Homework 2 Report

AIE2004: Probability for Artificial Intelligence

-Iris flower identification using Bayesian Classifier -

12191656 이채연

Part A. Preliminaries

1. Cross Validation

보통 dataset 이 있을 때 이 dataset 을 train dataset 과 test dataset 으로 나눠 train dataset 으로 모델을 훈련한 뒤, test dataset 으로 모델을 evaluation 한다. 하지만 고정된 test dataset 으로만 모델의 성능을 evaluation 하게 되면 test dataset 에만 잘 동작하는 overfitting 이 발생할 수 있다는 문제점이 있다. 특히 dataset 의 크기가 작을 때 이러한 문제가 많이 발생한다. 그래서 이러한 문제를 해결하기 위해 Cross Validation 방법을 쓴다.

Cross Validation 은 모델을 evaluation 하는 방법 중 하나로 train set 을 train set 과 validation set 으로 분리한 뒤, validation set 을 이용하여 검증한다.

K-Fold Cross Validation 은 가장 일반적으로 사용되는 cross validation 방법이다. K-Fold Cross Validation 의 과정은 다음과 같다.

먼저, training dataset 을 K 개의 folds(subsets)로 나눈다. 그리고 내가 evaluation 하려는 모델이 있을 때 k 번째 subset 을 제외한 나머지 모든 fold 에 대해서 이 모델을 training 하고, k 번째 fold 에 대해 이 모델을 test 한다. K=1 에서부터 k=K 까지 K 번 이 과정을 반복하면, 모든 데이터를 K 번만큼 사용해서 train set 과 test set 을 만들 수 있다. 이때 각각의 subset 은 각 iteration 에서 K-1 번만큼 training 에 사용되고, 1 번은 test 에 사용된다. 이렇게 나온 총 K 개의 evaluation 결과의 평균을 해당 모델의 학습 성능이라고 한다.

K-Fold Cross Validation 은 이렇게 모든 train data 가 한 번은 evaluation 에 쓰일 수 있도록 하여 overfitting 문제를 해결하고 좀 더 일반화된 모델을 선택할 수 있도록 한다.

3. Acquiring dataset

```
import pandas as pd #data loader library
import matplotlib.pyplot as plt #graph visualization library
import numpy as np
#iris dataset을 얻고 class별로 분리해놓는다.
df = pd.read_csv('./iris/iris.data', names=['sepal length', 'sepal width', 'petal length', 'petal width', 'class'])
se=df[df['class'] =='Iris-setosa']
ve=df[df['class']=='Iris-versicolor']
vi=df[df['class']=='Iris-virginica']
```

Part B. Iris classification using text data

1.

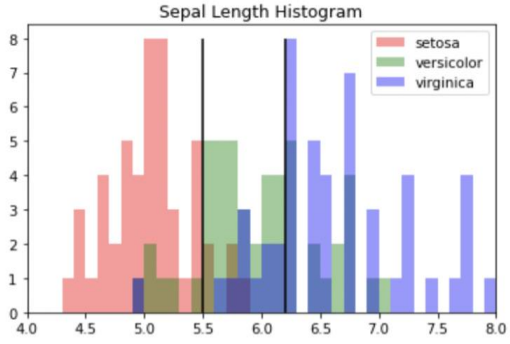
- **Sepal Length**

첫 번째 decision boundary 는 4.3 부터 7.0 까지 0.1 씩 증가시켜보고, 두 번째 decision boundary 는 4.9 부터 7.9 까지 0.1 씩 증가시켜보면서 세 class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

```
final_boundary1 = 0
final_boundary2 = 0
cost = 150
boundary1 = 4.2
for i in range(28):
    boundary1 += 0.1
    boundary1 = round(boundary1, 2)
    boundary2 = 4.8
    for j in range(31):
        boundary2 += 0.1
        boundary2 = round(boundary2, 2)
        if boundary1 > boundary2 : continue

        #첫 번째 decision boundary 미만일 때 setosa class라고 분류한다.
        se_p = df[df['sepal length'] < boundary1] #setosa positive
        se_tp = se_p[se_p['class'] =='Iris-setosa'] #setosa true positive

        #첫 번째 decision boundary 이상이고 두 번째 decision boundary 미만일 때 versicolor class라고 분류한다.
        temp = df[df['sepal length'] >= boundary1] #versicolor positive
        ve_p = temp[temp['sepal length'] < boundary2]
        ve_tp = ve_p[ve_p['class'] == 'Iris-versicolor'] #versicolor true positive

        #두 번째 decision boundary 이상일 때 virginica class라고 분류한다.
        vi_p = df[df['sepal length'] >= boundary2] #virginica positive
        vi_tp = vi_p[vi_p['class'] =='Iris-virginica'] #virginica true positive

        se_fp_n = len(se_p) - len(se_tp) #setosa false positive의 개수
        ve_fp_n = len(ve_p) - len(ve_tp) #versicolor false positive의 개수
        vi_fp_n = len(vi_p) - len(vi_tp) #virginica false positive의 개수

        c = se_fp_n + ve_fp_n + vi_fp_n #총 cost

        #cost가 제일 작은 것으로 decision boundary를 update해준다.
        if c < cost :
            cost = c
            final_boundary1 = boundary1
            final_boundary2 = boundary2
```

