DATA7001 - Introduction to Data Science
Module 4: Making the data confess

Xin GUO

Email: **xin.guo@uq.edu.au**
Office: **Priestley Building (67), Room 447**
Telephone: **+61 7 3346 9728**
Office Hours: **Tuesday 10–11 AM**

Linear Models

The model:

$$Y = \beta_0^* + \beta_1^* x_1 + \cdots + \beta_r^* x_r + \varepsilon.$$

For example:

- Y: called the response variable, or the dependent variable. For example, it could be the reaction of a patient to some specific drug.
- $x_1, x_2, \ldots, x_r$: called the predictor variables, or the independent variables, or the explanatory variables. For example, they could be the gene expression level, blood pressure, BMI, etc., respectively, of the patient. We assume that the predictors are deterministic (not random).
- $\beta_0^*, \beta_1^*, \ldots, \beta_r^*$: deterministic (not random) constants, assumed unknown. Coefficients of the model. While β_0^* is often called the intercept, $\beta_1^*, \ldots, \beta_r^*$ are sometimes called the effects.
- ε: called the error term, or the noise. We assume that $\varepsilon \sim N(0, \sigma^2)$ for some unknown $\sigma^2 > 0$.

The data:

$$D = \left\{(x_1^i, x_2^i, \ldots, x_r^i, y^i)\right\}_{i=1}^n \subset \mathbb{R}^{r+1}$$

We have

$$\begin{aligned} y^1 &= \beta_0^* + \beta_1^* x_1^1 + \cdots + \beta_r^* x_r^1 + \varepsilon^1, \\ y^2 &= \beta_0^* + \beta_1^* x_1^2 + \cdots + \beta_r^* x_r^2 + \varepsilon^2, \\ &\vdots \\ y^n &= \beta_0^* + \beta_1^* x_1^n + \cdots + \beta_r^* x_r^n + \varepsilon^n. \end{aligned}$$

We assume that the error terms are independent. That is,

$$\varepsilon^1, \varepsilon^2, \ldots, \varepsilon^n \overset{\text{i.i.d.}}{\sim} N(0, \sigma^2),$$

where "i.i.d." stands for "independent and identically distributed".

We rewrite the equations.

$$\begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_r^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_r^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^n & x_2^n & \cdots & x_r^n \end{bmatrix} \begin{bmatrix} \beta_0^* \\ \beta_1^* \\ \vdots \\ \beta_r^* \end{bmatrix} + \begin{bmatrix} \varepsilon^1 \\ \varepsilon^2 \\ \vdots \\ \varepsilon^n \end{bmatrix}$$

$$\boldsymbol{y} = \mathbb{X} \quad \boldsymbol{\beta}^* + \boldsymbol{\varepsilon}$$

To estimate the unknown coefficient vector $\boldsymbol{\beta}^*$, we try different vectors $\boldsymbol{\beta} \in \mathbb{R}^{r+1}$ to minimize the least-squares loss function $S(\boldsymbol{\beta})$,

$$\begin{aligned} S(\boldsymbol{\beta}) &= \|\boldsymbol{y} - \mathbb{X}\boldsymbol{\beta}\|^2 \\ &= \sum_{i=1}^{n} \left(y^i - \beta_0 - \beta_1 x_1^i - \cdots - \beta_r x_r^i\right)^2. \end{aligned}$$

We denote $\hat{\boldsymbol{\beta}}_{\mathsf{ls}}$ the minimizer of the function $S(\boldsymbol{\beta})$.

Multivariate Normal Distributions

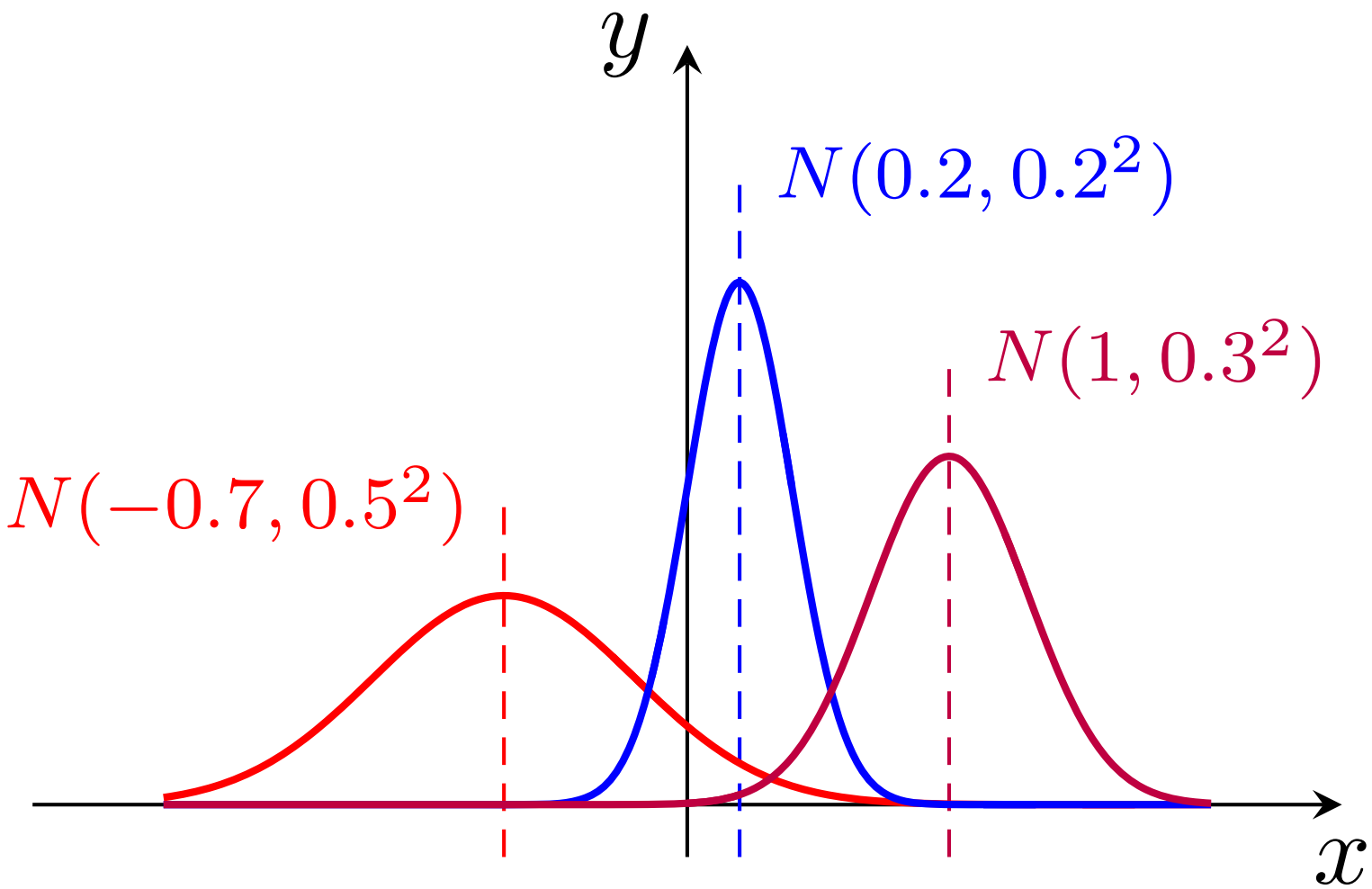
Recall the one-dimensional normal distributions $N(\mu, \sigma^2)$, of which the density function is

$$f_{\mu,\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}, \quad x \in \mathbb{R}.$$

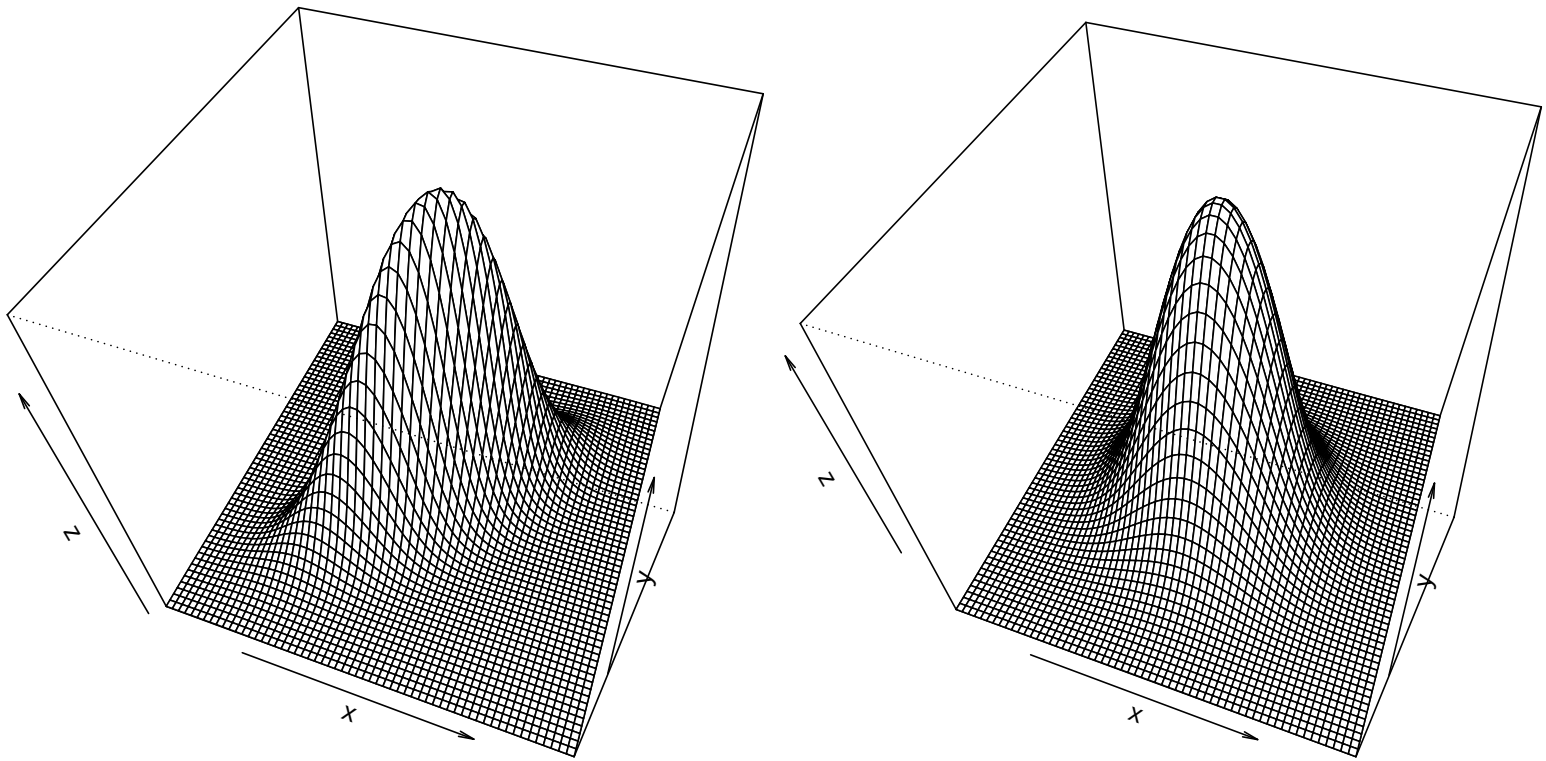
Let $d \geq 1$ be an integer. Let $\boldsymbol{\mu}$ be a d-dimensional real vector. That is, $\boldsymbol{\mu} \in \mathbb{R}^d$. Let Σ be a $d \times d$ real matrix, i.e., $\Sigma \in \mathbb{R}^{d\times d}$. Assume that Σ is symmetric and positive definite. The d-dimensional normal distribution $N(\boldsymbol{\mu}, \Sigma)$ is defined through the density function

$$f_{\boldsymbol{\mu},\Sigma}(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}\sqrt{\det \Sigma}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})'\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right\}.$$

Example: density functions of some 1-dimensional normal distributions



Example: density functions of some 2-dimensional normal distributions



Theorem

Suppose $\boldsymbol{X} = (X_1, X_2, \ldots, X_d)' \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. *Write*

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_d \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \Sigma_{1,1} & \Sigma_{1,2} & \cdots & \Sigma_{1,d} \\ \Sigma_{2,1} & \Sigma_{2,2} & \cdots & \Sigma_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{d,1} & \Sigma_{d,2} & \cdots & \Sigma_{d,d} \end{bmatrix}.$$

Then for any $1 \leq i \leq d$,

$$X_i \sim N(\mu = \mu_i, \sigma^2 = \Sigma_{i,i}).$$

Moreover, for $i \neq j$, X_i *and* X_j *are independent if and only if* $\Sigma_{i,j} = 0$.

We skip the proof.

A Toy Example: Inference for the Population Mean

Recall the process of estimating the mean of a 1-dim normal distribution.

- Data: $\xi^1, \xi^2, \ldots, \xi^n \overset{\text{i.i.d.}}{\sim} N(\mu, \sigma^2)$, where μ and σ^2 are unknown.
- Estimate of the population mean:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \xi^i = \arg \min_{-\infty<t<\infty} \sum_{i=1}^{n} (t - \xi^i)^2.$$

Here "argmin$f(t)$" means the minimizer of the function $f(t)$. For example, argmin$(t-5)^2 = 5$.

- We have (skipping the mathematics) $\hat{\mu} \sim N(\mu, \sigma^2/n)$. So, $\sigma^2_{\hat{\mu}} := \mathrm{Var}(\hat{\mu}) = \sigma^2/n$.
- The variance $\sigma^2_{\hat{\mu}}$ is unknown but is useful in constructing the confidence interval of $\hat{\mu}$ and conducting hypothesis test (e.g. $H_0 : \mu = 0$ v.s. $H_1 : \mu \neq 0$).
- We compare $\hat{\mu}$ with $\hat{\boldsymbol{\beta}}_{\mathsf{ls}}$ (recall that $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_r)'$),

$$\hat{\mu} = \arg\min_{t \in \mathbb{R}} \sum_{i=1}^{n} (t - \xi^i)^2,$$

$$\hat{\boldsymbol{\beta}}_{\mathsf{ls}} = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^{r+1}} \sum_{i=1}^{n} \left(\beta_0 + \beta_1 x_1^i + \cdots + \beta_r x_r^i - y^i\right)^2.$$

- Usually, one uses the estimate $\hat{\sigma}^2 = s^2$, called the sample variance,

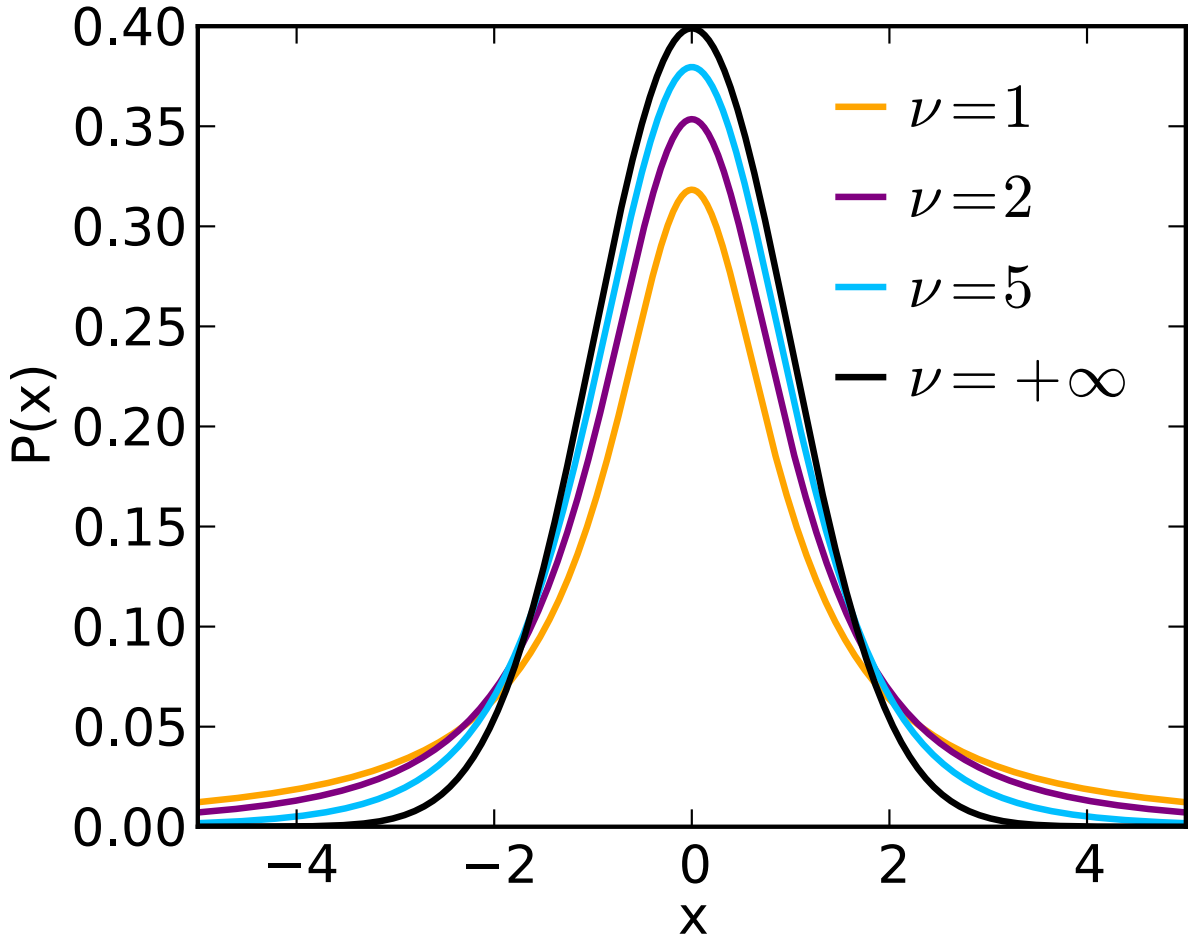
$$s^2 := \frac{1}{n-1} \sum_{i=1}^{n} \left(\xi^i - \hat{\mu}\right)^2 .$$

