Dimensionality reduction

Why reduce the number of features in a data set?

1. It reduces storage and computation time.
2. High-dimensional data often has a lot of redundancy.
3. Remove noisy or irrelevant features.

Dimensionality reduction

1. PCA
2. t-SNE
3. Singular Value Decomposition
4. Restricted Boltzmann Machines
5. High correlation
6. Low Variance
7. Using ML model (Forward Selection, Backward elimination, RF etc.)

Principal Components Analysis

- PCA produces a low-dimensional representation of a dataset. It finds a sequence of linear combinations of the variables that have maximal variance, and are mutually uncorrelated.
- Apart from producing derived variables for use in supervised learning problems, PCA also serves as a tool for data visualization.

Principal Components Analysis: details

- The first principal component of a set of features $X_1, X_2, \ldots, X_p$ is the normalized linear combination of the features

$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \ldots + \phi_{p1}X_p$

that has the largest variance. By normalized, we mean that,

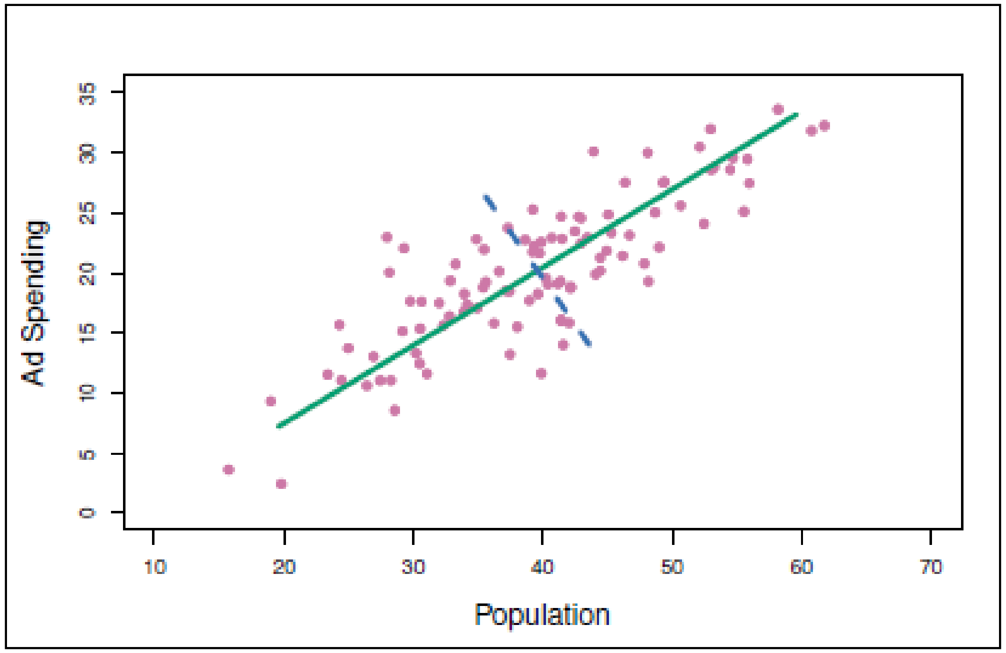
$\sum_{j=1}^{p} \phi_{j1}^2 = 1$

- We refer to the elements $\phi_{11}, \ldots, \phi_{p1}$ as the loadings of the first principal component; together, the loadings make up the principal component loading vector,

$\phi_1 = (\phi_{11}, \phi_{21}, \ldots, \phi_{p1})^T$.

- We constrain the loadings so that their sum of squares is equal to one, since otherwise setting these elements to be arbitrarily large in absolute value could result in an arbitrarily large variance.

PCA: example



The population size (pop) and ad spending (ad) for 100 different cities are shown as purple circles. The green solid line indicates the first principal component direction, and the blue dashed line indicates the second principal component direction.

Computation of Principal Components

- Suppose we have a n p data set X. Since we are only interested in variance, we assume that each of the variables in X has been centered to have mean zero (that is, the column means of X are zero).
- We then look for the linear combination of the sample feature values of the form

$z_{i1} = \phi_{11}X_{i1} + \phi_{21}X_{i2} + \ldots + \phi_{p1}X_{ip}$ (1)

for i = 1, 2, ..., n that has largest sample variance, subject to the constraint that, $\sum_{j=1}^{p} \phi_{j1}^2 = 1$

- Since each of the x_{ij} has mean zero, then so does z_{i1} (for any values of ϕ_{j1}). Hence the sample variance of the z_{i1} can be written as $\frac{1}{n}\sum_{i=1}^{n} z_{j1}^2$

Computation: continued

- Plugging in (1) the first principal component loading vector solves the optimization problem-

$$\boxed{\underset{\phi_{11},\ldots,\phi_{p1}}{\text{maximize}} \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{j1}x_{ij}\right)^{2} \text{ subject to } \sum_{j=1}^{p}\phi_{j1}^{2}=1.}$$

- This problem can be solved via a singular-value decomposition of the matrix X, a standard technique in linear algebra.
- We refer to Z_1 as the first principal component, with realized values $z_{11}, \ldots, z_{n1}$.

Geometry of PCA

- The loading vector ϕ_1 with elements $(\phi_{11}, \phi_{21}, ..., \phi_{p1})$ defines a direction in feature space along which the data vary the most.
- If we project the n data points $x_1,..., x_n$ onto this direction, the projected values are the principal component scores $z_{11}, ... , z_{n1}$ themselves.

Further principal components

- The second principal component is the linear combination of X_1, ... , X_p that has maximal variance among all linear combinations that are *uncorrelated* with Z_1.

- The second principal component scores $z_{12}, z_{22}, ..., z_{n2}$ take the form

$$z_{i2} = \phi_{12}x_{i1} + \phi_{22}x_{i2} + ... + \phi_{p2}x_{ip},$$

where 2 is the second principal component loading vector, with elements $\phi_{12}, \phi_{22}, ..., \phi_{p2}$.

- It turns out that constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction ϕ_2 to be orthogonal (perpendicular) to the direction ϕ_1. And so on.

PCA finds the hyper-plane closest to the observations

- PCA find the hyper-plane closest to the observations The first principal component loading vector has a very special property: it defines the line in p-dimensional space that is *closest* to the n observations (using average squared Euclidean distance as a measure of closeness)
- The notion of principal components as the dimensions that are closest to the n observations extends beyond just the first principal component.
- For instance, the first two principal components of a dataset span the plane that is closest to the n observations, in terms of average squared Euclidean distance.

Scaling of the variables matters

- If the variables are in different units, scaling each to have standard deviation equal to one is recommended.
- If they are in the same units, you might or might not scale the variables.

Proportion Variance Explained

- To understand the strength of each component, we are interested in knowing the proportion of variance explained (PVE) by each one.

