UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

www.ust.ac.kr

8장

로지스틱 회귀

이 홍 석 (hsyi@kisti.re.kr)
한국과학기술정보연구원 슈퍼컴퓨팅응용센터

Contents

목차

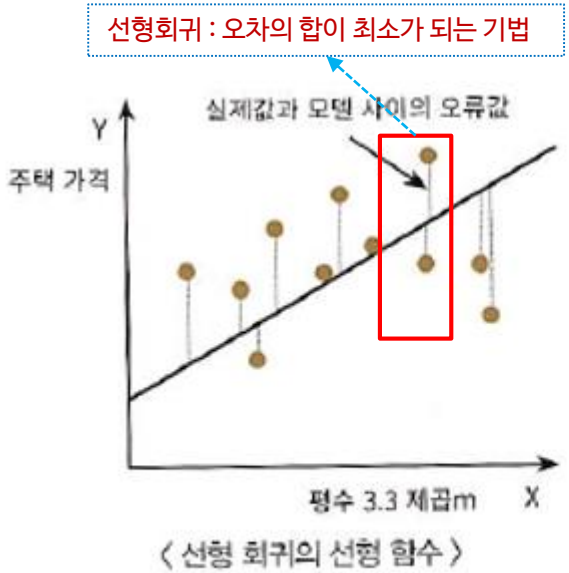
UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

01.
로지스틱 회귀 개요

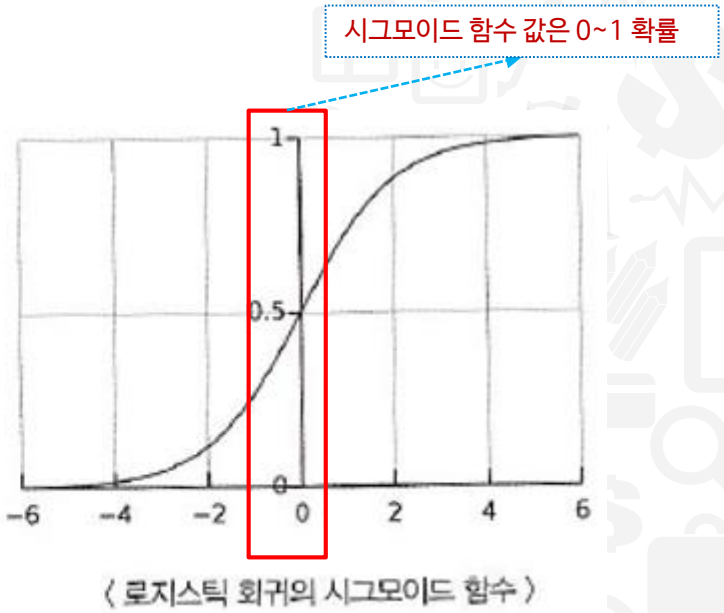
이홍석 (hsyi@kisti.re.kr)

로지스틱 회귀 개요 (1)

- 로지스틱 회귀
  - 선형회귀 방정식을 분류에 적용한 알고리즘, 분류에 사용
  - 시그모이드 함수의 최적점을 찾고, 이 시그모이드 함수의 반환값을 확률로 간주해 확률에 따라 분류를 결정함



〈 선형 회귀의 선형 함수 〉



〈 로지스틱 회귀의 시그모이드 함수 〉

로지스틱 회귀 개요 (2)

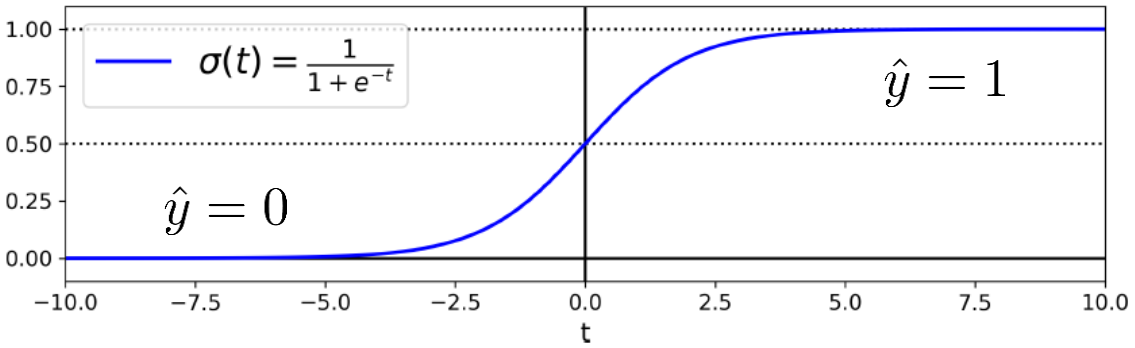
- 로지스틱(Logistic, 혹은 Logit) 회귀
  - 회귀(Regression)은 분류에도 사용 가능함
  - 샘플이 특정 클래스에 속할 확률을 추정하는데 널리 사용
  - 입력 특성의 가중치의 합을 계산하는데, 선형회귀처럼 결과를 바로 출력하지 않음
  - 대신, 결과를 로지스틱으로 출력함

로지스틱 회귀 모델의 확률

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\mathbf{x}^T \boldsymbol{\theta}\right)$$

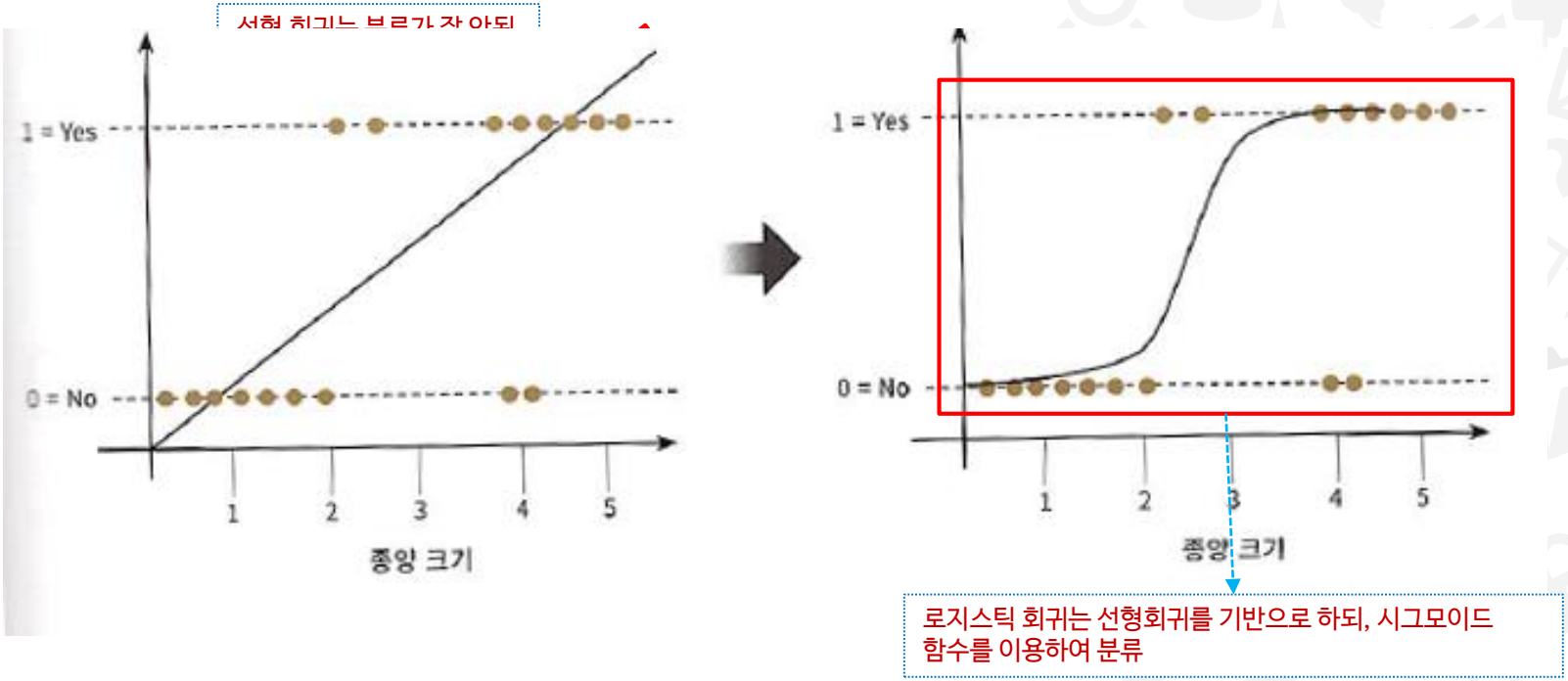
시그모이드(Sigmoid) 함수

$$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5 \\ 1 \text{ if } \hat{p} \geq 0.5 \end{cases}$$



로지스틱 회귀 개요 (3)

- 회귀를 분류로 사용시 예제 데이터
  - 종양의 크기에 따라서 악성종양(Yes), 그렇지 않은지 (No)
  - 회귀를 이용하여 0~1를 예측하시오.



