Decision trees
Dimensionality reduction

Lecture 2b

Decision trees

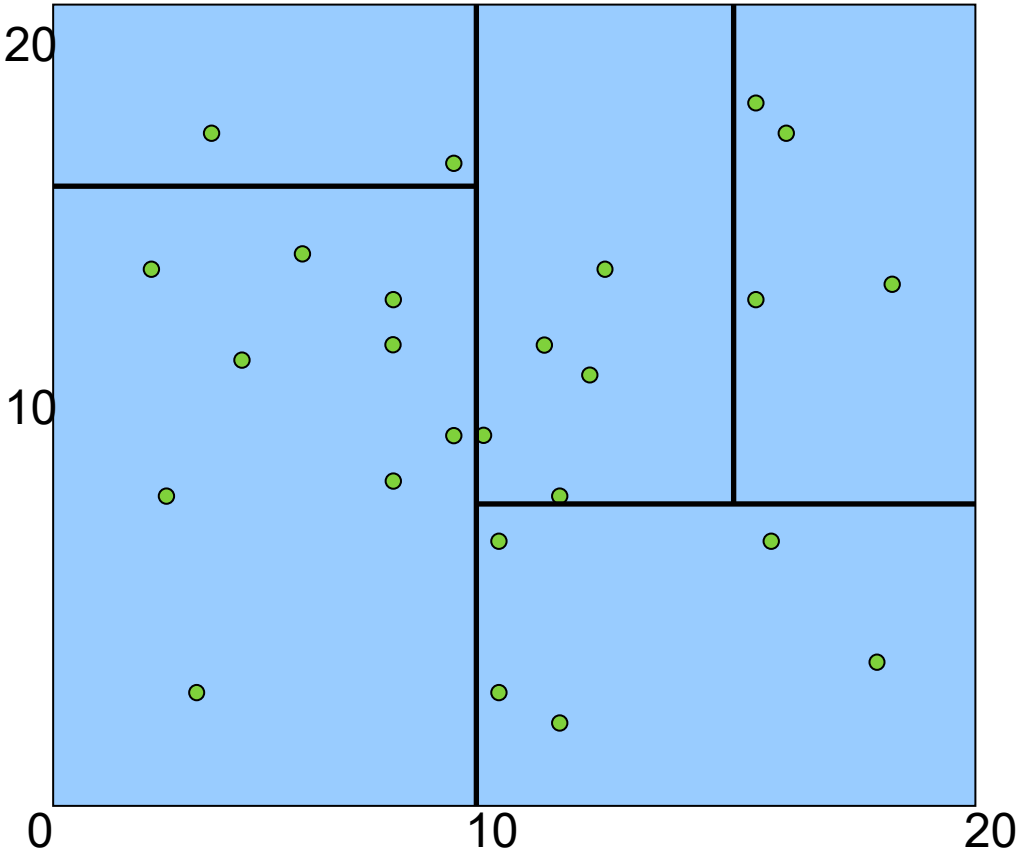
Idea

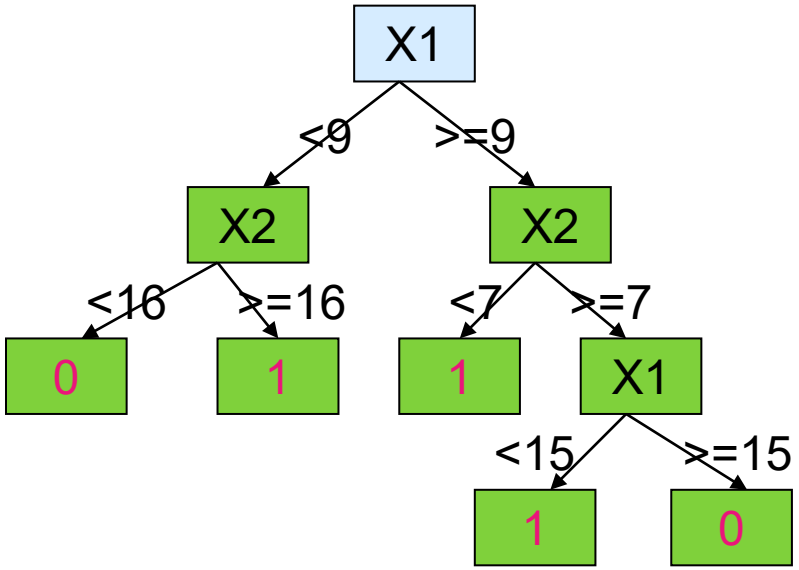
Split the domain of feature set into the set of hypercubes (rectangles, cubes) and define the target value to be constant within each hypercube

$$\widehat{y}(\mathbf{x}_\star) = \sum_{\ell=1}^{L} \widehat{y}_\ell \mathbb{I}\{\mathbf{x}_\star \in R_\ell\}$$

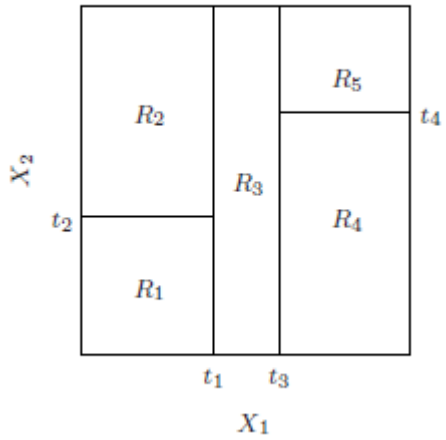
- Regression trees:
  - Target is a continuous variable
- Classification trees
  - Target is a class (qualitative) variable