이때 cost 는 misclassification 하는 개수, 즉 setosa false positive 의 개수와 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary :  5.5
두 번째 boundary :  6.2
cost :  38
```



그 결과, cost 를 최소로 하는 decision boundary 는 5.5 와 6.2 가 나왔다.

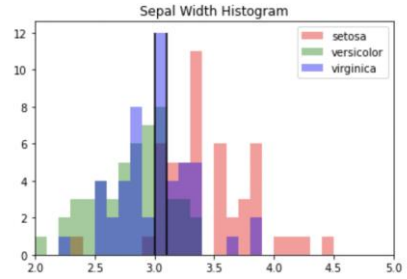
이 decision boundary 로 분류를 할 때 sepal length 가 5.5 미만이면 setosa, 5.5 이상 6.2 미만이면 versicolor, 6.2 이상이면 virginica 이며 cost 는 38 이다.

- **Sepal Width**

첫 번째 decision boundary 는 2.0 부터 3.8 까지 0.1 씩 증가시켜보고, 두 번째 decision boundary 는 2.2 부터 4.4 까지 0.1 씩 증가시켜보면서 세 class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

이때 cost 는 misclassification 하는 개수, 즉 setosa false positive 의 개수와 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary :  3.0
두 번째 boundary :  3.1
cost :  62
<function matplotlib.pyplot.show(close=None, block=None)>
```



그 결과, cost 를 최소로 하는 decision boundary 는 3.0 과 3.1 이 나왔다.

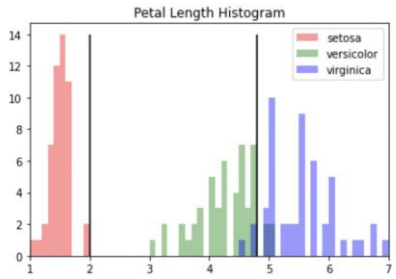
이 decision boundary 로 분류를 할 때 sepal width 가 3.0 미만이면 versicolor, 3.0 이상 3.1 미만이면 virginica, 3.1 이상이면 setosa 이며 cost 는 62 이다.

- **Petal Length**

첫 번째 decision boundary 는 1.0 부터 5.1 까지 0.1 씩 증가시켜보고, 두 번째 decision boundary 는 3.0 부터 6.9 까지 0.1 씩 증가시켜보면서 세 class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

이때 cost 는 misclassification 하는 개수, 즉 setosa false positive 의 개수와 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary :  2.0
두 번째 boundary :  4.8
cost :  7
<function matplotlib.pyplot.show(close=None, block=None)>
```



그 결과, cost 를 최소로 하는 decision boundary 는 2.0 과 4.8 이 나왔다.

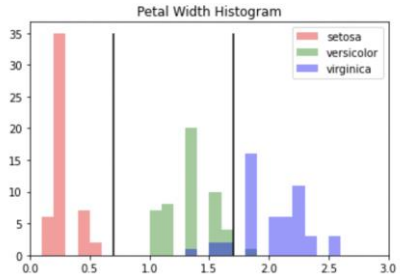
이 decision boundary 로 분류를 할 때 petal length가 2.0 미만이면 setosa, 2.0 이상 4.8 미만이면 versicolor, 4.8이상이면 virginica이며 cost는 7이다.

- **Petal Width**

첫 번째 decision boundary 는 0.1 부터 1.8 까지 0.1 씩 증가시켜보고, 두 번째 decision boundary 는 1.0 부터 2.5 까지 0.1 씩 증가시켜보면서 세 class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

이때 cost 는 misclassification 하는 개수, 즉 setosa false positive 의 개수와 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary :  0.7
두 번째 boundary :  1.7
cost :  6
<function matplotlib.pyplot.show(close=None, block=None)>
```



그 결과, cost 를 최소로 하는 decision boundary 는 0.7 과 1.7 이 나왔다.

이 decision boundary 로 분류를 할 때 petal width 가 0.7 보다 작으면 setosa, 0.7 이상 1.7 미만이면 versicolor, 1.7 이상이면 virginica 이며 cost 는 6 이다.

2.

