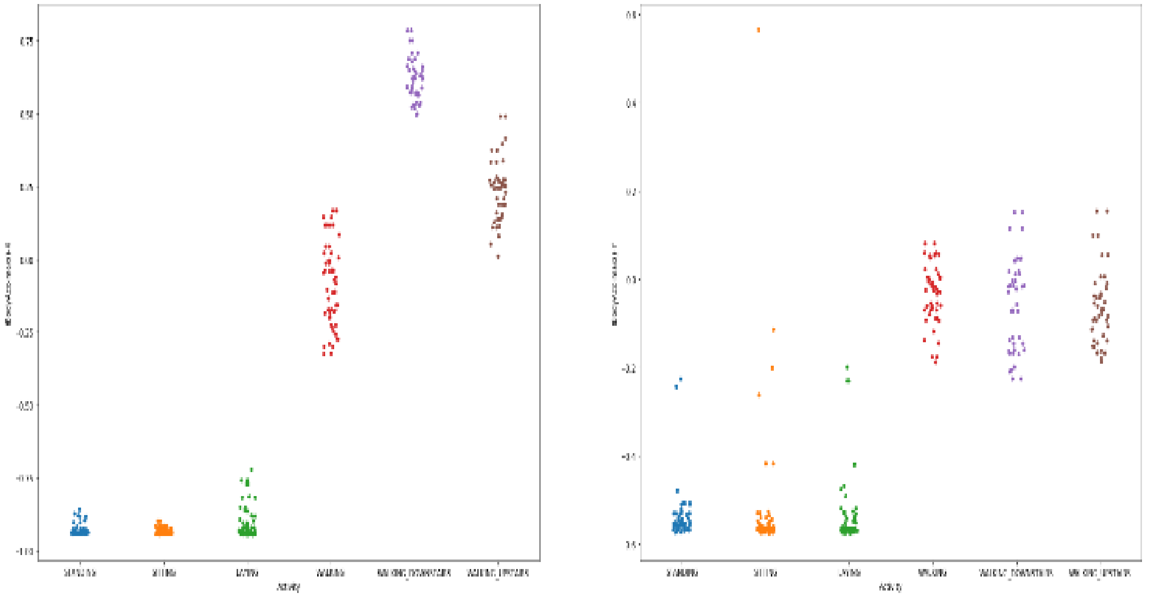
```
fig = plt.figure(figsize=(32,24))
ax1 = fig.add_subplot(221)
ax1 = sns.stripplot(x='Activity', y=sub21.iloc[:,0], data=sub21, jitter=True)
ax2 = fig.add_subplot(222)
ax2 = sns.stripplot(x='Activity', y=sub21.iloc[:,1], data=sub21, jitter=True)
plt.show()
```



So, the mean body acceleration is more variable for walking activities than for passive ones especially in the X direction.

Plotting maximum acceleration with activity.

```
fig = plt.figure(figsize=(32,24))
ax1 = fig.add_subplot(221)
ax1 = sns.stripplot(x='Activity', y='tBodyAcc-max()-X', data=sub15, jitter=True)
ax2 = fig.add_subplot(222)
ax2 = sns.stripplot(x='Activity', y='tBodyAcc-max()-Y', data=sub15, jitter=True)
plt.show()
```



Passive activities fall mostly below the active ones. It actually makes sense that maximum acceleration is higher during the walking activities.

	WALKING	WALKING_UPSTAIRS	WALKING_DOWNSTAIRS	SITTING	STANDING	LAYING
WALKING	534	471	490	0	0	0
WALKING_UPSTAIRS	0	0	0	0	0	0
WALKING_DOWNSTAIRS	0	11	25	0	0	0
SITTING	0	0	0	374	35	10
STANDING	0	0	0	78	311	19
LAYING	3	9	17	44	74	442

Python (SKLearn) : Accuracy of 57.2107 %

pred	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
LAYING	322	5	8	0	0	0
SITTING	212	368	54	0	0	0
STANDING	0	111	455	0	0	0
WALKING	0	0	0	416	80	9
WALKING_DOWNSTAIRS	0	0	0	42	257	11
WALKING_UPSTAIRS	3	7	15	38	83	451

R (NaiveBayes) : Accuracy of 76.9935 %

CLASSIFICATION : NAÏVE BAYES

NAÏVE BAYES WITH LAPLACE SMOOTHING

No change observed with alpha = 1, beta = 10; and alpha = 1, beta = 1000.

