Introduction to Data Science-Topic 7

- Instructor: Professor Henry Horng-Shing Lu,

  Institute of Statistics, National Chiao Tung University, Taiwan

  Email: hslu@stat.nctu.edu.tw

- WWW: http://www.stat.nctu.edu.tw/misg/hslu/course/DataScience.htm
- Reference:

  M. A. Pathak, Beginning Data Science with R, 2014, Springer-Verlag.
- Evaluation: Homework: 50%, Term Project: 50%
- Office hours: By appointment

Course Outline

- **Introduction of Data Science**
- **Introduction of R**
- **More on R**
- **Process Real Data by R**
- **Data Visualization**
- **Exploratory Data Analysis**
- **Regression**
- **Classification**
- **Text Mining**
- **Clustering**

Classification

References:

Ch. 7, M. A. Pathak, Beginning Data Science with R, 2014, Springer-Verlag.

7.1 Introduction

Regression - predicting a numeric value
Classification - classify data points into multiple categories or classes

Application: spam filtering, computational advertising, speech and handwriting recognition, and biometric authentication…and so on.

In this chapter, we'll introduce:
Parametric model: Naïve Bayes, Logistic Regression, SVM
Nonparametric: Nearest Neighbors, Decision Tree-based models

In this topic, we'll use the Titanic Dataset on Kaggle:
https://www.kaggle.com/c/titanic/data

```
> titanic <- read.csv("train.csv")
```

In this topic, we’ll use the Titanic Dataset on Kaggle:
https://www.kaggle.com/c/titanic/data

```
> titanic <- read.csv("train.csv")
```

Preprocessing:

Select the columns we want to build a model.

```
> titanic <- titanic[, c(2,3,5,6,7,8,10,12)]
```

Remove NA.

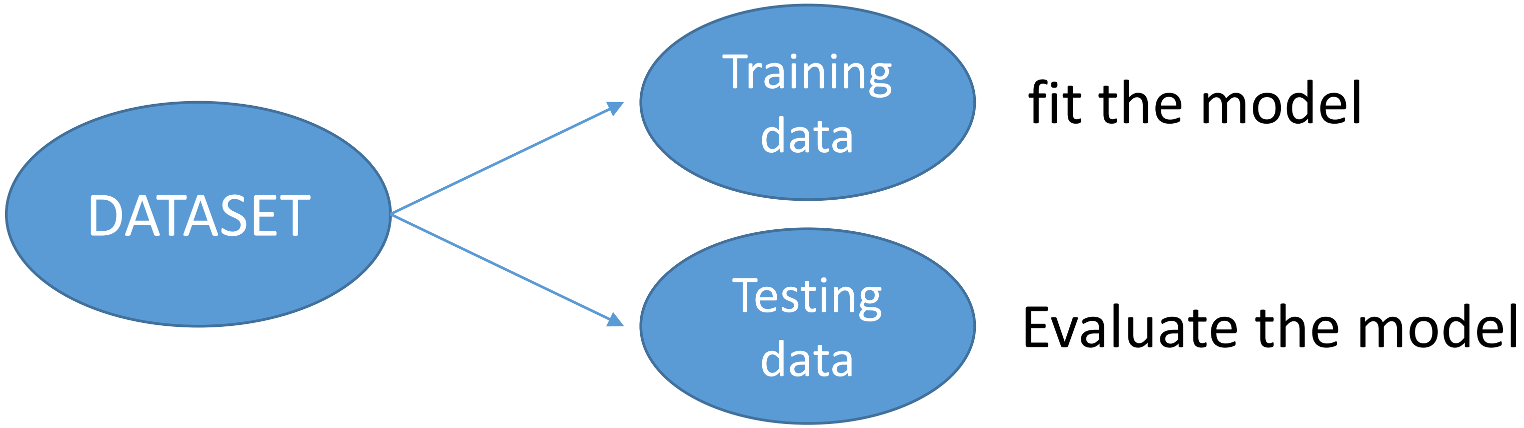
```
> titanic <- na.omit(titanic)
```

Turn factor variables into correct type.

```
> titanic$Survived <- as.factor(titanic$Survived)
> titanic$Pclass <- as.factor(titanic$Pclass)
```

