회귀분석과 데이터 모델링

데이터 모델링이란?

- 주어진 데이터에서 사용하고자 하는 x, 알고싶은 값 y이 있을때

  y = f(x) 라는 함수를 통해서 x와 y의 관계를 설명할 수 있다면?

  이때 x를 독립변수, y를 종속변수 라고 합니다
  (혹은 x를 feature, y를 label이라고도 부릅니다)

- y와 x의 관계를 효과적으로 설명하는 f() 함수를 만드는 일을 모델링이라고 합니다

- y 값이 존재하는 경우 supervised learning,
  y 값이 존재하지 않는 경우 unsupervised learning

  y 값이 continuous(연속형) 값인 경우 regression task,
  y 값이 categorical 값인 경우 classification task

데이터 모델링 - Bayes theorem의 관점

- 조건부 확률: 어떤 사건이 일어났다는 '전제 하에' 다른 사건이 일어날 확률
  두 사건 A, B가 있을 때, 사건 A가 일어났을 때 B가 일어날 확률은
  $P(B|A) = P(A \cap B) / P(A)$

  이 때 $P(A|B) = P(A \cap B) / P(B)$이므로
  $P(B|A) = P(A|B)P(B) / P(A)$ 로 표현이 가능합니다

- 데이터 모델링의 관점에서는
  $P(\theta|X) = P(X|\theta)P(\theta) / P(X)$ 로 표현이 가능합니다

- X: 관측된 데이터
  θ: 데이터에 대한 가설, 즉 모델의 parameter값

  P(X): Marginal probability, 데이터 X 자체의 분포
  P(θ): Prior probability, 데이터 관측 이전의(사전) Parameter의 확률 분포
  (보통은 모든 값에 대한 확률이 동일하다고 가정)
  P(X|θ): Likelihood, Parameter가 주어졌을 때 X 데이터가 관측될 확률 분포
  P(θ|X): Posterior probability, Observation (dataset X)가 '주어졌을 때' parameter의 확률 분포

- 궁극적으로 우리의 목적은 P(θ|X)를 최대화 하는 θ를 찾는 것인데,
  P(θ)가 일정하다는 가정 하에서는 P(X|θ), Likelihood를 최대화 하는 θ를 추정(MLE)
  그렇지 않을 때는 P(X|θ)P(θ)를 최대화 하는 θ를 추정(MAP)하는 과정이 데이터 모델링이라고 할 수 있습니다

데이터 모델링 과정

- 데이터를 전처리 및 분석합니다
- 데이터를 training set, test set으로 나눕니다
- training set에 대해서 사용할 모델을 학습시킵니다 (model.fit(training_set))
- test set에 대해서 학습된 모델의 예측값을 통해 모델의 성능을 평가합니다 (실제값과 예측값의 비교를 통해서)
- (모델의 성능이 충분히 쓸만하다고 판단될때) 새로운 데이터에 대해 학습된 모델을 이용해 y 값을 예측합니다 (inference 과정)

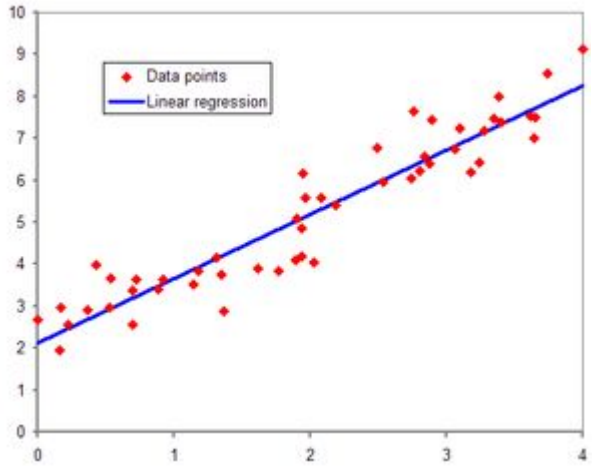
```
from sklearn.datasets import make_regression
X,y = make_regression(n_samples=100, n_features=5, noise=50,
random_state=42)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=33)


lr = LinearRegression(normalize=True)
lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)
print(mean_squared_error(y_test, y_pred))


y_infer = lr.predict(X_infer)
```

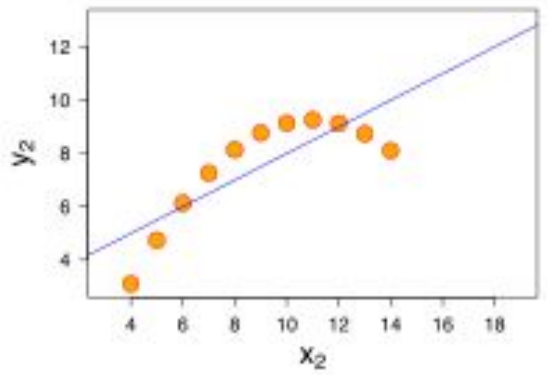
데이터 모델링 - Linear Regression

$$y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i = \boldsymbol{x}_i^T \boldsymbol{\beta} + \epsilon_i$$

- x와 y 간에 '선형의' 관계(y = b + wx)
  가 있다고 가정할 때 주로 사용됩니다

- b + wx = y_{pred} 라는 값에 대해서
  y_{pred} 와 실제 y가 최대한 가까워지는
  w(weight), b(bias) 값을 찾도록 학습됩니다





출처:wikipedia

```
# Linear Regression
from sklearn.linear_model import LinearRegression
lr = LinearRegression(normalize=True)

# training
lr.fit(X_train, y_train)

# LInear regression evaluation
y_pred = lr.predict(X_test)

# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

# Mean Squared Error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))

# R² Score
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))

30.190970764476468
1398.6018192888414
0.9170517679219927
```

데이터 모델링 - Linear Regression의 가정

$$y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i = \boldsymbol{x}_i^T \boldsymbol{\beta} + \epsilon_i$$

Linear regression에서 오차항 ε에 대해 가정하는 5가지

- 오차항의 기대치는 0이다
  $E(\varepsilon_i) = 0$

- 오차항은 일정한 분산을 가진다
  $Var(\varepsilon_i) = \sigma^2$

- i != j일때 ε_i와 ε_j는 서로 독립이다
  $Cov(\varepsilon_i, \varepsilon_j) = 0$ if i != j

- 독립변수 X와 ε_i는 서로 독립이다
  $Cov(X, \varepsilon_i) = 0$

- ε_i는 정규분포 $N(0, \sigma^2)$를 따른다
  $\varepsilon_i \sim N(0, \sigma^2)$

오차항에 대한 가정을 통해 다음과 같은 가정이 가능합니다

- 회귀모형은 선형성을 가진다.
  즉, $E(Y_i) = E(b + wX_i + \varepsilon_i) = b + wX_i + E(\varepsilon_i)$
  이 때, $E(\varepsilon_i)=0$ 이므로 $E(Y_i) = b + wX_i$

