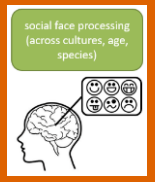
분석 인프라 활용 AI교육(R)
Day 3

Cho Heeseung

hscho9384@korea.ac.kr

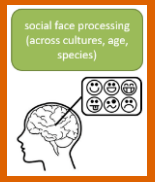
목차



CognitiveSystems

1. K-최근접이웃(KNN)
2. 의사결정나무(Decision Tree)
3. 서포트 벡터 머신(Support Vector Machine)

들어가기 전에



CognitiveSystems

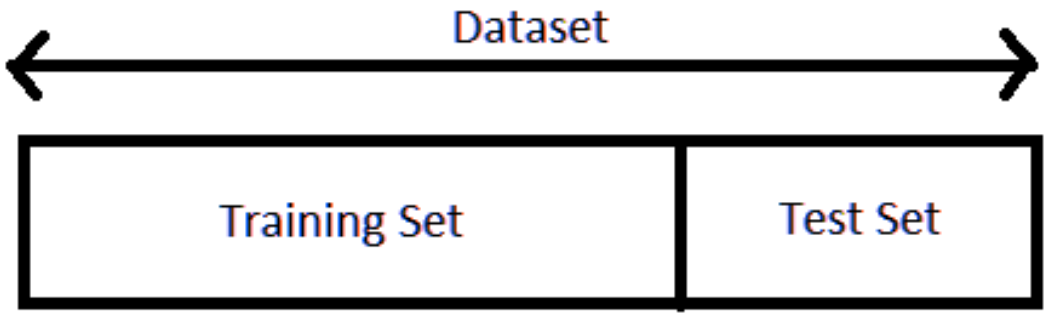
- **훈련 데이터(Train data)**

모델을 학습시키는데 들어가는 데이터.

- **검증 데이터(Test data)**

학습된 모델의 성능을 평가하기 위한 데이터. 모델의 일반화를 판단하는데 사용.

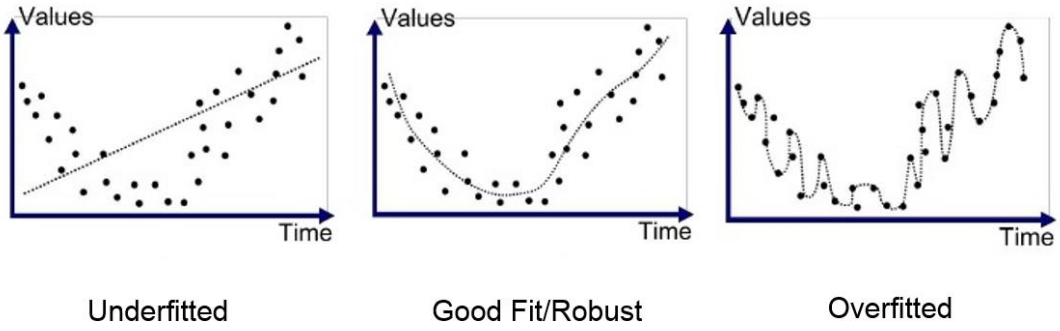
** 검증 데이터는 절대로 훈련 데이터를 포함해서는 안된다.

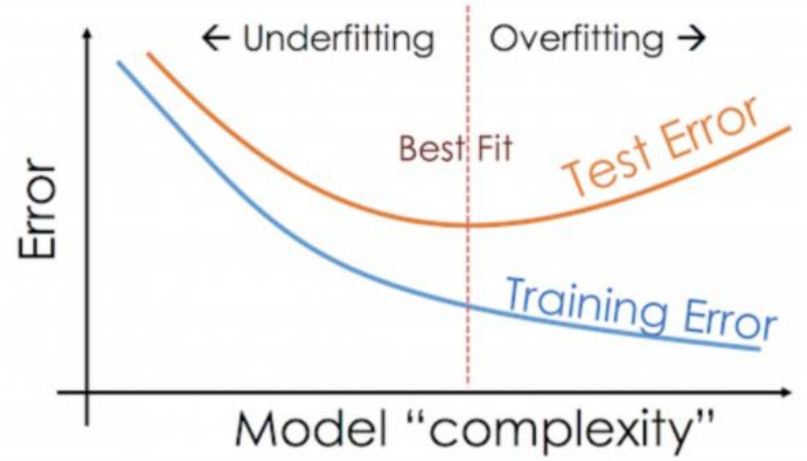


들어가기 전에

CognitiveSystems

- **과적합(Overfitting)**: 모델이 너무 복잡하여 훈련 데이터에는 적합하지만 일반화하기 힘든 상황
- **부적합(Underfitting)**: 모델이 너무 단순하여 훈련 데이터에서의 성능이 좋지 않아 일반화하기 힘든 상황





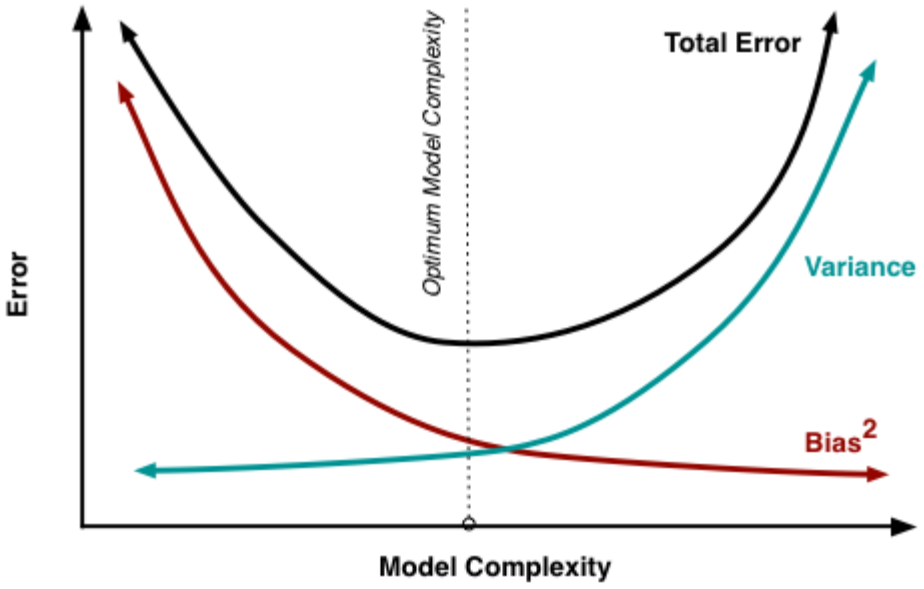
들어가기 전에

CognitiveSystems

Bias-Variance Trade-off

적합에 관련하여 모델의 성능이 정확해질 수록 편향(Bias)를 가지게 되고 모델의 변동성(Variance)이 높을수록 일반화 정도가 높아진다.

그러나 편향이 높아지면 반대로 변동성이 작아지고, 편향이 줄어들면 변동성이 높아지는 상충관계(Trade-off)에 있다.



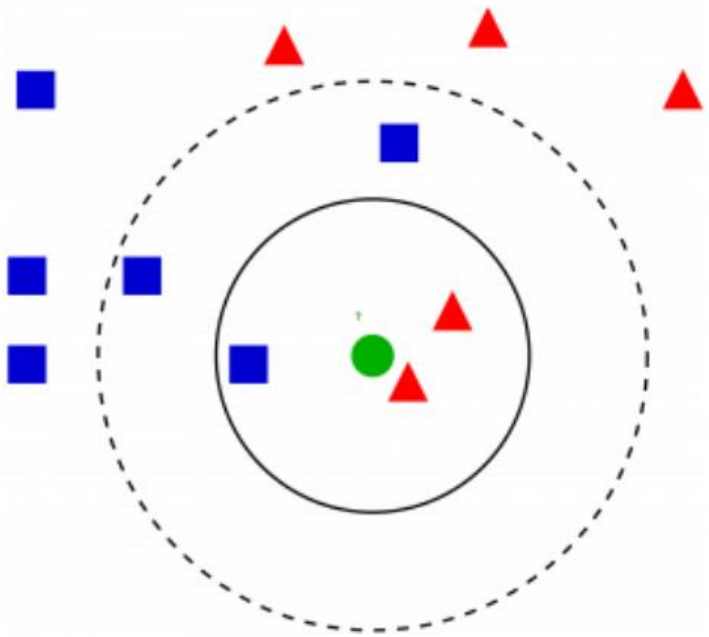
1. K-최근접이웃(KNN)

KNN

K-Nearest Neighbors(KNN)

주어진 독립변수 값에 대해 K(주로 홀수)개의 가장 가까운 관측치들을 이용하여 원하는 종속변수의 값을 예측하는 방법.

