KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information

2023

인공지능특론
(Advanced Artificial Intelligence)
11주 신경망 소개

Hongsuk Yi KISTI

Multi-Class Classification

입력 데이터와 판다스

KISTI SINCE 1962

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('./vds.csv')
```

In [3]:
```
df.head(3)
```

Out[3]:

	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
0	2017-04-02 0:00	43	34	9	0	50.3	1.90
1	2017-04-02 0:05	45	32	13	0	58.9	1.84
2	2017-04-02 0:10	46	34	12	0	50.6	1.87

In [4]:
```
df.set_index('Date', inplace=True)
```

클래스 레이블링은 3개

KISTI SINCE 1962

3개의 클래스로 레이블링

In [5]:
```
num_classes = 3
class_labels= ['Jam', 'Slow', 'Normal']
```

In [6]:
```
def get_score(speed):
    if speed < 30:
        label = 'Jam'
    elif speed < 50:
        label = 'Slow'
    else :
        label = 'Normal'
    return label
```

레이블링을 추가

In [7]:
```
df["label_speed"] = df["Speed"].apply(lambda spd: get_score(spd))
df.head()
```

Out[7]:

Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate	label_speed
2017-04-02 0:00	43	34	9	0	50.3	1.90	Normal
2017-04-02 0:05	45	32	13	0	58.9	1.84	Normal
2017-04-02 0:10	46	34	12	0	50.6	1.87	Normal
2017-04-02 0:15	45	36	9	0	50.9	1.72	Normal
2017-04-02 0:20	27	13	13	1	62.2	1.12	Normal

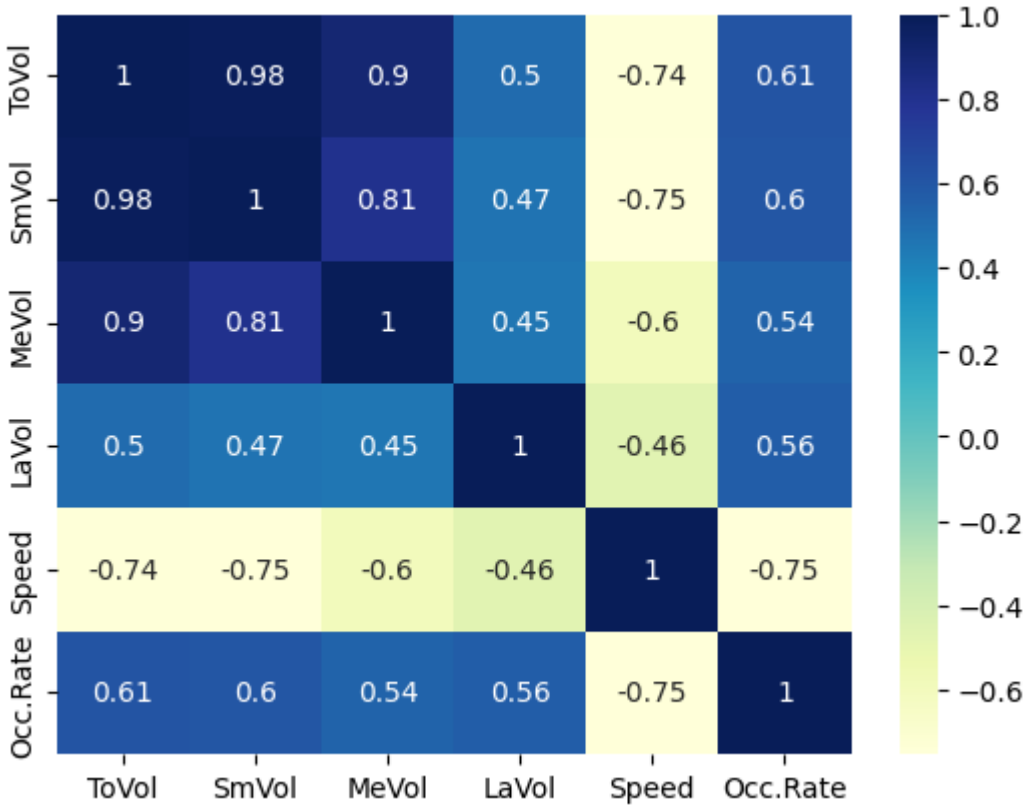
In [8]:
```
df["label_speed"].unique()
```

Out[8]:
```
array(['Normal', 'Slow', 'Jam'], dtype=object)
```

Seaborn의 열지도

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.heatmap(df.corr(), annot = True,  cmap='YlGnBu')
```



클래스를 정수로 인코딩

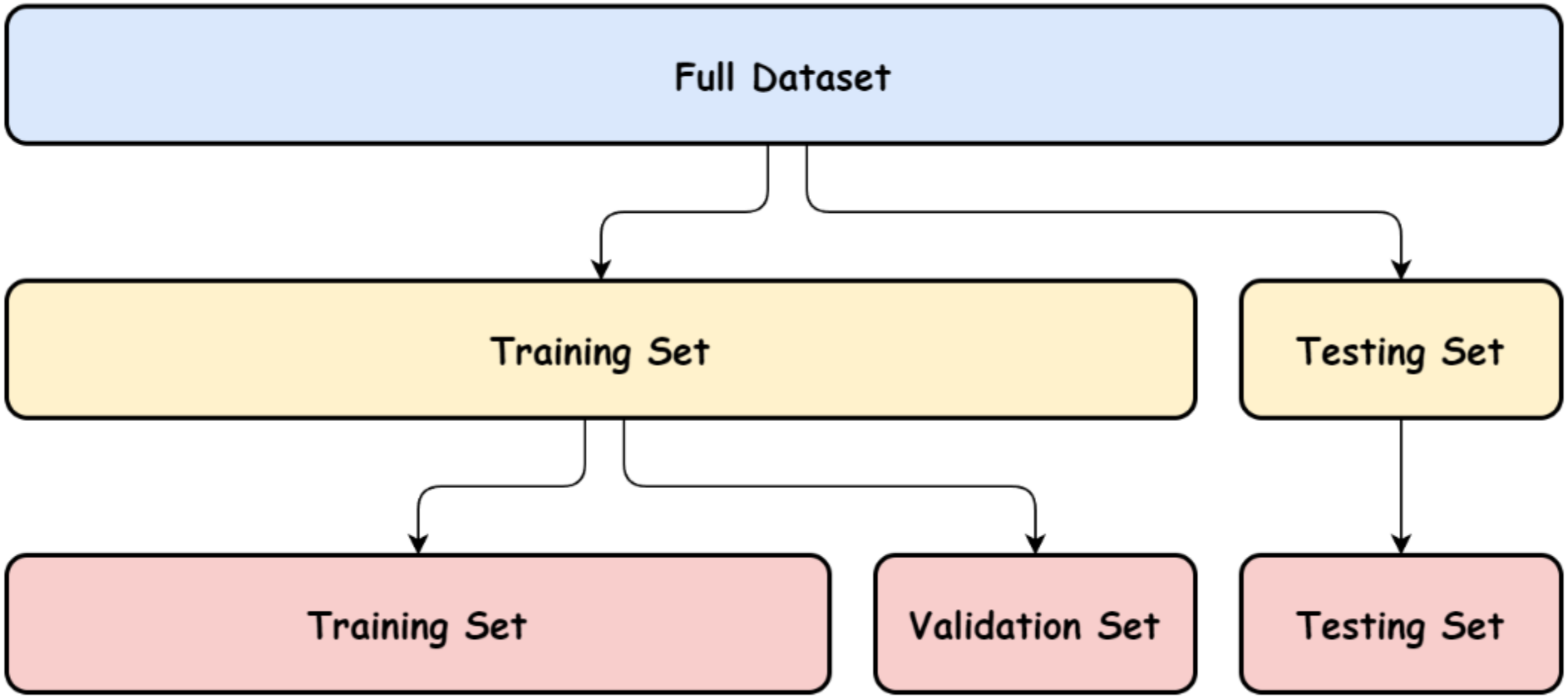
In [11]:
```
class_dic = {'Jam':0, 'Slow':1, 'Normal':2}
#class_dic = {'Jam':0, 'Normal':1}
df['label_speed'] = df['label_speed'].apply(lambda i : class_dic[i])
df.head(2)
```

Out[11]:

Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate	label_speed
2017-04-02 0:00	43	34	9	0	50.3	1.90	2
2017-04-02 0:05	45	32	13	0	58.9	1.84	2

- Speed는 레이블 만들때 사용해서 feature에서 제외해보자

Splitting Your Data: Training, Testing, and Validation Datasets



Sklearn을 이용한 데이터 분할

In [12]:
```
features = ['ToVol','LaVol','MeVol','SmVol','Occ.Rate']
X = df[features]
y = df['label_speed']
```

In [13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,shuffle=False)
print(X_train.shape, y_train.shape)
print(X_test.shape,  y_test.shape)
```

```
(6451, 5) (6451,)
(1613, 5) (1613,)
```

입력 데이터를 표준화하자

KISTI SINCE 1962

In [14]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [15]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [16]:
```
print(X_train.shape)
```

```
(6451, 5)
```

In [ ]:
```
print(X_train[1].shape)
```

In [17]:
```
num_features = len(X_train[1])
print('number of features :', num_features)
```

```
number of features : 5
```

Tensorflow 설정

Deep Neural Network ¶

In [18]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential, optimizers
from tensorflow.keras.layers import Flatten, Dense, Softmax
```

- binary classification 문제로, 마지막 Dense층에는 'sigmoid'를 사용해야한다.

멀티클래스 MLP 모델 (#2)

KISTI SINCE 1962

```
num_neurons = 20
def model_mlp_softmax():
    model = Sequential([
        Dense(num_neurons, activation='relu', input_shape=[num_features]),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

모델 파라미터

In [20]:
```
print('number of features (Input) : ',num_features)
print('number of classes (Output) : ', num_classes)
print('number of neurons (Hidden) : ', num_neurons)
```

```
number of features (Input) :  5
number of classes (Output) :  3
number of neurons (Hidden) :  20
```

In [21]:
```
print('(x+b)*w :', (num_features+1)*num_neurons)
print('(neurons+1)*classes:', (num_neurons+1)*num_classes)
```

```
(x+b)*w : 120
(neurons+1)*classes: 63
```

Model.summary()

KISTI SINCE 1962

In [22]:

```
model = model_mlp_softmax()
model.summary()
```

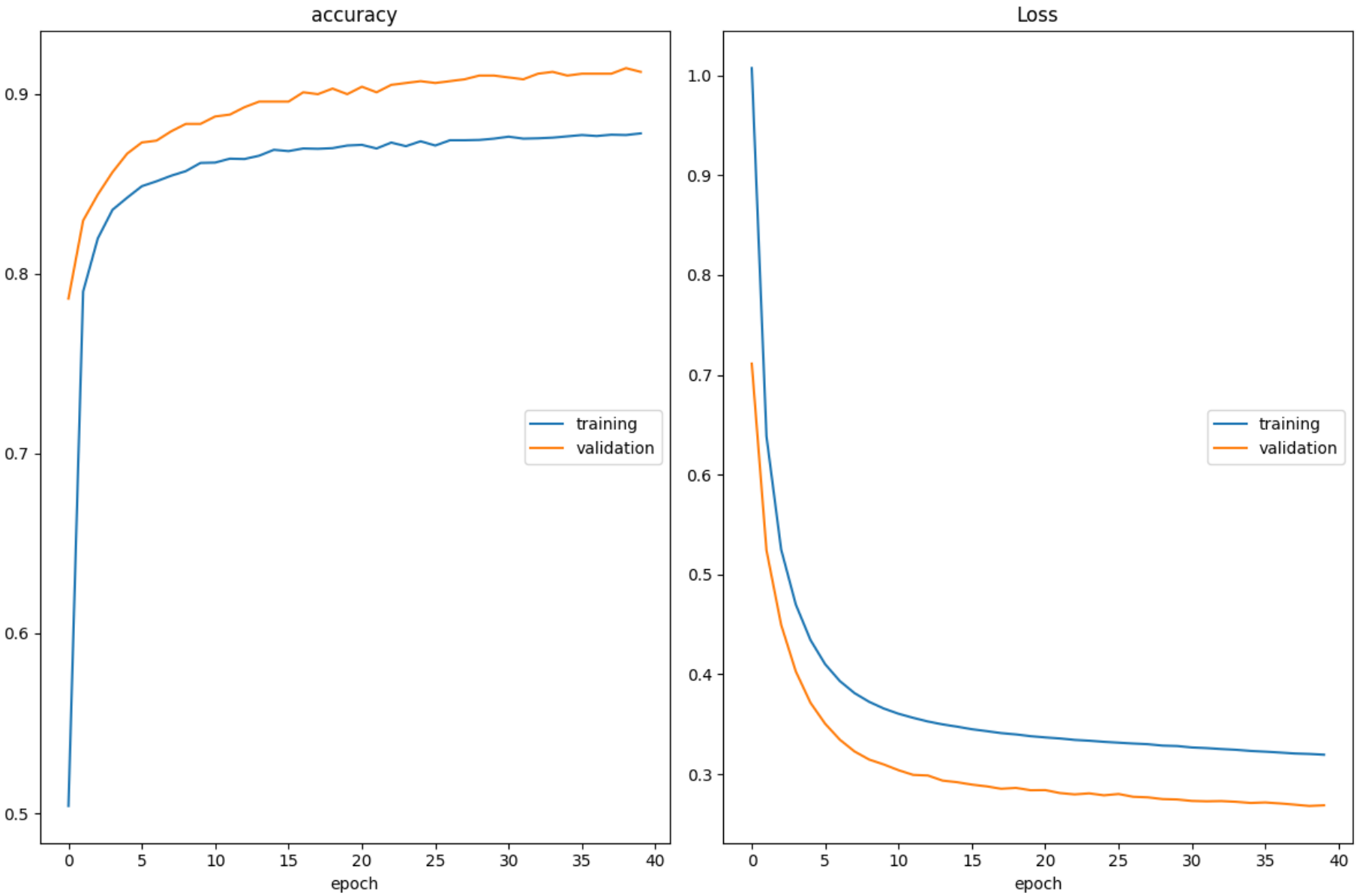
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                120

 dense_1 (Dense)             (None, 3)                 63

=================================================================
Total params: 183
Trainable params: 183
Non-trainable params: 0
_________________________________________________________________
```

모델 훈련

In [23]:
```
# !pip install livelossplot
from livelossplot import PlotLossesKeras
```

In [24]:
```
history = model.fit(X_train,y_train, epochs=40,
                    validation_split=0.15,
                    batch_size=64,
                    callbacks=[PlotLossesKeras()])
```

PlotLossKeras



정확도 (validation vs test)

```
accuracy
        training                (min:    0.504, max:    0.878, cur:    0.878)
        validation              (min:    0.786, max:    0.914, cur:    0.912)
Loss
        training                (min:    0.319, max:    1.007, cur:    0.319)
        validation              (min:    0.268, max:    0.711, cur:    0.269)
86/86 [==============================] - 1s 9ms/step - loss: 0.3195 - accuracy: 0.8780 - val_loss: 0.2689 - val_accuracy: 0.9122
```

```
model.evaluate(X_test,y_test)
```

```
51/51 [==============================] - 0s 2ms/step - loss: 0.3263 - accuracy: 0.8785
```

```
[0.32632961869239807, 0.8784872889518738]
```

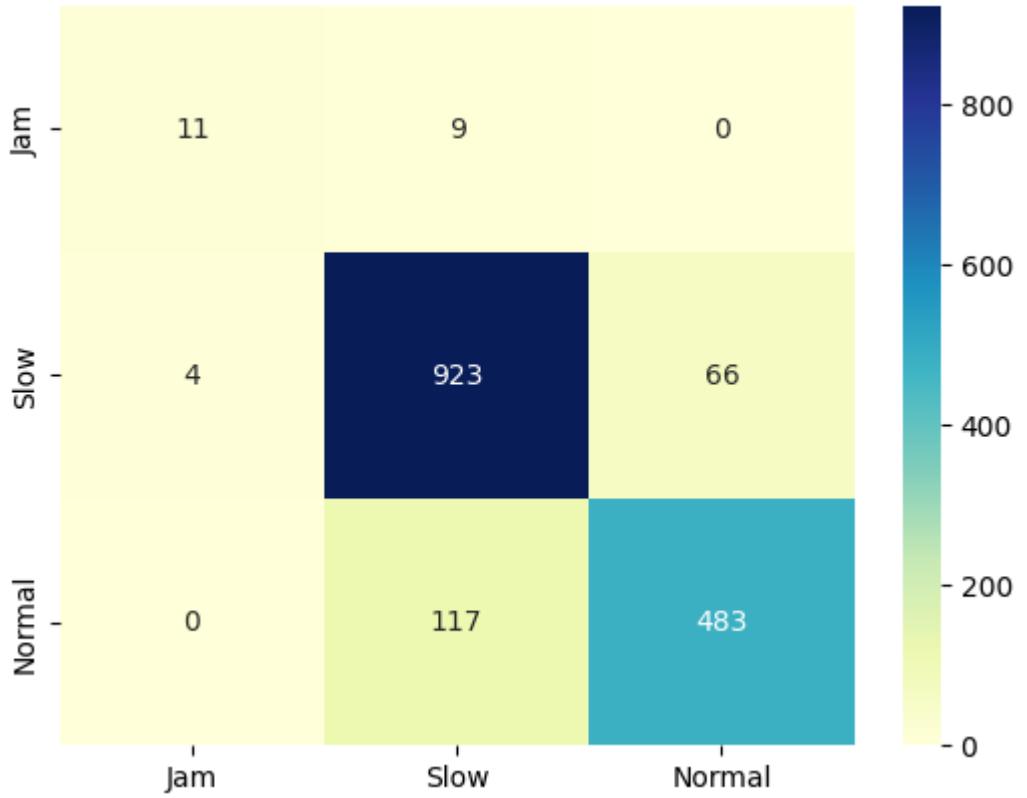
Confusion matrix

```
y_pred = model.predict(X_test)
```

```
51/51 [==============================] - 0s 2ms/step
```

```
from sklearn.metrics import confusion_matrix
import numpy as np
y_pred_labels = [np.argmax(label) for label in y_pred]
cm = confusion_matrix(y_test,y_pred_labels)
sns.heatmap(cm , annot = True,  cmap='YlGnBu',
            fmt = 'd',xticklabels = class_labels, yticklabels = class_labels)
```

Confusion matrix heatmap



교통 흐름 MLP 선형회귀 모델

회귀(Regression) 소개(1)

❖ 회귀의 역사

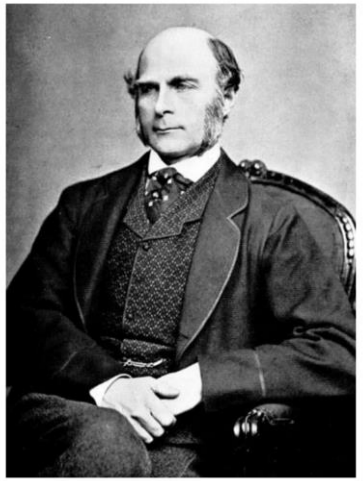
✓영국의 통계학자 갈톤(Galton)의 유전적 특성중에 부모와 자식의 키 관계
✓"사람의 키는 평균 키로 회귀(Regression)하려는 경향을 가진다는 자연의 법칙이 있다"
✓회귀 분석은 데이터 값이 평균과 같은 일정한 값으로 돌아가려는 경향을 이용한 통계학 기법

❖ 지도학습은 2가지 유형으로 나눔

✓회귀는 연속적인 숫자 값
✓분류는 예측값이 카테고리와 같은 이산형 클래스 값

❖ 선형회귀

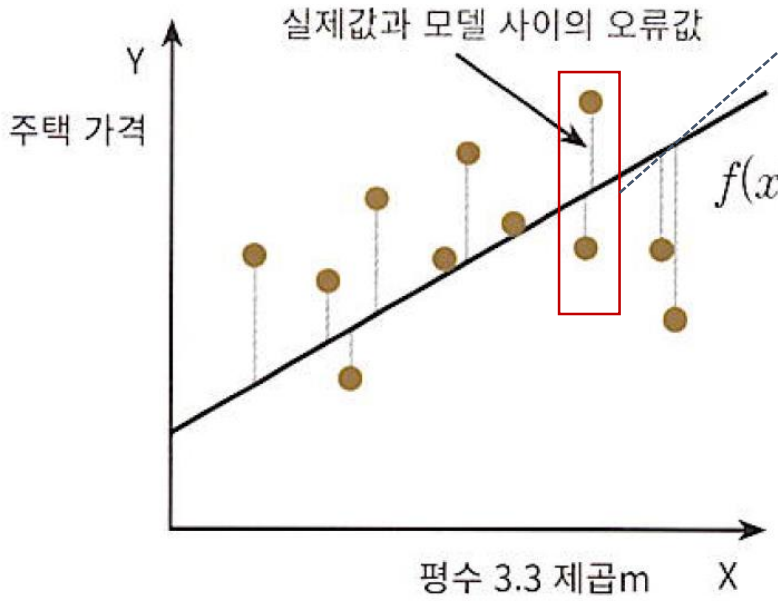
✓실제 값과 예측 값의 차이를 최소화하는 직선형 회귀선을 최적화하는 방식
✓단순 선형회귀는 1개의 독립변수, 1개의 종속변수.
✓예) 주택 가격이 주택의 크기로만 결정된다고 해보면,



Sir Francis Galton (1822 ~ 1911)

회귀(Regression) 소개

❖ 단순 선형회귀를 통한 회귀의 이해 : 단순 선형회귀는 1개의 독립변수, 1개의 종속변수



오차는 양수(+) 혹은 음수(-) 가능하므로, 절대값을 취하거나(MAE), 오차 값의 제곱 (MSE)를 사용.
보통 RSS(Residual Sum of Square)~(오차)의 제곱