- 종속변수 Y의 분산은 일정한 값 σ^2를 가진다
  $Var(Y_i) = Var(b + wX_i + \varepsilon_i) = Var(\varepsilon_i) = \sigma^2$

- ε_i와 ε_j는 서로 독립이며, ε_i는 자기상관성이 없다
  → 시계열 데이터의 경우 자기상관성이 존재할 수 있습니다

- Y는 정규분포를 따른다
  $E(Y_i) = b + wX_i$, $Var(Y_i) = \sigma^2$이므로
  $Y \sim N(b + wX , \sigma^2)$

데이터 모델링 - Linear Regression의 cost function

비용함수(Cost function)

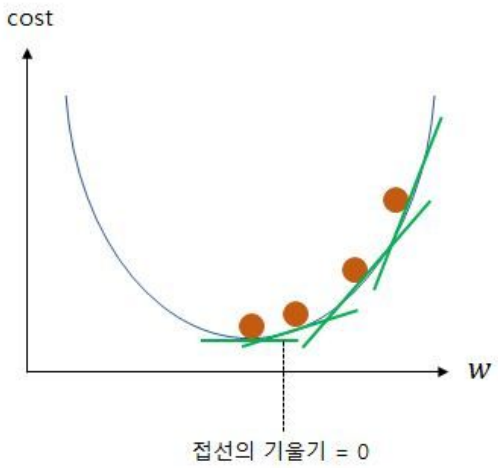
- 실제 Y와 Ypred의 오차를 계산하는 식을 세우고, 이 오차를 최소화하는 w, b를 찾아야 됩니다

- 이 때, 실제값과 예측값의 오차에 대한 식을 손실함수(loss function), 비용함수(cost function), 혹은 목적함수(objective function)이라고 합니다

- 선형회귀에서는 기본적으로 Residual sum of squares(RSS), 혹은 Mean squared error(MSE)를 cost function으로 사용합니다

$$\frac{1}{n}\sum_{i=1}^{n}[y_i - \hat{y}_i]^2$$

- Cost function이 differentiable(미분가능)하고 convex(아래로 볼록)할 때, cost의 최저점은 cost function을 w, b로 각각 편미분 했을 때 0이 되는 지점입니다.



데이터 모델링 - Ordinary Least Squares

Ordinary Least Squares(OLS, 최소제곱법)

- 오차 ε를 최소화하는 w, b를 추정하는 방법

- ε의 제곱의 합이 최소가 되는 점을 찾기 위해서는
  ε의 제곱의 합을 w, b로 표현한 뒤
  w, b로 각각 편미분 했을 때 0이 되면서
  2차 편미분(Jacobian) 값이 Positive definite
  ($\frac{\delta^2 S}{\delta b^2}$>0, |H|>0)이 되도록 해야됩니다(볼록 함수

$$H = \begin{bmatrix} \frac{\delta^2 S}{\delta b^2} & \frac{\delta^2 S}{\delta b w} \\ \frac{\delta^2 S}{\delta w b} & \frac{\delta^2 S}{\delta w^2} \end{bmatrix}$$

$$MinS^2 = Min \sum_{i=1}^{n} \epsilon_i^2 = Min \sum_{i=1}^{n} (y_i - b - wx_i)^2$$

$$\frac{\delta S}{\delta b} = -2 \sum_{i=1}^{n} (y_i - b - wx_i) = 0$$

$$\frac{\delta S}{\delta w} = -2 \sum_{i=1}^{n} x_i (y_i - b - wx_i) = 0$$

를 정리하면

$$w = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

$$b = \bar{y} - w\bar{x}$$

이며 동시에

$$\frac{\delta^2 S}{\delta b^2} = 2n > 0$$

$$|H| = \begin{bmatrix} 2n & 2\sum x_i \\ 2\sum x_i & 2\sum x_i^2 \end{bmatrix} = 4n \sum (x_i - \bar{x})^2 > 0$$

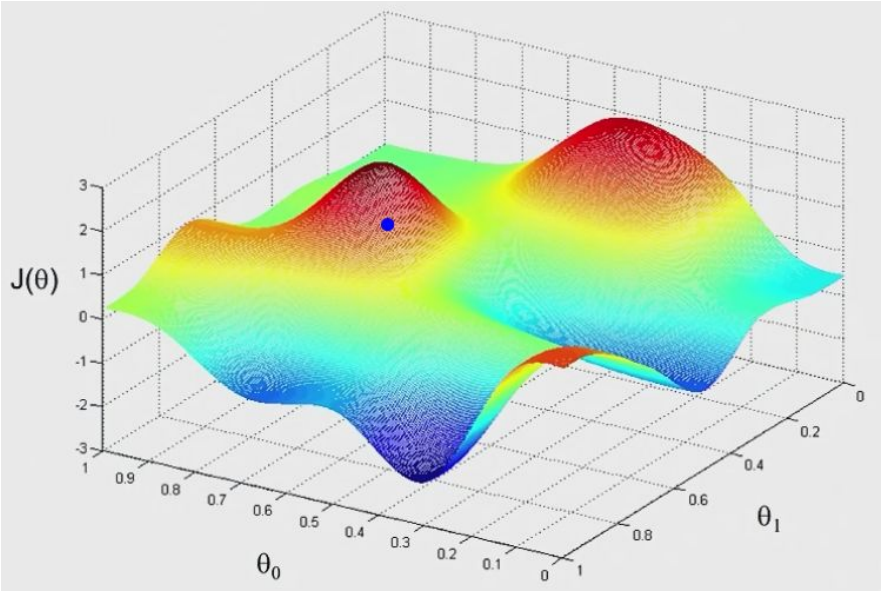
를 만족하므로

오차가 최소화 되는 w, b를 바로 추정할 수 있습니다

데이터 모델링 - Gradient descent

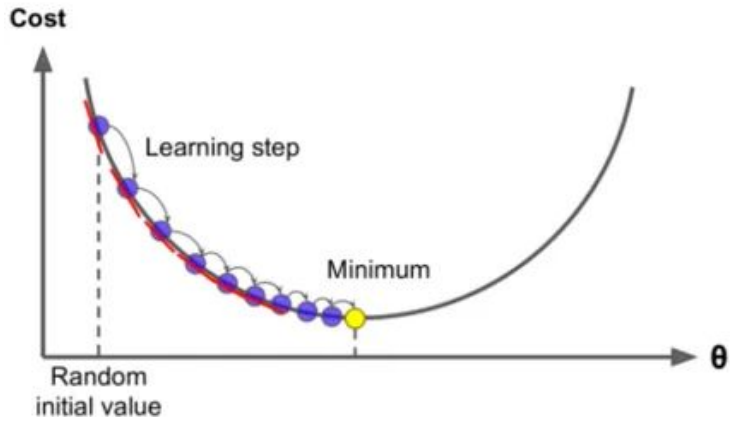
Gradient descent(경사 하강법)