This is probably because the activities are uniformly distributed.

The data is from a controlled experiment

LOGISTIC REGRESSION

Python (SKLearn) : Accuracy of 96.4031 %

	WALKING	WALKING_UPSTAIRS	WALKING_DOWNSTAIRS	SITTING	STANDING	LAYING
WALKING	537	0	0	0	0	0
WALKING_UPSTAIRS	0	426	16	0	0	0
WALKING_DOWNSTAIRS	0	64	516	0	0	0
SITTING	0	0	0	487	1	6
STANDING	0	0	0	3	412	1
LAYING	0	1	0	6	7	464

R (glmnet) : Accuracy of 95.7245 %

pred_class	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
LAYING	536	2	0	0	0	0
SITTING	1	431	13	0	0	0
STANDING	0	57	518	0	3	0
WALKING	0	0	1	496	2	28
WALKING_DOWNSTAIRS	0	0	0	0	398	1
WALKING_UPSTAIRS	0	1	0	0	17	442

>

LASSO REGRESSION – A PRIMER

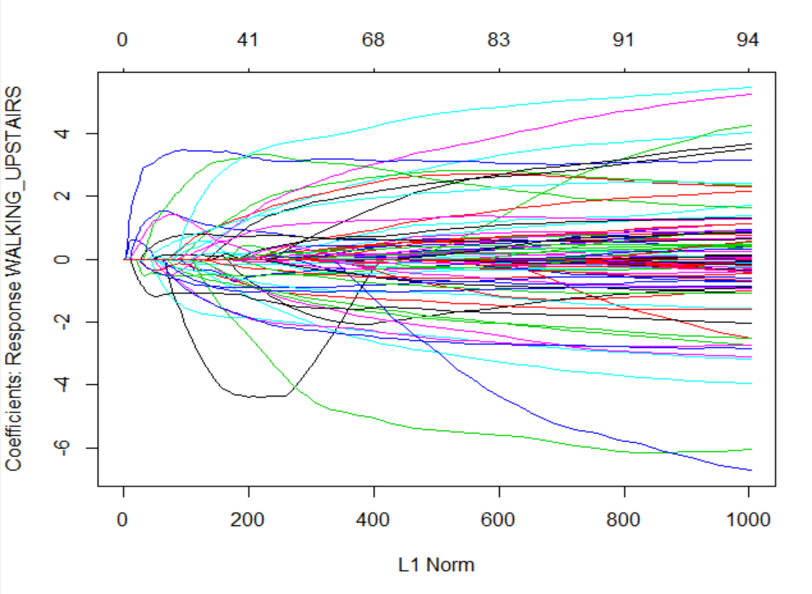
We try to minimize the quantity

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|.$$

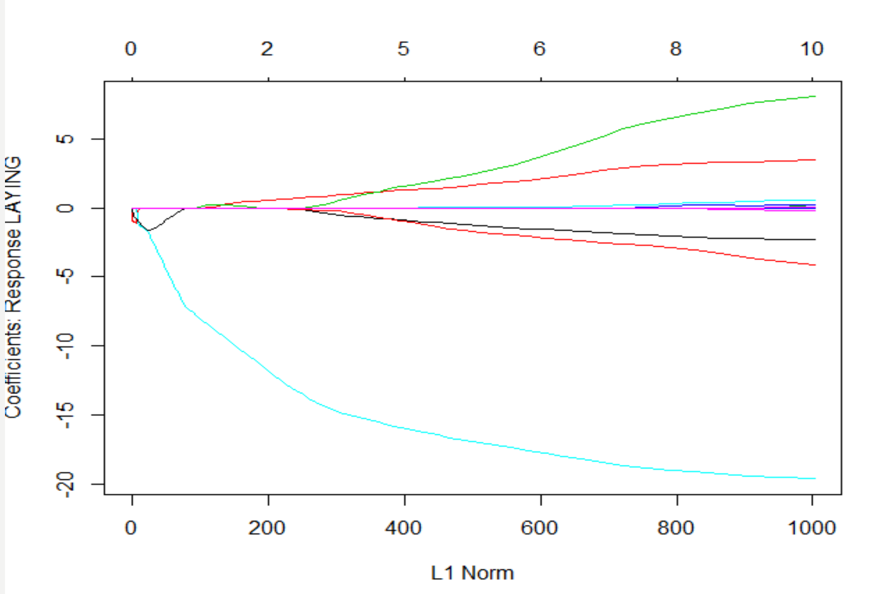
- Add a penalty term that ensures that coefficients "shrink". For instance, the modified least square cost function is shown above
- Contrast with Ridge regression, where we use L2-norm, $|\beta_j|^2$
- In both Ridge regression and Lasso regression, the variance decreases, but the bias increases by a little
- The effect in lasso regression is that some of the coefficients end up being zero
- Allows "variable selection", resulting in sparser models.
- Helps with model interpretability