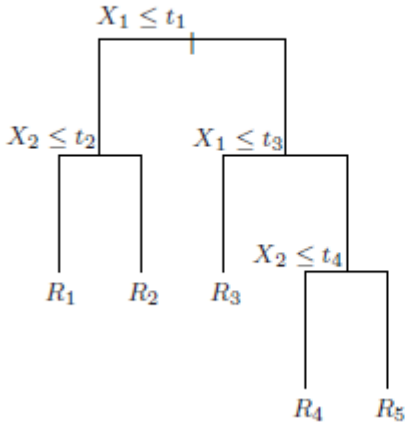
Classification tree toy example

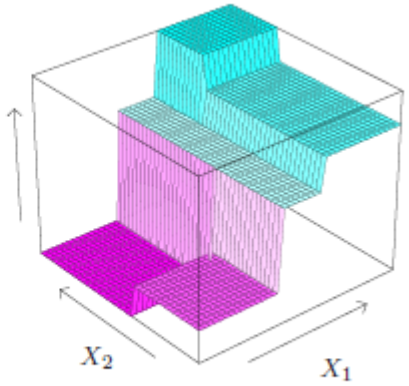




Regression tree toy example



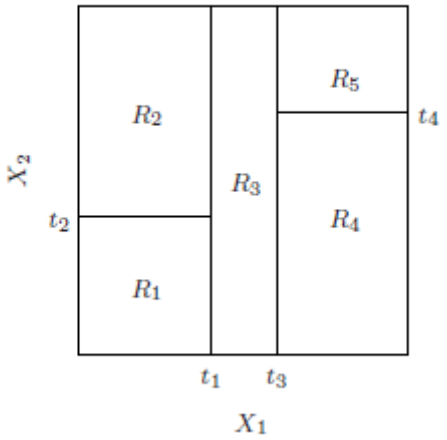




Decision trees

- A tree $T =< r_i, s_{r_i}, R_j, i = 1 \dots S, j = 1 \dots L >$
  - $x_{r_i} \leq s_{r_i}$ splitting rules (conditions), S- their amount
  - R_j-terminal nodes, L- their amount
  - labels μ_j in each terminal node

- Learning by MLE:
  - Step 1: Finding optimal tree
  - Step 2: Finding optimal labels in terminal nodes
  - Problem: NP-hard task!



Decision trees

- **Normal model** leads to regression trees
  - Objective: MSE
- **Multinomial model** leads to classification trees
  - Objective: cross-entropy (deviance)

Classification trees

- Target is categorical

- Classification probability $\pi_{\mathrm{lm}} = p(y = m | x \in R_l)$ is estimated for every class in a node
- How to estimate π_{lm} for class m and node R_l?

Class proportions

$$\hat{\pi}_{lm} = \frac{1}{n_l} \sum_{i: x_i \in R_l} I(y_i = m)$$

- For any node (leave), a label can be assigned

$$\hat{y}_l = \arg\max_m \pi_{lm}$$

Classification trees

- Impurity measure $\mathrm{Q}(R_l)$
  - R_l is a tree node (region)
  - Node can be split unless it is pure

- **Misclassification rate** $Q(R_l) = 1 - \max_m \hat{\pi}_{lm}$
- **Gini index** $Q(R_l) = \sum_{m=1}^{M} \hat{\pi}_{lm}(1 - \hat{\pi}_{lm})$
- **Cross-entropy** $Q(R_l) = -\sum_{m=1}^{M} \hat{\pi}_{lm} \ln \hat{\pi}_{lm}$

- Note: In many sources, deviance is $\mathrm{Q}(R_l)\, n_l$

Learning classification trees: CART

Step 1: Finding optimal tree: grow the tree in order to minimize global objective

1. Let C_0 be a hypercube containing all observations
2. Let queue C={C_0}
3. Pick up some C_i from C and find a variable x_j and value s that split C_j into two hypercubes

$$R_1 = \{x | x_j < s\} \ and \ R_2 = \{x | x_j \geq s\}$$

and minimizes

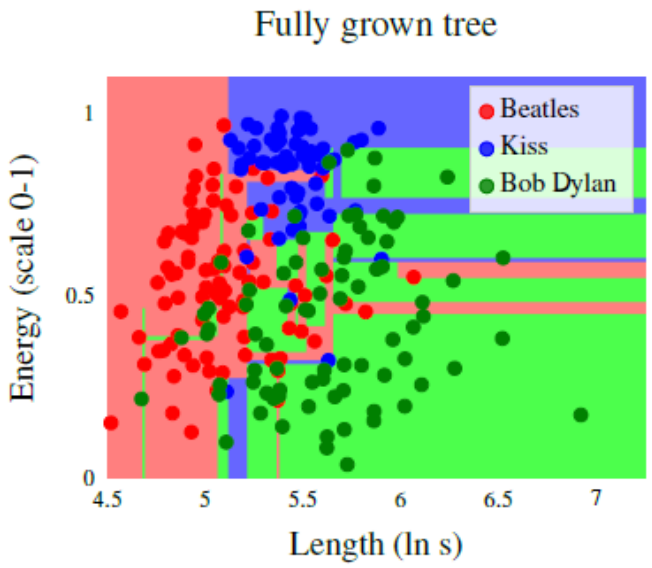
$$\min_{j,s}[n_1 Q(R_1) + n_2 Q(R_2)]$$

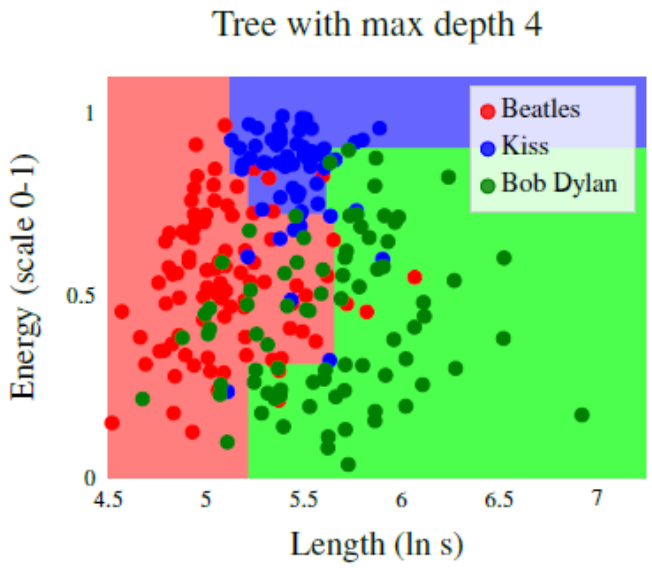
4. Remove C_j from C and add R_1 and R_2
5. Repeat 3-4 as many times as needed (until some stopping criterion or until each cube has only 1 observation)

Greedy algorithm (optimal tree is not found)

CART: comments

- When to stop tree growing?





CART: comments

- The largest tree will interpolate the data → large trees = overfitting the data
- Too small trees=underfitting (important structure may not be captured)
- Optimal tree length?