  We skip the mathematics but point out that $\mathbb{E}[s^2] = \sigma^2$. Therefore it is reasonable to use $\hat{\sigma}^2_{\hat{\mu}} = s^2/n$ as an estimate of the unknown variance $\sigma^2_{\hat{\mu}} = \sigma^2/n$.
- One defines the $(1-\alpha) \times 100\%$ confidence interval of μ,

$$\left[\hat{\mu} - t_{\alpha/2,n-1}\hat{\sigma}_{\hat{\mu}}, \hat{\mu} + t_{\alpha/2,n-1}\hat{\sigma}_{\hat{\mu}}\right],$$

  where $t_{\alpha/2,n-1}$ is the top $\alpha/2$ quantile of the t distribution with $n-1$ degrees of freedom (we skip the details).
- One defines the rejection region for the null hypothesis $H_0 : \mu = 0$ (against the alternative hypothesis $H_1 : \mu \neq 0$),

$$R_\alpha := \{|\hat{\mu}| > t_{\alpha/2,n-1}\hat{\sigma}_{\hat{\mu}}\}.$$



Density functions of t-distributions with different degrees of freedom. Note that $t_\infty = N(0, 1)$. As $\nu \to \infty$, t_ν becomes more and more concentrated.

Picture: wikipedia

Properties of the least squares estimator $\hat{\boldsymbol{\beta}}_{\mathsf{ls}}$

Recall the linear model $\boldsymbol{y} = \mathbb{X}\boldsymbol{\beta}^* + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon} = (\varepsilon^1, \ldots, \varepsilon^n)' \sim N(\mathbf{0}, \sigma^2 I)$ is the noise vector. The unknown coefficient vector $\boldsymbol{\beta}^*$ is estimated by the least squares method $\hat{\boldsymbol{\beta}}_{\mathsf{ls}} = \operatorname{argmin}_{\boldsymbol{\beta}} \|\boldsymbol{y} - \mathbb{X}\boldsymbol{\beta}\|^2$. We have (proof skipped)

$$\hat{\boldsymbol{\beta}}_{\mathsf{ls}} \sim N(\boldsymbol{\beta}^*, \sigma^2(\mathbb{X}'\mathbb{X})^{-1}).$$

Therefore, for $0 \le i \le r$, the i-th coordinate $(\hat{\boldsymbol{\beta}}_{\mathsf{ls}})_i$ has the normal distribution

$$(\hat{\boldsymbol{\beta}}_{\mathsf{ls}})_i \sim N(\beta_i^*, \sigma^2[(\mathbb{X}'\mathbb{X})^{-1}]_{i,i}).$$

With the help of the estimator $\hat{\sigma}^2 = \|\boldsymbol{y} - \mathbb{X}\hat{\boldsymbol{\beta}}_{\mathsf{ls}}\|^2/(n-r-1)$, the $(1-\alpha)\times 100\%$ confidence interval of β_i^* is given by

$$\left[(\hat{\boldsymbol{\beta}}_{\mathsf{ls}})_i - t_{\alpha/2,n-r-1}\hat{\sigma}\sqrt{[(\mathbb{X}'\mathbb{X})^{-1}]_{i,i}}, (\hat{\boldsymbol{\beta}}_{\mathsf{ls}})_i + t_{\alpha/2,n-r-1}\hat{\sigma}\sqrt{[(\mathbb{X}'\mathbb{X})^{-1}]_{i,i}}\right].$$

Test of Significance of Regression

- The test of significance of regression

$$H_0 : \beta_1^* = \cdots = \beta_r^* = 0, \qquad \text{v.s.} \qquad H_1 : \text{otherwise}$$

- Insight: to test if there is a linear relationship between Y and $x_1, \ldots, x_r$.
- Test statistic

$$F = \frac{(SSY - SSE)/r}{SSE/(n-r-1)}$$

- Write $\hat{\boldsymbol{y}} = (\hat{y}^1, \ldots, \hat{y}^n)' := \mathbb{X}\hat{\beta}_{\text{ls}}$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^n y^i$. We have

$$SSE = \sum_{i=1}^n (y^i - \hat{y}^i)^2 = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2, \quad SSY = \sum_{i=1}^n (y^i - \bar{y})^2.$$

- Under H_0, $F \sim F_{r,n-r-1}$, the F distribution with degrees of freedom r and $n-r-1$. One rejects H_0 if F is too large (depending on the test significance), in which case $\beta_1^*, \ldots, \beta_r^*$ are significant.

Test on Individual Regression Coefficient

- The test of significance of β_j^*, $1 \leq j \leq r$,

$$H_0 : \beta_j^* = 0, \qquad \text{v.s.} \qquad H_1 : \beta_j^* \neq 0.$$

- Test statistic

$$T_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{[(\mathbb{X}'\mathbb{X})^{-1}]_{i,i}}}$$

- Under the null hypothesis H_0, $T_j \sim t_{n-r-1}$. The rejection region is determined accordingly.
- Recall that $\hat{\sigma}^2 = \|\boldsymbol{y} - \mathbb{X}\hat{\boldsymbol{\beta}}_{\text{ls}}\|^2/(n-r-1)$. The product $\hat{\sigma}\sqrt{[(\mathbb{X}'\mathbb{X})^{-1}]_{i,i}}$ is called the **estimated standard error**, or simply **standard error** of the coefficient $\hat{\beta}_j$.

Linear Regression with R

Example. In this example, we generate artificial data use the model

$$Y = \beta_0^* + \beta_1^* x_1 + \beta_2^* x_2 + \varepsilon,$$

with $(\beta_0^*, \beta_1^*, \beta_2^*) = (1, -3, 2)$, and $\varepsilon \sim N(0, \sigma^2 = 0.2^2)$. For convenience, we generate the predictors (x_1, x_2) through standard normal distribution $N(0, 1)$. We shall see that as the sample size n tends to infinity, the error vector $\hat{\boldsymbol{\beta}}_{\text{ls}} - \boldsymbol{\beta}^*$ becomes smaller and smaller.

```
betaOracle <- c(1, -3, 2)
genData <- function(n){
  x <- cbind(1, matrix(rnorm(2 * n), ncol = 2))
  epsilon <- rnorm(n = n, sd = 0.2)
  y <- x %*% betaOracle + epsilon
  dta <- as.data.frame(cbind(y, x))
  colnames(dta) <- c("y", "1", "x1", "x2")
  return(dta)
}
print(betaOracle)
set.seed(123)
for(n in c(1e2, 1e3, 1e4, 1e5, 1e6)){
  a <- lm(y ~ x1 + x2, data = genData(n))
  print(paste("data size:", n))
  print(rbind(a$coefficients, a$coefficients - betaOracle))
}
```

The program output is listed below.

```
[1]  1 -3  2
[1] "data size: 100"
     (Intercept)          x1          x2
[1,]  1.02701309 -3.02663431 2.004762258
[2,]  0.02701309 -0.02663431 0.004762258
[1] "data size: 1000"
       (Intercept)           x1          x2
[1,]  0.9998569029 -3.008709992 2.009790489
[2,] -0.0001430971 -0.008709992 0.009790489
[1] "data size: 10000"
       (Intercept)           x1           x2
[1,]  0.9997507513 -2.997007679  1.996400911
[2,] -0.0002492487  0.002992321 -0.003599089
[1] "data size: 1e+05"
       (Intercept)           x1           x2
[1,]  0.9990855487 -3.001188885 2.0003540759
[2,] -0.0009144513 -0.001188885 0.0003540759
[1] "data size: 1e+06"
      (Intercept)           x1            x2
[1,] 1.000079e+00 -3.0002596327  1.9999565308
[2,] 7.899183e-05 -0.0002596327 -0.0000434692
```

We use the above data generator to study the program output.

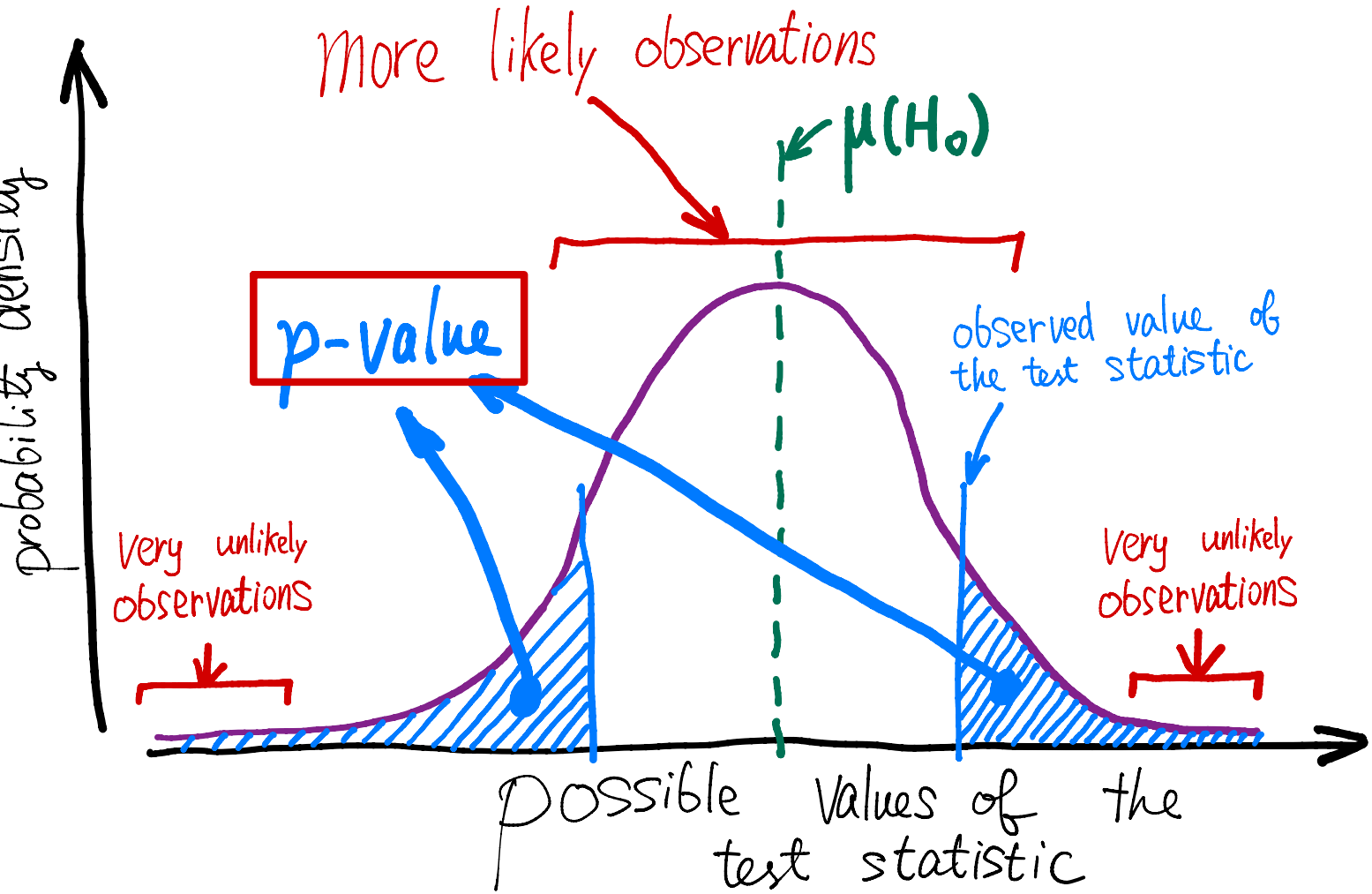
```
betaOracle <- c(1, -3, 2)
genData <- function(n){
  x <- cbind(1, matrix(rnorm(2 * n), ncol = 2))
  epsilon <- rnorm(n = n, sd = 0.2)
  y <- x %*% betaOracle + epsilon
  dta <- as.data.frame(cbind(y, x))
  colnames(dta) <- c("y", "1", "x1", "x2")
  return(dta)
}
print(betaOracle)
set.seed(123)
print(summary(lm(y ~ x1 + x2, data = genData(50))))
```

The program output is listed below.

```
[1]  1 -3  2

Call:
lm(formula = y ~ x1 + x2, data = genData(50))

Residuals:
     Min       1Q   Median       3Q      Max
-0.37627 -0.14811 -0.01275  0.10503  0.45409

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.95401    0.02861   33.34   <2e-16 ***
x1          -2.99497    0.03080  -97.23   <2e-16 ***
x2           1.96608    0.03150   62.42   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1995 on 47 degrees of freedom
Multiple R-squared:  0.9966,^^IAdjusted R-squared:  0.9965
F-statistic:  6902 on 2 and 47 DF,  p-value: < 2.2e-16
```

Here, “Residuals” refer to the vector

$$\hat{\boldsymbol{\varepsilon}} := \boldsymbol{y} - \mathbb{X}\hat{\boldsymbol{\beta}}_{\mathsf{ls}}.$$


More likely observations
$\mu(H_0)$
p-value
observed value of the test statistic
probability density
Very unlikely observations
Very unlikely observations
possible values of the test statistic

In the test below, we purposely add one irrelevant variable to better understand the program output

```
betaOracle <- c(1, -3, 2)
genData <- function(n){
  x <- cbind(1, matrix(rnorm(2 * n), ncol = 2))
  epsilon <- rnorm(n = n, sd = 0.2)
  y <- x %*% betaOracle + epsilon
  dta <- as.data.frame(cbind(y, x))
  colnames(dta) <- c("y", "1", "x1", "x2")
  return(dta)
}
print(betaOracle)
set.seed(123)
a <- cbind(genData(50), rnorm(50))
colnames(a) <- c("y", "1", "x1", "x2", "x3")
print(summary(lm(y ~ x1 + x2 + x3, data = a)))
```