LASSO– TUNING THE PARAMETER

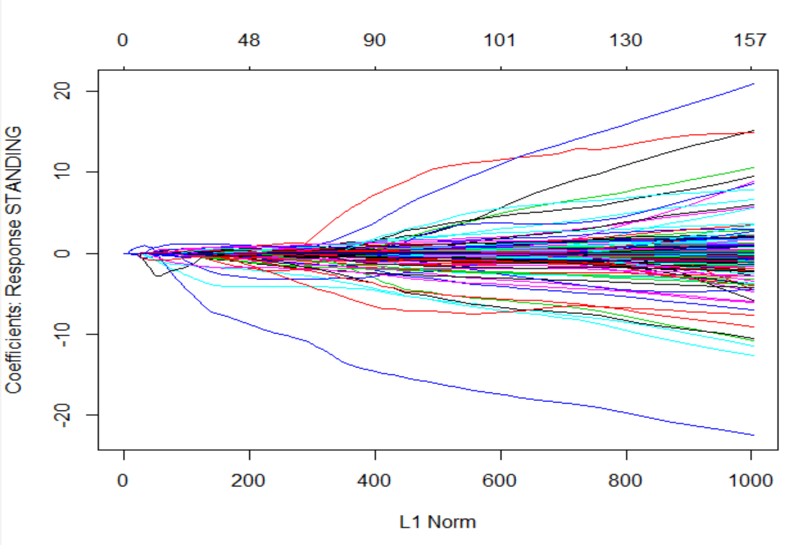
Walking Upstairs



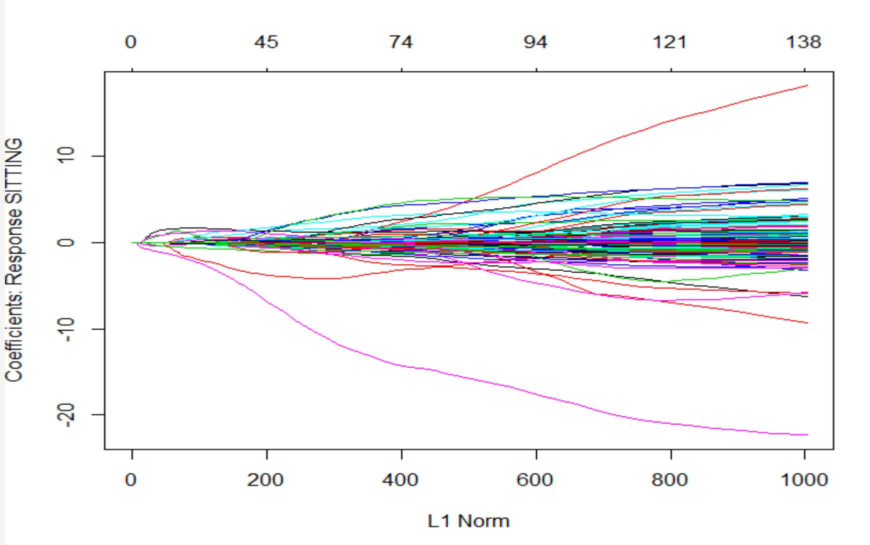
Laying



Standing



Sitting



LASSO – CROSS VALIDATION FOR λ

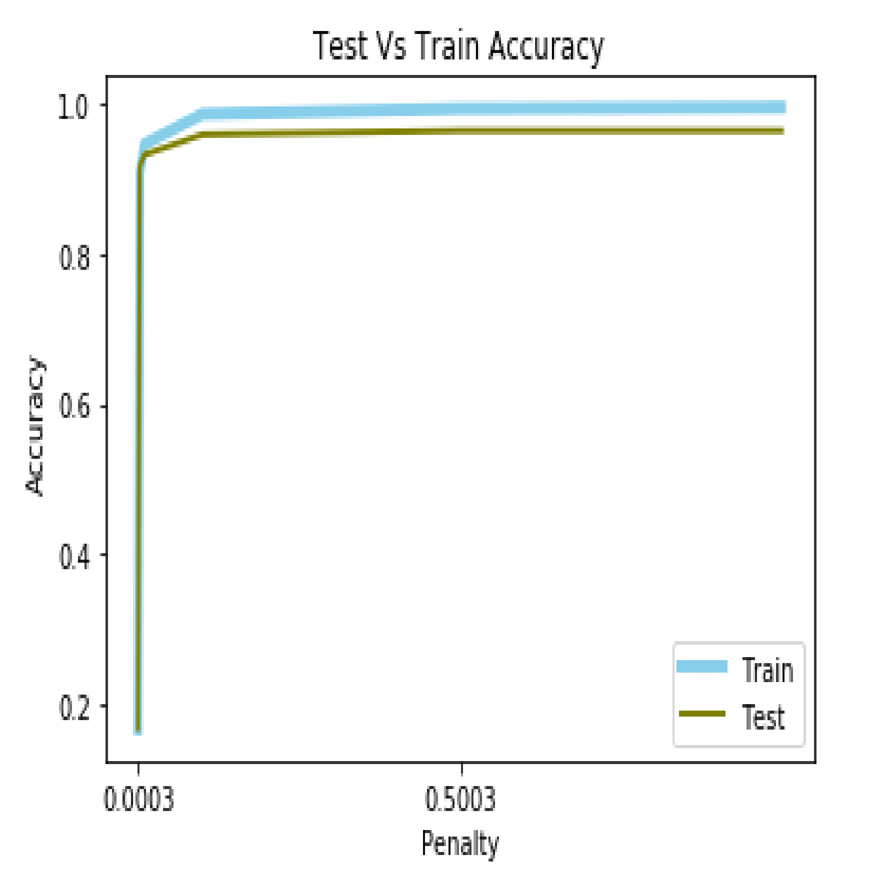
```
Training Accuracy for penalty 1: 0.9952393906420022
Testing Accuracy for penalty 1: 0.9640312181879878
Training Accuracy for penalty 0.5: 0.9933351468988031
Testing Accuracy for penalty 0.5: 0.9636918900576857
Training Accuracy for penalty 0.1: 0.986126224156692
Testing Accuracy for penalty 0.1: 0.9589412962334578
Training Accuracy for penalty 0.01: 0.9465451577801959
Testing Accuracy for penalty 0.01: 0.9317950458092976
Training Accuracy for penalty 0.003: 0.9073721436343852
Testing Accuracy for penalty 0.003: 0.9161859518154055
Training Accuracy for penalty 0.0003: 0.16675734494015235
Testing Accuracy for penalty 0.0003: 0.168306752629793
========================================================
Optimum Penalty value: 0.003
Maximum Testing Accuracy: 0.9161859518154055
Maximum Training Accuracy: 0.9073721436343852
```