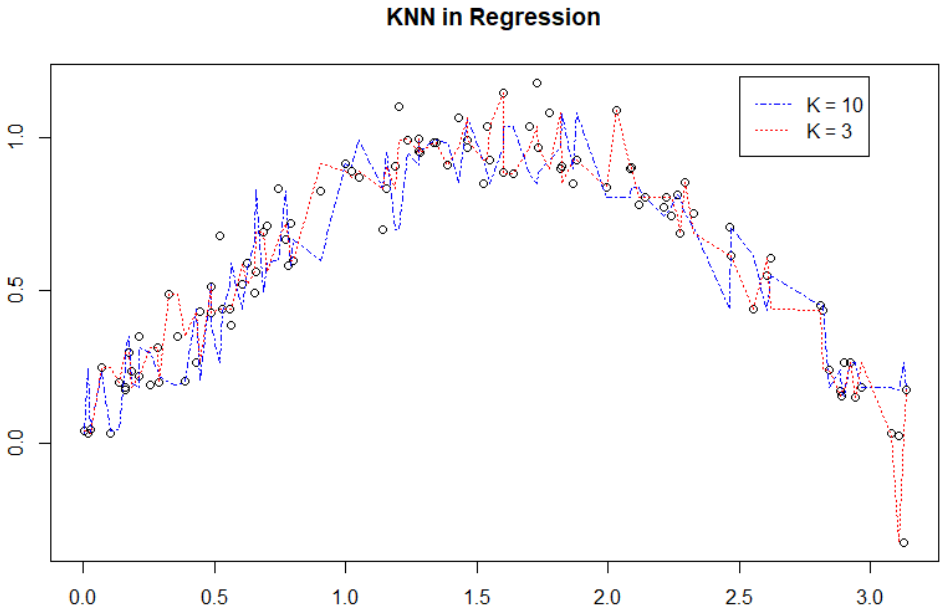
라이브러리: **class**

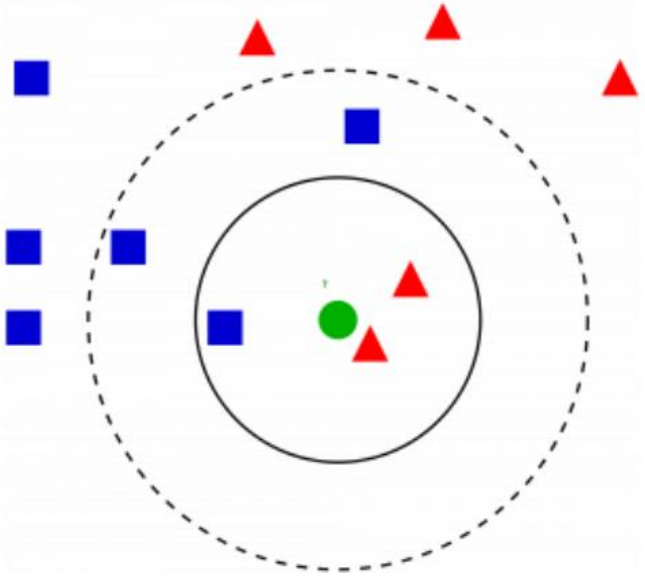


KNN

회귀분석: 해당 점에서 가장 가까운 K개의 관측치들에 대한 종속변수 값들의 평균을 사용하여 해당 점의 종속변수 값을 추정

분류분석: 해당 점에서 가장 가까운 K개의 관측치들에 대한 종속변수 의 Class를 비교, 가장 많은 Class를 예측값으로 추정





K의 개수

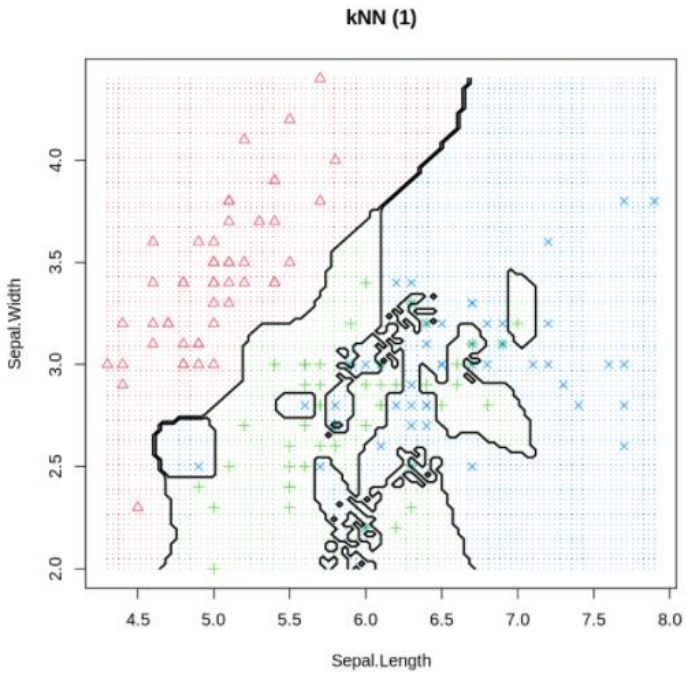
K가 낮을수록: 변동이 커짐, 항목간 경계가 분명해짐

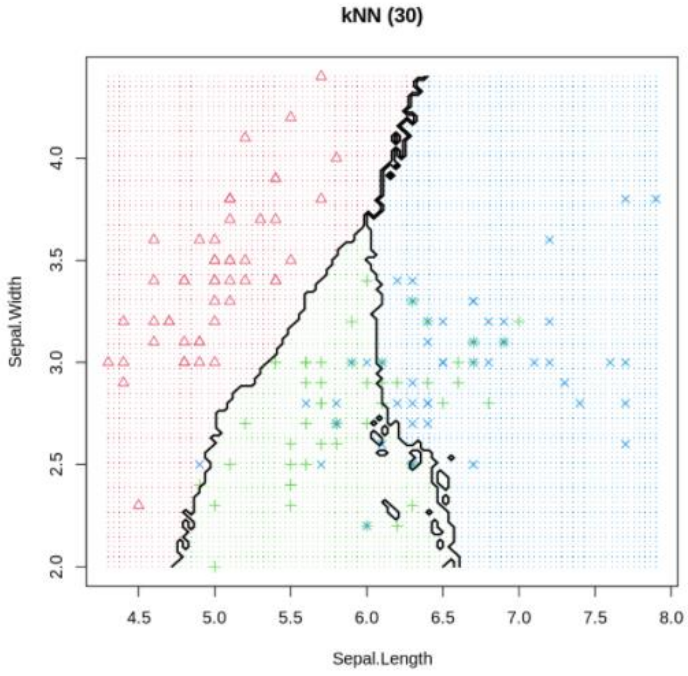
- 과적합(Overfitting) 발생, Low bias and high variance

K가 높을수록: 변동이 작아짐, 항목간 경계가 불분명해짐

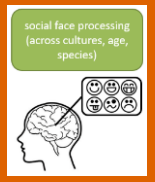
- 부적합(Underfitting) 발생, Low variance and high bias

* 대체적으로, K의 개수는 데이터 수의 제곱근을 이용





KNN의 특징



CognitiveSystems

장점:

1) 단순하지만 성능이 좋다.

2) 비모수적 추정

* 선형회귀, 로지스틱 회귀와 달리 모델에 대한 어떤 가정 없이도 추정이 가능

단점:

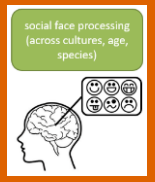
1) 데이터에 매우 의존적

* 분류모델에서 항목 분포가 편향될 때, 예측 결과는 빈번한 항목이 예측을 지배하는 경향

* 같은 데이터에서의 샘플끼리도 결과 차이가 많이 날 수 있다.

2) 일반화에 대한 어려움

KNN – Code(1)



CognitiveSystems

iris 데이터를 전처리하여, 학습할 데이터와 테스트 데이터로 나눈 후 K=3일 때의 KNN 알고리즘을 계산한 결과이다.

```
library(class)
data(iris)
set.seed(43)

## Preprocessing
summary(iris$Species)
idx = sample(1:nrow(iris), size = 100)
train = iris[idx,]
test = iris[-idx,]
X_train = train[,c(1,2,3,4)]
y_train = train[,5]
X_test = test[,c(1,2,3,4)]
y_test = test[,5]

## Training
fit = knn(train = X_train, test = X_test, cl = y_train,  k = 3)
table(y_test, fit)

## Try another K values!
fit = knn(train = X_train, test = X_test, cl = y_train,  k = ?)
table(y_test, fit)
```

Code(1) 성능 평가

CognitiveSystems

1번 예시의 분류 정확도를 평가해보자.

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

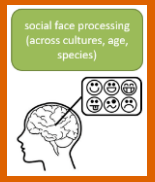
$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$specificity = \frac{TN}{TN + FP}$$

Q1. K = 3일 때 모델의 정확도(Accuracy)는?

Q2. K = 10일 때 모델의 정확도(Accuracy)는?

KNN – Code(2)



CognitiveSystems

ISLR 패키지 내의 데이터인 Caravan를 전처리하여, 학습할 데이터와 테스트 데이터로 나눈 후 K=1일 때의 KNN 알고리즘을 계산한 결과이다.

```
### Caravan example
library(ISLR)
data(Caravan)
summary(Caravan$Purchase)

##Preprocessing
# Standardize
standardized.X = scale(Caravan[, -86])
idx = sample(1:nrow(Caravan), size = 1000)

# Split train, test
train.X = standardized.X[-idx,]
train.Y = Caravan$Purchase[-idx]
test.X = standardized.X[idx,]
test.Y = Caravan$Purchase[idx]

## Training
knn.pred = knn(train.X, test.X, train.Y, k = 1)
table(knn.pred ,test.Y)

## Try another K value!
knn.pred = knn(train.X, test.X, train.Y, k = ??)
table(knn.pred ,test.Y)
```

Code(2) 성능 평가

2번 예시의 분류 정확도를 평가해보자.

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$specificity = \frac{TN}{TN + FP}$$

Q1. K = 1일 때 모델의 정확도(Accuracy)는?

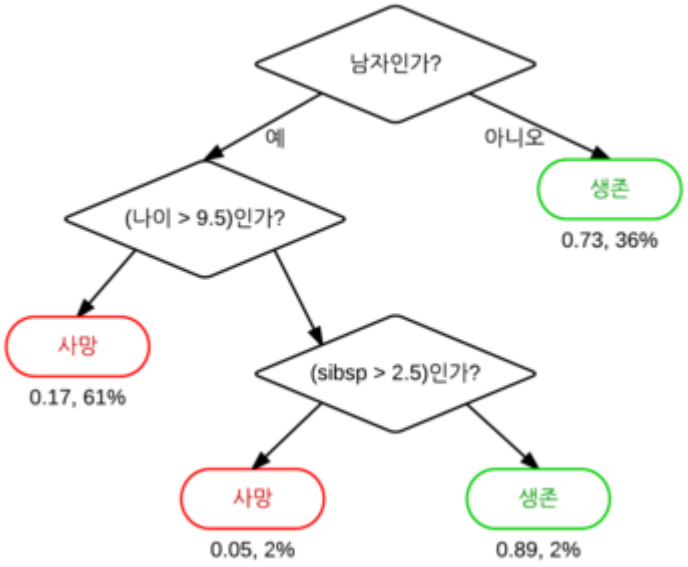
Q2. K = 10일 때 모델의 정확도(Accuracy)는?

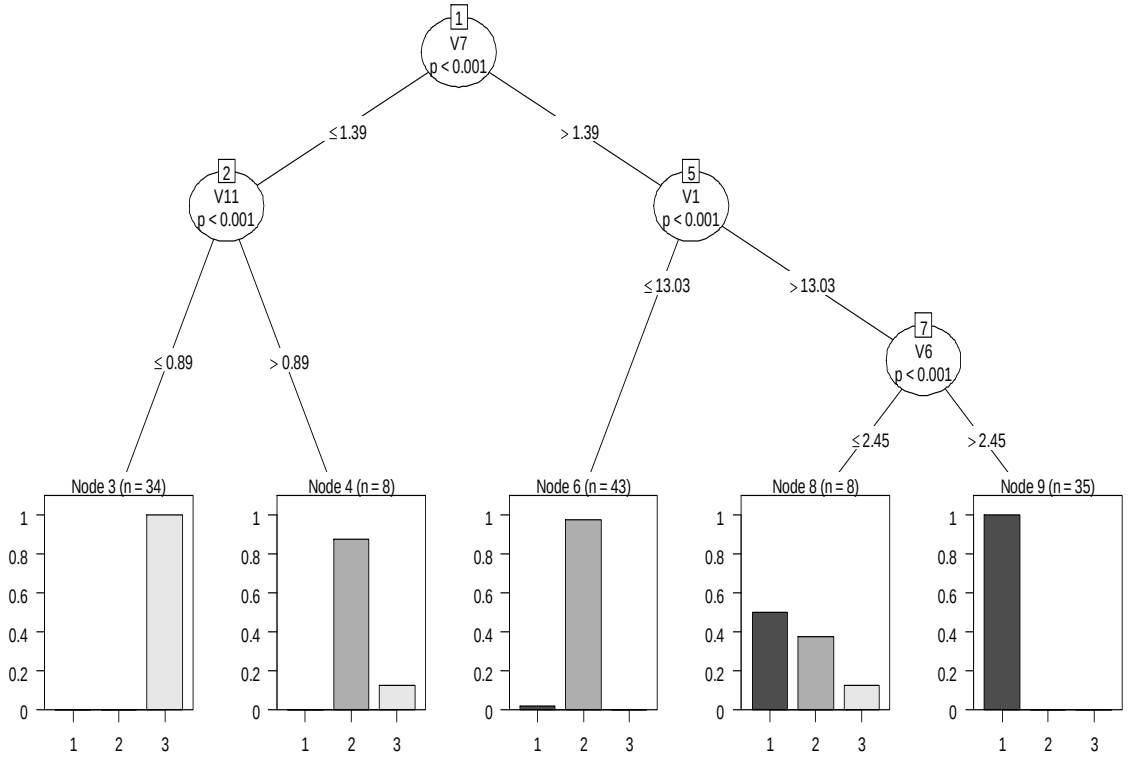
2. 의사결정나무

의사결정나무

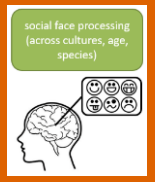
트리 기반의 회귀 및 분류 모델로, 설명변수의 공간을 분할하는데 사용되는 분할규칙들을 트리 형식으로 요약한 기법