- The total variance present in a data set (assuming that the variables have been centered to have mean zero) is defined as

$$\sum_{j=1}^{p} \mathrm{Var}(X_j) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2,$$

and the variance explained by the *mth* principal component is

$$\mathrm{Var}(Z_m) = \frac{1}{n} \sum_{i=1}^{n} z_{im}^2.$$

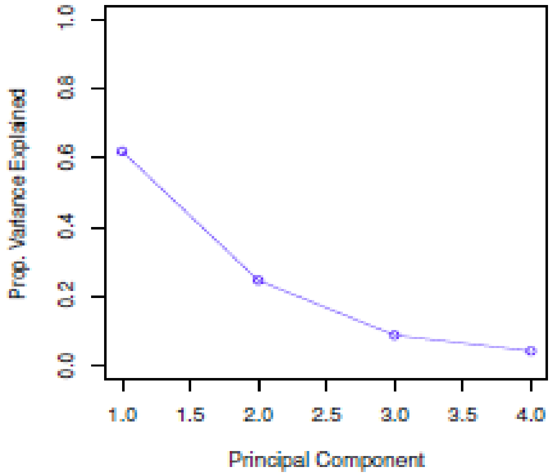
- It can be shown that $\sum_{j=1}^{p} \mathrm{Var}(X_j) = \sum_{m=1}^{M} \mathrm{Var}(Z_m)$, with M = min(n-1, p).

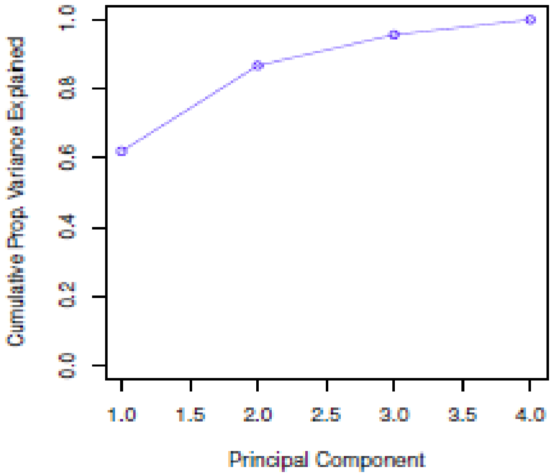
Proportion Variance Explained: continued

- Therefore, the PVE of the *m*th principal component is given by the positive quantity between 0 and 1

$$\boxed{\frac{\sum_{i=1}^{n} z_{im}^2}{\sum_{j=1}^{p}\sum_{i=1}^{n} x_{ij}^2}.}$$

- The PVEs sum to one. We sometimes display the cumulative PVEs.





- These are called scree plots.

How many principal components should we use?

- If we use principal components as a summary of our data, how many components are sufficient?
  - No simple answer to this question, as cross-validation is not available for this purpose.

    When could we use cross-validation to select the number of components?
- The “scree plot” on can be used as a guide: we look for an “elbow".

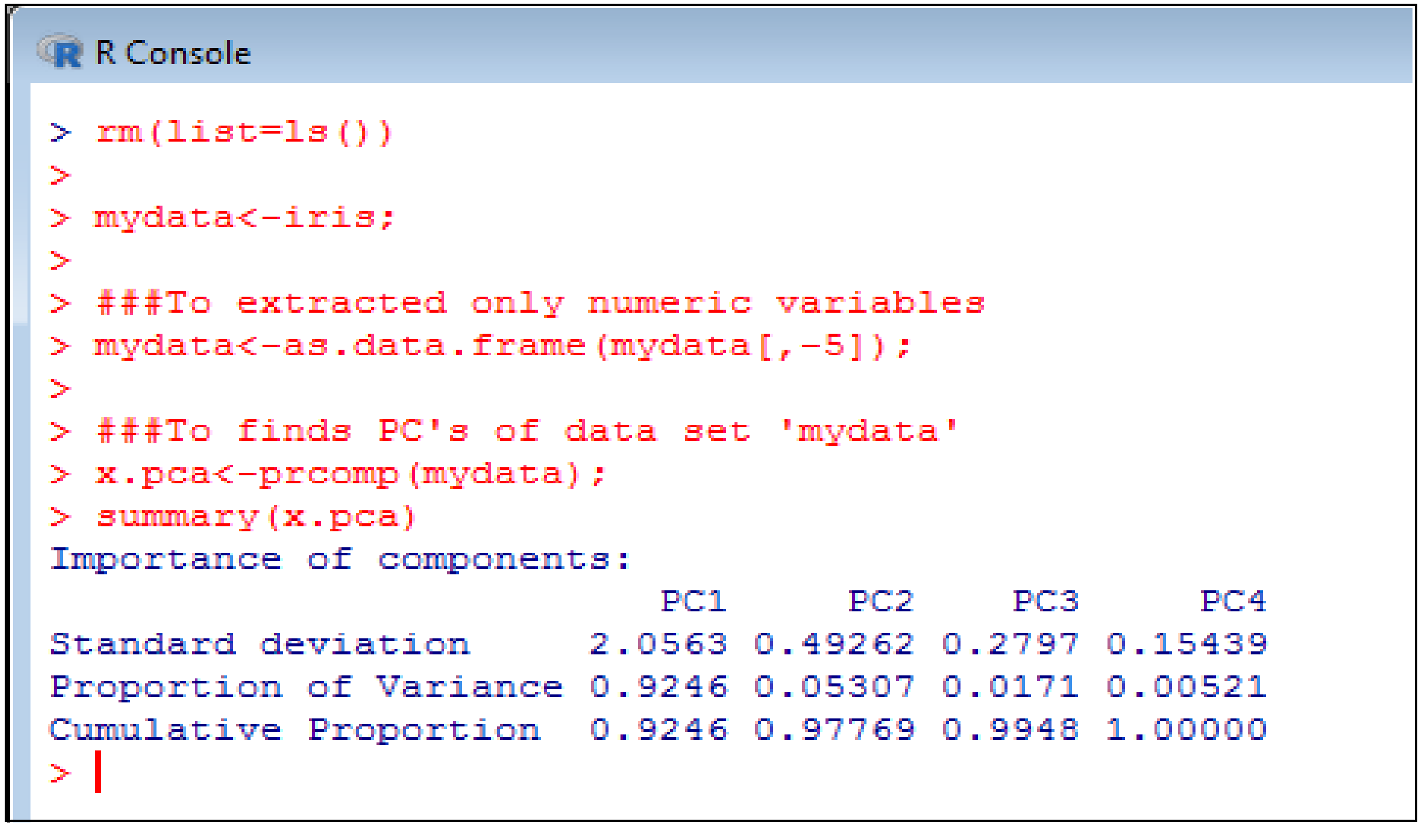
PCA in few steps…(without any package)

- Get the data
- Find variance covariance matrix
- Find Eigen value – Eigen vectors of var-cov matrix
- Select few Eigen values in decreasing order and corresponding Eigen vectors.
- Transform original data using Eigen vectors, the resulting columns are Principal Components.

