PCA

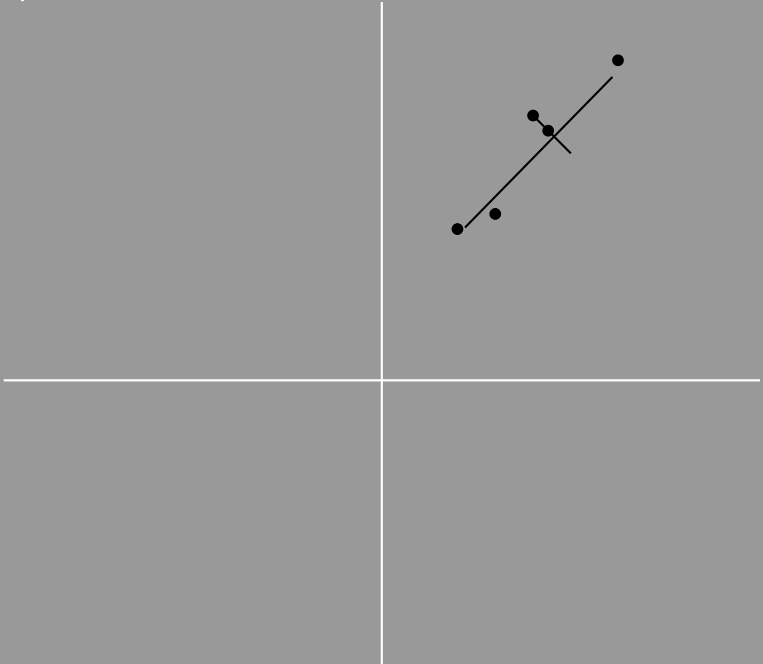
Martin Varbanov 165 Statistics

Table of Content

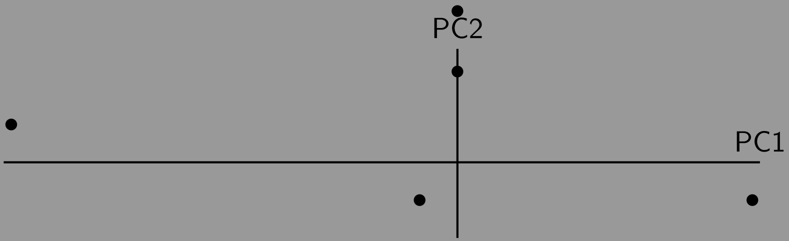
What is PCA

- PCA is a method for dimension reduction
- principal component analysis is used to select the features with the most information

Sample



Sample



Sample

x	y	$PC1$	$PC2$
x_1	y_1	$PC1_1$	$PC2_1$
x_2	y_2	$PC1_2$	$PC2_2$
...	...	....	...

- x and y are features
- (x*PC1*,*x*PC2) and (y*PC1*, *y*PC2) are our new points
- If we had 200 dimensional data, we'd be able to convert it to 200 points

DATA

- Source:
  https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State#
- 19 features from Electroencephalography sensors; 1 feature representing if eye is opened or closed

