Regression, Logistic Regression

Machine Learning

Today's Code Practice

- Scikit-learn
  - Logistic Regression
  - Linear Regression

https://scikit-learn.org/stable/

Logistic Regression

- What is the Logistic Regression?
  - In regression analysis, Logistic Regression is estimating the parameters of a logistic model; it is a form of binomial regression
  - Odds ratio

$$\frac{p}{1-p}$$

  - Logit function
    - Logarithm of odds, the logit of the probability

$$logit(p) = ln\frac{p}{1-p} \quad for \;\; 0 < p < 1$$

    - Logit function is the link function in this kind of generalized linear model

$$logit\big(p(y=1|x)\big) = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{i=0}^{m} w_ix_i = w^Tx$$

  - Logistic function(Sigmoid function)

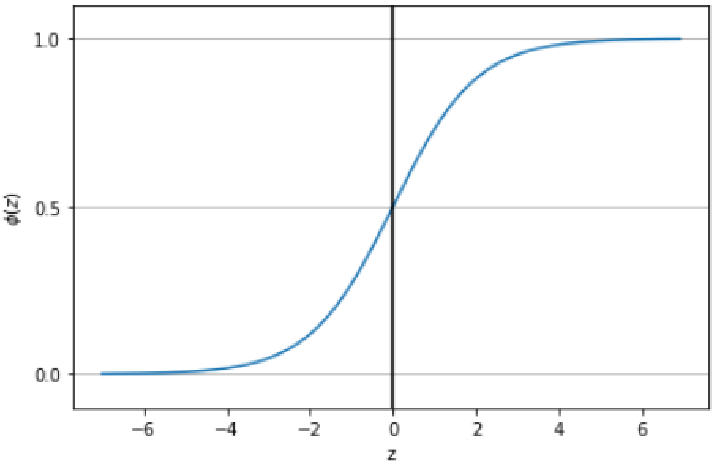
$$\phi(z) = \frac{1}{1+e^{-z}}$$

Logistic Regression

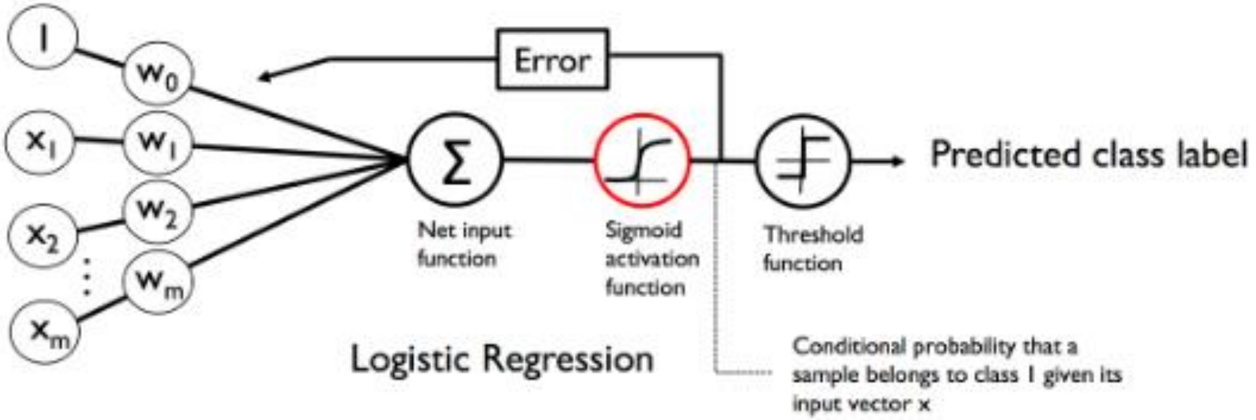
- What is the Logistic Regression?
  - Logistic function(Sigmoid function)

$$\phi(z) = \frac{1}{1+e^{-z}}$$

$$\hat{y} = \begin{cases} 1 & if\ \phi(z) \geq 0.5 \\ 0 & otherwise \end{cases} \qquad \hat{y} = \begin{cases} 1 & if\ z \geq 0.0 \\ 0 & otherwise \end{cases}$$