Variable name	Variable type	Description
PassengerId	Unique ID	Passenger’s id
Survived	Categorical	Survival, 0 = No, 1 = Yes
Pclass	Categorical	Ticket class, 1 = 1st, 2 = 2nd, 3 = 3rd
Name	Character	Passenger’s name
Sex	Categorical	Passenger’s sex
Age	Numeric	Age in years
SibSp	Numeric	# of siblings / spouses aboard the Titanic
Parch	Numeric	# of parents / children aboard the Titanic
Ticket	ID	Ticket number
Fare	numeric	Passenger fare
Cabin	ID	Cabin number
Embarked	Categorical	Port of Embarkation, C = Cherbourg, Q = Queenstown, S = Southampton

7.1.1 Training and Test Datasets




```
> library(caret)
> set.seed(20180430)
> train.ind <- createDataPartition(titanic$Survived, p = 2/3, list = F)
> train <- titanic[train.ind, ]
> test <- titanic[-train.ind, ]
> dim(train) ; dim(test)
[1] 477   8
[1] 237   8
```

7.2 Parametric Classification Models

7.2.1 Naive Bayes

$P(\theta) = prior\ probability, P(x|\theta) = likelihood\ of\ the\ data\ x$

$$posterior\ probability\ P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)}$$

In a naive Bayes classifier, we compute the conditional probability of a data point having a class label $y = \{-1, 1\}$ and data point x,

$$P(y = 1|x) = \frac{P(x|y = 1)P(y = 1)}{P(x)}$$

$$P(y = -1|x) = \frac{P(x|y = -1)P(y = -1)}{P(x)}$$

We can assign the label $y = 1$ or $y = -1$ to the data point x depending on which posterior probability is greater.

$P(y=1)$ *and* $P(y=-1)$: *marginal probabilities or simply marginals*

$P(x|y=1)$ is hard to compute while we have many features

To solve this problem, the NB model makes an assumption that the features of a data point are conditional independence:

$$P(x|y=1) = P\left(x_1, x_2, \cdots x_p \middle| y=1\right) = P(x_1|y=1)P(x_2|y=1)\cdots P(x_p|y=1)$$

7.2.1.1 Training an NB Classifier Using the e1071 Package

```
> library(e1071)
> model.nb <- naiveBayes(Survived ~ ., train)
```

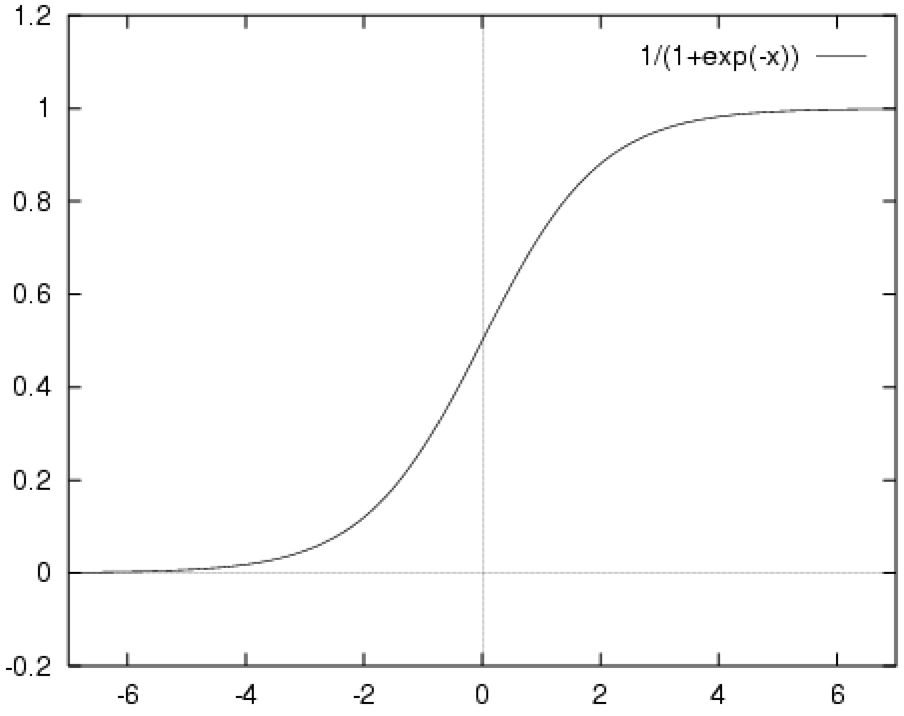
```
> model.nb

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        0         1 
0.5932914 0.4067086 

Conditional probabilities:
   Pclass
Y           1         2         3
  0 0.1554770 0.2190813 0.6254417
  1 0.4175258 0.3041237 0.2783505

   Sex
Y       female      male
  0 0.1590106 0.8409894
  1 0.6752577 0.3247423
       …
```

Use `predict()` to obtain predict result

```
> predict(model.nb, test)
  [1] 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0
 [33] 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0
       …
Levels: 0 1
> table(predict(model.nb,test) == test$Survived)/length(test$Survived)

    FALSE      TRUE
0.2025316 0.7974684
```

The accuracy of our naive Bayes classifier model is 79.7 %.