R- Session on PCA

PCA in R

- For illustration of PCA, we consider ‘iris’ data in R

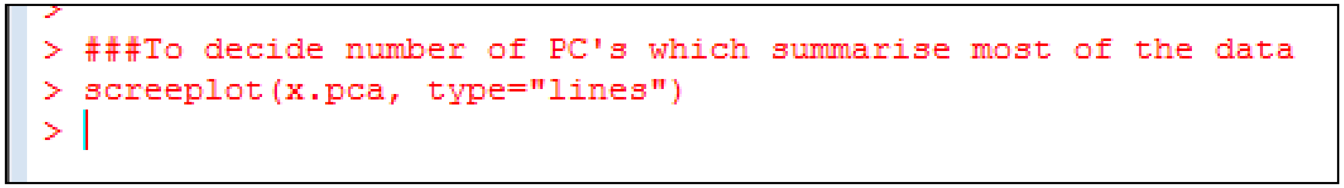


R Console

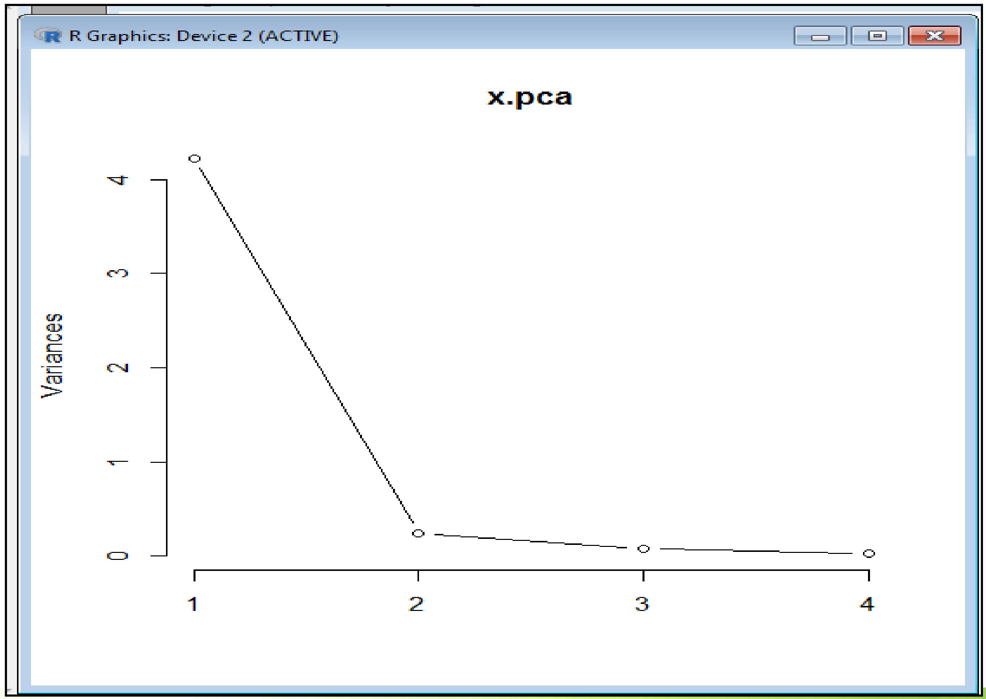
```
> rm(list=ls())
>
> mydata<-iris;
>
> ###To extracted only numeric variables
> mydata<-as.data.frame(mydata[,-5]);
>
> ###To finds PC's of data set 'mydata'
> x.pca<-prcomp(mydata);
> summary(x.pca)
Importance of components:
                          PC1     PC2    PC3     PC4
Standard deviation     2.0563 0.49262 0.2797 0.15439
Proportion of Variance 0.9246 0.05307 0.0171 0.00521
Cumulative Proportion  0.9246 0.97769 0.9948 1.00000
> 
```

PCA in R...

- Scree plot

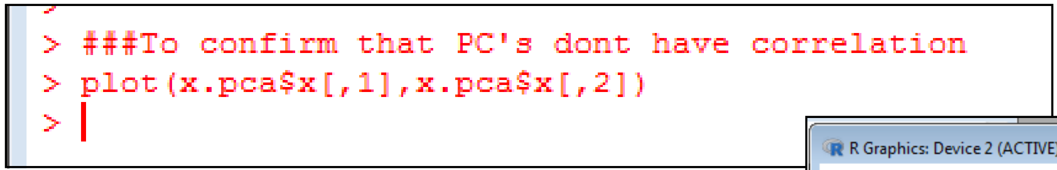


```
> 
> ###To decide number of PC's which summarise most of the data
> screeplot(x.pca, type="lines")
> 
```

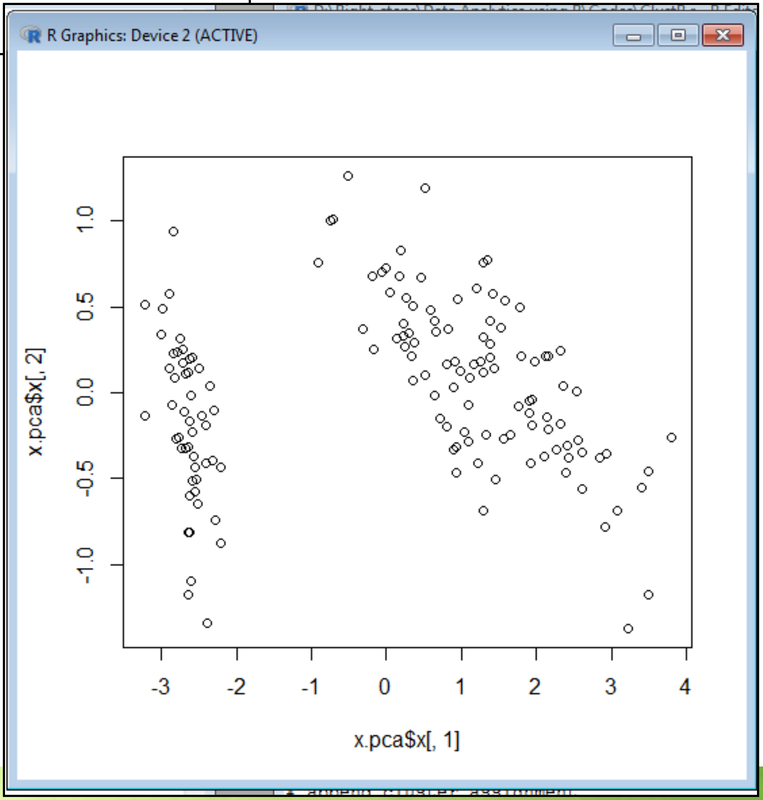


PCA in R...

- Lets plot pc’s to confirm that, they are independent.