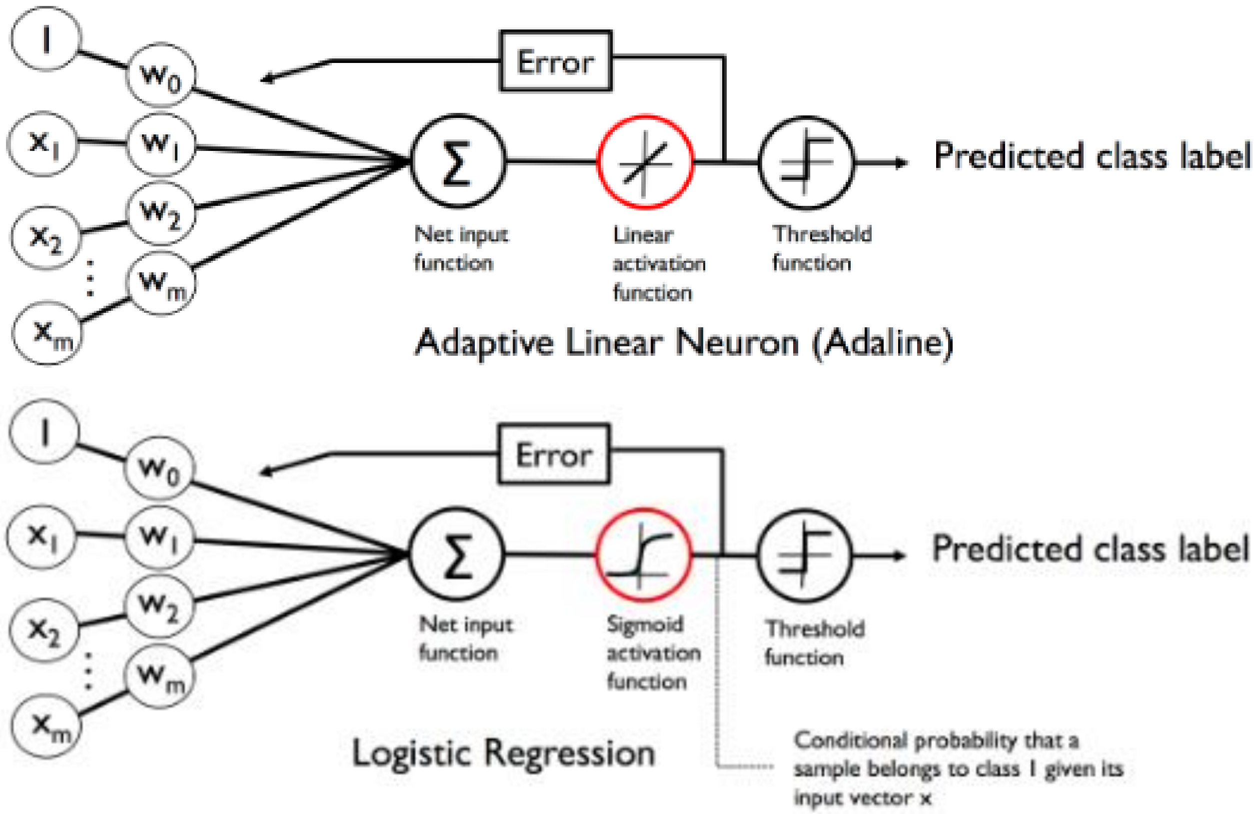


  - Logistic Regression



Logistic Regression

- Adaline vs Logistic Regression



Logistic Regression

- Learning the weights of the logistic cost function
  - Cost function

$$J(w) = \sum_{i} \frac{1}{2}\left(\phi\left(z^{(i)}\right) - y^{(i)}\right)^2$$

  - Maximum likelihood Estimation
    - Likelihood

$$L(w) = P(y|x;w) = \prod_{i=1}^{n} P\left(y^{(i)}|x^{(i)};w\right) = \prod_{i=1}^{n}\left(\phi\left(z^{(i)}\right)\right)^{y^{(i)}}\left(1-\phi\left(z^{(i)}\right)\right)^{1-y^{(i)}}$$

    - Log-Likelihood

$$l(w) = logL(w) = \sum_{i=1}^{n}\left[y^{(i)}\log\left(\phi\left(z^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-\phi\left(z^{(i)}\right)\right)\right]$$

