



예측 모델

KU-BIG 학술부



1. Introduction

Modeling (*모델 : 가정에 따라 생성될 수 있는 함수들의 집합)

1. 모델 정하기(데이터 형태 가정하고 목적을 고려)
2. 모델의 학습 목표 수식화하기
3. 실제 데이터로 모델 학습하기(최적화)
4. 평가하기
5. 위의 과정 반복하기

1. Introduction

Modeling (*모델 : 가정에 따라 생성될 수 있는 함수들의 집합)

1. 모델 정하기(데이터 형태 가정하고 목적을 고려)
2. 모델의 학습 목표 수식화하기
3. 실제 데이터로 모델 학습하기(최적화)
4. 평가하기
5. 위의 과정 반복하기

데이터가 1차식의 관계 가질 것이라고 가정.

$$y=ax$$

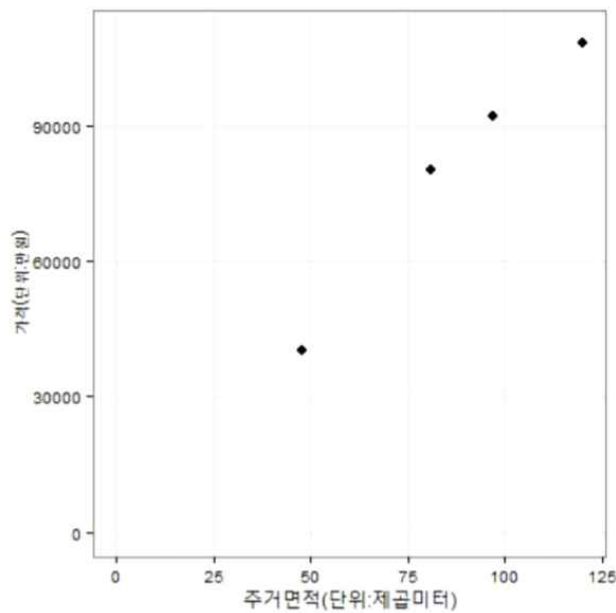
a 즉, 모델의 parameter(모수)를 추측

Loss function

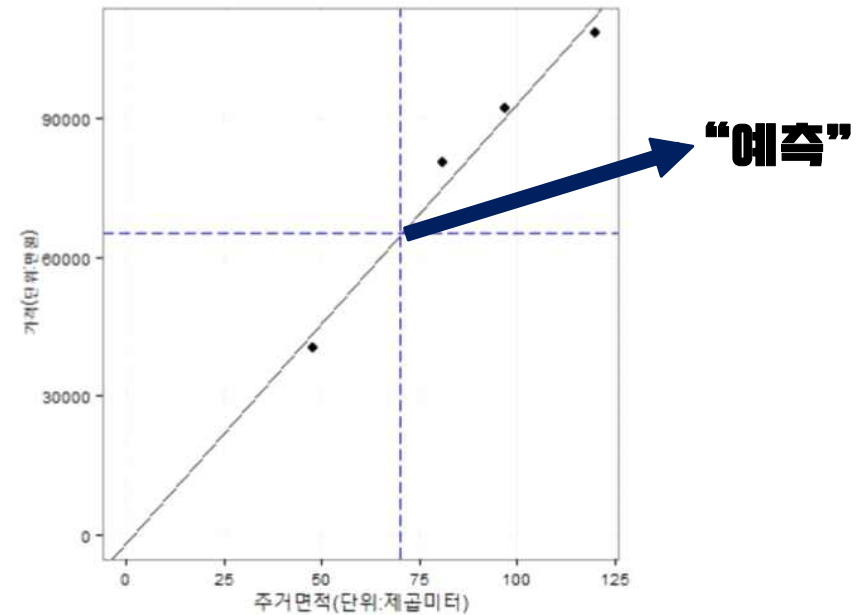
모델이 표현하는 함수 집합 중에서 가장 데이터에 적합한 함수를 찾는다

1. Introduction

예측모델링 : 데이터와 통계를 활용하여 데이터 모델의 결과를 예측하는 절차



<출처 - 2015년 19월 기준, 출처: 국토교통부 실거래가 공개시스템>



아파트 실거래가 데이터에 추정선을 추가한 플롯.
파란색 수직, 수평선은 각각 $x=70$, $y=65000$ 위치에서 그린 직선이다.

2. Linear based model

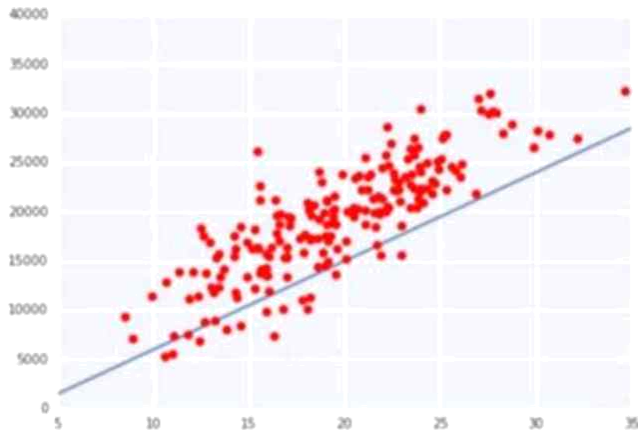
*Regression Analysis(회귀분석)

$$y = h(x_1, x_2, \dots, x_k; \beta_1, \beta_2, \dots, \beta_k) + e$$

Linear Regression(선형 회귀 모형)

종속변수 y 와 한 개 이상의 독립변수 x 와의 선형 관계를 통해 결과값을 예측하는 방법

$$y_i = \alpha + \beta x_i + \varepsilon_i.$$



종속변수 y : 택시요금
독립변수 x : 이동 거리

2. Linear based model

Linear Regression(선형 회귀 모형)

최소제곱법(LSM) : 잔차의 제곱합을 최소화시키는 방법

잔차제곱합(SSE) : $\sum (y_i - \hat{y}_i)^2$.

단순한 모델 -> 해석이 쉬우나 많은 가정을 필요로 한다.

1. 종속변수와 독립변수는 선형관계
2. 오차항은 평균이 0이고 분산이 일정한 정규분포 따름
3. 독립변수와 오차항은 서로 독립

2. Linear based model

Generalized Linear Model (GLM : 일반화 선형모형)

- > 오차항의 정규성 조건이 만족하지 않은 경우 사용하는 모델
- > 연속형 반응변수에 대한 회귀모형, 분산분석모형, 범주형 반응변수에 대한 모형 포함
- > 각 반응변수 특성에 맞게 연결함수를 이용해서 종속변수와 독립변수 선형 결합

예) 반응변수가 포아송 분포를 따르는 경우,

$$g(\mu) = \log(\mu) : \text{log link function, } 0 \leq \mu < \infty$$

$$\log(\mu) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p \Rightarrow \text{Poisson Regression}$$

2. Linear based model

Penalized linear model

핵심목표 : overfitting(과적합) 방지

모델이 복잡할수록 bias는 감소하고 variance는 증가 → 과적합 문제 발생

모델의 Bias 증가시키더라도 Variance를 더욱 감소시키자!