7.2.2 Logistic Regression



Sigmoid function: $sig(t) = \frac{1}{1+e^{-t}}$

- Range: 0-1
- monotonically increasing

If the class label y takes two values 1and −1, and has only one predictor variable, we have

$$P(y = 1|x) = sig(w_0 + w_1 x) = \frac{1}{1 + e^{-(w_0+w_1 x)}}$$

$$P(y = -1|x) = \frac{e^{-(w_0+w_1 x)}}{1 + e^{-(w_0+w_1 x)}} = \frac{1}{\frac{1}{e^{-(w_0+w_1 x)}} + 1} = \frac{1}{1 + e^{(w_0+w_1 x)}} = sig(-(w_0 + w_1 x))$$

$$\Rightarrow P(y|x) = sig(y(w_0 + w_1 x))$$

In logistic regression model, we select the coefficient vector w that maximizes the log-likelihood, given this log-likelihood function, we obtain w using optimization techniques such as gradient descent.

7.2.2.1 Using the glm() Function

Fit the logistic model with one variable `Pclass`

```
> model.lr.pclass <- glm(Survived ~ Pclass, data = train,
+                        family = "binomial")
> model.lr.pclass

Call:  glm(formula = Survived ~ Pclass, family = "binomial", data = train)

Coefficients:
(Intercept)      Pclass2      Pclass3
     0.6103      -0.6599      -1.7974

Degrees of Freedom: 476 Total (i.e. Null);  474 Residual
Null Deviance:          644.6
Residual Deviance: 581.1   AIC: 587.1
```

```
> summary(model.lr.pclass)

Call:
glm(formula = Survived ~ Pclass, family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.4451  -0.7298  -0.7298   0.9315   1.7049

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.6103     0.1873   3.259  0.00112 **
Pclass2      -0.6599     0.2611  -2.528  0.01148 *
Pclass3      -1.7974     0.2434  -7.385 1.53e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 644.56  on 476  degrees of freedom
Residual deviance: 581.07  on 474  degrees of freedom
AIC: 587.07

Number of Fisher Scoring iterations: 4
```

`fit()` output the numeric score : $w_0 + w_1 x$

```
> predict(model.lr.pclass, test)
          1           3           7           9          13 
-1.18716569 -1.18716569  0.61025952 -1.18716569 -1.18716569 
         14          16          19          24          31 
-1.18716569 -0.04959694 -1.18716569  0.61025952  0.61025952 
       …
```

We can pass the above value to sigmoid function are simply use parameter `type="response"` to obtain the probabilities.

```
> predict(model.lr.pclass, test, type = "response")
        1         3         7         9        13        14        16 
0.2337662 0.2337662 0.6480000 0.2337662 0.2337662 0.2337662 0.4876033 
       19        24        31        39        41        44        50 
0.2337662 0.6480000 0.6480000 0.2337662 0.2337662 0.4876033 0.2337662 
       …
```

We need to set a threshold to convert these probabilities into class labels. For example, we set 0.5 as threshold.

```
> p <- predict(model.lr.pclass, test, type = "response")
> labels <- ifelse(p > 0.5, "1", "0")
> labels
  1   3   7   9  13  14  16  19  24  31  39  41  44  50  54  57  61 
"0" "0" "1" "0" "0" "0" "0" "0" "1" "1" "0" "0" "0" "0" "0" "0" "0" 
 69  72  75  81  82  85  86  89  92  95  98  99 101 106 109 116 117 
"0" "0" "0" "0" "0" "0" "0" "1" "0" "0" "1" "0" "0" "0" "0" "0" "0" 
  …
> table(labels == test$Survived)/length(test$Survived)

    FALSE      TRUE 
0.3164557 0.6835443 
```

The accuracy of our logistic regression model using `Pclass` alone is 68.4 %.

We train the logistic regression model to predict the `Survived` with all features.