Python (SKLearn) : Optimum at 0.003

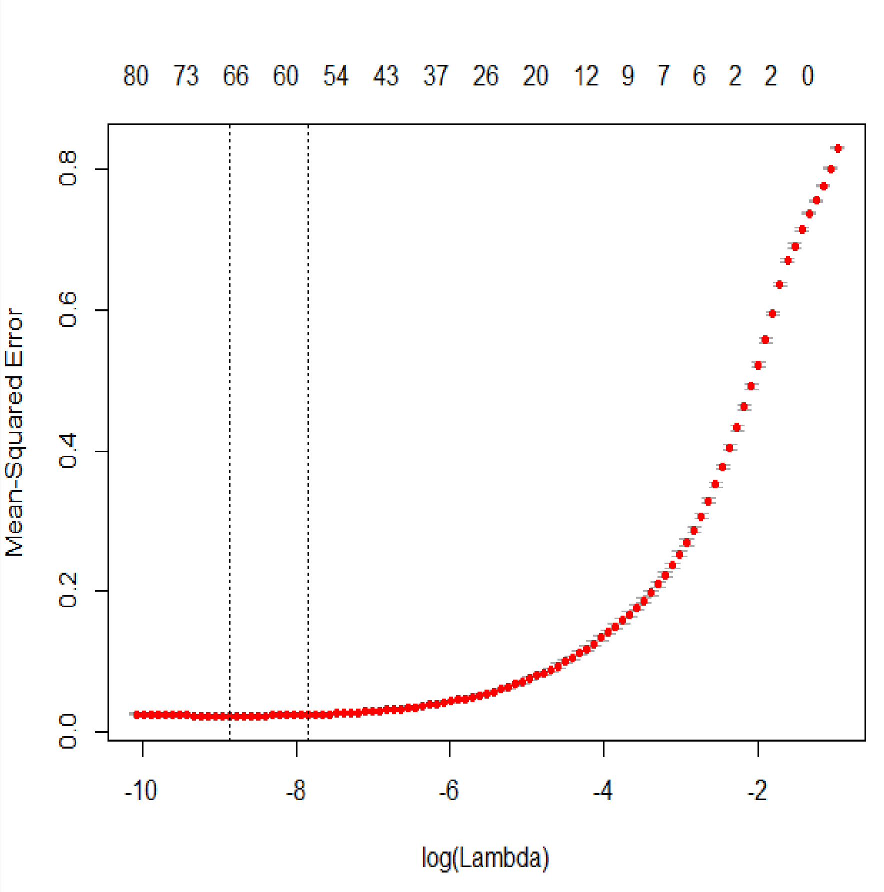
```
> fit

Call:  glmnet(x = x, y = y, family = "multinomial")

       Df                 %Dev      Lambda
 [1,]   0 -0.00000000000004157399 0.38331630000
 [2,]   1  0.04141737000000000196 0.34926350000
 [3,]   1  0.07203302999999999789 0.31823590000
 [4,]   2  0.09651331000000000493 0.28996470000
 [5,]   2  0.11703879999999999839 0.26420500000
 [6,]   3  0.14749750000000000361 0.24073380000
 [7,]   3  0.17987059999999999160 0.21934770000
 [8,]   5  0.20648649999999998950 0.19986140000
 [9,]   9  0.25238909999999997735 0.18210630000
[10,]  11  0.30734909999999998620 0.16592850000
[11,]  12  0.35578799999999999315 0.15118780000
[12,]  12  0.39920909999999998341 0.13775670000
[13,]  12  0.43746920000000000250 0.12551880000
[14,]  14  0.47325610000000001287 0.11436800000
[15,]  17  0.50743930000000003755 0.10420790000
[16,]  20  0.53934269999999995271 0.09495036000
[17,]  21  0.57018950000000001577 0.08651524000
[18,]  23  0.59759940000000000282 0.07882946000
[19,]  26  0.62297539999999995697 0.07182647000
[20,]  27  0.64614050000000000651 0.06544560000
[21,]  28  0.66710329999999995465 0.05963160000
[22,]  30  0.68684009999999995344 0.05433409000
[23,]  33  0.70516639999999997102 0.04950720000
[24,]  36  0.72194219999999997839 0.04510912000
[25,]  37  0.73763869999999998051 0.04110175000
[26,]  39  0.75195149999999999491 0.03745038000
[27,]  40  0.76546840000000004878 0.03412339000
[28,]  45  0.77797349999999998449 0.03109196000
[29,]  46  0.78962140000000002882 0.02832984000
[30,]  48  0.80050189999999998847 0.02581309000
[31,]  51  0.81063169999999995508 0.02351993000
[32,]  53  0.82063580000000002634 0.02143048000
[33,]  56  0.83023029999999997663 0.01952666000
[34,]  64  0.83946639999999994597 0.01779196000
[35,]  68  0.84828729999999996636 0.01621138000
[36,]  68  0.85659459999999998381 0.01477120000
[37,]  74  0.86430919999999999970 0.01345897000
[38,]  79  0.87184329999999998773 0.01226331000
[39,]  82  0.87902919999999995504 0.01117387000
[40,]  86  0.88572139999999999205 0.01018122000
[41,]  90  0.89220610000000000195 0.00927674500
[42,]  90  0.89839329999999995024 0.00845262500
[43,]  93  0.90406350000000001987 0.00770171700
[44,]  98  0.90956130000000001701 0.00701751700
[45,] 105  0.91467229999999999368 0.00639410100
[46,] 108  0.91962900000000002976 0.00582606600
[47,] 111  0.92419640000000002900 0.00530849500
[48,] 117  0.92845639999999995950 0.00483690300
[49,] 117  0.93243909999999996518 0.00440720500
[50,] 119  0.93611529999999998353 0.00401568200
```