$$f(x) = w_0 + w_1 * x$$

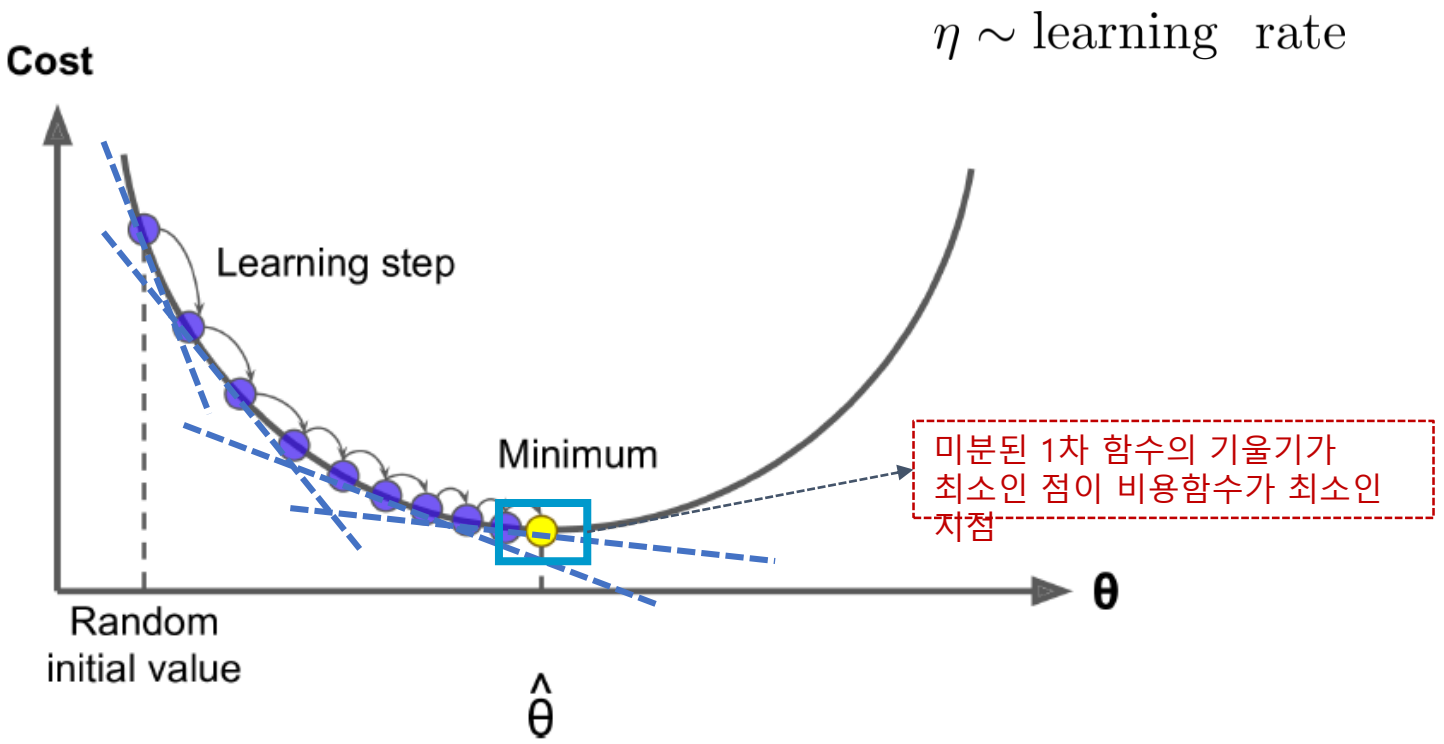
RSS = (#1 주택 가격 – $(w_0+w_1$ #1 주택크기$)^2$
+ (#2 주택 가격 – $(w_0+w_1$ #2 주택크기$)^2$
+ (#3 주택 가격 – $(w_0+w_1$ #3 주택크기$)^2$
+ … (모든 학습 데이터에 대해 RSS 수행)

회귀에서 RSS는 비용(cost)이며, 혹은 비용함수,
혹은 손실(loss) 함수라고도 함

$$\mathrm{RSS}(w_0, w_1) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_o + w_1 \times x_i))^2$$

비용 최소화 _ 경사 하강법(1)

❖ 경사 하강법 (Gradient Descent)



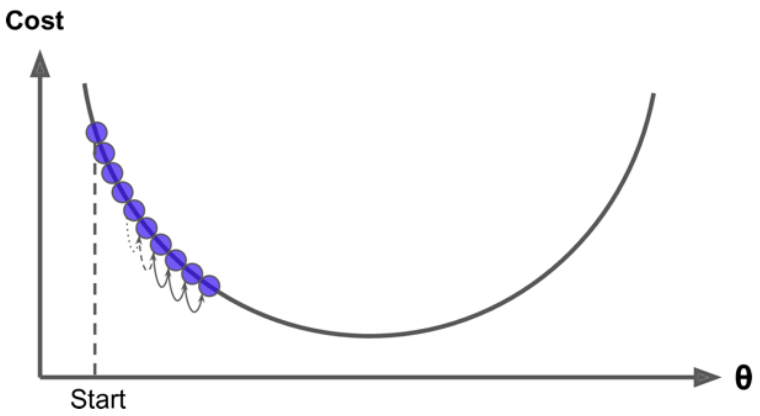
비용 최소화 _ 경사 하강법(2)

❖ 학습률

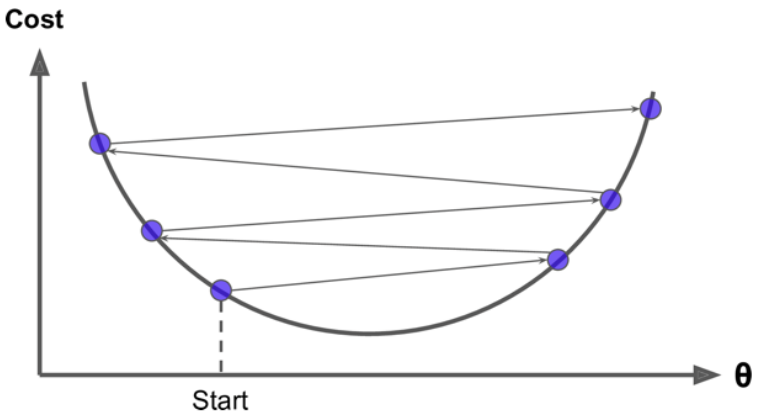
$$w = w - \eta \frac{2}{N} \sum_{i=1}^{N} (\text{real}_i - \text{pred}_i)$$

학습률 도입

$\eta \sim$ learning rate



a) too small

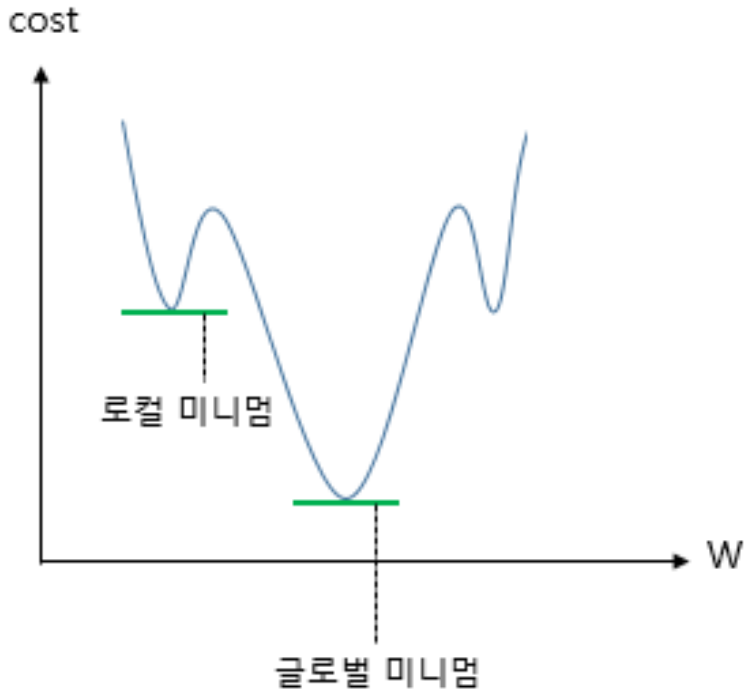


a) too big

로지스틱 비용 함수(Cost function)

❖ 로컬미니멈의 위험

- ✓로지스틱 회귀 또한 경사 하강법을 사용하여 가중치를 찾아내지만, 비용 함수로는 평균 제곱 오차를 사용하지 않습니다.
- ✓이유는 시그모이드 함수에 비용 함수를 평균 제곱 오차로 하여 그래프를 그리면 다음과 비슷한 형태가 되기 때문입니다.

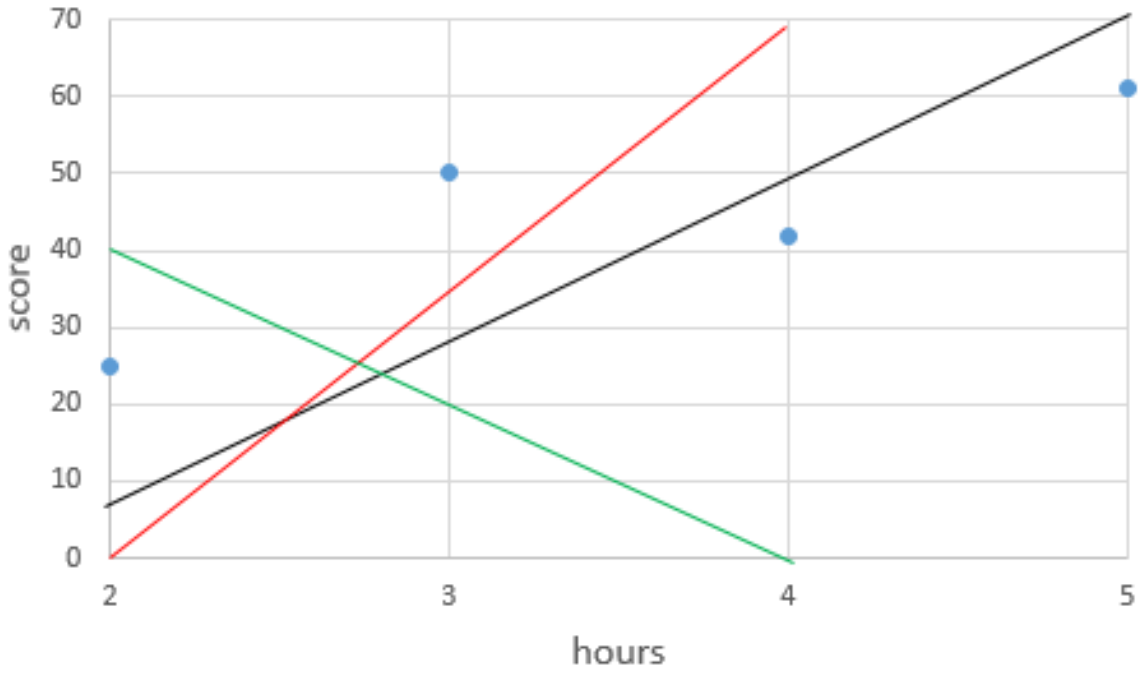


가설(Hypothesis) 세우기

- 어떤 학생의 공부 시간에 따라서 얻은 점수 데이터
- 예측을 위한 선형 가설을 세우자

$$H(x) = Wx + b$$

hours(x)	score(y)
2	25
3	50
4	42
5	61



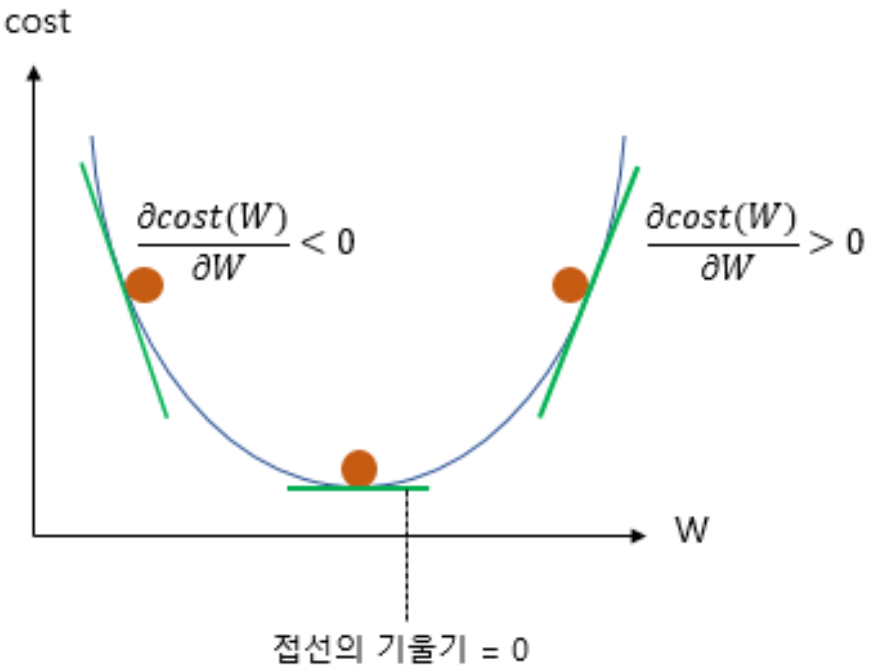
4. 옵티마이저 : 경사하강법(Gradient Descent)

