1. **TREE**
2. **randomForest**
3. **SVM**
4. **앙상블**

**1. Classification TREE**

**참고 블로그 :**

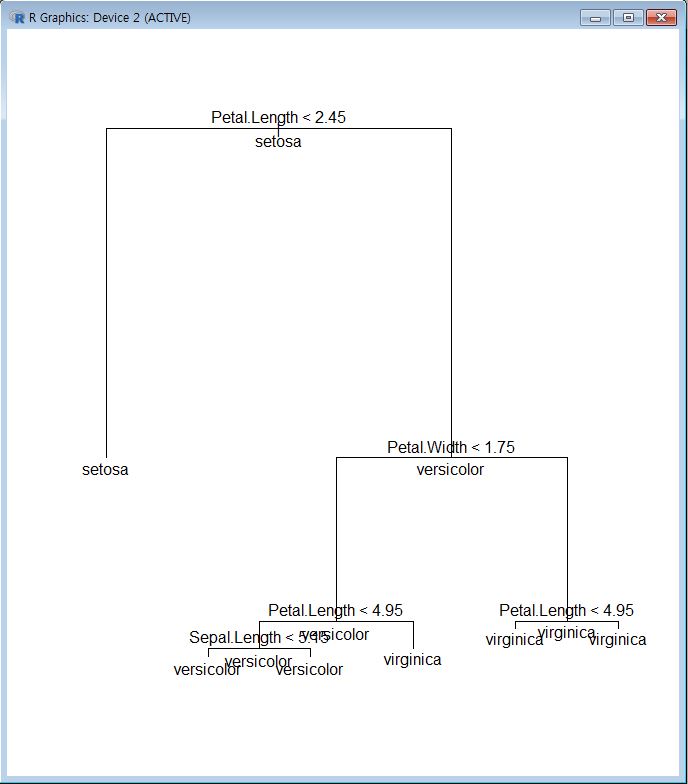
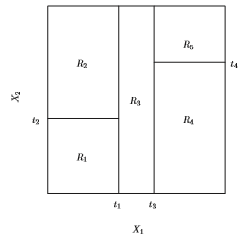
[**http://blog.naver.com/liberty264/221015963623**](http://blog.naver.com/liberty264/221015963623)

**: binary split that minimizes the “impurity” of child nodes**

**🡪 Impurity - Gini index / Deviance**

**🡪 Fitted value = majority class in a terminal node**

**- y의 값이 비슷한 구간으로 나누면 그 구간 안에서 y들은 간단한 모형으로 추정되지 않을까?**



**- tree size = terminal node의 개수 = fitted value의 개수**

**- n < p is OK**

**- TREE 미싱 있어도 상관없다.**

install.packages('tree')

library(tree)

tr<-tree(y ~ ., data=data)

plot(tr); text(tr)

tr.pred<-predict(tr, data); tr.pred

tr.pred<-predict(tr, data,type = "class")

**\*\*Bagging**

**Original data ᅳᅳᅳᅳᅳᅳᅳᅳᅳ→ Bootstrap sample 예측모형**

**Sample BS1 ᅳᅳᅳᅳᅳᅳᅳᅳᅳ→**

**with replacement BS2 ᅳᅳᅳᅳᅳᅳᅳᅳᅳ→**

**:**

**BSB ᅳᅳᅳᅳᅳᅳᅳᅳᅳ→**

**Regression ⇒ (평균을 취하면 분산이 줄어든다. 예측력이 좋아짐)**

**Classification ⇒ Voting (majority class from )**

**2. randomForests**

**: Special bagging trees with ‘decorrelated’ trees**

**→ Bootstrap sample은 독립이 아니다. Variance Reduction을 위해서는 Corr.을 감소시켜야 한다.**

**Q. How to reduce corr among the trees from bootstrap sample?**

**A. When we find the splitting variable, use just random m(<p) predictors.**

**Xj ← X1,…,Xp 중에 random하게 m개만 사용(m<p)해서 TREE를 만듬.**

**① Test error에 비해 training error가 훨씬 작다. 따라서 OOB ERROR로 비교해야 한다.**

**② Noise Variable이 많은 경우에 performance↓**

**는 simple tree보다 performance가 나쁘지만 이를 모으면 Diversity로 인해 performance가 훨씬 좋아진다.**

**③ var Importance Plot → Impurity를 최소화하는 변수가 중요 변수**

install.packages('randomForest')

library(randomForest)

rf<-randomForest(y ~ . , data=data, ntree=1000) #ntree=1000 : 예측모형의 개수

plot(rf)

varImpPlot(rf)