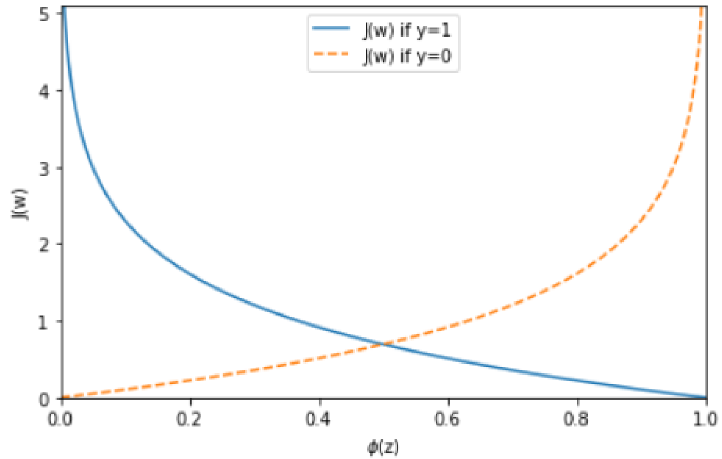
Logistic Regression

- Learning the weights of the logistic cost function
  - Log likelihood for Gradient Descent

$$J(w) = \sum_{i=1}^{n} [-y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right)\log(1 - \phi\left(z^{(i)}\right))]$$

  - For one sample instance,

$$J(\phi(z), y; w) = \begin{cases} -\log(\phi(z)) & if\ y = 1 \\ -\log(1 - \phi(z)) & if\ y = 0 \end{cases}$$



Logistic Regression - Binary Classification

- Logistic regression via scikit-learn
  - Load Iris Dataset

```
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
# get X, y
X = iris.data[0:100, [0, 2]] # select 2 features, 2:petal length and 3:petal width
y = iris.target[0:100]
print(X.shape)
print(y.shape)
print(X[:3])
print(y)
print('Class labels:', np.unique(y))
```

```
(100, 2)
(100,)
[[ 5.1 1.4]
[ 4.9 1.4]
[ 4.7 1.3]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Class labels: [0 1]
```

Logistic Regression - Binary Classification

- Logistic regression via scikit-learn
  - Splitting data into 70% training data & 30% test data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=1, stratify=y)
```

```
print('Labels counts in y:', np.bincount(y))
print('Labels counts in y_train:', np.bincount(y_train))
print('Labels counts in y_test:', np.bincount(y_test))
```

```
Labels counts in y: [50 50 50]
Labels counts in y_train: [35 35 35]
Labels counts in y_test: [15 15 15]
```

Logistic Regression - Binary Classification

- Logistic regression via scikit-learn
  - Standardizing the the features

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

X_train_std[:3]
X_test_std[:3]
```

```
array([[-0.94135169, -1.0309217 ],
       [ 1.12647106,  1.2740875 ],
       [-0.94135169, -0.8536133 ]])

array([[ 1.26432591,  1.0967791 ],
       [ 0.09255969,  0.5648539 ],
       [-1.01027912, -1.2082301 ]])
```

Logistic Regression - Binary Classification

- Logistic regression via scikit-learn
  - Logistic regression via scikit-learn

```
from sklearn.linear_model import LogisticRegression
# training the model
lr = LogisticRegression(C=100.0, random_state=1)
lr.fit(X_train_std, y_train)
```

```
LogisticRegression(C=100.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=1, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
# predicting y
y_pred = lr.predict(X_test_std)
y_pred
```

```
array([1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
1, 1, 0, 0, 1, 1, 0])
```

```
y_test
```

```
array([1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
1, 1, 0, 0, 1, 1, 0])
```

Logistic Regression - Binary Classification

- Logistic regression via scikit-learn
  - Logistic regression via scikit-learn

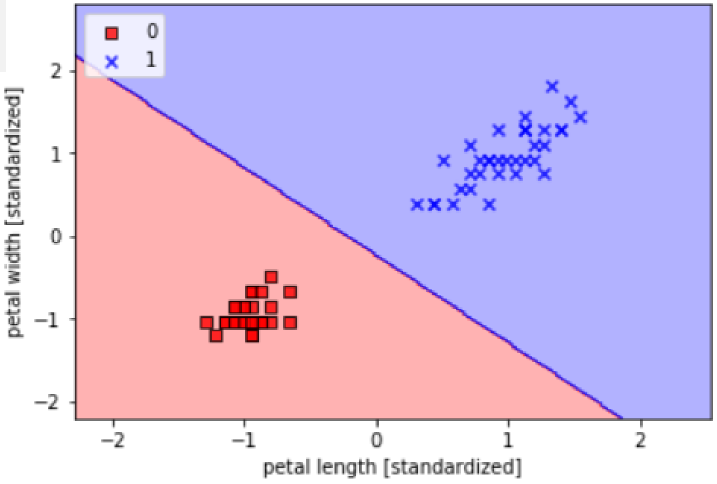
```
# number of misclassification
print('Misclassified test samples: %d' % (y_test != y_pred).sum())

# accuracy of the model
print('Training accuracy: %.2f' % lr.score(X_train_std, y_train))
print('Test accuracy: %.2f' % lr.score(X_test_std, y_test))

# model parameters
print('w = ', lr.coef_)
print('b = ', lr.intercept_)
```

```
Misclassified test samples: 0

Training accuracy: 1.00
Test accuracy: 1.00

w =  [[ 4.63943217  4.38537802]]
b =  [ 1.03162359]
```