- 변수가 많아지거나 cost function이 복잡해지는 경우,
  최소 제곱법을 통해서 한번에 loss가 최소인 parameter를 추정하기 어렵습니다
- 이러한 경우, 각 parameter에 대해서 cost function을 편미분한 값(gradient)을 learning rate α만큼 여러번 update 해주는 것을 gradient descent라고 합니다



출처:wikipedia

- Gradient descent를 사용하는 경우에는 보통 데이터가 매우 많으므로,
  모든 학습 데이터에 대해서 한번에 gradient descent step을 계산하는 batch gradient descent 보다는
  매 스텝에서 mini-batch 만큼의 데이터를 샘플링 해서 학습하는
  mini-batch stochastic gradient descent(SGD) 방식으로 학습합니다



데이터 모델링 - **Overfitting**

- Overfitting(과적합, high variance): 학습 데이터에 대해서 과하게 fitting되는 경우를 의미합니다
- Generalization(일반화): 학습할 때와 추론할 때의 성능 차이가 많이 나지 않는 경우. 즉, 모델이 여러 inference 상황에서 잘 쓰일 수 있음을 의미
- Overfitting이 발생하는 경우, test set에서의 loss가 증가할 수 있기 때문에 실제 inference 상황에서 모델이 generalization(일반화)되기 어렵게 됩니다
- Overfitting은 보통 데이터에 내재된 복잡도(complexity)보다 모델의 복잡도가 더 과한 경우나, 데이터셋 사이즈가 작은 경우 자주 발생합니다
- Linear regression의 경우, 고려하는 변수가 많아질수록 overfitting이 발생할 가능성이 높아집니다

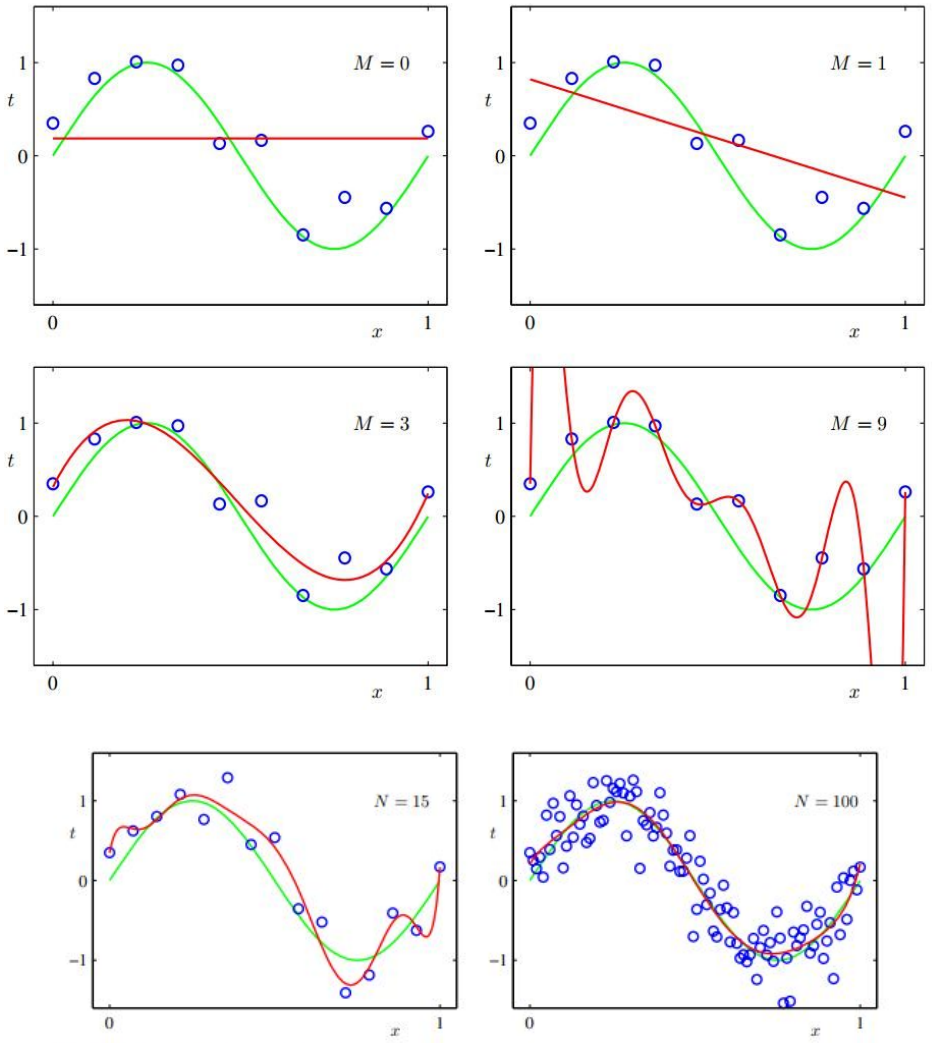


Figure 1.6 Plots of the solutions obtained by minimizing the sum-of-squares error function using the $M = 9$ polynomial for $N = 15$ data points (left plot) and $N = 100$ data points (right plot). We see that increasing the size of the data set reduces the over-fitting problem.