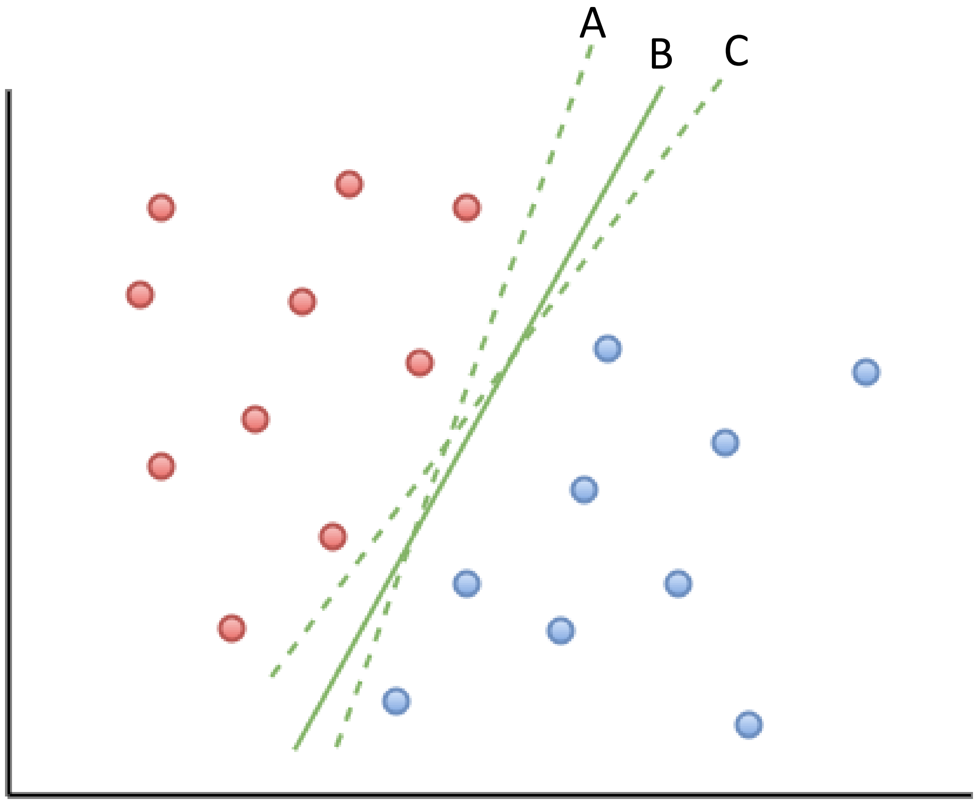
```
> model.lr <- glm(Survived ~ ., data = train, family = "binomial")
> model.lr

Call:  glm(formula = Survived ~ ., family = "binomial", data = train)

Coefficients:
(Intercept)      Pclass2      Pclass3      Sexmale          Age
   16.07743     -1.17150     -2.62148     -2.64887     -0.04341
      SibSp        Parch         Fare    EmbarkedC    EmbarkedQ
   -0.53061     -0.02572      0.00173    -11.71033    -12.12269
  EmbarkedS
  -11.93030

Degrees of Freedom: 476 Total (i.e. Null);  466 Residual
Null Deviance:          644.6
Residual Deviance: 420.7   AIC: 442.7
```

We need to set a threshold to convert these probabilities into class labels. For example, we set 0.5 as threshold.

```
> p <- predict(model.lr, test, type = "response")
> labels <- ifelse(p > 0.5, "1", "0")
> table(labels == test$Survived)/length(test$Survived)

    FALSE      TRUE 
0.1687764 0.8312236 
```