- **Sepal Length, Sepal Width**

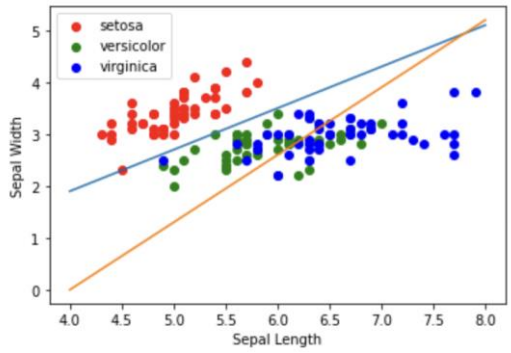
두 decision boundary 는 직선 형태이다. 첫 번째 decision boundary 는 y = ax+b 로 a 는 -1.9 부터 2.0 까지 0.1 씩 증가시켜보고, b 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 setosa class 와 versicolor class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다. 두 번째 decision boundary 는 y = cx+d 로 c 는 -1.9 부터 2.0 까지 0.1 씩 증가시켜보고, b 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 versicolor class 와 virginica class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

```
final_a = 0
final_b = 0
cost_ab = 150

a= -2.0
for i in range(40):
    a += 0.1
    a = round(a,1)
    b= -6.0
    for j in range(120):
        b += 0.1
        b = round(b,1)
        se_ve = df[df['class']!='Iris-virginica']
        boundary= a * se_ve['sepal length'] + b
        se_p = se_ve[se_ve['sepal width'] >= boundary]
        se_tp = se_p[se_p['class'] =='Iris-setosa']

        ve_p = se_ve[se_ve['sepal width'] < boundary]
        ve_tp = ve_p[ve_p['class'] =='Iris-versicolor']

        n_se_fp = len(se_p)-len(se_tp)
        n_ve_fp = len(ve_p)-len(ve_tp)
        cost = n_se_fp + n_ve_fp

        if cost < cost_ab :
            cost_ab = cost
            final_a = a
            final_b = b
```

```
final_c = 0
final_d = 0
cost_cd = 150

c= -2.0
for i in range(40):
    c += 0.1
    c = round(c,1)
    d= -6.0
    for j in range(120):
        d += 0.1
        d = round(d,1)

        ve_vi = df[df['class']!='Iris-setosa']
        boundary= c * ve_vi['sepal length'] + d
        ve_p = ve_vi[ve_vi['sepal width'] >= boundary]
        ve_tp = ve_p[ve_p['class'] =='Iris-versicolor']

        vi_p = ve_vi[ve_vi['sepal width'] < boundary]
        vi_tp = vi_p[vi_p['class'] =='Iris-virginica']

        n_ve_fp = len(ve_p)-len(ve_tp)
        n_vi_fp = len(vi_p)-len(vi_tp)
        cost = n_ve_fp + n_vi_fp
        if cost < cost_cd :
            cost_cd = cost
            final_c = c
            final_d = d
```

이때 cost 는 misclassification 하는 개수, 즉 첫 번째 decision boundary 는 setosa false positive 의 개수와 versicolor false positive 의 개수의 합으로 지정하였고, 두 번째 decision boundary 는 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary : y = 0.8x -1.3
두 번째 boundary : y = 1.3x -5.2
cost :  27
```



그 결과, cost 를 최소로 하는 decision boundary 는 y=0.8x-1.3 과 y=1.3x-5.2 가 나왔다.

x 축은 sepal length 이고 y 축은 sepal width 이다.

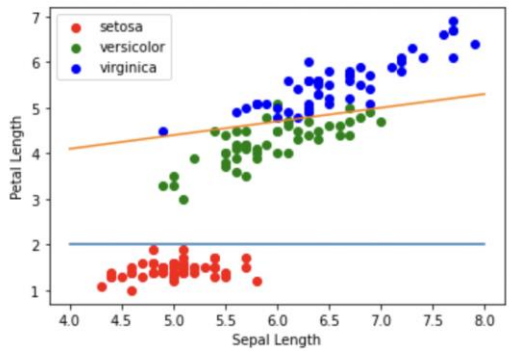
이 decision boundary 로 분류를 할 때 y>=0.8x -1.3 이면 setosa, y<0.8x -1.3 이고 y>=1.3x-5.2 이면 versicolor, y<1.3x-5.2 이면 virginica 이며 cost 는 27 이다.

- **Sepal Length, Petal Length**

두 decision boundary 는 직선 형태이다. 첫 번째 decision boundary 는 y = ax+b 로 a 는 0.0 부터 0.9 까지 0.1 씩 증가시켜보고, b 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 setosa class 와 versicolor class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다. 두 번째 decision boundary 는 y = cx+d 로 c 는 0.0 부터 0.9 까지 0.1 씩 증가시켜보고, d 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 versicolor class 와 virginica class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

이때 cost 는 misclassification 하는 개수, 즉 첫 번째 decision boundary 는 setosa false positive 의 개수와 versicolor false positive 의 개수의 합으로 지정하였고, 두 번째 decision boundary 는 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary : y = 0.0x +2.0
두 번째 boundary : y = 0.3x +2.9
cost :  4
```



그 결과, cost 를 최소로 하는 decision boundary 는 y = 0.1x +1.5 과 y = 0.3x +2.9 가 나왔다.

x 축은 sepal length 이고 y 축은 petal length 이다.

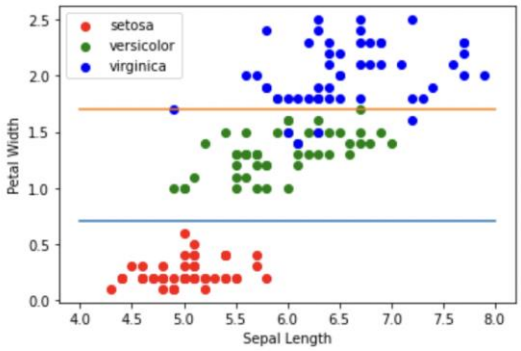
이 decision boundary 로 분류를 할 때 y<0.1x +1.5 이면 setosa, y>=0.1x +1.5 이고 y<0.3x+2.9 이면 versicolor, y>=0.3x+2.9 이면 virginica 이며 cost 는 4 이다.