출처:Biship, Pattern recognition and machine learning

데이터 모델링 - **regularization**

- Overfitting을 해결하는 방법은 크게 두 가지 입니다
  - 모델의 복잡도 줄이기:
  모델이 가질 수 있는 parameter를 줄이거나
  (e.g. 딥러닝에서 모델의 사이즈)
  모델이 고려하는 feature 중에서
  상대적으로 중요한 feature 들만 모델의 input으로 사용해볼 수 있습니다

  - 정규화(Regularization):
  모델이 가지는 복잡도를 제한하는 방법으로
  보통 모델의 parameter가 가지는 값의 크기를
  cost function에 추가해주는 방식으로
  parameter 값의 크기를 제한합니다

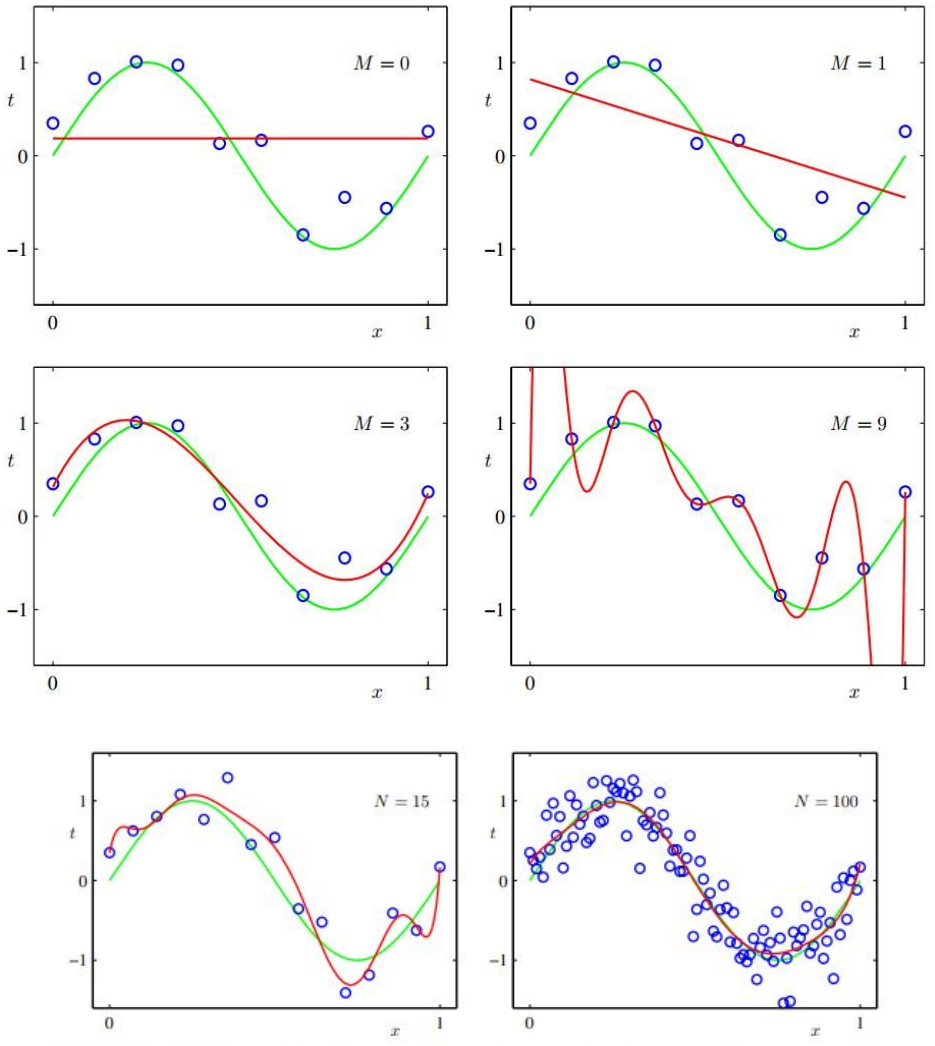


Figure 1.6 Plots of the solutions obtained by minimizing the sum-of-squares error function using the $M = 9$ polynomial for $N = 15$ data points (left plot) and $N = 100$ data points (right plot). We see that increasing the size of the data set reduces the over-fitting problem.

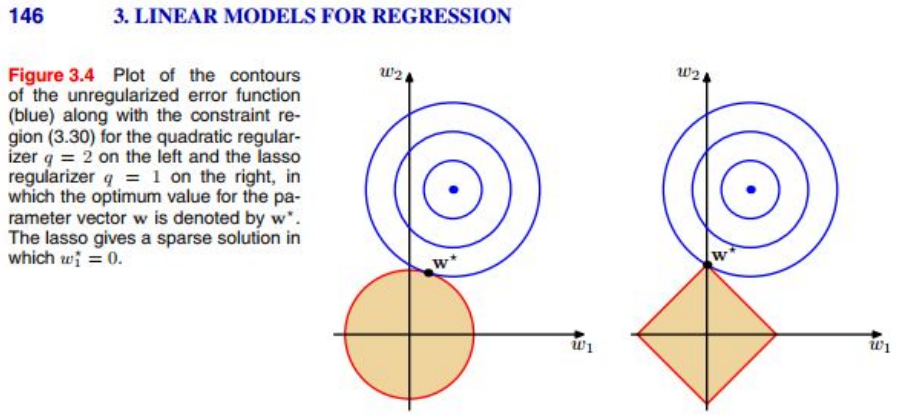
출처:Biship, Pattern recognition and machine learning

데이터 모델링 - **Ridge**

- Ridge regression: 선형 회귀 모델에
  L2 loss를 추가해서 parameter를 정규화 해줍니다

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

- Ridge regression모델의 패널티항은 모델의
  weight parameter들의 제곱합이며,
  lambda hyperparameter를 이용해서
  모델에 주어지는 패널티를 조절합니다



- Parameter가 갖는 원형의 제약 조건 내에서
  cost의 최적점에 가장 가까운 w*를 사용합니다

```python
# Ridge regression
from sklearn.linear_model import Ridge
rlr = Ridge(alpha=1.0)

# training
rlr.fit(X_train, y_train)

# Ridge regression evaluation
y_pred = rlr.predict(X_test)

# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

# Mean Squared Error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))

# R² Score
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))
```

```
30.640326463335143
1425.0070000551837
0.9154857303035218
```

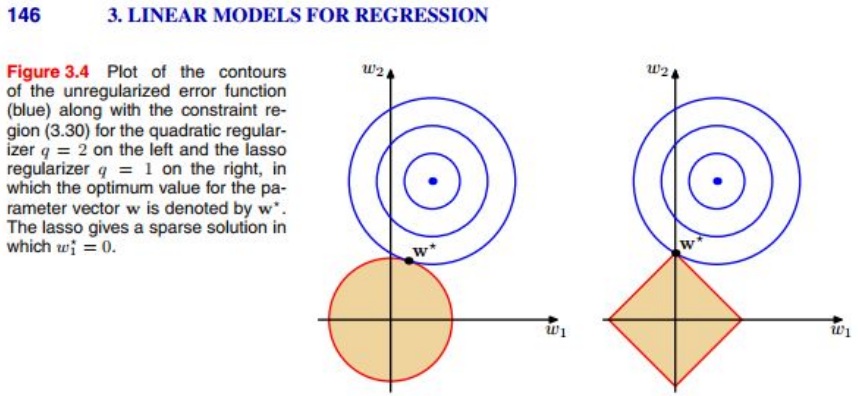
출처:Biship, Pattern recognition and machine learning

