

---

# **Experiments : Upper bound of generalization loss**

(kNN, Decision tree, Logistic regression)

AI Expert @ Samsung DS

20. 07. 15 / 22

Instructed by **Mingyu Kim**

# 학습 자료

[https://github.com/MingyuKim87/VC\\_Dim](https://github.com/MingyuKim87/VC_Dim)

1. 커멘드 창에서 적절한 위치에 directory를 생성
2. Git clone [https://github.com/MingyuKim87/VC\\_Dim](https://github.com/MingyuKim87/VC_Dim)

1. 학습목표

2. 배경지식

3. 사용할 알고리즘 별 VC Dimensions

4. 실습

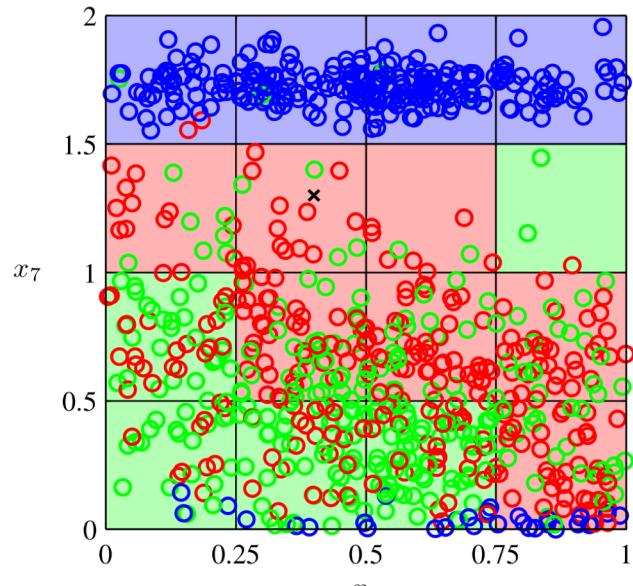
5. 문제

# 학습목표

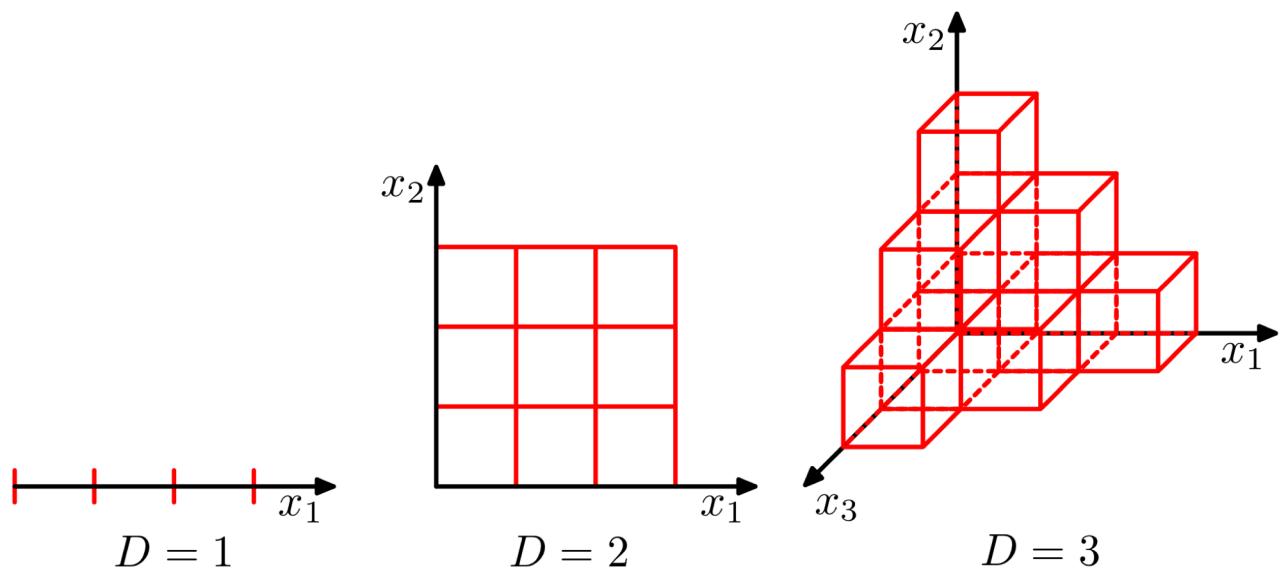
Curse of Dimensionality

## ■ Curse of Dimensionality (차원의 저주)

- 매우 단순한 classification 알고리즘을 도입한다고 가정한다.
  - 알고리즘 : 각 구간별로 나눈 후 training set에서 각 구간 내 속한 데이터 중 가장 많은 클래스로 예측
- 위 알고리즘의 경우 dimension이 증가할 수록 구간이 ‘**기하급수적**’으로 증가
- 해당 알고리즘이 충분한 설명력을 갖기 위해선 큰 차원에서 많은 데이터가 필요  
(구간 내 Training Data 수를 고려한다면)
- 이를 “수학적” 방식으로 접근하여 Generalization loss의 upper bound 구함 → Learning theory



$$D = 2$$



# Background

VC dimensions and upper bound of generalization loss

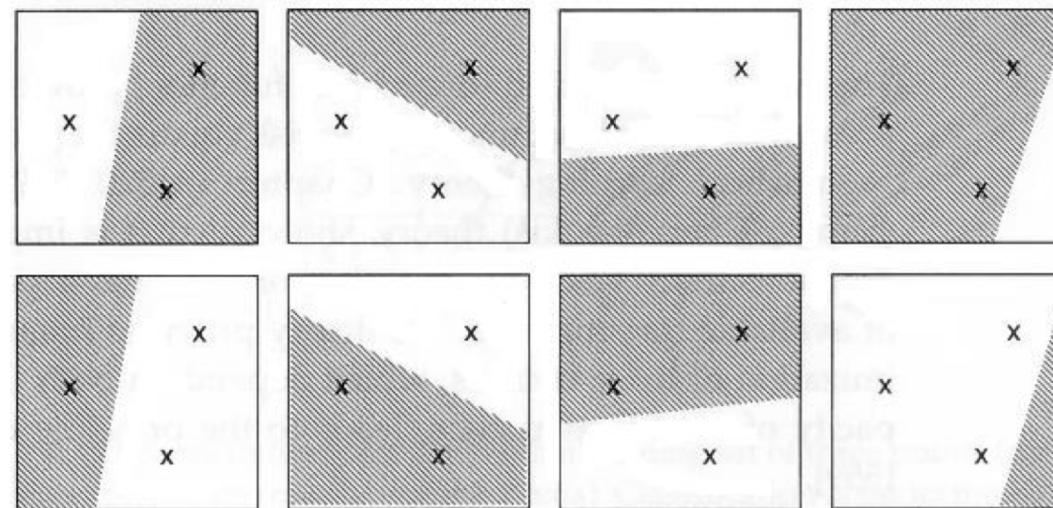
# VC-Dimension

- 주어진 데이터셋  $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ 에 대해서,
  - 이 예제에서는 Binary classification 상황만을 고려하며,
  - 따라서 모든  $y_i \in \{-1, +1\}$  만을 생각합니다.
- Dichotomy: 데이터 포인트들에 대해서 임의로 라벨링을 진행할 때  $(y_1, y_2, \dots, y_m)$ 를 의미합니다.
  - 가능한  $(y_1, y_2, \dots, y_m)$ 중 하나의 예시를 dichotomy라고 이해하면 됩니다.
- Shattering: 만약  $H$ 가  $S$ 의 모든 dichotomy를 표현 할 수 있다고 하면, 우리는  $S$ 가  $H$ 에 의해 shattered 되었다고 표현합니다.
  - 이 때, 함수의 파라미터 셋의 가능한 모든 경우를 가진 집합을 hypothesis set  $H$ 라고 합니다.