방법 : cost function에 페널티항을 추가

2. Linear based model

Penalized linear model – Ridge Regression (L2-norm Regularization)

– L2 norm : 가중치(회귀계수)의 제곱합

$$\text{RSS}_{\text{Ridge}}(\mathbf{w}, b) = \sum_{i=1}^m (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^p w_j^2$$

$$\hat{\mathbf{w}}_{\text{ridge}} = \min_{\mathbf{w}} \sum_{i=1}^m (y_i - \hat{y}_i)^2 \text{ subject to } \sum_{j=1}^p w_j^2 \leq s$$

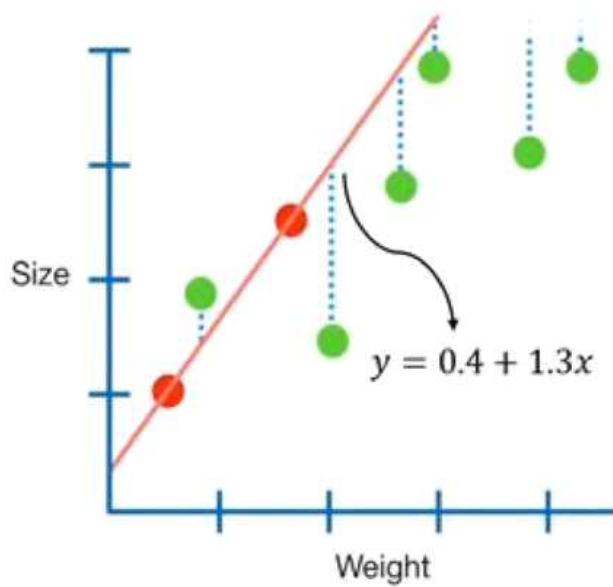
1. 큰 가중치 값을 가진 모델들에게 더 큰 페널티로 작용
2. 비용함수의 전반적인 합이 작은 모델 선호 (변수들의 회귀계수가 작은 경우)

* 알파가 크면 페널티효과가 강해져서 대부분의 회귀계수 값이 거의 0에 가까워진다. -> 단순

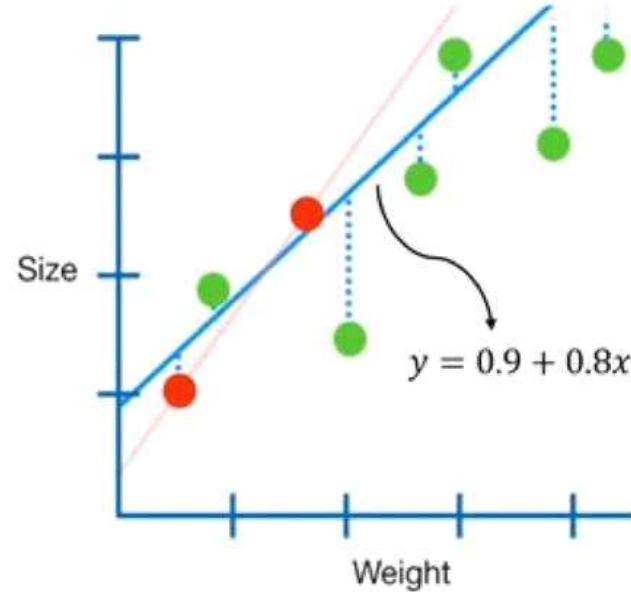
* 알파가 0이면 일반적인 선형회귀와 같은 효과

2. Linear based model

Penalized linear model – Ridge Regression (L2-norm Regularization)



1.69



0.74

Bais를 준 결과, variance가 감소한다.

예측결과의 폭을 줄여 안정적인 결과 얻는다.

2. Linear based model

Penalized linear model – Lasso Regression (L1-norm Regularization)

- L1 norm : 가중치(회귀계수)의 절대값

$$RSS_{Lasso}(\mathbf{w}, \mathbf{b}) = \sum_{i=1}^m (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^p |w_j|$$

$$\hat{\mathbf{w}}_{Lasso} = \min_{\mathbf{w}} \sum_{i=1}^m (y_i - \hat{y}_i)^2 \text{ subject to } \sum_{j=1}^p |w_j| \leq s$$

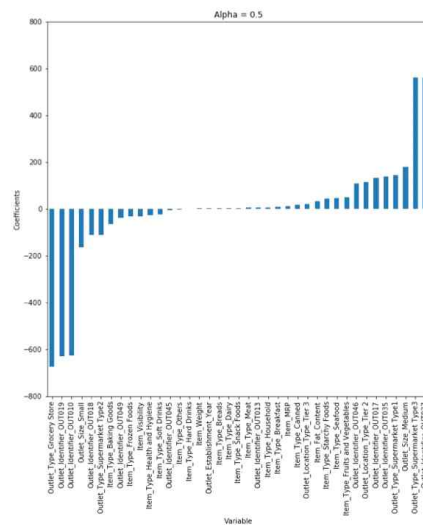
일부 회귀계수를 0으로 보낼 수 있다

-> Ridge는 0에 가깝게 보내지만 0이 되게 하지는 않는다는 점에서 차이

-> 알파값이 커질수록 제일 중요한 변수들만 남고 나머지는 0이 되기 때문에 feature selection(변수 선택)기능을 한다

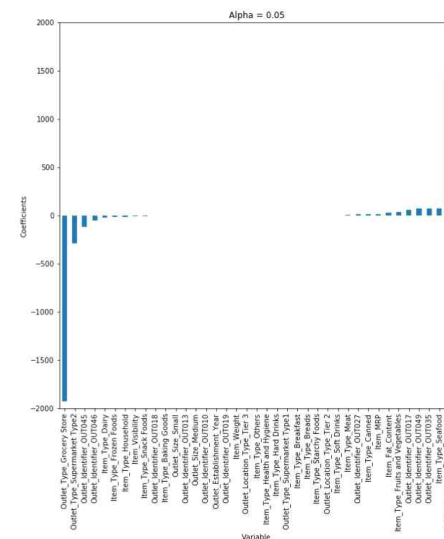
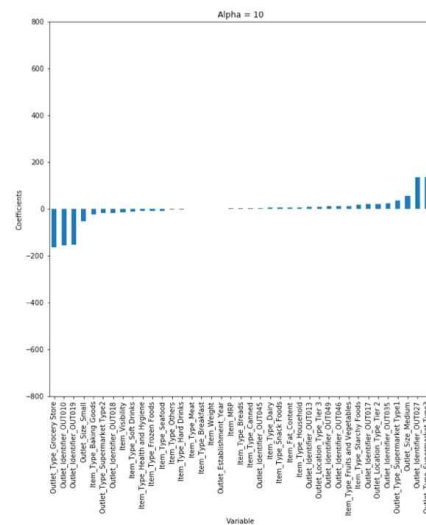
2. Linear based model