Optimal trees

- **Postpruning**

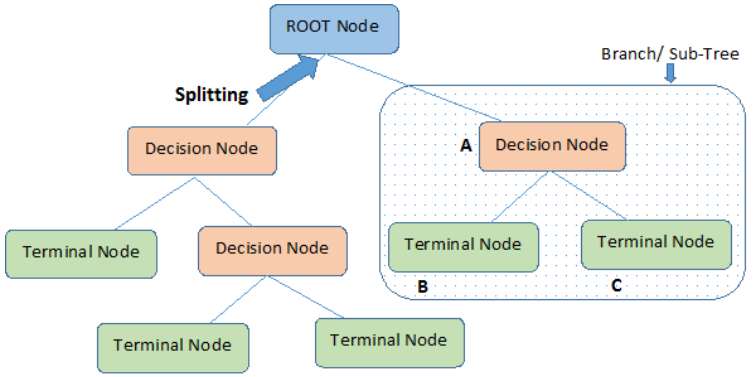
Weakest link pruning:

1. Merge two leaves that have smallest *N(parent)*Q(parent)-N(leave1)Q(leave1)-N(leave2)Q(leave2)*
2. For the current tree T, compute

$$\mathrm{I(T)} = \sum_{R_l \in leaves} N(R_l)Q(R_l) + \alpha|T|$$

$|T|$ =#leaves

3. Repeat 1-2 until the tree with one leave is obtained
4. Select the tree with smallest I(T)



How to find the optimal α? Cross-validation.

Decision trees: comments

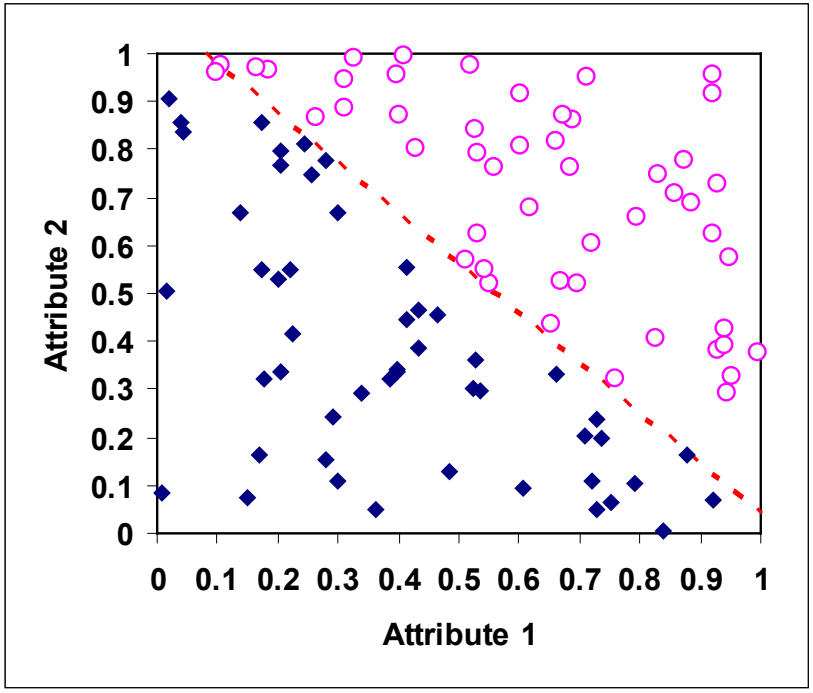
- Similar algorithms work for regression trees
  - replace $\mathrm{n} \cdot Q(R)$ by $SSE(R) = \sum_{i:x_i \in R_l} (y_i - \hat{y}_l)^2$

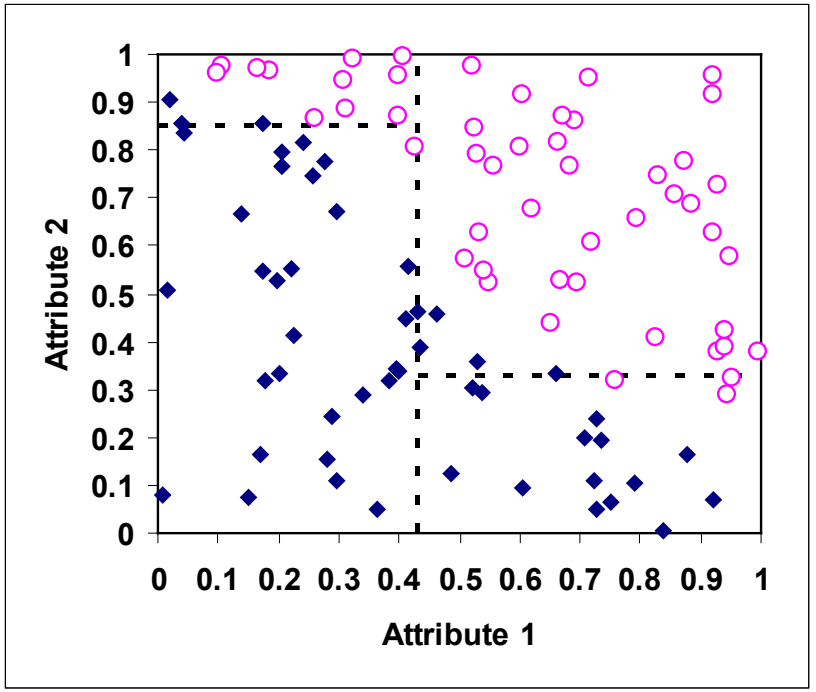
- Belongs to the class of **interpretable ML models**
- Easy to handle all types of features in one model
- **Automatic variable selection**
- Relatively robust to outliers
- Handle large datasets

- Trees have high variance: a small change in response→ totally different tree
- Greedy algorithms → fit may be not so good
- Lack of smoothness

Decision trees: issues

- Large trees may be needed to model an easy system:





Decision trees in R

- **tree** package
  - Alternative: **rpart**

tree(formula, data, weights, control, split = c("deviance", "gini"), …)

print(), summary(), plot(), text()

Example: breast cancer as a function av biological measurements

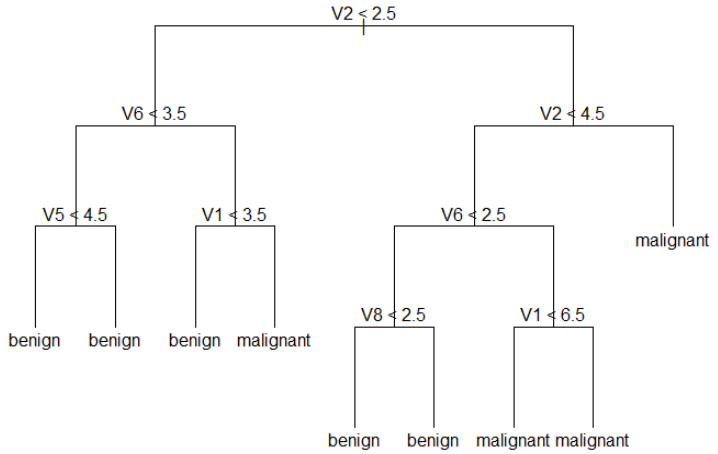
```
library(tree)
n=dim(biopsy)[1]
fit=tree(class~., data=biopsy)
plot(fit)
text(fit, pretty=0)
fit
summary(fit)
```