Logistic Regression - Binary Classification

- Logistic regression via scikit-learn
  - Plotting Decision Regions

```
# decision boundary of the model
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

plot_decision_regions(X_combined_std, y_combined,
classifier=lr, test_idx=range(105, 150))

plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



Logistic Regression - Multinomial Classification

- Logistic regression via scikit-learn
  - Load Iris Dataset

```
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
# get X, y
X = iris.data[0:100, [0, 2]] # select 2 features, 2:petal length and 3:petal width
y = iris.target[0:100]
print(X.shape)
print(y.shape)
print(X[:3])
print(y)
print('Class labels:', np.unique(y))
```

```
(150, 2)
(150,)
[[ 1.4 0.2]
[ 1.4 0.2]
[ 1.3 0.2]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2]
Class labels: [0 1 2]
```

Logistic Regression - Multinomial Classification

- Logistic regression via scikit-learn
  - Splitting data into 70% training data & 30% test data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                     random_state=1, stratify=y)
```

```
print('Labels counts in y:', np.bincount(y))
print('Labels counts in y_train:', np.bincount(y_train))
print('Labels counts in y_test:', np.bincount(y_test))
```

```
Labels counts in y: [50 50 50]
Labels counts in y_train: [35 35 35]
Labels counts in y_test: [15 15 15]
```

Logistic Regression - Multinomial Classification

- Logistic regression via scikit-learn
  - Standardizing the the features

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

X_train_std[:3]
X_test_std[:3]
```

```
array([[-1.33269725, -1.30380366],
       [-1.16537974, -1.30380366],
       [ 0.84243039,  1.44465434]])
```

```
array([[ 0.89820289,  1.44465434],
       [-1.16537974, -1.04204575],
       [-1.33269725, -1.17292471]])
```

Logistic Regression - Multinomial Classification

- Logistic regression via scikit-learn
  - Logistic regression via scikit-learn

```
from sklearn.linear_model import LogisticRegression
# training the model
lr = LogisticRegression(C=100.0, random_state=1)
lr.fit(X_train_std, y_train)
```

```
LogisticRegression(C=100.0, class_weight=None, dual=False, fit_intercept=True,
           intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
           penalty='l2', random_state=1, solver='liblinear', tol=0.0001,
           verbose=0, warm_start=False)
```

```
python # predicting y
y_pred = lr.predict(X_test_std)
y_pred
```

```
array([2, 0, 0, 1, 1, 1, 2, 1, 2, 0, 0, 2, 0, 1, 0, 1, 2, 1, 1, 2, 2, 0, 1,
2, 1, 1, 1, 2, 0, 2, 0, 0, 1, 1, 2, 2, 0, 0, 0, 1, 2, 2, 1, 0, 0])
```

```
y_test
```

```
array([2, 0, 0, 2, 1, 1, 2, 1, 2, 0, 0, 2, 0, 1, 0, 1, 2, 1, 1, 2, 2, 0, 1,
2, 1, 1, 1, 2, 0, 2, 0, 0, 1, 1, 2, 2, 0, 0, 0, 1, 2, 2, 1, 0, 0])
```

Logistic Regression - Binary Classification

- Logistic regression via scikit-learn
  - Logistic regression via scikit-learn

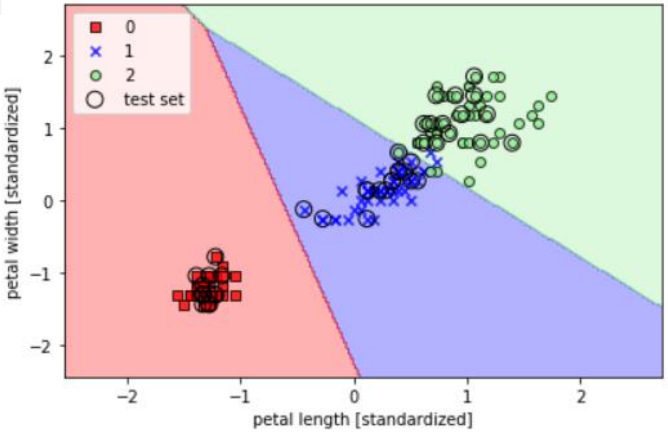
```
# number of misclassification
print('Misclassified test samples: %d' % (y_test != y_pred).sum())