Penalized linear model



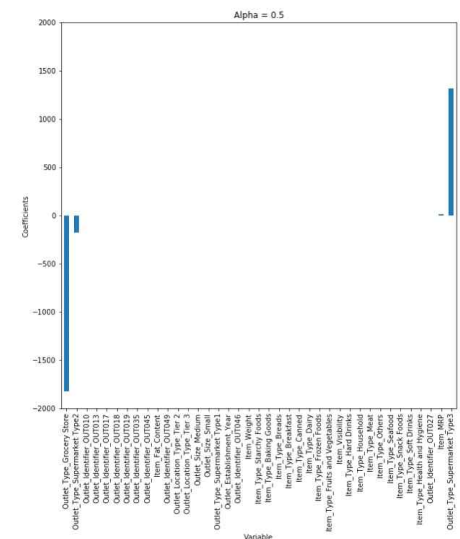
<https://bit.ly/2rSSCJu>

Ridge



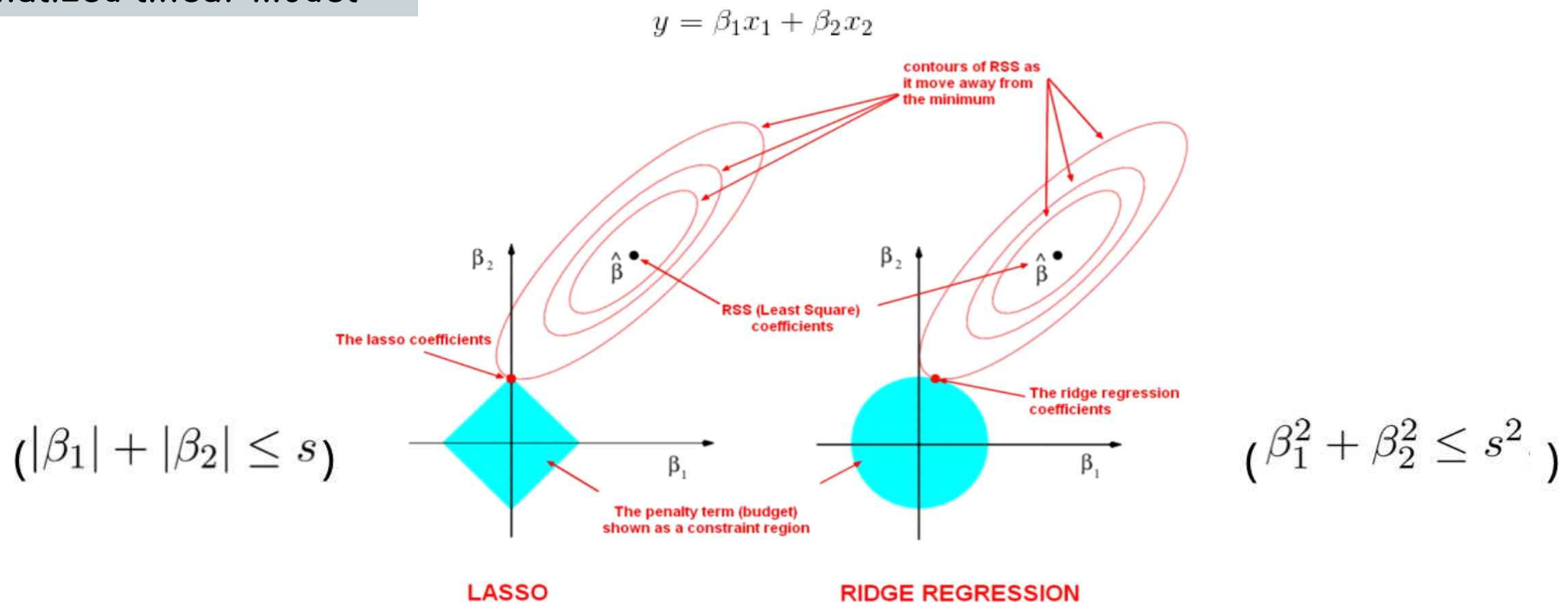
<https://bit.ly/2rSSCJu>

Lasso



2. Linear based model

Penalized linear model



<https://bit.ly/2Ugf7Hu>

2. Linear based model

Penalized linear model

일부 변수들만이 중간 이상의 영향력을 행사한다면 → Lasso

많은 변수들이 중간 이하의 영향력을 모두 행사한다면 → Ridge

2. Linear based model

Penalized linear model – Elastic Net

Lasso와 Ridge를 결합한 모델

$$\text{RSS}_{\text{Elastic}}(\mathbf{w}, b) = \sum_{i=1}^m (y_i - \hat{y}_i)^2 + \lambda \left[\frac{1}{2} (1 - \alpha) \sum_{j=1}^p w_j^2 + \alpha \sum_{j=1}^p |w_j| \right]$$

* λ : penalty의 강도

** α : Lasso penalty의 비율 $\rightarrow \alpha$ 가 1이면 Lasso, α 가 0이면 Ridge

α 와 λ 의 범위를 정해 이들의 조합 중 error가 최소가 되는 점을 찾아 선택

2. Linear based model

Penalized linear model – Elastic Net

변수들 간 상관관계가 존재할 때 유용

→ Lasso와 Ridge는 변수들을 대부분 없애기 때문에 정보 손실의 문제 발생

변수들끼리 그룹을 지어 계수들을 감소시키고, 그룹 단위로 묶어 모델에 남기거나 제거한다.

Elastic Net = Lasso의 feature elimination 기능 + Ridge의 coefficients reduction 기능

3. Classification based prediction

What is classification?

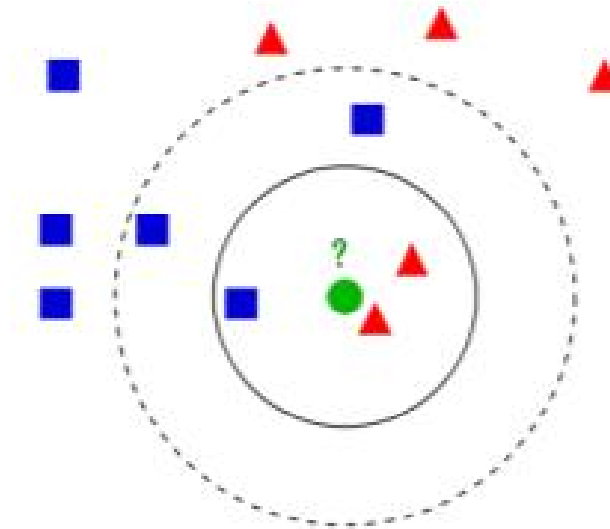
Classification has predetermined (predefined) classes
(supervised learning, label Y is available)

Data: $\{(X_i, Y_i), i = 1, \dots, n\}$ with multivariate observations
 $X \in \mathbb{R}^p$ and population labels or class membership
 $Y_i \in \{1, 2, \dots, K\}$ (categorical).

Ex) KNN

객체는 k 개의 최근접 이웃 사이에서 가장 공통적인 항목에 할당되는 객체로 과반수 의결에 의해 분류된다.
(k 는 양의 정수이며 통상적으로 작은 수).
만약 $k = 1$ 이라면 객체는 단순히 하나의 최근접 이웃의 항목에 할당된다.

3. Classification based prediction



3. Classification based prediction

Difference between non-parametric and parametric model evaluation

KNN -> 비모수적 방법 -> CV로 모델 평가 <-> 어떤 k 값이 가장 misclassification rate



회귀분석 -> 모수적 방법 -> AIC, BIC, LRT, 카이제곱 검정 등 많은 평가 방법이 존재

3. Classification based prediction

다변량 과제6

Use cross validation to select tuning parameter. Conduct k-nearest neighbor classifiers to the pendigit data, for $k = 2, 3, \dots, 15$. For each k , calculate the cross-validation error.

#I use 10-fold cross validation

```
which.k = numeric()
```

```
for(k in 2:15){
```

```
  for(i in 1:10){
```

```
    train = whole[id!=i,]
```

```
    test = whole[id==i,]
```

```
    knn_pred_y = knn(na.omit(training_data), na.omit(testing_data),
```

```
na.omit(digit.train), k = k)
```

```
    mis.knn <- c( (knn.table[2,1]+knn.table[1,2])/sum(t(diag(knn.table))) )
```

```
  }
```

```
  which.l[k] = mean(mis.knn)
```

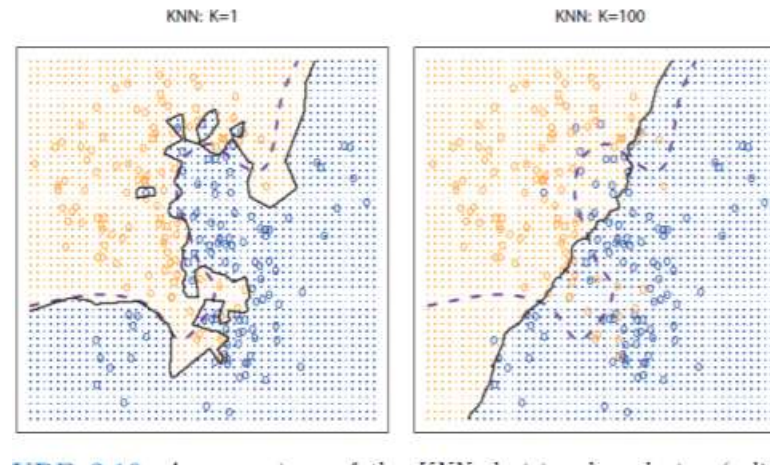
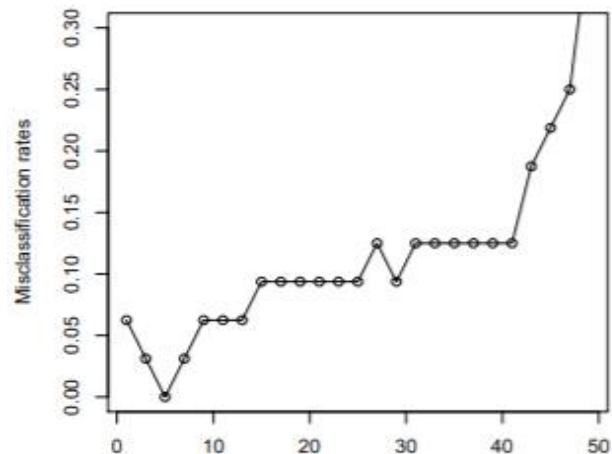
```
}
```

```
plot(which.k, ylab="error", type = 'b')
```

3. Classification based prediction

결과 해석

1. $k=5!!$
2. If k is too small, sensitive to noise points (“overfitting”).
3. If k is too large, underfitting



4. Clustering based prediction

What is clustering?

the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar in some sense to each other than to those in other groups.