Decision trees in R

- Adjust the splitting in the tree with *control* parameter (leaf size for ex)

```
> fit
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 683 884.400 benign ( 0.650073 0.349927 )
   2) V2 < 2.5 418 108.900 benign ( 0.971292 0.028708 )
     4) V6 < 3.5 395  25.130 benign ( 0.994937 0.005063 )
       8) V5 < 4.5 389   0.000 benign ( 1.000000 0.000000 ) *
       9) V5 > 4.5 6   7.638 benign ( 0.666667 0.333333 ) *
     5) V6 > 3.5 23  31.490 benign ( 0.565217 0.434783 )
      10) V1 < 3.5 11   0.000 benign ( 1.000000 0.000000 ) *
      11) V1 > 3.5 12  10.810 malignant ( 0.166667 0.833333 ) *
   3) V2 > 2.5 265 217.900 malignant ( 0.143396 0.856604 )
     6) V2 < 4.5 90 120.300 malignant ( 0.388889 0.611111 )
      12) V6 < 2.5 30  27.030 benign ( 0.833333 0.166667 )
        24) V8 < 2.5 19   0.000 benign ( 1.000000 0.000000 ) *
        25) V8 > 2.5 11  15.160 benign ( 0.545455 0.454545 ) *
      13) V6 > 2.5 60  54.070 malignant ( 0.166667 0.833333 )
        26) V1 < 6.5 28  35.160 malignant ( 0.321429 0.678571 ) *
        27) V1 > 6.5 32   8.900 malignant ( 0.031250 0.968750 ) *
     7) V2 > 4.5 175  30.350 malignant ( 0.017143 0.982857 ) *
```




```
> summary(fit)

Classification tree:
tree(formula = class ~ ., data = biopsy)
Variables actually used in tree construction:
[1] "V2" "V6" "V5" "V1" "V8"
Number of terminal nodes:  9
Residual mean deviance:  0.1603 = 108 / 674
Misclassification error rate: 0.03221 = 22 / 683
```

Decision trees in R

- Misclassification results

```
Yfit=predict(fit, newdata=biopsy, type="class")
table(biopsy$class,Yfit)
```

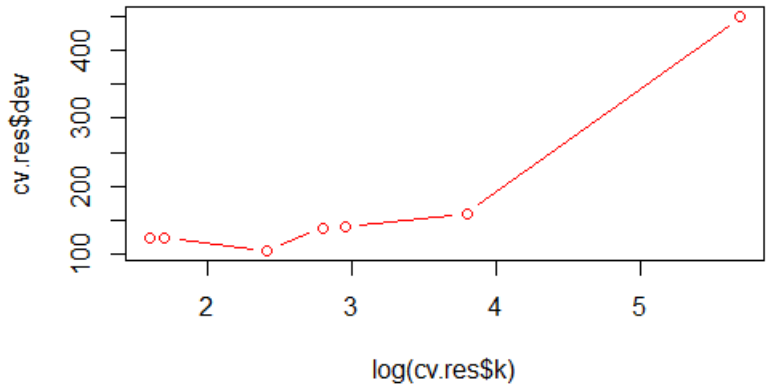
```
> table(biopsy$class,Yfit)
           Yfit
            benign malignant
  benign       440        18
  malignant      7       234
```

Decision trees in R

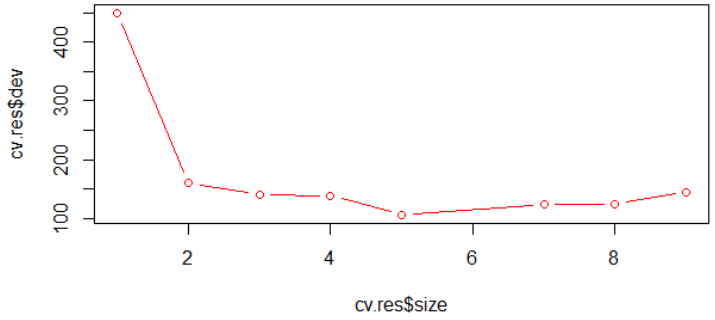
- Selecting optimal tree by penalizing
  - Cv.tree()

```
set.seed(12345)
ind=sample(1:n, floor(0.5*n))
train=biopsy[ind,]
valid=biopsy[-ind,]

fit=tree(class~., data=train)
set.seed(12345)
cv.res=cv.tree(fit)
plot(cv.res$size, cv.res$dev, type="b",
col="red")
plot(log(cv.res$k), cv.res$dev,
type="b", col="red")
```



What is optimal number of leaves?



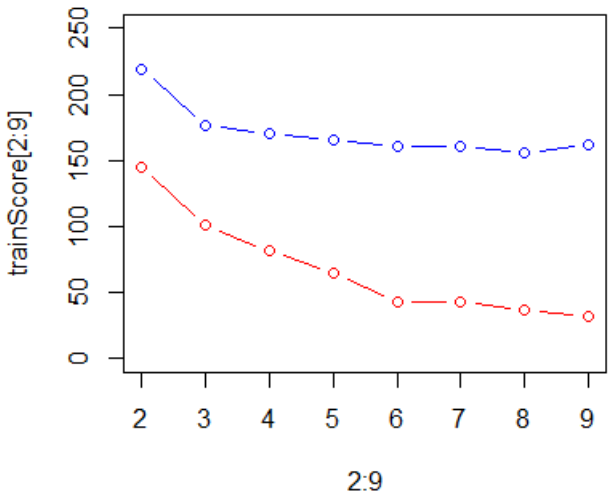
Decision trees in R

- Selecting optimal tree by train/validation

```
fit=tree(class~., data=train)

trainScore=rep(0,9)
testScore=rep(0,9)

for(i in 2:9) {
  prunedTree=prune.tree(fit,best=i)
  pred=predict(prunedTree, newdata=valid,
type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
plot(2:9, trainScore[2:9], type="b", col="red",
ylim=c(0,250))
points(2:9, testScore[2:9], type="b", col="blue")
```



What is optimal number of leaves?