R (glmnet) : Optimum value at 0.0003

LASSO – FINDING THE BEST LAMBDA



Python (SKLearn) : Plot of λ with MSE



R (glmnet) : Plot of log(λ) with MSE

LASSO – CONFUSION TABLE

Python (SKLearn) : Accuracy
91.6186 %

	WALKING	WALKING_UPSTAIRS	WALKING_DOWNSTAIRS	SITTING	STANDING	LAYING
WALKING	537	1	0	0	0	0
WALKING_UPSTAIRS	0	430	38	0	0	2
WALKING_DOWNSTAIRS	0	57	494	0	0	0
SITTING	0	0	0	490	2	16
STANDING	0	0	0	6	402	9
LAYING	0	3	0	0	16	444

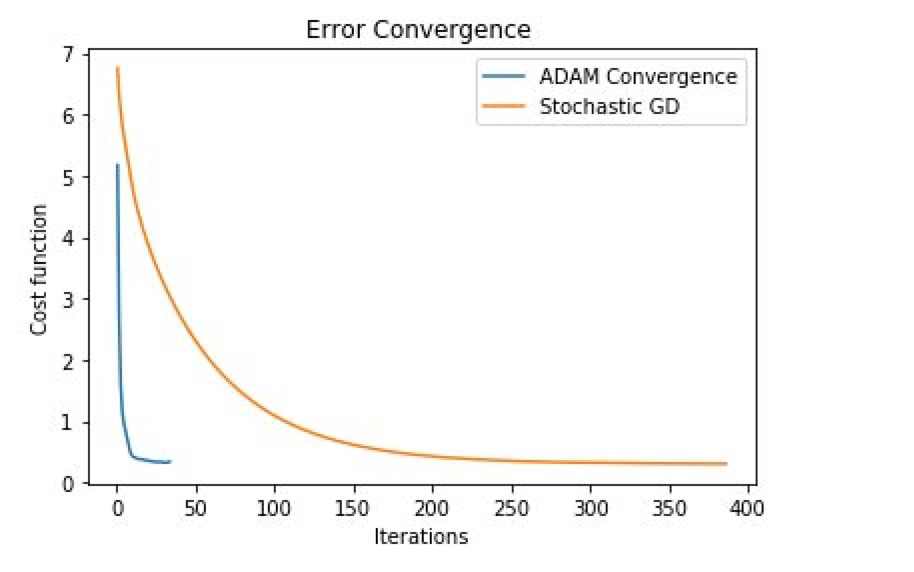
R (glmnet) : Accuracy 94.9440 %

max_Levels	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
LAYING	535	0	0	0	0	0
SITTING	0	428	13	0	0	1
STANDING	2	60	518	0	0	0
WALKING	0	0	1	494	4	34
WALKING_DOWNSTAIRS	0	0	0	1	391	4
WALKING_UPSTAIRS	0	3	0	1	25	432