4. Clustering based prediction

Contrast to classification

Classification: supervised learning, label Y is available

Clustering: unsupervised learning, No Y !!

4. Clustering based prediction

Clustering algorithms

1. Partitioning clustering: starts with an initial clustering of observations and iteratively update the clustering until the best clustering is found. ex) K-means
2. Hierarchical clustering: constructs a tree-like structure to show groups of observations

4. Clustering based prediction

Clustering algorithms -> Partitioning clustering -> K-means

```
> X
  X1 X2
A  5  4
B  4  5
C  1 -2
D  0 -3
```

```
> kmeans(X,2)
Clustering vector:
A B C D
2 2 1 1
```

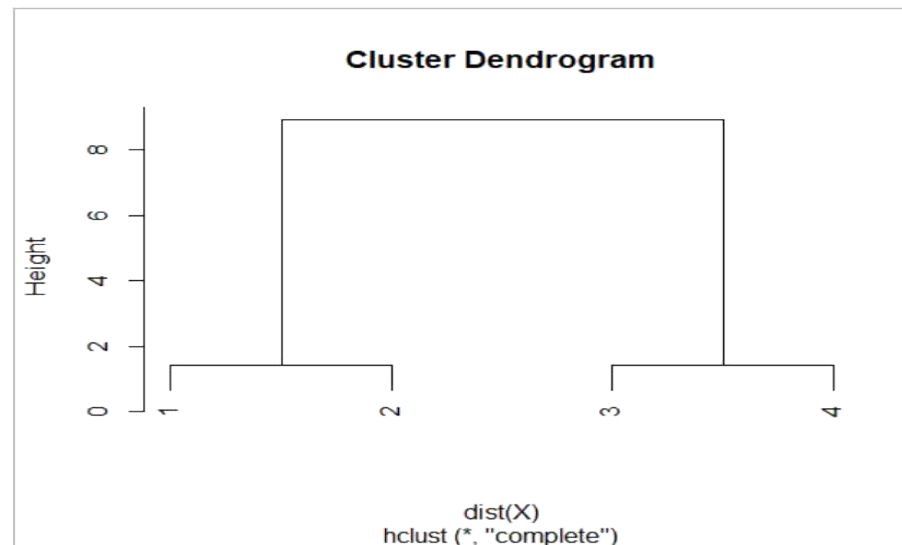
#starts with an initial partition -> calculate the centroids of each cluster in the partition -> reassigns objects to the cluster with the closest centroid.

4. Clustering based prediction

Clustering algorithms -> Hierarchical clustering

```
>plot(hclust(dist(X)))
```

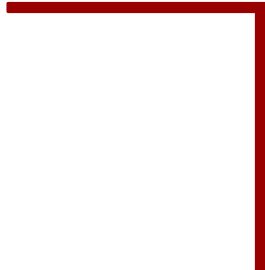
A, B, C, D A joins B at distance $d = 2 \rightarrow (A,B)$, C, D C joins D at distance $d = 3 \rightarrow (A,B), (C,D)$ (A,B) joins (C,D) at distance $d = 6$. 4



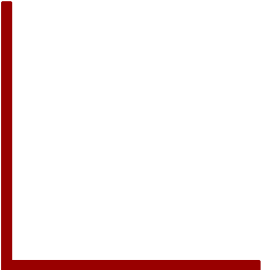


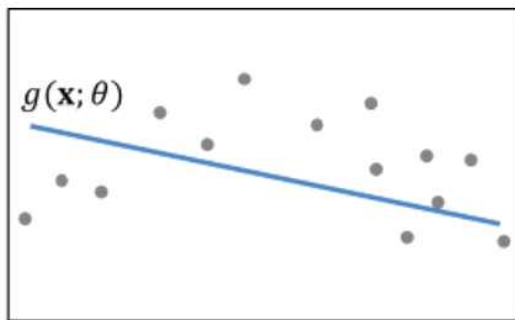
Ensemble



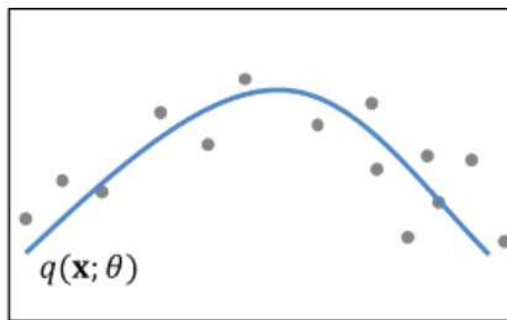


여러 모델(Weak Learner)을 올바르게 결합하여
더 정확하고 견고한 모델(Strong Learner)을 생성하는 것.