데이터 모델링 - Lasso

- Lasso regression: 선형 회귀 모델에
  L1 loss를 추가해서 parameter를 정규화 해줍니다

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

- Lasso regression모델의 패널티항은 모델의
  weight parameter들의 절대값의 합입니다



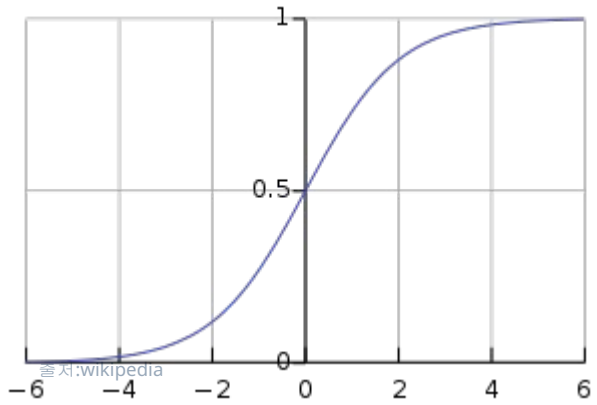
- Parameter가 갖는 사각형의 제약 조건 내에서
  cost의 최적점에 가장 가까운 w*를 사용하며,
  유의미하지 않은 변수들에 대한 계수를 0으로 주어
  중요한 변수를 선택할 때 유용하게 사용
  가능합니다

```python
# Lasso regression
from sklearn.linear_model import Lasso
llr = Lasso(alpha=1.0)

# training
llr.fit(X_train, y_train)

# Lasso regression evaluation
y_pred = llr.predict(X_test)

# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

# Mean Squared Error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))

# R² Score
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))
```

데이터 모델링 - Logistic regression

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

- Linear regression의 output y 값에 'Sigmoid function'을 적용해서 output y의 값이 0~1 사이 값이 되도록 만듭니다

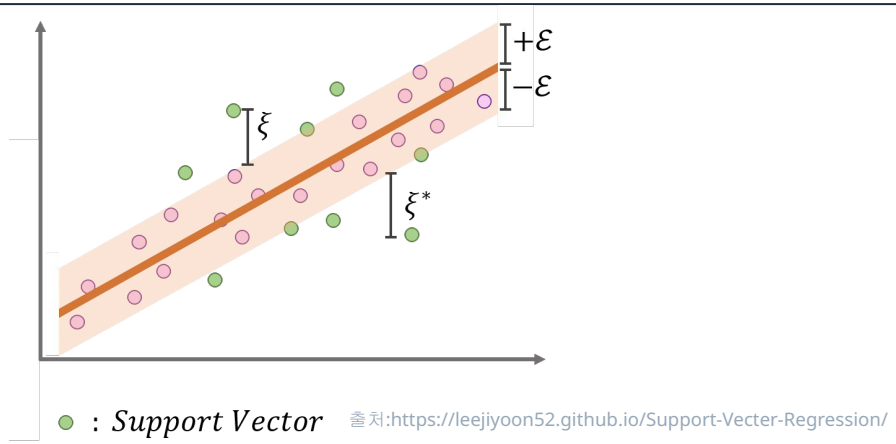


때문에, Classification에 적합화된 Linear regression model이라고 생각하시면 됩니다

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()

# training
lgr.fit(X, y)

# Logistic regression evaluation
y_pred = lgr.predict(X_test)

from sklearn.metrics import accuracy_score
print('accuracy: ', accuracy_score(y_test, y_pred))

from sklearn.metrics import precision_score
print('precision: ', precision_score(y_test, y_pred))

from sklearn.metrics import recall_score
print('recall: ', recall_score(y_test, y_pred))

from sklearn.metrics import f1_score
print('f1: ', f1_score(y_test, y_pred))
```

```
accuracy:  0.96
precision:  1.0
recall:  0.9230769230769231
f1:  0.9600000000000001
```

데이터 모델링 - Support Vecter Machine(SVM)



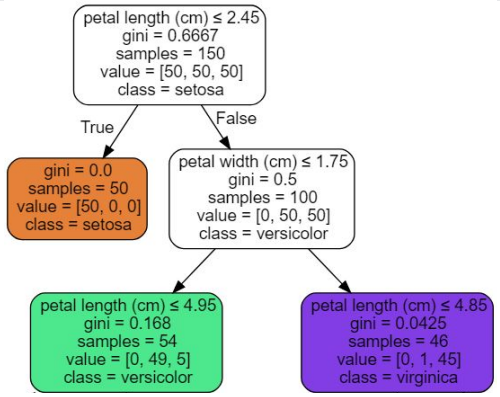
● : *Support Vector* 출처:https://leejiyoon52.github.io/Support-Vecter-Regression/

- Linear regression과 비슷하지만,
  "어느 정도의 허용오차(C) 안에 있는 오차값은 허용해준다" 라고 생각하시면 됩니다
- 허용 오차 C 값을 잘 설정해야 모델이 좋은 성능을 낼 수 있습니다
- kernel= 설정을 통해 모델이 가정하는 x와 y의 관계(선형, 2차 선형, ...) 등을 잘 잡아낼 수 있습니다

```
# Support Vector Machines (SVM)
from sklearn.svm import SVR
svr= SVR(kernel='linear', C=10.)