- **Sepal Length, Petal Width**

두 decision boundary 는 직선 형태이다. 첫 번째 decision boundary 는 y = ax+b 로 a 는 0.0 부터 0.9 까지 0.1 씩 증가시켜보고, b 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 setosa class 와 versicolor class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다. 두 번째 decision boundary 는 y = cx+d 로 c 는 0.0 부터 0.9 까지 0.1 씩 증가시켜보고, d 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 versicolor class 와 virginica class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

이때 cost 는 misclassification 하는 개수, 즉 첫 번째 decision boundary 는 setosa false positive 의 개수와 versicolor false positive 의 개수의 합으로 지정하였고, 두 번째 decision boundary 는 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary : y = 0.0x +0.7
두 번째 boundary : y = 0.0x +1.7
cost :  6
```



그 결과, cost 를 최소로 하는 decision boundary 는 y = 0.0x +0.7 과 y = 0.0x +1.7 가 나왔다.

x 축은 sepal length 이고 y 축은 petal width 이다.

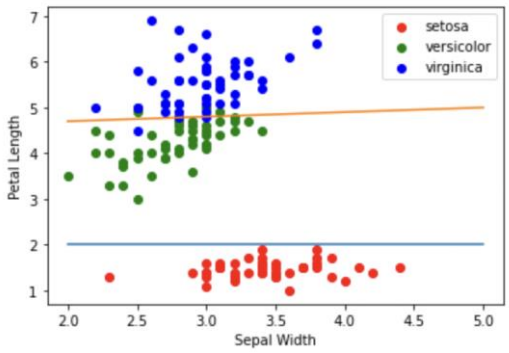
이 decision boundary 로 분류를 할 때 y<0.0x +0.7 이면 setosa, y>=0.0x +0.7 이고 y<0.0x+1.7 이면 versicolor, y>=0.0x+1.7 이면 virginica 이며 cost 는 6 이다.

- **Sepal Width, Petal Length**

두 decision boundary 는 직선 형태이다. 첫 번째 decision boundary 는 y = ax+b 로 a 는 0.0 부터 0.9 까지 0.1 씩 증가시켜보고, b 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 setosa class 와 versicolor class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다. 두 번째 decision boundary 는 y = cx+d 로 c 는 0.0 부터 0.9 까지 0.1 씩 증가시켜보고, d 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 versicolor class 와 virginica class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

이때 cost 는 misclassification 하는 개수, 즉 첫 번째 decision boundary 는 setosa false positive 의 개수와 versicolor false positive 의 개수의 합으로 지정하였고, 두 번째 decision boundary 는 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary : y = 0.0x +2.0
두 번째 boundary : y = 0.1x +4.5
cost :  6
```



그 결과, cost 를 최소로 하는 decision boundary 는 y = 0.0x +2.0 과 y = 0.1x +4.5 가 나왔다.

x 축은 sepal width 이고 y 축은 petal length 이다.

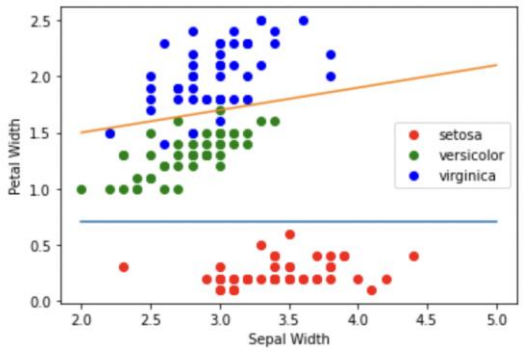
이 decision boundary 로 분류를 할 때 y<0.0x +2.0 이면 setosa, y>=0.0x +2.0 이고 y<0.1x+4.5 이면 versicolor, y>=0.1x+4.5 이면 virginica 이며 cost 는 6 이다.

- **Sepal Width, Petal Width**

두 decision boundary 는 직선 형태이다. 첫 번째 decision boundary 는 y = ax+b 로 a 는 0.0 부터 0.9 까지 0.1 씩 증가시켜보고, b 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 setosa class 와 versicolor class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다. 두 번째 decision boundary 는 y = cx+d 로 c 는 0.0 부터 0.9 까지 0.1 씩 증가시켜보고, d 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 versicolor class 와 virginica class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