❖ 옵티마이저(Optimizer) 또는 최적화 알고리즘

✓머신 러닝 학습은 결국 비용 함수를 최소화하는 가중치, 편향을 찾기 위한 작업을 수행

$$w_{\text{updated}} \approx w - \alpha \cdot \frac{\partial \text{cost}}{\partial w}$$

$$W, b \rightarrow minimize\ cost(W, b)$$



(lab) 케라스로 구현하는 선형 회귀

- 케라스로 모델을 만드는 기본적인 형식

- model = keras.models.Sequential()
  - ✓Seauential로 모델을 이라는 이름을 만들고
- model.add(keras.layers.Dense(1, input_dim=1))
  - ✓add를 통해서 필요한 사항을 추가해 나간다.
  - ✓첫 인자 1은 출력의 차원
  - ✓두 번째 인자 input_dim은 입력의 차원을 정의

케라스로 구현하는 선형 회귀

KISTI SINCE 1962

[2]
```
import tensorflow as tf
import numpy as np
print(tf.__version__)
```

```
2.8.0
```

[3]
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers
```

[4]
```
X=np.array([1,2,3,4,5,6,7,8,9])
# 공부하는 시간
y=np.array([12,25,50,42,61, 67, 79, 85, 90])
# 각 공부하는 시간에 맵핑되는 성적
```

케라스로 구현하는 선형 회귀

```
[5] model = Sequential()

    model.add(Dense(1, input_dim=1, activation='linear'))
```

```
[6] # sgd는 경사 하강법을 의미.
    #학습률(learning rate, lr)은 0.01.
    sgd = optimizers.SGD(lr=0.01)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:102:
  super(SGD, self).__init__(name, **kwargs)
```

케라스로 구현하는 선형 회귀

회귀에서

- 손실함수는 오차 MSE을 주로 사용한다.

```
[7] # 손실 함수(Loss function)은 평균제곱오차 mse를 사용합니다.
    model.compile(optimizer=sgd ,loss='mse',metrics=['mse'])
```

```
[8] # 주어진 X와 y데이터에 대해서 오차를 최소화하는 작업을 300번 시도합니다.
    history=model.fit(X,y, batch_size=1, epochs=30, shuffle=False)
```

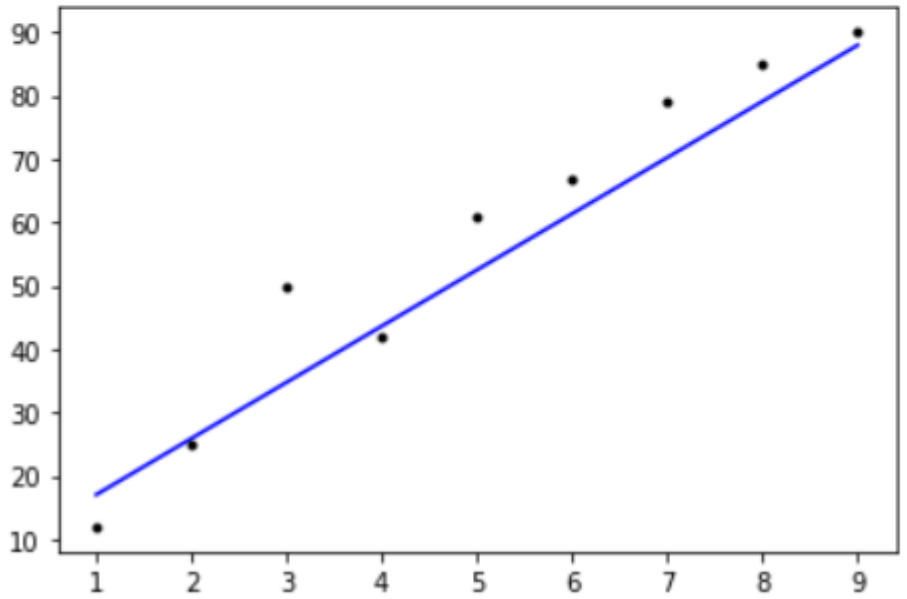
케라스로 구현하는 선형 회귀

```
%matplotlib inline
```

```
import matplotlib.pyplot as plt
```

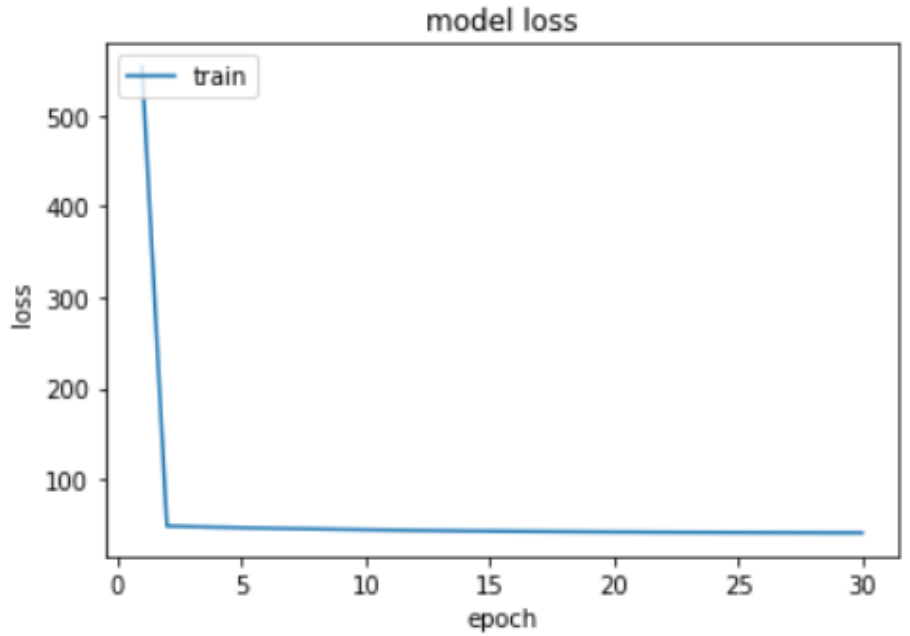
```
plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```

```
[<matplotlib.lines.Line2D at 0x7fda5a760a90>,
 <matplotlib.lines.Line2D at 0x7fdad0150490>]
```



케라스로 구현하는 선형 회귀

```
epochs = range(1, len(history.history['mse']) + 1)
plt.plot(epochs, history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



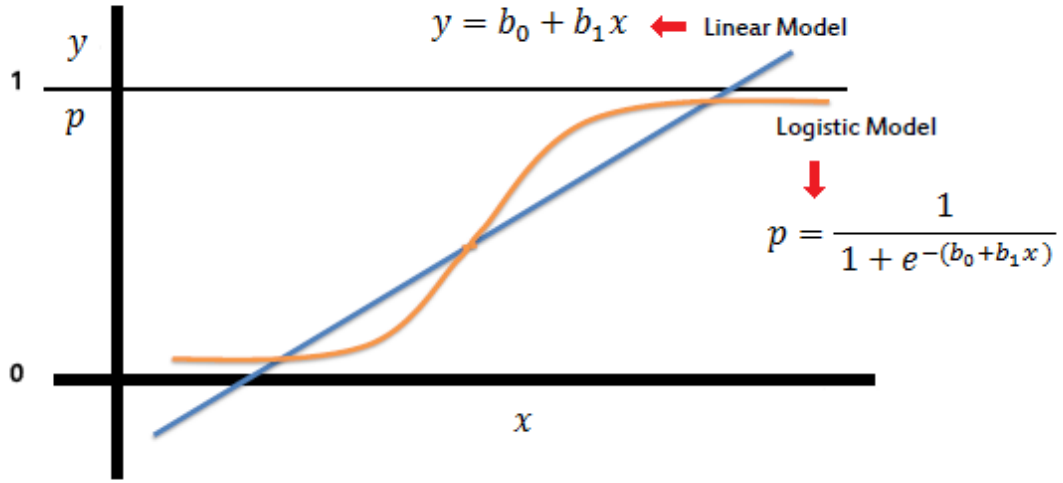
TF로 비선형회귀

로지스틱 회귀(Logistic Regression)

❖ 로지스틱 회귀 개요

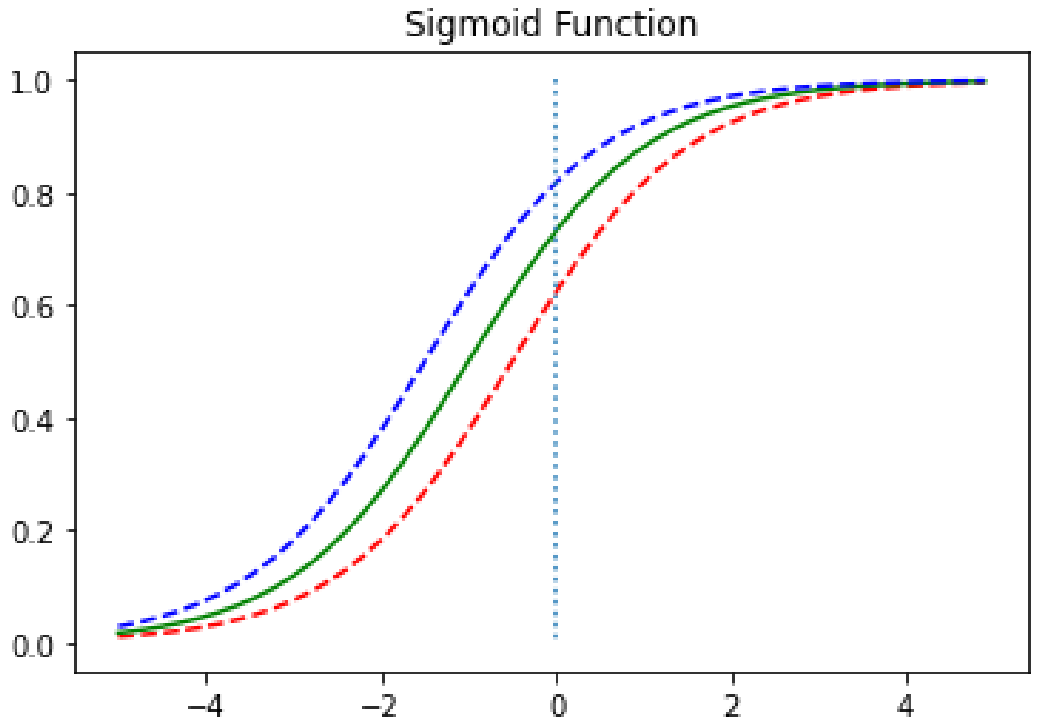
✓둘 중 하나를 결정하는 문제를 이진 분류(Binary Classification)라고 합니다.

- 이런 문제를 풀기 위한 알고리즘으로 로지스틱 회귀(Logistic Regression)가 있다.



시그모이드 함수(Sigmoid function)

$$H(X) = \frac{1}{1+e^{-(Wx+b)}} = \text{sigmoid}(Wx+b) = \sigma(Wx+b)$$