EEG

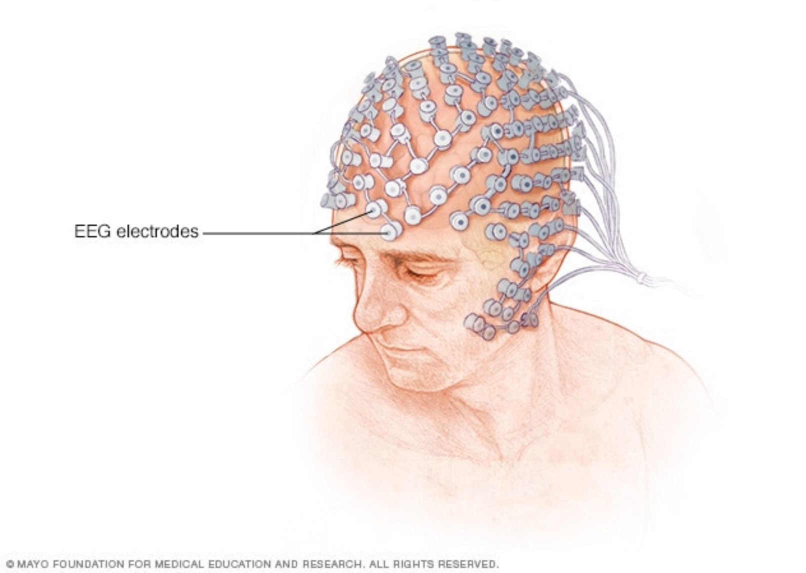


Figure 1: Electroencephalography

logistic regression on FUll data

- naive_model = glm.fit <- glm(result ~ X1+X2+X3+X4+X5+X6+X7+X8+X9+X10+X11+X12+X13+X14+ data = raw_data)
- table(naive_results,true_results)

	0	1
0	87	49
1	20	75

- We got 70% succsses rate

FULL PCA

- First We do A full PCA on all features(except the result feature)
- We get Variance of the factors is too smalls and we must use 5 PCs to reach 80% variance
- PC1 has Var of 18% which is too small

Plot Full PCA

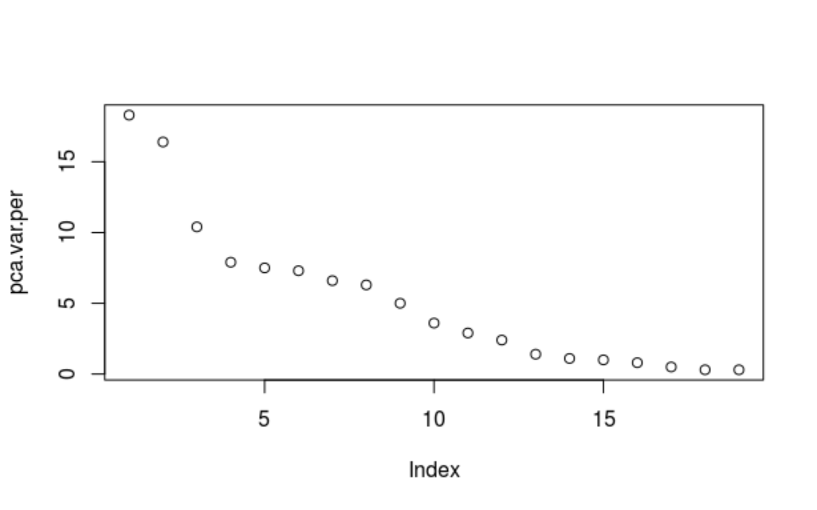
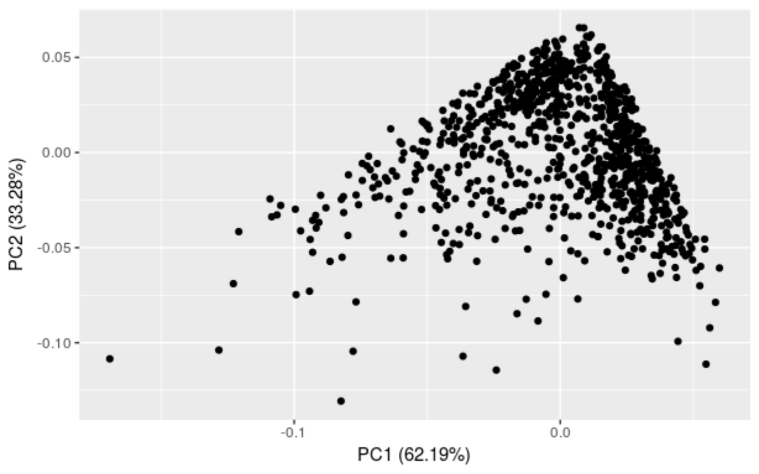


Figure 2: Percentage Full PCA

Plotting PCA

- library(ggfortify)
- autoplot(prcomp(train))