NEURAL NETWORKS

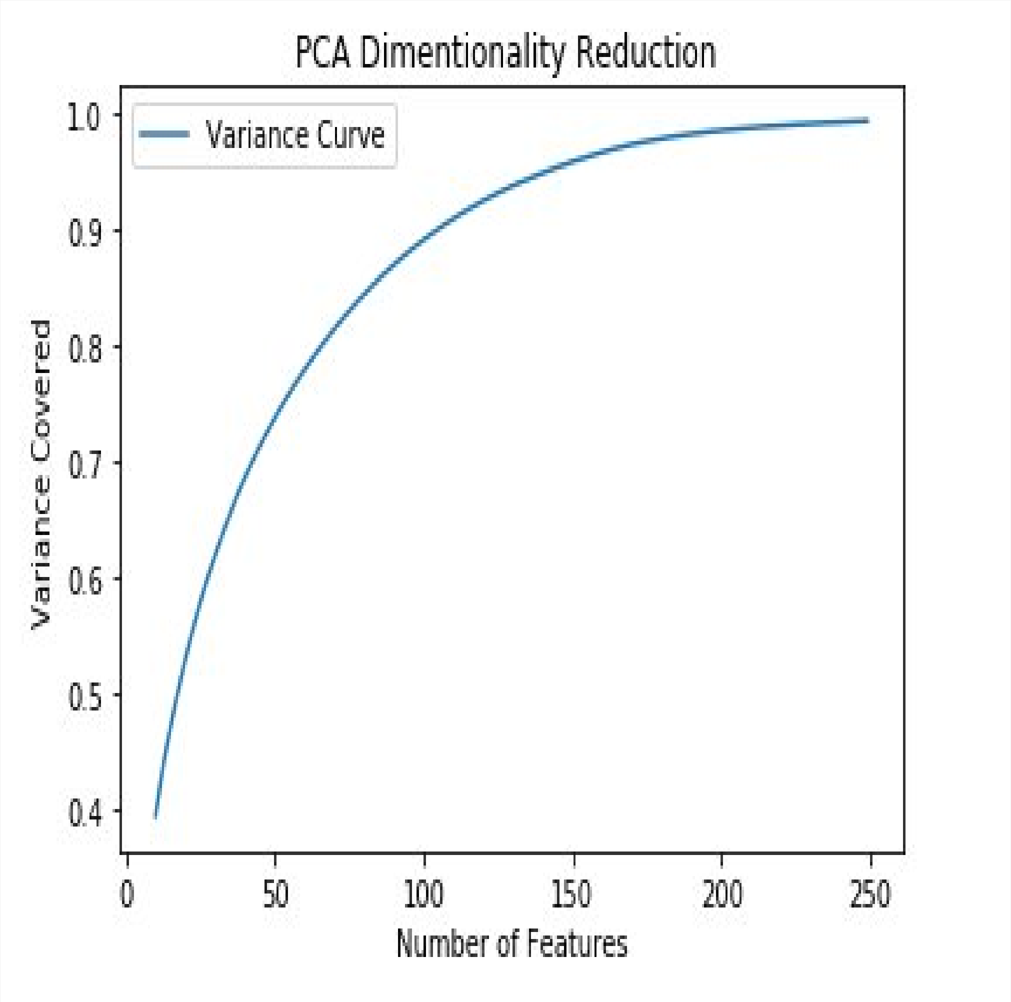
	WALKING	WALKING_UPSTAIRS	WALKING_DOWNSTAIRS	SITTING	STANDING	LAYING
WALKING	531	4	0	0	0	0
WALKING_UPSTAIRS	0	467	70	0	0	0
WALKING_DOWNSTAIRS	6	19	462	0	0	0
SITTING	0	0	0	487	3	4
STANDING	0	0	0	7	399	1
LAYING	0	1	0	2	18	466