# accuracy of the model
print('Training accuracy: %.2f' % lr.score(X_train_std, y_train))
print('Test accuracy: %.2f' % lr.score(X_test_std, y_test))

# model parameters
print('w = ', lr.coef_)
print('b = ', lr.intercept_)
```

```
Misclassified test samples: 1

Training accuracy: 0.95
Test accuracy: 0.98

w = [[-5.61119214 -4.3095919 ]
     [ 2.38375195 -2.04552965]
     [ 9.51463313 5.40199177]]
b = [-5.83309891 -0.75660259 -9.21677488]
```

Logistic Regression - Multinomial Classification

- Logistic Regression Using Scikit-learn
  - Plotting Decision Regions

```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

# decision boundary of the model
plot_decision_regions(X_combined_std, y_combined,
classifier=lr, test_idx=range(105, 150))

plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



Logistic Regression - Multinomial Classification

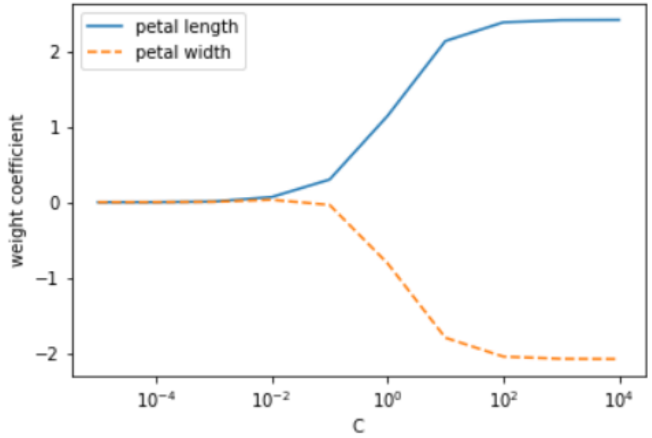
- Logistic Regression Using Scikit-learn
  - Tackling Overfitting via Regularization – (1)

```
params = [] # C for regularization
weights = [] # weights for each C
test_acc = [] # test accuracy for each C

# computing weights and accuracy for each C
for c in np.arange(-5, 5):
     lr = LogisticRegression(C=10.**c, random_state=1)
     lr.fit(X_train_std, y_train)
     params.append(10.**c)
     weights.append(lr.coef_[1])
     test_acc.append(lr.score(X_test_std, y_test))
```

Logistic Regression - Multinomial Classification

- Logistic Regression Using Scikit-learn
  - Tackling Overfitting via Regularization – (2)

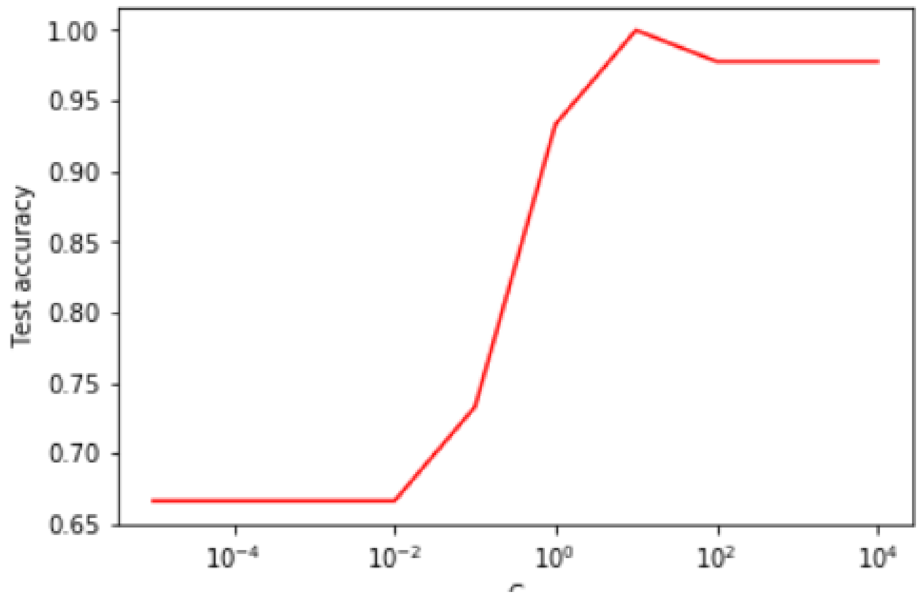
```
weights = np.array(weights)

# plotting weights each C
plt.plot(params, weights[:, 0], label='petal length')
plt.plot(params, weights[:, 1], label='petal width', linestyle='--')
plt.ylabel('weights')
plt.xlabel('C')
plt.legend(loc='upper left')
plt.xscale('log')
plt.show()
```