## ■ Recap : Learning Theory (PAC learning and VC dimension)

### Example. 2D linear classifier

황성주 교수님, [AI501] Introduction of machine learning 교안 참고



- 해당 예시에서 classifier가 생성할 수 있는 dichotomy의 개수는 총 8개입니다.
- 만약, 세 점이 한 직선 위에 위치한다면, 그 때 classifier가 생성할 수 있는 dichotomy의 개수는 3개입니다.
- 세 점으로 만들 수 있는 dichotomy의 최대 개수는 8개이기 때문에,
  - Set  $S = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\} \rightarrow |S| = 3$
  - Dichotomies  $\{(-, -, -), (+, -, -), \dots, (+, +, +)\} \rightarrow 8$
- 우리는 세 점이 한 직선 위에 있는 경우를 제외한다면,
- 이차원 평면에 세 점이 있는 경우, linear classifier가 이 데이터를 fully shattering 한다고 얘기할 수 있습니다.

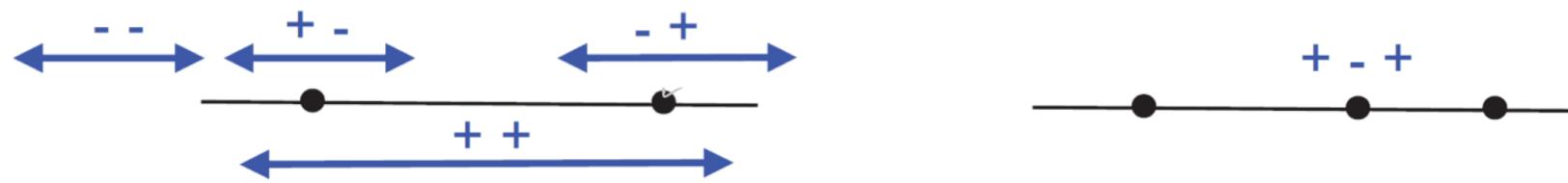
# VC-Dimension

- VC-dimension은 hypothesis set  $H$  (혹은 classifier)가 shattering할 수 있는 최대 point의 개수를 의미합니다.
- 저희가 다루는 예제가 binary classification이기 때문에 수식으로는 다음처럼 표현 할 수 있습니다.
  - $VCdim(H) = \max\{m: \prod_H(m) = 2^m\}$
- $VCdim(H) = d$ 라는 의미는 hypothesis set  $H$ 가 최대  $d$ 개의 points set을 shattering할 수 있다는 의미이며,
- 무조건  $d$ 개의 data points를 shattering 할 수 있다는 의미는 아닙니다.
- 앞 슬라이드의 예시를 들자면,
- 세 점이 한 직선 위에 위치하는 경우, linear classifier는 해당 data points를 shattering 할 수 없습니다.
- 하지만 해당 경우를 제외하면, linear classifier는 임의의 세 점을 shattering 할 수 있습니다.
- 따라서 해당 경우의 VD-dimension의 값은 3 이상이 될 것입니다.

# VC-Dimension

### Example 1. Intervals on the real line

Our first example involves the hypothesis class of intervals on the real line.

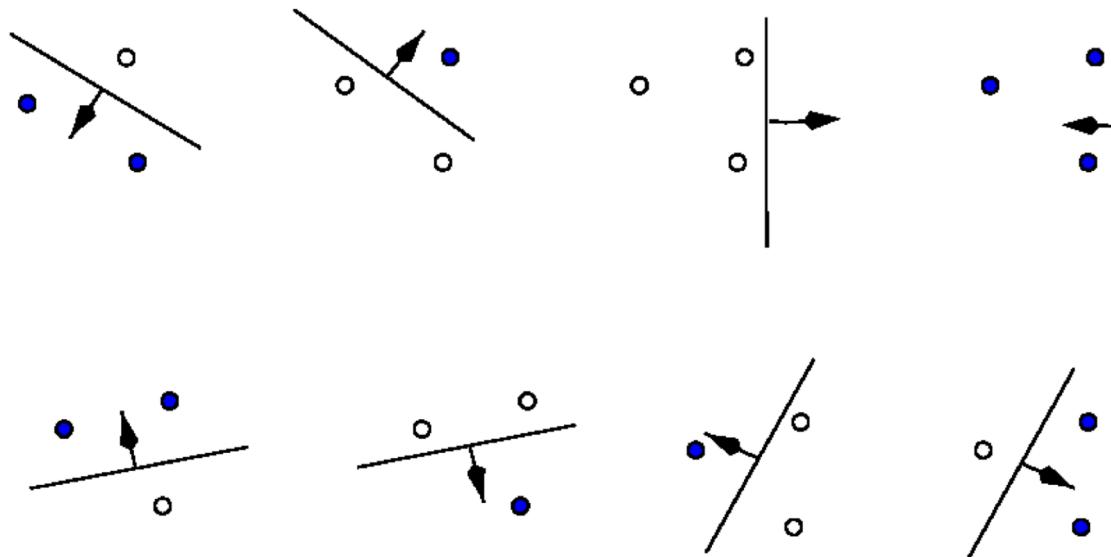


- 간단한 예시를 들어, 1차원 직선에서 2개의 data point가 존재할 때
- 가능한 모든 dichotomy의 종류는  $\{(+,+), (-,-), (+,-), (-,+)\} \rightarrow 4$  가지 입니다.
- Linear classifier는 이 모든 dichotomy를 표현 할 수 있기 때문에,  $VCdim \geq 2$  입니다.
  
- 반면, 1차원 직선에서 3개의 data point가 존재할 때,
- Linear classifier는  $(+,-,+)$ 이나  $(-,+, -)$ 를 표현할 수 없고,  $VCdim < 3$ 입니다.
- 따라서 1차원 직선에서 linear classifier의  $VCdim = 2$ 입니다.

# VC-Dimension

## Example 2. Hyperplane

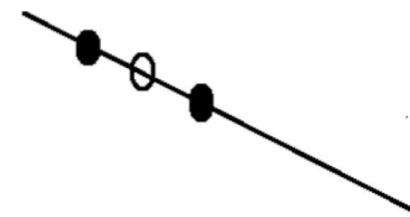
Consider the set of hyperplanes in  $\mathbb{R}^2$ .



- 앞서 언급하였듯이 세 점이 한 직선 위에 존재하는 경우를 제외하면,
- Linear classifier는 세 개의 data point들을 shattering 할 수 있습니다.

# VC-Dimension

Three colinear points in  $\mathbb{R}^2$  cannot be shattered.



- 하지만 세 점이 한 직선 위에 존재하게 된다면,
  - Linear classifier는 해당 예제를 shattering 할 수가 없습니다.
- 
- 일반적으로  $VCdim(H) = d$  이더라도 d개의 data points로 표현 할 수 있는 모든 경우를
  - Shattering 하기는 어렵습니다.

# VC-Dimension Generalization Bounds

$$\mathcal{R}(h) \leq \hat{\mathcal{R}}_S(h) + \sqrt{\frac{2 \log \Pi_{\mathcal{H}}(m)}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}$$

- Generalization bounds
  - $R(h)$  :  $h \in H$ 의 전체 데이터에 대한 general risk (실제 데이터 분포를 알기 어렵기 때문에 계산 불가능)
  - $\hat{R}_S(h)$  : 현재 가진 data points sample  $S$ 에서의  $h$ 의 empirical risk ( $\cong$  Training Loss)
  - $R(h) = \mathbb{E}_{S \sim D}[\hat{R}_S(h)]$  where  $D$  is distribution of dataset
- $\delta$ : failure rate
  - “다음 수식이 최소  $1 - \delta$  확률로 만족하게 된다”라는 의미.
  - 좀 더 강건한 upper bound를 구하고 싶다면  $\delta$ 의 값을 줄이면 됩니다.
  - 즉,  $\delta$ 는 위 부등식이 만족하지 않은 확률이라고 생각하시면 됩니다.