라이브러리: **tree**, rpart, cart, C50 등





Tree의 구조

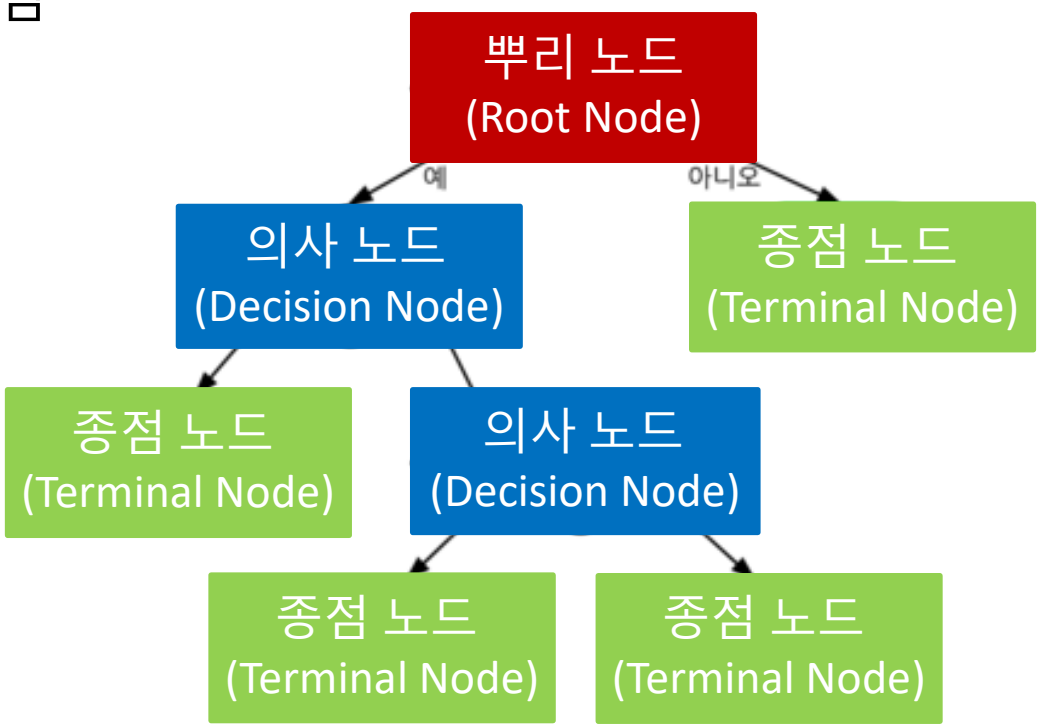


CognitiveSystems

뿌리노드(Root Node): 총 데이터로, 최초의 분할이 일어나는 시점

의사노드(Decision Node): 분할 가능성이 있는 데이터로, 의사결정에 따라 분할

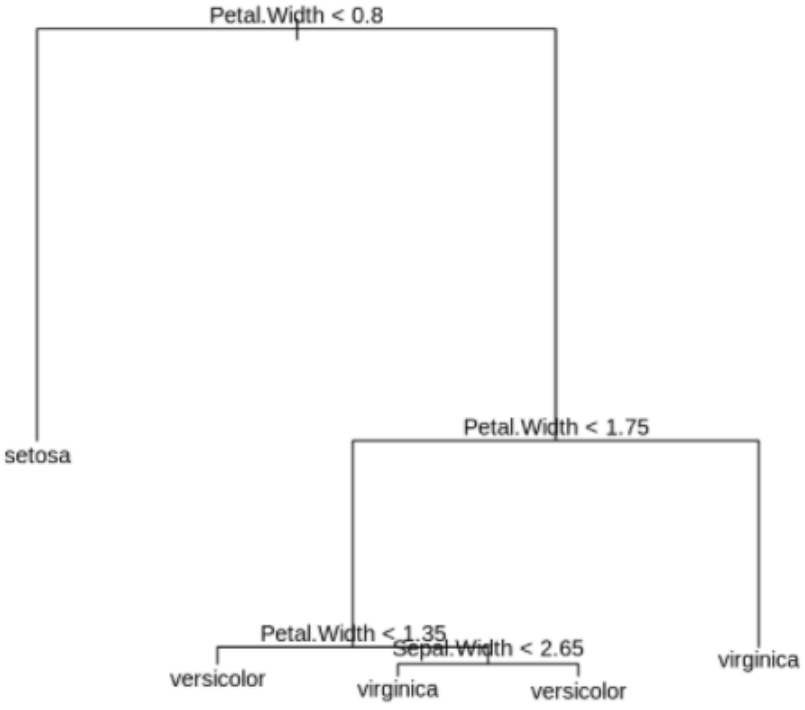
종점노드(Terminal Node): 더 이상 분할이 이루어지지 않아 의사결정이 마무리되는 지점

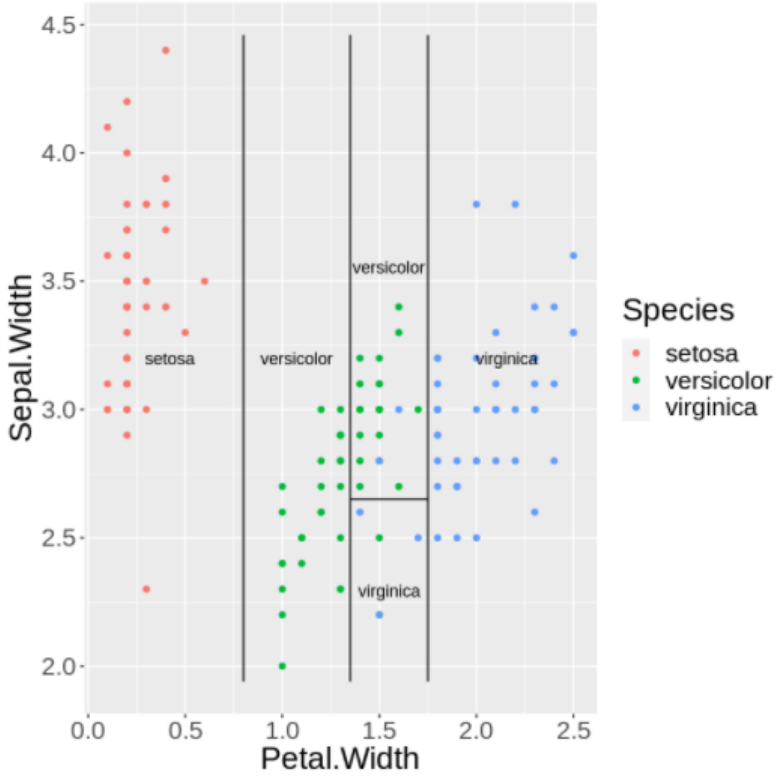


의사결정나무의 종류

1) 분류나무: 타겟이 범주형

지니 불순도(Gini index) 또는 엔트로피(Entropy)를 기준으로 분할, 분할 내의 기준값이 최소화되도록 한다.

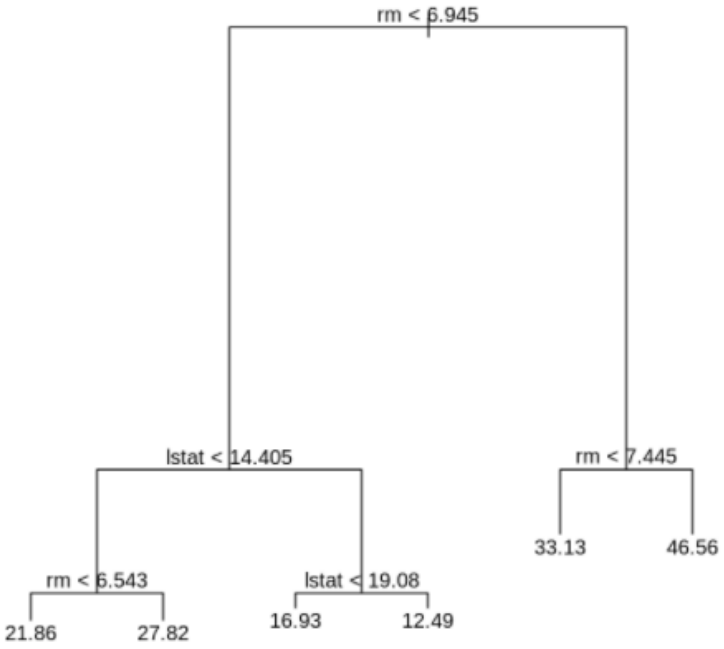


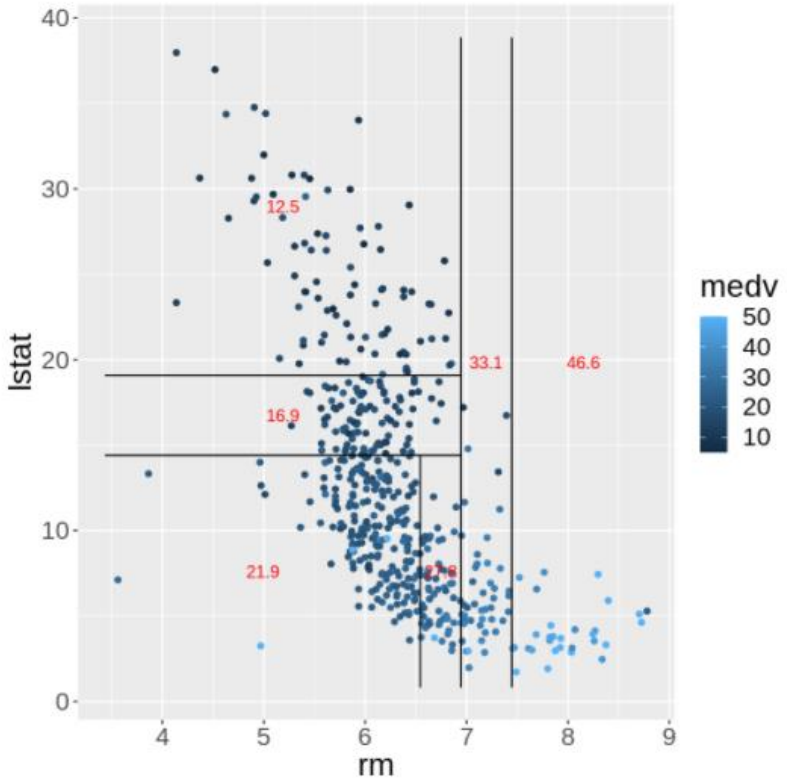


의사결정나무의 종류

2) 회귀나무: 타겟이 연속형

RSS(Residual Sum of Square)을 기준으로 분할하여, 분할 내의 RSS 값이 최대한 작아지도록 분할을 진행



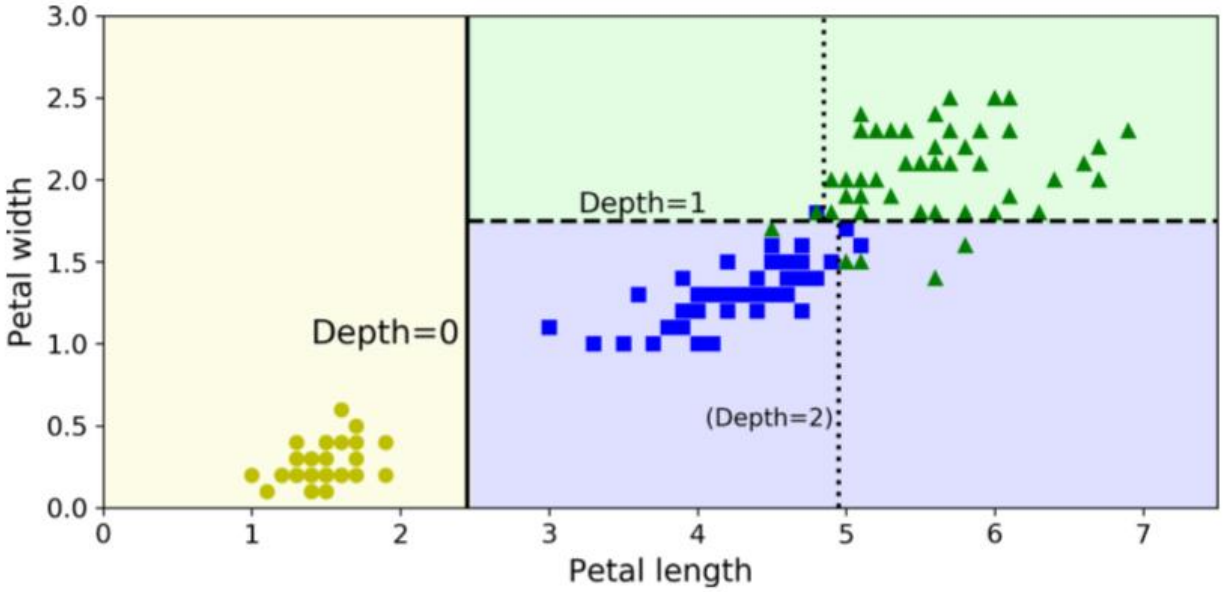


나무의 깊이

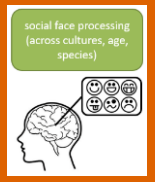
깊이(Depth): 설명공간을 분할하는 횟수
깊이가 깊어질 수록 종점노드의 수가 많아진다.

깊이가 깊을수록: 변동이 커짐, 항목간 경계가 분명해짐
- 과적합(Overfitting) 발생, Low bias and high variance

깊이가 얕을수록: 변동이 작아짐, 항목간 경계가 불분명해짐
- 부적합(Underfitting) 발생, Low variance and high bias



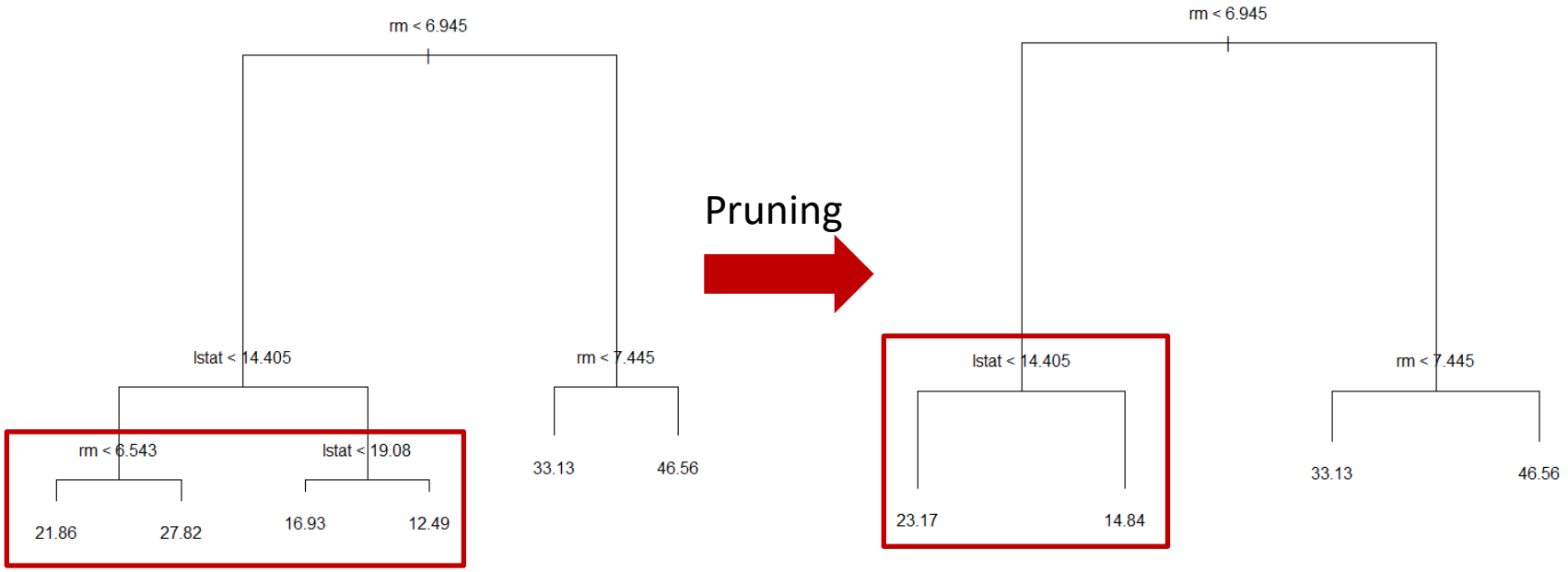
가지치기(Pruning)



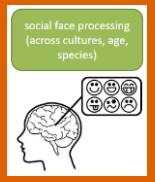
CognitiveSystems

가지치기(Pruning): 분할의 반대로, 필요 이상의 분할이 되는 곳을 합쳐서 종점 노드의 수를 줄이는 것

트리의 깊이가 깊어지면, 종점 노드 수가 많아져 더 많은 예측값을 기대할 수 있다. 그러나 너무 깊어지면 오히려 복잡도가 늘어나 과적합(Overfitting)의 위험이 있다. 따라서 Pruning을 통해 필요 이상으로 분할된 곳을 제거, 더욱 일반화된 모델을 얻을 수 있다.