Logistic Regression - Multinomial Classification

- Logistic Regression Using Scikit-learn
  - Tackling Overfitting via Regularization – (3)

```
# plotting accuracies for each C
plt.plot(params, test_acc, color='red')
plt.ylabel('Test accuracy')
plt.xlabel('C')
plt.xscale('log')
plt.show()
```



Linear Regression

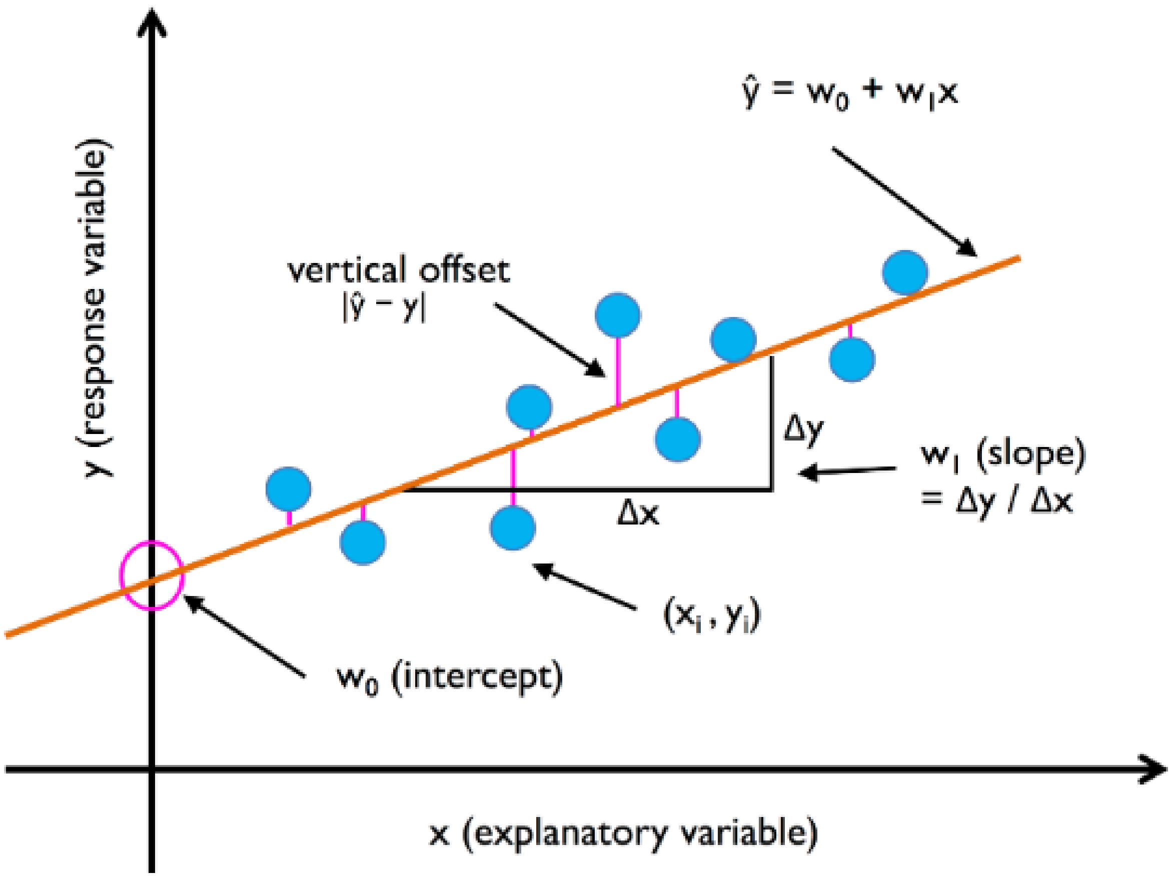
- What is the Linear Regression?
  - Linear regression is a linear approach to modelling the relationship between a scalar response(or dependent variable) and one or more explanatory variables
  - In linear regression, the relationships are modeled using linear predictor function whose unknown model parameters are estimated from the data
  - Cost function

$$y = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{i=0}^{m} w_i x_i = w^T x$$