Accuracy – 91.4191 % using Python

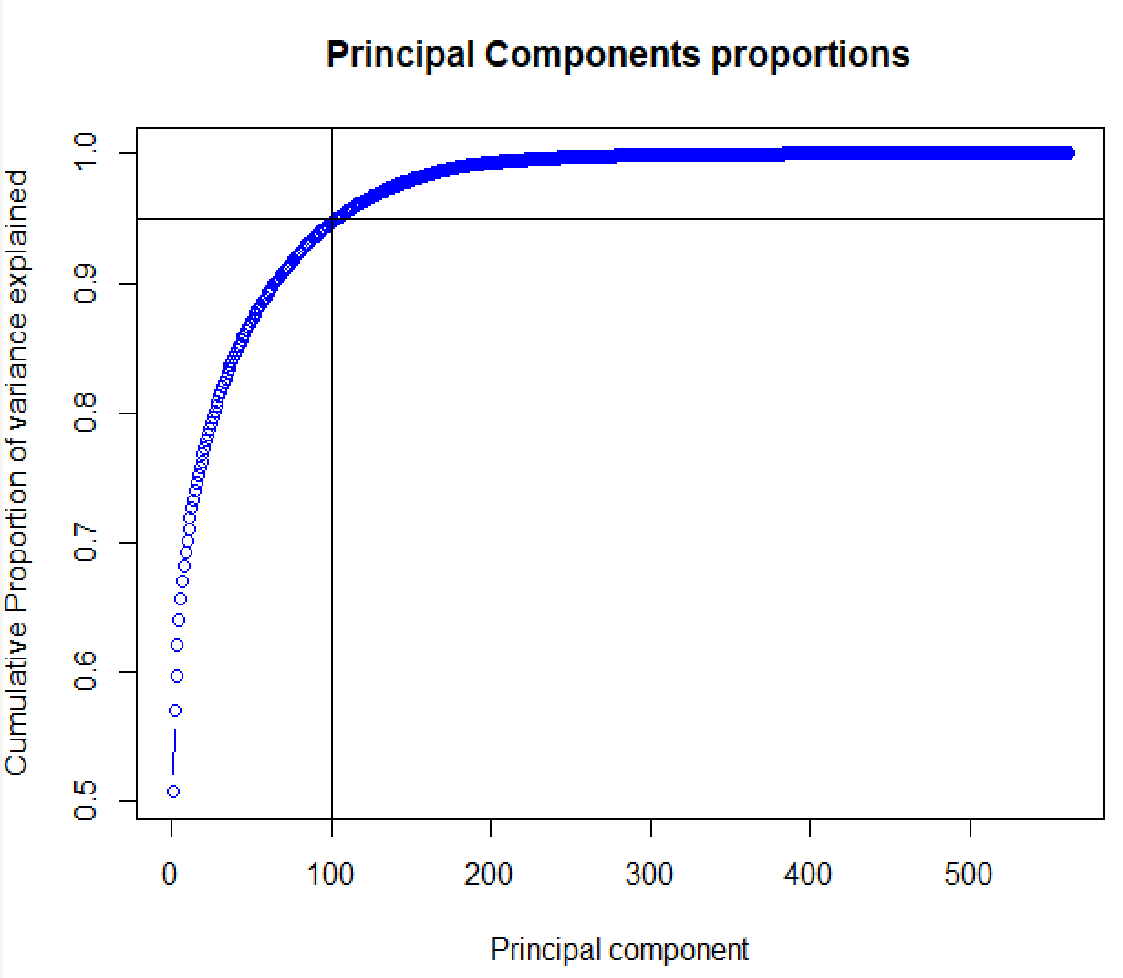


Error convergence using Stochastic Gradient Descent and ADAM Convergence using Python

FUTURE WORK : PCA



PCA using Python – Around 100 features cover 90 % of variance



PCA using R – Upto 100 features cover 95 % of variance

LEARNINGS AND CONCLUSIONS

Naïve Bayes does not perform well for this dataset. It may be because of the linear dependence of some columns

Even though Logistic regression without any reduction of features performs slightly better, we may want to go with a reduced model such as Lasso regression for better interpretability

We picked Lasso regression over Ridge regression because it shrinks some coefficients to 0

Principal Component Analysis shows that around 80 to 100 components are enough to explain about 95 % of variance

THANK YOU