이때 cost 는 misclassification 하는 개수, 즉 첫 번째 decision boundary 는 setosa false positive 의 개수와 versicolor false positive 의 개수의 합으로 지정하였고, 두 번째 decision boundary 는 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary : y = 0.0x +0.7
두 번째 boundary : y = 0.2x +1.1
cost :  5
```



그 결과, cost 를 최소로 하는 decision boundary 는 y = 0.0x +0.7 과 y = 0.2x +1.1 가 나왔다.

x 축은 sepal width 이고 y 축은 petal width 이다.

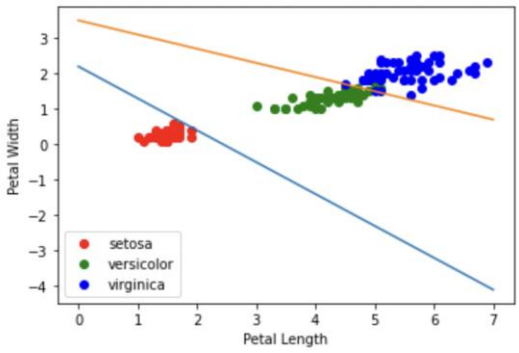
이 decision boundary 로 분류를 할 때 y<0.0x +0.7 이면 setosa, y>=0.0x +0.7 이고 y<0.2x+1.1 이면 versicolor, y>=0.2x+1.1 이면 virginica 이며 cost 는 5 이다.

- **Petal Length, Petal Width**

두 decision boundary 는 직선 형태이다. 첫 번째 decision boundary 는 y = ax+b 로 a 는 -0.9 부터 0.0 까지 0.1 씩 증가시켜보고, b 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 setosa class 와 versicolor class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다. 두 번째 decision boundary 는 y = cx+d 로 c 는 -0.9 부터 0.0 까지 0.1 씩 증가시켜보고, d 는 -5.9 부터 6.0 까지 0.1 씩 증가시켜보면서 versicolor class 와 virginica class 를 분류할 때 cost 를 최소화하는 decision boundary 를 최종 decision boundary 로 정하였다.

이때 cost 는 misclassification 하는 개수, 즉 첫 번째 decision boundary 는 setosa false positive 의 개수와 versicolor false positive 의 개수의 합으로 지정하였고, 두 번째 decision boundary 는 versicolor false positive 의 개수와 virginica false positive 의 개수의 합으로 지정하였다.

```
첫 번째 boundary : y = -0.9x +2.2
두 번째 boundary : y = -0.4x +3.5
cost :  3
```



그 결과, cost 를 최소로 하는 decision boundary 는 y = -0.9x +2.2 과 y = -0.4x +3.5 가 나왔다.

x 축은 petal width 이고 y 축은 petal length 이다.

이 decision boundary 로 분류를 할 때 y<-0.9x +2.2 이면 setosa, y>=-0.9x +2.2 이고 y<-0.4x+3.5 이면 versicolor, y>=-0.4x+3.5 이면 virginica 이며 cost 는 3 이다.

3.

5-fold cross validation 을 하기 위해 iris dataset 을 섞는다.

```
#iris dataset을 섞는다.
import random

idx = list(range(0,150))
random.shuffle(idx)
```

- **Sepal Length feature 를 사용한 classifier**

Evaluation 을 위해 5-fold cross validation 을 하므로 1 번째 fold 부터 5 번째 fold 까지 5 개의 fold 가 있다. for 문으로 i 를 0,1,2,3,4 로 iteration 하고 각 iteration 에서 (i+1)번째 fold 는 evaluation 에 사용하고 (i+1)번째 fold 를 제외한 4 개의 fold 를 training 에 사용한다.

```
idx_eval = idx[i*30: (i+1)*30]
idx_data = idx[:i*30]+idx[(i+1)*30:]
```

Prior 는 전체 120 개 instance 에 있는 각 class 의 instance 의 비율로 설정한다.

```
se_prior = se_size/120
ve_prior = ve_size/120
vi_prior = vi_size/120
```

Sepal length likelihood 의 probability mass function 은 4.0 부터 8.0 까지 0.1 의 간격으로 41 개의 value 에 대해 구한다. 각 class 에서 likelihood 는 특정 value 를 가지는 데이터를 세서 해당 class 에 속하는 전체 데이터 개수로 나눠서 구한다.

```
se_likelihood = np.zeros(41)
ve_likelihood = np.zeros(41)
vi_likelihood = np.zeros(41)

sl = 3.9
for j in range(41):
    sl += 0.1
    sl = round(sl,1)
    index = int(round((sl-4.0)/0.1,1))
    se_likelihood[index] = len(se_data[se_data['sepal length'] == sl])/se_size
    ve_likelihood[index] = len(ve_data[ve_data['sepal length'] == sl])/ve_size
    vi_likelihood[index] = len(vi_data[vi_data['sepal length'] == sl])/vi_size
```

Posterior 는 posterior = likelihood * prior / evidence 식을 이용하여 구한다.

```
se_posterior = np.zeros(41)
ve_posterior = np.zeros(41)
vi_posterior = np.zeros(41)

for j in range(41) :
    evidence = se_likelihood[j] * se_prior + ve_likelihood[j] * ve_prior + vi_likelihood[j] * vi_prior
    if evidence == 0 :
        se_posterior[j] = 0
        ve_posterior[j] = 0
        vi_posterior[j] = 0
        continue

    se_posterior[j] = se_likelihood[j] * se_prior / evidence
    ve_posterior[j] = ve_likelihood[j] * ve_prior / evidence
    vi_posterior[j] = vi_likelihood[j] * vi_prior / evidence
```

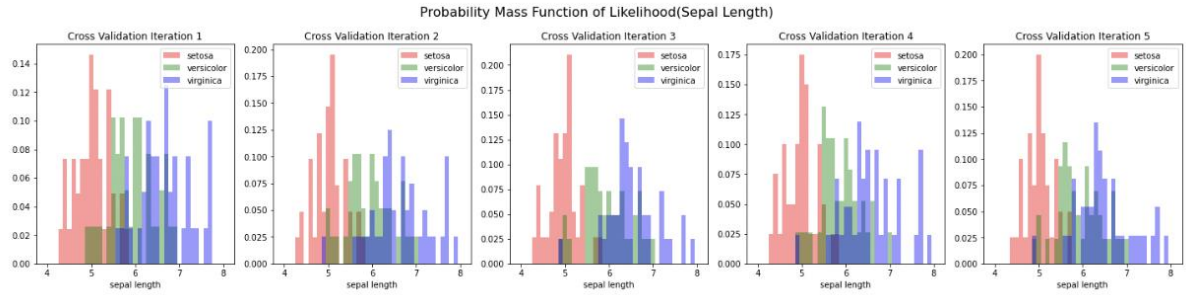
구한 posterior 를 바탕으로 (i+1)번째 fold 에 대해 evaluation 을 한다. Posterior probability 가 가장 높은 class 를 해당 data 의 class 로 분류한다.