로지스틱 모형 훈련과 비용함수

- 훈련은 어떻게 시킬까
  - 앞의 그래프에서 ‘t’가 양성 샘플에서는 높은 확률로 추정하고,
  - 음성 샘플에서는 낮은 확률을 추정하는 모델 파라미터 벡터(θ)를 찾는 것
  - 한 개의 훈련에 대한 비용함수는

$$c(\boldsymbol{\theta}) = \begin{cases} -\log\left(\hat{p}\right) & \text{if } y = 1 \\ -\log\left(1 - \hat{p}\right) & \text{if } y = 0 \end{cases}$$

  - Logistic 회귀의 비용함수와 편미분 함수

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \hat{p}^{(i)}\right)\right]$$

$$\frac{\partial}{\partial\theta_j}\mathrm{J}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

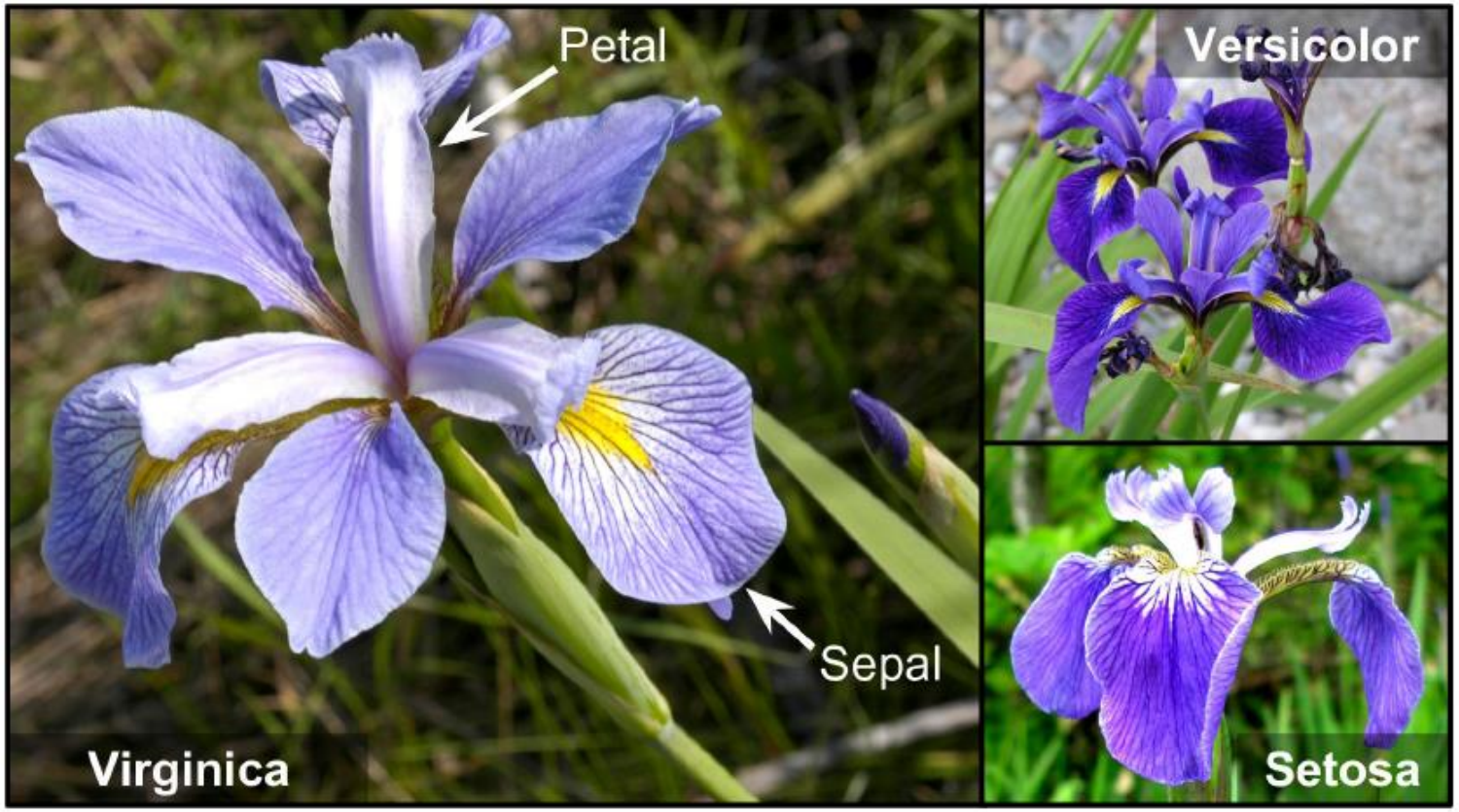
UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

02. 실습
붓꽃 데이터와 소프트 맥스

이홍석 (hsyi@kisti.re.kr)

결정 경계(Decision Boundaries)

숙제: 3종류의 붓꽃 각 150개 중에서 Virginica 종을 감지하는 분류기를 만들어라!



결정 경계(Decision Boundaries)

```
from sklearn import datasets
iris = datasets.load_iris()
list(iris.keys())
```

```
['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename']
```

```
#print(iris.DESCR)
```

```
X = iris["data"][:, 3:]  # petal width
y = (iris["target"] == 2).astype(np.int)  # 1 if Iris virginica, else 0
```

꽃잎의 넓이

Virginica 이면 1, 아니면 0

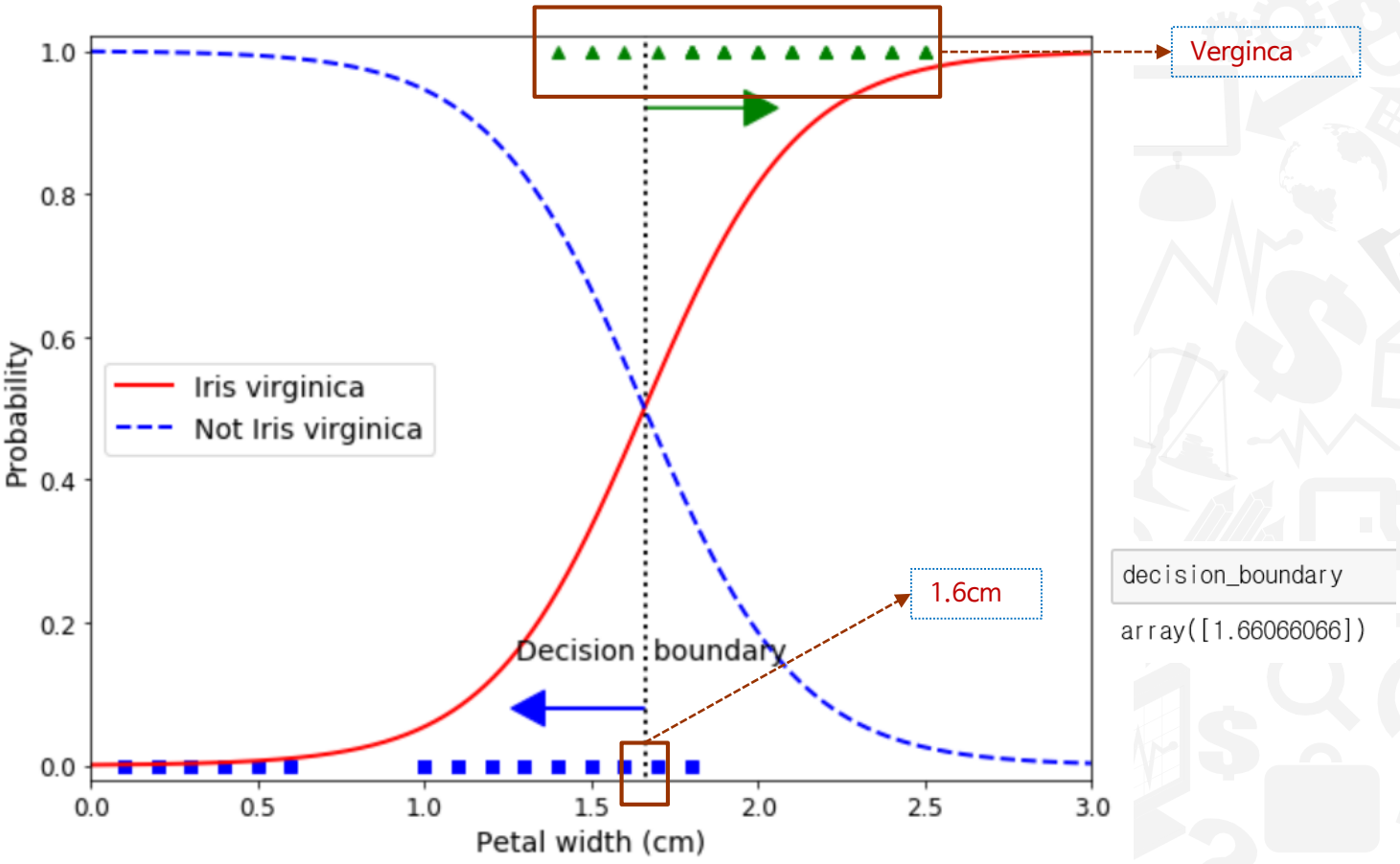
Note: To be future-proof we set `solver="lbfgs"` since this will be the default value in Scikit-Learn 0.22.

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver="lbfgs", random_state=42)
log_reg.fit(X, y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