# VC-Dimension Generalization Bounds

- Sauer's lemma에 의해  $\Pi_H(m) \leq \left(\frac{em}{d}\right)^d$  가 성립하며, 이에 따라
- VC-dimension을 이용한 generalization bound를 계산할 수 있습니다.

**Proof.**

$$\Pi_{\mathcal{H}}(m) \leq \sum_{i=0}^d \binom{m}{i} \quad \text{Sauer's lemma}$$

$$\leq \sum_{i=0}^d \binom{m}{i} \left(\frac{m}{d}\right)^{d-i} \leq \sum_{i=0}^m \binom{m}{i} \left(\frac{m}{d}\right)^{d-i} \quad m \geq d$$

$$\begin{aligned}
 &= \left(\frac{m}{d}\right)^d \sum_{i=0}^m \binom{m}{i} \left(\frac{d}{m}\right)^i = \left(\frac{m}{d}\right)^d \left(1 + \frac{d}{m}\right)^m \quad \text{By the binomial formula:} \\
 &\quad \sum_{i=0}^m \binom{m}{i} x^i = (1+x)^m \\
 &\leq \left(\frac{m}{d}\right)^d e^{\frac{d}{m} \cdot m} = \left(\frac{m}{d}\right)^d e^d
 \end{aligned}$$

## ■ Recap : Learning Theory (PAC learning and VC dimension)

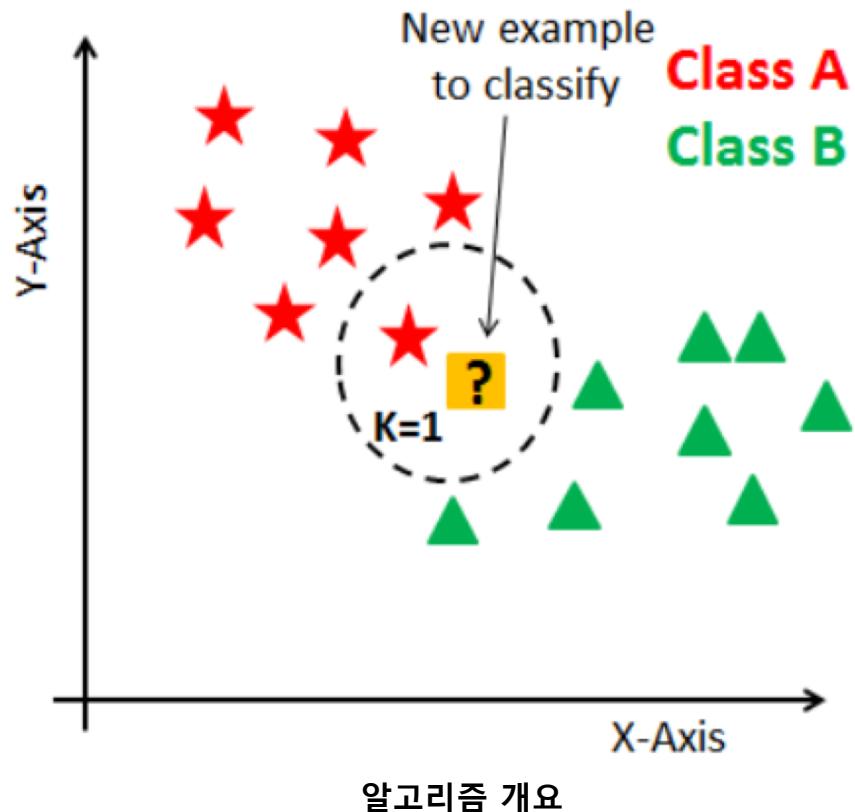
---

- Practices
  - VC Dim 의 경우 일반적으로 구하기가 매우 어려움
  - 몇개의 모델의 경우 수학적으로 엄밀하게 구할 수 있음. (kNN, Decision Tree, linear plane)
  - 일반적으로 학습 데이터만 주어져 있다고 가정하여  
학습 데이터를 통해 평가한 loss만 구할 수 있다.
  - 모조 데이터를 통한 실험에서는 True generalization loss를 구할 수 없기 때문에  
Test Set을 생성하여 구한 loss와 upper bound를 비교해 본다.

$$\mathcal{R}(h) \leq \hat{\mathcal{R}}_S(h) + \sqrt{\frac{2d \log \frac{em}{d}}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}$$

# VC dimensions

kNN, decision tree and logistic regression



```

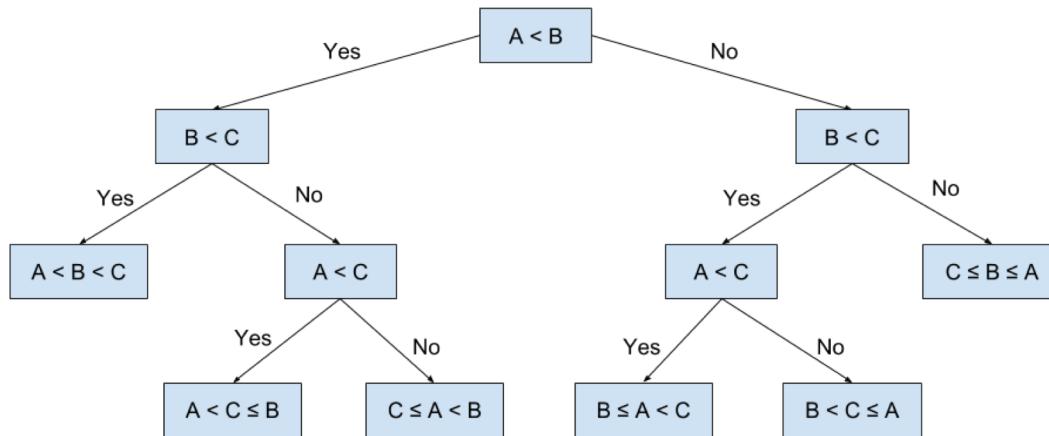
k-Nearest Neighbor
Classify (X, Y, x) // X: training data, Y: class labels of X, x: unknown sample
for  $i = 1$  to  $m$  do
    Compute distance  $d(X_i, x)$ 
end for
Compute set  $I$  containing indices for the  $k$  smallest distances  $d(X_i, x)$ .
return majority label for  $\{Y_i \text{ where } i \in I\}$ 

```

#### Pseudo Codes of kNN

- VC Dimension
  - K = 1일 때, training set의 data 개수가 VC Dim이 된다.
  - 총 K개의 영역으로 나눌 수 있기 때문