# training
svr.fit(X_train, y_train)

# SVR evaluation
y_pred = svr.predict(X_test)

# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

# Mean Squared Error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))

# R² Score
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))
```

```
34.98650758106123
1799.2599725700743
0.8932895469496103
```

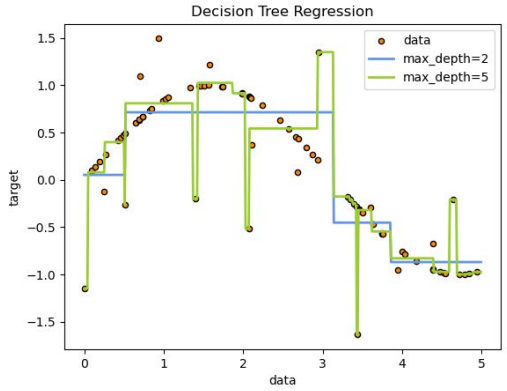
데이터 모델링 - Random Forest



출처:scikit-learn.org

- Decision Tree는 x를 기준삼아,
  "해당 기준을 만족/불만족시 y값이 ~값일것이다"
  라는 조건(x)-결과(y) 를 나무처럼 발전시킵니다

- Classification에서는 y값이 클래스,
  Regression에서는 y값이 평균값

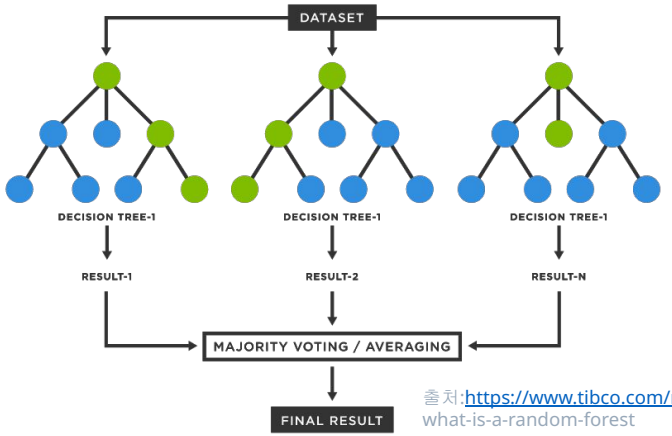


출처:scikit-learn.org

```python
# Random Forest
from sklearn.ensemble import RandomForestRegressor
rfr= RandomForestRegressor(n_estimators=100, max_depth=200)

# training
rfr.fit(X_train, y_train)

# Random forest evaluation
y_pred = rfr.predict(X_test)

# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

# Mean Squared Error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))

# R² Score
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))
```

```
49.41516092197967
3399.6525321095874
0.798373516086659
```

데이터 모델링 - Random Forest



출처:https://www.tibco.com/reference-center/what-is-a-random-forest

- RandomForest는 Decision Tree를 무작위로 여러개 만든 후, 각 tree마다 나온 decision들을 voting(ensemble)을 통해서 최종적인 y 값을 예측합니다

- n_estimators(트리의 개수), max_depth(각 트리의 길이) 등을 설정하여 모델의 성능을 높일 수 있습니다.
다만, 무조건 트리 개수나 길이값이 크다고 성능이 좋아지진 않으며,
학습 시간이나 차지하는 메모리가 지나치게 커질 수 있습니다

```python
# Random Forest
from sklearn.ensemble import RandomForestRegressor
rfr= RandomForestRegressor(n_estimators=100, max_depth=200)

# training
rfr.fit(X_train, y_train)

# Random forest evaluation
y_pred = rfr.predict(X_test)

# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

# Mean Squared Error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))

# R² Score
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))
```

```
49.41516092197967
3399.6525321095874
0.798373516086659
```

데이터 모델링 - Naive Bayes

- y를 우리가 분류하고자 하는 class,
  X는 input feature라고 하면
  $P(y|X) = P(X|y)P(y)/P(X)$
- $X = [x_1, \ldots, x_n]$ 일 때 $P(x_1,\ldots,x_n|y)$는
  $P(x_n|x_{n-1},\ldots,x_1,y)\ldots P(x_2|x_1,y)P(x_1|y)$
  이 때, 각 feature들이 conditional independent면

$$P(x_1, \cdots, x_n|y) = \prod_{i=1}^{n} P(x_i|y)$$ 이므로

- Naive Bayes는 X가 주어졌을 때 feature간의 conditional independent를 가정하고

$$\tilde{y} = h_{NB}(X) = \underset{y}{\operatorname{argmax}}\, p(y) \prod_{i=1}^{n} P(x_i|y)$$ 값으로

```
# Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# training
gnb.fit(X, y)

# Naive Bayes Classifier evaluation
y_pred = gnb.predict(X_test)

from sklearn.metrics import accuracy_score
print('accuracy: ', accuracy_score(y_test, y_pred))

from sklearn.metrics import precision_score
print('precision: ', precision_score(y_test, y_pred))

from sklearn.metrics import recall_score
print('recall: ', recall_score(y_test, y_pred))

from sklearn.metrics import f1_score
print('f1: ', f1_score(y_test, y_pred))

accuracy:  0.8
precision:  0.7222222222222222
recall:  1.0
f1:  0.8387096774193548
```

데이터 모델링 - regression 평가 방법(evaluation)

- 모델을 제대로 평가하는 metric(지표)을 사용해야 주어진 데이터에 가장 효과적인 모델을 사용할 수 있습니다

- Mean Squared Error(MSE):
  오차(틀린값)의 제곱을 평균으로 나눈 값입니다

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

  0에 가까울수록 좋은 성능이라고 볼 수 있습니다

  오차값이 큰 데이터점에 대해서 민감하게 반응합니다 (Outlier에 민감하게 반응)

```python
# Mean Squared Error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))

# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

# R² Score
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))
```

데이터 모델링 - regression 평가 방법(evaluation)

- Mean Absolute Error(MAE):
  오차의 절대값을 평균으로 나눈 값입니다

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

0에 가까울수록 좋은 성능이라고 볼 수 있습니다

오차값이 큰 데이터점에 대해서 상대적으로
덜 민감하게 반응합니다

```
# Mean Squared Error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))

# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

# R² Score
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))
```

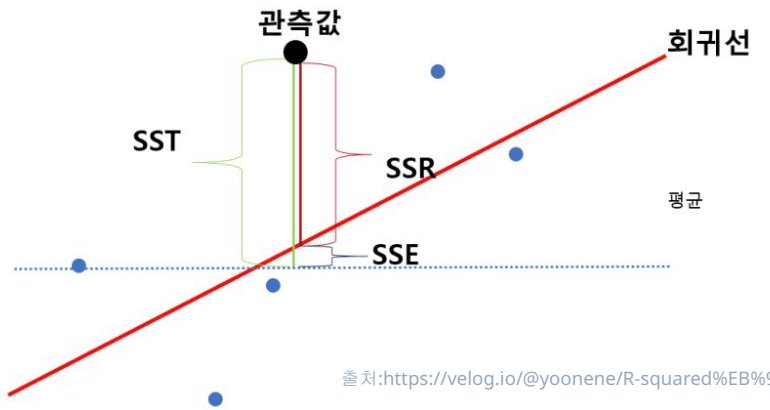
데이터 모델링 - regression 평가 방법(evaluation)

- R-square(결정계수):
  독립변수 x 가 종속변수 y를 얼마나
  잘 설명하는 지를 나타냅니다

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

- 전체 y의 평균으로부터 각 y값이 멀리 떨어질수록,
  예측한 y값과 실제 y값이 가까울수록
  1에 가까워집니다

- 0~1 사이 값을 가지며, 1에 가까울수록
  좋은 성능이라고 볼 수 있습니다



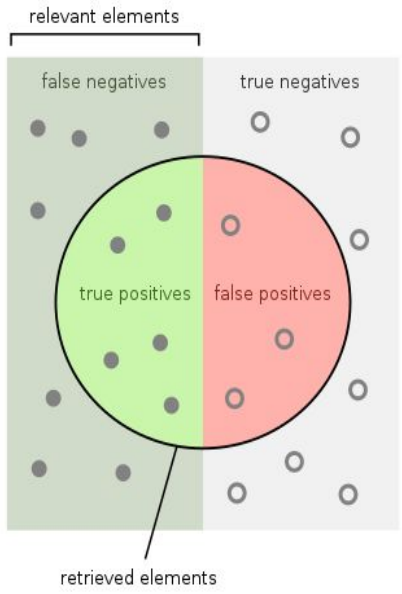
출처:https://velog.io/@yoonene/R-squared%EB%9E%80

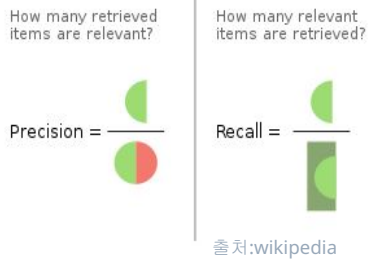
```python
# Mean Squared Error
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))

# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

# R² Score
from sklearn.metrics import r2_score
print(r2_score(y_test, y_pred))
```

데이터 모델링 - Classification 평가 방법(evaluation)

- 모델을 제대로 평가하는 metric(지표)을 사용해야 주어진 데이터에 가장 효과적인 모델을 사용할 수 있습니다

- Precision / Recall





출처:wikipedia

Precision은
"실제로 positive인 샘플중
몇개를 positive라고 잡았는지"
Recall은
"positive라고 잡은 샘플 중
몇개가 실제로 positive인지"

Precision/recall은 positive threshold에 따라 조절이 가능하며,

False positive가 늘어나는 것이 문제인 경우에는 precision를 높이는 것이 중요하지만,
False positive가 늘어나도 어떻게든 positive들을 잡아내야 되는 경우 (e.g. 전염병, 암 진단)
recall 을 높이는 것이 중요합니다

```
from sklearn.metrics import accuracy_score
print('accuracy: ', accuracy_score(y_test, y_pred))

from sklearn.metrics import precision_score
print('precision: ', precision_score(y_test, y_pred))

from sklearn.metrics import recall_score
print('recall: ', recall_score(y_test, y_pred))

from sklearn.metrics import f1_score
print('f1: ', f1_score(y_test, y_pred))
```

```
accuracy:  0.96
precision:  1.0
recall:  0.9230769230769231
f1:  0.9600000000000001
```

데이터 모델링 - Classification 평가 방법(evaluation)

- 모델을 제대로 평가하는 metric(지표)을 사용해야 주어진 데이터에 가장 효과적인 모델을 사용할 수 있습니다

- F1 score

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{2\mathrm{tp}}{2\mathrm{tp} + \mathrm{fp} + \mathrm{fn}}$$

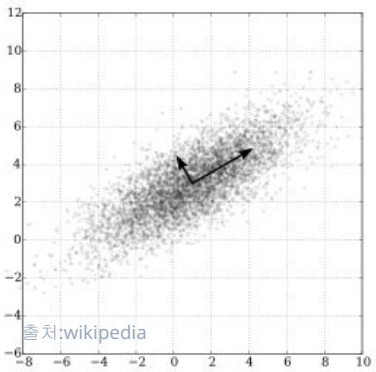
Precision과 recall을 둘다 고려한 metric이라고 볼 수 있습니다

```
from sklearn.metrics import accuracy_score
print('accuracy: ', accuracy_score(y_test, y_pred))

from sklearn.metrics import precision_score
print('precision: ', precision_score(y_test, y_pred))

from sklearn.metrics import recall_score
print('recall: ', recall_score(y_test, y_pred))

from sklearn.metrics import f1_score
print('f1: ', f1_score(y_test, y_pred))
```

```
accuracy:  0.96
precision:  1.0
recall:  0.9230769230769231
f1:  0.9600000000000001
```

데이터 모델링 - Principal component analysis (PCA)

- PCA는 고차원의(feature가 많은) x에 대해서 주어진 x 들의 분포를 가장 잘 설명하는 x축, y축을 찾아내는 기술입니다

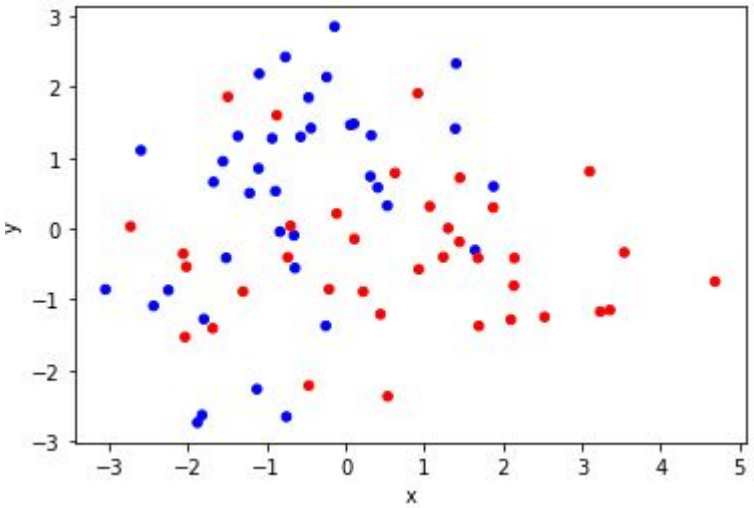


이렇게 x축과 y축을 찾아낸 후, 우리가 가진 데이터를 새로운 x축과 y축에 나타낼 수 있습니다. (특히 feature가 너무 많은 경우)

주로 데이터의 전반적인 분포를 visualization 할때 매우 유용합니다. (Scatter plot 사용)

- 또한, n_components를 2이외의 값으로 주어 모델의 feature 개수를 줄여 효과적인 학습을 할 수도 있습니다