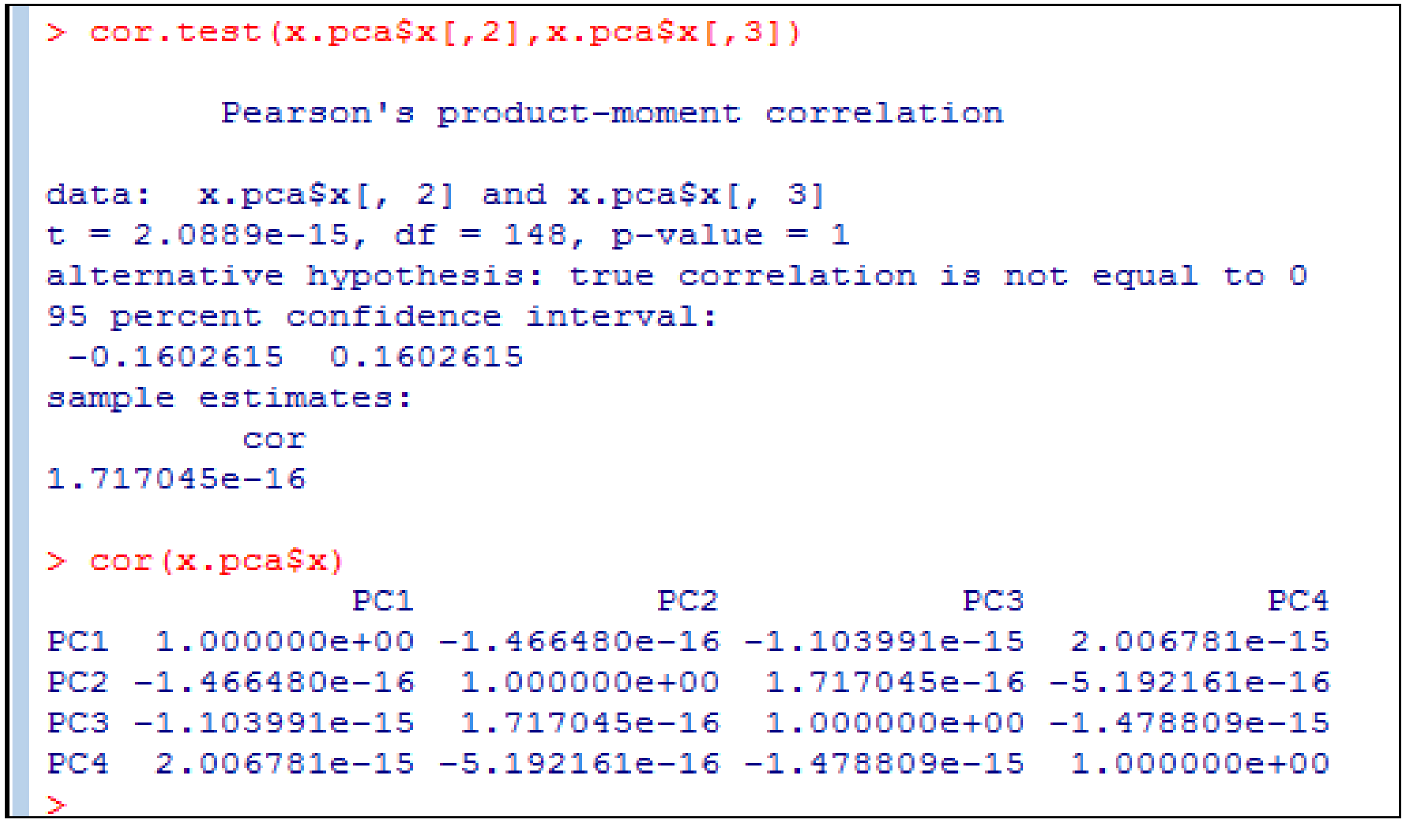


```
> ###To confirm that PC's dont have correlation
> plot(x.pca$x[,1],x.pca$x[,2])
>
```



PCA in R...

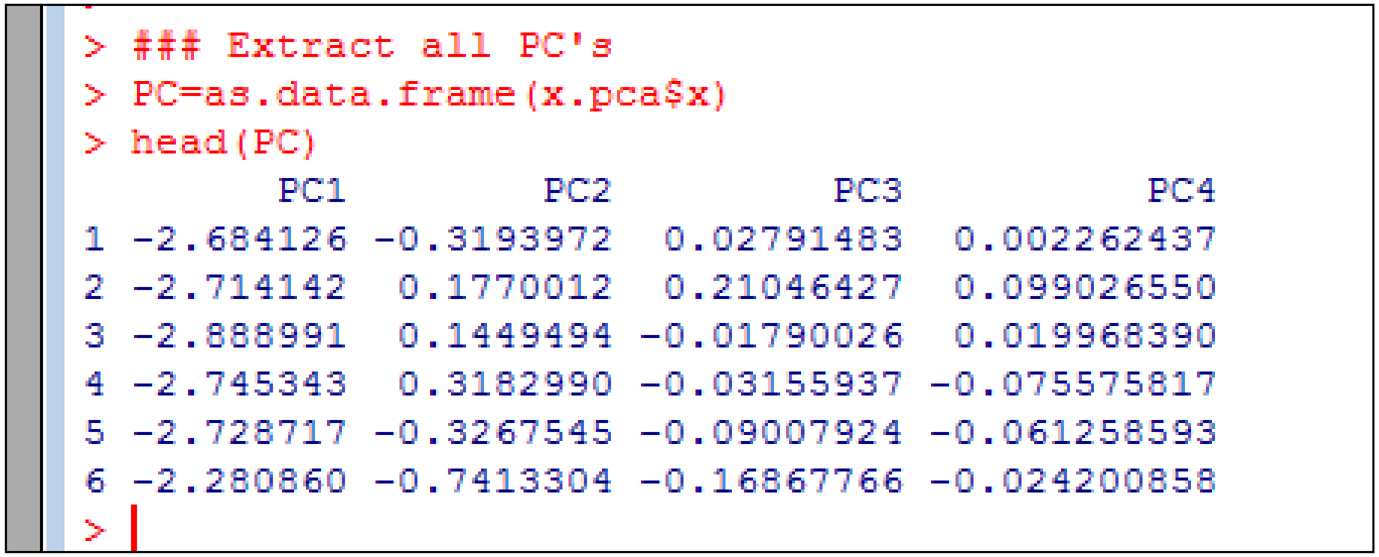
- Correlation test and matrix



```
> cor.test(x.pca$x[,2],x.pca$x[,3])

        Pearson's product-moment correlation

data:  x.pca$x[, 2] and x.pca$x[, 3]
t = 2.0889e-15, df = 148, p-value = 1
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.1602615  0.1602615
sample estimates:
         cor
1.717045e-16

> cor(x.pca$x)
              PC1           PC2           PC3           PC4
PC1  1.000000e+00 -1.466480e-16 -1.103991e-15  2.006781e-15
PC2 -1.466480e-16  1.000000e+00  1.717045e-16 -5.192161e-16
PC3 -1.103991e-15  1.717045e-16  1.000000e+00 -1.478809e-15
PC4  2.006781e-15 -5.192161e-16 -1.478809e-15  1.000000e+00
>
```

PCA in R...

- We can extract PC's as a data frame for further use.



```
> ### Extract all PC's
> PC=as.data.frame(x.pca$x)
> head(PC)
        PC1        PC2         PC3          PC4
1 -2.684126 -0.3193972  0.02791483  0.002262437
2 -2.714142  0.1770012  0.21046427  0.099026550
3 -2.888991  0.1449494 -0.01790026  0.019968390
4 -2.745343  0.3182990 -0.03155937 -0.075575817
5 -2.728717 -0.3267545 -0.09007924 -0.061258593
6 -2.280860 -0.7413304 -0.16867766 -0.024200858
> 
```

THE END

t-SNE

Limitations of PCA

- PCA is a linear algorithm.