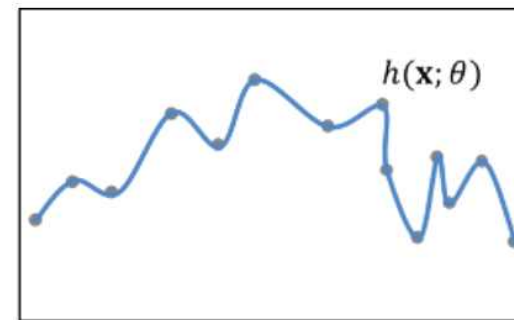




(a) Underfit



(b) Ideal fit

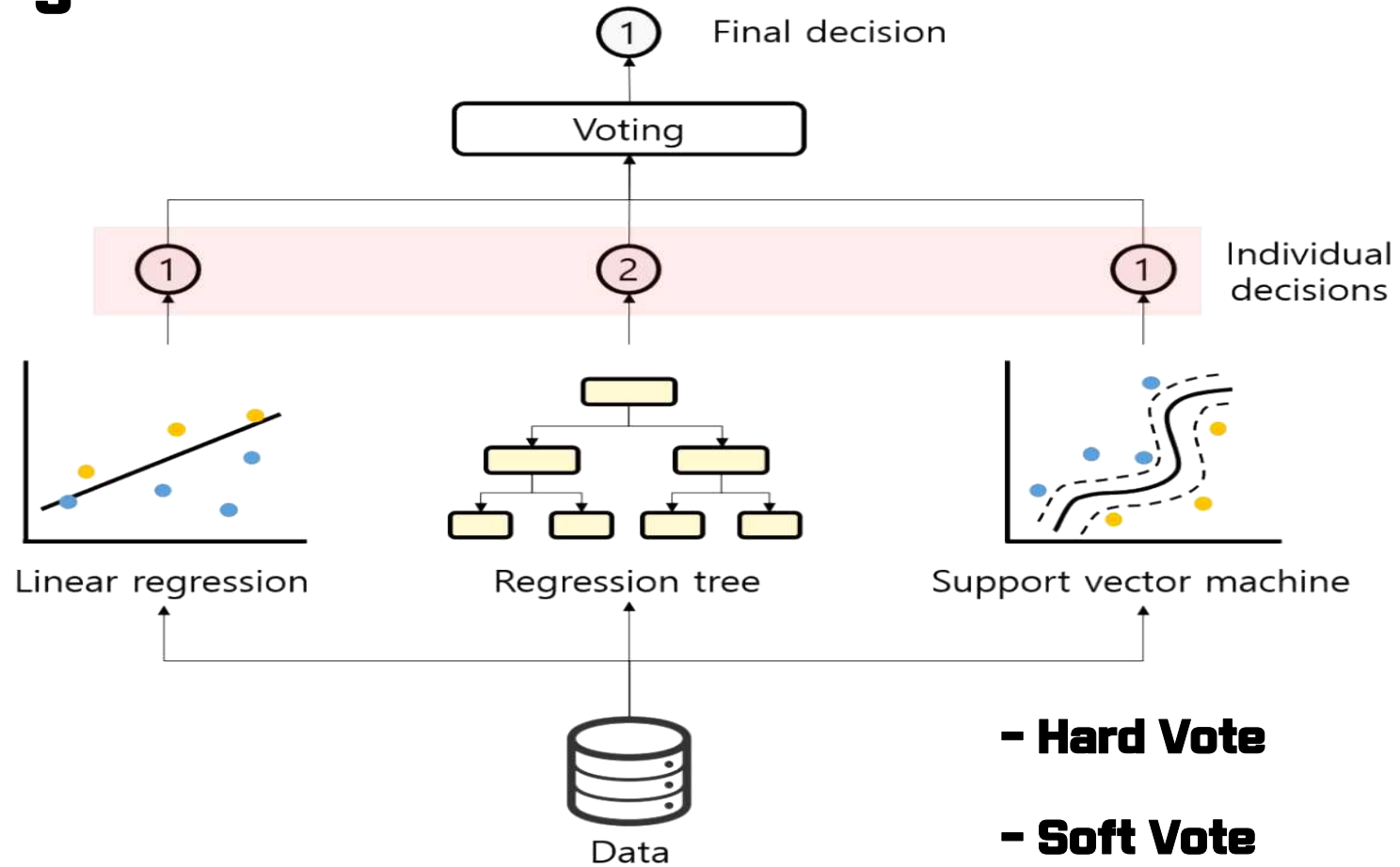


(c) Overfit

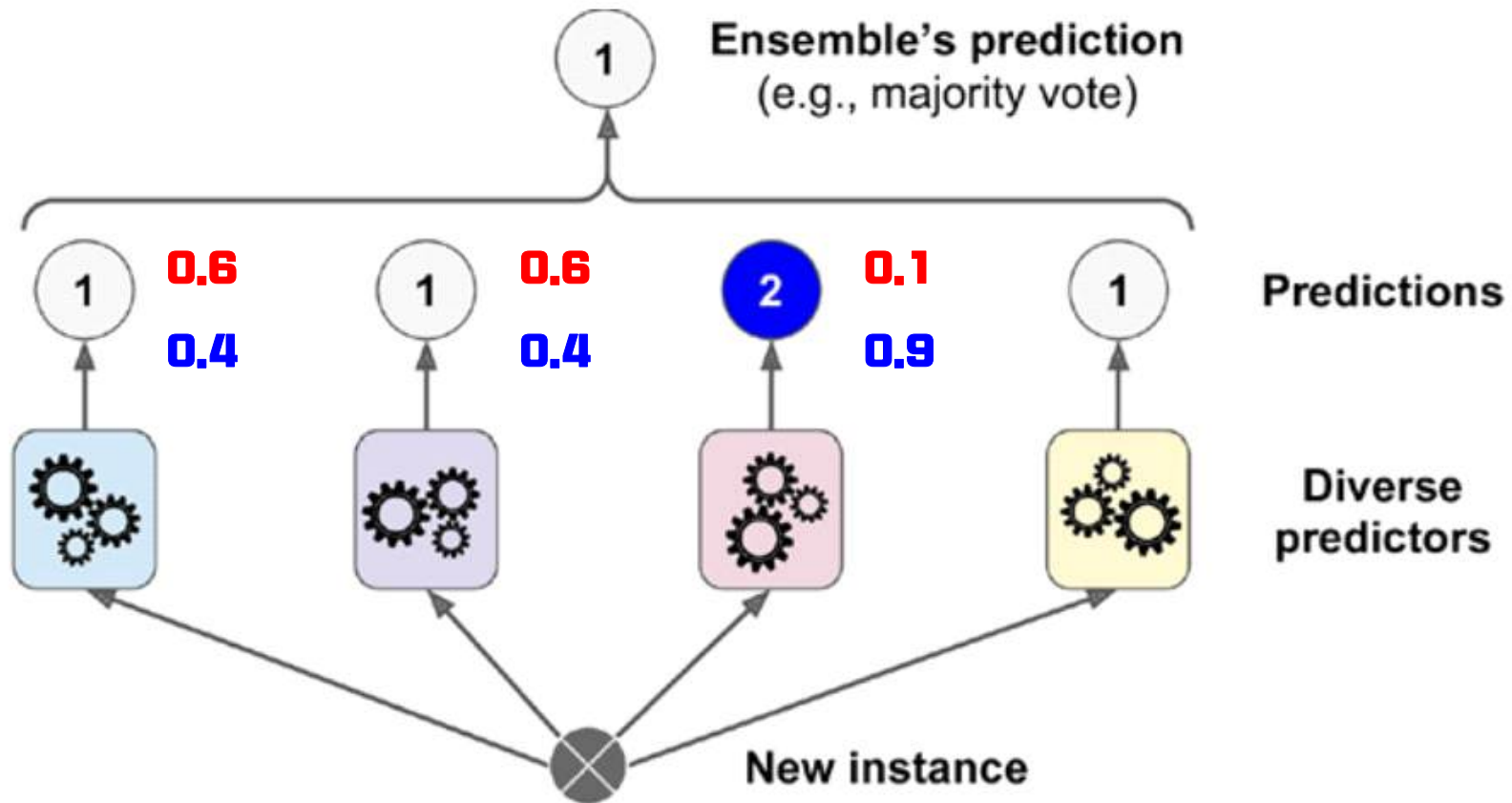
- 평균을 취함
- 여러 모형의 의견을 취함함

Voting
Bagging
Boosting

1. Voting



1. Voting



1. Voting

```
1 #Voting Classifier - voting 파라미터로 hard/soft 선택가능
2 votingC = VotingClassifier(estimators=[('rfc', RFC_best),
3   ('svc', SVMC_best), ('gbc', GBC_best), ('xgb', XGBC_best)], voting='hard', n_jobs=4)
4
5 votingC = votingC.fit(train_data, target)
6
7 #예측 진행
8 prediction = votingC.predict(test_data)
```

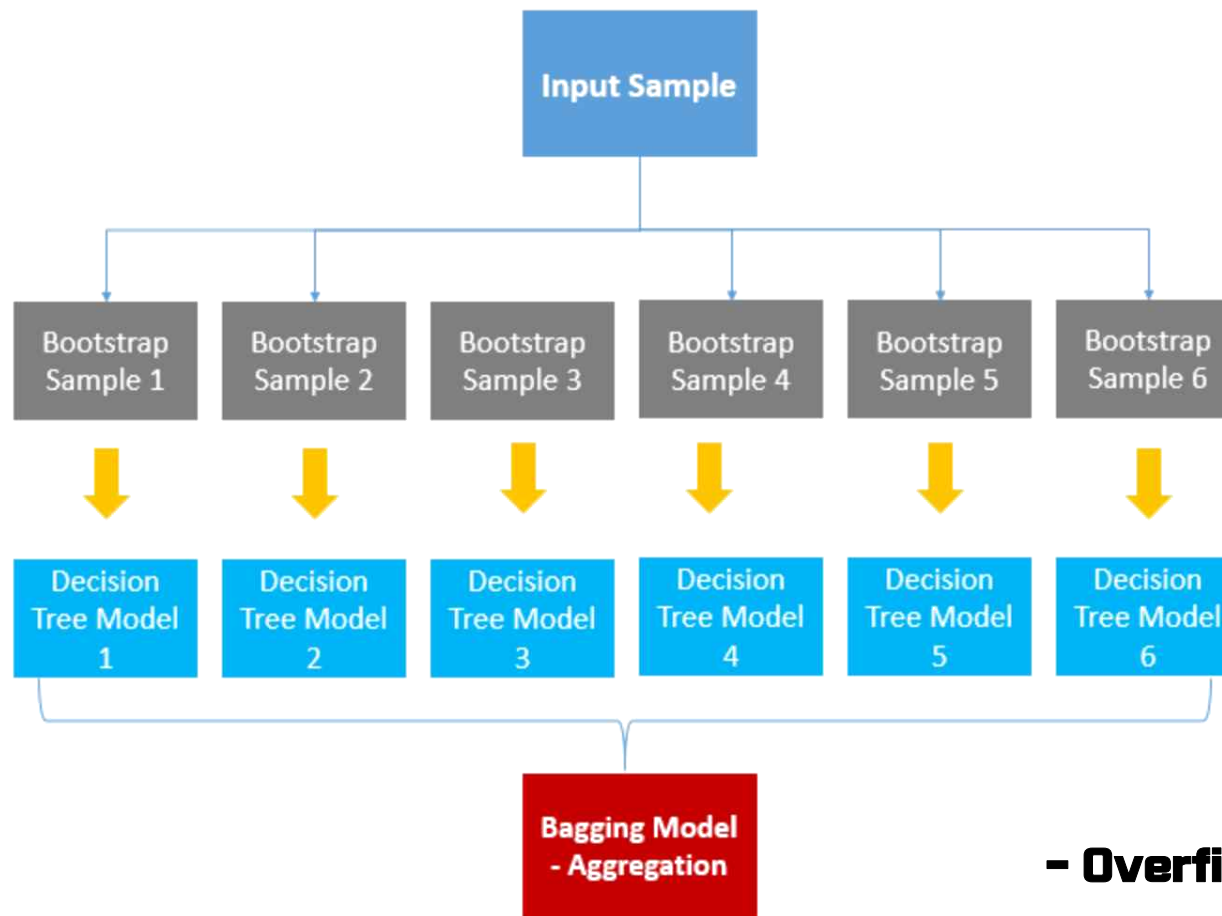
estimators) 개별 모형 목록, 리스트나 named parameter 형식으로 입력