Decision trees in R

- Final tree: 5 leaves

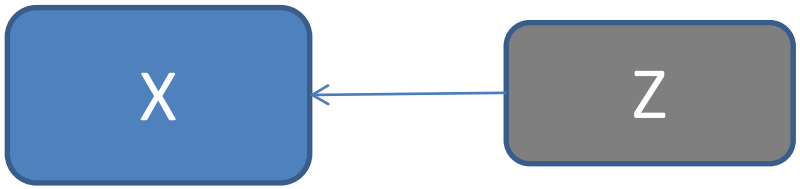
```
finalTree=prune.tree(fit, best=5)
Yfit=predict(finalTree, newdata=valid,
type="class")
table(valid$class,Yfit)
```

```
> table(valid$class,Yfit)
           Yfit
            benign malignant
  benign       222         8
  malignant      6       114
```

Latent variables

- Sometimes data depends on the variables we can not measure (or hard to measure)
  - Answers on the test depend on Intelligence
  - Brain activity in the brain is measured by sensors
  - Stock prices depend on market confidence

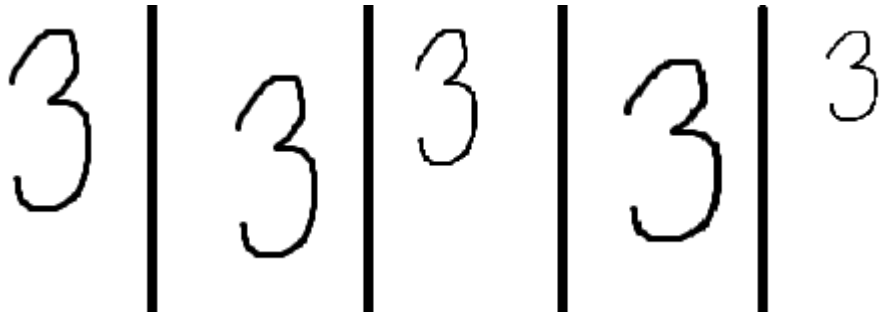
Source: Leadliaison.com



$$X = g(Z)$$

Latent variables

- Latent factor discovered→ data storage may decrease a lot



- Latent factors
  - Center
  - Scaling
- Original vs compressed
  - 100x100x5=50000
  - 100x100+2*5+2*5=10020

Principal Component Analysis (PCA)

- *PCA* is a **feature reduction** / **representation learning** technique, aims to learn latent features from x: $\tilde{z} = f(x)$

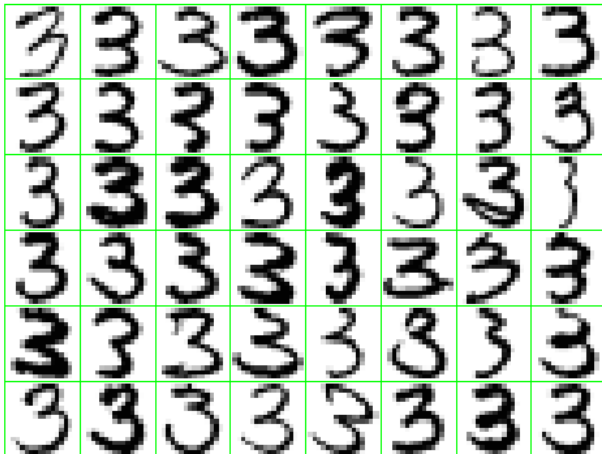
- Used to approximate high dimensional data with a few **informative** features –> much less data to store

Applications

- Industry (sensors)
- Medicine (genes)
- Text analysis (word counts)
- …

Principal Component Analysis (PCA)

- Example 1: Hadwritten digits
  - Can we get a more compact summary?



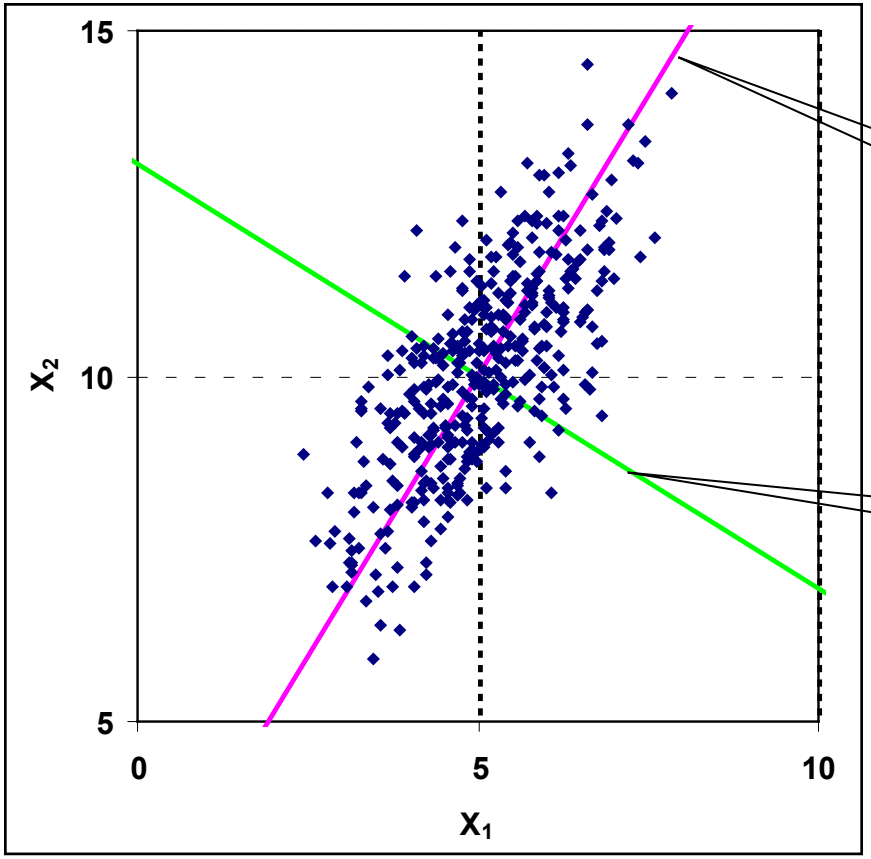
Principal components analysis

Idea: Introduce a new coordinate system (PC1, PC2, …) where

- The first principal component (PC1) is the direction that maximizes the variance of the projected data
- The second principal component (PC2) is the direction that maximizes the variance of the projected data after the variation along PC1 has been removed
- The third principal component (PC3) is the direction that maximizes the variance of the projected data after the variation along PC1 and PC2 has been removed
- ….