## ■ Decision Tree



알고리즘 개요

---

### Algorithm 1.1 C4.5(D)

---

**Input:** an attribute-valued dataset  $D$

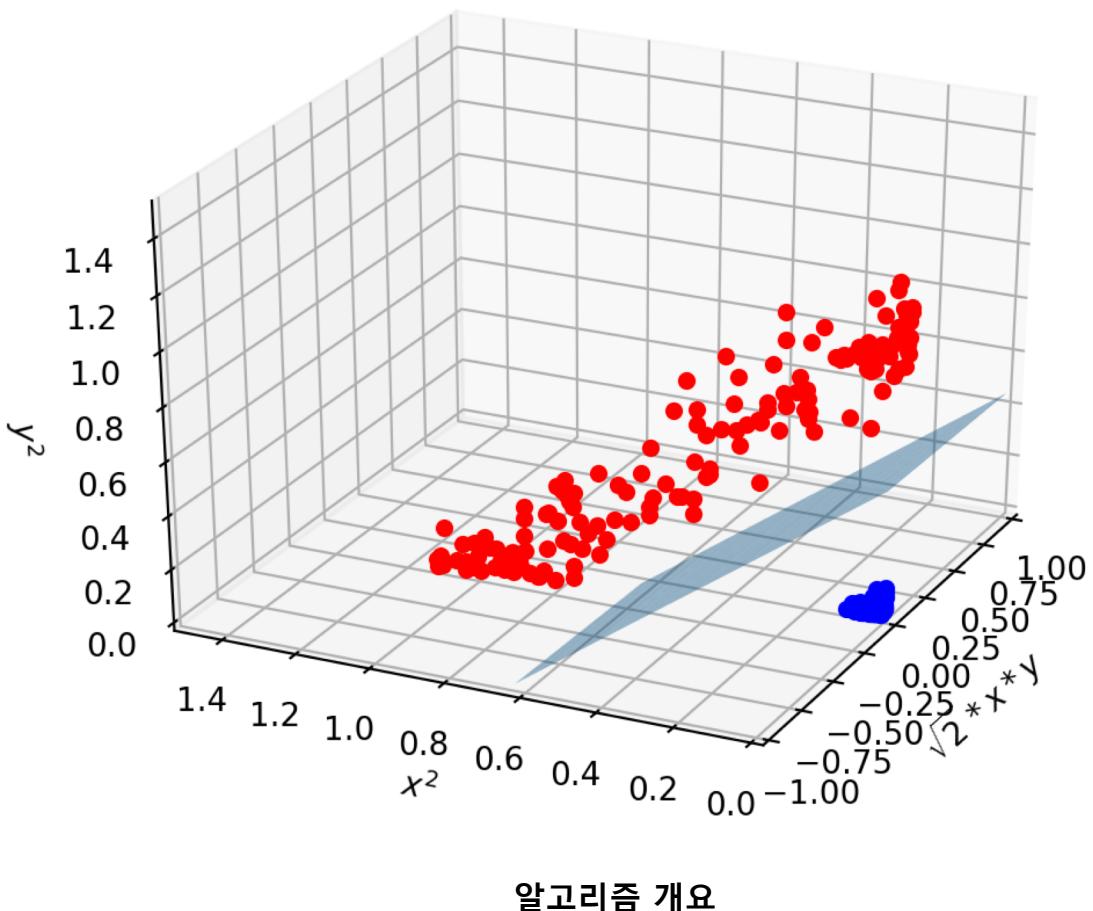
```
1: Tree = {}
2: if  $D$  is "pure" OR other stopping criteria met then
3:   terminate
4: end if
5: for all attribute  $a \in D$  do
6:   Compute information-theoretic criteria if we split on  $a$ 
7: end for
8:  $a_{best}$  = Best attribute according to above computed criteria
9: Tree = Create a decision node that tests  $a_{best}$  in the root
10:  $D_v$  = Induced sub-datasets from  $D$  based on  $a_{best}$ 
11: for all  $D_v$  do
12:   Treev = C4.5( $D_v$ )
13:   Attach Treev to the corresponding branch of Tree
14: end for
15: return Tree
```

---

Pseudo Codes of decision tree

- VC Dimension
  - Terminal node의 개수가 VC dim 이 된다.

## ■ Linear separable plane (logistic regression)



- VC Dimension
  - 앞서 언급한 것처럼,  $VCdim = d + 1$  이 된다.
  - 해당 예제는  $VCdim = 2 + 1 = 3$  이다.

---

### Algorithm Logistic Regression based MA on APUF

---

```
1: procedure LR( $N$  CRP  $(\mathbf{c}_i, r_i)$  or  $(\Phi_i, r_i)$  ( $r_i \in \{-1, +1\}$ ), increment  $\eta$ , threshold  $\epsilon$ )
2:   Randomly generate  $\mathbf{w} = (\mathbf{w}[0], \dots, \mathbf{w}[n - 1], 1)$ 
3:   while (1) do
4:      $\nabla l = (0, \dots, 0)$ 
5:     for  $i \leftarrow 0, N - 1$  do
6:        $\nabla l = \nabla l + (\hat{y}_i - r_i)\Phi_i$ 
7:     end for
8:      $\mathbf{w} = \mathbf{w} - \eta \times \nabla l$ 
9:     if  $\|\nabla l\| \leq \epsilon$  then
10:      Break While Loop
11:    end if
12:  end while
13:  output the model  $\mathbf{w}$ 
14: end procedure
```

---

$$\hat{y} = \sigma(f(\mathbf{w}, \mathbf{c})) = \frac{1}{1 + e^{-f(\mathbf{w}, \mathbf{c})}}$$

Pseudo Codes of logistic regression

# 실험설계

VC dim for kNN, decision tree and logistic regression

## ■ 실험의 중점 사항

- 본 실험의 관심사
  - 지금까지 실습의 경우 데이터가 주어져 있고, 이를 예측할 알고리즘을 생성하는데 관심
  - 본 실험에서는 예측 알고리즘은 주어져 있는 상태에서 학습 데이터 포인트의 개수와 차원을 조절하여 실제 generalization loss의 upper bound를 구해보고자 한다.
  - True generalization loss를 구하기가 어려우니 이를 검증하기 위한 방법으로 Test data set을 생성하고 loss를 평가하여 bound의 유효성을 체험해보고자 한다.

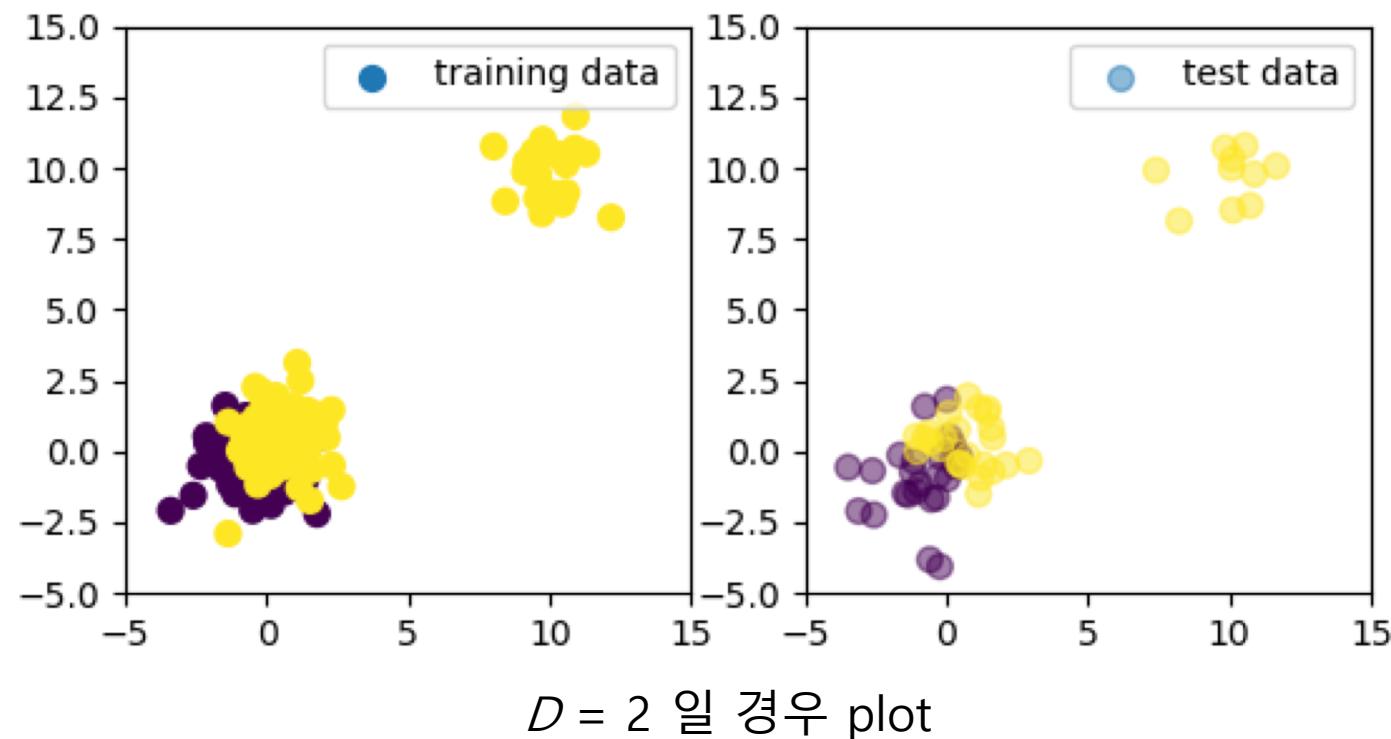
$$\mathcal{R}(h) \leq \hat{\mathcal{R}}_S(h) + \sqrt{\frac{2d \log \frac{em}{d}}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}$$