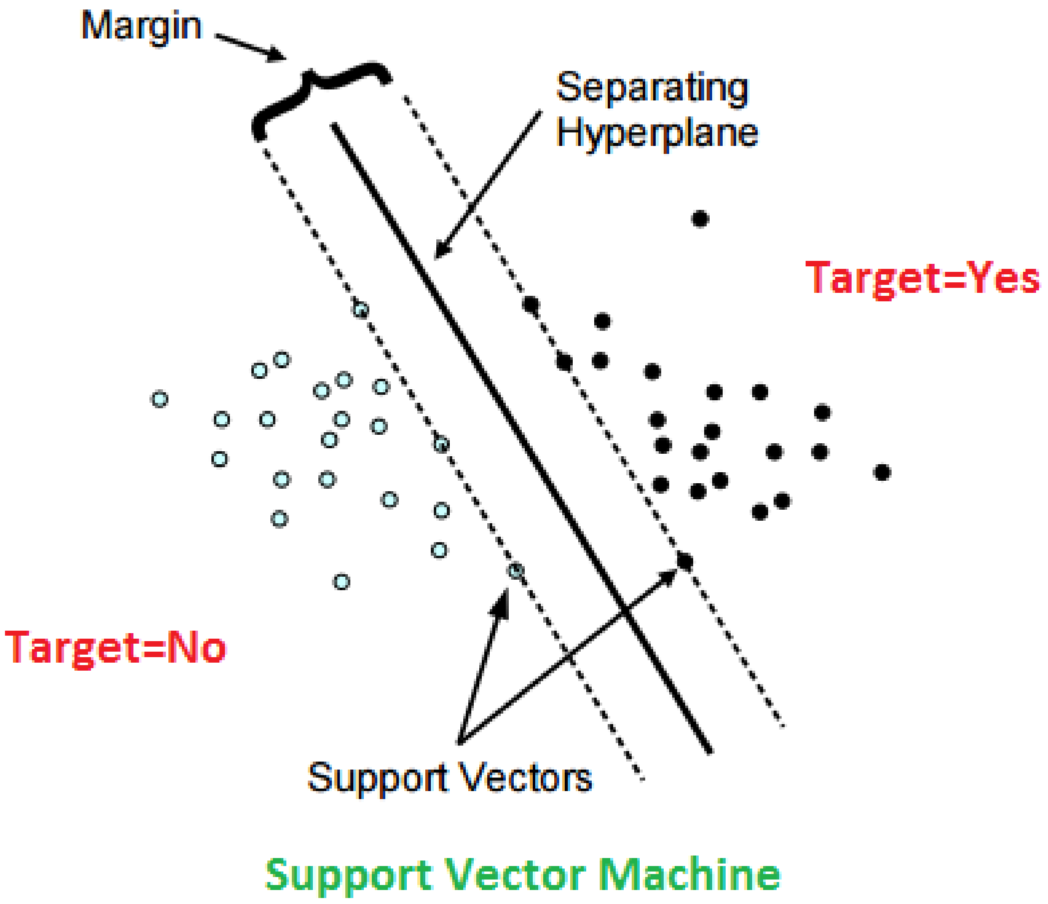
The accuracy of our logistic regression model is 83.1 %, is better than we use `Pclass` alone.

7.2.2 Support Vector Machines



All A, B and C can separate data points perfectly,
Which one is better?

https://eight2late.wordpress.com/2017/02/07/a-gentle-introduction-to-support-vector-machines-using-r/



The goal of SVM:
Maximized the margin

→ solving a quadratic optimization problem.

Use `svm()` in `e1071` package:
an interface to LIBSVM
https://www.csie.ntu.edu.tw/~cjlin/libsvm/

https://ryannng.github.io/2016/12/13/MLND%E5%9B%9E%E9%A1%BE-%E7%9B%91%E7%9D%A3%E5%AD%A6%E4%B9%A0(1%EF%BC%89%E6%94%AF%E6%8C%81%E5%90%91%E9%87%8F%E6%9C%BA/

We can use `svm( )` function to train a svm model

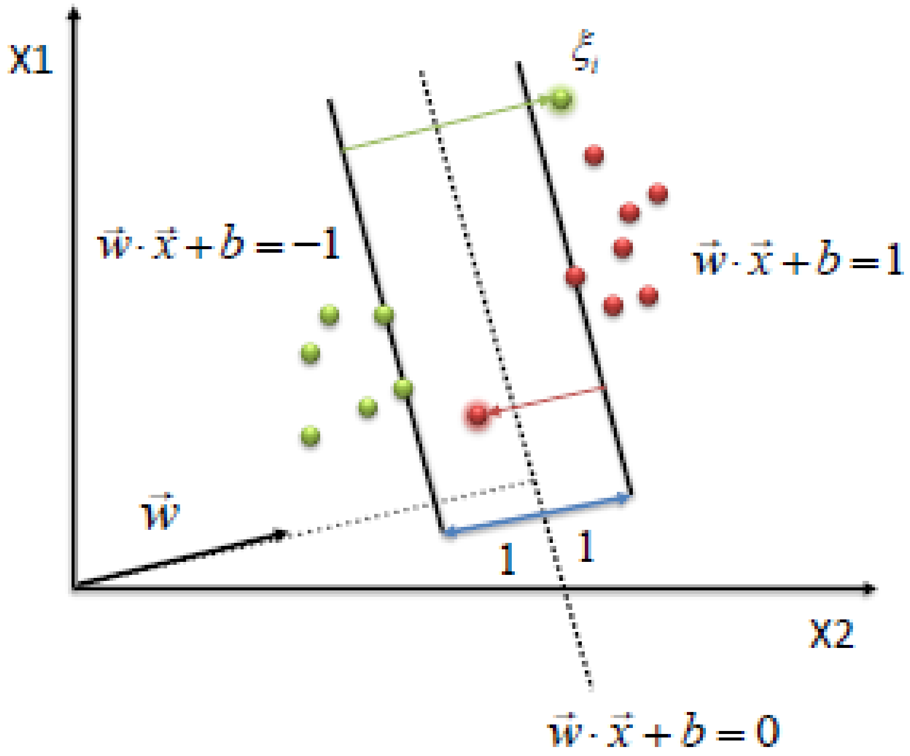
```
> model.svm <- svm(Survived ~ ., data = train, kernel = "linear")
> model.svm

Call:
svm(formula = Survived ~ ., data = train, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.09090909

Number of Support Vectors:  236
```