In the new coordinate system, coordinates corresponding to the last principal components are very small → can take away these columns

Principal Component Analysis
- two inputs

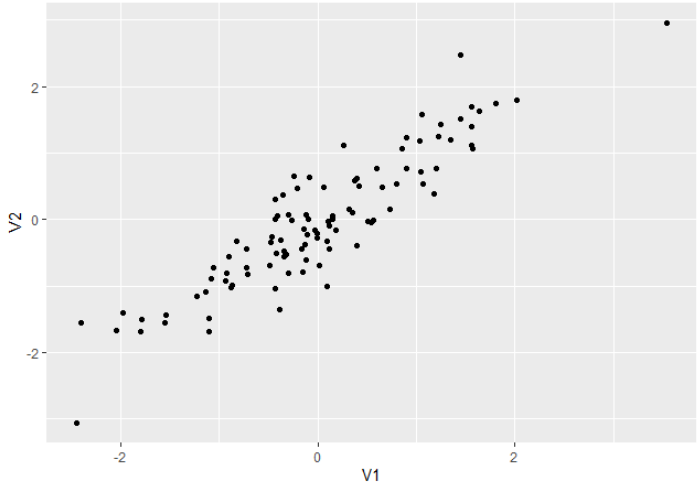


PC1

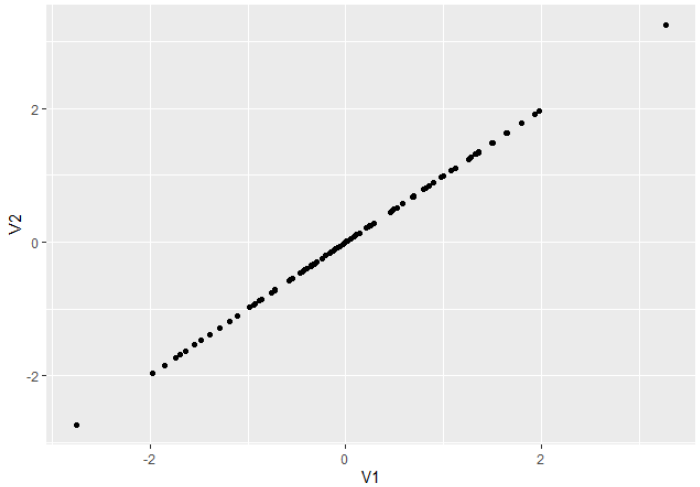
PC2

PCA- after reducing dimensionality

Original data



After compression

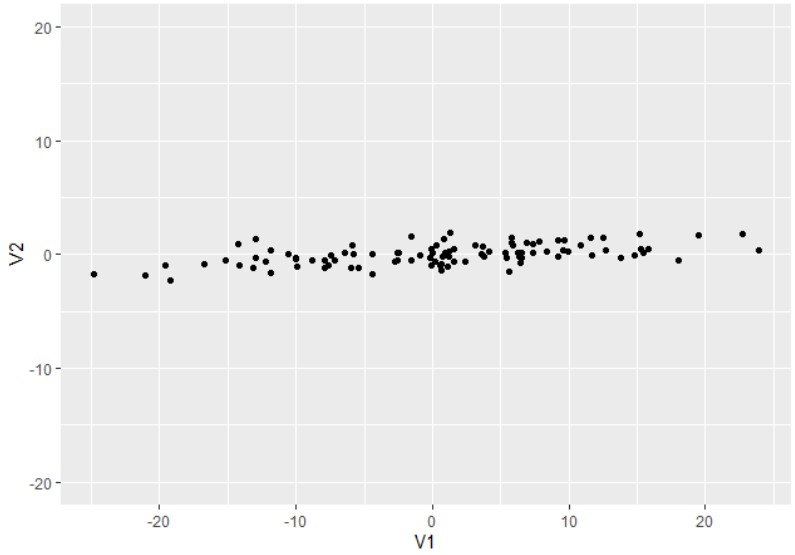


- Data became approximate (but less data to store)
- $PC_1, \ldots PC_p$ are actually eigenvections of **sample covariance** (first largest eigenvalue,…,pth largest egenvalue)

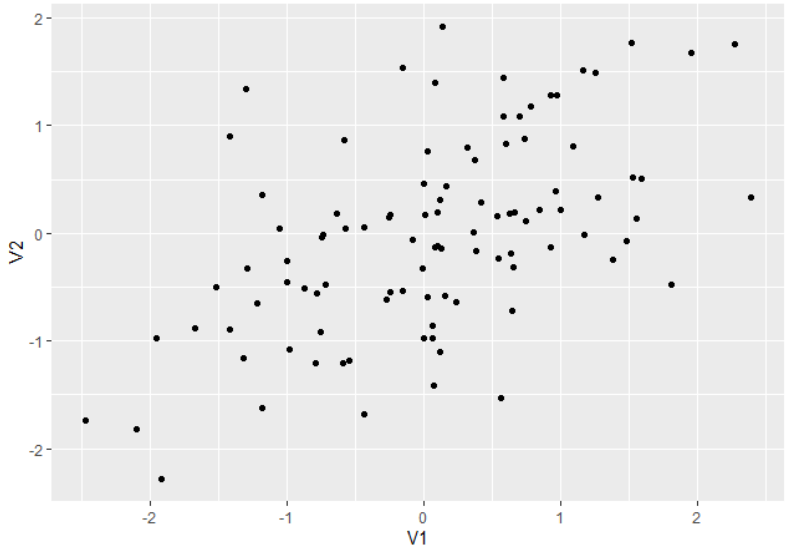
PCA and scaling

- Do we need to scale features?

Without scaling



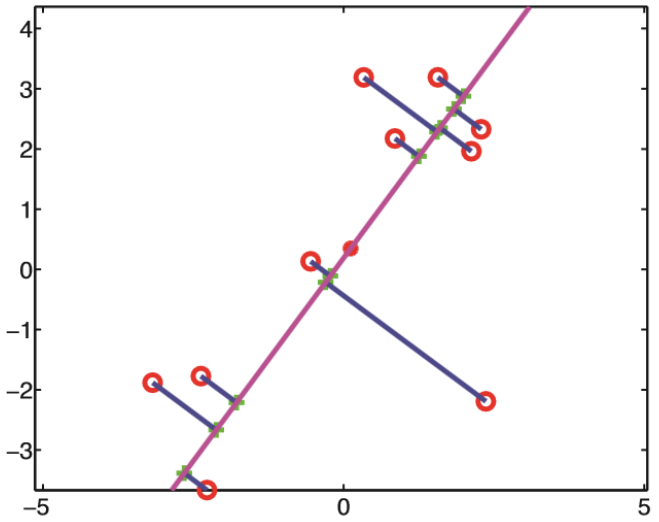
After scaling



PCA: another view

- Aim: minimize the distance between the original and projected data

$$\min_{U_M} \sum_{i=1}^{N} \|x_n - \tilde{x}_n\|^2$$



Source: Murphy

PCA: computations

Data $T = \|\boldsymbol{x^1} \ \ldots \ \boldsymbol{x^p}\|, \quad \boldsymbol{x}^j = (x_{1j}, \ldots, x_{nj})$