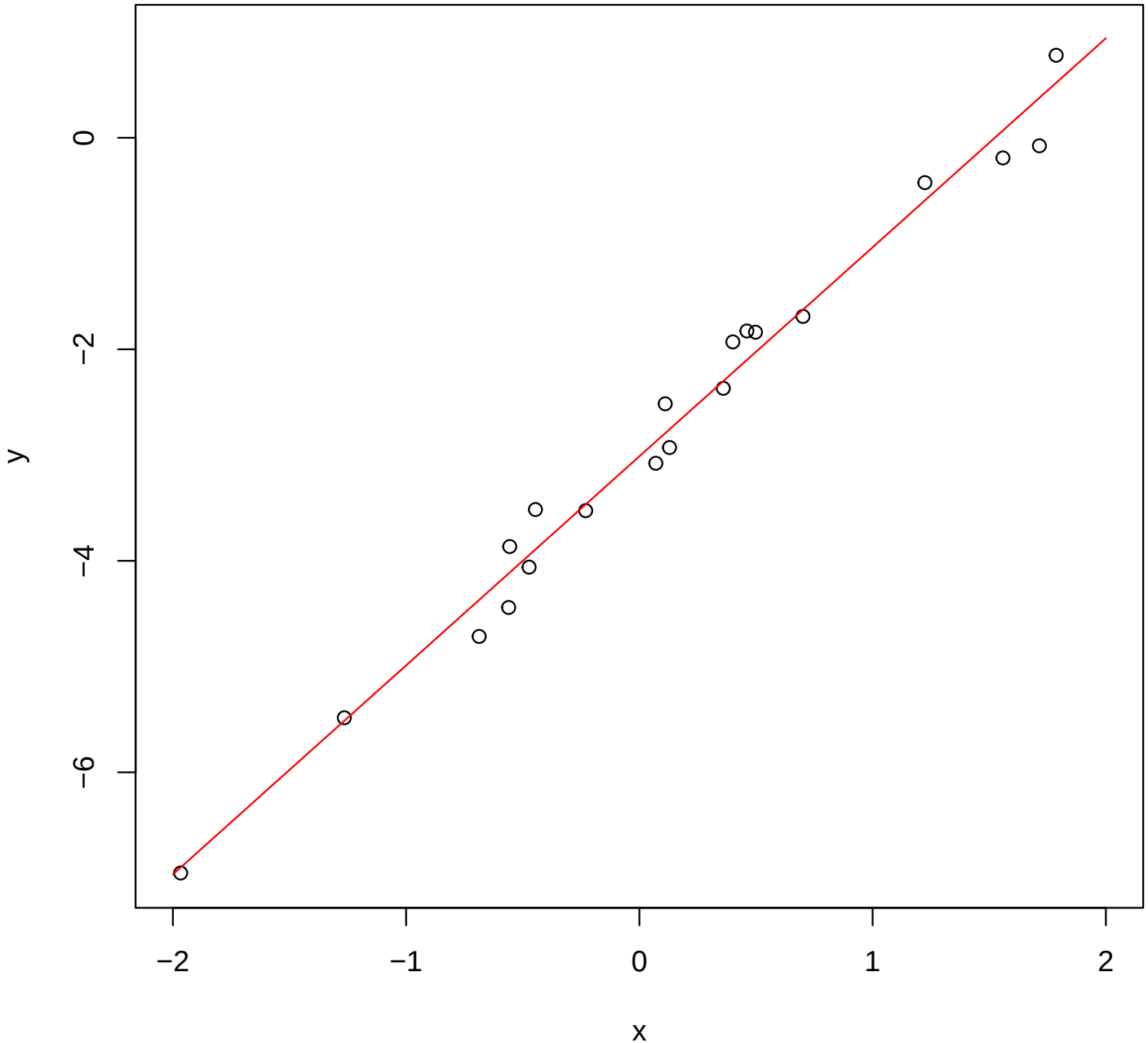
The program output is listed below. We see that the p-value for the variable x_3 is too large (for example, larger than 0.05). In practice, one usually removes x_3 and starts over.

```
[1]  1 -3  2

Call:
lm(formula = y ~ x1 + x2 + x3, data = a)

Residuals:
     Min       1Q   Median       3Q      Max
-0.37779 -0.14871 -0.01149  0.10858  0.45011

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.954448   0.028988  32.925   <2e-16 ***
x1          -2.995801   0.031386 -95.451   <2e-16 ***
x2           1.965011   0.032248  60.934   <2e-16 ***
x3          -0.006452   0.031581  -0.204    0.839
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2016 on 46 degrees of freedom
Multiple R-squared:  0.9966,^^IAdjusted R-squared:  0.9964
F-statistic:  4508 on 3 and 46 DF,  p-value: < 2.2e-16
```

The following code generates an artificial dataset with model $y = -3 + 2x + \epsilon$, carries out least squares algorithm on the data, and generates predictions.

```
set.seed(123)
x.trn <- rnorm(20)
y.trn <- -3 + 2 * x.trn + rnorm(length(x.trn), sd = 0.3)
data.trn <- as.data.frame(cbind(y.trn, x.trn))
colnames(data.trn) <- c("y", "x")

x.tst <- seq(-2, 2, len = 50)
data.tst <- as.data.frame(x.tst)
colnames(data.tst) <- "x"

model <- lm(y ~ x, data = data.trn)
y.predicted <- predict(model, newdata = data.tst)

plot(range(c(x.trn, x.tst)), range(c(y.trn, y.predicted)),
  type = "n", xlab = "x", ylab = "y")
points(x.trn, y.trn, col = "black")
lines(x.tst, y.predicted, col = "red")
```

The obtained figure is pasted below.



Recall the least squares estimation

$$\hat{\boldsymbol{\beta}}_{\text{ls}} = \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^{r+1}} \|\boldsymbol{y} - \mathbb{X}\boldsymbol{\beta}\|^2 . \tag{1}$$

The following theorem provides an analytical solution to (1).

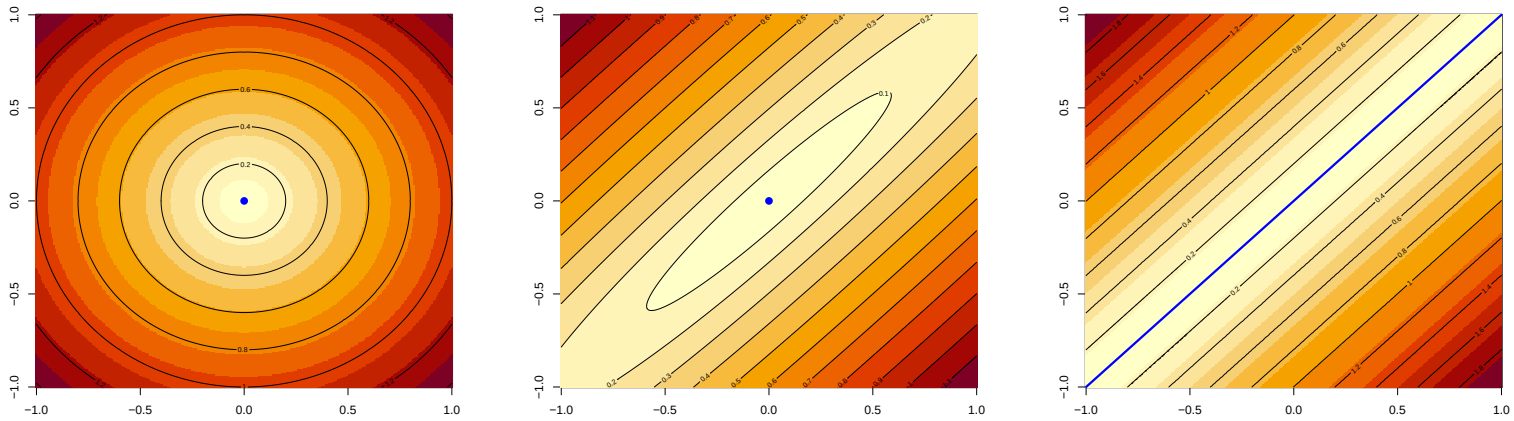
Theorem

When the square matrix $\mathbb{X}'\mathbb{X}$ is invertable, we have a unique solution to (1),

$$\hat{\boldsymbol{\beta}}_{\text{ls}} = (\mathbb{X}'\mathbb{X})^{-1}\mathbb{X}'\boldsymbol{y}.$$

The theorem is proved by taking gradient of the function $S(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbb{X}\boldsymbol{\beta}\|^2$. In fact, $\nabla S(\boldsymbol{\beta}) = 2\mathbb{X}'(\mathbb{X}\boldsymbol{\beta} - \boldsymbol{y})$. We skip the proof.

What if the minimizer of the least squares is not unique?



We may set a preference (a penalty) together with least squares. For example the ridge regression ($\lambda > 0$)

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^{r+1}} \left\{ \|\boldsymbol{y} - \mathbb{X}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2 \right\},$$

which prefers $\boldsymbol{\beta}$ with a smaller Euclidean norm.

Theorem

$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (\mathbb{X}'\mathbb{X} + \lambda I)^{-1}\mathbb{X}'\boldsymbol{y}$. *(Again, we skip the proof.)*

Different preference functions/penalties lead to different algorithms

- LASSO (least absolute shrinkage and selection operator), $\lambda > 0$.

$$\hat{\boldsymbol{\beta}}_{\mathsf{LASSO}} = \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^{r+1}} \left\{ \|\boldsymbol{y} - \mathbb{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1 \right\},$$

  where $\|\boldsymbol{\beta}\|_1 := |\beta_0| + \cdots + |\beta_r|$ is the 1-norm of vectors. Note that however, the term $|\beta_0|$ is usually removed from the penalty.

- Elastic Net, $\lambda_1, \lambda_2 > 0$.

$$\hat{\boldsymbol{\beta}}_{\mathsf{EN}} = \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^{r+1}} \left\{ \|\boldsymbol{y} - \mathbb{X}\boldsymbol{\beta}\|^2 + \lambda_1 \|\boldsymbol{\beta}\|_1 + \lambda_2 \|\boldsymbol{\beta}\|^2 \right\}.$$

- Set preference to some existing vector $\boldsymbol{\xi}^*$. Here $\lambda > 0$.

$$\hat{\boldsymbol{\beta}}_{\boldsymbol{\xi}^*} = \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^{r+1}} \left\{ \|\boldsymbol{y} - \mathbb{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta} - \boldsymbol{\xi}^*\|^2 \right\}.$$

Prediction and Generalization

Recall that prediction is one of the most important purposes of linear regression. Let $(x_1, \ldots, x_r)'$ be a new instance. We denote $\boldsymbol{x} = (1, x_1, \ldots, x_r)'$. Then, the output $y = \boldsymbol{x}'\boldsymbol{\beta}^*$ of the linear model is predicted through

$$\hat{y} = \boldsymbol{x}'\hat{\boldsymbol{\beta}}.$$

Here, $\hat{\boldsymbol{\beta}}$ could be any estimator, for example, $\hat{\boldsymbol{\beta}}_{\text{ls}}$, $\hat{\boldsymbol{\beta}}_{\text{LASSO}}$, $\hat{\boldsymbol{\beta}}_{\text{EN}}$, among others. The power of precisely predicting the labels of new instances that may not be included in the training data, is referred to as generalization ability.

A straightforward way to achieve generalization is to make sure that the estimation error $\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\|$ is small. Thanks to Cauchy's inequality,

$$|\boldsymbol{x}'\hat{\boldsymbol{\beta}} - \boldsymbol{x}'\boldsymbol{\beta}^*| = |\langle \boldsymbol{x}, \hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^* \rangle| \leq \|\boldsymbol{x}\| \cdot \|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\|,$$

for any $\boldsymbol{x} \in \mathbb{R}^{r+1}$. In particular, we have

Definition

The mean of squared error (MSE) of an estimator $\hat{\boldsymbol{\beta}}$ is defined by

$$\text{MSE}(\hat{\boldsymbol{\beta}}) = \mathbb{E}[\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\|^2].$$

We have the following decomposition of MSE.

Theorem

For any estimator $\hat{\boldsymbol{\beta}}$ *of* $\boldsymbol{\beta}_*$,

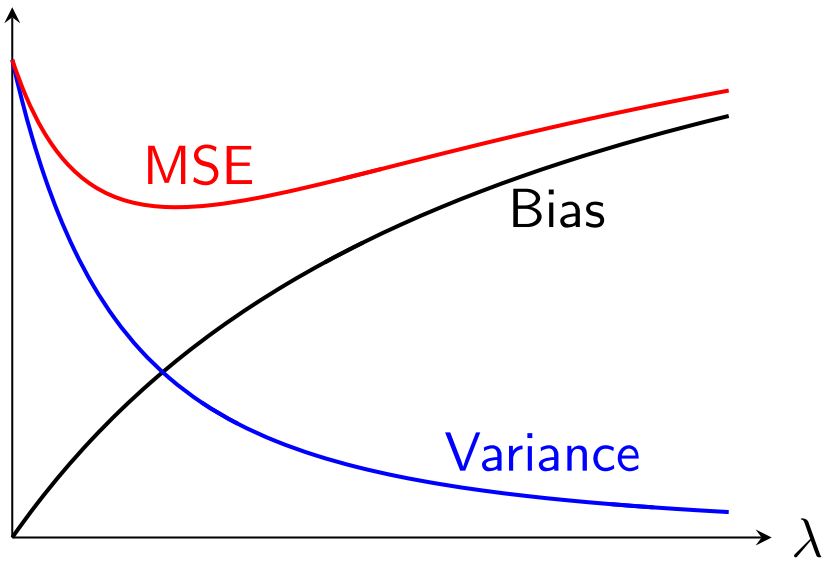
$$\mathrm{MSE}(\hat{\boldsymbol{\beta}}) = \|\mathrm{Bias}(\hat{\boldsymbol{\beta}})\|^2 + \mathrm{Trace}(\mathrm{Cov}(\hat{\boldsymbol{\beta}})),$$

where $\mathrm{Bias}(\hat{\boldsymbol{\beta}}) = \mathbb{E}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta}^*$ *is the bias, and the variance term* $\mathrm{Trace}(\mathrm{Cov}(\hat{\boldsymbol{\beta}}))$ *is the trace of the covariance matrix of* $\hat{\boldsymbol{\beta}}$.

For $\hat{\boldsymbol{\beta}}_{\text{ls}}$ and $\hat{\boldsymbol{\beta}}_{\text{Ridge}}$, we have (proof skipped)

$\hat{\boldsymbol{\beta}}$	"Bias", $\|\text{Bias}(\hat{\boldsymbol{\beta}})\|^2$	"Variance", $\text{Trace}(\text{Cov}(\hat{\boldsymbol{\beta}}))$
$\hat{\boldsymbol{\beta}}_{\text{ls}}$	0	$\sigma^2\text{Trace}((\mathbb{X}'\mathbb{X})^{-1})$
$\hat{\boldsymbol{\beta}}_{\text{Ridge}}$	$-\lambda(\mathbb{X}'\mathbb{X}+\lambda I)^{-1}\boldsymbol{\beta}^*$	$\sigma^2\text{Trace}\left[(\mathbb{X}'\mathbb{X})(\mathbb{X}'\mathbb{X}+\lambda I)^{-2}\right]$

- As the smallest singular value of $\mathbb{X}$ tends to zero, $\text{Trace}((\mathbb{X}'\mathbb{X})^{-1})$ diverges to infinity.
- **Bias Variance trade-off**



Example (collinearity). In this example, we build a design matrix $\mathbb{X}$ with two columns,

$$\mathbb{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2],$$

push one column towards the other,

$$\begin{aligned} \boldsymbol{x}_1 &= \boldsymbol{a}, \\ \boldsymbol{x}_2 &= \boldsymbol{x}_2(u) = u\boldsymbol{a} + (1-u)\boldsymbol{b}, \end{aligned}$$

where $0 \leq u \leq 1$. We would observe the process that the variance term tends to infinity.

The code is listed below.

```
set.seed(123)
n <- 30
a <- rnorm(n)
b <- rnorm(n)
f <- function(u){
  X <- cbind(a, u * a + (1 - u) * b)
  return(diag(solve(t(X) %*% X)))
}
print(f(0.6))
print(f(0.8))
print(f(0.9))
print(f(0.99))
```

The program output is listed below.

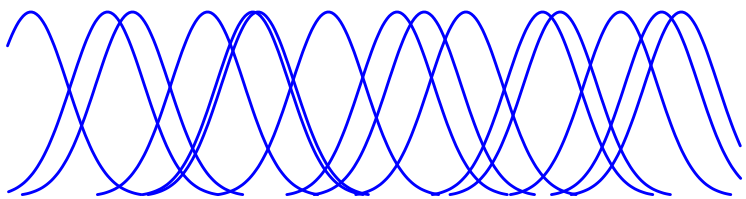
```
        a
0.1251278 0.3032513
        a
0.7576558 1.2130051
       a
3.842219 4.852020
       a
474.2121 485.2020
```

This test demonstrates how collinearity generates big “variance” term.