로지스틱 회귀 : 붓꽃 분류기

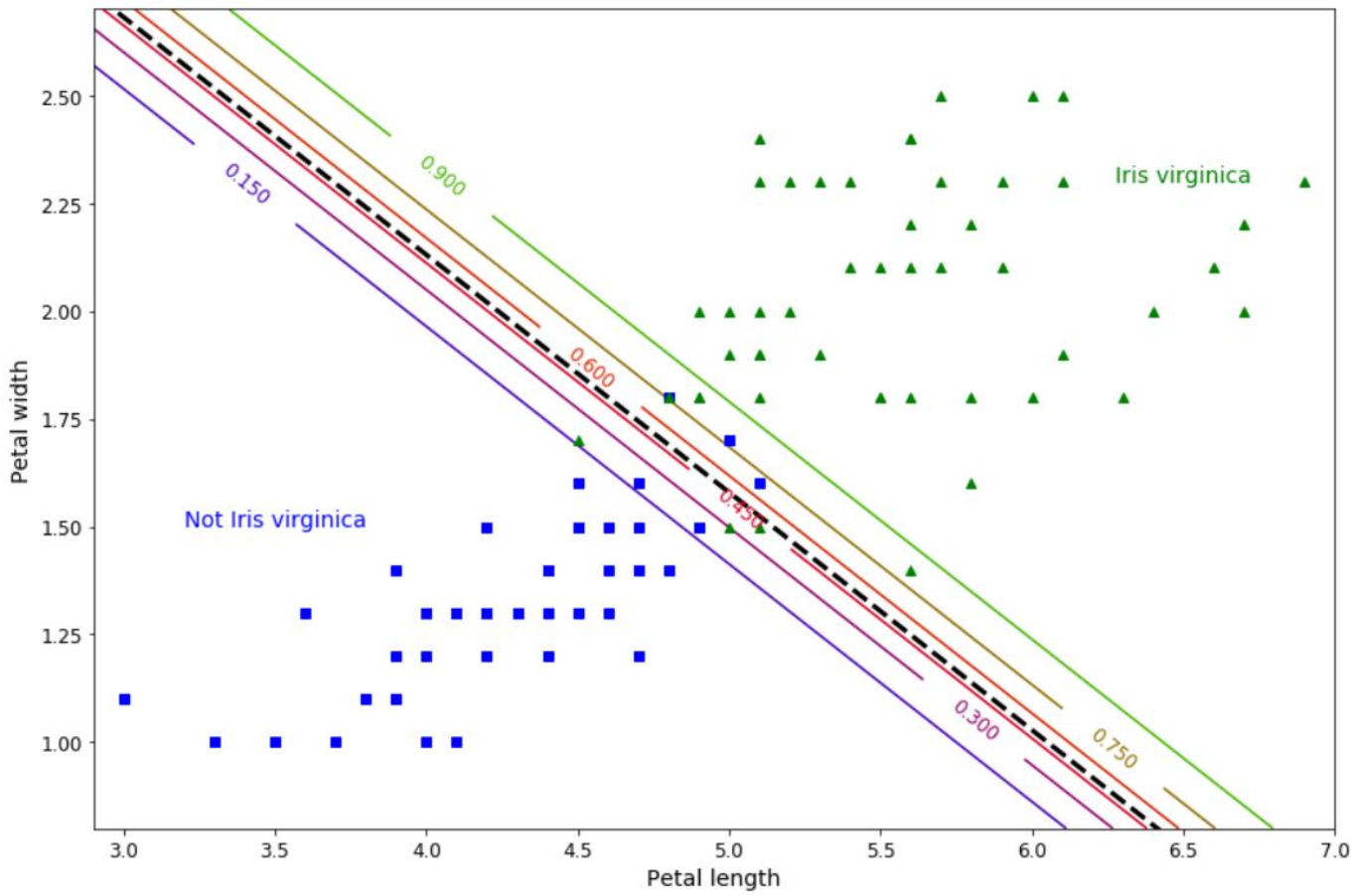
꽃잎 너비가 0~3인 꽃에 대한 분류




```
decision_boundary
array([1.66066066])
```

로지스틱 회귀 : 붓꽃 분류기

선형 결정 경계



소프트맥스

- Softmax는 다중분류기
  - 로지스틱은 이중분류기

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

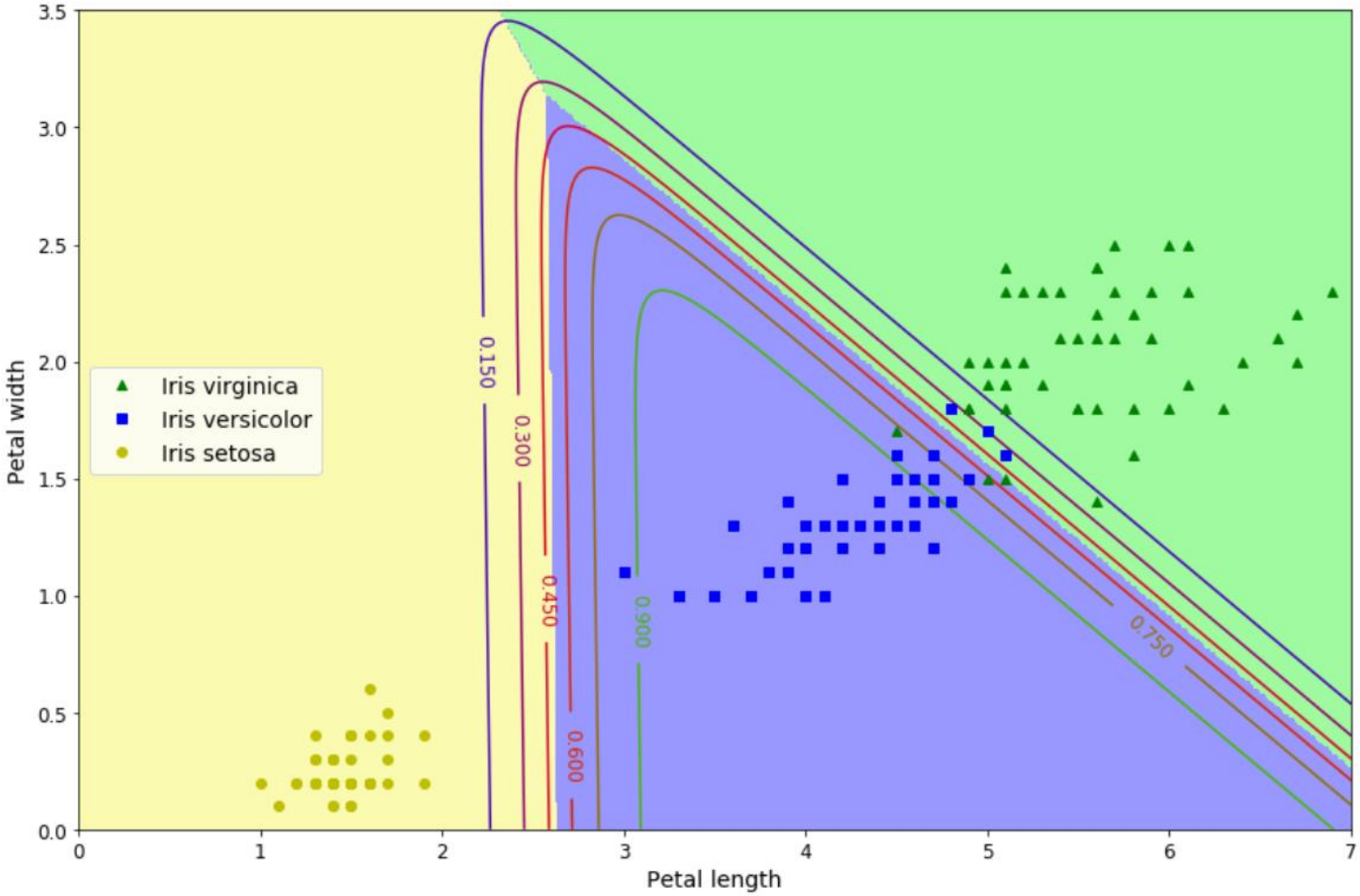
$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\boldsymbol{\theta}^{(k)}\right)^T \mathbf{x}\right)$$

- 크로스 엔트로피 비용 함수와 벡터 편미분 함수

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

$$\nabla_{\boldsymbol{\theta}^{(k)}} J(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$$

Softmax로 IRIS 분류기



UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

03. 실습
유방암 예측 데이터 적용

이홍석 (hsyi@kisti.re.kr)

유방암 데이터 로지스틱회귀(1)

실습1. 로지스틱 회귀

위스콘신 유방암 데이터

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression

cancer = load_breast_cancer()
```

```
cancer
```

```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01
         8.902e-02],
```

```
cancer.keys()
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

```
print(cancer['DESCR'])
print(cancer['target_names'])
```

```
    - Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign
```

```
print(cancer['feature_names'])
print(cancer['data'])
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
```

유방암 데이터 로지스틱회귀(2)

```
cancer['data'].shape
```

```
(569, 30)
```

```
df = pd.DataFrame(np.c_[cancer['data'], cancer['target']], columns = np.append(cancer['feature_names'], ['target']))
```

```
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20

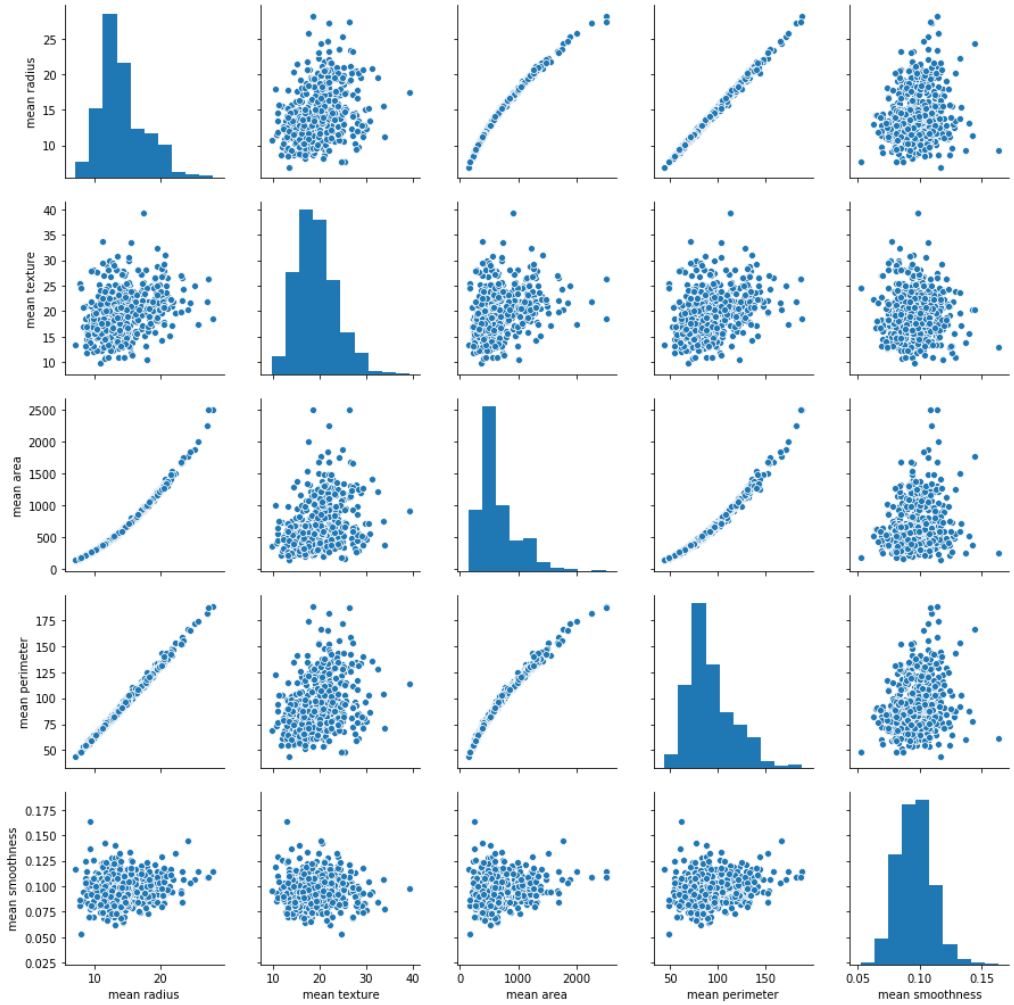
5 rows × 31 columns