```
for j in range(30):
    d = eval_data.iloc[j]
    sl = float(d['sepal length'])
    sl_idx = int(round((sl-4.0)/0.1,1))

    max_prob = max(max(se_posterior[sl_idx],ve_posterior[sl_idx]),vi_posterior[sl_idx])
    if max_prob == se_posterior[sl_idx] :
        eval_prediction.append('Iris-setosa')
        se_p_n += 1
        if d['class'] == 'Iris-setosa' : se_tp_n += 1
    elif max_prob == ve_posterior[sl_idx] :
        eval_prediction.append('Iris-versicolor')
        ve_p_n += 1
        if d['class'] == 'Iris-versicolor' : ve_tp_n += 1
    elif max_prob == vi_posterior[sl_idx] :
        eval_prediction.append('Iris-virginica')
        vi_p_n += 1
        if d['class'] == 'Iris-virginica' : vi_tp_n += 1
```

각 class 에 대한 precision 과 recall 을 구해서 평균을 구하고, 5-fold cross validation 을 통해 나온 5 개의 evaluation 결과의 평균을 이 classifier 의 최종 성능으로 한다.

```
    #각 class에 대한 precision과 recall을 구한다.
    #precision = tp/(tp+fp)
    #recall = tp/(tp+fn)
    if se_p_n == 0 :
        se_precision = 0
        se_recall = 0
    else :
        se_precision = se_tp_n / se_p_n
        se_recall = se_tp_n / (50-se_size)

    if ve_p_n == 0 :
        ve_precision = 0
        ve_recall = 0
    else :
        ve_precision = ve_tp_n / ve_p_n
        ve_recall = ve_tp_n / (50-ve_size)

    if vi_p_n == 0 :
        vi_precision = 0
        vi_recall = 0
    else :
        vi_precision = vi_tp_n / vi_p_n
        vi_recall = vi_tp_n / (50-vi_size)

    #구한 3개의 precision의 평균과 3개의 recall의 평균을 구한다.
    precision.append((se_precision+ve_precision+vi_precision)/3)
    recall.append((se_recall+ve_recall+vi_recall)/3)

plt.tight_layout()
plt.show()

#5-fold cross validation을 통해 나온 5개의 evaluation 결과의 평균을 이 classifier의 최종 성능으로 한다.
mean_precision = sum(precision)/5
mean_recall = sum(recall)/5
```

구한 sepal length likelihood 의 probability mass function 은 다음과 같다.



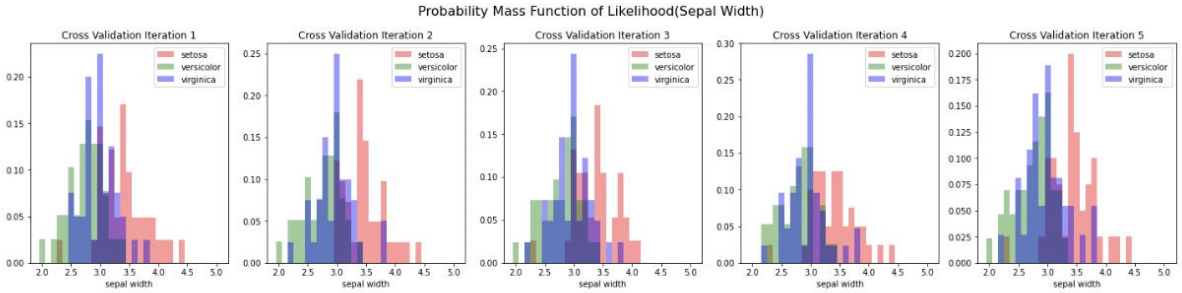

```
precision : 0.6676170126170127
recall : 0.6677154327154328
```

Sepal length feature 를 사용한 classifier 의 성능은 precision 이 약 0.67, recall 이 약 0.67 이다.

- **Sepal Width feature 를 사용한 classifier**

Sepal width likelihood 의 probability mass function 은 2.0 부터 5.0 까지 0.1 의 간격으로 31 개의 value 에 대해 구한다.

구한 sepal width likelihood 의 probability mass function 은 다음과 같다.



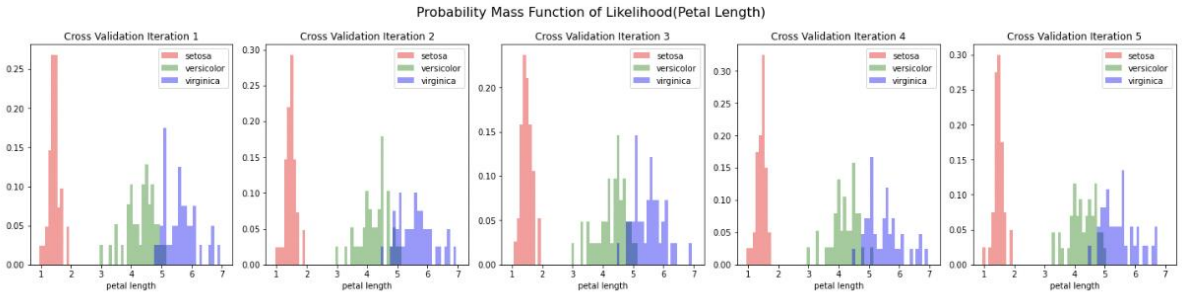