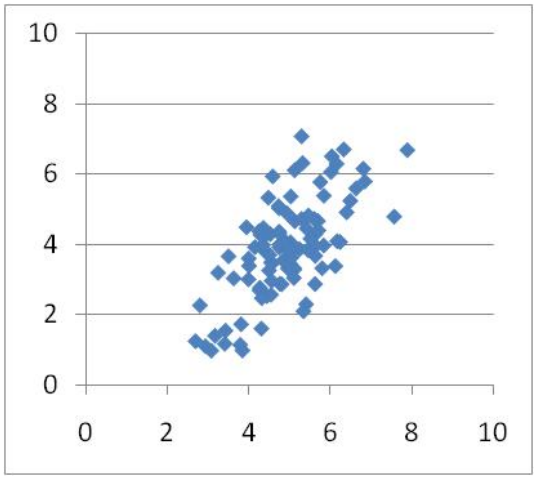
1. Centred data

$$\boldsymbol{X} = \|\boldsymbol{x}^1 \ - \bar{\boldsymbol{x}}^1 \ \boldsymbol{x}^2 \ - \bar{\boldsymbol{x}}^2 \ \ldots \ \boldsymbol{x}^p \ - \bar{\boldsymbol{x}}^p \ \|,$$

2. Covariance matrix

$$\boldsymbol{S} = \frac{\boldsymbol{1}}{\boldsymbol{n}} \boldsymbol{X}^T \boldsymbol{X}$$

3. Search for eigenvectors and eigenvalues of **S**
   - Equivalent: SVD of $\boldsymbol{X}$



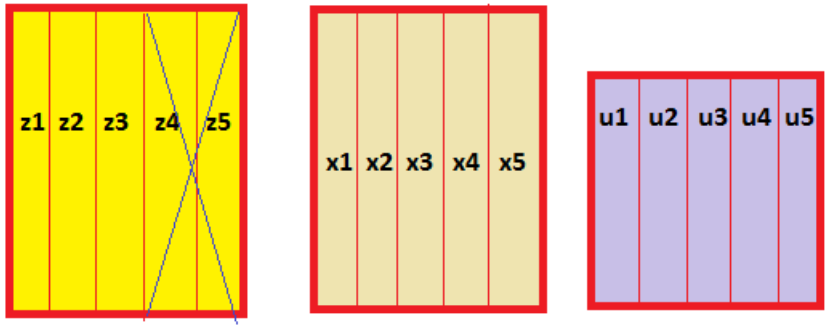
	Column 1	*Column 2*
Column 1	0.951	0.905
Column 2	0.905	1.883

PCA: computations

4. Coordinates of any data point $x=(x_1 \ldots x_p)$ in the new coordinate system:

$$z = (z_1, \ldots z_n),\ z_i = x^T u_i$$

Matrix form: $Z = X\ U$



5. Discard principle components after some *q:*

$$Z = X\ U_q$$

Store: n x q+ p x q
instead n x p

6. New data will have dimensions n x q instead of n x p

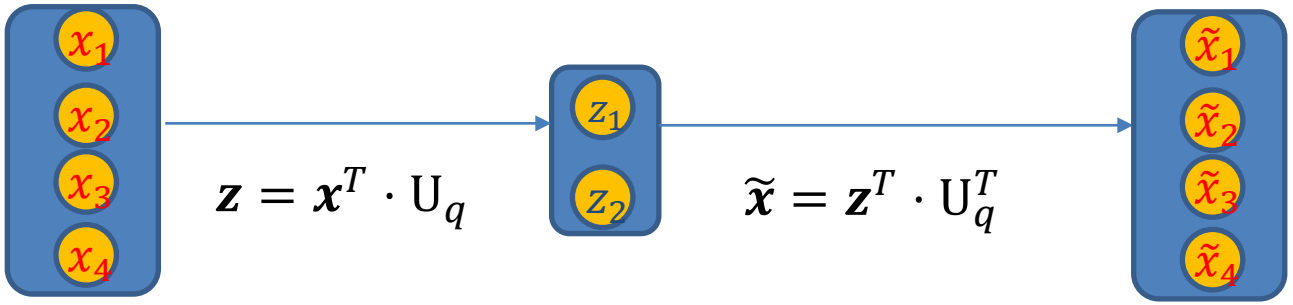
100*50 vs
100*4+50*4

Getting approximate original data:

$$\tilde{X} = ZU_q^T$$

PCA: computations

- PCA makes a **linear** compression of features

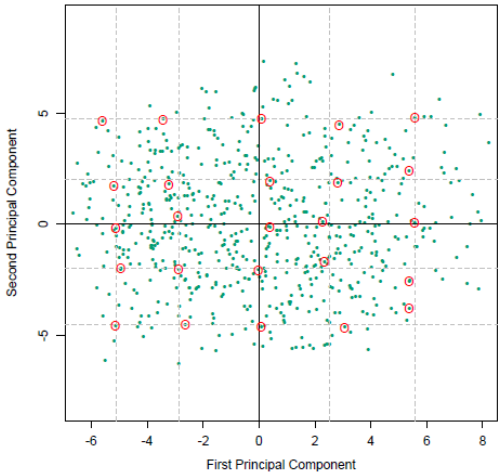


$$\min_{U_q} \sum_{i=1}^{n} \|x_n - \tilde{x}_n\|^2$$

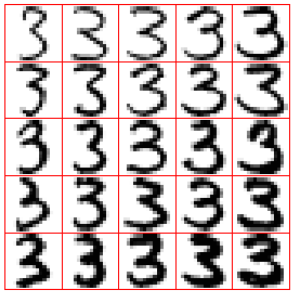
Principal Component Analysis

- Digits: two eigenvectors extracted

- Interptretation of eigenvectiors



Lower tail length

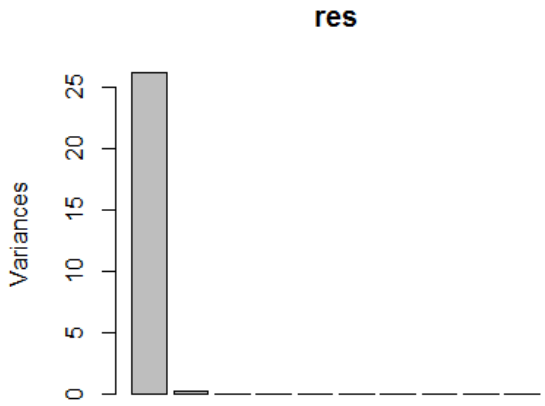


thickness

PCA in R

- Prcomp(), biplot(), screeplot()

```
mydata=read.csv2("tecator.csv")
data1=mydata
data1$Fat=c()
res=prcomp(data1)
lambda=res$sdev^2
#eigenvalues
lambda
#proportion of variation
sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res)
```




```
> lambda
  [1] 2.612713e+01 2.385369e-01 7.844883e-02 3.018501e-
  [7] 2.052212e-04 1.084213e-04 2.077326e-05 1.150359e-
```

```
> sprintf("%2.3f",lambda/sum(lambda)*100)
  [1] "98.679" "0.901"  "0.296"  "0.114"  "0.006
  [9] "0.000"  "0.000"  "0.000"  "0.000"  "0.000
```

Only 1 component captures the 99% of variation!