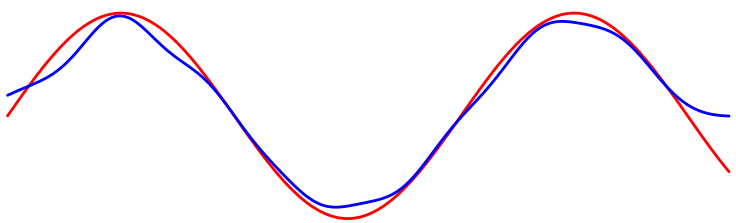
Kernel Methods

Consider a continuous function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. K is called a reproducing kernel (or Mercer kernel) on $\mathbb{R}^d$ if it is symmetric ($K(x,u) = K(u,x)$ for any $x, u \in \mathbb{R}^d$), and positive semi-definite (for any $x_1, \ldots, x_m \in X$, the Gram matrix $(K(x_i, x_j))_{1 \leq i,j \leq m}$ is positive semi-definite). Examples of reproducing kernels:

- linear kernel, $K(x,u) = \langle x, u \rangle$ on $\mathbb{R}^d$;
- Gaussian kernel, $K(x,u) = \exp\left(-\frac{1}{2\sigma^2}\|x-u\|^2\right)$ on $\mathbb{R}^d$, where $\sigma > 0$;
- Polynomial kernel $K(x,u) = (1 + \langle x, u \rangle)^r$ on $\mathbb{R}^d$, where $r \geq 1$ is an integer.
- Inverse multiquadrics, $K(x,u) = \left(c^2 + \|x-u\|^2\right)^{-\alpha}$ on $\mathbb{R}^d$, for any $c, \alpha > 0$.



functions $K(x_i, \cdot)$



$\sum c_i K(x_i, \cdot) \sim f(\cdot)$

Kernel Ridge Regression

- Model: $y = f_*(x) + \epsilon$, where $x \in \mathbb{R}^d$ is a vector of predictors, $\epsilon \in \mathbb{R}$ is a random noise, $y \in \mathbb{R}$ is the observed value of the dependent variable, and f_* is a function (e.g. nonlinear).
- Data: $\{(x_i, y_i)\}_{i=1}^m$, assumed sampled from the above model, so $y_i = f_*(x_i) + \epsilon_i$ for $1 \leq i \leq m$.
- Reproducing kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. Note that it might not be easy to find a kernel that matches the model well.
- Algorithm output: $\hat{f}_{\text{krr}}$, a function on $\mathbb{R}^d$ defined by

$$\hat{f}_{\text{krr}}(x) = \sum_{i=1}^{m} c_i^* K(x_i, x),$$

where $\boldsymbol{c}^* = (c_1^*, c_2^*, \ldots, c_m^*)$ is the vector obtained from the training process (i.e., the process of solving an optimization problem)

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c} \in \mathrm{R}^m} \left\{ \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{j=1}^{m} c_j K(x_i, x_j) - y_i \right)^2 + \lambda \sum_{i,j=1}^{m} c_i c_j K(x_i, x_j) \right\}$$

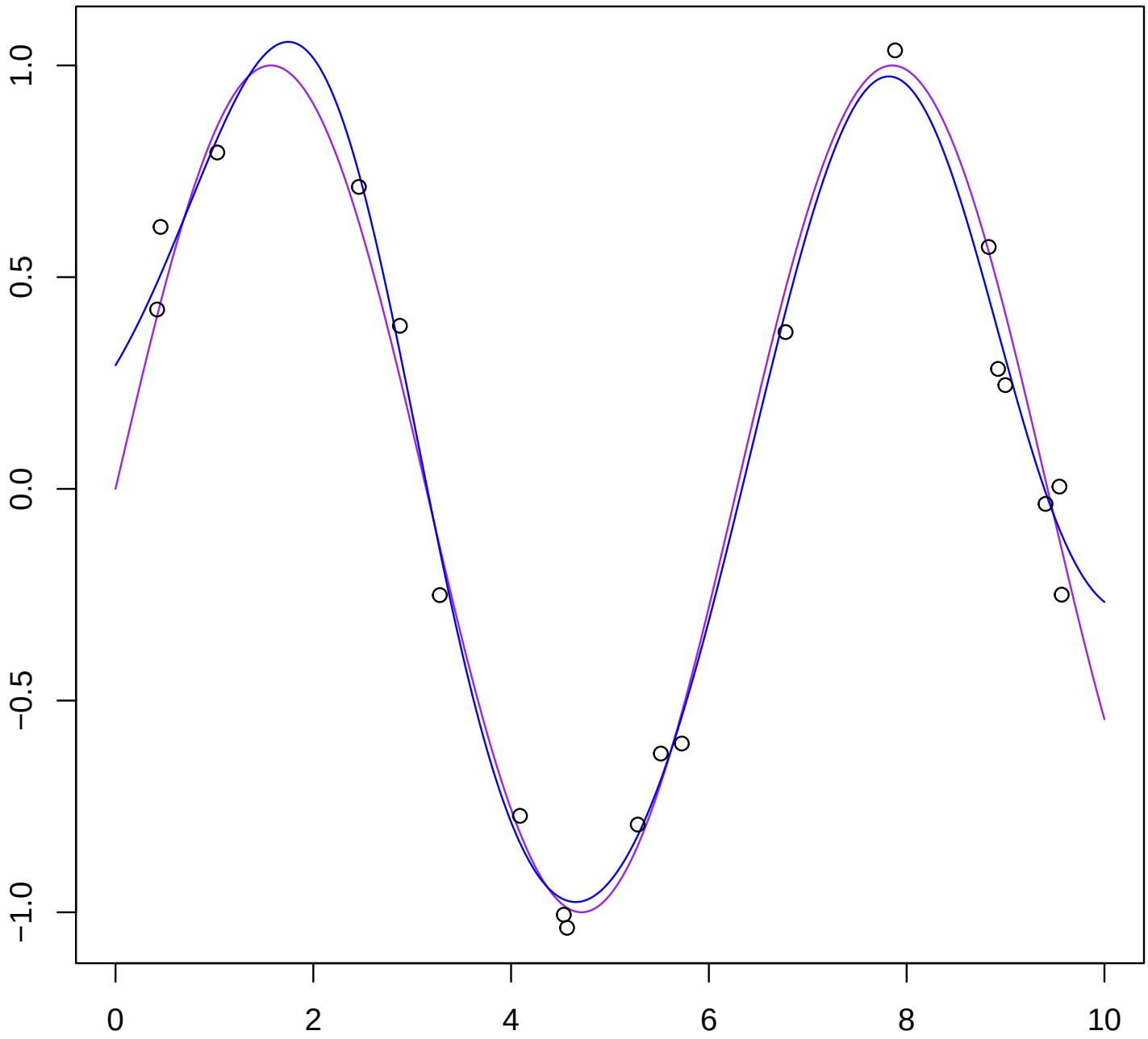
where $\lambda > 0$ is the regularization parameter.

Unfortunately it is difficult to find implementations of kernel ridge regression in R. In package "`e1071`", a similar support vector regression algorithm is implemented with

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}\in \mathrm{R}^m} \left\{ \frac{1}{m}\sum_{i=1}^{m} \left| \sum_{j=1}^{m} c_j K(x_i, x_j) - y_i \right| + \frac{1}{C}\sum_{i,j=1}^{m} c_i c_j K(x_i, x_j) \right\},$$

where $C > 0$ plays the role of regularization.
Please note that specific implementations may introduce some technical changes to the algorithms for good.

```r
library("e1071")

### generating artificial data
set.seed(123)
N <- 20
x.train <- runif(n = N, min = 0, max = 10)
y.train <- sin(x.train) + rnorm(n = N, sd = 0.1)
x.test <- seq(0, 10, len = 300)
y.oracle <- sin(x.test)
dt.trn <- as.data.frame(cbind(y.train, x.train))
dt.tst <- as.data.frame(x.test)
colnames(dt.trn) <- c("y", "x")
colnames(dt.tst) <- "x"

### training
gSeq <- function(from, to, len) exp(seq(log(from), log(to), len = len))
a <- tune.svm(y ~ x, data = dt.trn,
  gamma = gSeq(1, 10, 10), cost = gSeq(1, 10, 10))
print(summary(a$best.model))

### prediction and visualization
y.predict <- predict(a$best.model, dt.tst)
yRange <- range(c(y.train, y.oracle, y.predict))
plot(c(0, 10), yRange, type = "n", xlab = "", ylab = "")
lines(x.test, y.oracle, col = "purple")
lines(x.test, y.predict, col = "blue")
points(x.train, y.train)
```


1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10

A test on real data, "mtcars"

The dataset "mtcars" is provided in R.

```
mtcars                package:datasets                R Documentation
Motor Trend Car Road Tests

Description:
     The data was extracted from the 1974 _Motor Trend_ US magazine,
     and comprises fuel consumption and 10 aspects of automobile design
     and performance for 32 automobiles (1973-74 models).

Usage:
     mtcars

Format:
     A data frame with 32 observations on 11 (numeric) variables.

       [, 1]  mpg   Miles/(US) gallon
       [, 2]  cyl   Number of cylinders
       [, 3]  disp  Displacement (cu.in.)
       [, 4]  hp    Gross horsepower
       [, 5]  drat  Rear axle ratio
       [, 6]  wt    Weight (1000 lbs)
       [, 7]  qsec  1/4 mile time
       [, 8]  vs    Engine (0 = V-shaped, 1 = straight)
       [, 9]  am    Transmission (0 = automatic, 1 = manual)
       [,10]  gear  Number of forward gears
       [,11]  carb  Number of carburetors
......
```

```r
library("e1071")

dt.trn <- mtcars[-1, ]
dt.tst <- mtcars[1, ]

### training
gSeq <- function(from, to, len) exp(seq(log(from), log(to), len = len))
a <- tune.svm(mpg ~ disp + hp + drat + wt, data = dt.trn,
  gamma = gSeq(0.1, 5, 10), cost = gSeq(1, 10, 10))
print(summary(a$best.model))

### prediction
print(predict(a$best.model, dt.tst))
print(dt.tst)
```