```
df.tail()
```

mean ctness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060	0.07115	0.0

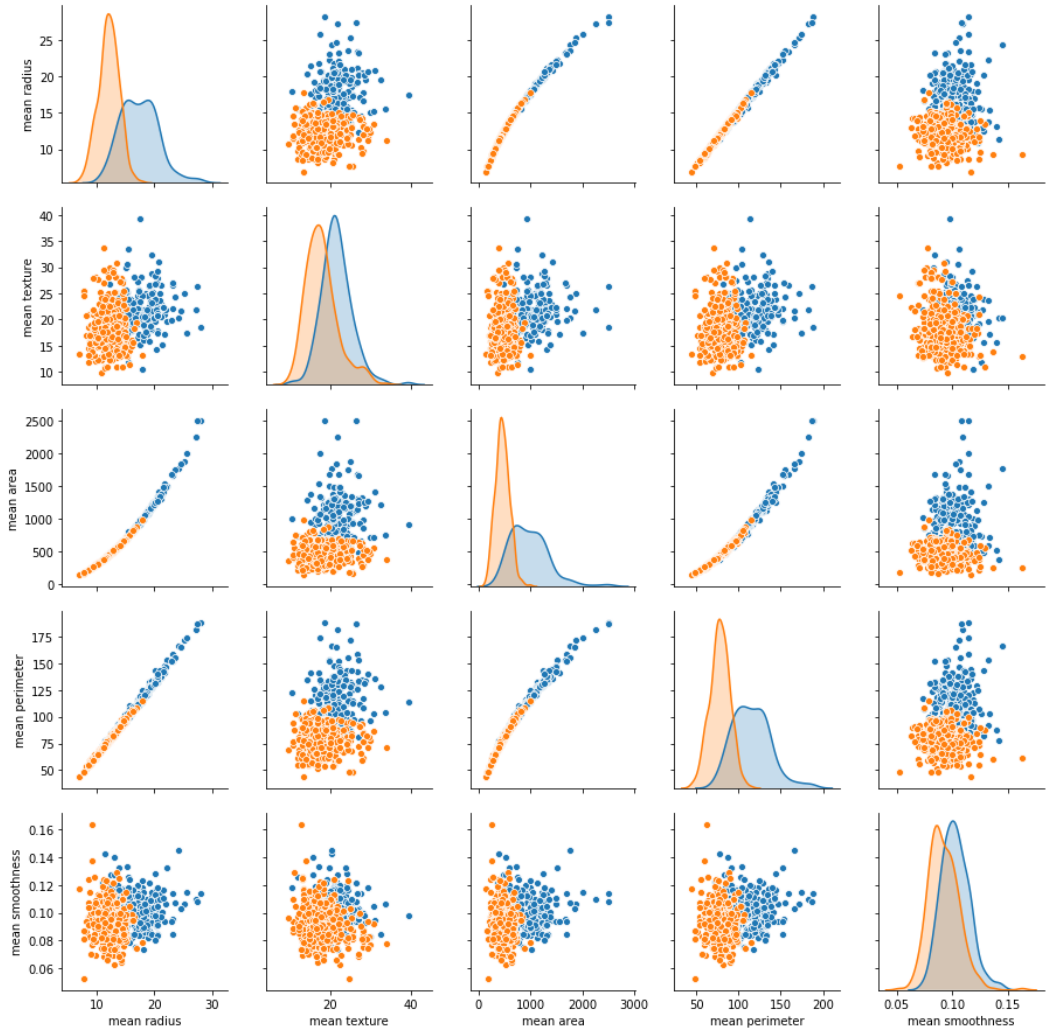
유방암 데이터 로지스틱회귀(3)

```
sns.pairplot(df, vars = ['mean radius', 'mean texture', 'mean area', 'mean perimeter', 'mean smoothness'] )
```



유방암 데이터 로지스틱회귀(4)

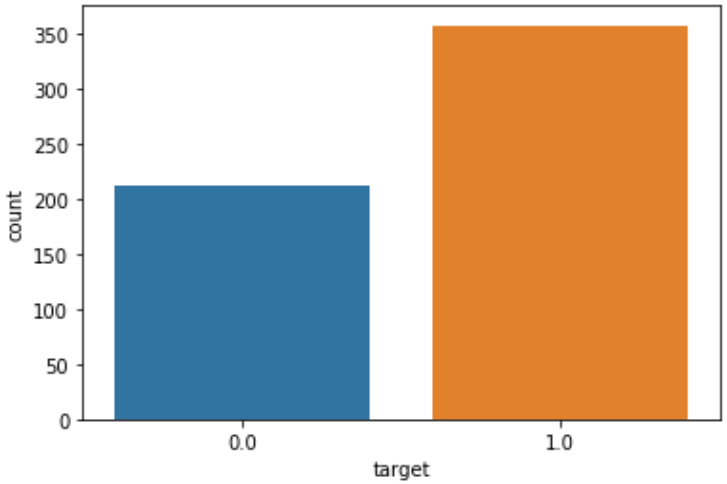
```
sns.pairplot(df, hue = 'target', vars = ['mean radius', 'mean texture', 'mean area', 'mean perimeter', 'mean smoothness'] )
```



유방암 데이터 로지스틱회귀(5)

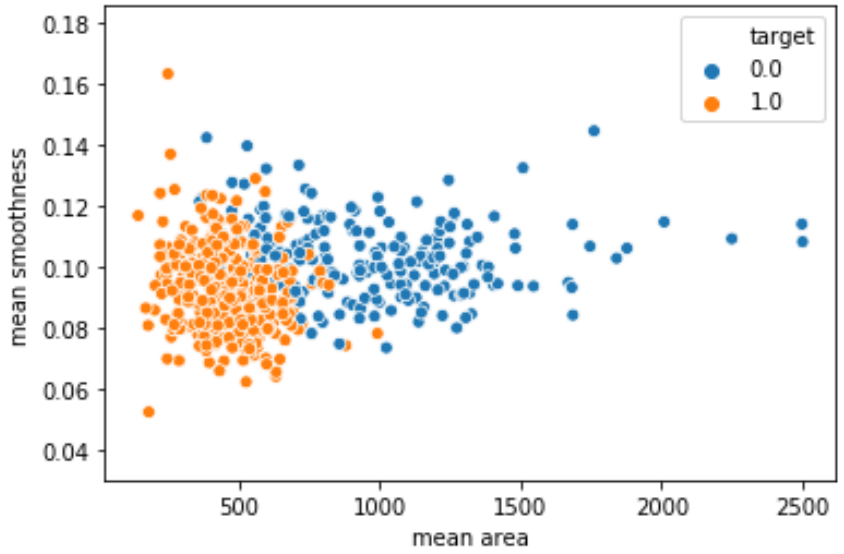
```
sns.countplot(df['target'], label = "Count")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x195c7113be0>
```




```
sns.scatterplot(x = 'mean area', y = 'mean smoothness', h
```

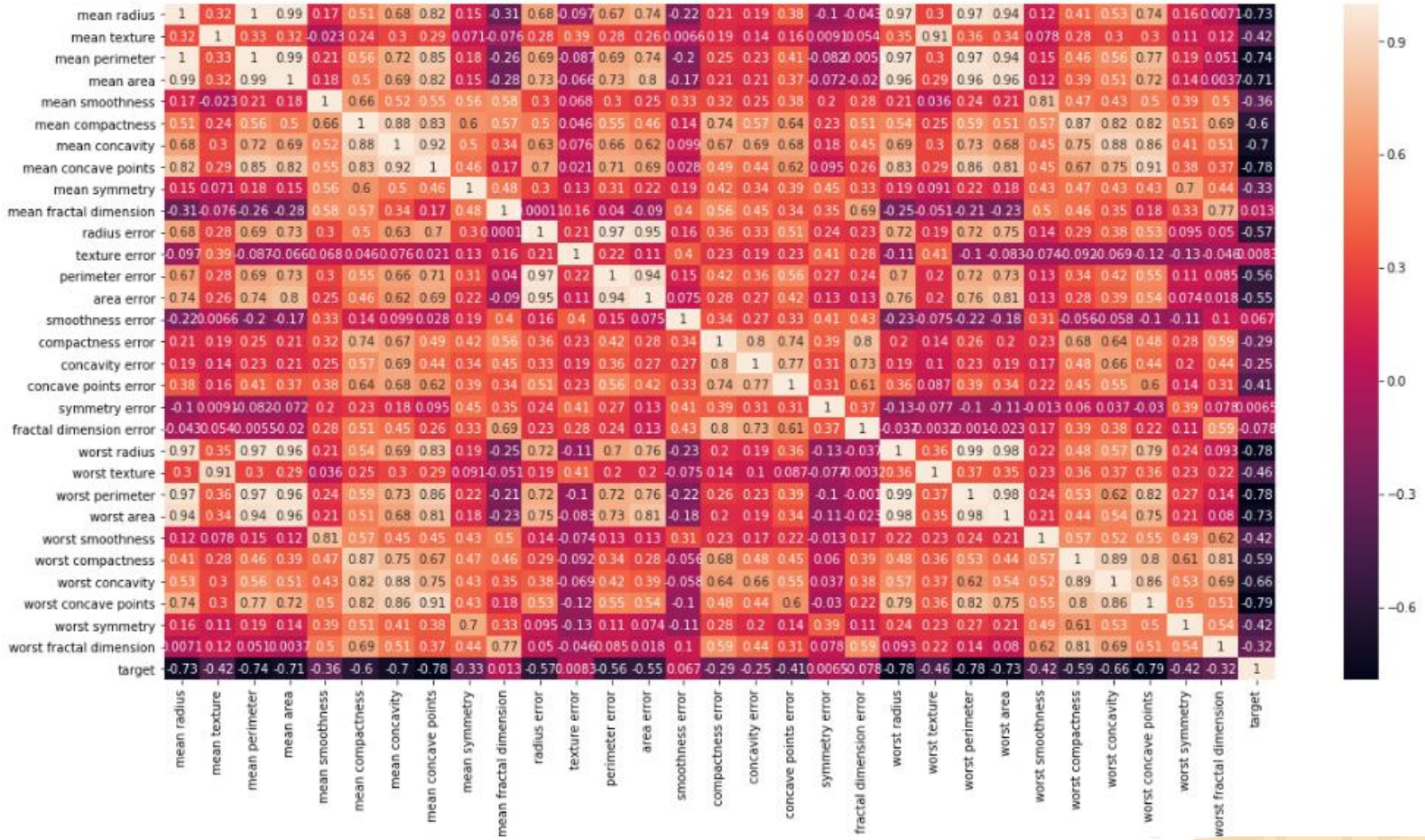
```
<matplotlib.axes._subplots.AxesSubplot at 0x195cb4de9e8>
```



유방암 데이터 로지스틱회귀(6)