# varImpPlot(rf) : MeanDecreaseGini가 높을수록 중요한 변수

**3. Support Vector Machines( SVM )**

**참고 블로그**

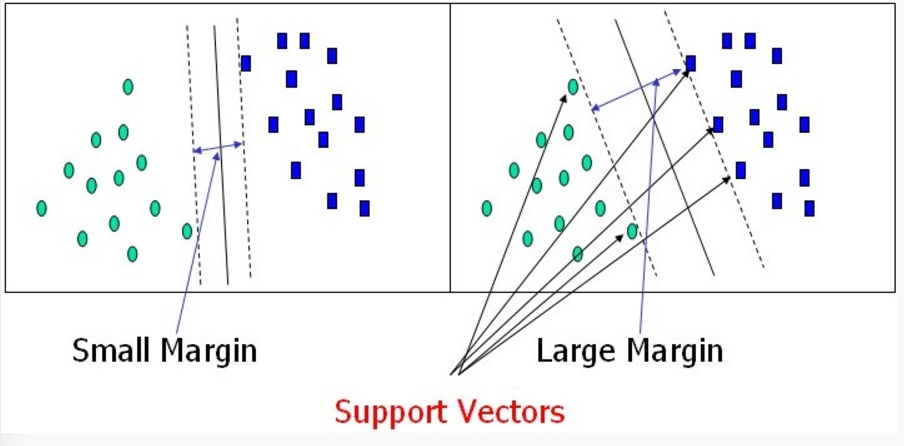
<http://blog.naver.com/tjdudwo93/221051481147>