의사결정나무 특징



CognitiveSystems

장점:

1) 직관적이고 설명이 쉬움 – 인간의 의사결정과 유사

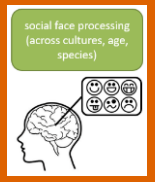
2) 만드는 방법이 복잡하지 않고 빠르면서, 한번 모델링을 하면 소속집단을 모르는 데이터를 분류하는 작업도 매우 빠름

단점:

1) 불필요한 변수가 많아지면 나무의 크기가 커져 분류 성능에 영향을 미침

2) 데이터의 구조에 따라 편향된 결과를 나타내거나 분류율이 떨어질 수 있음

의사결정나무 코드 - 분류



CognitiveSystems

ISLR 패키지 내의 데이터인 Caravan를 전처리하여, 학습할 데이터와 테스트 데이터로 나눈 후 의사결정나무 모델을 구현하였다.

```
### 1. Classification Tree
library(ISLR)
library(tree)
#Preprocessing Data
set.seed(1)
carseats<-Carseats
High = factor(ifelse(carseats$Sales<=8, "No", "Yes"))
carseats = data.frame(carseats, High)
train = sample(1:nrow(carseats),nrow(carseats)*.7)

#Model
tree.carseats = tree(High~.-Sales, data=carseats, subset = train)
```

의사결정나무 코드 - 분류

CognitiveSystems

다음 결과를 해석해보자.

```
#Summary
names(carseats)
summary(tree.carseats)
```

```
Classification tree:
tree(formula = High ~ . - Sales, data = carseats, subset = train)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Advertising" "Price"       "CompPrice"   "US"
[6] "Income"      "Age"
Number of terminal nodes:  22
Residual mean deviance:  0.4222 = 108.9 / 258
Misclassification error rate: 0.09643 = 27 / 280
```

Q1. 모델 구현에 몇 개가 사용되었는가?

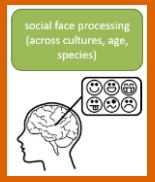
Q2. 종점 노드는 몇 개인가?

Q3. 훈련 데이터의 정확도는?

의사결정 나무의 결과를 그려보자.

```
#Plot
plot(tree.carseats)
text(tree.carseats, pretty = 0)
```

의사결정나무 코드 - 분류



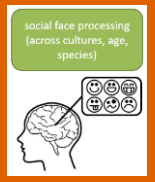
CognitiveSystems

검증 데이터를 통해 구현된 의사결정모델이 얼마나 일반화가 가능한지 알아보자.

```
#Result
prob = predict(tree.carseats,carseats[-train,])
pred = ifelse(prob[,1] >= 0.5, "No", "Yes")
table(carseats[-train,"High"], pred)
```

Q. 검증 데이터의 정확도는?

의사결정나무 코드 - 분류



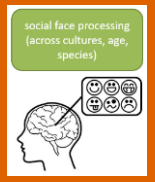
CognitiveSystems

의사결정나무를 pruning을 하여 검증 데이터의 성능이 개선되는 지 알아본다.

```
# Pruning
#Select terminal nodes = 16
prune.carseats = prune.tree(tree.carseats, best = 16)
plot(prune.carseats)
text(prune.carseats,pretty = 0)

prob = predict(prune.carseats,carseats[-train,])
pred = ifelse(prob[,1] >= 0.5, "No", "Yes")
result = table(carseats[-train,"High"], pred)
(result[1,1] + result[2,2])/sum(result)
```

Q.검증 데이터의 정확도는? Pruning 전과 후를 비교해보자.

의사결정나무 코드 - 회귀



CognitiveSystems

MASS 패키지 내의 데이터인 Boston을 전처리하여, 학습할 데이터와 테스트 데이터로 나눈 후 의사결정나무 모델을 구현하였다.

```
### Regression Tree
# Preprocessing
library(MASS)
data(Boston)
set.seed(1)
train = sample(1:nrow(Boston),nrow(Boston)*.7)
# Model
tree.boston = tree(medv~., Boston, subset = train)
```

의사결정나무 코드 - 회귀

CognitiveSystems

다음 결과를 해석해보자.

```
#Summary
names(Boston)
summary(tree.boston)
```

```
Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "rm"    "lstat" "crim"
Number of terminal nodes:  6
Residual mean deviance:  14.86 = 5172 / 348
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-11.36000  -2.25600  -0.04933   0.00000   2.16700  28.14000
```

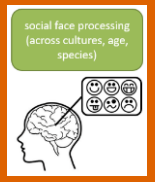
Q1. 모델 구현에 몇 개가 사용되었는가?

Q2. 종점 노드는 몇 개인가?

의사결정 나무의 결과를 그려보자.

```
#Plot
plot(tree.boston)
text(tree.boston, pretty = 0)
```

의사결정나무 코드 - 회귀



CognitiveSystems

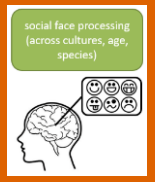
검증 데이터를 통해 구현된 의사결정모델이 얼마나 일반화가 가능한지 알아보자.

```
#Result
rsq <- function (x, y) cor(x, y) ^ 2

train_pred = predict(tree.boston,Boston[train,])
rsq(Boston[train,"medv"],train_pred)

test_pred = predict(tree.boston,Boston[-train,])
rsq(Boston[-train,"medv"],test_pred)
```

Q1. 훈련 데이터의 설명력(R2)은?

Q2. 검증 데이터의 설명력(R2)은?

의사결정나무 코드 - 회귀



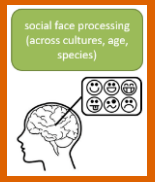
CognitiveSystems

의사결정나무를 pruning을 하여 검증 데이터의 성능이 개선되는 지 알아본다.

```
# Pruning
#Select terminal nodes = 16
prune.carseats = prune.tree(tree.carseats, best = 16)
plot(prune.carseats)
text(prune.carseats,pretty = 0)

prob = predict(prune.carseats,carseats[-train,])
pred = ifelse(prob[,1] >= 0.5, "No", "Yes")
result = table(carseats[-train,"High"], pred)
(result[1,1] + result[2,2])/sum(result)
```

Q. 검증 데이터의 설명력(R2)은? Pruning 전과 후를 비교해보자.

의사결정나무 코드



CognitiveSystems

다음 그림을 통해 설명변수의 분할 과정을 이해해보자.

1) 분류모델

```
#교육생용 Plot
student.carseats = tree(High~CompPrice+Price, carseats, subset = train)
plot(carseats$CompPrice, carseats$Price,
     xlab="CompPrice", ylab="Price", xlim = c(60,200))
partition.tree(student.carseats, add = T, cex = 1.5)
```

2) 회귀모델

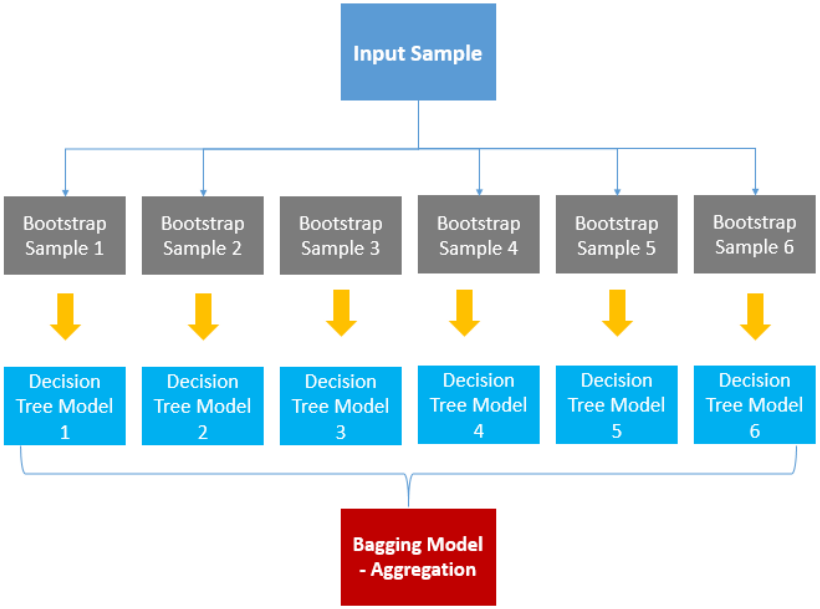
```
#교육생용
tree.boston = tree(medv~rm+lstat, Boston, subset = train)
plot(Boston$rm, Boston$lstat, xlab="rm", ylab="lstat")
partition.tree(tree.boston, add = T, cex = 1.5)
```

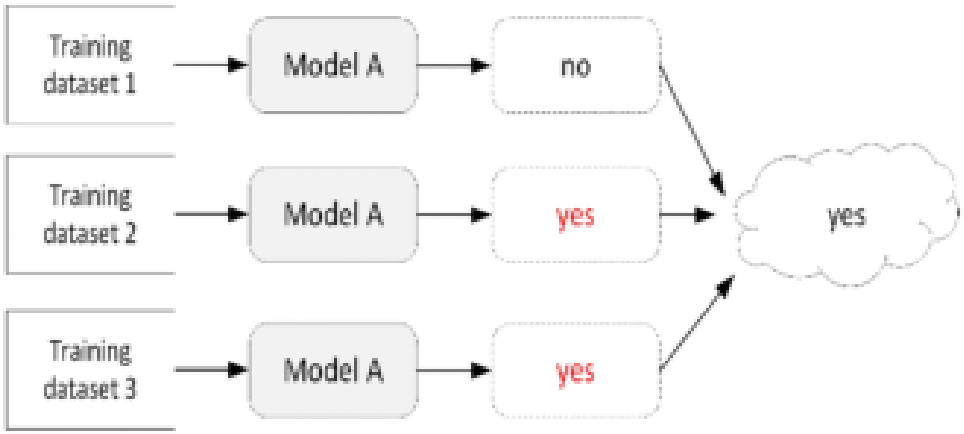
앙상블

1) 부트스트랩(Bootstrap): 주어진 데이터로부터 샘플을 반복 추출하여 원하는 값을 추정

2) 배깅(Bagging, Bootstrap aggregating): 샘플을 반복해서 뽑아(Bootstrap) 각 모델을 학습시켜 결과물을 집계(aggregating)하는 알고리즘