```
# x + 0.5

# x + 1.5
```

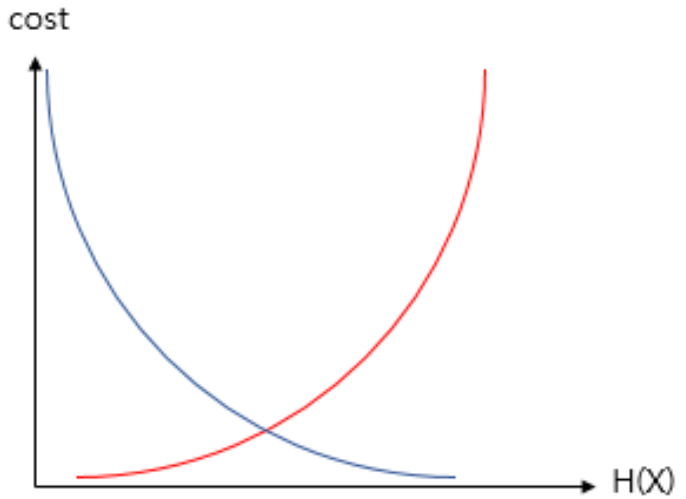
e=2.718281… 자연 상수

로지스틱 비용 함수(Cost function)

- 목적 함수(objective function)

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} cost\left(H(x^{(i)}), y^{(i)}\right)$$

$$H(X) = \frac{1}{1+e^{-(Wx+b)}} = \sigma(Wx+b)$$



- 크로스 엔트로피 함수 (Cross Entropy)
  - ✓로지스틱 회귀에서 찾아낸 비용함수를 말한다

$$\text{if } y = 1 \rightarrow \text{cost}\left(H(x), y\right) = -\log(H(x))$$

$$\text{if } y = 0 \rightarrow \text{cost}\left(H(x), y\right) = -\log(1 - H(x))$$

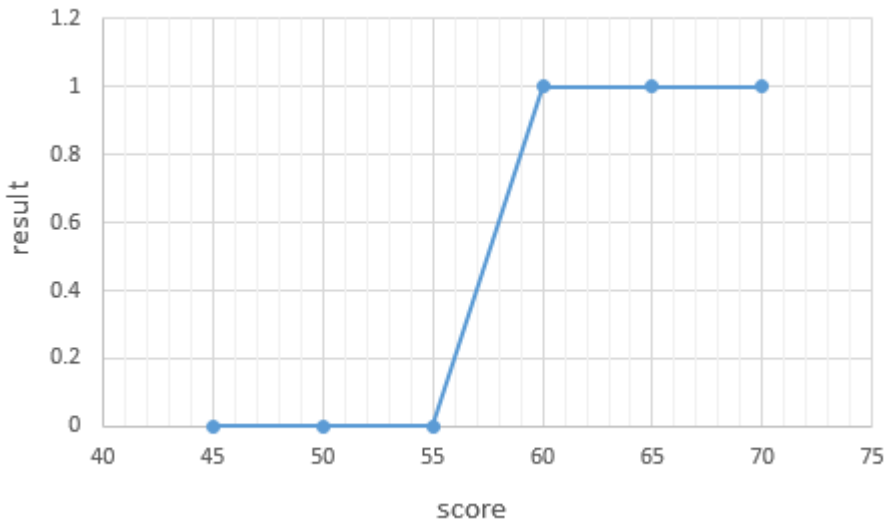
$$J(W) = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)} log H(x^{(i)}) + (1-y^{(i)}) log(1 - H(x^{(i)}))]$$

$$\text{cost}\left(H(x), y\right) = -[y log H(x) + (1-y) log(1 - H(x))]$$

이진 분류(Binary Classification)

- 예제: 학생들이 시험 성적에 따라서 합격, 불합격이 기재된 데이터가 있다고 가정
  - ✓시험 성적이 x라면, 합불 결과는 y입니다.
  - ✓이 시험의 커트라인은 공개되지 않았는데 이 데이터로부터 특정 점수를 얻었을 때의 합격, 불합격 여부를 판정

score(x)	result(y)
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격



Tensorflow 2.x로 시작하는 로지스틱 회귀

```
[1] import numpy as np
```

```
[2] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras import optimizers
```

```
[3] X=np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50])
    y=np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```

```
[4] model=Sequential()
```

```
[5] model.add(Dense(1, input_dim=1, activation='sigmoid'))
```

Tensorflow 2.x로 시작하는 로지스틱 회귀

```
[6] sgd=optimizers.SGD(lr=0.01)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:102: UserWarning: The
  super(SGD, self).__init__(name, **kwargs)
```

- 회귀의 손실함수는 오차(MSE)를 사용해야 하지만, 로지스틱은 특별히 Accuracy를 자주 쓴다.

```
[7] model.compile(optimizer=sgd ,
                  loss='binary_crossentropy',metrics=['binary_accuracy'])
```

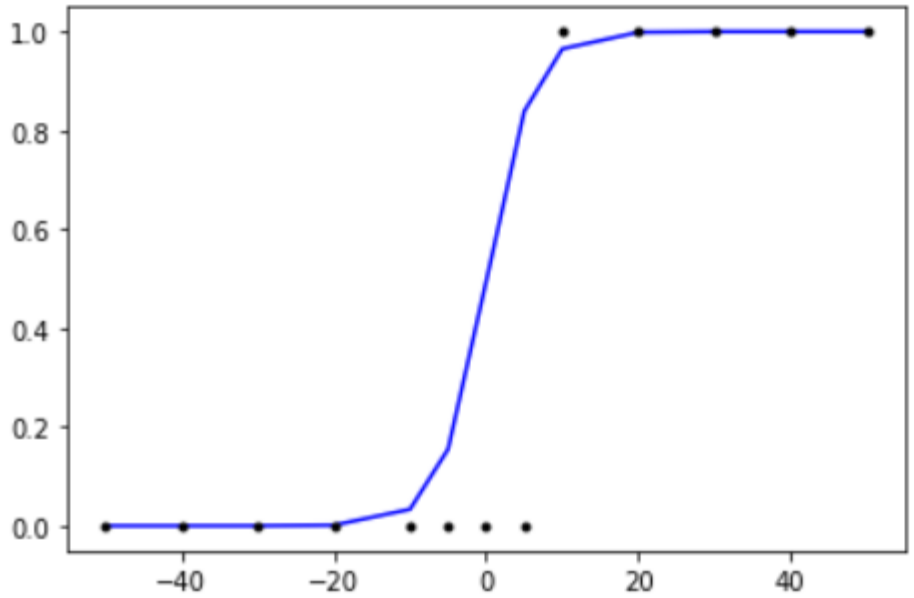
```
# 옵티마이저는 경사하강법 sgd를 사용합니다.
# 손실 함수(Loss function)는 binary_crossentropy를 사용합니다.
history = model.fit(X,y, epochs=20, shuffle=False)
```

Tensorflow 2.x로 시작하는 로지스틱 회귀

```
[9] import matplotlib.pyplot as plt
    %matplotlib inline
```

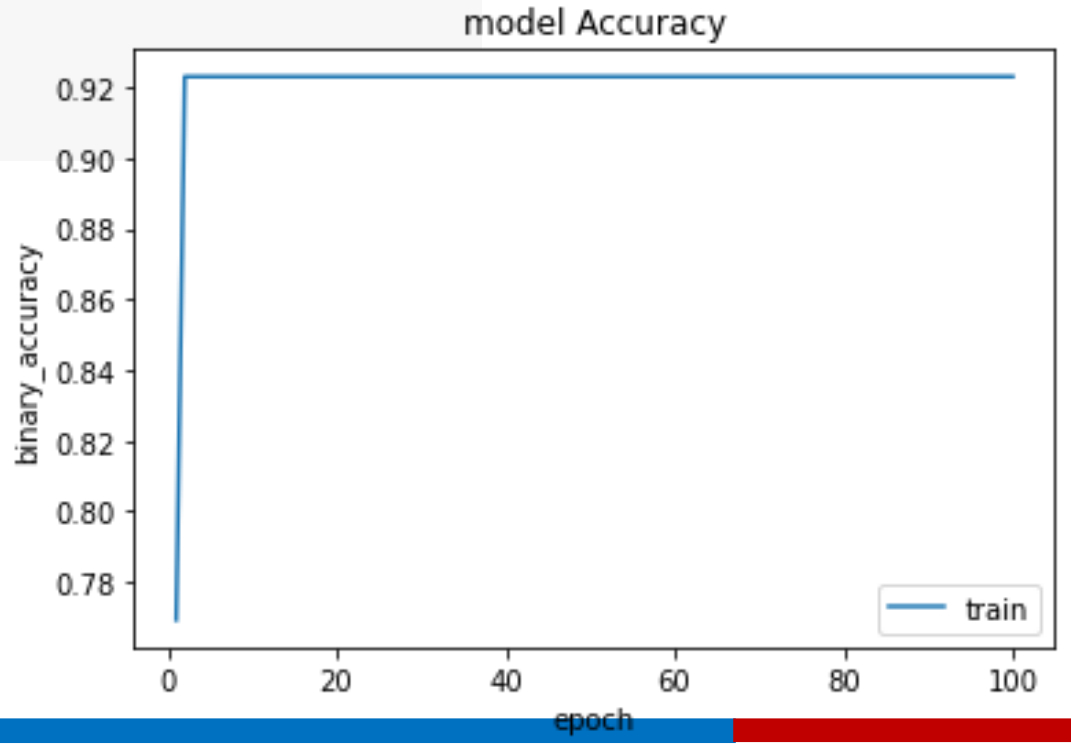
```
plt.plot(X, model.predict(X), 'b', X,y, 'k.')
```

```
[<matplotlib.lines.Line2D at 0x7f1c7c723950>,
 <matplotlib.lines.Line2D at 0x7f1c7af7da10>]
```



Tensorflow 2.x로 시작하는 로지스틱 회귀

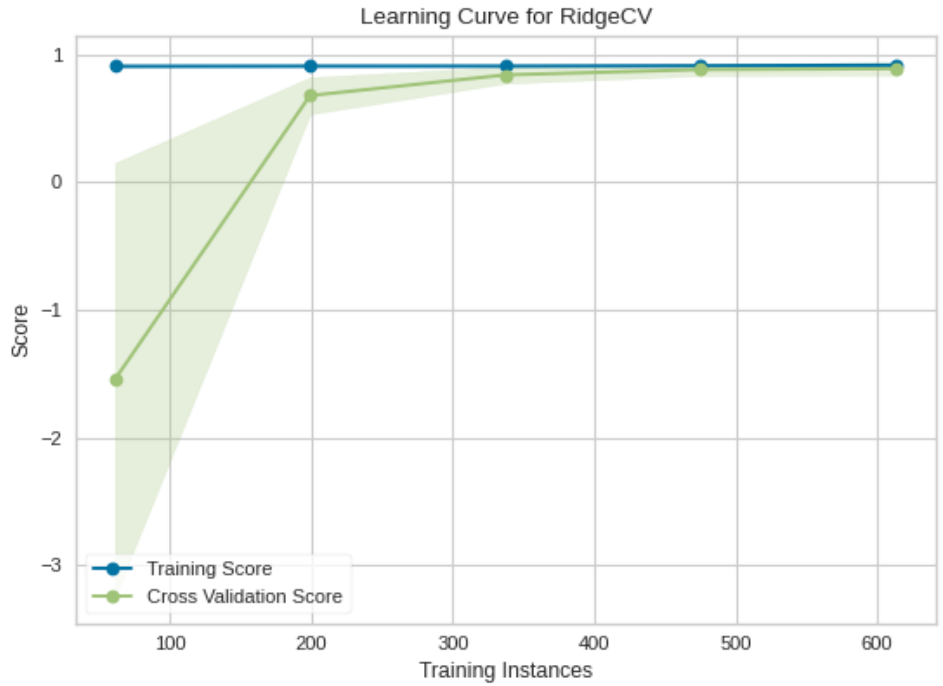
```
epochs = range(1,
               len(history.history['binary_accuracy']) + 1)
plt.plot(epochs, history.history['binary_accuracy'])
plt.title('model Accuracy')
plt.ylabel('binary_accuracy')
plt.xlabel('epoch')
plt.legend(['train'])
plt.show()
```



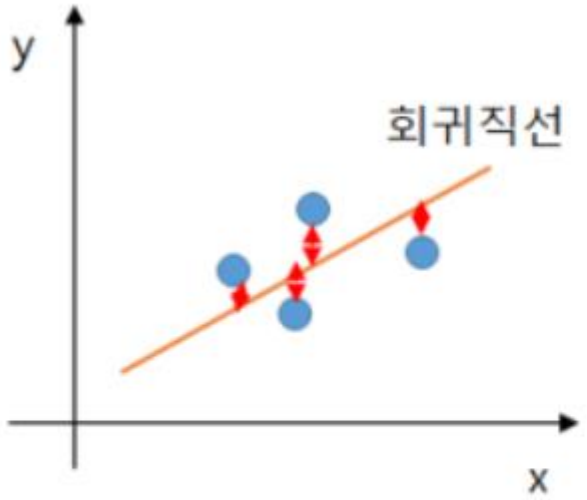
sklearn learning curve : Regression에서 R2 스코어 사용

- sklearn에서 learning_curve의 score는 높은 값이 좋은 성능으로 표현함.
  - ✓회귀에서 MSE 보다는 Negative MSE를 사용함
  - ✓R2도 사용함
- sklearn에서 Regression learning_curve의 score는 deafult로 R2를 사용함.