`predict()` function to predict labels

```
> predict(model.svm, test)
  1   3   7   9  13  14  16  19  24  31  39  41  44  50  54  57  61
  0   1   0   1   0   0   1   1   0   0   1   1   1   1   1   1   0
 69  72  75  81  82  85  86  89  92  95  98  99 101 106 109 116 117
  1   1   0   0   0   1   1   1   0   0   0   1   1   0   0   0   0
       …
Levels: 0 1
> table(predict(model.svm, test) == test$Survived)/length(test$Survived)

    FALSE      TRUE
0.2067511 0.7932489
```

Our SVM classifier has 79.3% accuracy.

If data points are not linear separable



slack variable:

ξ_i

Allow some instances to fall off the margin, but penalize them

We can control the slack using the `cost` parameter of `svm()`.

By default cost is set to 1,

```
> model.svm.cost <- svm(Survived ~ ., data = train, kernel = "linear",
+                       cost = 0.1)
```

http://www.saedsayad.com/support_vector_machine.htm

7.2.3.1 Kernel Trick

An alternative strategy of dealing with nonlinearly separable data is to use the kernel trick. The idea behind the kernel trick is to project the data points of the training data into a higher dimensional space, in which the dataset is linearly separable.

Some common kernels: polynomial, sigmoid and radial kernel.

We can specify the type of the kernel using the `kernel` argument

```
> model.svm.radial <- svm(Survived ~ ., data = train, kernel = "radial")
> model.svm.radial

Call:
svm(formula = Survived ~ ., data = train, kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.09090909

Number of Support Vectors:  267
```

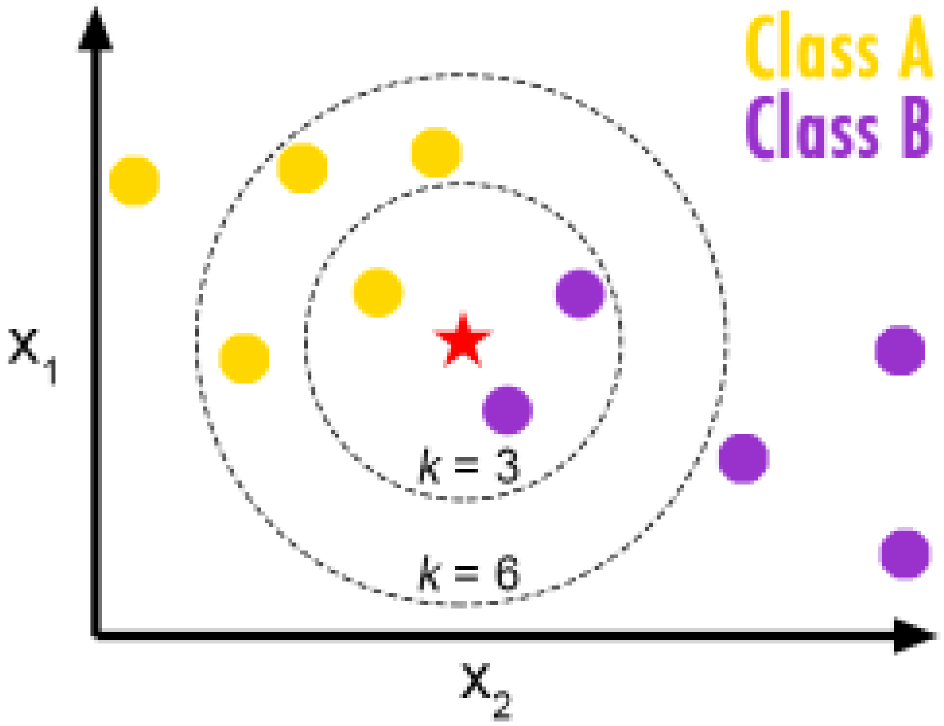
```
> table(predict(model.svm.radial, test) == test$Survived)/length(test$Survived)

    FALSE      TRUE 
0.1940928 0.8059072 
```

The accuracy is higher than we use linear kernel.