```
precision : 0.5318196182902065
recall : 0.488934398934399
```

Sepal width feature 를 사용한 classifier 의 성능은 precision 이 약 0.53, recall 이 약 0.49 이다.

- **Petal Length feature 를 사용한 classifier**

Petal length likelihood 의 probability mass function 은 1.0 부터 7.0 까지 0.1 의 간격으로 61 개의 value 에 대해 구한다.

구한 petal length likelihood 의 probability mass function 은 다음과 같다.



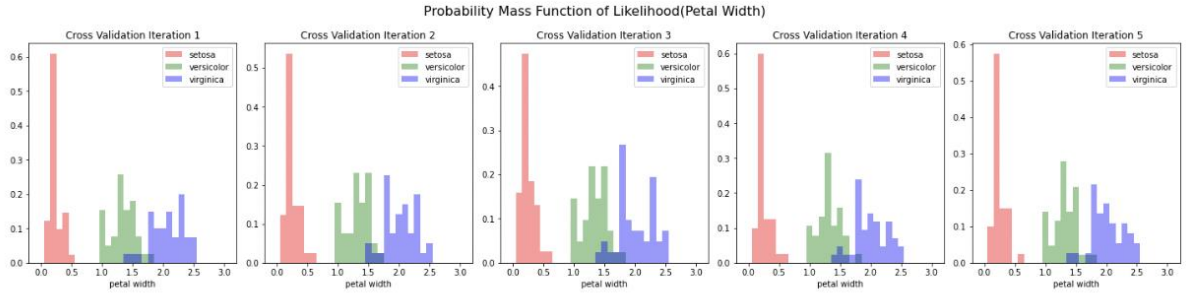

```
precision : 0.8474527324527324
recall : 0.8093164243164243
```

Petal length feature 를 사용한 classifier 의 성능은 precision 이 약 0.85, recall 이 약 0.81 이다.

- **Petal Width feature 를 사용한 classifier**

Petal width likelihood 의 probability mass function 은 0.0 부터 3.0 까지 0.1 의 간격으로 31 개의 value 에 대해 구한다.

구한 petal width likelihood 의 probability mass function 은 다음과 같다.




```
precision : 0.9547940947940947
recall : 0.9581274281274281
```

Petal width feature 를 사용한 classifier 의 성능은 precision 이 약 0.95, recall 이 약 0.96 이다.

- **4 개의 feature 를 모두 사용한 classifier**

Sepal length, Sepal width, Petal length, Petal width feature 들은 서로 independent 하다고 가정하였다. 따라서 각각의 likelihood 를 모두 곱하고 마지막에 prior 를 곱하였다. Evidence 는 normalize 역할만 하고 대소 비교에는 영향을 주지 않으므로 생략하였다.

```
for j in range(30):
    d = eval_data.iloc[j]
    sl = float(d['sepal length'])
    sw = float(d['sepal width'])
    pl = float(d['petal length'])
    pw = float(d['petal width'])
    sl_idx = int(round((sl-4.0)/0.1,1))
    sw_idx = int(round((sw-2.0)/0.1,1))
    pl_idx = int(round((pl-1.0)/0.1,1))
    pw_idx = int(round((pw-0.0)/0.1,1))

    #evidence는 normalize 역할만 하고 대소 비교에는 영향을 주지 않으므로 생략한다.
    se_prob = se_l_sl[sl_idx]*se_l_sw[sw_idx]*se_l_pl[pl_idx]*se_l_pw[pw_idx]*se_prior
    ve_prob = ve_l_sl[sl_idx]*ve_l_sw[sw_idx]*ve_l_pl[pl_idx]*ve_l_pw[pw_idx]*ve_prior
    vi_prob = vi_l_sl[sl_idx]*vi_l_sw[sw_idx]*vi_l_pl[pl_idx]*vi_l_pw[pw_idx]*vi_prior
```

```
precision : 0.8046247563352826
recall : 0.7092266992266991
```

4 개의 feature 를 모두 사용한 classifier 의 성능은 precision 이 약 0.80, recall 이 약 0.71 이다.

4.

5-fold cross validation 을 하기 위해 iris dataset 을 섞는다.

```
#iris dataset을 섞는다.
import random

idx = list(range(0,150))
random.shuffle(idx)
```

Gaussian distribution 을 정의한다.

```
#gaussian distribution을 정의한다.
def gaussian(x, mean, sigma):
    variance = sigma**2
    return (1/np.sqrt(2*np.pi*variance)) * np.exp(-((x-mean)**2)/(2*variance))
```

- **Sepal Length feature 를 사용한 classifier**

  Gaussian distribution 의 parameter 인 mean 과 standard deviation 을 구해 sepal length likelihood 를 gaussian distribution 으로 모델링한다.