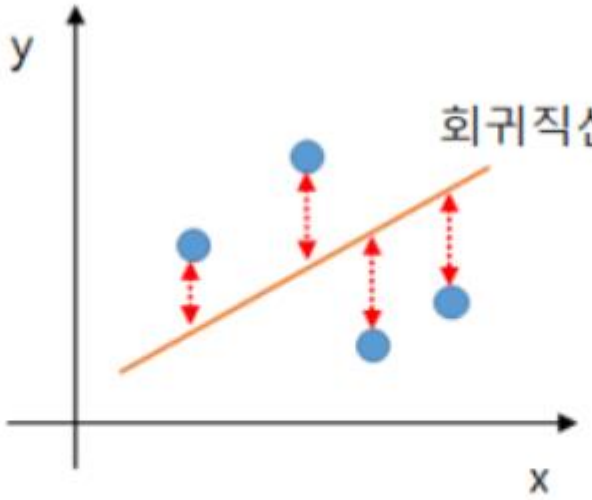
$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$



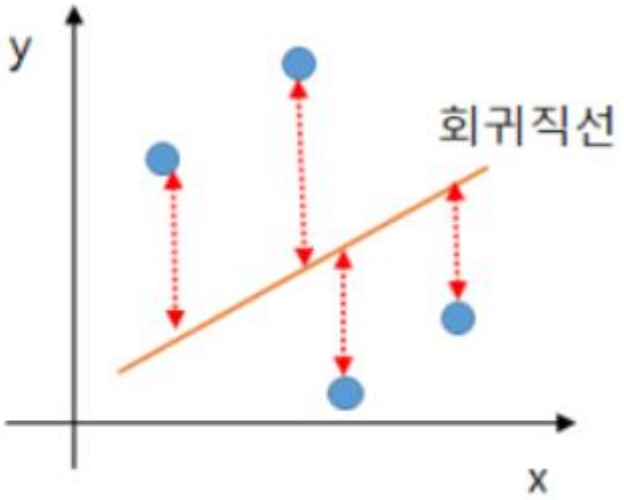
회귀의 평가지표(default): 설명력(결정계수, R2)



R^2가 1에 가까운 경우



R^2가 0.5에 가까운 경우



R^2가 0에 가까운 경우

딥러닝 교통 회귀 문제

코랩에서 github 데이터 읽어오기

```
import pandas as pd
```

```
df = pd.read_csv('https://raw.githubusercontent.com/hongsukyi/Lectures/main/vds/vds.csv')
```

```
df.head(2)
```

:

	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
0	2017-04-02 0:00	43	34	9	0	50.3	1.90
1	2017-04-02 0:05	45	32	13	0	58.9	1.84

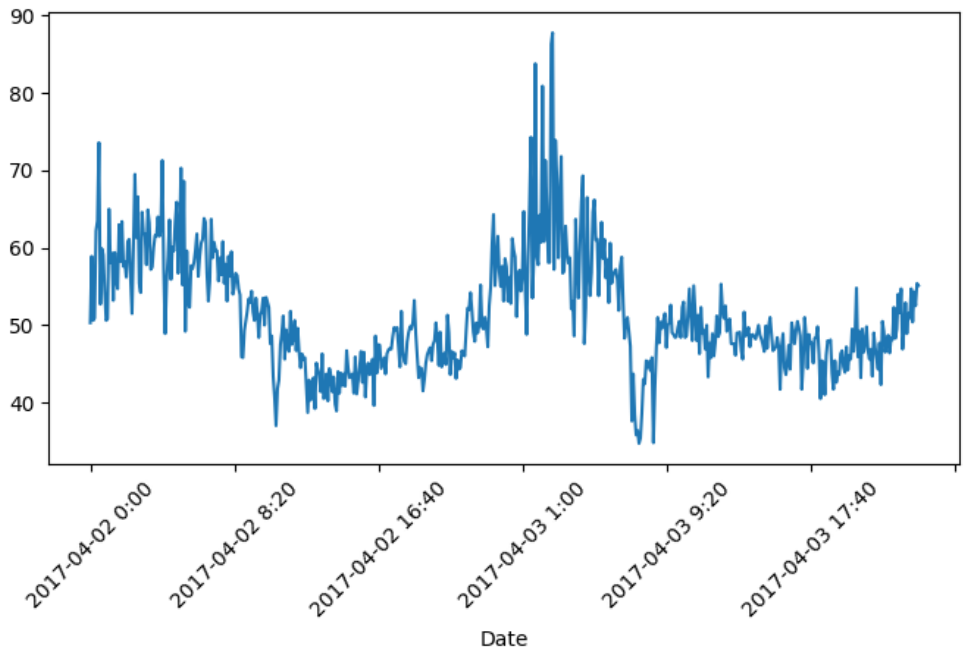
```
df.set_index('Date', inplace=True)
```

```
df.head(2)
```

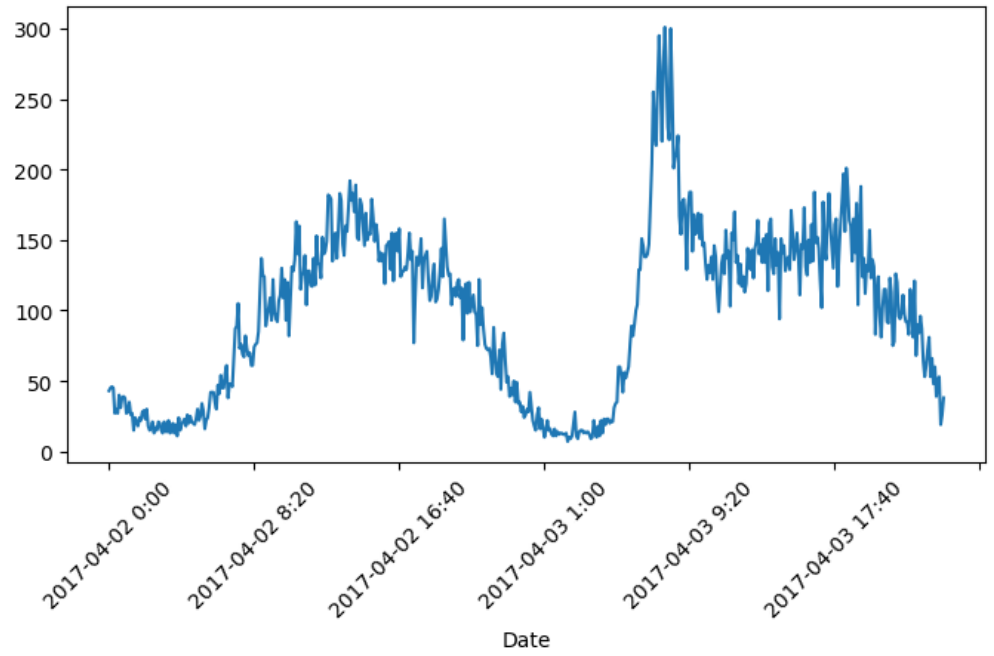
Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
2017-04-02 0:00	43	34	9	0	50.3	1.90
2017-04-02 0:05	45	32	13	0	58.9	1.84

교통량(ToVol)와 차량속도(Speed)

```
df['Speed'][:576].plot(rot=45,figsize=(8,4))
```

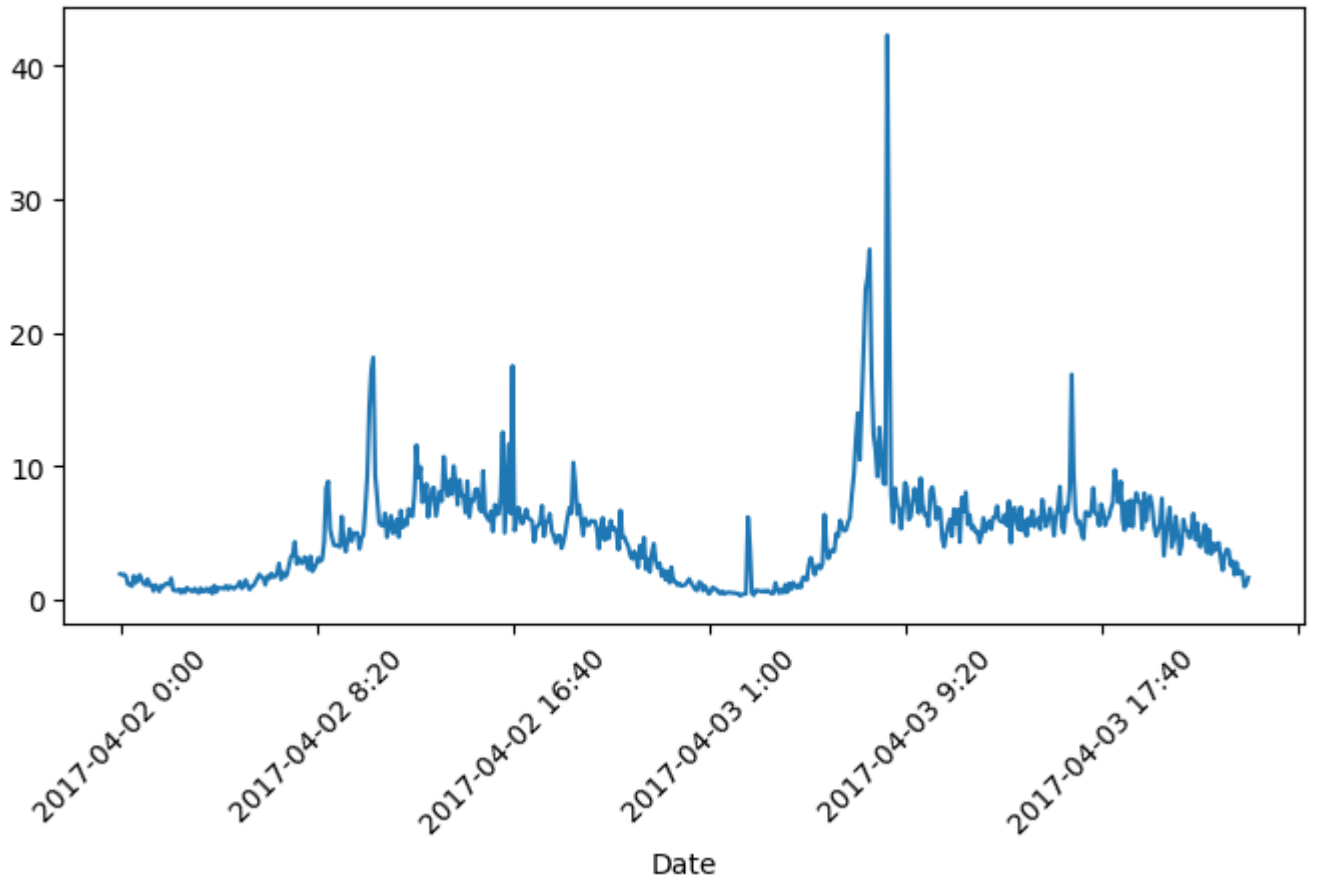



```
df['ToVol'][:576].plot(rot=45,figsize=(8,4))
```