- $d \uparrow m \downarrow$  일 때의 upper bound
- $d \downarrow m \uparrow$  일 때의 upper bound

Where,  $\hat{\mathcal{R}}_S$  : training loss

## ■ 실험 설계

- 본 실험 데이터
  - 모조 데이터로 실습
    - 모조 데이터의 dimensions과 samples의 개수를 조절할 수 있음
    - 모조 데이터의 경우 임의의 normal distribution 두개에서 sampling 하여 선별
      - 노란색 Target 1 / 보라색 Target 0



# 구현 1 : 제공한 파일 내 작성

Implementation : 각 알고리즘 별 학습 방법

- ◆ knn.py, decision\_tree.py, logistic\_regression.py 내  
if \_\_name\_\_ == '\_\_main\_\_': 아래 작성  
indentation 중요

- kNN Training (1)

```
def knn_train(dimensions, train_count, test_count, iterations=10):  
    # For evaluation  
    BCE_list_train = []  
    BCE_list_test = []  
  
    for i in range(iterations):  
        # File import  
        train_x_data, train_y_data = create_toy_data(dimensions, train_count, test_count, add_outliers=True, training=True)  
        test_x_data, test_y_data = create_toy_data(dimensions, train_count, test_count, add_outliers=True, training=False)  
  
        # Feature engineering  
        X_train = train_x_data  
        X_test = test_x_data  
  
        # Model  
        knn = kNN()  
  
        # Training (Learning)  
        knn.fit(X_train, train_y_data)  
  
        # Predicting a training set  
        y_hat_train = knn.predict(X_train)  
  
        # Predicting a test set  
        y_hat_test = knn.predict(X_test)
```

데이터  
생성

모델  
생성

학습

학습  
데이터  
예측

Test  
데이터  
예측

## • kNN Training (2)

```
# Evaluating
BCE_value_train = binary_cross_entropy(train_y_data, y_hat_train)
BCE_value_test = binary_cross_entropy(test_y_data, y_hat_test)

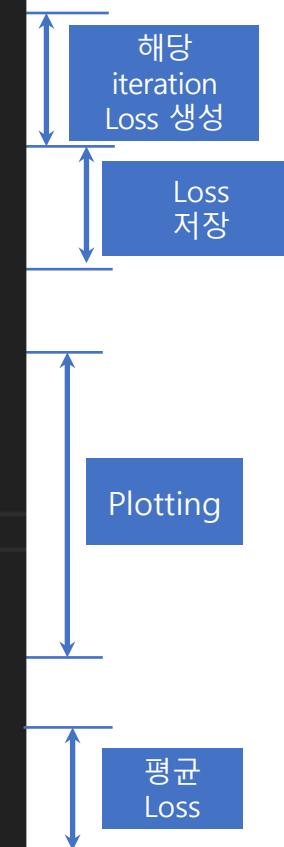
# Appending
BCE_list_train.append(BCE_value_train)
BCE_list_test.append(BCE_value_test)

# Plotting
if dimensions == 2 and i == 0:
    x1_test, x2_test = np.meshgrid(np.linspace(-5, 15, 100), np.linspace(-5, 15, 100))
    X_test_plot = np.array([x1_test, x2_test]).reshape(2, -1).T

    # logistic regression
    y_hat_plot = knn.predict(X_test_plot)
    plot(train_x_data, train_y_data, test_x_data, test_y_data, x1_test, x2_test, y_hat_plot, "./Results/knn_result")

# average MSE
(average_BCE_train, BCE_std_train) = average_metric(BCE_list_train)
(average_BCE_test, BCE_std_test) = average_metric(BCE_list_test)

return (average_BCE_train, BCE_std_train), (average_BCE_test, BCE_std_test)
```

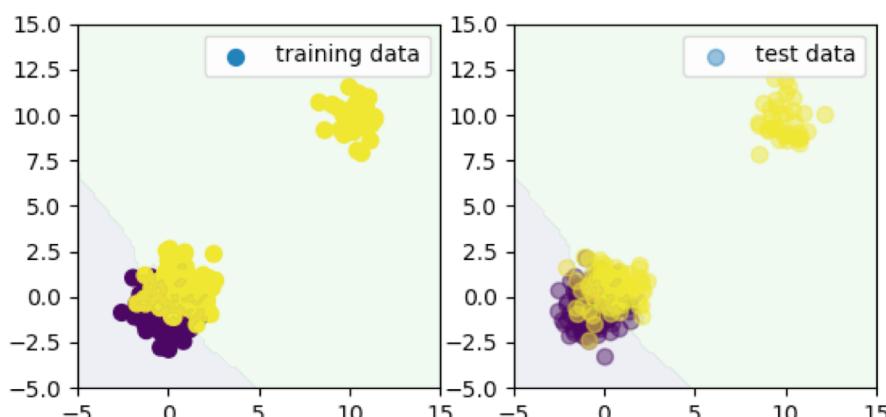


## ■ 구현 (실습)

- kNN Training 실습 예제
  - 데이터 생성 : dimensions 2의 모조 데이터 생성
  - Training Set의 데이터 개수 : 60 개 / Test Set의 데이터 개수 : 25개

```
if __name__ == "__main__":
    # For evaluation
    dimensions = 2
    train_count = 60
    test_count = 25

    # training
    model_train_result, model_test_result = knn_train(dimensions, train_count, test_count, 10)
```



```
* * * * * dimensions : 2 train_count : 60 * * * * *
* * * * * train * * * * *
[Average BCE]
-0.0000
[BCE_std]
0.0000
* * * * * test * * * * *
[Average BCE]
3.7340
[BCE_std]
0.7641
```