voting) 문자열 {hard, soft} hard voting 과 soft voting 선택. 디폴트는 hard

weights) 사용자 가중치 리스트

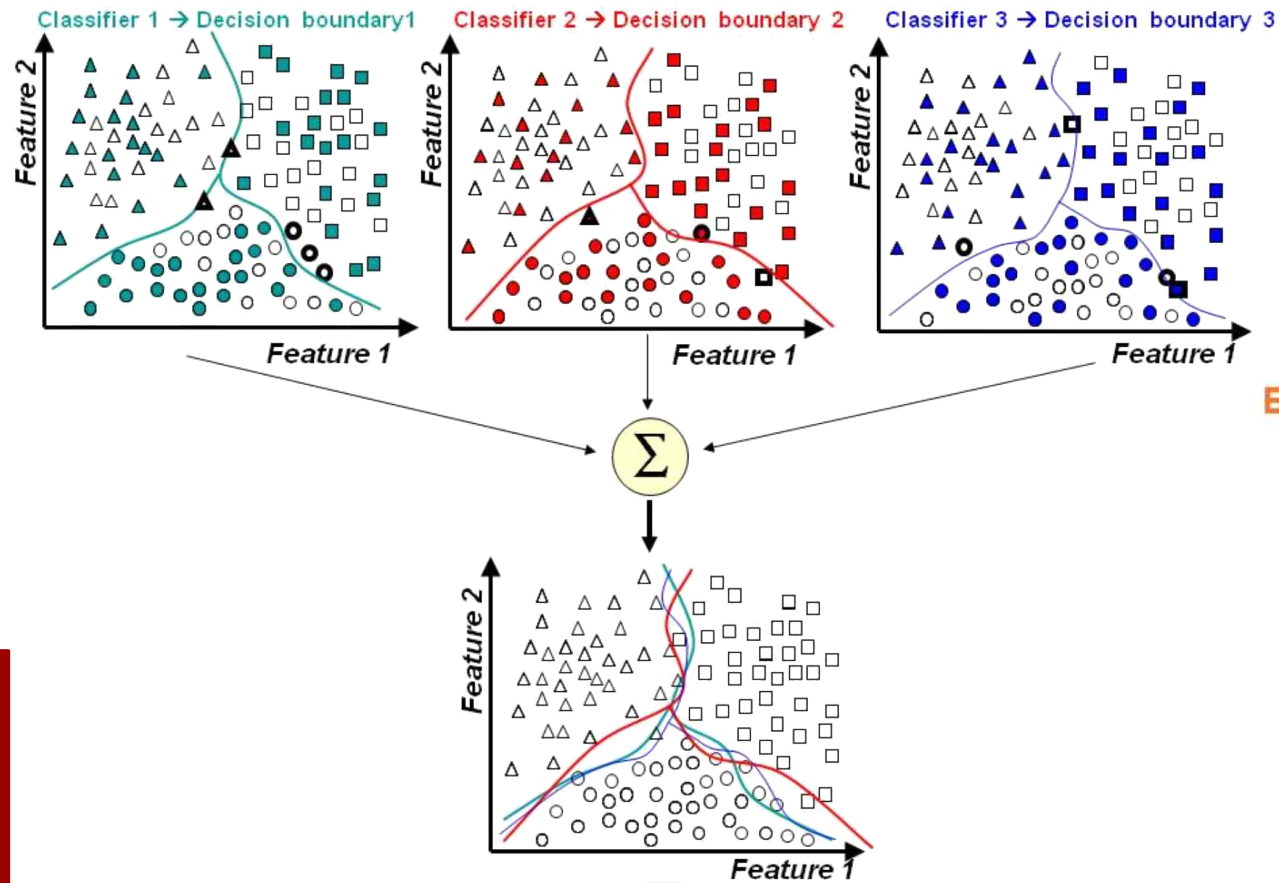
Hard Vote 보다 Soft Vote 방식이 더욱 합리적

2. Bagging (bootstrap aggregating)



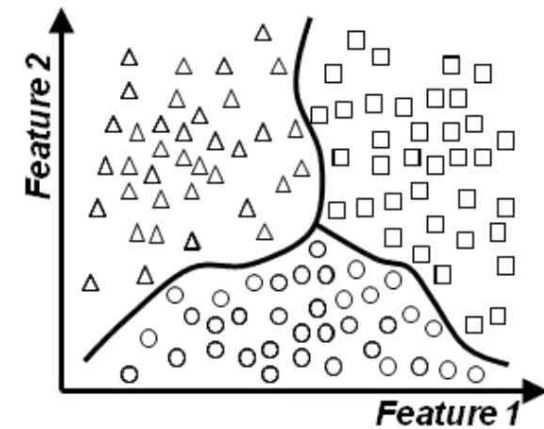
- Overfitting 개선

2. Bagging - Random Forest



**Sampling
(sample, feature)**

Ensemble based decision boundary



2. Bagging - Random Forest

```
In [9]: y = titanic_df['Survived']  
X = titanic_df.drop('Survived', axis = 1)
```

```
In [10]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=10)
```

```
In [11]: rf = RandomForestClassifier(random_state=0)  
rf.fit(X_train, y_train)
```

d:\anaconda3\envs\soojin\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
"10 in version 0.20 to 100 in 0.22.", FutureWarning)

```
Out[11]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',  
                                max_depth=None, max_features='auto', max_leaf_nodes=None,  
                                min_impurity_decrease=0.0, min_impurity_split=None,  
                                min_samples_leaf=1, min_samples_split=2,  
                                min_weight_fraction_leaf=0.0, n_estimators=10,  
                                n_jobs=None, oob_score=False, random_state=0, verbose=0,  
                                warm_start=False)
```

```
In [12]: pred = rf.predict(X_test)  
print("정확도 :{0:.3f}".format(accuracy_score(y_test, pred)))
```

정확도 :0.810

2. Bagging - Random Forest

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
forest = RandomForestClassifier(n_estimators=100, random_state=0)
forest.fit(X_train, y_train)
```

n_estimator : 결정 트리의 개수. default는 10

max_features : 데이터의 feature를 참조할 비율, 개수. default는 'auto'

min_samples_leaf : 리프노드가 되기 위한 최소한의 샘플 데이터 수

max_depth

하이퍼 파라미터를 어떻게 튜닝하느냐에 따라 성능이 크게 달라짐

2. Bagging - Random Forest

```
In [17]: rf_param_grid = {  
    'n_estimators' : [100, 200],  
    'max_depth' : [6, 8, 10, 12],  
    'min_samples_leaf' : [3, 5, 7, 10],  
    'min_samples_split' : [2, 3, 5, 10]  
}
```

```
In [18]: rf_grid = GridSearchCV(rf, param_grid = rf_param_grid, scoring="accuracy", n_jobs= -1, verbose = 1)  
rf_grid.fit(X_train, y_train)
```