도로 점유율

```
df['Occ.Rate'][:576].plot(rot=45,figsize=(8,4))
```



데이터 전처리를 위하여 sklearn을 사용하자

In [9]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [10]:
```
df.head(2)
```

Out[10]:

Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
2017-04-02 0:00	43	34	9	0	50.3	1.90
2017-04-02 0:05	45	32	13	0	58.9	1.84

In [11]:
```
features = ['ToVol','LaVol','MeVol','SmVol','Occ.Rate']
X = df[features]
y = df.iloc[:,4:5].values
```

최대 속도 확인

In [12]:
```
print(y)
```

```
[[50.3]
 [58.9]
 [50.6]
 ...
 [50.6]
 [59.3]
 [52.5]]
```

In [13]:
```
max_speed = y.max(); max_speed
```

Out[13]: 87.8

정규화

In [14]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
#scaler = StandardScaler()
scaler = MinMaxScaler()
scaled_X = scaler.fit_transform(X)
scaled_y = scaler.fit_transform(y)
```

In [15]:
```
print(scaled_X)
print(scaled_y)
```

```
[[0.11144578 0.         0.06206897 0.12903226 0.02039819]
 [0.11746988 0.         0.08965517 0.12096774 0.01966532]
 [0.12048193 0.         0.08275862 0.12903226 0.02003176]
 ...
 [0.07831325 0.         0.02758621 0.10483871 0.01380237]
 [0.0753012  0.         0.06896552 0.0766129  0.01429095]
 [0.09939759 0.         0.04137931 0.125      0.01844387]]
[[0.52350699]
 [0.63278272]
 [0.52731893]
```

Train과 test를 8:2로 분할

In [16]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_X, scaled_y,
                                                    test_size=0.2, shuffle=False)
print(X_train.shape, y_train.shape)
print(X_test.shape,  y_test.shape)
```

```
(6451, 5) (6451, 1)
(1613, 5) (1613, 1)
```

In [17]:
```
print(X_train)
```

```
[[0.11144578 0.         0.06206897 0.12903226 0.02039819]
 [0.11746988 0.         0.08965517 0.12096774 0.01966532]
 [0.12048193 0.         0.08275862 0.12903226 0.02003176]
```

특성(feature) 정규화

```
print(X_train.shape)
print(X_train[1].shape)
num_features = len(X_train[1])
print('number of features :', num_features)
```

```
(6451, 5)
(5,)
number of features : 5
```

DNN 회귀 모델 만들자

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
n1 = 32
n2 = 32
n3 = 32
n4 = 32
```

```
def dnn_reg1():
    model = Sequential([
        Dense(n1, activation='relu', input_shape=[num_features]),
        Dense(1)
    ])
    model.compile(optimizer='adam',loss= 'mse', metrics=['mae'])
    return model
```

회귀 모델 아키텍처 및 파라미터 구하기

In [28]:
```
model = dnn_reg1()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                192

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 225
Trainable params: 225
Non-trainable params: 0
_________________________________________________________________
```

DNN 회귀 모델

```
def dnn_reg2():
    model = Sequential([
        Dense(n1, activation='relu',
              input_shape=[num_features]),
        Dense(n2, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam',loss= 'm
    return model
```

```
def dnn_reg3():
    model = Sequential([
        Dense(n1, activation='relu',
              input_shape=[num_features]),
        Dense(n2, activation='relu'),
        Dense(n3, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam',loss= 'mse', metrics=['mae'])
    return model
```

DNN 회귀모델

In [27]:

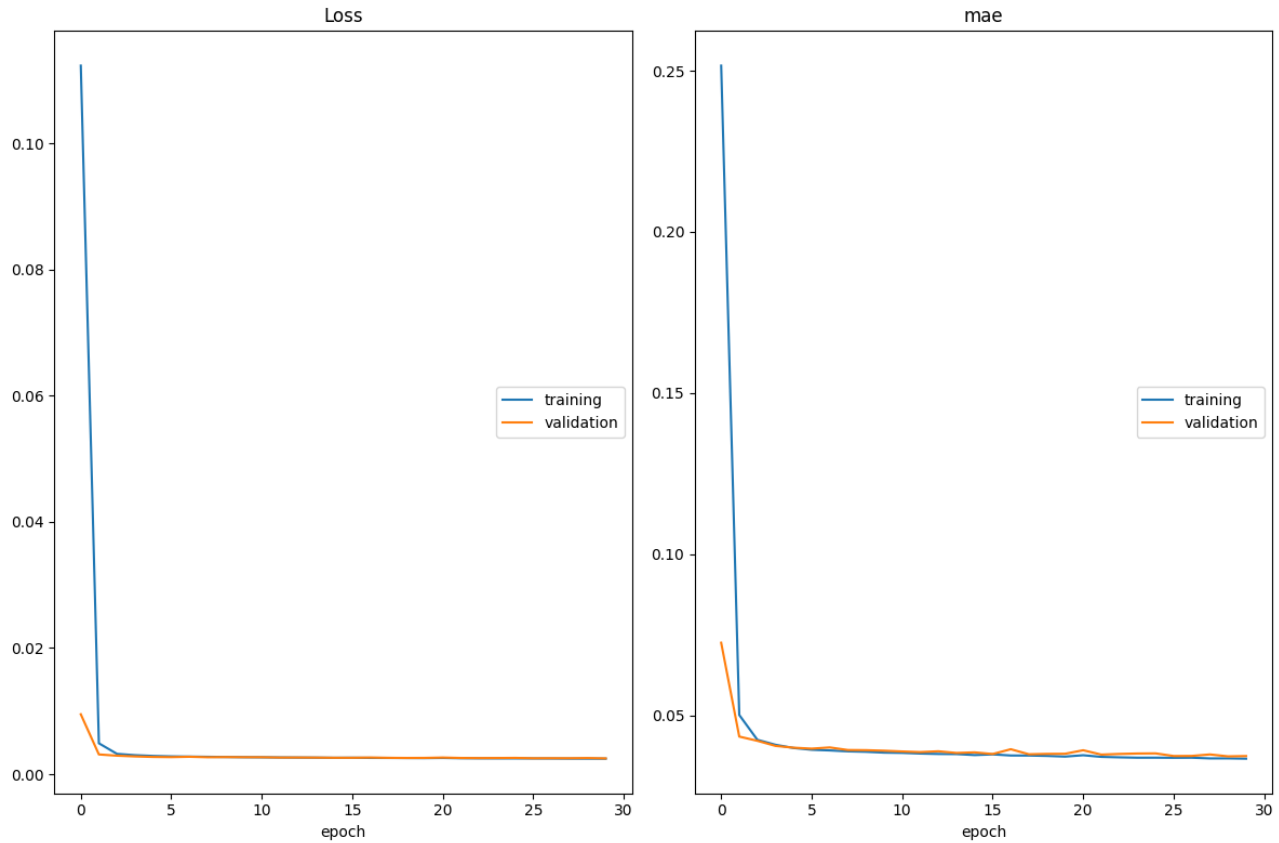
```
def dnn_reg4():
    model = Sequential([
        Dense(n1, activation='relu',
              input_shape=[num_features]),
        Dense(n2, activation='relu'),
        Dense(n3, activation='relu'),
        Dense(n4, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam',loss= 'mse', metrics=['mae'])
    return model
```

모델 훈련 및 실시간 가시화

```
from livelossplot import PlotLossesKeras
```

```
history = model.fit(X_train,y_train, epochs=30, validation_split=0.2, batch_size=32, callbacks=[PlotLossesKeras()])
```

!pip install livelossplot



테스트 데이터로 예측하기

```
Loss
        training                (min:    0.002, max:    0.112, cur:    0.002)
        validation              (min:    0.003, max:    0.009, cur:    0.003)
mae
        training                (min:    0.037, max:    0.252, cur:    0.037)
        validation              (min:    0.037, max:    0.072, cur:    0.037)
162/162 [==============================] - 1s 6ms/step - loss: 0.0025 - mae: 0.0365 - val_loss: 0.0025 - val_mae: 0.0373
```

```
# 테스트 데이터 세트로 모델 평가: Loss 값 확인
mse, mae = model.evaluate(X_test, y_test, verbose=0)
print("\n Neuron1={:3d},  MSE={:8.4f}, MAE={:8.4f}".format(neuron_1,mse,mae))
```

```
 Neuron1= 32,  MSE=  0.0025, MAE=  0.0362
```

훈련과정 Loss와 MAE

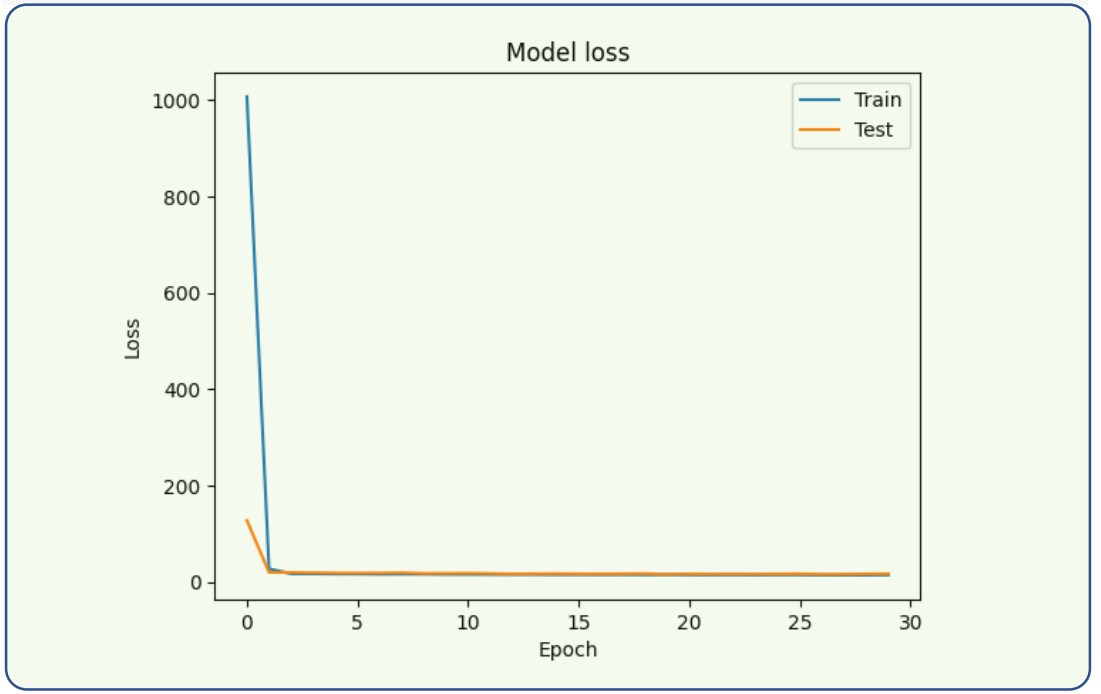
```
history.history.keys()
```