$$J(w) = \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

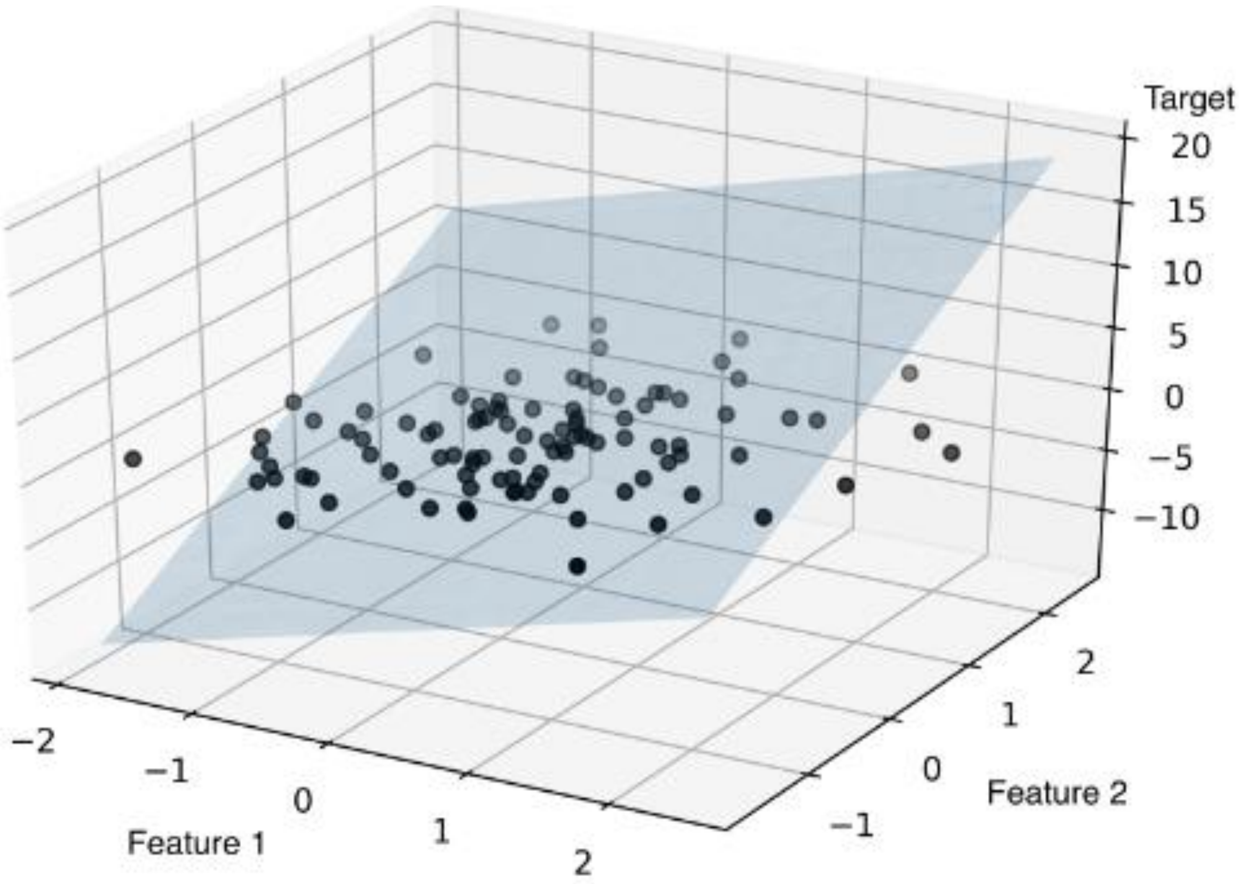
Linear Regression

- Simple Linear Regression



Linear Regression

- Multiple Linear Regression



Linear Regression

- Housing dataset

1	CRIM	per capita crime rate by town
2	ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
3	INDUS	proportion of non-retail business acres per town
4	CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5	NOX	nitric oxides concentration (parts per 10 million)
6	RM	average number of rooms per dwelling
7	AGE	proportion of owner-occupied units built prior to 1940
8	DIS	weighted distances to five Boston employment centres
9	RAD	index of accessibility to radial highways
10	TAX	full-value property-tax rate per $10,000
11	PTRATIO	pupil-teacher ratio by town
12	B	1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13	LSTAT	% lower status of the population
14	MEDV	Median value of owner-occupied homes in $1000s