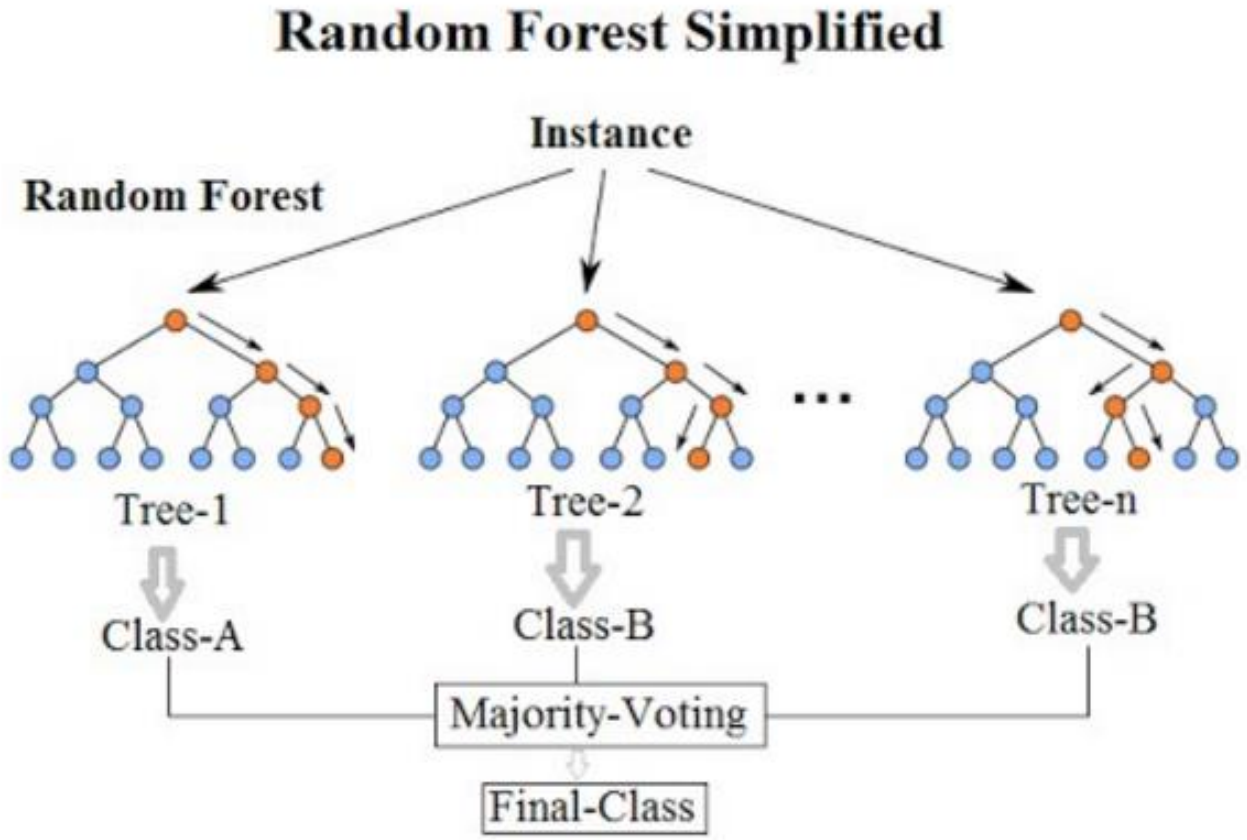
3) 보팅(Voting): 투표처럼 다수결의 결과를 따라가는 알고리즘



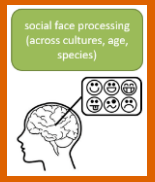


랜덤 포레스트

의사결정나무의 앙상블 모델로, 배깅을 통해 다수의 의사결정나무를 생성하고 이들을 집계하여 분류 또는 회귀분석을 하는 모델



랜덤 포레스트 특징



CognitiveSystems

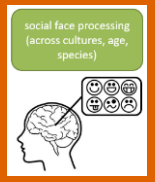
장점:

1) 의사결정나무에 비해 월등히 높은 정확성
2) 일반화 성능이 좋음

단점:

1) 구동 소요시간이 오래 걸린다.
2) 새로운 데이터가 추가되어도 급격한 성능향상이 일어나지 않는다.

랜덤 포레스트 코드



CognitiveSystems

검증 성능이 좋지 않았던 Boston 데이터에 대해 랜덤 포레스트 모델을 구현하였다.

```
## 3. Random Forest
# Model
rf.boston = randomForest(medv~., Boston, subset = train)

train_pred = predict(rf.boston,Boston[train,])
rsq(Boston[train,"medv"],train_pred)
test_pred = predict(rf.boston,Boston[-train,])
rsq(Boston[-train,"medv"],test_pred)
```

Q1. 훈련 데이터의 설명력(R2)은?

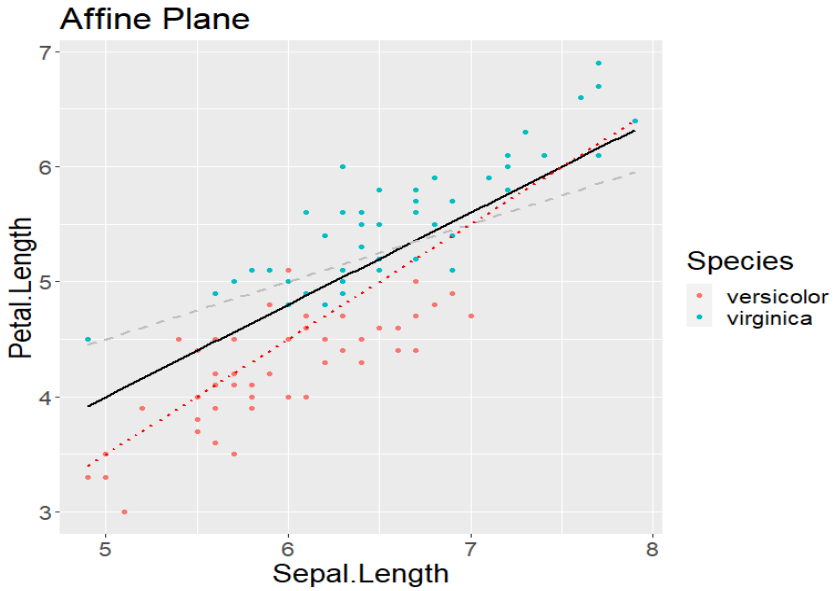
Q2. 검증 데이터의 설명력(R2)은?

3. 서포트 백터 머신

초평면

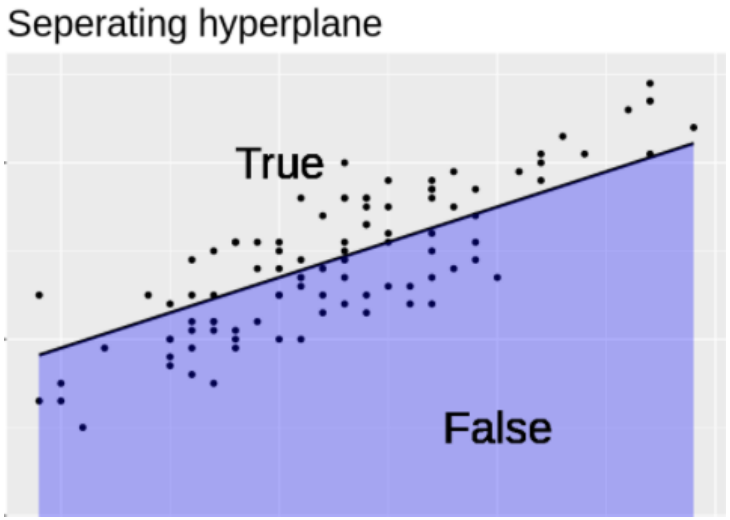
1. 초평면(Hyperplane)

N차원 공간을 분리할 수 있는
N-1차원의 평면
ex) 2D 평면의 초평면은 직선(1D)
3D 공간의 초평면은 평면(2D)



2. 분리 초평면(Hyperplane)

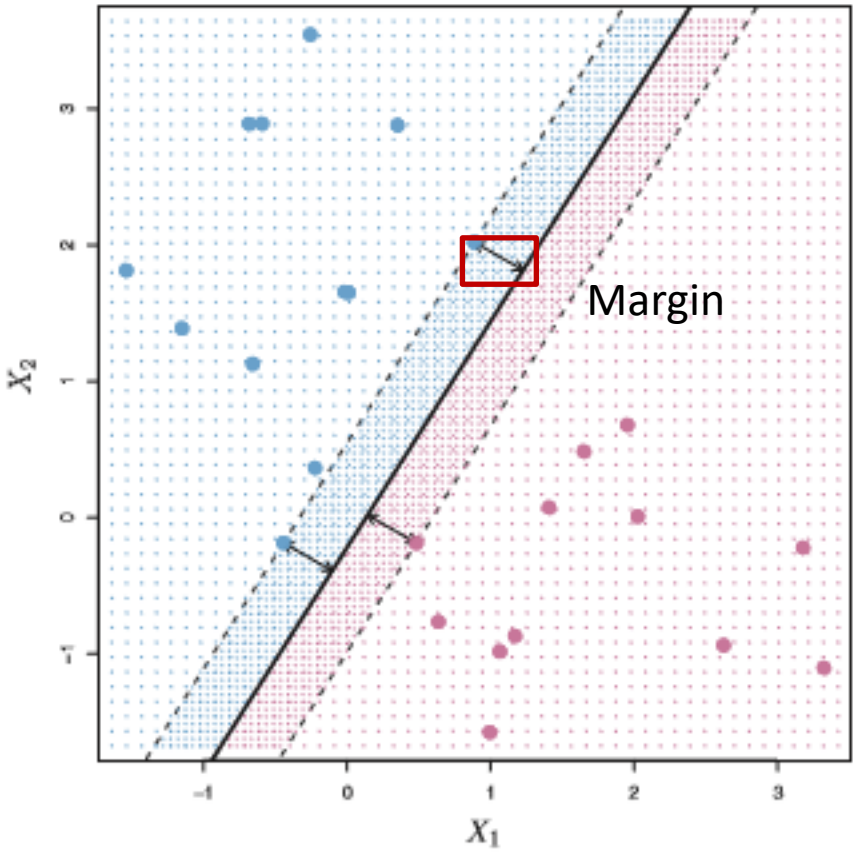
초평면을 기준으로 데이터를 분리하여
클래스를 할당하는 분류기.
의사 경계(Decision boundary) 고도 불린다.



최대 마진 분류기

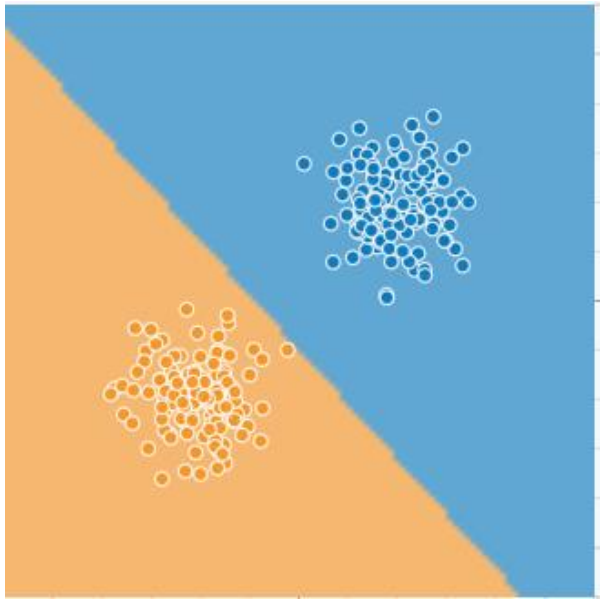
최대 마진 분류기(Maximal margin classifier)

관측치에서 초평면까지의 가장 짧은 수직 거리를 마진(Margin)이라 할 때, 이 마진이 최대가 되도록 분리 초평면을 선택하는 것.

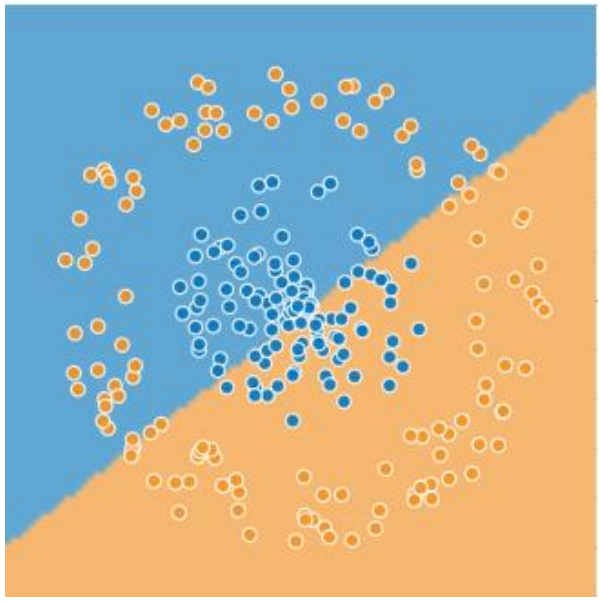


최대 마진 분류기

분리 초평면이 존재할 경우, 최대 마진 분류기는 반드시 존재.
그렇지 않을 때는 최대 마진 분류기를 이용할 수 없다.
또한, 개별 관측치에 민감하기 때문에 데이터의 작은 변화로도 분류기의 성능의 변화가 커진다.