- Decision Tree Training (1)

```
def decision_tree_train(dimensions, train_count, test_count, iterations=10):
    # Set Hyperparameters
    max_depth = 5
    min_size = 10

    # For evaluation
    BCE_list_train = []
    BCE_list_test = []
    Terminal_count_list = []

    for i in range(iterations):
        # File import
        train_x_data, train_y_data = create_toy_data(dimensions, train_count, test_count, add_outliers=True, training=True)
        test_x_data, test_y_data = create_toy_data(dimensions, train_count, test_count, add_outliers=True, training=False)

        # Feature engineering
        X_train = train_x_data
        X_test = test_x_data

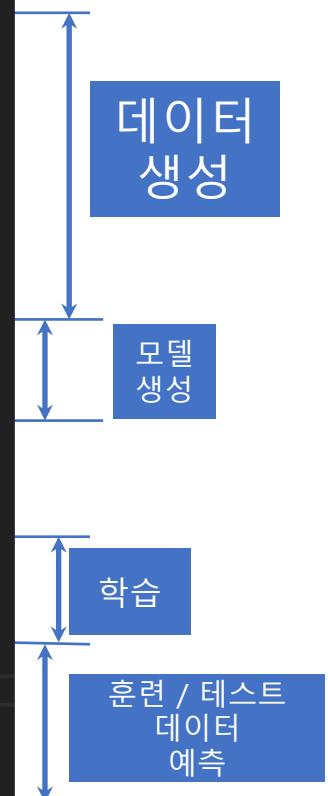
        # Model
        decision_tree = Decision_tree()

        # combine the features and targets
        train_y_data_tree = train_y_data[:, None]
        Tree_input = np.hstack((X_train, train_y_data_tree))

        # Training (Learning)
        tree, terminal_count = decision_tree.build_tree(Tree_input, max_depth, min_size, np.shape(Tree_input)[1]-1)

        # Predicting a training set
        y_hat_train = decision_tree.predicts(tree, X_train)

        # Predicting a test set
        y_hat_test = decision_tree.predicts(tree, X_test)
```



## ■ 구현 (연습 1)

- Decision Tree Training (2)

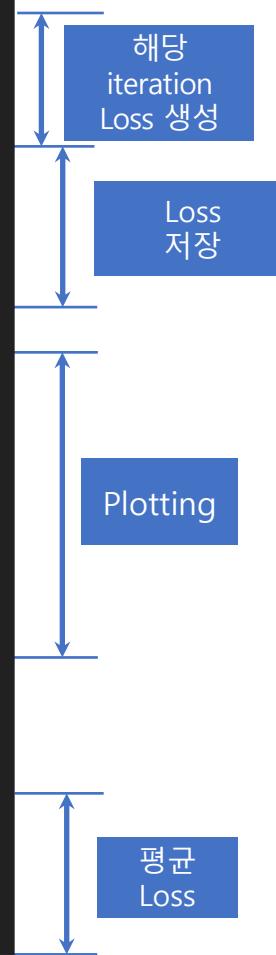
```
# Evaluating
BCE_value_train = binary_cross_entropy(train_y_data, y_hat_train)
BCE_value_test = binary_cross_entropy(test_y_data, y_hat_test)

# Appending
BCE_list_train.append(BCE_value_train)
BCE_list_test.append(BCE_value_test)
Terminal_count_list.append(terminal_count)

# Plotting
if dimensions == 2 and i == 0:
    x1_test, x2_test = np.meshgrid(np.linspace(-5, 15, 100), np.linspace(-5, 15, 100))
    X_test_plot = np.array([x1_test, x2_test]).reshape(2, -1).T

    # logistic regression
    y_hat_plot = decision_tree.predicts(tree, X_test_plot)
    plot(train_x_data, train_y_data, test_x_data, test_y_data, x1_test, x2_test, y_hat_plot, "./Results/tree_result")

# average MSE
(average_BCE_train, BCE_std_train) = average_metric(BCE_list_train)
(average_BCE_test, BCE_std_test) = average_metric(BCE_list_test)
average_terminal_count = np.average(Terminal_count_list)
```

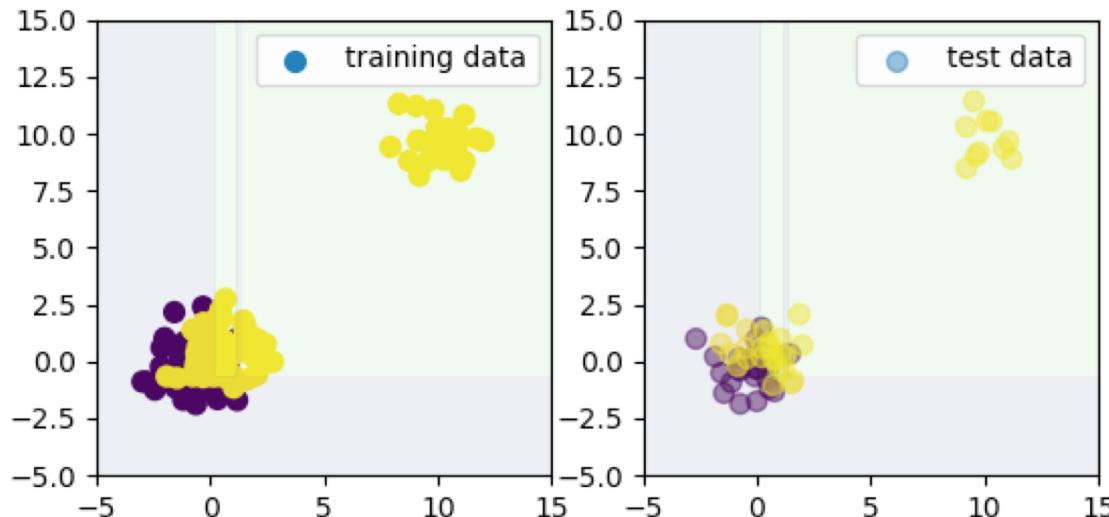


## ■ 구현 (실습)

- Decision Tree Training 실습 예제
  - 데이터 생성 : dimensions 2의 모조 데이터 생성
  - Training Set의 데이터 개수 : 60 개 / Test Set의 데이터 개수 : 25개

```
if __name__ == "__main__":
    # For evaluation
    dimensions = 2
    train_count = 60
    test_count = 25

    model_train_result, model_test_result, average_terminal_count = decision_tree_train(dimensions, train_count, test_count, 10)
```



```
* * * * * dimensions : 2 train_count : 60 * * * * *
* * * * * train * * * * *
[Average BCE]
1.8133
[BCE_std]
0.5790
* * * * * test * * * * *
[Average BCE]
4.4056
[BCE_std]
1.0066
* * * * * test * * * * *
[Average terminal count]
15.8
```

## ■ 구현 (연습 1)

- Logistic Regression Training (1)

```
def logistic_regression_train(dimensions, train_count, test_count, iterations=10):  
  
    # For evaluation  
    BCE_list_train = []  
    BCE_list_test = []  
  
    for i in range(iterations):  
        # File import  
        train_x_data, train_y_data = create_toy_data(dimensions, train_count, test_count, add_outliers=True, training=True)  
        test_x_data, test_y_data = create_toy_data(dimensions, train_count, test_count, add_outliers=True, training=False)  
  
        # Feature engineering  
        X_train = train_x_data  
        X_test = test_x_data  
  
        # Model  
        logistic_regression = LogisticRegression()  
  
        # Training (Learning)  
        logistic_regression.fit(X_train, train_y_data)  
  
        # Predicting a training set  
        y_hat_train = logistic_regression.classify(X_train)  
  
        # Predicting a test set  
        y_hat_test = logistic_regression.classify(X_test)  
  
        # Evaluation  
        BCE_list_train.append(calculate_BCE(y_hat_train, train_y_data))  
        BCE_list_test.append(calculate_BCE(y_hat_test, test_y_data))
```