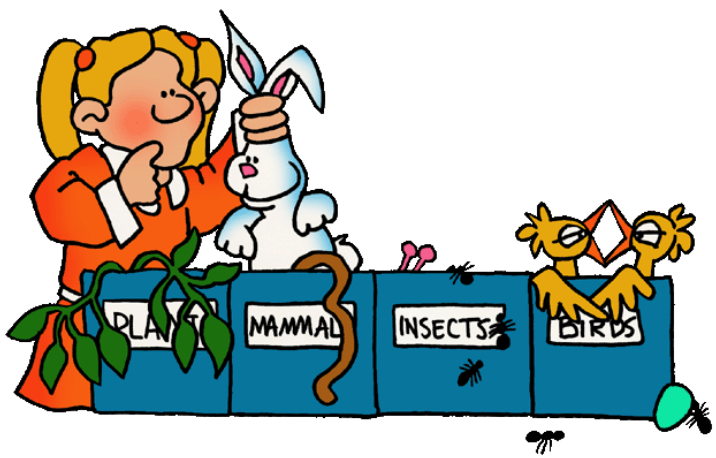
```
$ Rscript a.R
Call:
best.svm(x = mpg ~ disp + hp + drat + wt, data = dt.trn,
  gamma = gSeq(0.1, 5, 10), cost = gSeq(1, 10, 10))
Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1.668101
      gamma:  0.1
    epsilon:  0.1
Number of Support Vectors:  28

Mazda RX4
 22.47167
          mpg cyl disp  hp drat   wt  qsec vs am gear carb
Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

Classification

Suppose $\{x_i, y_i\}_{i=1}^n$ is a sample, where x_i's are from some set/space, and $y_i \in \{-1, 1\}$ are labels. Suppose the sample is drawn from some unknown yet fixed probability measure, or in another way of saying, some existing and fixed pattern.

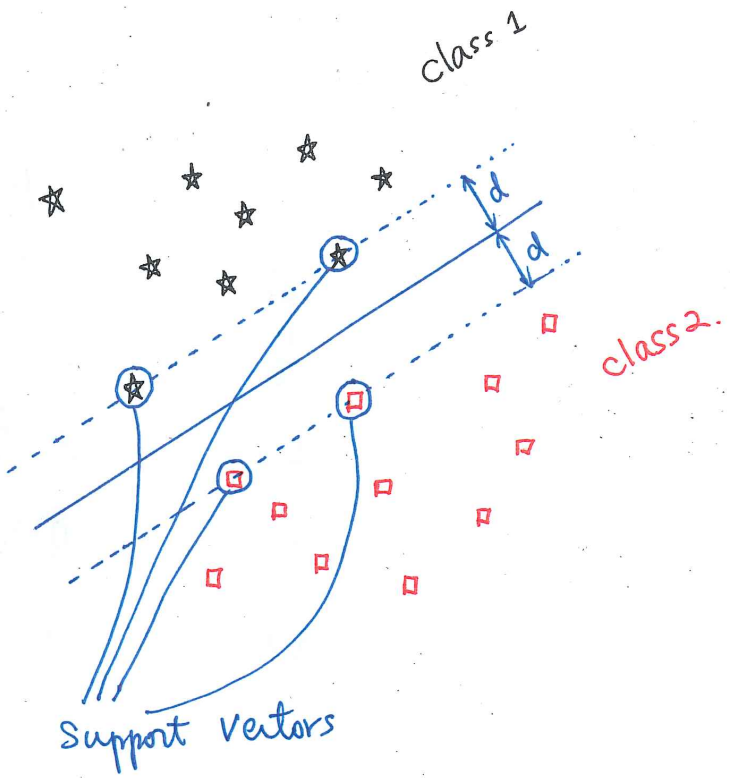
The classification task is to find some function f from the set where we draw x_i's, to the label set $\{-1, 1\}$, so that when some new datum x comes without label, we can have a large confidence that $f(x)$ is the right label that x should have. The function f is also called the **Classification Model**.

There are also "multi-class" classification tasks, where the label set $\{1, 2, \ldots, k\}$ has more than two elements.



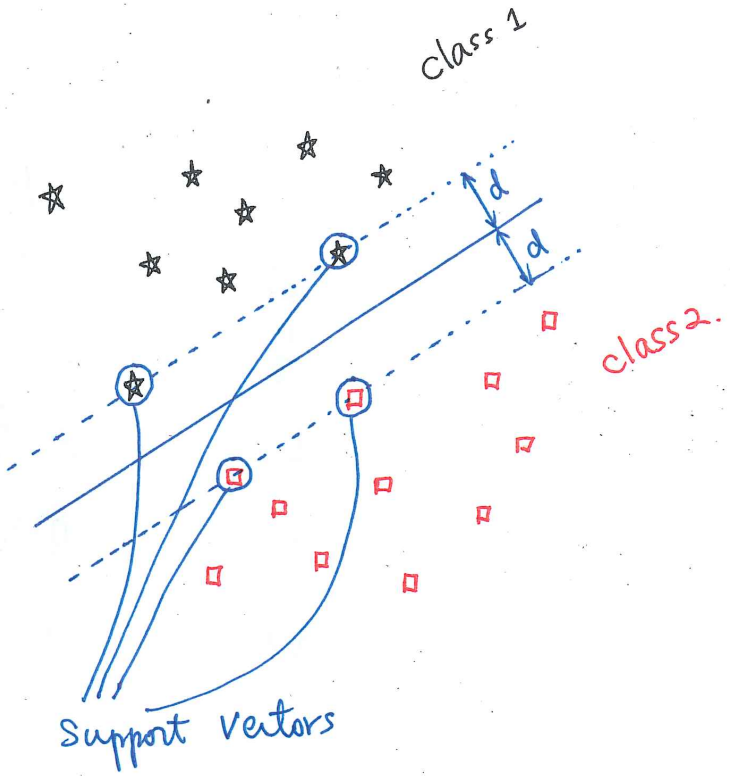
picture from: http://www.clipartpanda.com/categories/classification-20clipart.

Dividing points from two classes with a hyperplane



In the above picture, the solid blue line, a hyperplane, divides a point cloud into two parts according to their class label. The minimum distance between the hyperplane and the points in class 1 (called the *margin*) is d; for class 2, this minimum distance is also d.
The points that achieve this minimum are called *support vectors*.

Dividing points from two classes with a hyperplane



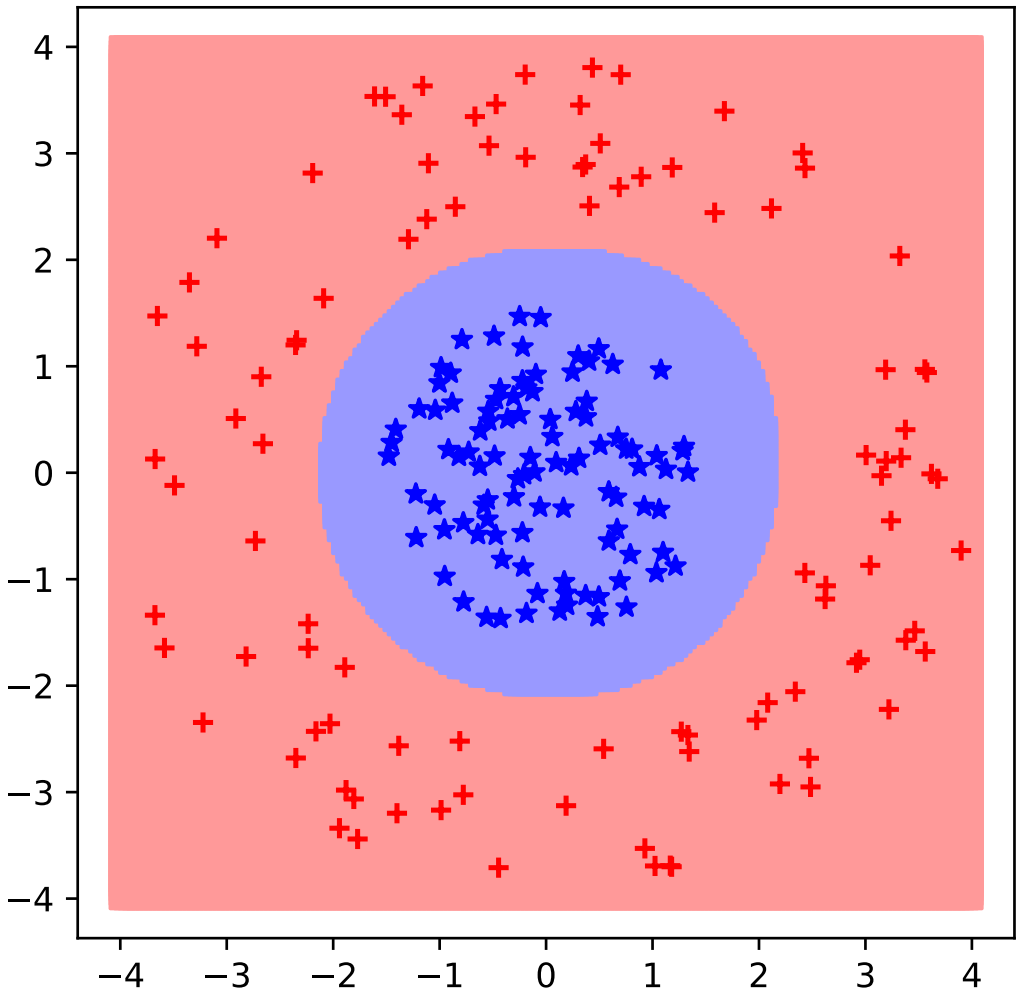
The blue hyperplane is found so that the margin of classification is maximized.
If you change the position, or twist the hyperplane a little, the margin is reduced.

- Recall the data $\{(x_i, y_i)\}_{i=1}^n$ for classification, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. For any i,
  - when $y_i = -1$, the vector x_i belongs to the first class;
  - when $y_i = 1$, the vector x_i belongs to the second class.
- **Support Vector Machine (SVM)** tries to find a function $f : \mathbb{R}^d \to \mathbb{R}$, so that its sign can be used for classification:

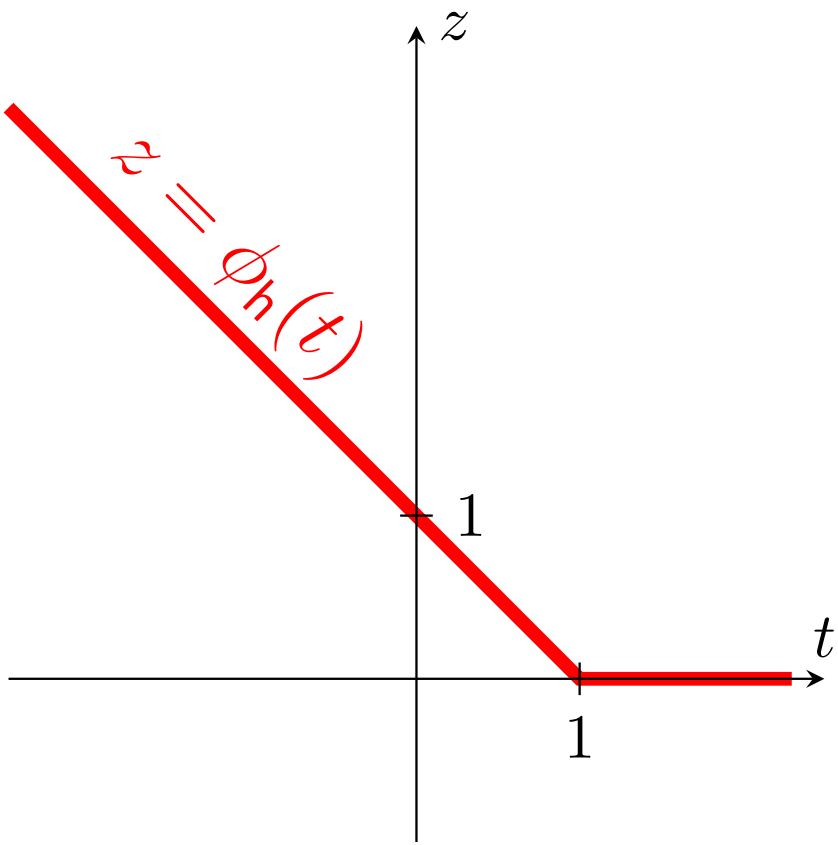
$$\text{sign}(f(x)) = \begin{cases} -1, & \text{when } f(x) < 0, \Rightarrow x \in \text{Class1}, \\ 1, & \text{when } f(x) \geq 0, \Rightarrow x \in \text{Class2}. \end{cases}$$

- An efficient way to achieve this, is to make sure $y_i f(x_i) > 0$ for all i.
- When f is a linear function (excluding the zero function $f \equiv 0$), the classification boundary $\{x \in \mathbb{R}^d : f(x) = 0\}$ is a hyperplane.

When $f(x) = \sum_{i=1}^{n} c_i^* K(x_i, x)$ is a linear combination of reproducing kernel functions, the classification boundary may be a nonlinear curve (or hypersurface in high-dimensional space).



- SVM tries to make sure $y_i f(x_i) > 0$ for all i, by setting a loss $\phi_{\mathsf{h}}(y_i f(x_i))$, where $\phi_{\mathsf{h}}(t) = \max\{0, 1 - t\}$ is called the **hinge loss**. The loss vanishes if $y_i f(x_i)$ is large enough: $y_i f(x_i) \geq 1$.



Support Vector Machine (SVM) with linear functions

- Input data: $\{(x_i, y_i)\}_{i=1}^m$, where $x_i \in \mathbb{R}^d$, and $y_i \in \{-1, 1\}$
- Output classifier: $\text{sign}(f^*(x))$, with

$$f^* = \arg\min_f \left\{ \frac{1}{m} \sum_{i=1}^m \phi_{\mathsf{h}} \left(y_i f(x_i) \right) + \gamma \Omega(f) \right\},$$

where $\gamma > 0$ is the regularization parameter, and the minimum is taken over all the linear functions f on $\mathbb{R}^d$ that have the form

$$f(x) = \langle \beta, x \rangle + c$$

where $\beta \in \mathbb{R}^d$ and $c \in \mathbb{R}$, with $\Omega(f) := \|\beta\|^2 = \beta_1^2 + \beta_2^2 + \cdots + \beta_d^2$.

SVM with reproducing kernels

Let K be a reproducing kernel on $\mathbb{R}^d$. Let $\gamma > 0$ be the regularization parameter. SVM with kernel K is defined by

- Input data: $\{(x_i, y_i)\}_{i=1}^m$, where $x_i \in \mathbb{R}^d$, and $y_i \in \{-1, 1\}$
- Output classifier: $\operatorname{sign}(f^*(x))$, with

$$f^* = \arg\min_f \left\{ \frac{1}{m} \sum_{i=1}^m \phi_{\mathsf{h}} \left(y_i f(x_i) \right) + \gamma \Omega(f) \right\},$$

where the minimum is taken over all the functions f on $\mathbb{R}^d$ that have the form

$$f(x) = c + \sum_{i=1}^m \beta_i K(x_i, x), \quad \text{with } \Omega(f) := \sum_{i=1}^m \sum_{j=1}^m \beta_i \beta_j K(x_i, x_j).$$

The “factor” data type in R

R uses a special data type “factor” to represent categorical data, which could be divided into (usually a small number of) groups. Examples include age group, race, sex, postal code, and educational level.
In R, the data type factor is internally stored as integer, and printed as character (string).

Example: The Iris Dataset

The Iris Dataset

- Introduced by R. A. Fisher[1]. For a full description, see the web page in the UCI Machine Learning Repository, `https://archive.ics.uci.edu/ml/datasets/iris`.

- Attribute Information:
  1. sepal length in cm
  2. sepal width in cm
  3. petal length in cm
  4. petal width in cm
  5. class:
     - Iris Setosa
     - Iris Versicolour
     - Iris Virginica

Picture from: https://archive.ics.uci.edu/ml/datasets/iris

[1] Fisher, Ronald A. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics* 7.2 (1936): 179-188.

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> str(iris)
'data.frame':^^I150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
> typeof(iris[, "Species"])
[1] "integer"
> unique(iris[, "Species"])
[1] setosa     versicolor virginica
Levels: setosa versicolor virginica
> write.table(iris, file = "iris.txt")
> irisDT <- read.table("iris.txt", stringsAsFactors = TRUE)
```

Now we use this real dataset to try SVM.

```
library("e1071")

### we provide the code for reading data from file
write.table(iris, file = "iris.txt", sep = ",", row.names = FALSE)
dataIRIS <- read.table("iris.txt", stringsAsFactors = TRUE,
  sep = ",", header = TRUE)
data.trn <- dataIRIS[-1, ]
data.tst <- dataIRIS[1, ]

### training
gSeq <- function(from, to, len) exp(seq(log(from), log(to), len = len))
a <- tune.svm(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = data.trn, gamma = gSeq(1, 10, 10), cost = gSeq(0.1, 5, 10))
print(summary(a$best.model))

### prediction
y.predict <- predict(a$best.model, data.tst)
print(y.predict)
```

The program output is listed below.

```
$ Rscript a.R

Call:
best.svm(x = Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
    data = data.trn, gamma = gSeq(1, 10, 10), cost = gSeq(0.1, 5, 10))


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.3684031

Number of Support Vectors:  77

 ( 17 31 29 )


Number of Classes:  3

Levels:
 setosa versicolor virginica



     1
setosa
Levels: setosa versicolor virginica
```

Usually the automatically selected kernel is good enough, but one may still specify a kernel themselves. Currently the package "`e1071`" implements four classes of kernels, "`linear`", "`polynomial`", "`radial basis`" (which is Gaussian kernel), and "`sigmoid`". Here, linear kernel recovers SVM with linear functions. For more details, please read the reference manual of the package,
`https://cran.r-project.org/web/packages/e1071/e1071.pdf`

```
library("e1071")

### we provide the code for reading data from file
write.table(iris, file = "iris.txt", sep = ",", row.names = FALSE)
dataIRIS <- read.table("iris.txt", stringsAsFactors = TRUE,
  sep = ",", header = TRUE)
data.trn <- dataIRIS[-1, ]
data.tst <- dataIRIS[1, ]

### training
gSeq <- function(from, to, len) exp(seq(log(from), log(to), len = len))
a <- tune.svm(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = data.trn, cost = gSeq(0.1, 5, 10), kernel = "linear")
print(summary(a$best.model))

### prediction
y.predict <- predict(a$best.model, data.tst)
print(y.predict)
```

The program output is listed below.

```
$ Rscript a.R

Call:
best.svm(x = Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
    data = data.trn, cost = gSeq(0.1, 5, 10), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.2385332

Number of Support Vectors:  47

 ( 3 24 20 )


Number of Classes:  3

Levels:
 setosa versicolor virginica



     1
setosa
Levels: setosa versicolor virginica
```

Logistic Regression

If we replace hinge loss $\phi_{\text{h}}(t) = \max\{0, 1-t\}$ by logistic loss $\phi_{\text{lg}}(t) = \ln(1+e^{-t})$, we obtain the Logistic Regression algorithm

- Input data: $\{(x_i, y_i)\}_{i=1}^m$, where $x_i \in \mathbb{R}^d$, and $y_i \in \{-1, 1\}$
- Output classifier: $\text{sign}(f^*(x))$, with

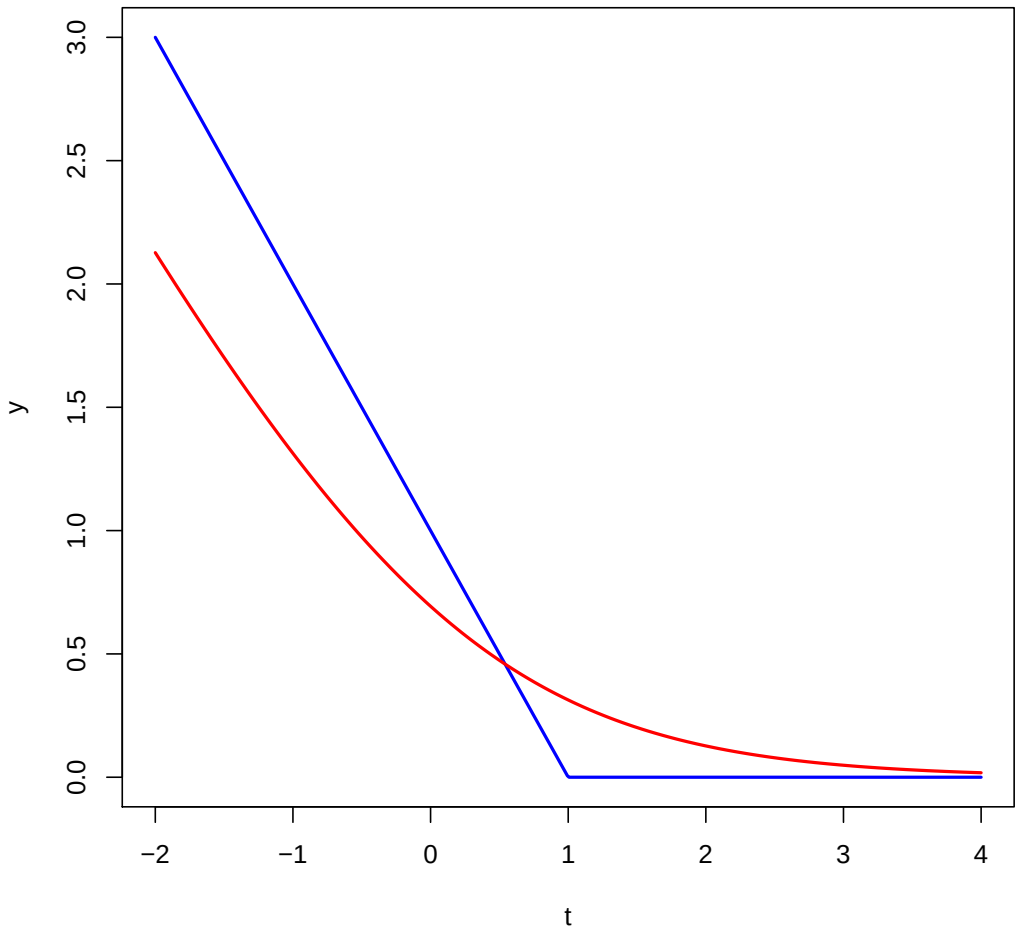
$$f^* = \arg\min_f \left\{ \frac{1}{m} \sum_{i=1}^m \phi_{\text{lg}}\left(y_i f(x_i)\right) + \gamma \Omega(f) \right\},$$

where $\gamma > 0$ is the regularization parameter, and the minimum is taken over all the linear functions f on $\mathbb{R}^d$ that have the form

$$f(x) = \langle \beta, x \rangle + c$$

where $\beta \in \mathbb{R}^d$ and $c \in \mathbb{R}$, with $\Omega(f) := \|\beta\|^2 = \beta_1^2 + \beta_2^2 + \cdots + \beta_d^2$.

Note that as $t \to \infty$, $e^{-t} \to 0$, so $\phi_{\text{lg}}(t) = \ln(1 + e^{-t}) \to 0$. As $t \to -\infty$, $\phi_{\text{lg}} = \ln e^{-t} + \ln(e^t + 1) = -t + \ln(e^t + 1)$, where $\ln(e^t + 1) \to 0$, so $\phi_{\text{lg}}(t) \approx -t$ as $t \to -\infty$. The following figure compares ϕ_{lg} (the red line) and ϕ_{h} (the blue line).