We can also tune these parameter using grid search to reach higher accuracy.

7.3 Nonparametric Classification Models

7.3.1 Nearest Neighbors



- without assuming any specific
- work well for nonlinear datasets

k: The only parameter we need to decide
We choose k based on the value that gives the highest accuracy on test dataset or using cross validation.

http://adataanalyst.com/machine-learning/knn/

Distance metrics:

The most popular one is the Euclidean distance, which is given by the linear distance between two data points:

$$Euclidean(x_1, x_2) = \sqrt{\sum_i (x_{1i} - x_{2i})^2}$$

City-block or Manhattan distance:

$$Manhattan(x_1, x_2) = \sum_i |x_{1i} - x_{2i}|$$

Minkowski distance:

$$Minkowski(x_1, x_2) = \sqrt[p]{\sum_i (x_{1i} - x_{2i})^p}$$

p = 2 gives us Euclidean distance, while setting p = 1 gives us Manhattan distance

We can use a kernel function to assign weight based on the distance metric.
We first use unweighted NN algorithm; this is equivalent to setting the `kernel` parameter to "`rectangular`". The default setting if `kknn()` function are use Minkowski distance with p=2 (equivalent to Euclidean distance), and k = 7.

```
> library(kknn)
> model <- kknn(Survived ~ ., train, test, kernel = "rectangular")
> model$fitted.values
  [1] 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0
 [33] 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 1 0
      …
> table(model$fitted.values == test$Survived)/length(test$Survived)

   FALSE     TRUE
0.185654 0.814346
```

The fitted value are return in `fitted.value`.
The accuracy of unweighted NN with k=7 is 81.4 %.

We can try different k:

```
> model.5 <- kknn(Survived ~ ., train, test, kernel = "rectangular",
+                k = 5)
> table(model.5$fitted.values == test$Survived)/length(test$Survived)

    FALSE      TRUE 
0.1729958 0.8270042 
```

The accuracy of unweighted NN with k=5 is 82.7 %, which is higher than the default k = 7.

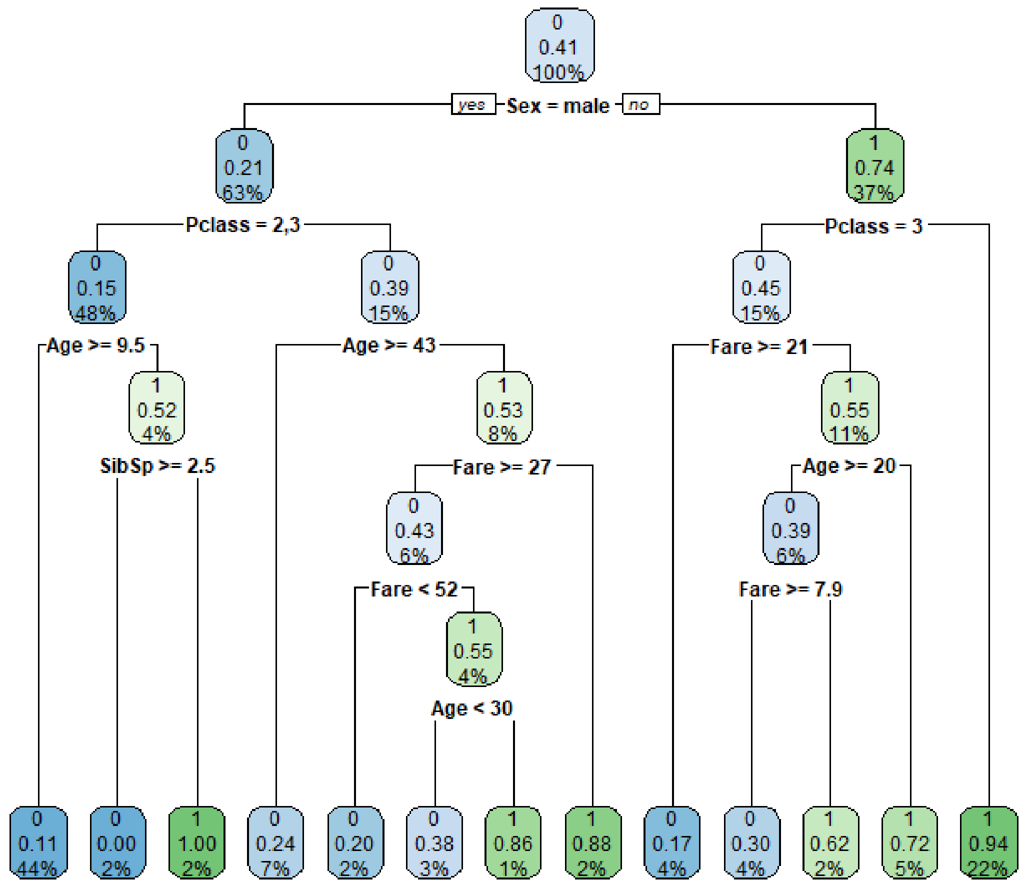
Similarly, we can also experiment with other kernels:

```
> model.gaussian <- kknn(Survived ~ ., train, test, kernel = "gaussian")
> table(model.gaussian$fitted.values == test$Survived)/length(test$Survived)

    FALSE      TRUE 
0.1940928 0.8059072 
```