- Will not be able to interpret complex polynomial relationship between features.

t-SNE

T-distributed Stochastic Neighbour Embedding

- t-SNE is based on probability distributions with random walk on neighborhood graphs to find the structure within the data.
- Linear dimensionality reduction algorithms concentrates only on placing dissimilar data points far apart in a lower dimension representation.
- But in order to represent high dimension data on low dimension, non-linear manifold, it is important that similar data points must be represented close together, which is not what linear dimensionality reduction algorithms do.

t-SNE

t-SNE tries to find out low dimensional structure of high dimensional data by preserving similarity / nearness of data points.

- The similarity between two points x_i and x_j is the conditional probability $P(x_j \mid x_i)$ similar to high dimensional Euclidean distance.

- High conditional probability indicates similarity and low conditional probability wide separation of points.

- For high dimensional $\underline{X}$ with conditional probabilities $P(x_j \mid x_i)$, t-SNE finds low dimensional structure $\underline{Y}$ with same conditional probability structure $P(y_j \mid y_i)$.

t-SNE

The conditional probability $p(x_j \mid x_i)$ and $q(y_j \mid y_i)$

$$p_{j|i} = \frac{\exp(-\| x_i - x_j \|^2 / (2 * \sigma_i^2))}{\sum_{k \neq i} \exp(-\| x_i - x_k \|^2 / (2 * \sigma_i^2))}$$

Where σ_i^2 is the variance of Gaussian centred around x_i

$$q_{j|i} = \frac{\exp(-\| y_i - y_j \|^2)}{\sum_{k \neq i} \exp(-\| y_i - y_k \|^2)}$$

t-SNE

In order to find probabilities, we need to consider some neighbours around the same points. As σ_i^2 increases points around x_i becomes sparse which demands more data points to calculate probabilities. The rough number of points to be considered can be determined using function called ‘Perplexity’ which is directly proportional to σ_i^2

$$Perp(P_i) = 2^{H(p_i)}$$

Where $H(p_i)$ is Shannon Entropy

$$H(p_i) = -\sum_j p_{j|i} \log_2(p_{j|i})$$

t-SNE

The perplexity can be interpreted as a smooth measure of the effective number of neighbors. The performance of SNE is fairly robust to changes in the perplexity, and typical values are between 5 and 50.

t-SNE also tries to minimize the sum of the difference in conditional probabilities. It uses Gradient descent to minimize cost function i.e. sum of the difference in conditional probabilities.

t-SNE in R

t-SNE in R

Singular Value Decomposition

- The conventional method for calculating PCA requires us to compute the full covariance matrix, and so it suffers from memory complexity and can be numerically unstable.
- It's possible to perform PCA by using SVD which is a purely numerical technique based on simple linear algebra - thus more stable and can be optimized for speed.

Singular Value Decomposition

Single value decomposition of a matrix.

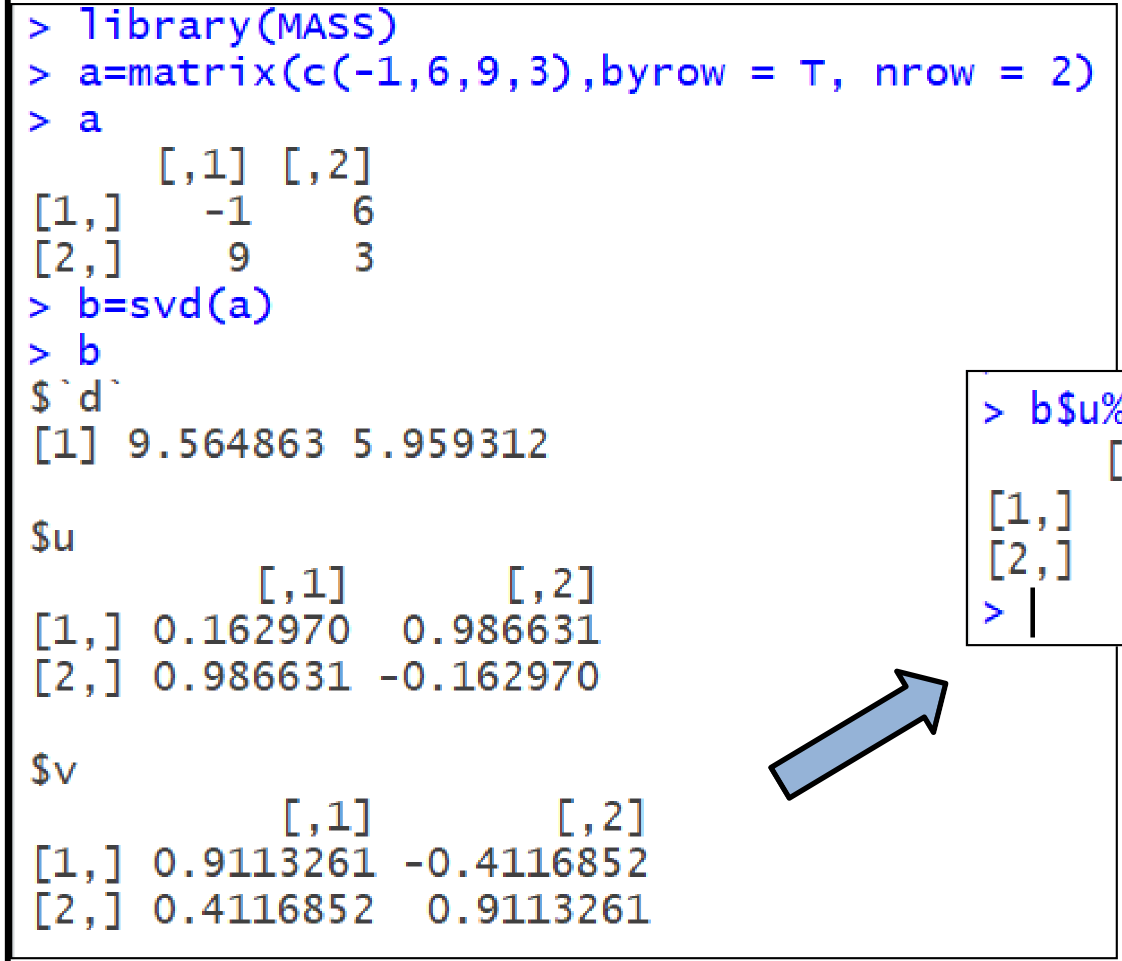
Singular value decomposition is a method of decomposing a matrix into three other matrices

$$A = USV^T$$

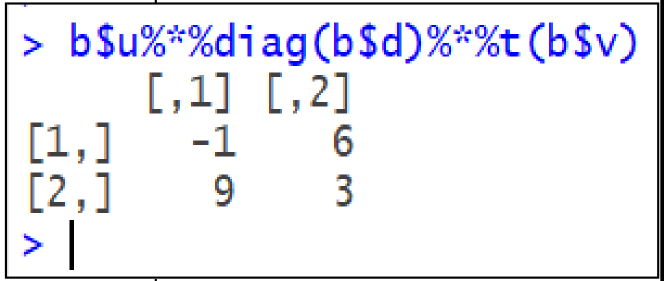
Where,

- A is an arbitrary m∗n matrix (for our purposes we will assume the values are all real),
-U is a m∗m orthonormal matrix of 'left-singular' (eigen) vectors of AA^T
-V^T is a n∗n orthonormal matrix of 'right-singular' (eigen) vectors of A^TA
-S is a m∗n diagonal matrix of the square root of nonzero eigen values of U *or* V, ordered by decreasing size.