최대 마진 분류 가능

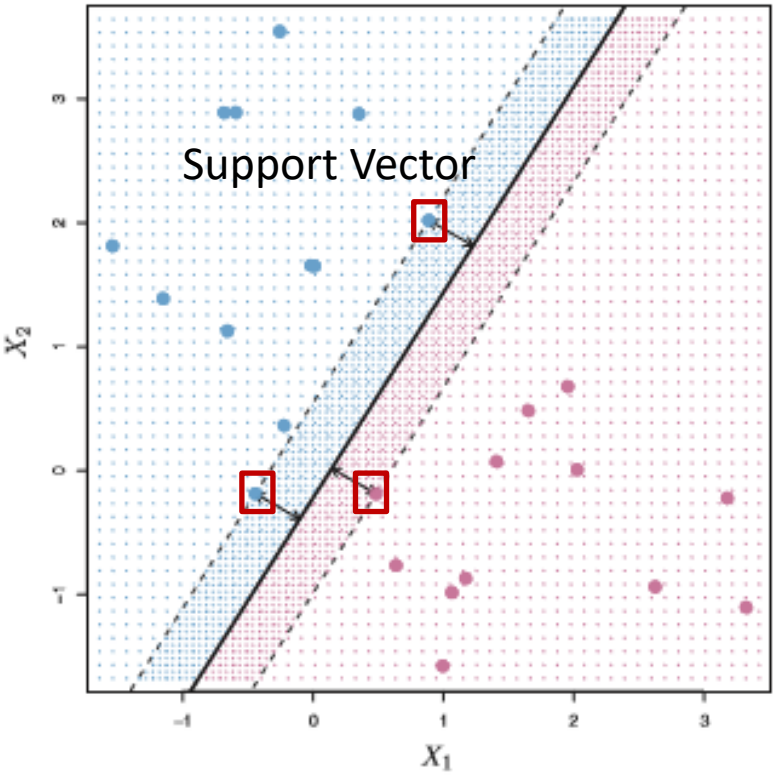


최대 마진 분류 불가

서포트 벡터 분류기

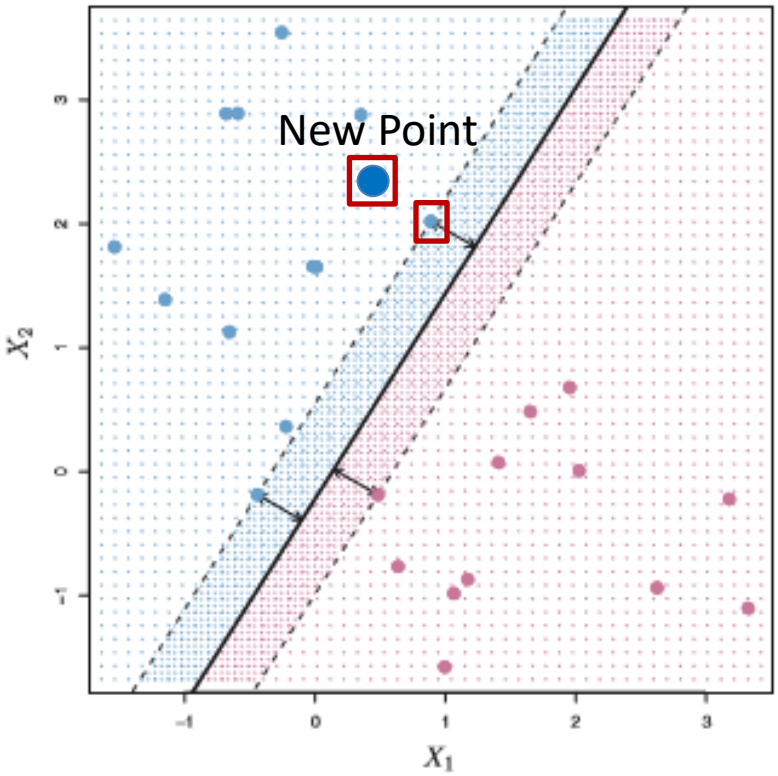
서포트 벡터 분류기(Support Vector Classifier, SVC)

소프트 마진 분류기(Soft margin classifier)라고도 불리며, 초평면에 가장 가까운 점인 서포트 벡터(Support vector)을 이용, 이들의 마진을 최대화 하도록 초평면을 결정하는 알고리즘

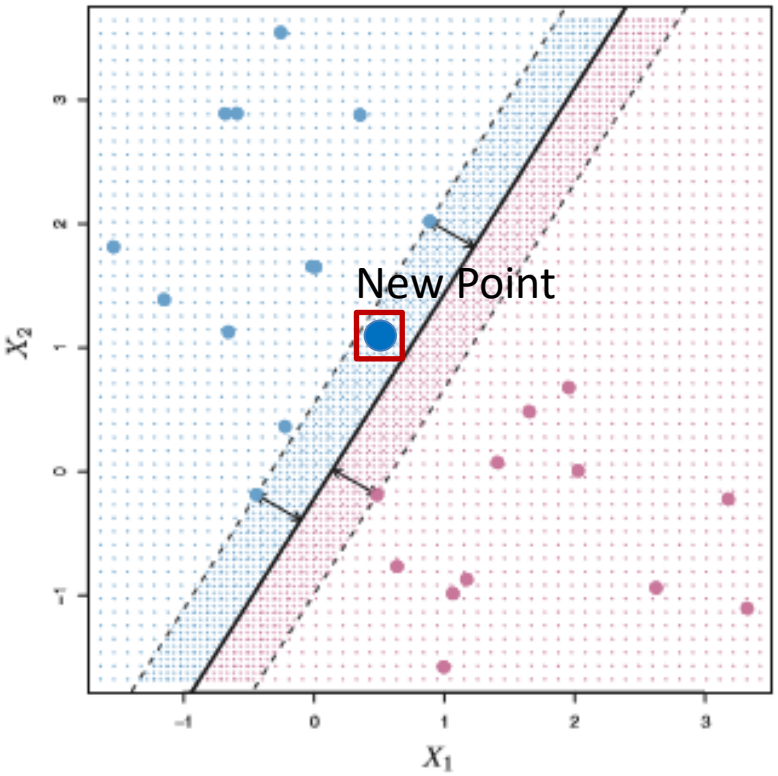


Hyperparameter: Cost

Cost(C): 마진에 대한 허용될 위반의 수와 정도를 결정하여 새로운 데이터에 대해 최대 마진 분류기보다 상대적으로 안정(Robust)적이고 효율적(Efficiency)



서포트 벡터보다 멀리 있어 모델변화 X



조율 파라미터 안에 있어 오차 허용, 모델변화 X

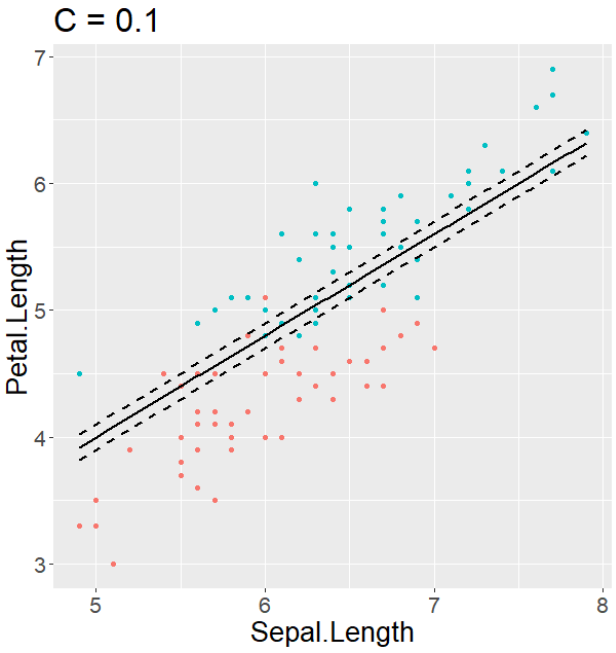
Hyperparameter: Cost

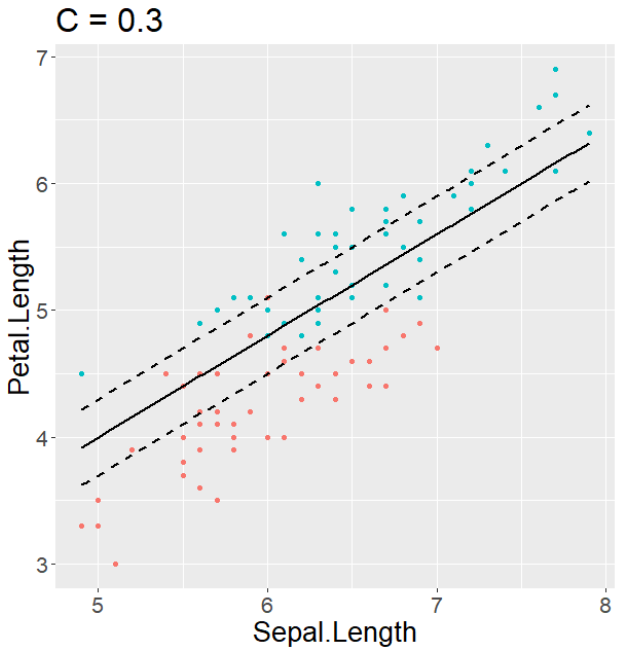
C가 커질수록: 변동이 커짐, 항목간 경계가 분명해짐

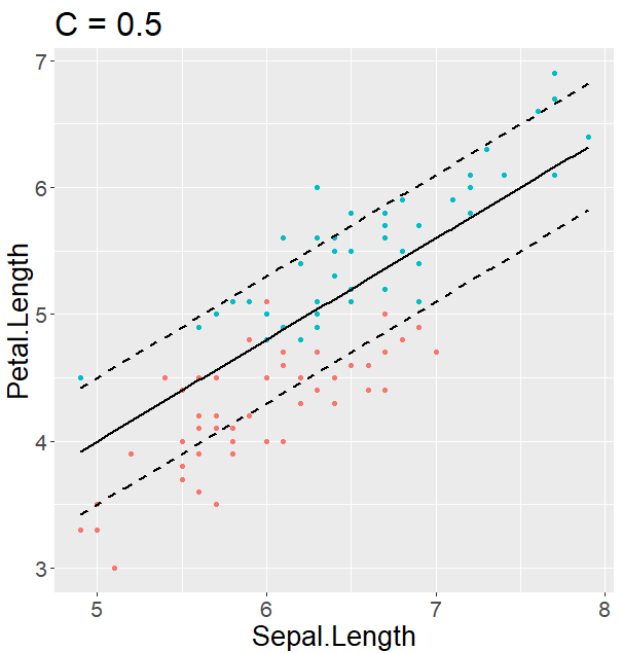
- 과적합(Overfitting) 발생, Low bias and high variance

C가 작아질수록: 변동이 작아짐, 항목간 경계가 불분명해짐

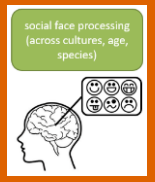
- 부적합(Underfitting) 발생, Low variance and high bias







서포트 벡터 머신

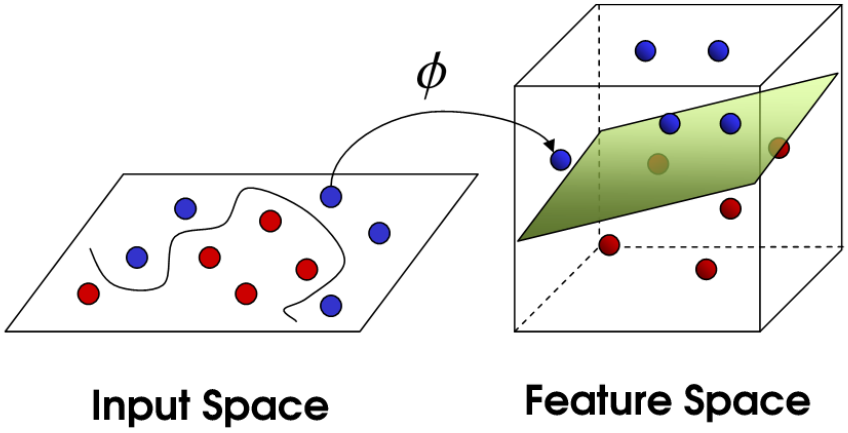


CognitiveSystems

Support Vector Machine(SVM)