```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x195ca1a93c8>
```



유방암 데이터 로지스틱회귀(7)

방법1. 로지스틱 회귀를 이용해 보자

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# StandardScaler( )로 평균이 0, 분산 1로 데이터 분포도 변환
scaler = StandardScaler()
data_scaled = scaler.fit_transform(cancer.data)

X_train , X_test, y_train , y_test = train_test_split(data_scaled, cancer.target, test_size=0.3, random_state=0)
```

데이터를 정규분포 형태의 표준 스케일링을 해줌

```
from sklearn.metrics import accuracy_score, roc_auc_score

# 로지스틱 회귀를 이용하여 학습 및 예측 수행.
lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)
lr_preds = lr_clf.predict(X_test)

# accuracy와 roc_auc 측정
print('accuracy: {:0.3f}'.format(accuracy_score(y_test, lr_preds)))
print('roc_auc: {:0.3f}'.format(roc_auc_score(y_test , lr_preds)))
```

로지스틱 회귀를 선택하고 훈련, 예측함.

```
accuracy: 0.977
roc_auc: 0.972
```

예측 정확도는 97.7% 이고, ROC-AUC 값은 97.2%임

유방암 데이터 로지스틱회귀(8)

```
from sklearn.model_selection import GridSearchCV

params={'penalty':['l2', 'l1'],
        'C':[0.01, 0.1, 1, 1, 5, 10]}

grid_clf = GridSearchCV(lr_clf, param_grid=params, scoring='accuracy', cv=3 )
grid_clf.fit(data_scaled, cancer.target)
print('최적 하이퍼 파라미터:{0}, 최적 평균 정확도:{1:.3f}'.format(grid_clf.best_params_,
                                                    grid_clf.best_score_))
```

기본적이 하이퍼 파리미터 서치로 사용할 개수를 나열함

```
최적 하이퍼 파라미터:{'C': 1, 'penalty': 'l2'}, 최적 평균 정확도:0.975
```

사이킷런 Logistic Regression의 하이퍼 파라미터

- Penalty : 규제(Regularization)가 있으며, L1, L2 규제가 있으며
  디폴트는 L2 규제이다.
- C : 규제 강도를 의미한다. 보통 1/alpha이고. alpha는 규제의 강도를 조절함

- 따라서, 2개의 Penality에 따른 7개의 C 파라리터 사용하여 , 총 14개 조함중에서 최적의 파라미터 1개를 찾음. (원시적인 방법)

유방암 데이터 로지스틱회귀(9)

방법2. SVM을 이용하자

```
df.head(3)
```

mean ctness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.6	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0.0
.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.8	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0.0
.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.5	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0.0

```
X = df.drop(['target'],axis=1)
y = df['target']
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# StandardScaler( )로 평균이 0, 분산 1로 데이터 분포도 변환
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

유방암 데이터 로지스틱회귀(10)

```
X_train , X_test, y_train , y_test = train_test_split(X, y, test_size=0.3, random_state=0)
#from sklearn.model_selection import train_test_split
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state=5)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(398, 30)
(171, 30)
(398,)
(171,)
```

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

svc_model = SVC()
svc_model.fit(X_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

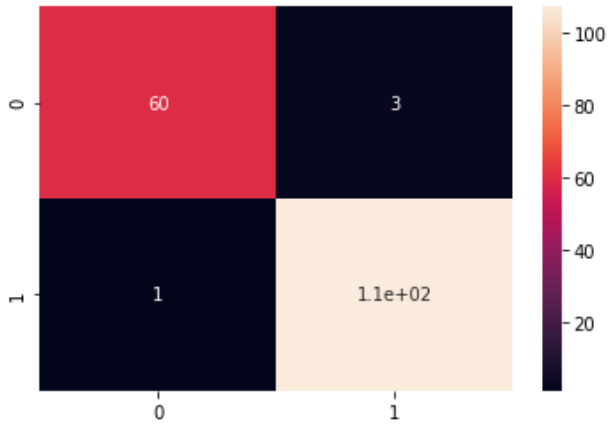
유방암 데이터 로지스틱회귀(11)

```
y_predict = svc_model.predict(X_test)
cm = confusion_matrix(y_test, y_predict)
```

```
print(classification_report(y_test, y_predict))

sns.heatmap(cm, annot=True)
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.95      0.97        63
         1.0       0.97      0.99      0.98       108

    accuracy                           0.98       171
   macro avg       0.98      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x195cb875438>
```



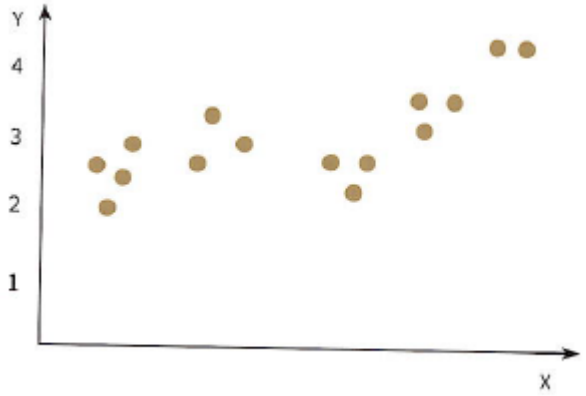
UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

03. 실습.회귀트리
보스턴 주택 가격 예측

이홍석 (hsyi@kisti.re.kr)

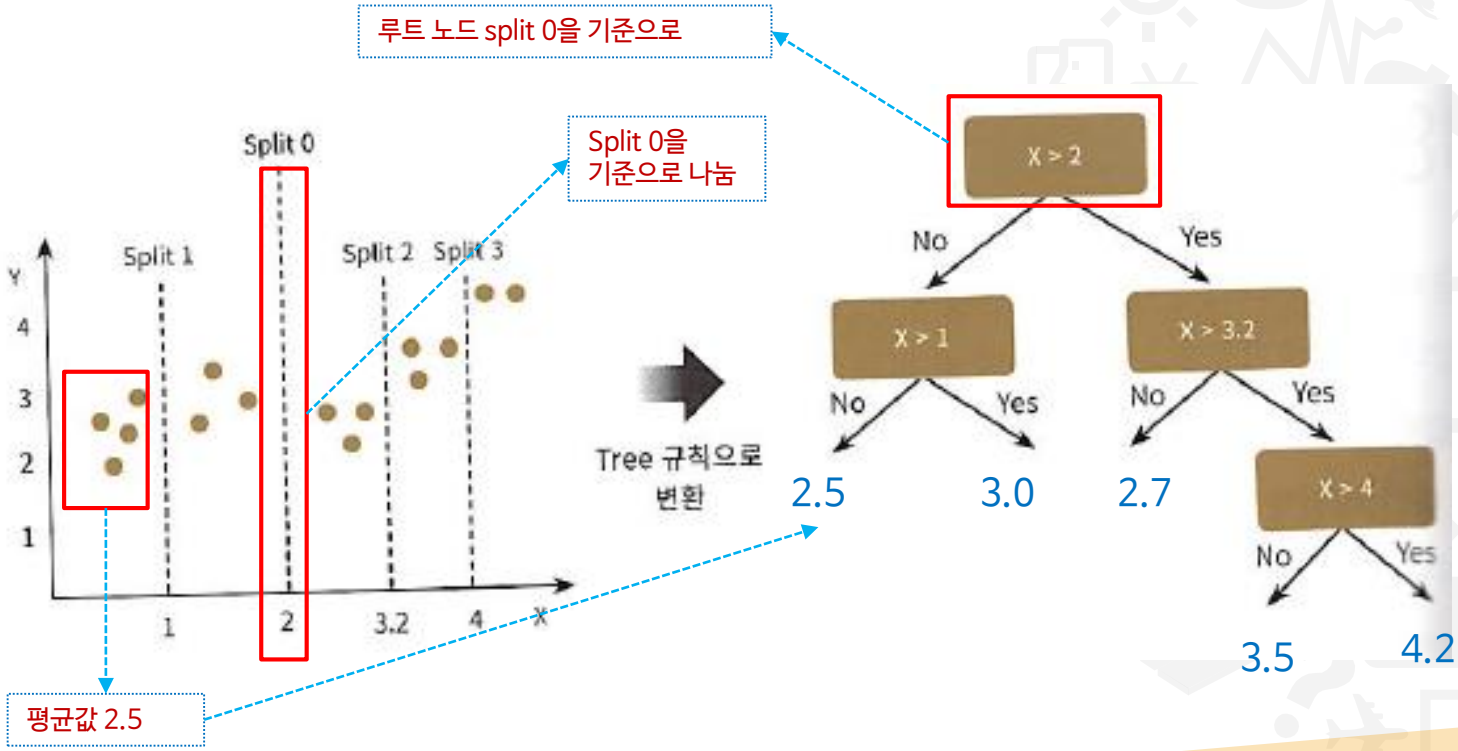
회귀 트리 (1)

- 회귀 함수 기반으로 하지 않고, 결정 트리로 회귀 방식을 소개함
- 회귀트리
  - 리프노드에 속한 데이터 값의 평균값을 구해 회귀 예측 값을 계산
  - 예) 2차원 데이터 분포



회귀 트리 (2)

- 방법
  - 이 데이터 세트의 X 피처를 결정 트리 기반으로 분류하면 X 값의 균일도를 반영한 Gini 계수에 따라 분할 (오른쪽)



회귀 트리: 보스턴 주택 가격예측(1)