Singular Value Decomposition



```
> library(MASS)
> a=matrix(c(-1,6,9,3),byrow = T, nrow = 2)
> a
     [,1] [,2]
[1,]   -1    6
[2,]    9    3
> b=svd(a)
> b
$`d`
[1] 9.564863 5.959312

$u
         [,1]      [,2]
[1,] 0.162970  0.986631
[2,] 0.986631 -0.162970

$v
          [,1]       [,2]
[1,] 0.9113261 -0.4116852
[2,] 0.4116852  0.9113261
```

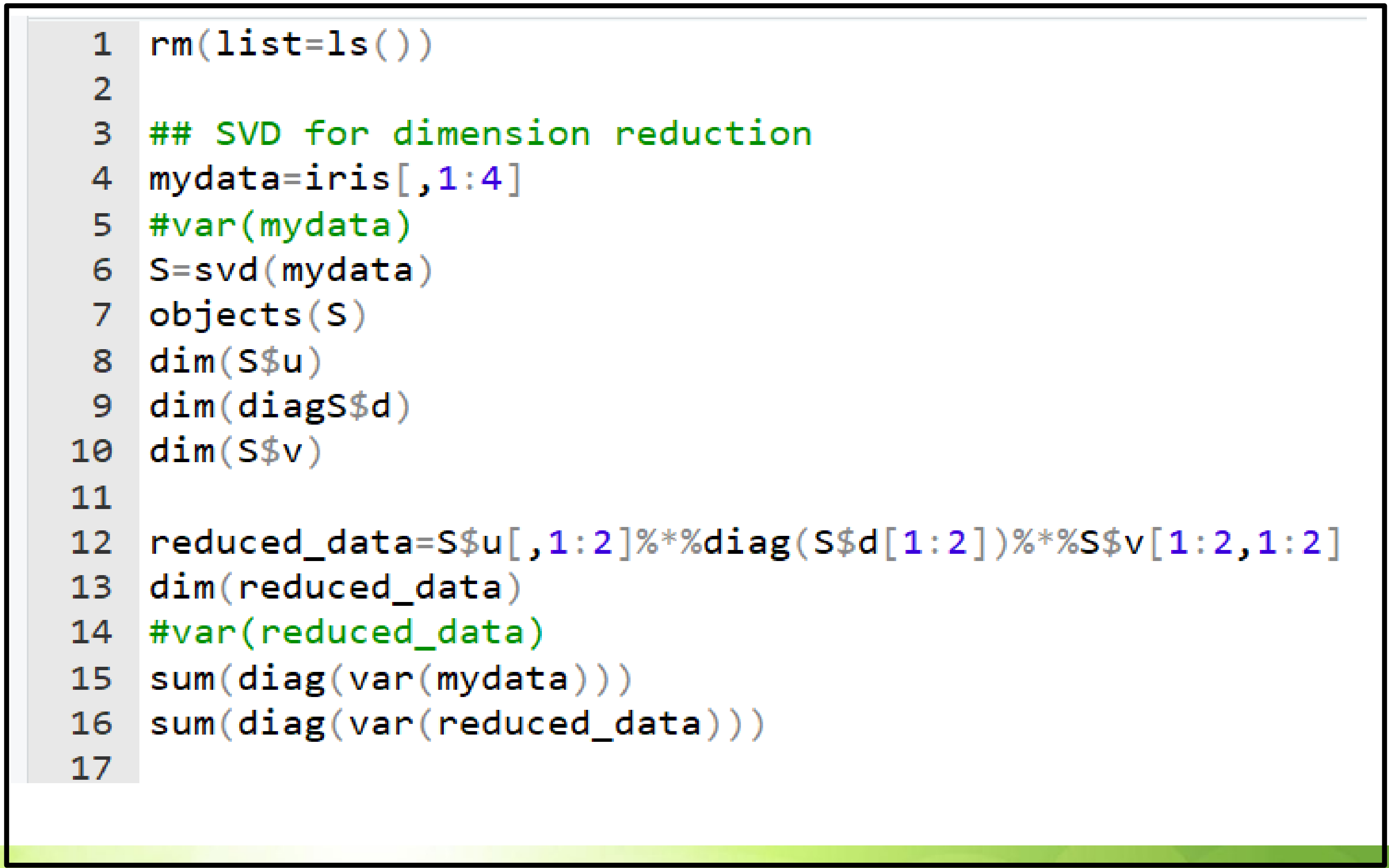


```
> b$u%*%diag(b$d)%*%t(b$v)
     [,1] [,2]
[1,]   -1    6
[2,]    9    3
> |
```

Singular Value Decomposition…steps for dimension reduction

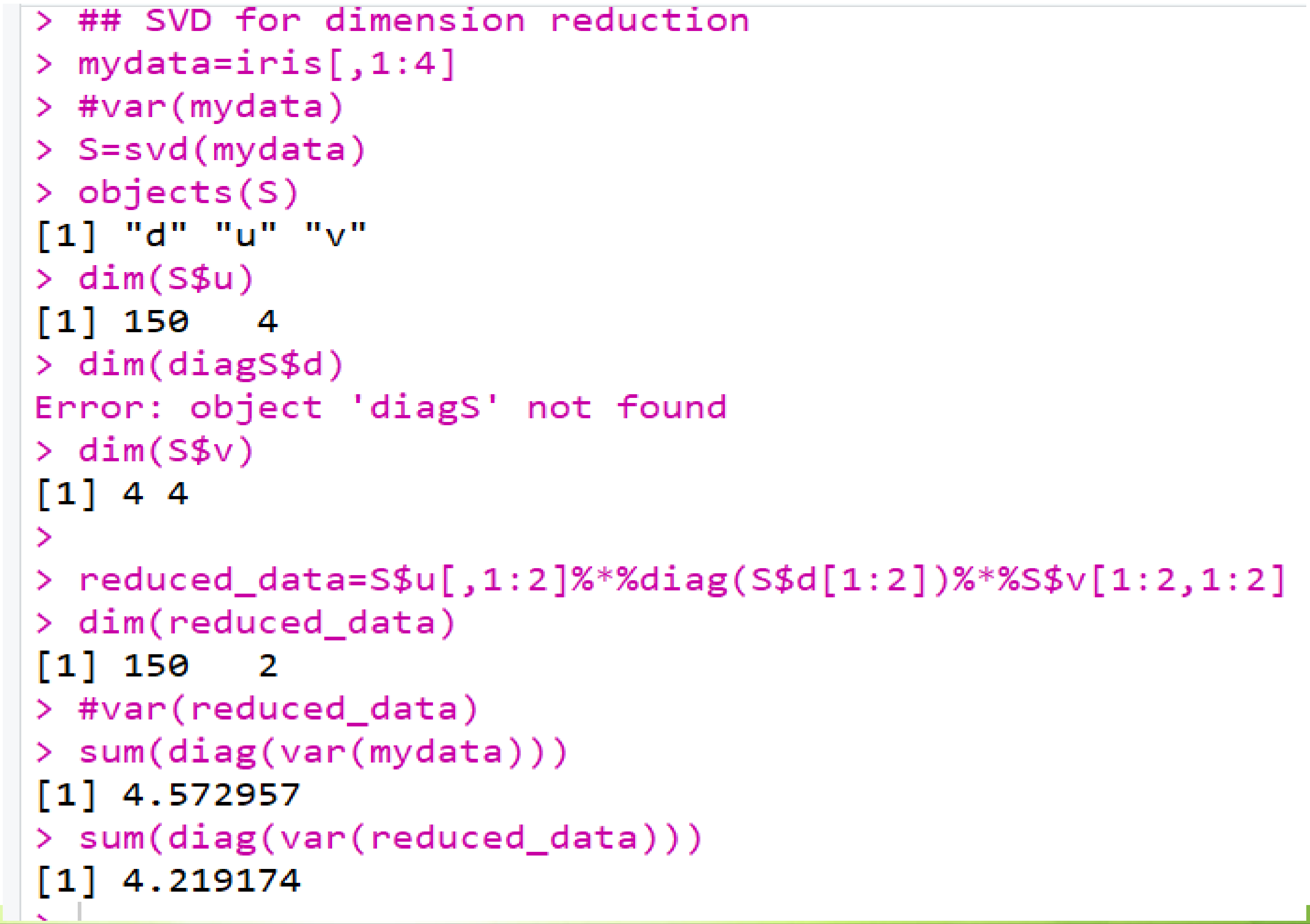
- Get data matrix A with n rows and P (say) columns
- Find SVD of the same USV^T
- Find top k (< P) eigen values from S above.
- Extract elements of USV^T such that resulting data matrix will have n rows and k columns