## ■ 구현 (연습 1)

- kNN Training (2)

```
# Evaluating
BCE_value_train = binary_cross_entropy(train_y_data, y_hat_train)
BCE_value_test = binary_cross_entropy(test_y_data, y_hat_test)

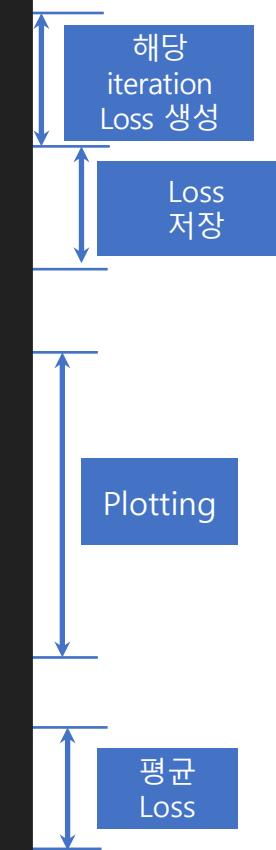
# Appending
BCE_list_train.append(BCE_value_train)
BCE_list_test.append(BCE_value_test)

# Plotting
if dimensions == 2 and i == 0:
    x1_test, x2_test = np.meshgrid(np.linspace(-5, 15, 100), np.linspace(-5, 15, 100))
    X_test_plot = np.array([x1_test, x2_test]).reshape(2, -1).T

    # logistic regression
    y_hat_plot = logistic_regression.classify(X_test_plot)
    plot(train_x_data, train_y_data, test_x_data, test_y_data, x1_test, x2_test, y_hat_plot, "./Results/logits_result")

# average MSE
(average_BCE_train, BCE_std_train) = average_metric(BCE_list_train)
(average_BCE_test, BCE_std_test) = average_metric(BCE_list_test)

return (average_BCE_train, BCE_std_train), (average_BCE_test, BCE_std_test)
```

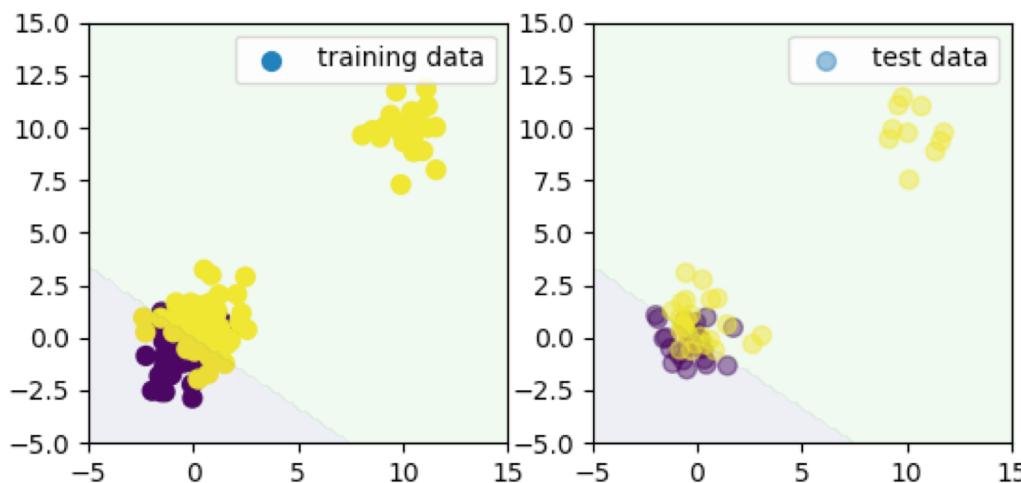


## ■ 구현 (실습)

- Logistic Regression Training 실습 예제
  - 데이터 생성 : dimensions 2의 모조 데이터 생성
  - Training Set의 데이터 개수 : 60 개 / Test Set의 데이터 개수 : 25개

```
if __name__ == "__main__":
    # For evaluation
    dimensions = 2
    train_count = 60
    test_count = 25

    # Training
    model_train_result, model_test_result = logistic_regression_train(dimensions, train_count, test_count, 10)
```



```
* * * * * dimensions : 2 train_count : 60 * * * * *
* * * * * train * * * * *
[Average BCE]
3.4587
[BCE_std]
0.5739
* * * * * test * * * * *
[Average BCE]
3.5728
[BCE_std]
0.8667
```

# 구현 2 : main.py 생성

VC-Dimension Generalization Bounds

Dimension / Sample 수에 따른 Train, Test set에 따른 loss와 Bound 비교

## ■ 구현 (실습)

- VC Dimension 함수 작성
  - Arguments : train\_loss, vc\_dim, sample\_count, failure\_rate

$$\mathcal{R}(h) \leq \hat{\mathcal{R}}_S(h) + \sqrt{\frac{2d \log \frac{em}{d}}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}$$

-----  
Generalization  
loss  $\approx$  Test Set Loss

Upper Bound

```
def vc_dimension_upper_bound(train_loss, vc_dim, sample_count, failure_rate):  
    upper_bound = train_loss \  
    + np.sqrt((2*vc_dim*np.log(np.exp(1) * sample_count) / vc_dim)) / sample_count) \  
    + np.sqrt(np.log(1/failure_rate) / (2 *sample_count))  
  
    return upper_bound
```

## ■ 구현 (실습)

- Main 함수 작성 (1)
  - Containers 만들기

```
if __name__ == "__main__":
    # failtuer_rate
    failure_rate = .05

    # Index
    dimension_list = []
    train_count_list = []

    # Train loss list
    logistic_regression_loss_train = []
    knn_loss_train = []
    decision_tree_loss_train = []

    # loss Upper bound list
    logistic_regression_upper = []
    knn_loss_upper = []
    decision_tree_loss_upper = []

    # Test loss list
    logistic_regression_loss_test = []
    knn_loss_test = []
    decision_tree_loss_test = []
```

Failure\_rate  
설정

각 모델 별  
train loss,  
upper bound,  
test loss  
담을 containers 생성

## ■ 구현 (실습)

- Main 함수 작성 (2)
  - Dimensions / Sample Count를 변경해가며 실험 수행

```
for i in range(1,10,1):
    for j in range(10, 101, 20):
        # Experimental settings
        dimensions = i
        train_count = j
        test_count = j

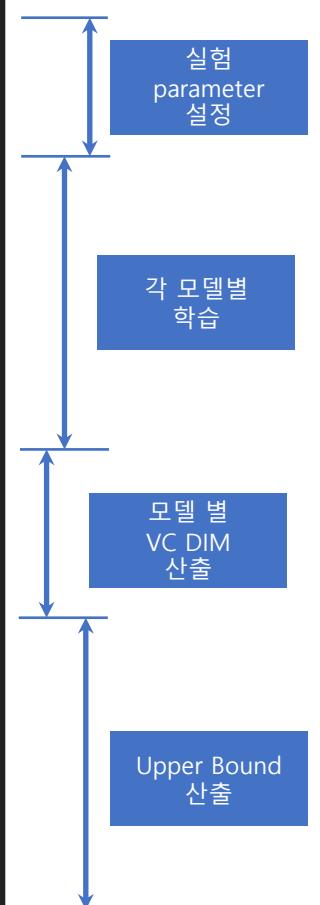
        # Model Train

        train_result_logistic, test_result_logistic \
            = logistic_regression_train(dimensions, train_count, test_count, 10)
        train_result_knn, test_result_knn \
            = knn_train(dimensions, train_count, test_count, 10)
        train_result_tree, test_result_tree, average_terminal_count \
            = decision_tree_train(dimensions, train_count, test_count, 10)

        # VC dimension
        vc_dimension_linear = i + 1
        vc_dimension_knn = j
        vc_dimension_decision_tree = average_terminal_count

        # Upper bound

        upper_bound_logistic \
            = vc_dimension_upper_bound(train_result_logistic[0], vc_dimension_linear, train_count, failure_rate)
        upper_bound_knn \
            = vc_dimension_upper_bound(train_result_knn[0], vc_dimension_knn, train_count, failure_rate)
        upper_bound_tree \
            = vc_dimension_upper_bound(train_result_tree[0], vc_dimension_decision_tree, train_count, failure_rate)
```