```
from sklearn.datasets import load_boston
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import numpy as np

# 보스턴 데이터 세트 로드
boston = load_boston()
df= pd.DataFrame(boston.data, columns = boston.feature_names)
```

```
df.keys()
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')
```

회귀 트리: 보스턴 주택 가격예측(2)

```
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

```
df['PRICE'] = boston.target
y_target = df['PRICE']
X_data = df.drop(['PRICE'], axis=1,inplace=False)
```

타깃. 추가

```
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

회귀 트리: 보스턴 주택 가격예측(3)

```
rf = RandomForestRegressor(random_state=0, n_estimators=1000)

neg_mse_scores = cross_val_score(rf, X_data,
                                 y_target,
                                 scoring="neg_mean_squared_error", cv = 5)

rmse_scores  = np.sqrt(-1 * neg_mse_scores)

avg_rmse = np.mean(rmse_scores)

print(' 5 교차 검증의 개별 Negative MSE scores: ', np.round(neg_mse_scores, 2))

print(' 5 교차 검증의 개별 RMSE scores : ', np.round(rmse_scores, 2))

print(' 5 교차 검증의 평균 RMSE : {0:.3f} '.format(avg_rmse))
```

```
 5 교차 검증의 개별 Negative MSE scores:  [ -7.93 -13.06 -20.53 -46.31 -18.8 ]
 5 교차 검증의 개별 RMSE scores :  [2.82 3.61 4.53 6.8  4.34]
 5 교차 검증의 평균 RMSE : 4.420
```

회귀 트리: 보스턴 주택 가격예측(4)

입력 모델과 데이터 세트를 입력 받아 교차 검증으로 평균 RMSE를 계산해주는 함수

```
def get_model_cv_prediction(model, X_data, y_target):

    neg_mse_scores = cross_val_score(model,
                                     X_data, y_target,
                                     scoring="neg_mean_squared_error", cv = 5)

    rmse_scores  = np.sqrt(-1 * neg_mse_scores)

    avg_rmse = np.mean(rmse_scores)

    print('##### ',model.__class__.__name__ , ' #####')

    print(' 5 교차 검증의 평균 RMSE : {0:.3f} '.format(avg_rmse))
```

회귀 트리: 보스턴 주택 가격예측(5)

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

dt_reg = DecisionTreeRegressor(random_state=0, max_depth=4)
rf_reg = RandomForestRegressor(random_state=0, n_estimators=1000)
gb_reg = GradientBoostingRegressor(random_state=0, n_estimators=1000)
xgb_reg = XGBRegressor(n_estimators=1000)
lgb_reg = LGBMRegressor(n_estimators=1000)

# 트리 기반의 회귀 모델을 반복하면서 평가 수행
models = [dt_reg, rf_reg, gb_reg, xgb_reg, lgb_reg]
for model in models:
    get_model_cv_prediction(model, X_data, y_target)
```

회귀 트리: 보스턴 주택 가격예측(6)

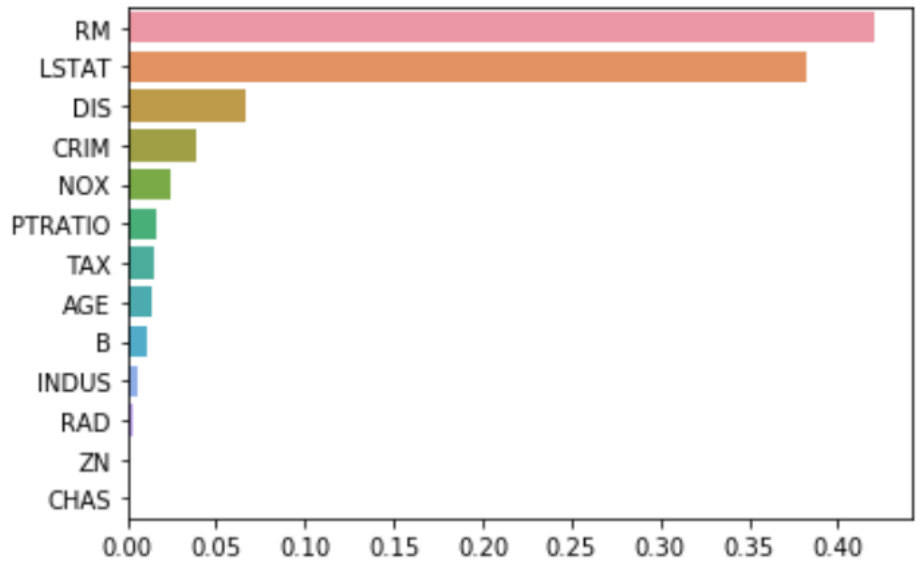
```
#####  DecisionTreeRegressor  #####
 5 교차 검증의 평균 RMSE : 5.978
#####  RandomForestRegressor  #####
 5 교차 검증의 평균 RMSE : 4.420
#####  GradientBoostingRegressor  #####
 5 교차 검증의 평균 RMSE : 4.269
#####  XGBRegressor  #####
 5 교차 검증의 평균 RMSE : 4.089
#####  LGBMRegressor  #####
 5 교차 검증의 평균 RMSE : 4.646
```

회귀 트리: 보스턴 주택 가격예측(7)

피처의 중요성을 시각화 feature_importances 이용

```
import seaborn as sns
%matplotlib inline
rf_reg = RandomForestRegressor(n_estimators=1000)
# 앞 예제에서 만들어진 X_data, y_target 데이터 셋을 적용하여 학습합니다.
rf_reg.fit(X_data, y_target)

feature_series = pd.Series(data=rf_reg.feature_importances_, index=X_data.columns )
feature_series = feature_series.sort_values(ascending=False)
sns.barplot(x= feature_series, y=feature_series.index)
```

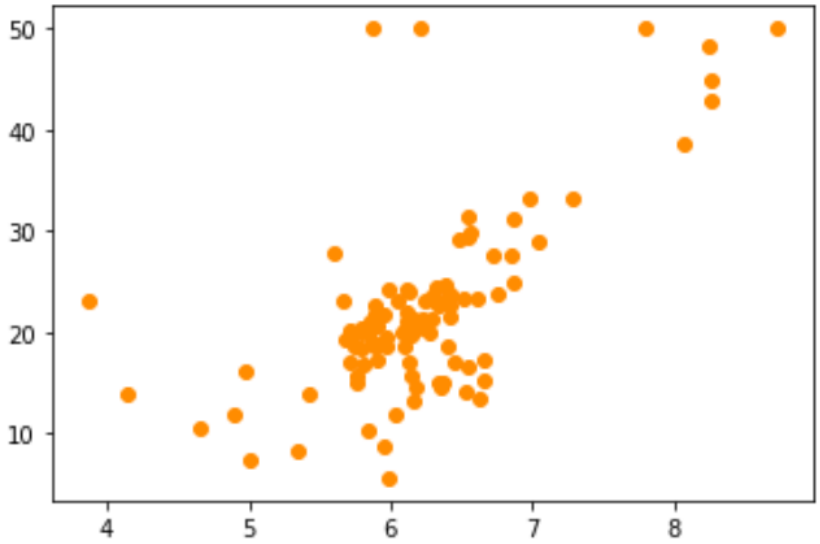
```
<matplotlib.axes._subplots.AxesSubplot at 0x195ce640320>
```



회귀 트리: 보스턴 주택 가격예측(8)

```
import matplotlib.pyplot as plt
%matplotlib inline

df_sample = df[['RM','PRICE']]
df_sample = df_sample.sample(n=100,random_state=0)
print(df_sample.shape)
plt.figure()
plt.scatter(df_sample.RM , df_sample.PRICE,c="darkorange")
```

```
(100, 2)

<matplotlib.collections.PathCollection at 0x195d7ab97f0>
```



회귀 트리: 보스턴 주택 가격예측(9)

```
import numpy as np
from sklearn.linear_model import LinearRegression

# 선형 회귀와 결정 트리 기반의 Regressor 생성. DecisionTreeRegressor의 max_depth는 각각 2, 7
lr_reg = LinearRegression()
rf_reg2 = DecisionTreeRegressor(max_depth=2)
rf_reg7 = DecisionTreeRegressor(max_depth=7)

# 실제 예측을 적용할 테스트용 데이터 셋을 4.5 ~ 8.5 까지 100개 데이터 셋 생성.
X_test = np.arange(4.5, 8.5, 0.04).reshape(-1, 1)

# 보스턴 주택가격 데이터에서 시각화를 위해 피처는 RM만, 그리고 결정 데이터인 PRICE 추출
X_feature = df_sample['RM'].values.reshape(-1,1)
y_target = df_sample['PRICE'].values.reshape(-1,1)

# 학습과 예측 수행.
lr_reg.fit(X_feature, y_target)
rf_reg2.fit(X_feature, y_target)
rf_reg7.fit(X_feature, y_target)

pred_lr = lr_reg.predict(X_test)
pred_rf2 = rf_reg2.predict(X_test)
pred_rf7 = rf_reg7.predict(X_test)
```