```
In [19]: print("최고 평균 정확도 : {0:.4f}".format(rf_grid.best_score_))  
print("최고의 파라미터 : ", rf_grid.best_params_)
```

최고 평균 정확도 : 0.8174

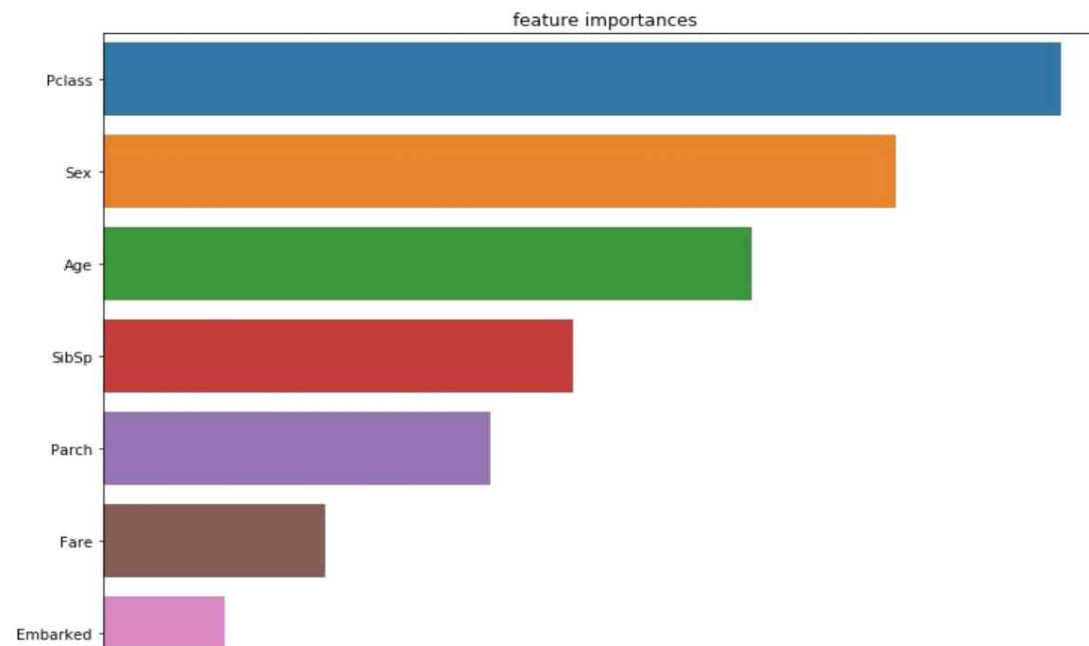
최고의 파라미터 : {'max_depth': 8, 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 100}

2. Bagging - Random Forest

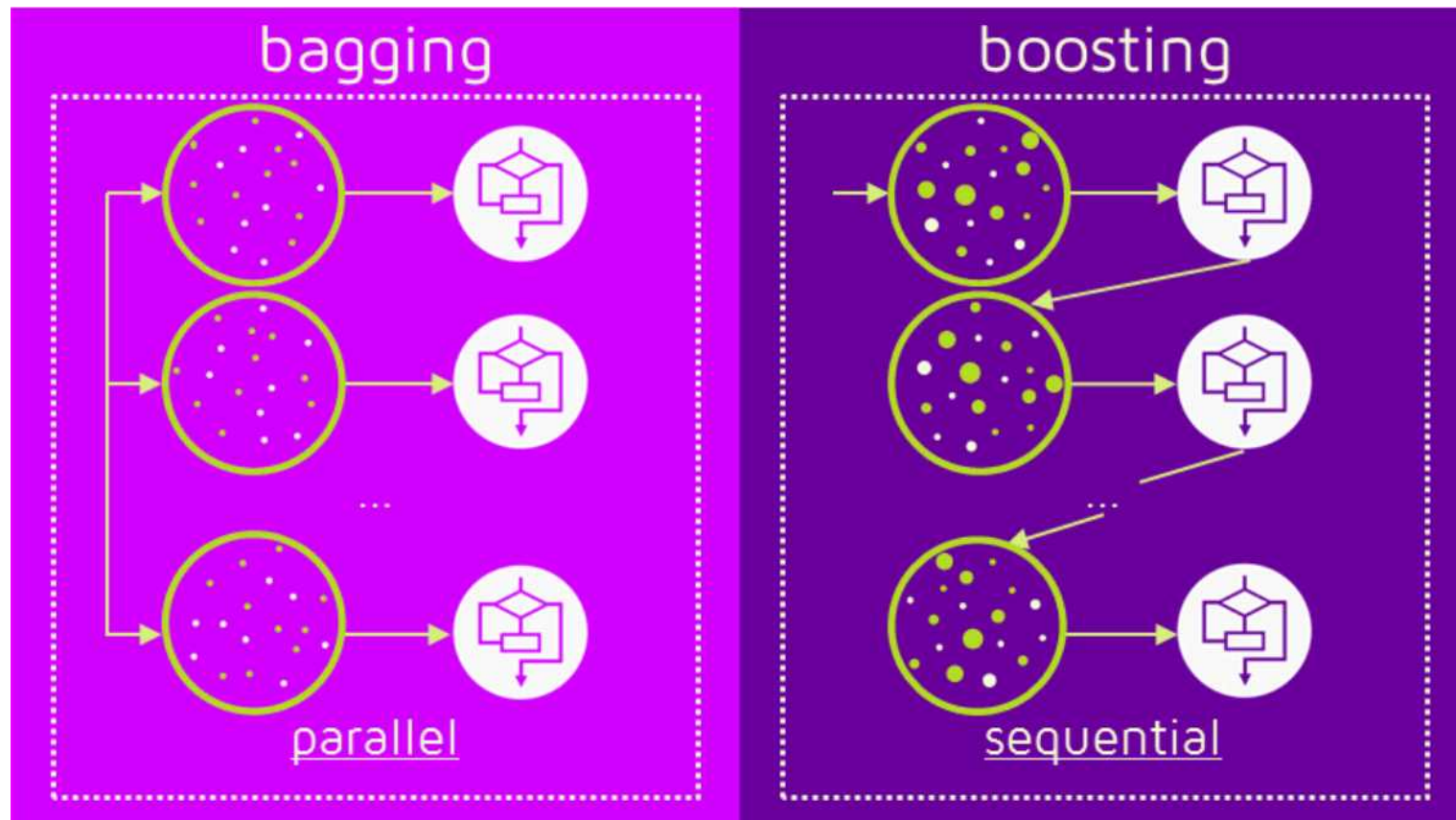
```
In [28]: feature_importances = model.feature_importances_
```

```
In [31]: ft_importances = pd.Series(feature_importances, index = X_train.columns)
ft_importances = ft_importances.sort_values(ascending=False)