The Python package "`sklearn`" provides a nice implementation of logistic regression. It is, however, difficult to find a similar one in R. We only provide a mature implementation with the "`glm`" function (standing for *generalized linear model*).

```
set.seed(123)
gen.data <- function(s1, s0, d){
  y <- c(rep(1, s1), rep(0, s0))
  x <- matrix(rnorm((s1 + s0) * d), ncol = d)
  x[,1] <- x[,1] + 4 * y - 2
  the.data <- cbind(y, as.data.frame(x))
  colnames(the.data) <- c("y", paste0("x", 1:d))
  return(the.data)
}
d.trn <- gen.data(300, 300, 3)
d.tst <- gen.data(100, 50, 3)

model <- glm(y ~ x1 + x2 + x3, family = binomial, data = d.trn)
pr <- predict(model, newdata = d.tst)
```

The artificial data is generated that the intercept, the variables x_2 and x_3 are all irrelevant to the classification. We see the consistent inference results below.

```
> summary(model)
Call:
glm(formula = y ~ x1 + x2 + x3, family = binomial, data = d.trn)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.18246  -0.00166   0.00000   0.00367   2.52705

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.7977     0.5207  -1.532    0.126
x1            6.4220     1.4088   4.558 5.16e-06 ***
x2           -0.2250     0.5050  -0.445    0.656
x3           -0.7091     0.5525  -1.283    0.199
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 831.777  on 599  degrees of freedom
Residual deviance:  34.772  on 596  degrees of freedom
AIC: 42.772

Number of Fisher Scoring iterations: 11
```

Note that the above implementation takes no regularization (i.e., using $\gamma = 0$), so the implementation may suffer from numerical problems. The good side is that like linear model, the program output provides much information on p-value, confidence interval, and so on.

Score-based Classifiers

Recall that last time the output classifier of SVM is based on a function (linear or nonlinear) $f_{\mathbf{z}}(x)$ which has real values. The classifier we defined is $\mathrm{sign}(f_{\mathbf{z}}(x))$. Equivalently speaking,

$$f_{\mathbf{z}}(x) \geq 0 \Rightarrow x \in \text{ class 1, or "Positive",}$$
$$f_{\mathbf{z}}(x) < 0 \Rightarrow x \in \text{ class 2, or "Negative".}$$

Sometimes, we hope to give one class some preference. For example, when we design a fire alarm, it won't hurt too much if the alarm bell rings when there is indeed no fire; but not conversely!!

Therefore we set a threshold λ, and the above design of classifier is only a special case of $\lambda = 0$.

$$f_{\mathbf{z}}(x) \geq \lambda \Rightarrow x \in \text{ class 1, or "Positive",}$$
$$f_{\mathbf{z}}(x) < \lambda \Rightarrow x \in \text{ class 2, or "Negative".}$$

For a specified λ, let $f_{\mathcal{C}}(x) = \text{P}$ when $f_{\mathbf{z}}(x) \geq \lambda$, and $f_{\mathcal{C}}(x) = \text{N}$ when $f_{\mathbf{z}}(x) < \lambda$. We have the following table of errors.

		TRUTH: Positive	TRUTH: Negative
PREDICTED RESULTS	Positive	True Positive (TP)	False Positive (FP) type I error
	Negative	False Negative (FN) type II error	True Negative (TN)

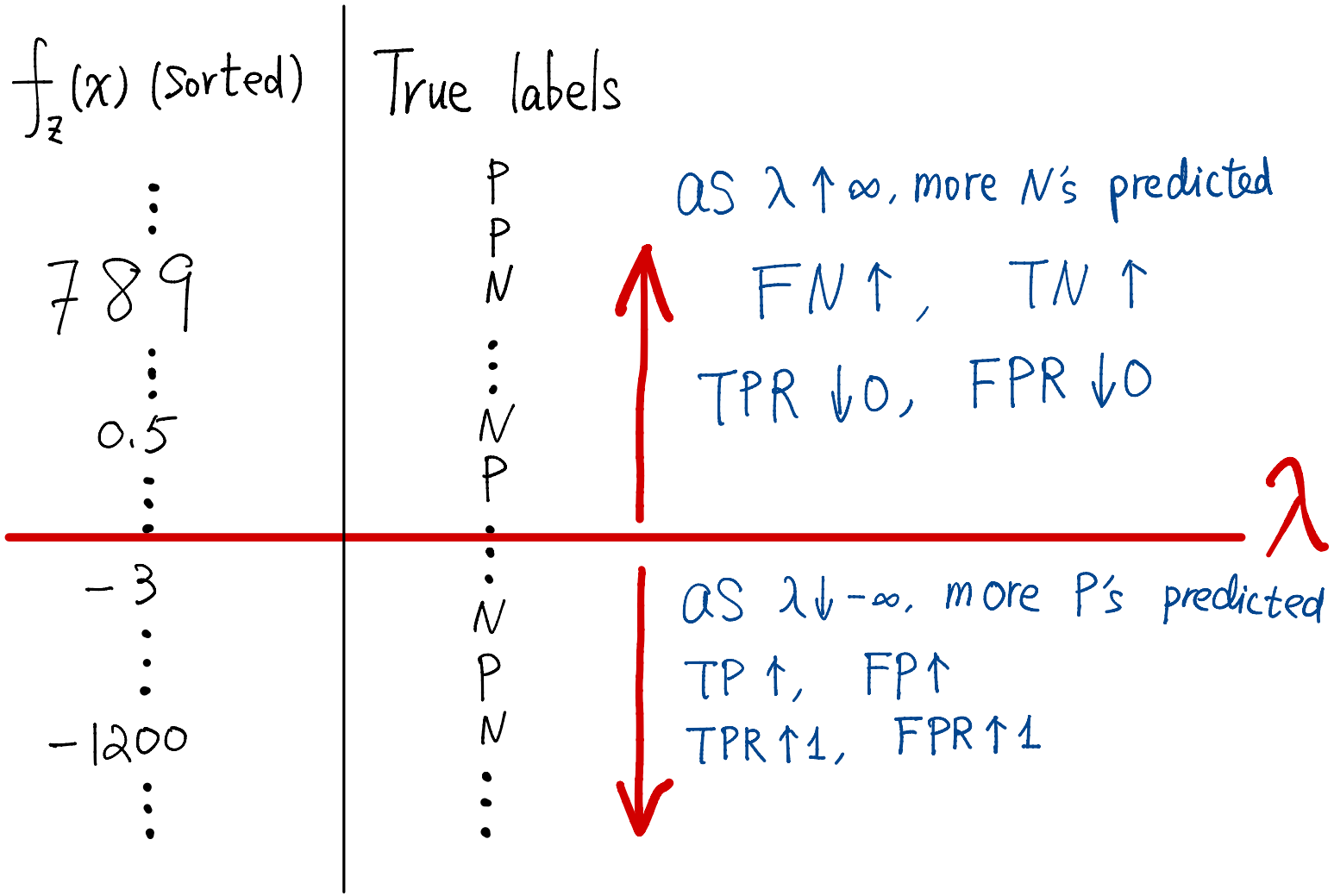
- True positive rate (also called Sensitivity): $\text{TPR} = \frac{TP}{TP+FN}$
- False positive rate: $\text{FPR} = \frac{FP}{FP+TN}$
- Specificity: $1 - \text{FPR} = \frac{TN}{FP+TN}$

Example

For the following classification prediction, compute the TPR and FPR.

Predicted	True Label
P	P
P	N
N	P
P	P
N	N
N	N
N	N
N	N
N	P
P	P
N	P

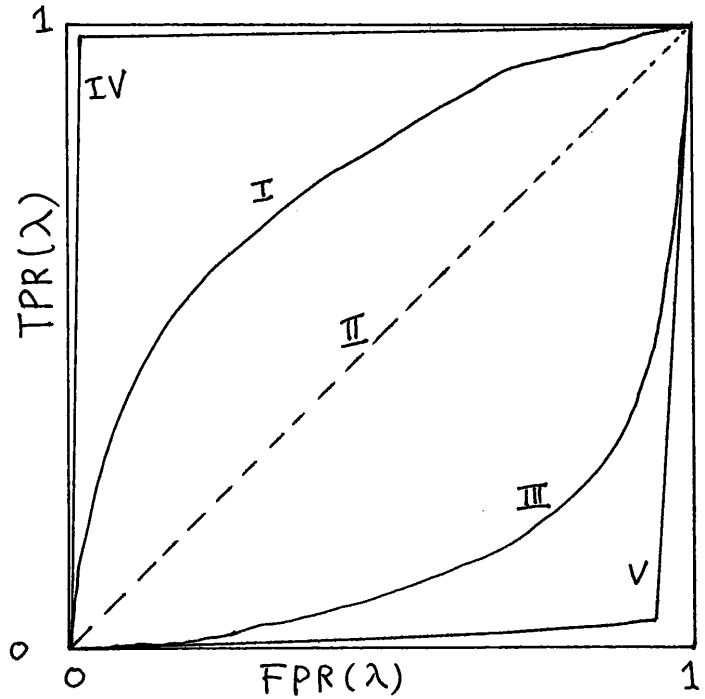
Recall: $\text{TPR} = \frac{TP}{TP+FN}$ and $\text{FPR} = \frac{FP}{FP+TN}$. Note that $TP + FN$ and $FP + TN$ are both independent of λ.



For a specified λ, let $f_{\mathcal{C}}(x) = \text{P}$ when $f_{\mathbf{z}}(x) \geq \lambda$, and $f_{\mathcal{C}}(x) = \text{N}$ when $f_{\mathbf{z}}(x) < \lambda$.

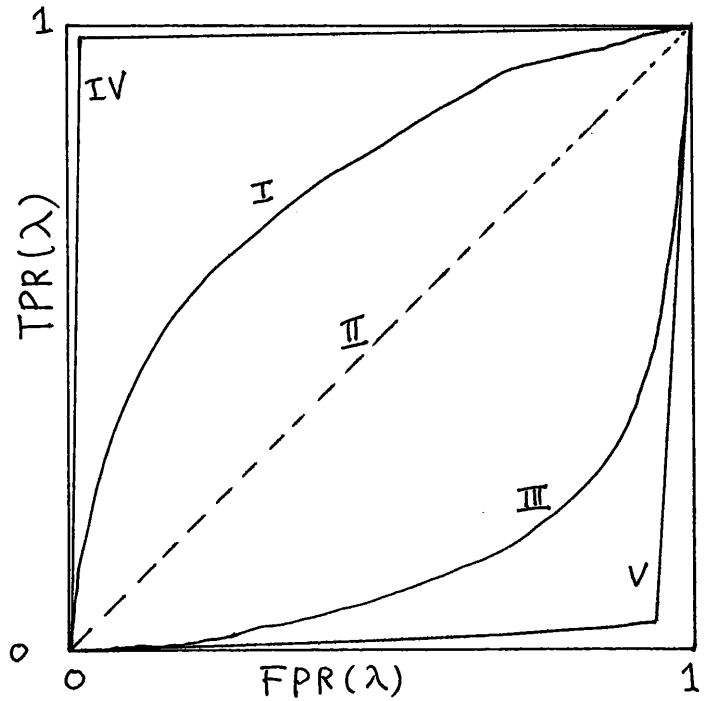
- We see that for $\lambda = -\infty$, the classifier predicts everything as positive. By the previous definition, $TPR = TPR(\lambda) = 1$, and $FPR = FPR(\lambda) = 1$. Of course, this is just a useless predictor.
- We see that for $\lambda = \infty$, the classifier predicts everything as negative. Then $TPR = TPR(\lambda) = 0$, and $FPR = FPR(\lambda) = 0$. Of course, this is another useless predictor.
- We see that as λ changes from ∞ to $-\infty$, $FPR(\lambda)$ and $TPR(\lambda)$ both increase.

The Receiver Operating Characteristics (ROC) Curve



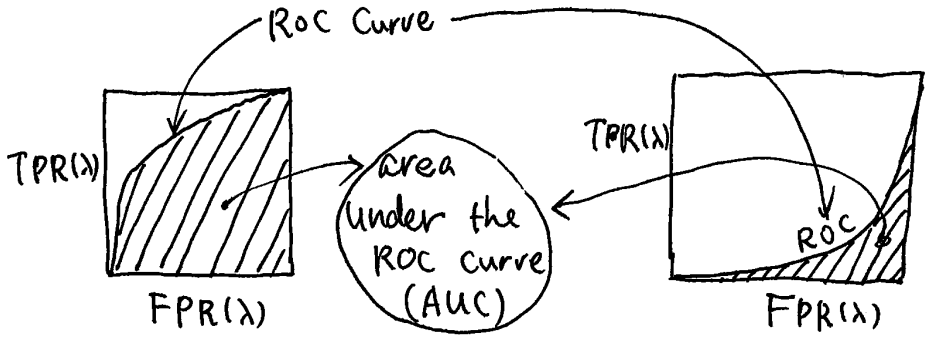
"The Receiver Operating Characteristics (ROC) curves were originally developed in signal detection theory in connection with radio signals, and have been used since then in many other applications, in particular for medical decision-making. Over the last few years, they have found increased interest in the machine learning and data mining communities for model evaluation and selection." Cited from C. Cortes and M. Mohri, NIPS 2004

The Receiver Operating Characteristics (ROC) Curve



Here we have five ROC curves. Curve I is typical. Curve II indicates a total random hence useless classifier, since for any specific λ, it always predict the actually positive observations as P with probability 0.5, and N with probability 0.5; same for the actually negative observations. Curve III is kind of useful in the sense that one benefits by using it reversely. IV is nearly a perfect classifier, so is V.

The Area Under ROC Curve (AUC)

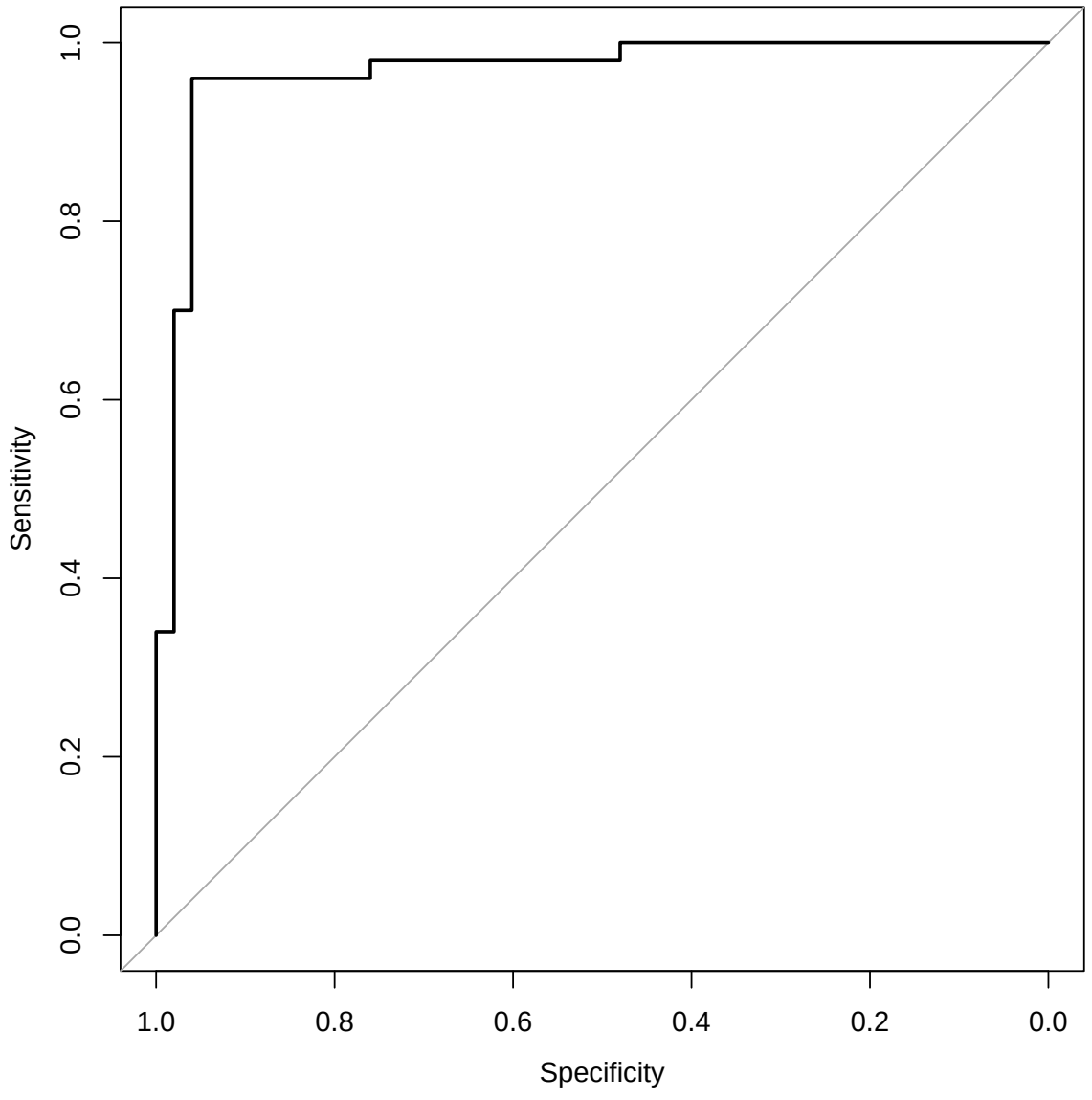


- AUC is defined as the area under the ROC curve. So an AUC score ranges from 0 to 1.
- AUC could be used to evaluate the performance of a classifier.
- AUC socre close to 1 indicates a good classifier.
- AUC close to 0 indicates a good classifier yet one should use it reversely.
- AUC close to 0.5 indicates a bad classifier. In particular, AUC≈ 0.5 means the classifier has nearly the same performance like a random guess. Note that a random guess is also a classifier, although with poor performance.
- AUC is designed for two-class classification problems only.