회귀 트리: 보스턴 주택 가격예측(10)

```
fig , (ax1, ax2, ax3) = plt.subplots(figsize=(14,4), ncols=3)

# X축값을 4.5 ~ 8.5로 변환하며 입력했을 때, 선형 회귀와 결정 트리 회귀 예측 선 시각화
# 선형 회귀로 학습된 모델 회귀 예측선
ax1.set_title('Linear Regression')
ax1.scatter(df_sample.RM, df_sample.PRICE, c="darkorange")
ax1.plot(X_test, pred_lr,label="linear", linewidth=2 )

# DecisionTreeRegressor의 max_depth를 2로 했을 때 회귀 예측선
ax2.set_title('Decision Tree Regression: ₩n max_depth=2')
ax2.scatter(df_sample.RM, df_sample.PRICE, c="darkorange")
ax2.plot(X_test, pred_rf2, label="max_depth:3", linewidth=2 )

# DecisionTreeRegressor의 max_depth를 7로 했을 때 회귀 예측선
ax3.set_title('Decision Tree Regression: ₩n max_depth=7')
ax3.scatter(df_sample.RM, df_sample.PRICE, c="darkorange")
ax3.plot(X_test, pred_rf7, label="max_depth:7", linewidth=2)
```

회귀 트리: 보스턴 주택 가격예측(10)

UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

04. 실습
자전거 대여 수요 예측

이홍석 (hsyi@kisti.re.kr)

(실습) 자전거 대여 수요 예측 (1)

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

bike_df = pd.read_csv(r"C:₩Users₩master₩work₩ust-hands-on₩Machine-Learning₩input₩bike_train.csv")
print(bike_df.shape)
bike_df.head(5)
```

해당 데이터는 2년간 날짜/시간, 기온, 습도, 풍속 등 정보를 기반으로 1시간 간격 동안의 자전건 대여 횟수를 수집한 것

(10886, 12)

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

라벨 값. 대여 횟수

(실습) 자전거 대여 수요 예측 (3)

```
bike_df.keys()
```

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

체감 온도

사전에 등록되지 않는
사용자가 대여한 횟수

```
bike_df.describe()
```

봄,여름,가을,겨울

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	c
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.00
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.57
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.14
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.00
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.00
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.00
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.00
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.00

1-맑음
2-안개
3-가벼운 눈,비
4-심한눈,천둥

(실습) 자전거 대여 수요 예측 (2)

```
bike_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
datetime      10886 non-null object
season        10886 non-null int64
holiday       10886 non-null int64
workingday    10886 non-null int64
weather       10886 non-null int64
temp          10886 non-null float64
atemp         10886 non-null float64
humidity      10886 non-null int64
windspeed     10886 non-null float64
casual        10886 non-null int64
registered    10886 non-null int64
count         10886 non-null int64
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.6+ KB
```

오브젝트 형태의 년-월-일 시:분:초 문자 형식으로 돼 있음

년, 월, 일, 시간 처럼 4개의 속성으로 분리

판다스에서 문자열을 'datetime' 타입을 변경.

(참고로, 판다스의 datatime 타입과 예제 데이터 세트의 datatime 컬럼명이 우연히 동일. 오해 없기를

(실습) 자전거 대여 수요 예측 (4)

```
bike_df['datetime'] = bike_df.datetime.apply(pd.to_datetime)

bike_df['year'] = bike_df.datetime.apply(lambda x : x.year)
bike_df['month'] = bike_df.datetime.apply(lambda x : x.month)
bike_df['day'] = bike_df.datetime.apply(lambda x : x.day)
bike_df['hour'] = bike_df.datetime.apply(lambda x: x.hour)
bike_df.head(3)
```

casual과 registered의 합이 count이므로, 중복으로 기제됨.

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	year	month	day	hour
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	2011	1	1	0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	2011	1	1	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	2011	1	1	2

```
drop_columns = ['datetime','casual','registered']
bike_df.drop(drop_columns, axis=1,inplace=True)
```

컬럼에 있는 항목을 제외함.

더 이상 필요가 없는 항목은 과감히 삭제. 만일 삭제하지 않으면 오히려 상관관계가 높아져서 예측을 저해할 우려도 발생할 수 있음

4개가 새롭게 추가됨.

(실습) 자전거 대여 수요 예측 (5)

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error

# log 값 변환 시 NaN등의 이슈로 log()가 아닌 log1p()를 이용하여 RMSLE 계산
def rmsle(y, pred):
    log_y = np.log1p(y)
    log_pred = np.log1p(pred)
    squared_error = (log_y - log_pred) ** 2
    rmsle = np.sqrt(np.mean(squared_error))
    return rmsle

# 사이킷런의 mean_square_error() 를 이용하여 RMSE 계산
def rmse(y,pred):
    return np.sqrt(mean_squared_error(y,pred))

# MSE, RMSE, RMSLE 를 모두 계산
def evaluate_regr(y,pred):
    rmsle_val = rmsle(y,pred)
    rmse_val = rmse(y,pred)
    # MSE 는 scikit learn의 mean_absolute_error() 로 계산
    mse_val = mean_absolute_error(y,pred)
    print('RMSLE: {0:.3f}, RMSE: {1:.3F}, MSE: {2:.3F}'.format(rmsle_val, rmse_val, mse_val))
```

RMSLE(Root Mean Square Log Error) 오류값의 로그에 대한 RMSE.

log1p(y)=1+log(y)를 사용하여 log 함수 사용시 오버플로우와 언더 플로우를 방지함

$$RMSLE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(log(y_i+1)-log(\hat{y}_i+1))^2}$$

(실습) 자전거 대여 수요 예측 (6)

1) 선형회귀를 사용할 때 예측 성능을 검증해보자

```
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.linear_model import LinearRegression , Ridge , Lasso

y_target = bike_df['count']

X_features = bike_df.drop(['count'],axis=1,inplace=False)

X_train, X_test, y_train, y_test = train_test_split(X_features,
                                                    y_target,
                                                    test_size=0.3, random_state=0)

lr_reg = LinearRegression()

lr_reg.fit(X_train, y_train)

pred = lr_reg.predict(X_test)

evaluate_regr(y_test ,pred)
```

훈련과 테스트 데이터 분할은 0.7 vs 0.3

선형회귀를 사용

```
RMSLE: 1.165, RMSE: 140.900, MSE: 105.924
```

실제 count 값을 비교하면 오차는 매우 큰 편이다.

(실습) 자전거 대여 수요 예측 (7)

2) 실제값과 예측값의 벗어난 정도를 보기 위하여 상위 5개만 확인

```
def get_top_error_data(y_test, pred, n_tops = 5):

    # DataFrame에 컬럼들로 실제 대여횟수(count)와 예측 값을 서로 비교 할 수 있도록 생성.

    result_df = pd.DataFrame(y_test.values, columns=['real_count'])

    result_df['predicted_count']= np.round(pred)

    result_df['diff'] = np.abs(result_df['real_count'] - result_df['predicted_count'])

    # 예측값과 실제값이 가장 큰 데이터 순으로 출력.
    print(result_df.sort_values('diff', ascending=False)[:n_tops])

get_top_error_data(y_test, pred, n_tops=5)
```

	real_count	predicted_count	diff
1618	890	322.0	568.0
3151	798	241.0	557.0
966	884	327.0	557.0
412	745	194.0	551.0
2817	856	310.0	546.0

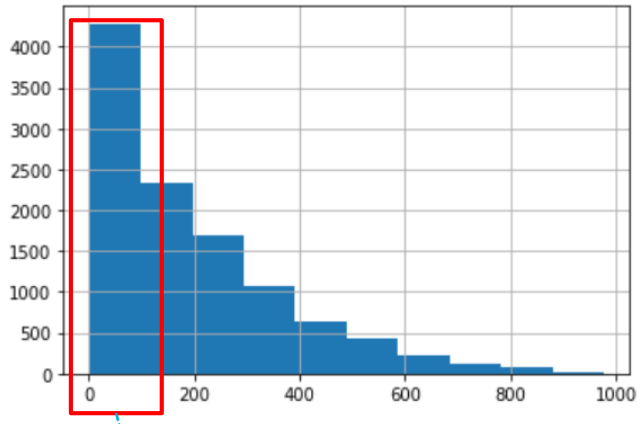
실제 값에 바교하여 오차가 매우 크다는 것을 알수 있다.
회귀에서 큰 오차의 원인은 target 값 분포가 왜곡된 형태 일때.

타켓 값이 정규분포이면 가장 좋다.

(실습) 자전거 대여 수요 예측 (8)

```
y_target.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28d0c84e0b8>
```

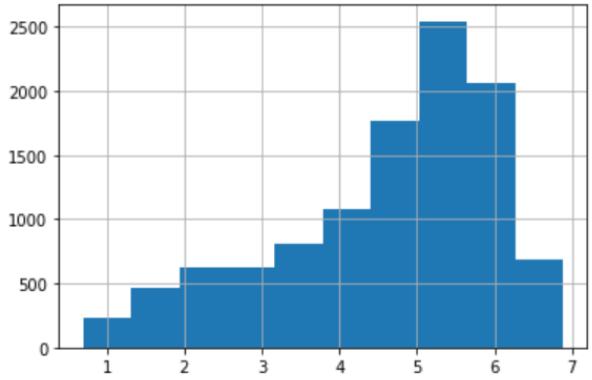


target 값이 정규 분포로 안보이고,
200 이하에 대부분 치우쳐 있다.