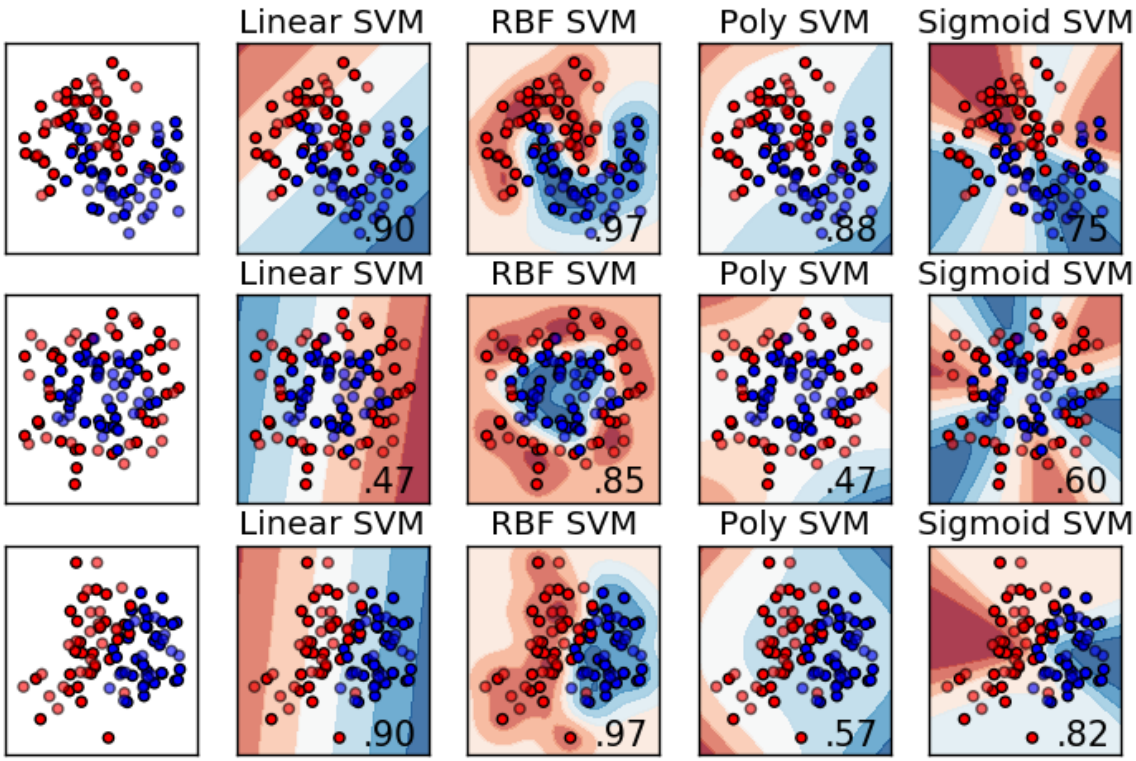
서포트 벡터 분류기의 한계인 비선형성을 해결하기 위해 커널(kernel)을 이용하여 더 높은 차원으로 변수공간을 확장, 확장된 공간에서의 초평면을 결정하는 알고리즘.

Library : **e1071**



커널의 종류

커널의 종류로는 linear, radial basis, polynomial, sigmoid가 대표적이며, 각각 커널을 적용했을 때의 분류 결과이다.



출처: https://www.kaggle.com/residentmario/kernels-and-support-vector-machine-regularization

Hyperparameter: γ

γ(Gamma)

각각의 훈련 데이터의 영향이 얼마나 멀리 도달하는지를 정의

γ의 값이 커질수록 영향이 커진다고 판단하여 좀 더 분류기를 세분화 시킬 수 있다.

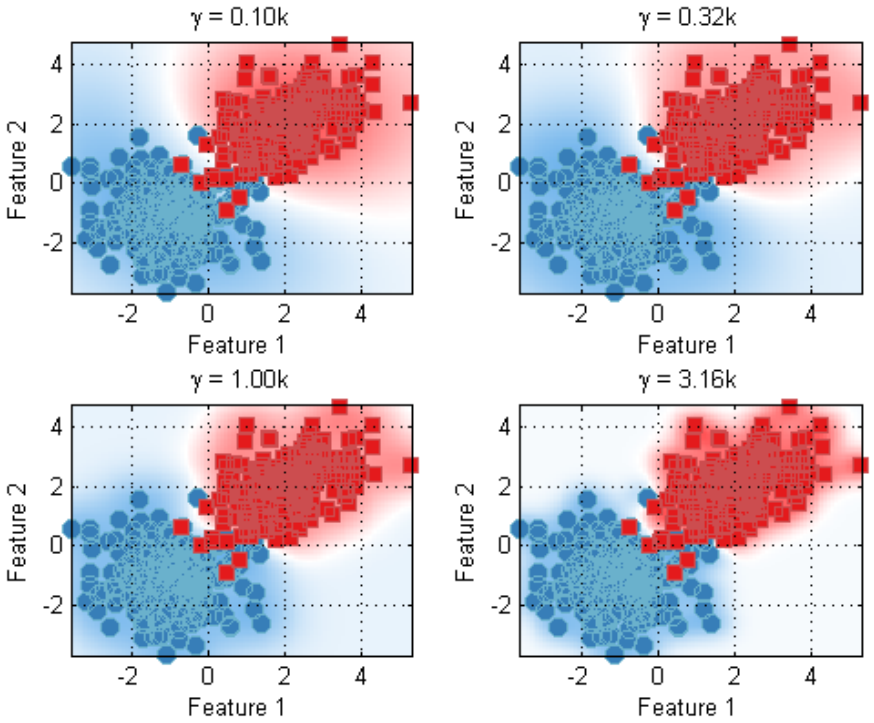
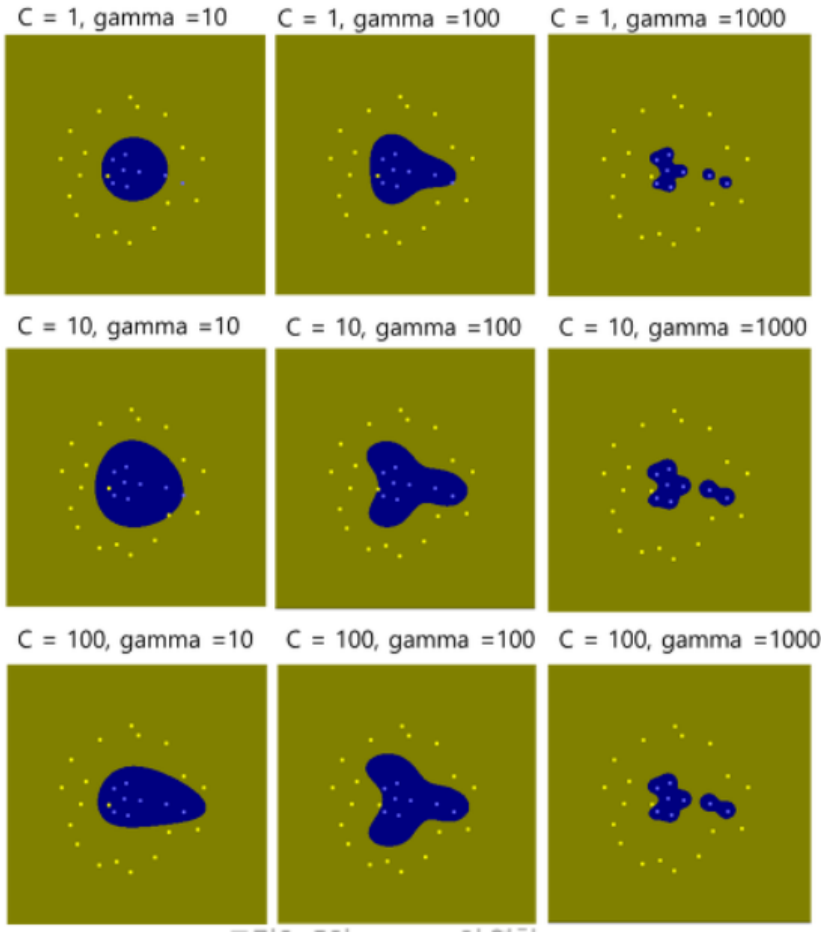


그림 출처: https://covartech.github.io/blog/2013/07/24/using-svms/

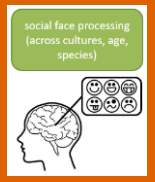
Hyperparameter

조율 파라미터의 변화에 따른 분류기의 변화를 살펴보자.



그림출처: https://bskyvision.com/163

서포트 벡터 머신 요약



CognitiveSystems

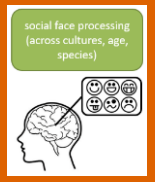
최대 마진 분류 → 서포트 벡터 분류 → 서포트 벡터 머신

- Non-linear (X)
- Robust (X)
- Efficiency (X)

- Non-linear (X)
- Robust (O)
- Efficiency (O)

- Non-linear (O)
- Robust (O)
- Efficiency (O)

서포트 벡터 머신 특징



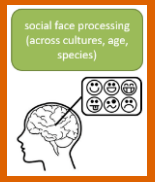
장점:

1) 고차원의 데이터에 대해 효율적이며 다중 클래스 분류 또한 가능하다.
2) 메모리 효율이 높다.
3) 선형과 비선형 모두 적용할 수 있어 다재다능하다.

단점:

1) 조율 파라미터에 민감하다.
2) 확률로 표현할 수 없다.

서포트 벡터 머신 코드(1)

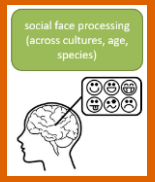


CognitiveSystems

SVM을 이해하기 위해 임의의 데이터를 만들어보도록 한다.

```
install.packages('e1071')
library(e1071)
library(dplyr)
library(ggplot2)

### Support Vector Machine
## 1. Make example data
set.seed(1)
x=matrix(rnorm(200*2), ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
plot(x, col=y)

# Preprocessing, split train:test = 7:3
train=sample(1:nrow(x),nrow(x)*.7)
```

서포트 벡터 머신 코드(1)



CognitiveSystems

다음 코드를 통해 SVM 모델을 훈련데이터를 통해 학습시키고, 해석해보자.

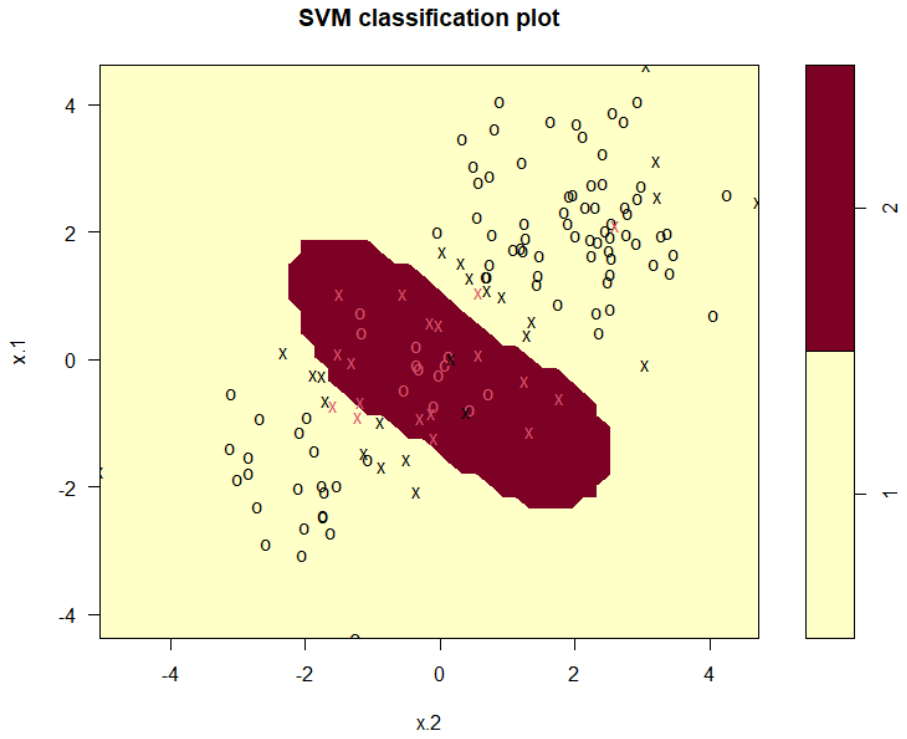
```
## 2.Model
svmfit=svm(y~., data=dat[train,], kernel="radial",  gamma=1, cost=1)
# Check Cost and support vector:
summary(svmfit)

Call:
svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1)


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

Number of Support Vectors:  42

 ( 18 24 )


Number of Classes:  2

Levels:
 1 2
```

Q. 서포트 벡터의 개수는 몇 개인가?

서포트 벡터 머신 코드(1)

서포트 벡터를 확인하고, 그림을 통해 초평면을 파악할 수 있다.

```
# X: support vectors, remaining observations as zeros
svmfit$index
# Check SVM's non-linear boundary
plot(svmfit, dat[train,])
```

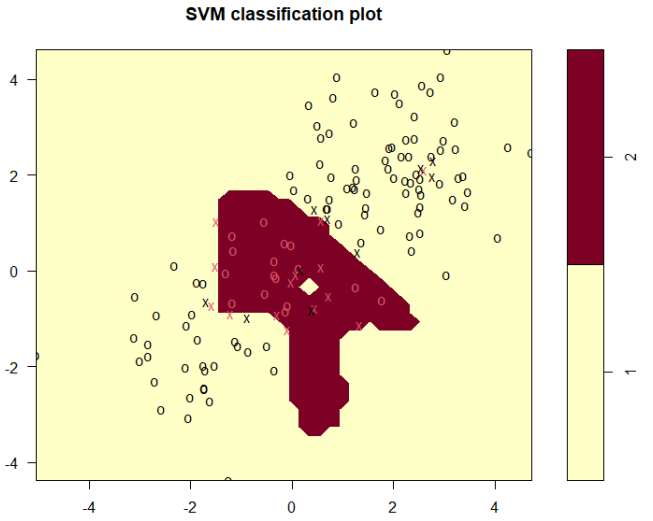


서포트 벡터 머신 코드(1)

Hyperparameter를 변경해보고, 변화를 알아보자.