Singular Value Decomposition



```r
rm(list=ls())

## SVD for dimension reduction
mydata=iris[,1:4]
#var(mydata)
S=svd(mydata)
objects(S)
dim(S$u)
dim(diagS$d)
dim(S$v)

reduced_data=S$u[,1:2]%*%diag(S$d[1:2])%*%S$v[1:2,1:2]
dim(reduced_data)
#var(reduced_data)
sum(diag(var(mydata)))
sum(diag(var(reduced_data)))

```

Singular Value Decomposition



```
> ## SVD for dimension reduction
> mydata=iris[,1:4]
> #var(mydata)
> S=svd(mydata)
> objects(S)
[1] "d" "u" "v"
> dim(S$u)
[1] 150   4
> dim(diagS$d)
Error: object 'diagS' not found
> dim(S$v)
[1] 4 4
>
> reduced_data=S$u[,1:2]%*%diag(S$d[1:2])%*%S$v[1:2,1:2]
> dim(reduced_data)
[1] 150   2
> #var(reduced_data)
> sum(diag(var(mydata)))
[1] 4.572957
> sum(diag(var(reduced_data)))
[1] 4.219174
```

Restricted Boltzmann Machines

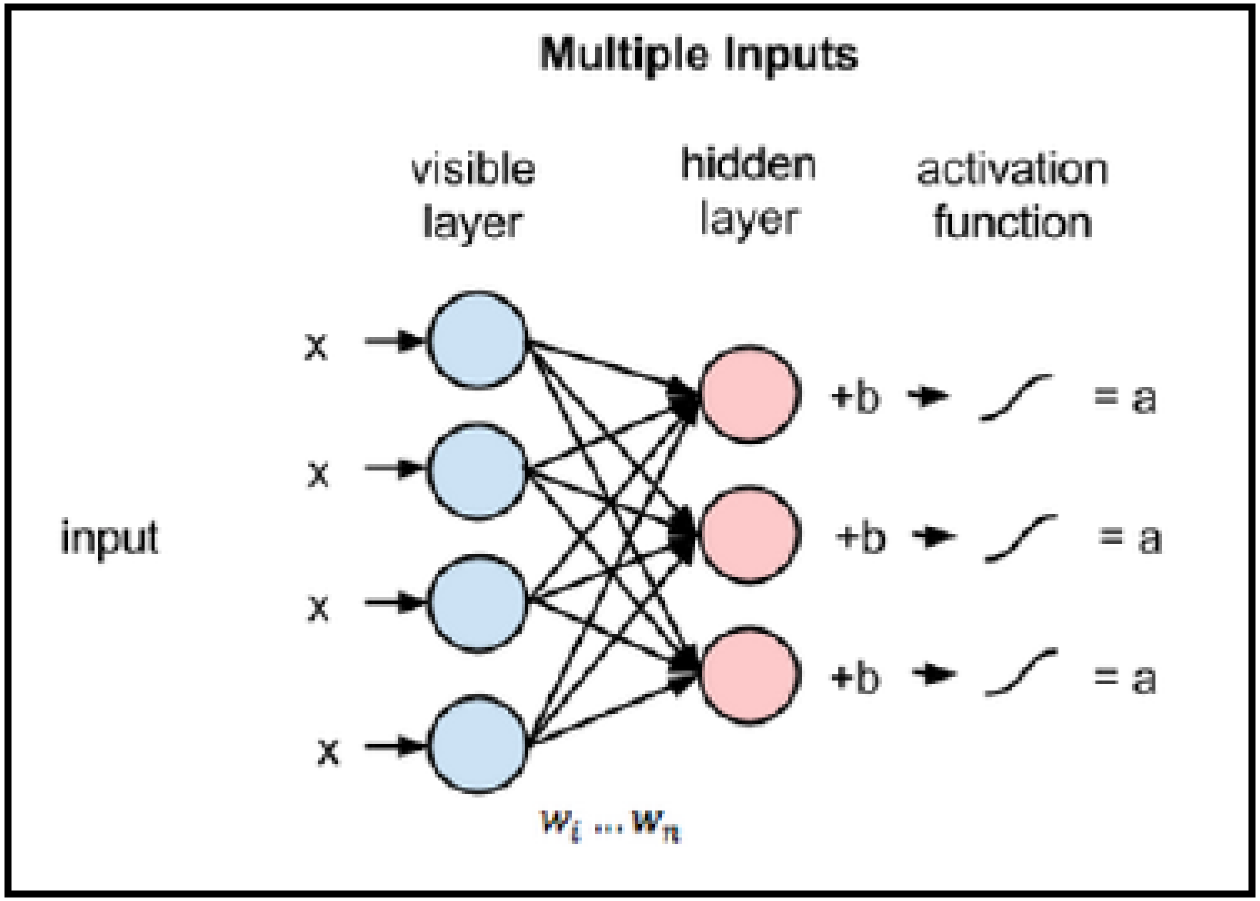
- RBMs are a family of unsupervised feature learning algorithms that use probabilistic models to learn new features.

- We can use RBMs to extract a new feature set from raw data and use them to enhance machine learning pipelines.

-The features that are extracted by RBMs tend to work best when followed by linear models such as linear regression, logistic regression, perceptron's, and so on.