Linear Regression

- Linear Regression
  - Loading the Housing dataset into a dataframe

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv("housing_data.txt", header=None, sep='\s+')

df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS',
              'NOX', 'RM', 'AGE', 'DIS', 'RAD',
              'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

df.head()
```

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

Linear Regression Using Scikit-learn

- Linear Regression
  - Correlations between variables

```
# check the correlation efficients between all varibles
df.corr()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000

…

⋮

Linear Regression Using Scikit-learn

- Linear Regression
  - Correlations between variables

```
# the correlation between the dependent variable and each independent variable - sorted
df.corr()[["MEDV"]].sort_values("MEDV", ascending=False)
```

	MEDV
MEDV	1.000000
RM	0.695360
ZN	0.360445
B	0.333461
DIS	0.249929
CHAS	0.175260
AGE	-0.376955
RAD	-0.381626
CRIM	-0.388305
NOX	-0.427321
TAX	-0.468536
INDUS	-0.483725
PTRATIO	-0.507787
LSTAT	-0.737663

Linear Regression Using Scikit-learn

- Linear Regression
  - Selecting the variable based on correlation between dependent variable

```
# plotting number of rooms(RM) vs. price(MEDV)
plt.scatter(df["RM"].values, df["MEDV"].values,
            c="steelblue", edgecolor="white", s=70)
plt.title('PRICES vs ROOM')
plt.xlabel('Number of rooms')
plt.ylabel('Price')
plt.grid()
plt.show()
```

Linear Regression Using Scikit-learn

- Linear Regression
  - Linear regression via scikit-learn

```
# get X, y
X = df[['RM']]
y = df[['MEDV']]
print(X.shape)
print(y.shape)
print(X[:5])
print(y[:5])
```

```
(506, 1)
(506, 1)
RM
0 6.575
1 6.421
2 7.185
3 6.998
4 7.147
MEDV
0 24.0
1 21.6
2 34.7
3 33.4
4 36.2
```

Linear Regression Using Scikit-learn

- Linear Regression
  - Linear regression via scikit-learn

```
from sklearn.linear_model import LinearRegression
# training the model
lr = LinearRegression()
lr.fit(X, y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
# model parameters
print('w = ', lr.coef_)
print('b = ', lr.intercept_)
```

```
w = [[ 9.10210898]]
b = [-34.67062078]
```

```
from sklearn.metrics import mean_squared_error
# mean squared error(MSE) of the prediction
y_pred = lr.predict(X)
print('MSE : %.3f' % mean_squared_error(y, y_pred))
```

```
MSE : 43.601
```

Linear Regression Using Scikit-learn

- Linear Regression
  - Plotting Linear Regression

```
# plotting the model
plt.plot(X, lr.predict(X), color='red', lw=2)
plt.scatter(df["RM"].values, df["MEDV"].values,
             c="steelblue", edgecolor="white", s=70)
plt.title('PRICES vs ROOM')
plt.xlabel('Number of rooms')
plt.ylabel('Price')
plt.grid()
plt.show()
```

Submit

- To make sure if you have completed this practice,
  Submit your practice file(Week05_givencode.ipynb) to e-class.
- **Deadline : tomorrow 11:59pm**
- Modify your ipynb file name as “Week05_StudentNum_Name.ipynb”
  Ex) **Week05_2020123456_홍길동.ipynb**
- You can upload this file without taking the quiz, but homework will be provided like a quiz every three weeks, so it is recommended to take the quiz as well.

Quiz 1 : Linear Regression

- Find a model predicting 'MEDV' from 'LSTAT' using Boston Housing Dataset
  - Train linear regression model with 'MEDV' as a dependent variable and 'LSTAT'(Lower status of population %) as an independent variable
  - Show the model(parameters), compute the MSE, and plot the model

Quiz 2 : Logistic Regression

- Find a model for cancer classification using Breast Cancer Wisconsin Dataset
  - Train logistic regression model using all the features. The target class is 0(malignant) or 1(benign)
  - Use 70% of dataset for training, 30% for testing. Standardize the features
  - Show the model(parameters), compute the accuracy, and plot the train and test accuracies for difference C values
  - Predict the class of following data:

    [[11.2, 18.5, 78.3, 451.00, 0.092,
    0.081, 0.031, 0.042, 0.19, 0.062,
    0.33, 1.37, 2.33, 27.2, 0.0075,
    0.016, 0.015, 0.010, 0.012, 0.0031,
    14.8, 28.6, 92.3, 632.1, 0.17,
    0.32, 0.26, 0.21, 0.38, 0.0943]]
  - Show the probability of prediction (use lr.predict_proba(X))