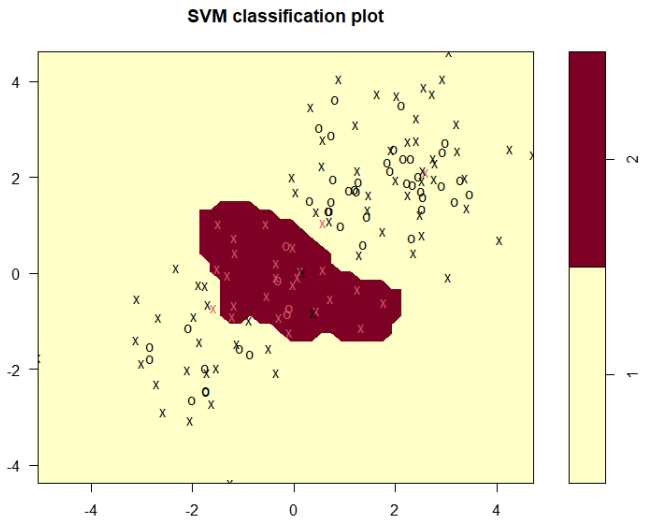
1. Cost: 1 -> 10000

```
# Change parameter : Cost
# As larger the cost, smaller # of support vectors
svmfit=svm(y~., data=dat[train,], kernel="radial",gamma=1,cost=1e4)
plot(svmfit,dat[train,])
summary(svmfit)
```



2. γ: 1 -> 10

```
# Change parameter : Gamma
# As larger the gamma, larger # of support vectors
svmfit=svm(y~., data=dat[train,], kernel="radial",gamma=10,cost=1)
plot(svmfit,dat[train,])
summary(svmfit)
```



서포트 벡터 머신 코드(1)

CognitiveSystems

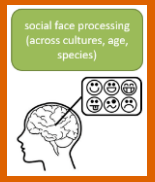
교차검증을 통해 Hyperparameter의 값을 변경하면서 비교해보자.

```
# we can use tune() to perform cross validation
set.seed(43)
tune.out=tune(svm, y~., data=dat[train,], kernel="radial",
              ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))
summary(tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.05

- Detailed performance results:
   cost      error dispersion
1 1e-01 0.21428571 0.10647943
2 1e+00 0.06428571 0.04054616
3 1e+01 0.05000000 0.04821061
4 1e+02 0.05000000 0.04821061
5 1e+03 0.05714286 0.05634362
```

Q. 어떤 파라미터를 사용했을 때가 가장 성능이 좋은가?

서포트 벡터 머신 코드(1)



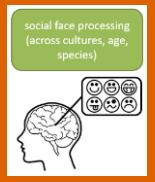
CognitiveSystems

Best model의 결과를 확인하고, 이 모델을 통해 학습된 모델을 test 데이터를 통해 검증해보자.

```
## Check the best model
bestmod = tune.out$best.model
summary(bestmod)

## Evaluate the trained model
newdata=dat[-train,]
ypred = predict(bestmod,newdata)

conf_mat = table(truth = newdata$y, predict = ypred)
accuracy = (conf_mat[1,1]+conf_mat[2,2])/sum(conf_mat)
```

Q. 정확도(accuracy)는 몇인가?

서포트 벡터 머신 코드(2)

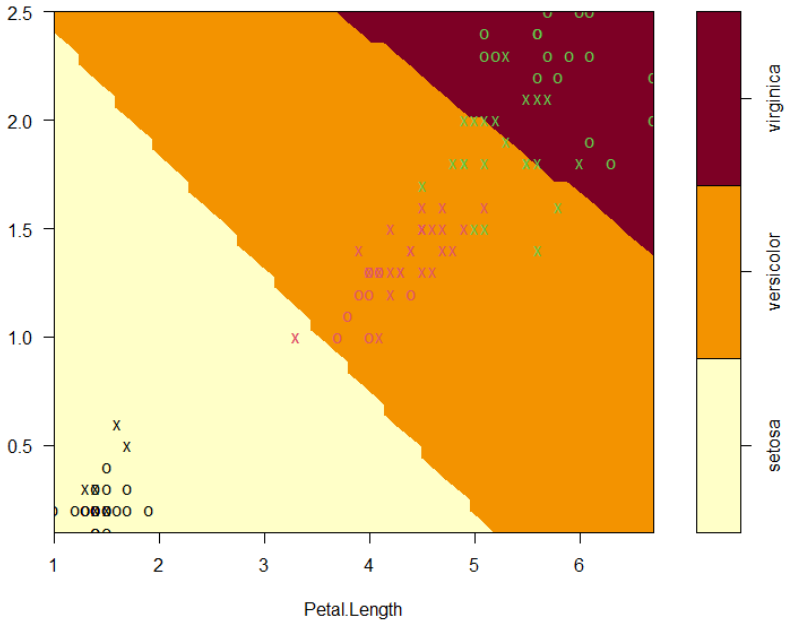


CognitiveSystems

다음은 iris 데이터를 통해 2개 이상의 클래스를 가지는 데이터의 SVM을 확인해보자.

1) Linear kernel

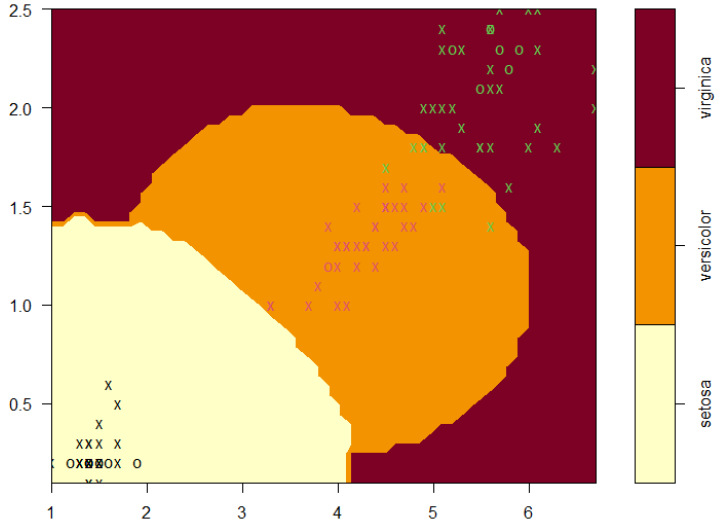
```
# Preprocessing
set.seed(1)
train_idx = sample(1:nrow(iris), nrow(iris)*.7)
IRIS = iris[train_idx,]

# SVM linear Code = SVC
svmfit_linear = svm(Species ~ ., data = IRIS,
                    kernel = 'linear', cost = 0.1, gamma = 0.5)
summary(svmfit_linear)
plot(svmfit_linear, data=IRIS,
     Petal.Width~Petal.Length,
     slice = list(Sepal.Width=3, Sepal.Length=4)
)
```



서포트 벡터 머신 코드(2)

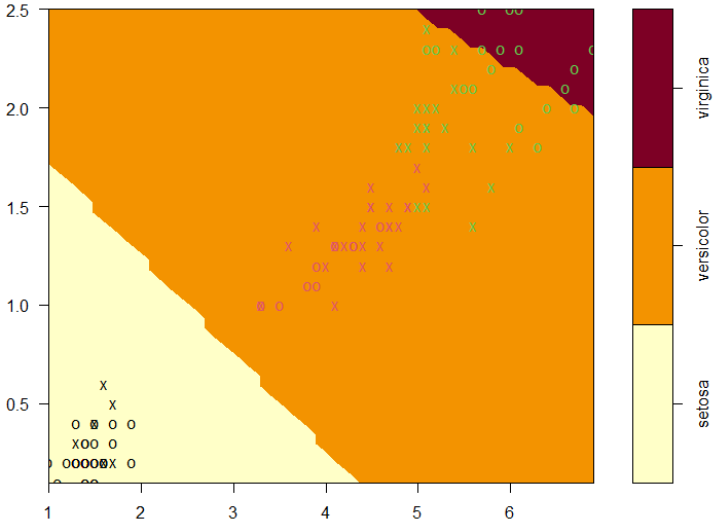
2) Radial kernel

```
# SVM radial Code
svmfit_radial = svm(Species ~ ., data = IRIS,
                    kernel = 'radial', cost = 0.1, gamma = 0.5)
summary(svmfit_radial)
plot(svmfit_radial, data=IRIS,
     Petal.Width~Petal.Length,
     slice = list(Sepal.Width=3, Sepal.Length=4)
)
```



3) Polynomial kernel

```
# SVM polynomial Code
svmfit_poly = svm(Species ~ ., data = IRIS,
                  kernel = 'poly', cost = 0.1, gamma = 0.5)
summary(svmfit_poly)
plot(svmfit_poly, data=IRIS,
     Petal.Width~Petal.Length,
     slice = list(Sepal.Width=3, Sepal.Length=4)
)
```



서포트 벡터 머신 코드(2)

CognitiveSystems

Q1. 어떤 kernel 이 가장 적합해 보이는가?

Q1에서 선택한 kernel 을 이용하여 다음 과정을 진행해보자.

```
## Choose your model!
my_model = ??  #linear or poly or radial

## Tune your model, and evaluate!
set.seed(43)
tune.out=tune(svm, Species ~ ., data = IRIS,  kernel=my_model,
              ranges=list(cost=c(0.1,1,10,100,1000), gamma = c(0.5,1,2,3,4)))
bestmod = tune.out$best.model

IRIS_test =iris[-train_idx,]
ypred = predict(bestmod,IRIS_test)

conf_mat = table(truth = IRIS_test$Species, predict = ypred)
accuracy = (conf_mat[1,1]+conf_mat[2,2]+conf_mat[3,3])/sum(conf_mat)
accuracy
```

Q2. 선택한 모델의 정확도(Accuracy)는?

서포트 벡터 머신 코드(번외)

CognitiveSystems

SVM또한 KNN, 의사결정나무처럼 회귀분석 또한 가능하다.

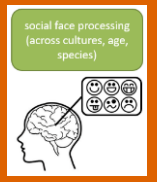
```
# SVR: Support Vector Machine Regression
svmfit_reg = svm(Petal.Length ~ Sepal.Length + Sepal.Length, data = IRIS,
                 kernel = 'linear', cost = 0.1, gamma = 0.5)
summary(svmfit_reg)

ypred = predict(svmfit_reg,IRIS_test)
rsq <- function (x, y) cor(x, y) ^ 2
rsq(IRIS_test$Petal.Length,ypred)
```

```
Call:
svm(formula = Petal.Length ~ Sepal.Length + Sepal.Length, data = IRIS, kernel = "linear",
    cost = 0.1, gamma = 0.5)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.5
    epsilon:  0.1


Number of Support Vectors:  89
```

Q&A

실습

실습



CognitiveSystems

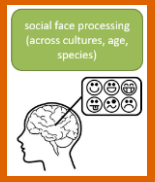
1. 다음은 신장 질환에 대한 연구로, 각 환자들의 다음 정보들을 이용하여 신장병 발병을 예측하는 모델을 구현하고 싶다.

- Glucose: 글루코오즈
- BloodPressure: 혈압
- SkinThickness: 피부 두께
- Insulin: 인슐린
- BMI: BMI
- Age: 나이
- DiabetesPedigreeFunction: 당뇨 유래 함수
- Outcome: 신장병이 발병 하였는지 (1 or 0)

KNN, decision tree, random forest, SVM 중 두가지를 이용하여 예측 모델을 생성하고 그 성능을 평가하여라.

출처: https://www.kaggle.com/uciml/pima-indians-diabetes-database

실습



CognitiveSystems

2. 다음은 부동산에 대한 데이터로, 각 매물들에 대한 정보들을 이용해서 집값을 예측하려는 모델을 구현하고 싶다.

- X1: house age (연수)
- X2: distance to the nearest MRT station (근처 지하철까지의 거리)
- X3: number of convenience stores (주변 편의점 수)
- X4: latitude (위도)
- X5: longitude (경도)
- Y: house price of unit area(평당 가격)

Decision tree, random forest, SVM 중 두가지를 이용하여 예측 모델을 생성하고 그 성능을 평가하여라.

출처: https://www.kaggle.com/quantbruce/real-estate-price-prediction