```
dict_keys(['loss', 'mae', 'val_loss', 'val_mae'])
```

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

	loss	mae	val_loss	val_mae	epoch
25	14.801393	2.847822	17.192307	2.992428	25
26	14.521421	2.806103	16.137362	2.960289	26
27	14.246248	2.754538	16.351469	2.987771	27
28	14.358523	2.774629	16.986986	2.998828	28
29	14.627845	2.818771	17.263212	3.201360	29

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper right')
```



속도 예측

In [54]:
```
y_pred = model.predict(X_test, verbose=0)
print(y_pred.shape)
```

```
(1613, 1)
```

In [55]:
```
print('(scaled) predictions : \n', y_pred[:2])
print('\n (scaled) actural values :\n',y_test[:2])
```

```
(scaled) predictions :
 [[0.4665929]
 [0.4981718]]

 (scaled) actural values :
 [[0.50190597]
 [0.54002541]]
```

스케일 이전으로 환원

In [56]:
```
y_inv_pred = scaler.inverse_transform(y_pred)
y_inv_test = scaler.inverse_transform(y_test)
```

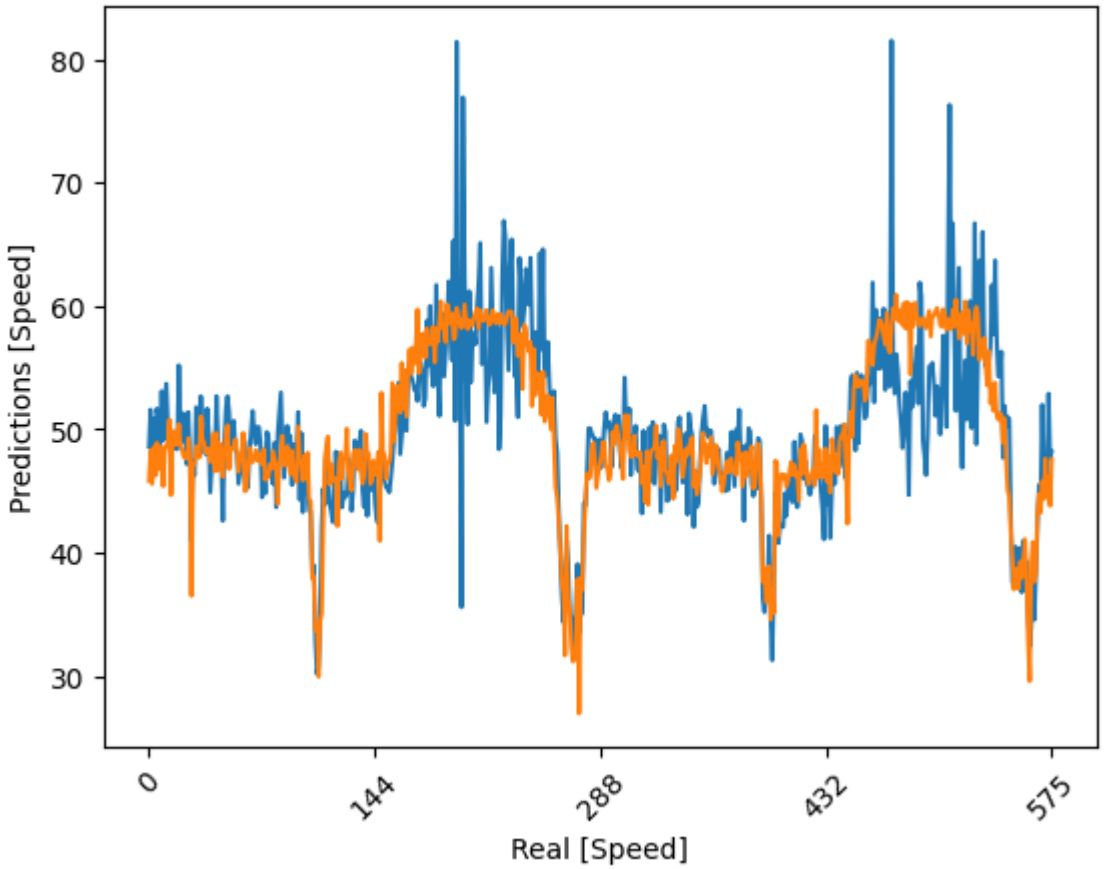
In [57]:
```
print('predictions : \n', y_inv_pred[:2])
print('\nactural values :\n',y_inv_test[:2])
```

```
predictions :
 [[45.82086]
 [48.30612]]

actural values :
 [[48.6]
 [51.6]]
```

2일간 속도 예측 : 대전시 유성구 대학로 (vds16)

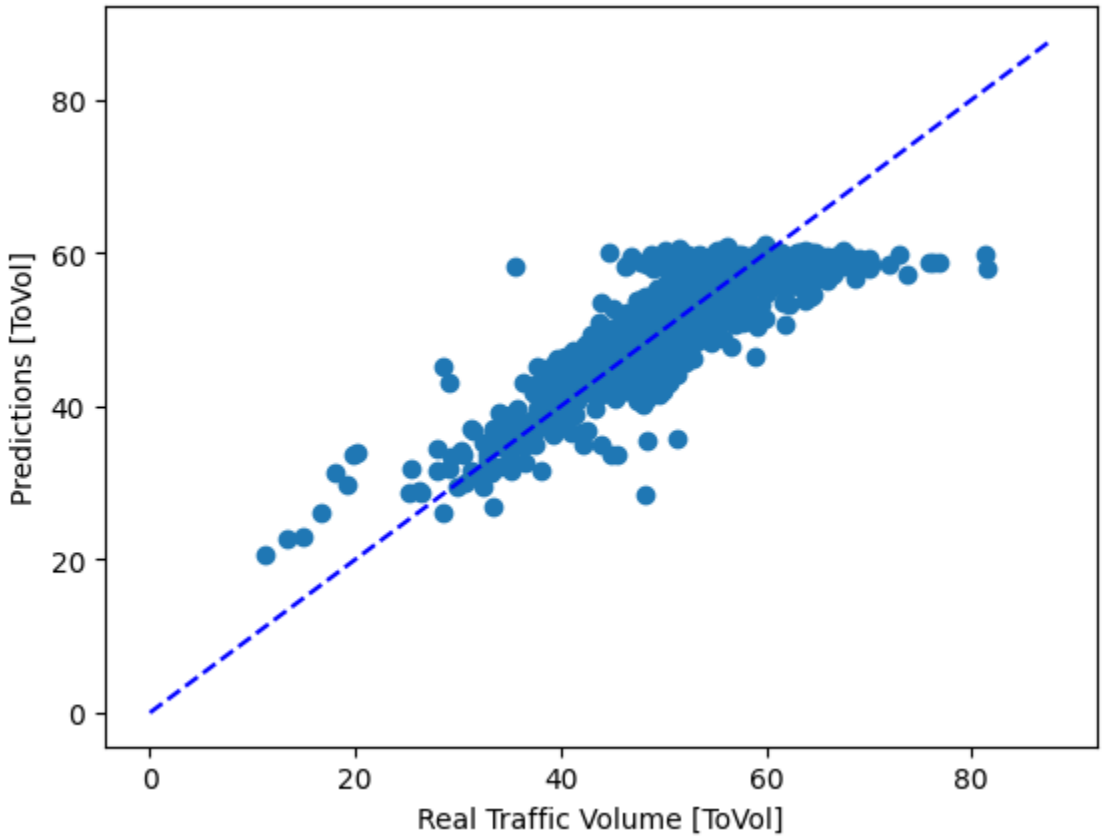
```
# 가시화를 위해서 2차원을 1차원으로
test_prediction = y_inv_pred.flatten()
test_real = y_inv_test.flatten()
```

```
plt.plot(test_real[:576])
plt.plot(test_prediction[:576])
plt.xticks(rotation=45)
plt.xticks([0, 144, 288, 432,575])
plt.xlabel('Real [Speed]')
plt.ylabel('Predictions [Speed]')
```



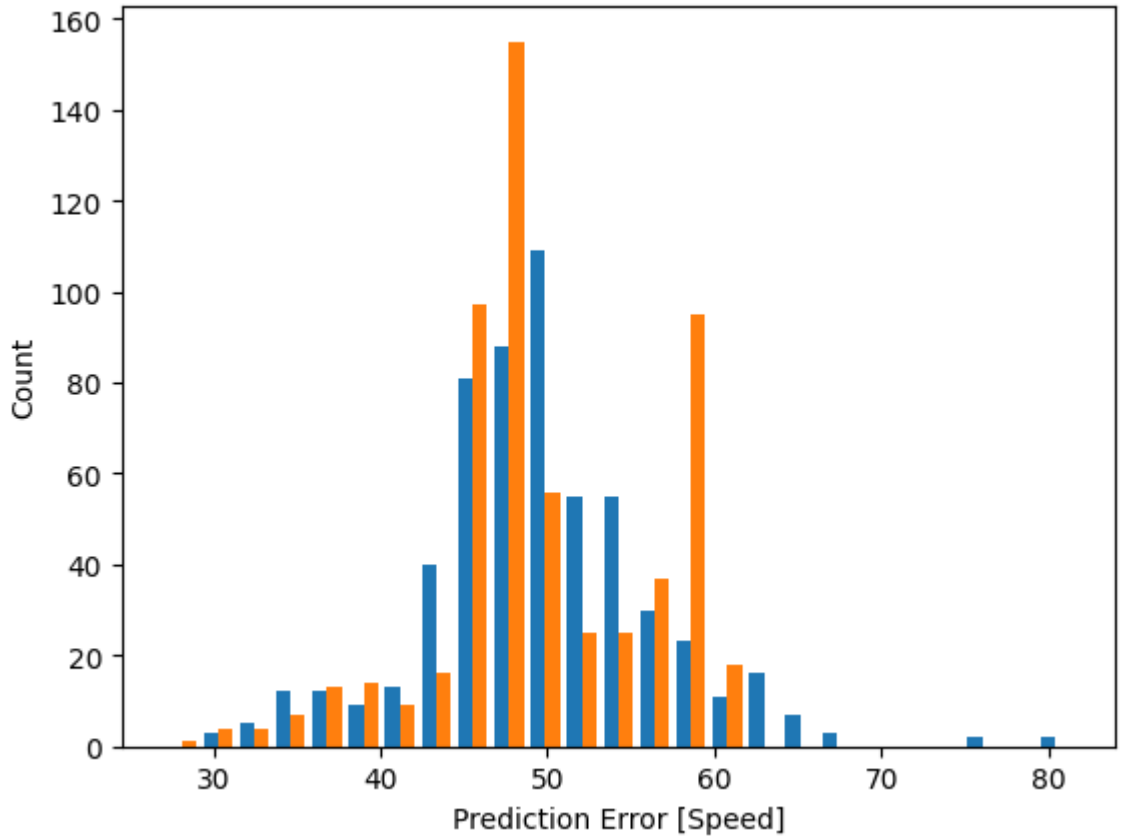
간단한 DNN 회귀 예측 오차

```
plt.scatter(test_real, test_prediction)
plt.xlabel('Real Traffic Volume [ToVol]')
plt.ylabel('Predictions [ToVol]')
plt.plot([0, max_speed], [0, max_speed],'b--')
```



회귀 예측 오차

```
error = test_real[:576], test_prediction[:576]
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [Speed]")
plt.ylabel("Count")
```



숙제

❖ DNN 모델 최적화

✓ Dense 개수는 변경하지 말고,

✓3개의 Dense에서 최적화 뉴런을 찾음

- 마지막 Dense(1) 수정하지 말것.

✓우수한 성능은 MSE 최소값임.

항목	n1	n2	n3	MSE
Dnn_reg1	32	32	32	18.28
Dnn_reg2				?
Dnn_reg3				?
Dnn_reg4				?
Dnn_reg5	?	?	?	?

KISTI SINCE 1962

2023 TRUST KISTI

Korea Institute of Science and Technology Information