PCA in R

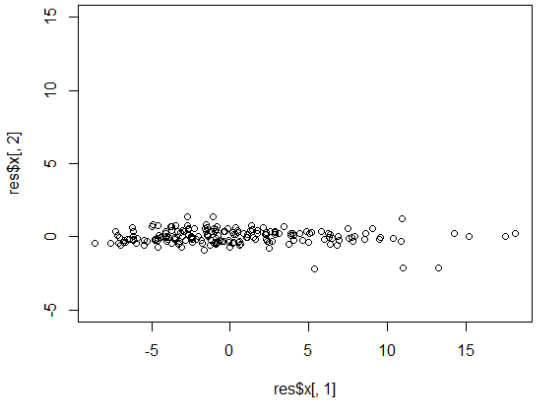
- Principal component **loadings (U)**

```
U=res$rotation
head(U)
```

```
> head(U)
                PC1       PC2        PC3
Channel1 0.07938192 0.1156228 0.08073156 -0.0927
Channel2 0.07987445 0.1170972 0.07887873 -0.0981
Channel3 0.08036498 0.1185571 0.07702127 -0.1031
Channel4 0.08085611 0.1200006 0.07515015 -0.1077
Channel5 0.08135022 0.1214075 0.07323819 -0.1119
Channel6 0.08184896 0.1227491 0.07125048 -0.1156
```

- Data in (PC1, PC2) – **scores (Z)**

```
plot(res$x[,1], res$x[,2], ylim=c(-5,15))
```

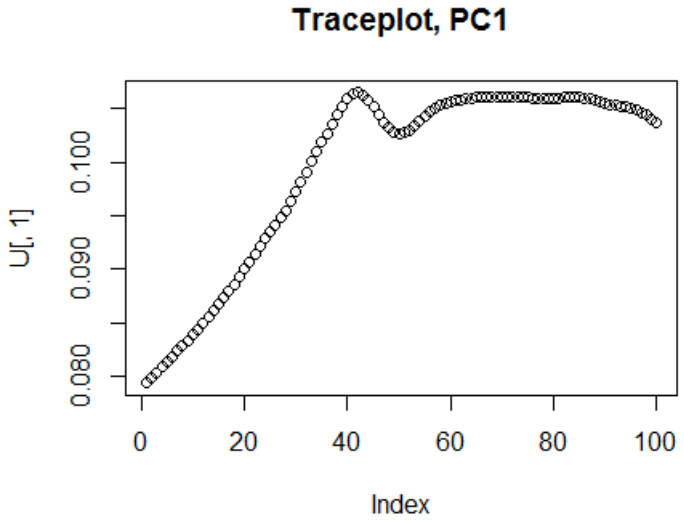


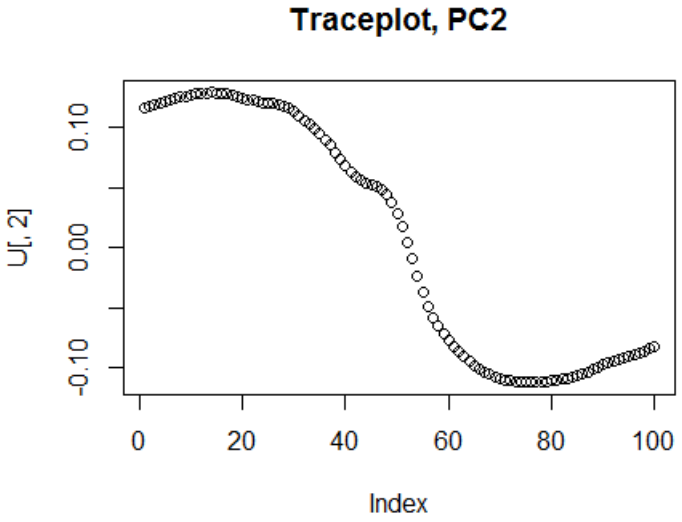
Do we need second dimension?

PCA in R

- **Trace plots**

```
U= res$rotation
plot(U[,1], main="Traceplot, PC1")
plot(U[,2],main="Traceplot, PC2")
```

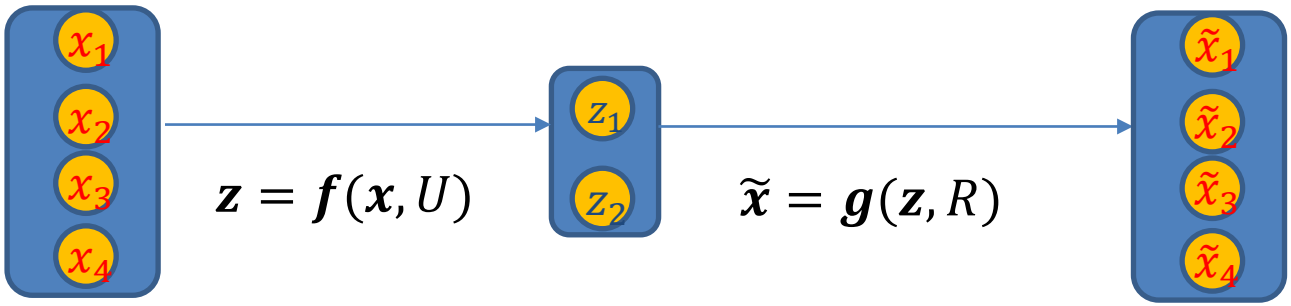




Which components contribute to PC1-2?

Autoencoders (nonlinear PCA)

- Why linear transformations? Take nonlinear instead!
- $f()$ and $g()$ are typically Neural Networks



$$\min_{U,R} \sum_{i=1}^{n} \|x_n - \tilde{x}_n\|^2$$

…or some other loss function

Other linear representation learning methods

- Probabilistic PCA
  - Similar to PCA but has more opportunities
    - Can be used to handle missing values directly
    - Can be easily embedded in Bayesian ML models
    - Can be used to generate new data
- Independent component analysis (ICA)
  - Sometimes shows better emprical results comp. PCA