In the following program, we use the first 100 observations in the iris dataset (recall that in this dataset, there are three classes, while the first 100 observations correspond to the first two classes), add some noise by altering some class labels randomly, train SVM for classification, obtain the decision values (the $f_{\mathbf{z}}(x)$ scores), plot the ROC curve, and evaluate the AUC score. Note that we use a `factor`→`character`→`factor` transform to restrict the class labels to two.

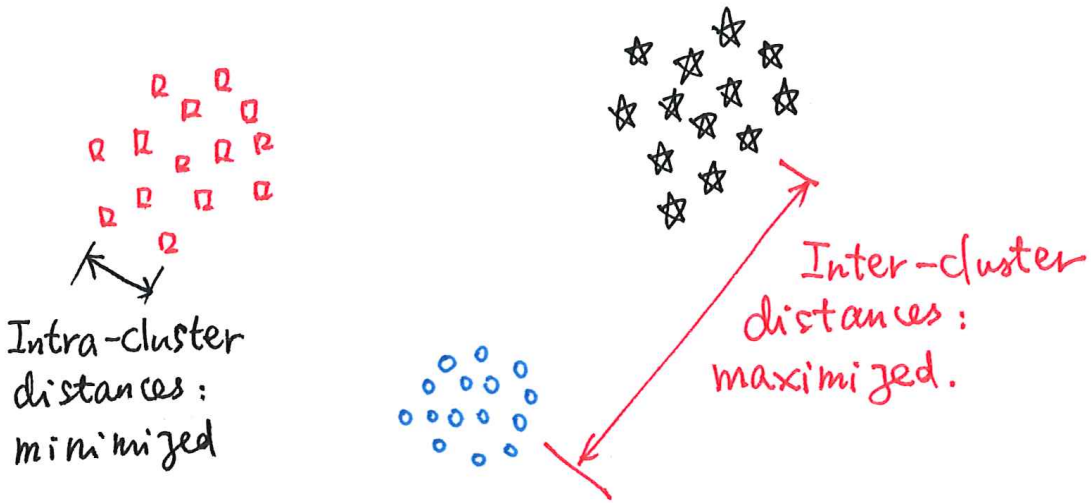
```
a <- iris[1:100, ]
a$Species <- as.factor(as.character(a$Species))
set.seed(123)
ind <- sample(1:100, 10)
a$Species[ind] <- sample(a$Species[ind])

library("e1071")
gSeq <- function(from, to, len) exp(seq(log(from), log(to), len = len))
w <- tune.svm(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = a, gamma = gSeq(1, 10, 10), cost = gSeq(0.1, 5, 10))
z <- predict(w$best.model, a, decision.values = TRUE)
dv <- c(attributes(z)$decision.values)

library("pROC")
### plotting ROC curve
plot(roc(predictor = dv, response = a$Species))
### computing AUC value
aucValue <- c(auc(predictor = dv, response = a$Species))
print(aucValue)
```

The ROC curve is plotted below.



Cluster Analysis

Cluster analysis, is to find groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups.



Notion of a cluster can be ambiguous. For most of the cluster analysis problems, it is an art to decide how many clusters to divide the sample into.

How many clusters?

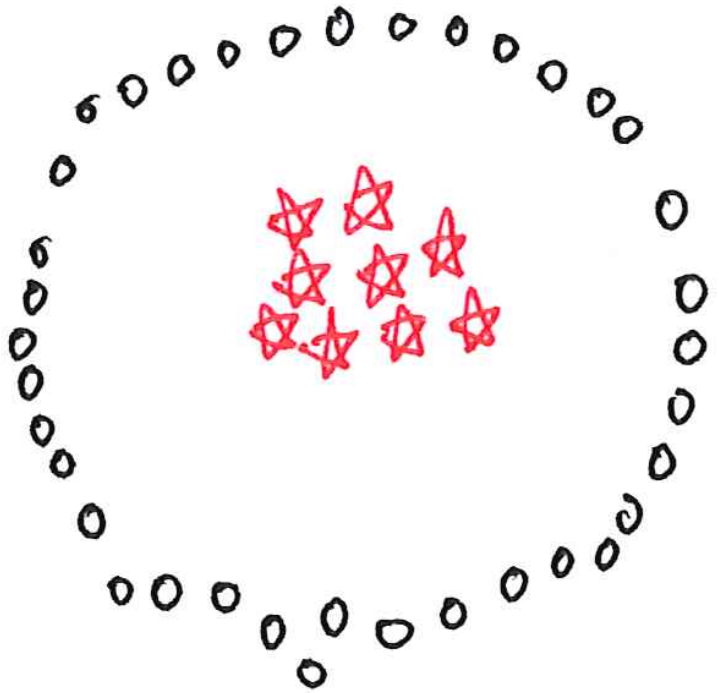
Six Clusters

Two Clusters

Four Clusters

Picture From: https://www-users.cse.umn.edu/~kumar001/dmbook/slides/chap7_basic_cluster_analysis.pdf

Intrinsic distance vs ambient distance



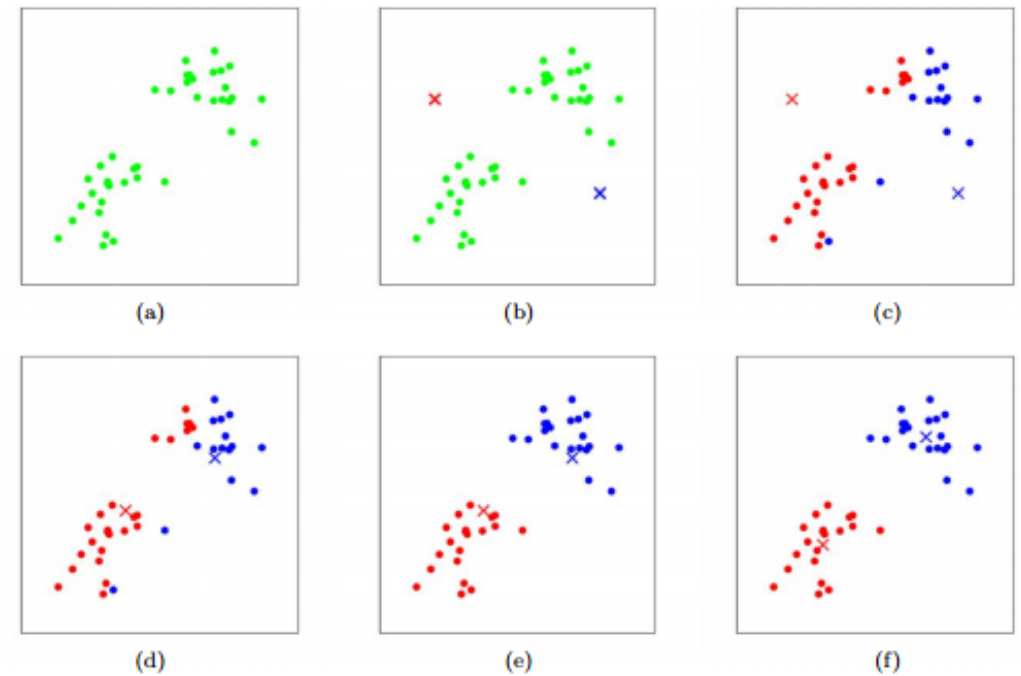
Although for any black circle, there is another black circle that is farther than a red star, we still wish to collect all the red stars as one group, and the black circles as another group. This is because the black circles are connected together by one another, forming a manifold structure.

K-means Clustering

Step 1 Select K points as the initial centroids.

Step 2 Form K clusters by assigning all points to the closest centroid.

Step 3 Recompute the centroid of each cluster. If all the centroid computed is the same as those before previous step 2, stop, otherwise go to step 2.



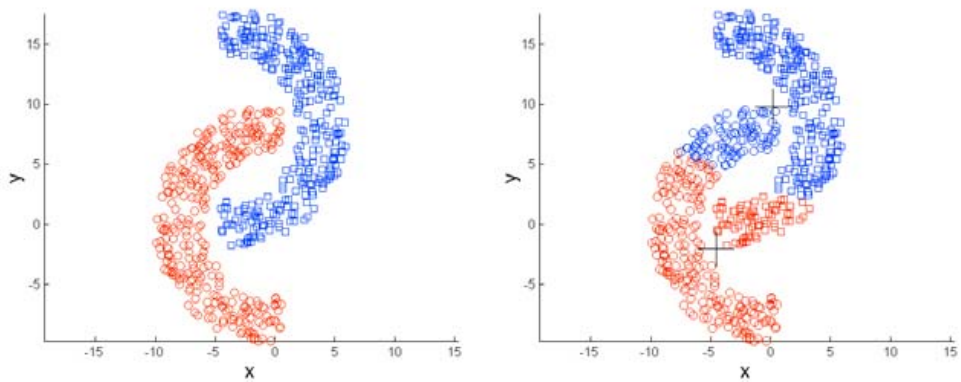
picture from https://stanford.edu/~cpiech/cs221/handouts/kmeans.html

Shortcomings

- Multiple runs: helps, but probability is not on your side
- May yield empty clusters. One may, for example, re-place the centroid of the empty cluster so that the following SSE is minimized

$$SSE = \sum_{i=1}^{K} \sum_{x \in Cluster_i} dist^2(Centoid_i, x)$$

- May get bad results when points are distributed on manifold yet ambient distance is used.

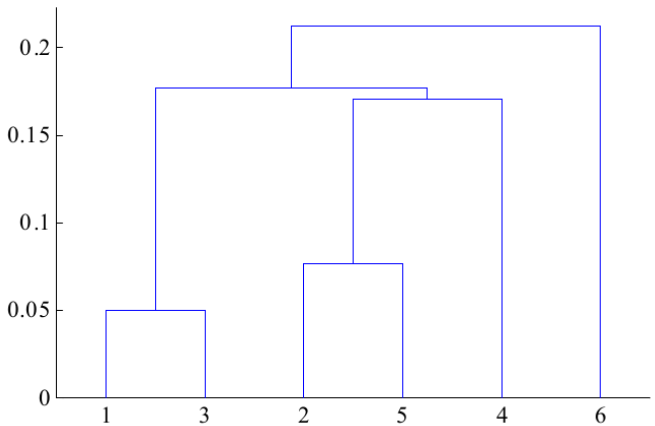


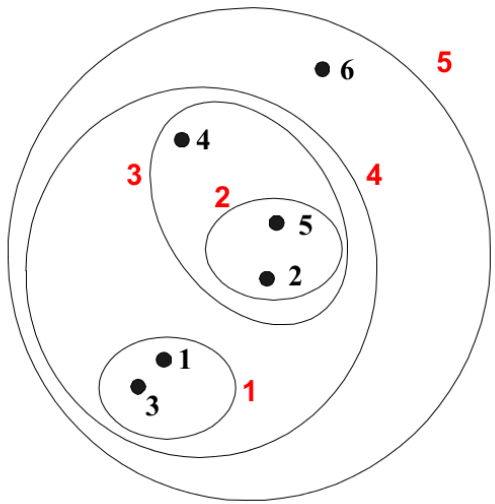
Original Points **K-means (2 Clusters)**

Picture from: https://www-users.cse.umn.edu/~kumar001/dmbook/slides/chap7_basic_cluster_analysis.pdf

Hierarchical Clustering

- Produces a set of nested clusters organized as a hierarchical tree
- Usually visualized as a dendrogram (cluster tree), which is a tree-like diagram that records the sequence of merges or splits.





Picture from: https://www-users.cse.umn.edu/~kumar001/dmbook/slides/chap7_basic_cluster_analysis.pdf

Strength of Hierarchical Clustering

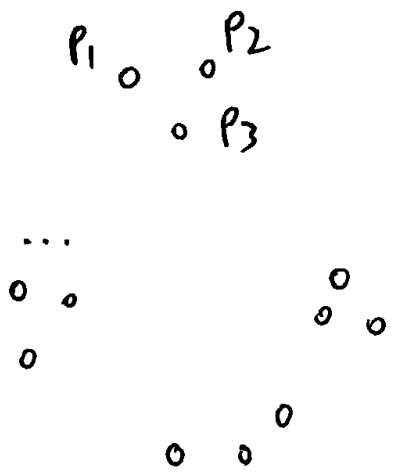
- One does not have to assume any particular number of clusters. One needs however to cut the dendrogram at a specific level to make the clusters.
- The dendrogram may well correspond to meaningful taxonomies.

Agglomerative Clustering Algorithm

- A polular hierarchical clustering algorithm
- The algorithm
  - Compute the proximity matrix
  - Let each data point be a cluster
  - **Repeat:**
    - Merge the two closest clusters
    - Update the proximity matrix
  - **Until:** only one single cluster remains.

Starting Situation

Start with clusters of individual points and a proximity matrix (which defines the distance between each pair of points).



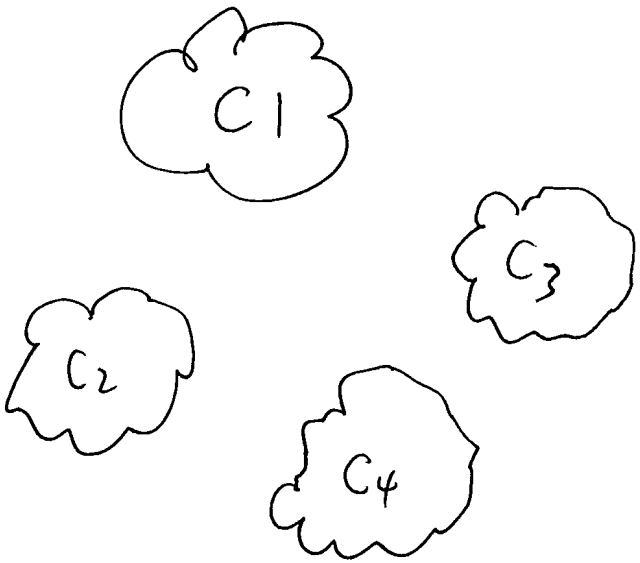
data

	p_1	p_2	p_3	p_4	...
p_1	0.2	0.5	0.3	0.4	
p_2	0.5	0.2			
p_3	0.3	...			
p_4		...		...	
⋮					

proximity matrix

Intermediate Situation

After some merging steps, we have some clusters, along with their proximity matrix.