<http://blog.naver.com/nieuwendyk/220820706749>

**# SVM find the decision boundary that maximizes “Margin”**

**# “Margin”=The shortest distance between the obs & dec bdry**

**: 서로 다른 분류에 속한 데이터 간에 간격이 최대가 되는 선 (평면)을 찾아 이를 기준으로 데이터를 분류하는 모델**



**# Support vector : obs that one used to determine dec bdry**

**# 분포가정 필요없음.**

**# n<<p is O.K**

**# If (# of class)k=4, Y={1,2,3,4}**

**①One vs One pred.class ②One vs All score**

**(1,2) f1 → 1 1, (2,3,4) → =0.71 ⇒ 1**

**(1,3) f2 → 1 voting 2, (1,3,4) → =0.68**

**(1,4) f3 → 1 1 3, (1,2,4) → =0.45**

**(2,3) f4 → 2 4, (1,2,3) → =0.21**

**(2,4) f5 → 4**

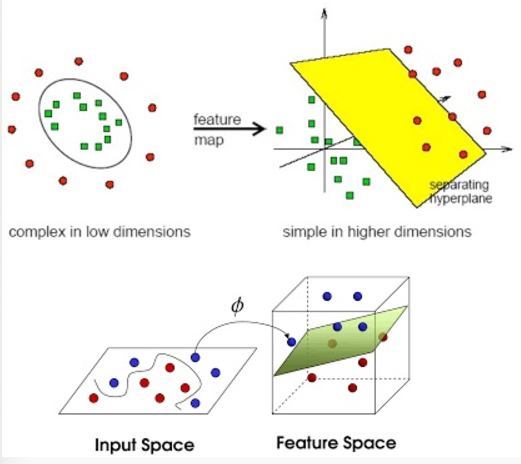
**(3,4) f6 → 3**

**# black-box 모델 : 왜 이런 결과가 나왔는지 알 수 없음**

**# X’s standardize**

**# gamma / cost(어느정도 오류를 허용해줄것인가)만 잘 찾으면 성능도 좋다.**

**# scale, kernel**

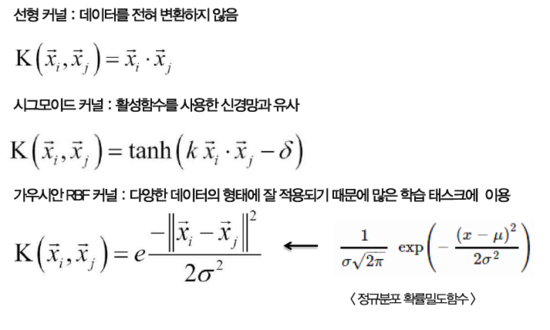


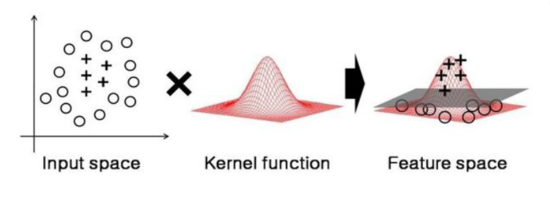
좌측그림으로는 두 데이터를 나눌 수 없다. 그래서 고차원으로 자료를 바꿔서 만들어서 두 그룹을 나눌 평면(직선)을 찾을 수 있다. 이것을 "커널 트릭" 이라고 한다.

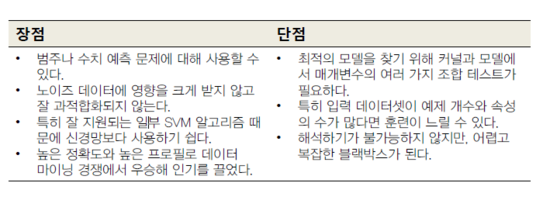
여기서, 커널(kernels)은 구별을 만들기 위해 데이터에 부가적인 차원을 추가하는 것이다. SVM의 핵심 수식은 벡터 간 내적 계산이다. 실제로 데이터를 고차원으로 변환하는 대신 고차원에서 벡터 간 내적 계산을 했을 때와 같은 값을 반환하는 함수들을 사용한다.

=> 데이터를 고차원으로 옮긴 듯한 효과만 발생시키며, 계산비용증가를 막을 수 있다.

=> 이런 함수를 '커널 함수' 라고 한다.







install.packages('e1071')

library(e1071)

svm<-svm(y ~ . , data=data\_name)

svm.pred<-predict(svm, data\_name)

table(data\_name$y, svm.pred)

# setosa versicolor virginica

# setosa 50 0 0

# versicolor 0 48 2

# virginica 0 2 48

#gamma cost를 변화시키면서 최적의 모형 찾기

svm.t1<-tune.svm(y ~ . , data= data\_name, cost=10^c(-2:0), gamma=2^c(-2:0))

# gamma=3, cost=3이므로 3\*3개의 경우의 수 비교

svm.t1

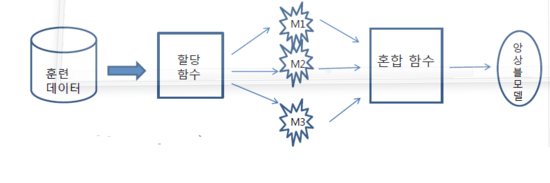
summary(svm.t1)

**4. 앙상블(Ensemble)**

**참고 블로그**

<http://blog.naver.com/lk3436/220234999214>

여러개의 분류기(classifier)를 생성하고 이것들을 결합하여 학습하는 방법으로 다양한 분류기의 예측 결과를 결합하여 단일 분류기보다 신뢰성이 높은 예측값을 얻는 것이 목표



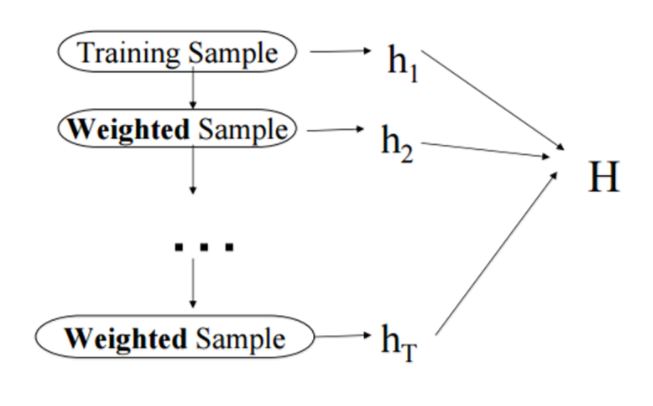
앙상블의 종류 3개 - Bootstrap aggregating (bagging) –배깅 / Boosting(부스팅) / Stacking(스태킹)

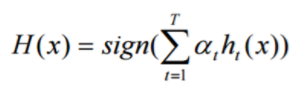
**Boosting(부스팅)**

분류기가 분류하기 어려운 사례에 집중하도록 훈련 예제들의 분포를 계속 변경시켜서 학습하는 방법이다.

weak learner로 strong learner를 만드는 것. weak learner은 아주 simple한 classifier. weak learner1를 통해서 대충 data를 classify한다. 그러면 맞은 것이 있고, 틀린 것이 있을 것이다. boosting은 여러 개의 weak learner를 사용한다고 했으므로, weak learner2를 사용해서 또 data를 classify 한다. 그런데 이때 처음에 weak learner1을 돌렸을 때 틀렸던 data들에 대해서는 조금 더 높은 weight을 두고 weak learner1에서 맞았던 data들에 대해서는 조금 더 낮은 weight을 둔다.