7.3.2 Decision Trees

We also use the `rpart` package to fit decision trees.

```
> library(rpart)
> model.dt <- rpart(Survived ~ .,train)
> library(rpart.plot)
> rpart.plot(model.dt)
```



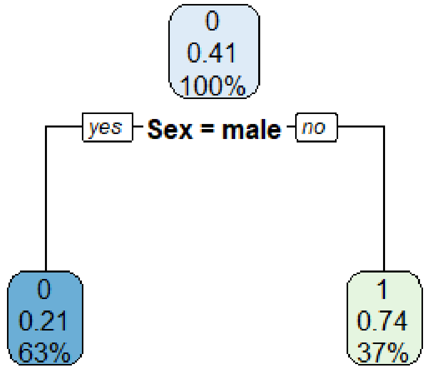
By default, the `predict()` function returns the class probabilities. We obtain the class labels by calling `predict()` with the argument `type="class"`.

```
> predict(model.dt, test, type = "class")
  1   3   7   9  13  14  16  19  24  31  39  41  44  50  54
  0   0   0   0   0   0   1   0   0   0   1   0   1   1   1
 57  61  69  72  75  81  82  85  86  89  92  95  98  99 101
  1   0   1   0   0   0   0   1   0   1   0   0   0   1   0
       …
Levels: 0 1
> table(predict(model.dt, test, type = "class") ==
test$Survived)/length(test$Survived)

    FALSE      TRUE
0.1940928 0.8059072
```

The accuracy of our decision tree classifier is 80.6 %.

we set the cp parameter to prune some of the branches off by setting it to 0.04. (default = 0.01)

```
> rpart.plot(model.dt)
> rpart.plot(model.dt.prune)
```



We can see the tree is smaller than the previous one.

```
> table(predict(model.dt.prune, test, type = "class") == test$Survived) /
+   length(test$Survived)

    FALSE      TRUE
0.2067511 0.7932489
```

The accuracy is lower than the tree without pruning, maybe we can consider the larger value of cp, means more bigger(complex) tree.

Homework

- Basic
  - Find a dataset you want to analysis.
  - Do proper data preprocessing before building classification model.
  - Practice the models introduce in this chapter.
- Advanced
  - Explained the what preprocessing you done and why to do that.
  - Compare the different model by different evaluation metric (accuracy, recall, precision, …), and choose one evaluation metric as the criteria of model selection, explain why you choose this evaluation metric.

    more information about evaluation metric:

    (https://en.wikipedia.org/wiki/Precision_and_recall)

Homework 7 (submitted to e3new.nctu.edu.tw before Nov 5, 2019)

- Use R and/or the other software practice classification model
- Explain the results you obtain
- Discuss possible problems you plan to investigate for future studies
- Possible source of open data:

  UCI Machine Learning Repository

  (http://archive.ics.uci.edu/ml/datasets.php)

References

1. https://eight2late.wordpress.com/2017/02/07/a-gentle-introduction-to-support-vector-machines-using-r/
2. http://www.saedsayad.com/support_vector_machine.htm
3. http://adataanalyst.com/machine-learning/knn/
4. https://www.kaggle.com/c/titanic/data
5. Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27, 1–27:27. http://www.csie.ntu.edu.tw/cjlin/libsvm. Accessed 1 Aug 2014.
6. Cortes, C., &Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297.
7. Geisser, S. (1993). Predictive inference. UK: Chapman and Hall.
8. Graham, P. (2002). A plan for spam. http://www.paulgraham.com/spam.html. Accessed 1 Aug 2014.
9. Karatzoglou, A., & Meyer, D. (2006). Support vector machines in R. Journal of Statistical Software, 15, 1–28.