PCA by corelation

- Lets select our top 5 features with biggest corelation towards the wanted output

Corelation between eye state and features

cor_table["result",]

[0.0628162508; -0.0769254158; 0.2926029110; 0.2663378517; 0.2346910985; 0.1975108984; 0.1616306742; 0.1278607878; 0.0580148247; 0.0004790855; 0.0382814237; 0.1042544888; 0.1422728633; 0.1514243876; 0.1847720945; 0.1773128303; 0.0084663109; -0.0308676736; -0.0421439497; 1.0000000000]

PCA with hight correlation with result

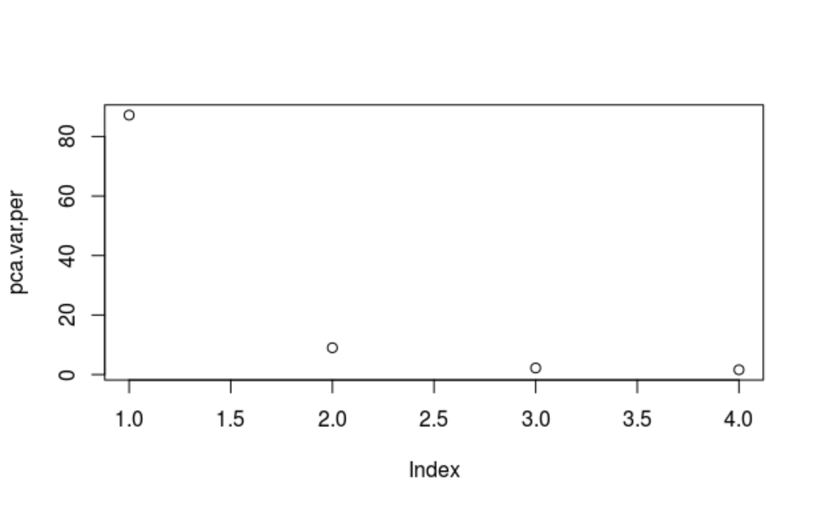


Figure 3: Percentage of

Results

- In this case PC1 accounts for more than 80% of variances # Selecting features with maximum magnetude

loading_scores <- full_pca_train$rotation[,1] ## We get the eigen vector of PC1
sensor_scores <- abs(loading_scores) ## get the magnitudes
sensor_score_ranked <- sort(sensor_scores, decreasing=TRUE)
top_10_sensors <- names(gene_score_ranked[1:10])
top_10_sensors
[1] "X6" "X5" "X7" "X4" "X3" "X8" "X19" "X9" "X10" "X16"

But if we want to use them in a logistic model, how do we choose how man of them to choose

Let's pick all of them

```
full_magnetude_model = glm.fit <- glm(result ~
X3+X4+X5+X6+X7+X8+X9+X10+X16+X19, data = train)
full_magnetude_predict = predict(full_magnetude_model,
newdata=test)
full_magnetude_results <- ifelse(full_magnetude_predict > 0.5, 1,
0)
true_results = test$result
```

Results

- table(full_magnetude_results,true_results)
- $\begin{bmatrix} & 0 & 1 \\ 0 & 86 & 53 \\ 1 & 21 & 71 \end{bmatrix}$
- We get 68% succsses rate, which is worser
- If we pick only the best features, we get a result which is weaker than if we take all the subjects

Lets try for top 5 factors

- full_magnetude_model = glm.fit <- glm(result ~ X6+X5+X7+X4+X3, data = train)
- ...
- $\begin{array}{cc} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} 94 & 59 \\ 13 & 65 \end{bmatrix} \end{array}$
- We get 69% succsses rate which is a middle result

Lets try for 4

- full_magnetude_model = glm.fit <- glm(result ~ X6+X5+X7+X4, data = train)
- ...
- $$\begin{bmatrix} & 0 & 1 \\ 0 & 96 & 58 \\ 1 & 11 & 66 \end{bmatrix}$$
- We get 70% succsses rate which is equivelent to a full regression prediction