* weight을 어떻게 classifier에 적용시키냐? ‘알아서’





Q. 배깅과 부스팅의 차이??

1. boostrap 표본추출시, 단계별로 표본추출 확률을 조정한다.

- 이는 기존의 부스트랩 표본 추출시 모든 데이터에 등확률을 이용해 다음 표본을 추출하게 되는데,

부스팅에서는 전 단계의 모형에서 오분류된 데이터의 확률을 기존보다 높게 설정하여 표본을 추출하는 방식이다.

이는 trainning이 미흡하여 관측치에 대한 학습(learning)을 강화하여 예측 정확도를 높이려는 것이다.

2. 각 모형에서의 예측 결과 결합시, 가중평균을 사용하는 것이다.

- 기존의 배깅에서는 단순평균, 즉 다수결의 원칙에 따라 결과를 예측하였다.

하지만 부스팅에서는 각 모형의 정확도를 계산하여 이에 맞게 가중치를 주는 가중평균을 사용한다.

즉, 정확도가 낮은 모형의 결과에는 낮은 가중치를 주어 모형마다의 가중치를 달리하게 된다.

**참고 블로그**

<http://laonple.blog.me/220834569716>

<http://operatingsystems.tistory.com/139>

<http://jetzt.tistory.com/1041> ★★★Ada, Gradient, Xg boost

**Stacking(스태킹)**

- 동일한 타입의 모델을 조합하는 배깅, 부스팅과 달리 스태킹은 다양한 학습알고리즘을 통해 구성된 모델을 조합하는 것이다.

- 조합할 분류기를 선택할 때, 메타 학습기를 이용해서 어떤 분류기를 신뢰할 수 있는지를 학습, 좋은 분류기들을 찾아내서 조합하는 것이다.

- 훈련집합에 대한 오차를 판단해서 이용하는 것이 아니라, 각 분류기의 성능을 추정한다.

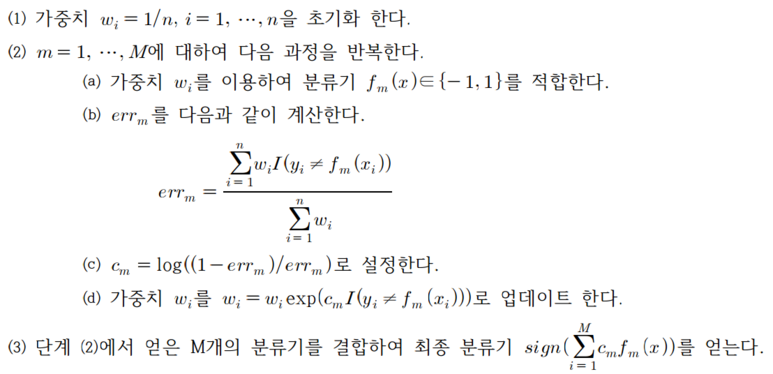
**Boosting**

**1. Apaptive Boosting**

**참고 블로그**

<http://blog.naver.com/sereno125/220984761270>

모형을 학습시키면 잘 분류되지 않은 weak learner가 존재하게 된다. 그 weak learner들을 강화하고 결합시킴으로써 강한 예측모형을 만드는 것이 Boosting 알고리즘이다. 최초의 boosting 알고리즘은 Adaboost인데 아래와 같다. 에러가 큰 learner에 대한 가중치를 높임으로써 적은 learner에는 낮춤으로써, 계속하여 분류기를 업데이트하는 것을 볼 수 있다.



**2. Gradient Boosting**

기존의 부스팅 알고리즘을 최적화 알고리즘인 Gradient decent로 해석한 알고리즘이다. 손실함수의 최저값을 찾아가는 방식이다.

**3. XG Boosting**