로그(log) : 보통 왜곡된 값을 정규분포 형태로 변화함

```
y_log_transform = np.log1p(y_target)
y_log_transform.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28d36b02978>
```



(실습) 자전거 대여 수요 예측 (9)

```
# 타겟 컬럼인 count 값을 log1p 로 Log 변환
y_target_log = np.log1p(y_target)

# 로그 변환된 y_target_log를 반영하여 학습/테스트 데이터 셋 분할
X_train, X_test, y_train, y_test = train_test_split(
    X_features, y_target_log, test_size=0.3, random_state=0)

lr_reg = LinearRegression()

lr_reg.fit(X_train, y_train)

pred = lr_reg.predict(X_test)

# 테스트 데이터 셋의 Target 값은 Log 변환되었으므로
# 다시 expm1를 이용하여 원래 scale로 변환
y_test_exp = np.expm1(y_test)

# 예측 값 역시 Log 변환된 타겟 기반으로 학습되어 예측되었으므로 다시 exmp1으로 scale변환
pred_exp = np.expm1(pred)

evaluate_regr(y_test_exp ,pred_exp)
```

타켓 로그로 변환하여 다시 훈련함

로그 변환 이전의 원래 target 값으로 다시 만들어 줌

```
RMSLE: 1.017, RMSE: 162.594, MSE: 109.286
```

RMSE와 MSE는 오히려 더 큰 오차를 보임. 왜 이런일이 발생한 것일까?
RMSLE는 오차가 작아졌음.(Good)

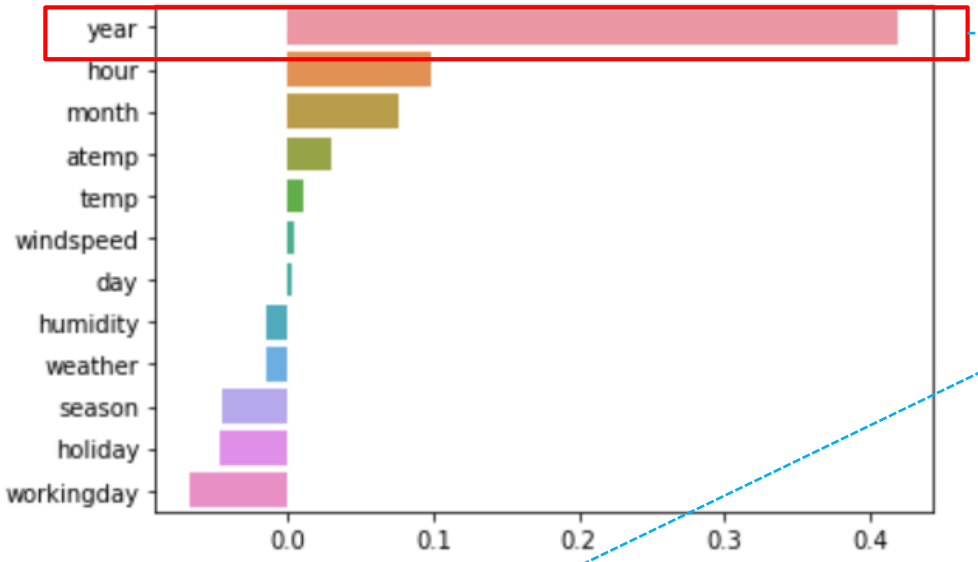
```
RMSLE: 1.165, RMSE: 140.900, MSE: 105.924
```

(실습) 자전거 대여 수요 예측 (10)

문제 파악을 위해 선형 회귀 계수 값을 그려보자.

```
coef = pd.Series(lr_reg.coef_, index=X_features.columns)
coef_sort = coef.sort_values(ascending=False)
sns.barplot(x=coef_sort.values, y=coef_sort.index)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28d36b7cc50>
```



year의 회귀 계수가 매우 크다.
year는 숫자로서 2011, 2012로 회귀에 사용됨.
하지만, year는 카타고리 형으로 다루어야 한다.
왜?

판다스의 One-Hot 인코딩은
get_dummies() 함수 이용

자전거 대여 데이터 중에서, 숫자 보다는 카타로리
형으로 변경이 가능 한 것은 다 바꾸자.

```
# 'year','month','hour','season','weather' feature들을 One Hot Encoding
X_features_ohe = pd.get_dummies(X_features, columns=['year','month','hour', 'holiday',
                                                     'workingday','season','weather'])
```

(실습) 자전거 대여 수요 예측 (11)

```
# 원-핫 인코딩이 적용된 feature 데이터 세트 기반으로 학습/예측 데이터 분할.
X_train, X_test, y_train, y_test = train_test_split(X_features_ohe, y_target_log,
                                                    test_size=0.3, random_state=0)

def get_model_predict(model, X_train, X_test, y_train, y_test, is_expm1=False):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    if is_expm1 :
        y_test = np.expm1(y_test)
        pred = np.expm1(pred)
    print('###',model.__class__.__name__,'###')
    evaluate_regr(y_test, pred)

lr_reg = LinearRegression()
ridge_reg = Ridge(alpha=10)
lasso_reg = Lasso(alpha=0.01)

for model in [lr_reg, ridge_reg, lasso_reg]:
    get_model_predict(model,X_train, X_test, y_train, y_test,is_expm1=True)
```

```
### LinearRegression ###
RMSLE: 0.589, RMSE: 97.483, MSE: 63.106
### Ridge ###
RMSLE: 0.589, RMSE: 98.407, MSE: 63.648
### Lasso ###
RMSLE: 0.634, RMSE: 113.031, MSE: 72.658
```

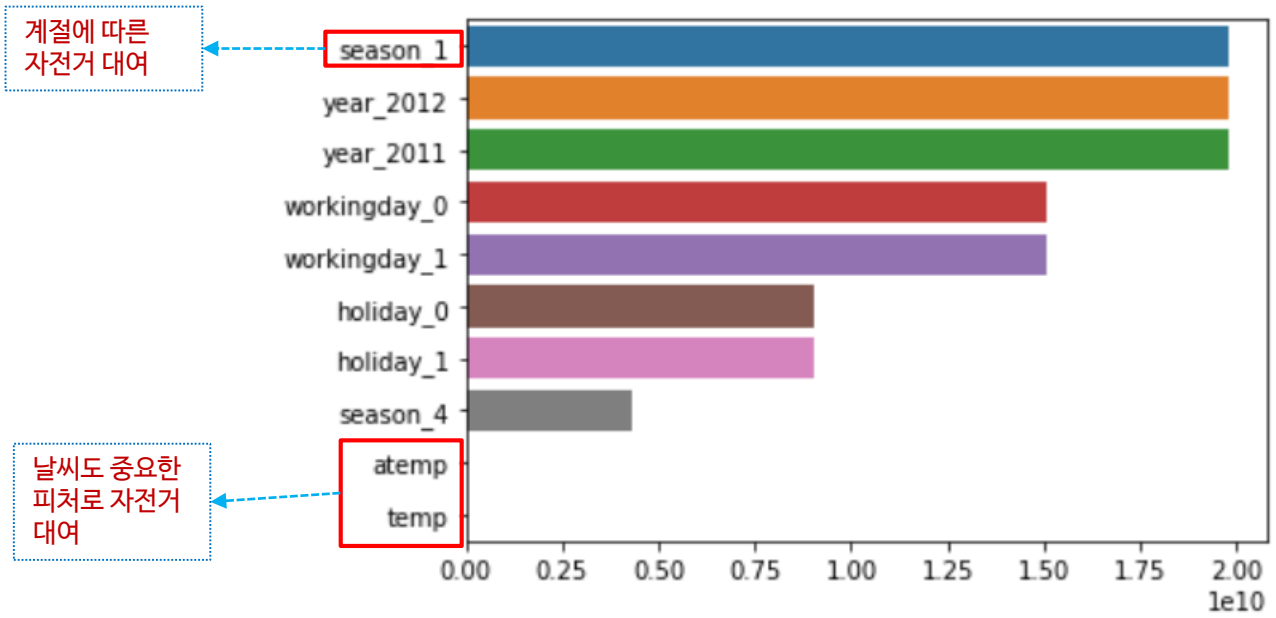
One-Hot 인코딩로 바꾸어 주었더니 큰 성능 향상

(실습) 자전거 대여 수요 예측 (12)

(중요) 선형회귀에서는 피처를 어떻게 인코딩 하는가가 성능을 좌우 함!

```
coef = pd.Series(lr_reg.coef_ , index=X_features_ohe.columns)
coef_sort = coef.sort_values(ascending=False)[:10]
sns.barplot(x=coef_sort.values , y=coef_sort.index)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28d36c32358>
```



(실습) 자전거 대여 수요 예측 (12)

램덤 포레스트, GBM, XGGoost, LightGBM에 따른 예측

```
### RandomForestRegressor ###
RMSLE: 0.355, RMSE: 50.756, MSE: 31.514
### GradientBoostingRegressor ###
RMSLE: 0.340, RMSE: 55.781, MSE: 34.337
### XGBRegressor ###
RMSLE: 0.346, RMSE: 56.474, MSE: 34.917
### LGBMRegressor ###
RMSLE: 0.316, RMSE: 46.473, MSE: 28.777
```

선형 회귀 모데에 비하여 매우 우수한 자전거 대여 예측

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor


# 랜덤 포레스트, GBM, XGBoost, LightGBM model 별로 평가 수행
rf_reg = RandomForestRegressor(n_estimators=500)

gbm_reg = GradientBoostingRegressor(n_estimators=500)

xgb_reg = XGBRegressor(n_estimators=500)

lgbm_reg = LGBMRegressor(n_estimators=500)

for model in [rf_reg, gbm_reg, xgb_reg, lgbm_reg]:
    get_model_predict(model,X_train, X_test, y_train, y_test,is_expm1=True)
```

정리 및 연습문제

- 퀴즈 및 숙제
  - 선형회귀의 장단점을 논해라 (BGD, SGD, Mini-BGD)
  - 훈련세트가 특성이 각기 다른 스케일로 구성되었다. 최적의 방법은
  - 로지스틱 회귀에서 로칼미너엄을 빠져나올 방법은?
  - 배치 경사 강하에서 에포크 마다 검증오차가 일정하게 상승한다면 문제점은?
  - 검증오차가 상승하면 미니매치 경사 하강법을 즉시 중단하는 것에 대하여
  - 다항회귀에서 검증과 훈련 오차가 사이 간격이 크다면 의미는
  - 밤과 낮, 실내와 실외 사진을 분류한다면, 좋은 분류기는 소프트맥스인가?

UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

Thank You!

www.ust.ac.kr