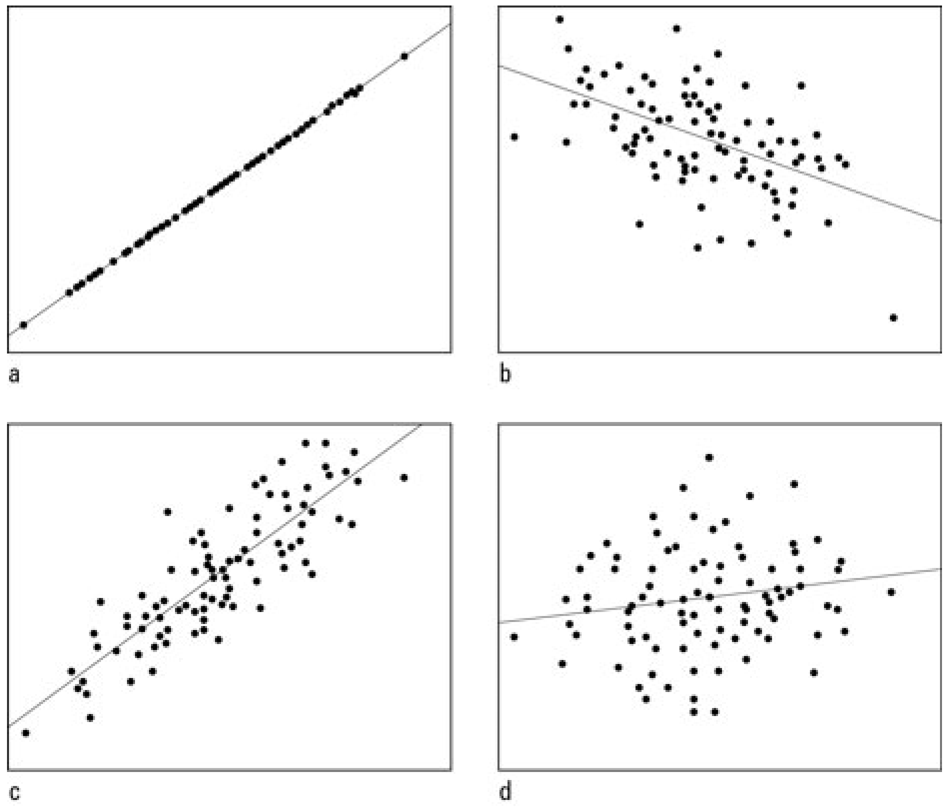
- Conceptually, RBMs are shallow (two-layer) neural networks.

Restricted Boltzmann Machines…in short



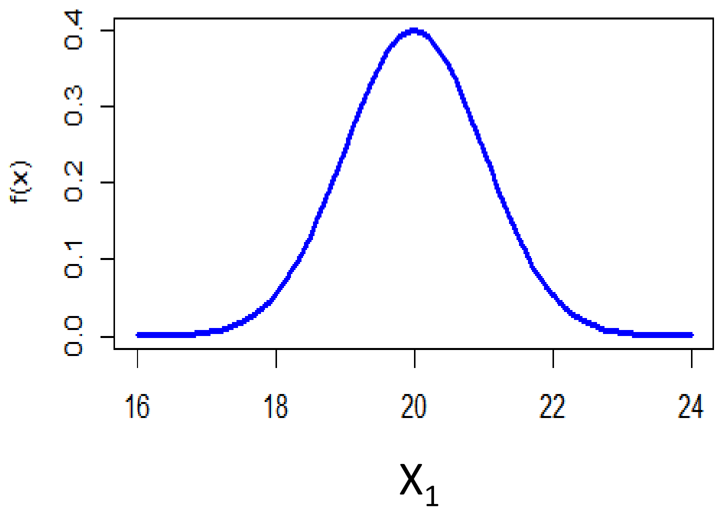
High correlation

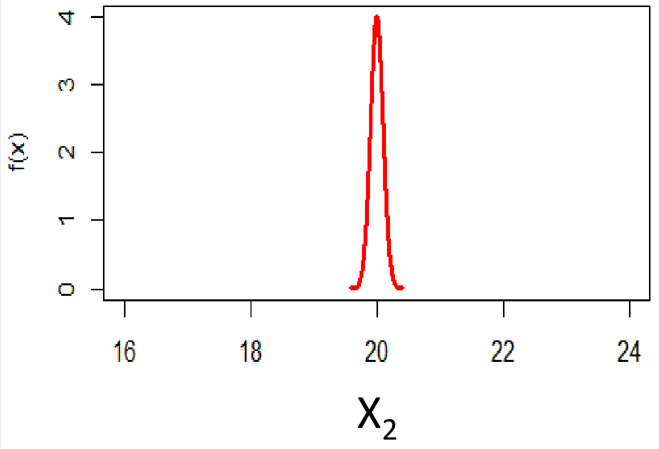
1. Interpret following diagram



Low Variance

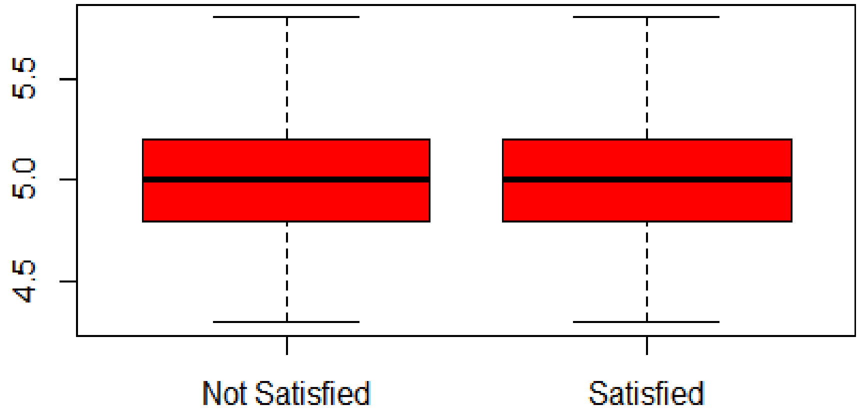
1. Low variance within ‘x’





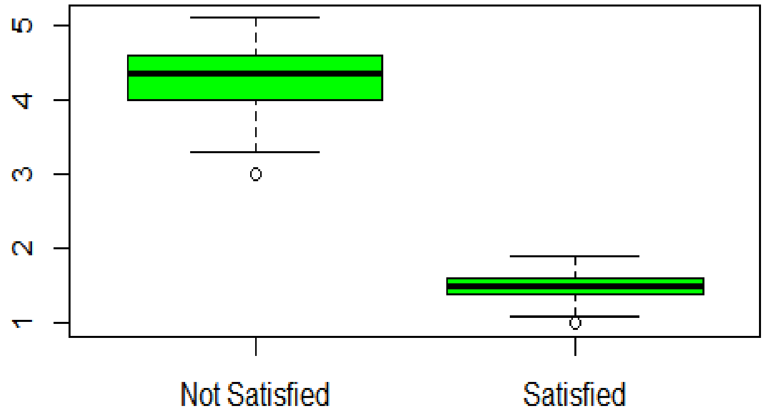
Low Variance

2. No significant variation with ‘y’



X_1

X_2



Using ML model (Forward Selection, Backward elimination, RF etc.)

Some ML models, calculates contribution of each ‘x’ to ‘y’ like Gini index, Entropy etc. Using such statistics, we can identify significant contributors for ‘y’ and remove those having negligible or no contribution to ‘y’.

The End