```
# Principal Component Analysis (PCA)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)

pca_model = pca.fit_transform(X_train) #Fit to data, then
transform it

pca_df = pd.DataFrame(columns=['x','y'])
pca_df['x'] = pca_model[:,0]
pca_df['y'] = pca_model[:,1]

pca_df.plot.scatter(x='x', y='y',
c=['b' if z ==0 else 'r' for z in y_train])
```



Feature analysis

- 데이터를 설명하는 feature x와 알아보고자 하는 label y가 있을 때,
  어떠한 feature가 y를 설명하는데에 있어
  중요한 feature 인지 아는 것 또한 매우 중요합니다

- Logistic regression이나 linear regression, Support vector machine, 그리고 random forest 모두
  학습이 된 상태에서 어떠한 feature를 주로 보고있는지를 보여주는 값들을 가지고 있습니다

- 마지막으로, 각 feature들과 label 간의 상관관계 분석을 통해 어떤 feature가 중요한지를 안다면
  의사결정 과정에서 큰도움이 될 수 있습니다
  (e.g. 문제를 일으키는 주된 원인이 무엇인지 파악할 경우)

Feature analysis - Linear regression, Logistic regression

- Regression에서의 Linear regression, Classification에서의 Logistic regression 모두 학습이 된 상태에서 .coef_ 와 .intercept_ 를 조회할 수 있습니다

- .coef_는 각 feature들에 대해서 각각 곱해지는 값, .intercept_ 는 절편을 의미합니다

- .coef_의 절대값이 상대적으로 큰 feature가 있다면, label을 설명하는데 중요한 feature일 수 있습니다

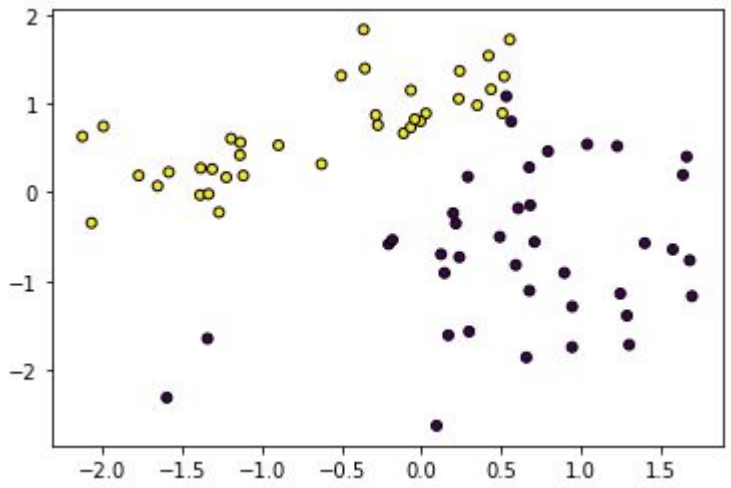
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()

# training
lgr.fit(X_train, y_train)

lgr.coef_
array([[-1.74894748e+00,  2.05970508e+00,  6.90864943e-01,
        -1.13234564e+00,  5.70392800e-01, -1.52202120e-03,
         3.35484430e-01,  2.87431155e-01,  1.24852192e-01,
         6.05941756e-01]])

lgr.intercept_
array([-0.489761])
```

Feature analysis - Linear regression, Logistic regression

- 상대적으로 .coef_ 값이 큰 첫 번째, 두 번째 feature에 대해서 label을 color로 주고 scatter plot을 그려보겠습니다

- 두 변수가 label을 설명하는데 중요한 feature일 수도 있다는 것을 알 수 있습니다

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()

# training
lgr.fit(X_train, y_train)

lgr.coef_
array([[-1.74894748e+00,  2.05970508e+00,  6.90864943e-01,
        -1.13234564e+00,  5.70392800e-01, -1.52202120e-03,
         3.35484430e-01,  2.87431155e-01,  1.24852192e-01,
         6.05941756e-01]])

plt.scatter(X_train[:, 0], X_train[:, 1], marker="o",
c=y_train, s=25, edgecolor="k")
plt.show()
```



Feature analysis - Support vector machine

- Support vector machine 역시 학습이 된 상태에서 .coef_ 와 .intercept_ 를 조회할 수 있습니다

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()

# training
lgr.fit(X_train, y_train)

lgr.coef_
array([[-1.74894748e+00,  2.05970508e+00,  6.90864943e-01,
        -1.13234564e+00,  5.70392800e-01, -1.52202120e-03,
         3.35484430e-01,  2.87431155e-01,  1.24852192e-01,
         6.05941756e-01]])

# Support Vector Machines (SVM)
from sklearn.svm import SVC
svc = SVC(kernel='linear')
# training
svc.fit(X, y)
svc.coef_
array([[-1.10086514,  0.96241273,  0.45270636, -0.46822568,
0.21103224,
         0.24624928,  0.21930329, -0.13029874, -0.22471887,
0.43350556]])
```

Feature analysis - Random forest

- Random forest 모델들에서는 자체적으로 feature importance score를 제공합니다

- .feature_importances_ 를 통해 각 feature의 importance score를 구할 수 있습니다

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()

# training
lgr.fit(X_train, y_train)

lgr.coef_
array([[-1.74894748e+00,  2.05970508e+00,  6.90864943e-01,
        -1.13234564e+00,  5.70392800e-01, -1.52202120e-03,
         3.35484430e-01,  2.87431155e-01,  1.24852192e-01,
         6.05941756e-01]])

# Random forest classifier
from sklearn.ensemble import RandomForestClassifier
rfc= RandomForestClassifier(n_estimators=100, max_depth=200)
# training
rfc.fit(X, y)
rfc.feature_importances_
array([0.25357745, 0.26571728, 0.09979086, 0.17900508,
0.02387162,
       0.02279699, 0.02225427, 0.04759967, 0.02854178, 0.056845
])
```

Feature analysis - 상관관계 분석

- 각 feature들과 label간의 상관관계 분석을 통해 feature의 중요도를 알아볼 수 있습니다

- 상관관계: 통계적 변인과 다른 여러 통계적 변인들이 공변(共變)하는 함수관계

- pearsonr(): 피어슨 상관계수
  두 변수간의 선형 상관관계의 정도를 나타내줍니다.

- spearmanr(): 스피어만 상관계수
  두 변수간의 크기 순서상의 상관관계의 정도를 나타내줍니다

- 두 상관계수 모두 1/-1에 가까울 수록 양/음의 상관관계가 있다고 이야기 할 수 있고,
  같이 나오는 p-value의 경우
  H0: 두 변수는 상관관계가 없다
  라고 하는 가설에 대한 p-value 입니다

```
import scipy.stats as stats

stats.pearsonr(X_train[:,0], y_train)
(-0.6388501021295765, 6.944360183441342e-10)

stats.spearmanr(X_train[:,0], y_train)
SpearmanrResult(correlation=-0.6689214712019207, pvalue=5.406591936684204e-11)
```