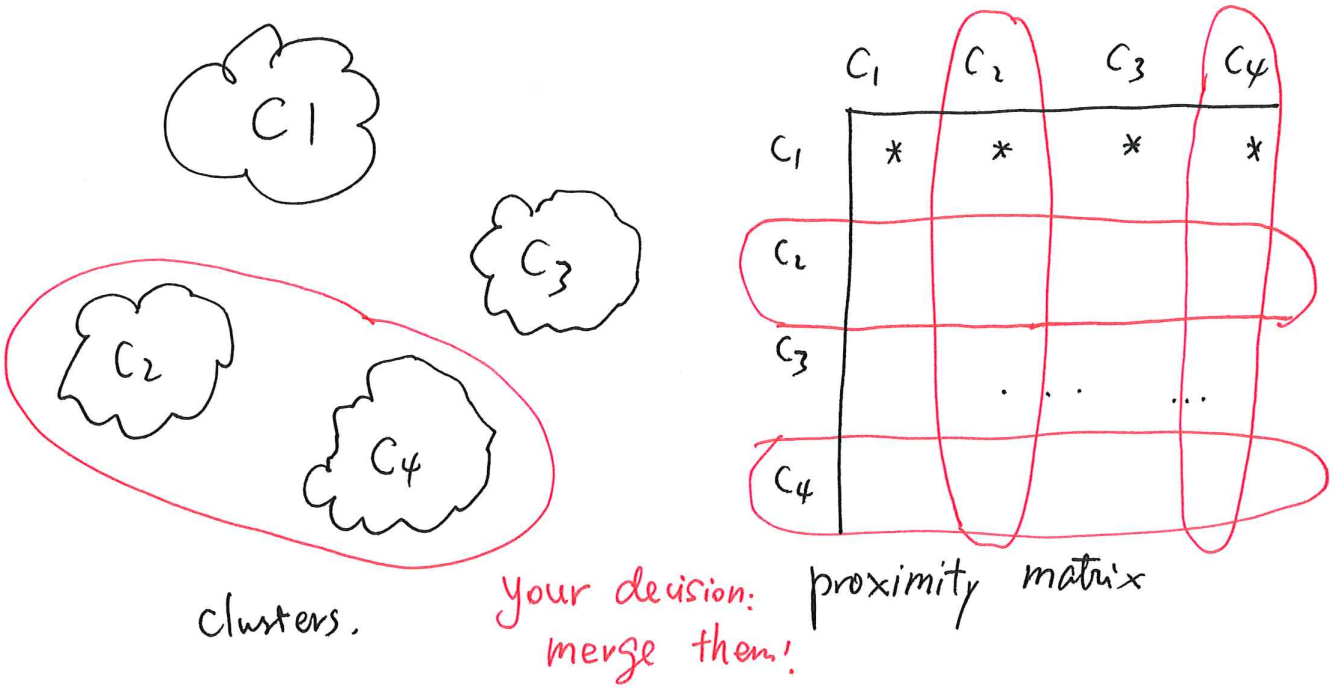


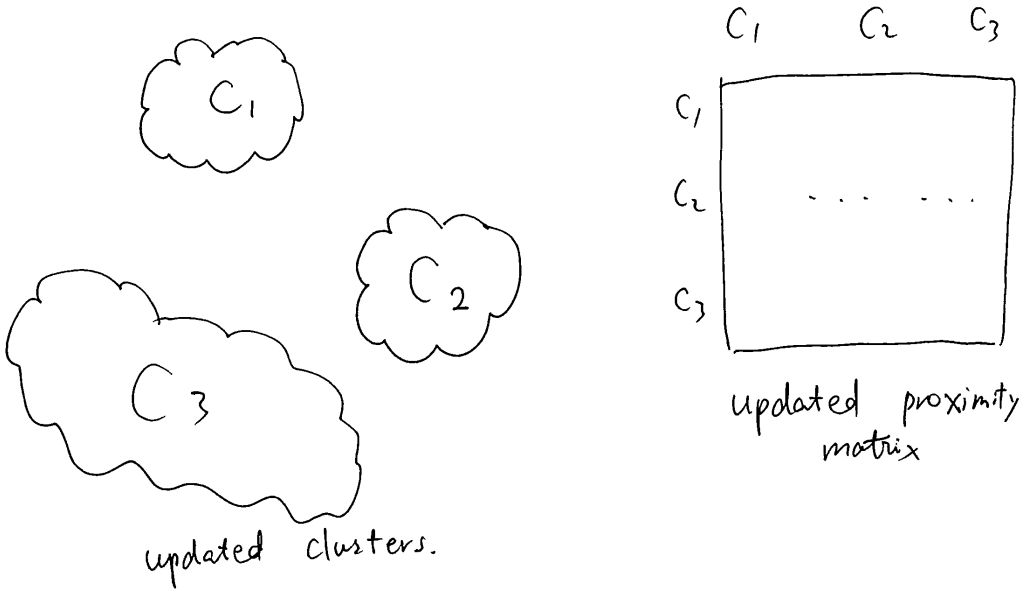
clusters.

	C_1	C_2	C_3	C_4
C_1	*	*	*	*
C_2				
C_3			. . .	...
C_4				

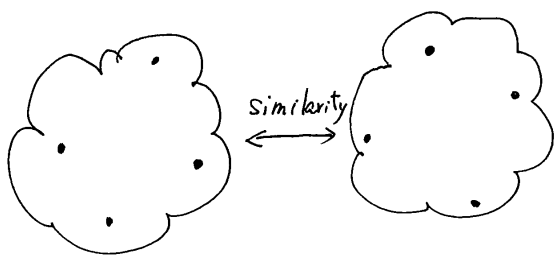
proximity matrix

We now want to merge the two closest clusters (C2 and C4, for example). We merge them and then update the proximity matrix.

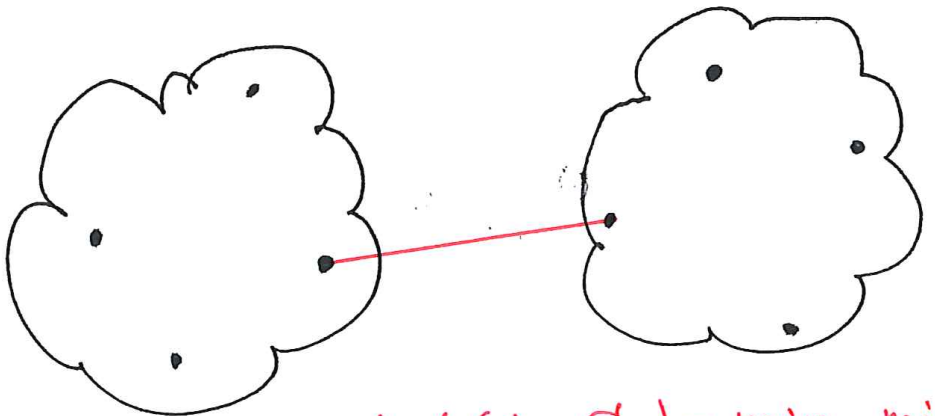




Then the question is, how to update the proximity matrix? Indeed, we update the matrix by computing the "distance" of the clusters. (Strictly speaking, what we compute may not be a mathematical distance. We call it linkage or similarity sometimes.)



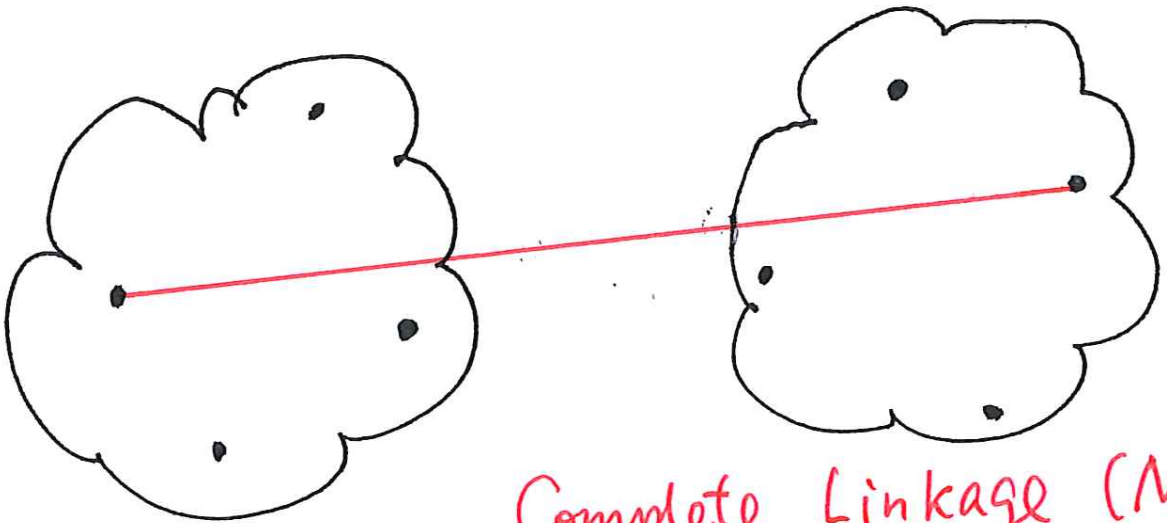
Inter-cluster similarity: single linkage



Single linkage
(MIN)

$$dist(C_1, C_2) = \min_{x \in C_1} \min_{y \in C_2} dist(x, y)$$

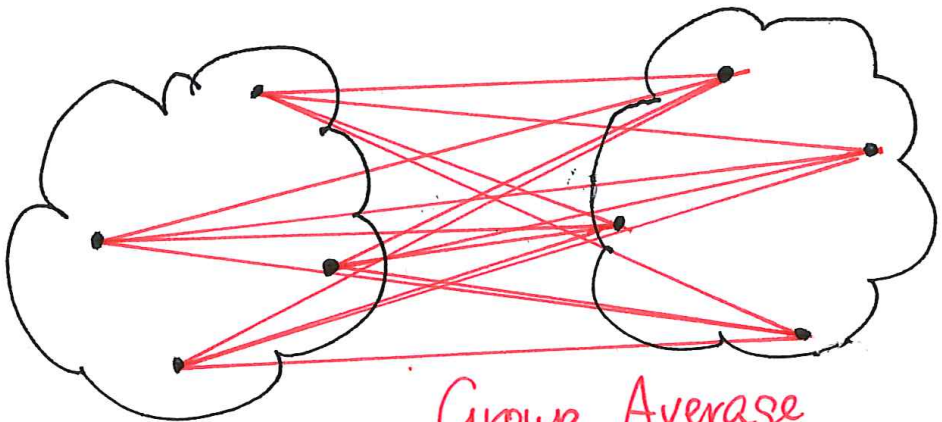
Inter-cluster similarity: complete linkage



Complete Linkage (MAX)

$$dist(C_1, C_2) = \max_{x \in C_1} \max_{y \in C_2} dist(x, y)$$

Inter-cluster similarity: group average linkage



Group Average

$$dist(C_1, C_2) = \frac{1}{\#C_1 \times \#C_2} \sum_{x \in C_1} \sum_{y \in C_2} dist(x, y)$$

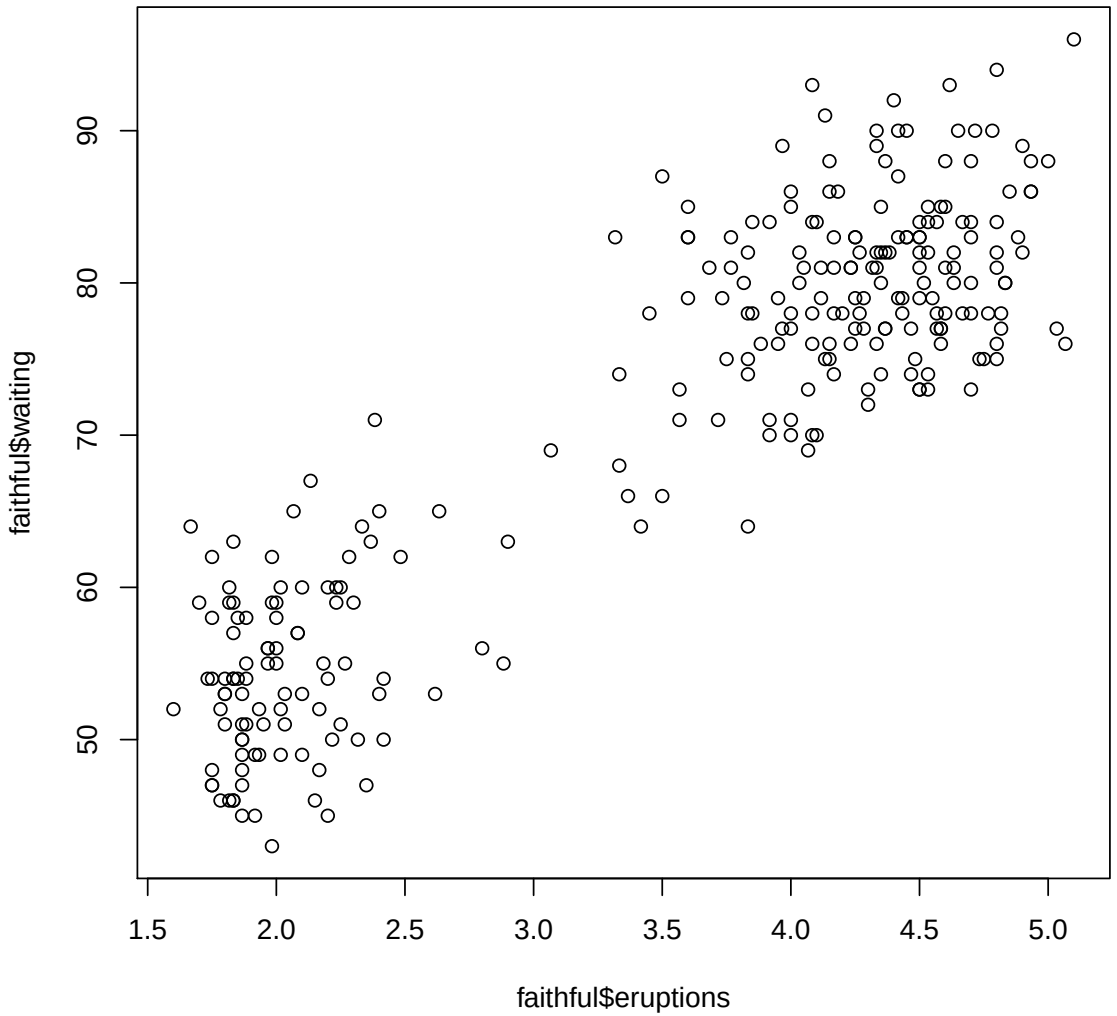
Example: The Old Faithful Geyser Dataset

```
Description: (From R manual):

     Waiting time between eruptions and the duration
     of the eruption for the Old Faithful geyser in
     Yellowstone National Park, Wyoming,
     USA.

     A data frame with 272 observations on
     2 variables.

eruptions  numeric  Eruption time in mins
waiting    numeric  Waiting time to next eruption

References:

     Hardle, W. (1991) Smoothing Techniques with
     Implementation in S.  New York: Springer.

     Azzalini, A. and Bowman, A. W. (1990).
     A look at some data on the Old Faithful
     geyser. Applied Statistics 39, 357-365.
```

Picture in the Public Domain

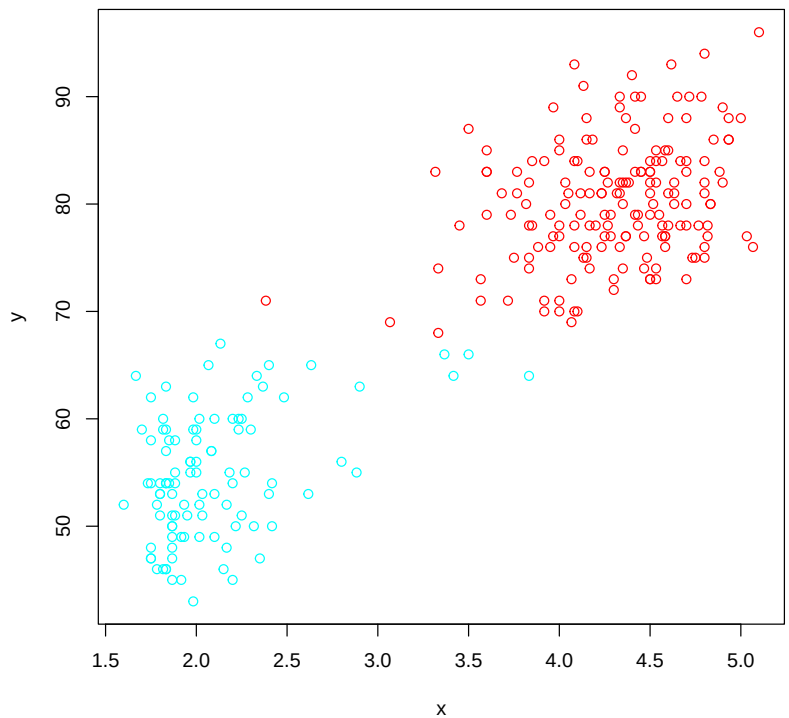
The Old Faithful Geyser Dataset is very famous and is included in R

```
> head(faithful)
  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
4     2.283      62
5     4.533      85
6     2.883      55
> plot(faithful$eruptions,
faithful$waiting)
```



The code below applies k-means algorithm on the Old Faithful Geyser Dataset. The generated plot is also provided below.

```r
nCluster <- 2
a <- kmeans(faithful, centers = nCluster, iter.max = 100)
plot(range(faithful[, 1]), range(faithful[, 2]),
  type = "n", xlab = "x", ylab = "y")
inds <- a$cluster
cols <- rainbow(nCluster)
for(i in 1:nCluster){
  points(faithful[inds == i, ], col = cols[i])
}
```



Building a cluster tree in R is also simple. The following figure is essentially generated by only one line of command:

```
plot(hclust(dist(iris[, 1:4])))
```