plt.figure(figsize=(12, 10))
plt.title("feature importances")
sns.barplot(x=ft_importances, y = X_train.columns)
plt.show()
```



3. Boosting



3. Boosting

Overfitting 문제 개선 - Bagging

단일 모델의 성능 개선 - Boosting / bias 감소에 초점 - 모수 조정이 중요 / outlier에 취약

- **Ada Boost**

데이터셋에서 샘플을 추출하여 여러 분류기에 적용해서 학습시킨다.
시행 결과 잘못 분류된 데이터를 집중적으로 학습하여 다음 fitting에 활용.
Noise나 Outlier가 심한 데이터의 경우, 문제가 발생할 수 있다.

- **Gradient Boost**

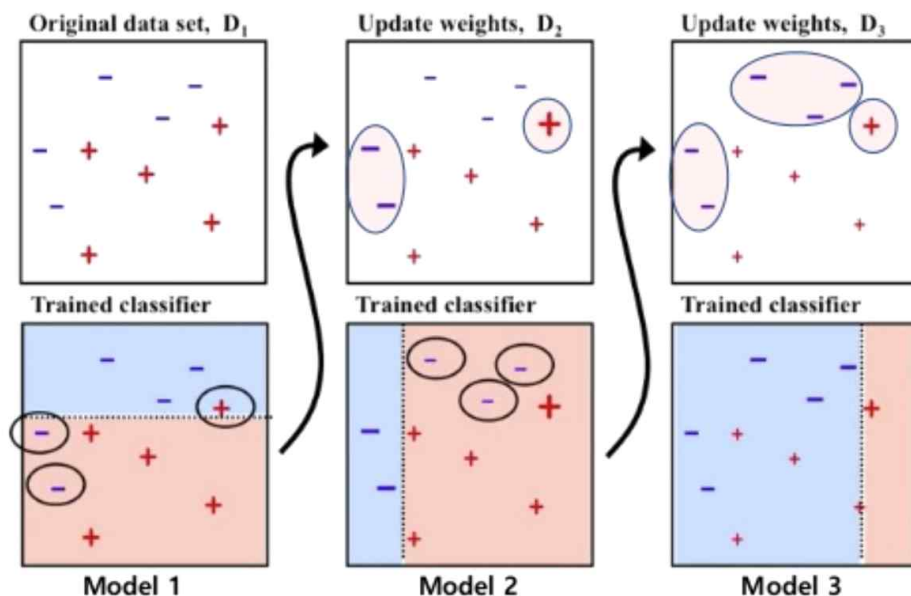
Gradient Descent를 Ada Boost에 적용한 기법.
Outlier, Noise 문제를 해결할 수 있으나, 연산량이 많아짐

- **XG Boost**

Gradient Boost의 많은 연산량을 CPU 분산처리기법 등을 통해 소화하는 방식.

3. Boosting - Ada Boost (Adaptive Boost)

- Model1에서 잘못 예측한 데이터에 가중치를 부여
- Model2는 잘못 예측한 데이터를 분류하는데 더 집중
- Model3는 Model1, 2가 잘못 예측한 데이터를 분류하는데 집중



- Cost Function : 가중치(w)를 반영하여 계산

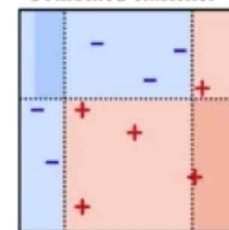
$$s_l(.) = s_{l-1}(.) + c_l \times w_l(.)$$

- 3개의 모델별로 계산된 가중치를 합산하여 최종 모델을 생성

$$.33 * \begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \\ \hline \end{array} + .57 * \begin{array}{|c|} \hline \text{red} \\ \hline \text{blue} \\ \hline \end{array} + .42 * \begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \\ \hline \end{array} \geq 0$$

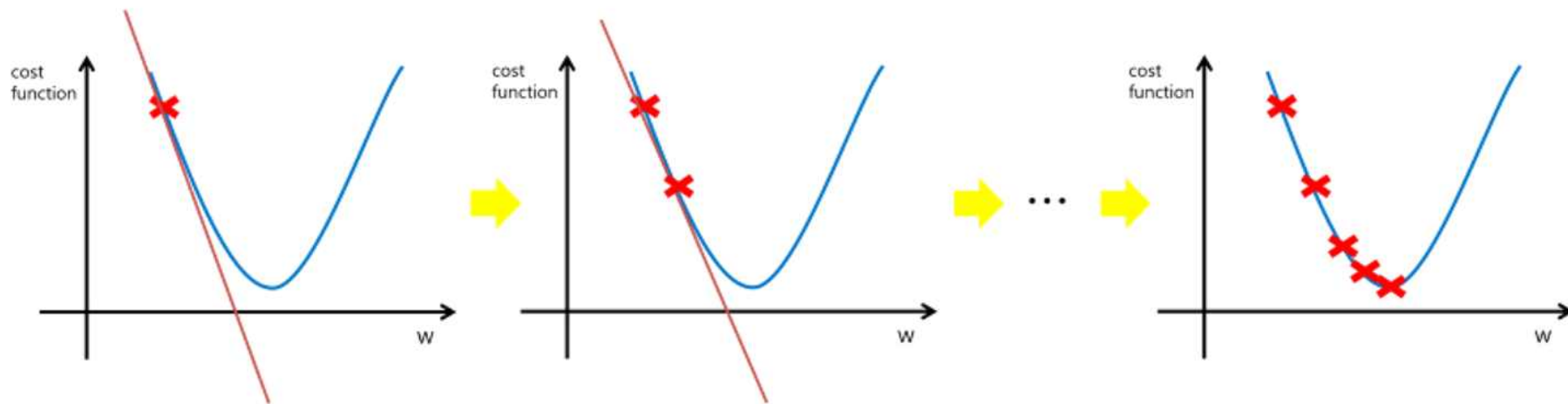


Combined classifier



1-node decision trees
"decision stumps"
very simple classifiers

3. Boosting - Gradient Boost



- AdaBoost과 기본 개념은 동일하고,
- 가중치(D)를 계산하는 방식에서
- Gradient Descent를 이용하여 최적의 파라미터를 찾아낸다.

3. Boosting - XG Boost (Extreme Gradient Boost)

- Gradient Boost 를 기반으로 한다.
- GBM 보다 빠르다. (병렬 처리 추가)
- 과적합 (overfitting) 방지가 가능한 규제가 포함되어 있다.
- 분류와 회귀 모두 가능
- 조기 종료(early stopping) 을 제공한다.

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

- Objective

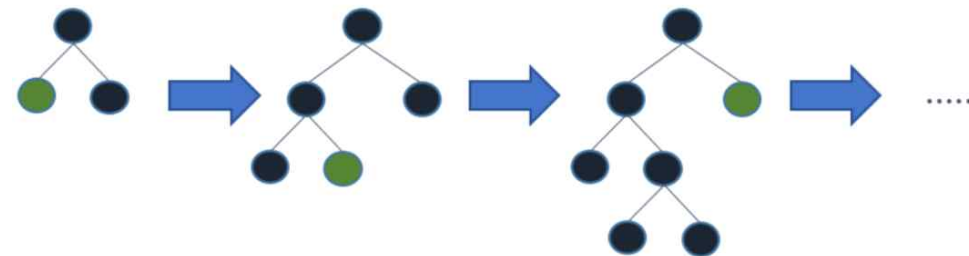
$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

Training loss

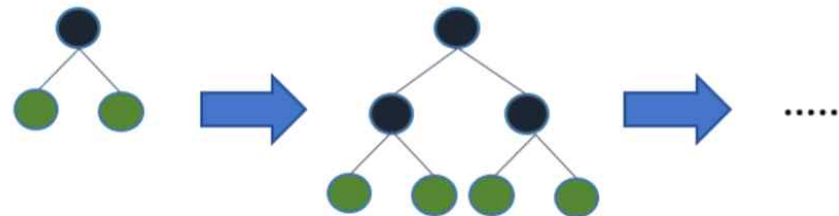
Complexity of the Trees

3. Boosting - Light GBM

- 학습 속도가 느린 XGBoost의 단점을 보완
- 대용량 데이터 처리가 가능
- 타 모델에 비해 적은 메모리 사용
- 적은 수의 데이터 사용시 과적합 문제 발생 가능
- 정보의 손실을 줄일 수 있음



Leaf-wise tree growth



Level-wise tree growth

3. Boosting

```
import lightgbm as lgb
train_ds = lgb.Dataset(X_train, label = y_train)
test_ds = lgb.Dataset(X_val, label = y_val)
params = {'learning_rate': 0.01,
          'max_depth': 16,
          'boosting': 'gbdt',
          'objective': 'regression',
          'metric': 'mse',
          'is_training_metric': True,
          'num_leaves': 144,
          'feature_fraction': 0.9,
          'bagging_fraction': 0.7,
          'bagging_freq': 5,
          'seed': 2020}

model = lgb.train(params, train_ds, 1000, test_ds, verbose_eval=100, early_stopping_rounds=100)
y_pred=model.predict(X_val)
```

```
#training model
cv_model1 = xgb.cv(data = x, label = as.numeric(y)-1, num_class = levels(y) %>% length, # claiming data to use
                  nfold = 5, nrounds = 200, early_stopping_rounds = 150, # about folds and rounds
                  objective = 'multi:softprob', eval_metric = 'mlogloss', # model options
                  verbose = F, prediction = T # do not print messages while training, make prediction
                  )
```

감사합니다