## ■ 구현 (실습)

- Main 함수 작성 (2)
  - 결과값 저장

```
# Append
dimension_list.append(dimensions)
train_count_list.append(train_count)

# logistic
logistic_regression_loss_train.append(train_result_logistic[0])
logistic_regression_upper.append(upper_bound_logistic)
logistic_regression_loss_test.append(test_result_logistic[0])

# knn
knn_loss_train.append(train_result_knn[0])
knn_loss_upper.append(upper_bound_knn)
knn_loss_test.append(test_result_knn[0])

# tree
decision_tree_loss_train.append(train_result_tree[0])
decision_tree_loss_upper.append(upper_bound_tree)
decision_tree_loss_test.append(test_result_tree[0])
```

- Main 함수 작성 (2)
  - Pandas DataFrame을 활용하여 table 형태로 살펴보기

```
# Make a dataframe
result_data = [dimension_list, train_count_list, \
    logistic_regression_loss_train, logistic_regression_upper, logistic_regression_loss_test, \
    knn_loss_train, knn_loss_upper, knn_loss_test, \
    decision_tree_loss_train, decision_tree_loss_upper, decision_tree_loss_test]

result_data = np.array(result_data)
result_data = result_data.T

result_column_name = ["dimensions", "train_count", \
    "logistic_train_loss", "logistic_upper_bound", "logistic_test_loss", \
    "knn_train_loss", "knn_upper_bound", "knn_test_loss", \
    "tree_train_loss", "tree_upper_bound", "tree_test_loss"]

result_DF = pd.DataFrame(result_data, columns=result_column_name)
```

# 결과

VC dim for kNN, decision tree and logistic regression

## ■ 구현 (실습)

- 결과 확인

Index	dimensions	train_count	logistic_train_loss	logistic_upper_bound	logistic_test_loss	knn_train_loss	knn_upper_bound	knn_test_loss	tree_train_loss	tree_upper_bound	tree_test_loss
0	1	10	3.962	5.371	3.627	0.000	1.801	5.306	2.821	1.618	4.768
1	1	30	4.074	5.001	4.074	0.000	1.638	5.731	2.642	1.400	4.858
2	1	50	4.150	4.904	4.513	0.000	1.587	5.373	3.197	1.258	5.091
3	1	70	4.125	4.782	3.962	0.000	1.560	5.497	2.821	1.202	4.519
4	1	90	4.074	4.666	4.164	0.000	1.543	5.276	2.992	1.142	4.888
5	2	10	3.089	4.626	3.492	0.000	1.801	4.835	1.477	1.632	4.164
6	2	30	2.776	3.812	3.224	0.000	1.638	4.768	1.500	1.421	4.298
7	2	50	3.371	4.221	3.250	0.000	1.587	4.150	1.558	1.323	4.285
8	2	70	3.329	4.072	3.416	0.000	1.560	4.404	2.044	1.226	4.576
9	2	90	3.612	4.282	3.865	0.000	1.543	4.112	2.045	1.169	3.947
10	3	10	1.746	3.371	3.895	0.000	1.801	4.701	1.276	1.610	3.694
11	3	30	2.328	3.448	2.507	0.000	1.638	3.336	1.500	1.390	3.649
12	3	50	2.659	3.584	2.404	0.000	1.587	3.559	1.477	1.304	3.707
13	3	70	2.706	3.516	2.696	0.000	1.560	3.607	1.468	1.228	3.531
14	3	90	3.059	3.793	2.642	0.000	1.543	3.641	1.627	1.123	3.462
15	4	10	1.209	2.897	2.552	0.000	1.801	3.425	1.209	1.646	4.634
16	4	30	2.485	3.673	2.127	0.000	1.638	3.246	1.231	1.409	3.268
17	4	50	1.921	2.907	2.257	0.000	1.587	3.439	1.128	1.265	3.304
18	4	70	2.178	3.045	2.226	0.000	1.560	3.243	1.123	1.202	3.185
19	4	90	2.485	3.271	2.313	0.000	1.543	3.395	1.224	1.145	3.156
20	5	10	1.075	2.808	2.351	0.000	1.801	2.619	0.672	1.625	4.500
21	5	30	1.589	2.835	1.858	0.000	1.638	3.022	1.231	1.373	3.962
22	5	50	1.410	2.449	1.813	0.000	1.587	2.592	1.142	1.280	3.747
23	5	70	1.641	2.557	2.015	0.000	1.560	3.195	1.007	1.181	3.454
24	5	90	1.672	2.504	2.216	0.000	1.543	2.940	1.172	1.112	3.149
25	6	10	0.537	2.302	2.418	0.000	1.801	3.224	0.470	1.665	3.694
26	6	30	1.343	2.637	1.769	0.000	1.638	2.888	0.895	1.409	3.224
27	6	50	1.424	2.508	1.854	0.000	1.587	2.565	0.833	1.292	3.197
28	6	70	1.334	2.293	1.727	0.000	1.560	2.226	0.892	1.212	3.252
29	6	90	1.485	2.357	1.716	0.000	1.543	2.433	1.007	1.135	3.067
30	7	10	0.134	1.920	2.149	0.000	1.801	2.686	0.537	1.659	4.164
31	7	30	0.918	2.254	1.477	0.000	1.638	1.948	0.784	1.406	3.671
32	7	50	1.007	2.133	1.276	0.000	1.587	2.001	0.860	1.265	3.318
33	7	70	1.190	2.187	1.650	0.000	1.560	2.351	0.739	1.177	3.022
34	7	90	1.037	1.946	1.515	0.000	1.543	2.179	0.709	1.127	3.209
35	8	10	0.201	1.999	1.880	0.000	1.801	2.216	0.269	1.625	4.231
36	8	30	0.582	1.955	1.388	0.000	1.638	1.970	0.761	1.415	3.783
37	8	50	0.712	1.874	1.545	0.000	1.587	2.109	0.833	1.247	3.318
38	8	70	0.835	1.867	1.353	0.000	1.560	2.149	0.758	1.163	3.214
39	8	90	0.992	1.934	1.463	0.000	1.543	1.963	0.739	1.108	2.985
40	9	10	0.067	1.868	1.075	0.000	1.801	1.746	0.403	1.602	4.164
41	9	30	0.269	1.675	1.276	0.000	1.638	2.216	0.560	1.433	3.156
42	9	50	0.497	1.692	1.249	0.000	1.587	1.625	0.712	1.263	3.331
43	9	70	0.796	1.860	1.142	0.000	1.560	1.928	0.662	1.163	3.147
44	9	90	0.806	1.778	1.134	0.000	1.543	1.686	0.649	1.098	3.045

# 문제

VC dim for kNN, decision tree and logistic regression

1. Test loss와 upper bound간의 비교를 진행해보자
    - 각 모델 별 Dimensions과 Sample Count에 따라 서술하라. (무엇에 더 민감할까?)
  2. Logistic Regression의 경우 Upper Bound보다 Test loss가 작았다. 이유가 무엇일까?
  3. kNN같은 경우 Overfit이 발생함을 알 수 있다. 이유가 무엇일까?
  4. Decision Tree는 어떤 이유 때문에 Test Loss가 줄어들지 않는 것일까?
- ❖ Bonus
- 각 모델별 Test Loss와 Upper Bound를 비교할 수 있는 Plot을 작성해보자.

## ■ Reference

---

1. 황성주 (2019), [AI501] Introduction of Machine Learning 교안
2. C. Bishop (2007), Pattern recognition and machine learning
3. M. Mohri, A. Rostamizadeh and A. Talwalkar (2012), Foundations of Machine learning 2<sup>nd</sup>