```
#sepal length gaussian distribution
se_mean = se_data['sepal length'].mean()
se_std = se_data['sepal length'].std()

ve_mean = ve_data['sepal length'].mean()
ve_std = ve_data['sepal length'].std()

vi_mean = vi_data['sepal length'].mean()
vi_std = vi_data['sepal length'].std()

#sepal length likelihood를 gaussian distribution으로 모델링한다.
se_gaussian = np.zeros(41)
ve_gaussian = np.zeros(41)
vi_gaussian = np.zeros(41)

sl = 3.9
for j in range(41):
    sl += 0.1
    sl = round(sl,1)
    index = int(round((sl-4.0)/0.1,1))
    se_gaussian[index] = gaussian(sl, se_mean, se_std)
    ve_gaussian[index] = gaussian(sl, ve_mean, ve_std)
    vi_gaussian[index] = gaussian(sl, vi_mean, vi_std)
```

  Prior 는 전체 120 개 instance 에 있는 각 class 의 instance 의 비율로 설정한다.

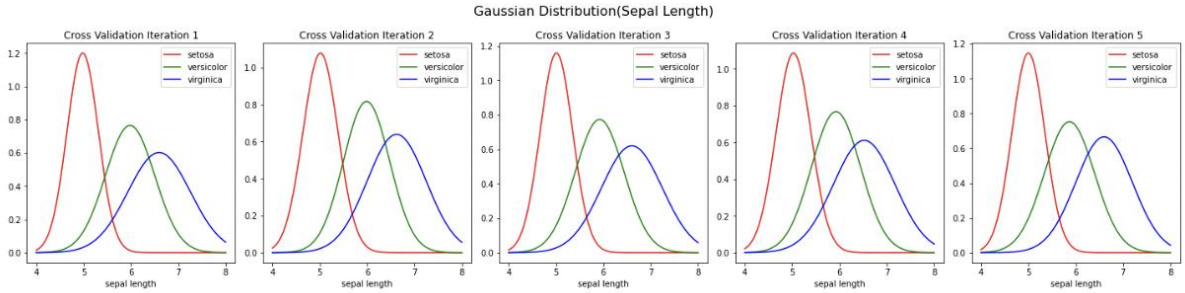
  Posterior 는 posterior = likelihood * prior / evidence 식을 이용하여 구한다.

```
for j in range(41) :
    evidence = se_gaussian[j] * se_prior + ve_gaussian[j] * ve_prior + vi_gaussian[j] * vi_prior
    if evidence == 0 :
        se_posterior[j] = 0
        ve_posterior[j] = 0
        vi_posterior[j] = 0
        continue

    se_posterior[j] = se_gaussian[j] * se_prior / evidence
    ve_posterior[j] = ve_gaussian[j] * ve_prior / evidence
    vi_posterior[j] = vi_gaussian[j] * vi_prior / evidence
```

  구한 posterior 를 바탕으로 (i+1)번째 fold 에 대해 evaluation 을 한다. Posterior probability 가 가장 높은 class 를 해당 data 의 class 로 분류한다. 모든 data 에 대해 p(error|x)가 최소화되도록 선택했기 때문에 총 cost 인 p(error)도 최소화된다.

  각 class 에 대한 precision 과 recall 을 구해서 평균을 구하고, 5-fold cross validation 을 통해 나온 5 개의 evaluation 결과의 평균을 이 classifier 의 최종 성능으로 한다.

  구한 sepal length 의 gaussian distribution 은 다음과 같다.



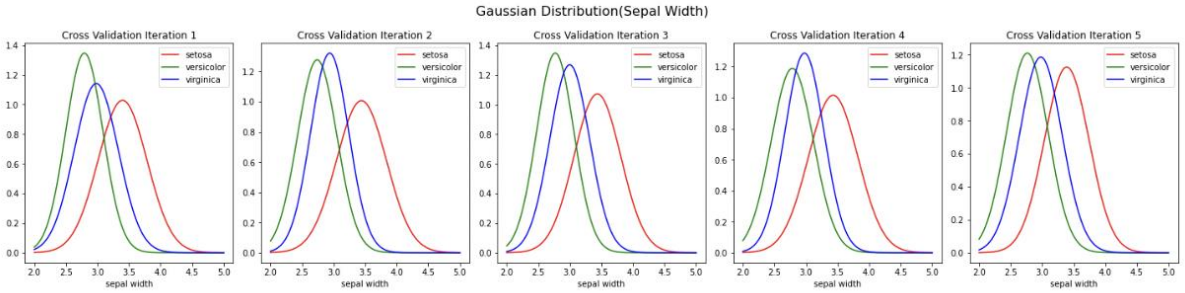

```
precision : 0.7393445443445443
recall : 0.7307094757094758
```

Sepal length feature 를 사용한 classifier 의 성능은 precision 이 약 0.74, recall 이 약 0.73 이다.

- **Sepal Width feature 를 사용한 classifier**

Gaussian distribution 의 parameter 인 mean 과 standard deviation 을 구해 sepal width likelihood 를 gaussian distribution 으로 모델링한다.

구한 sepal width 의 gaussian distribution 은 다음과 같다.



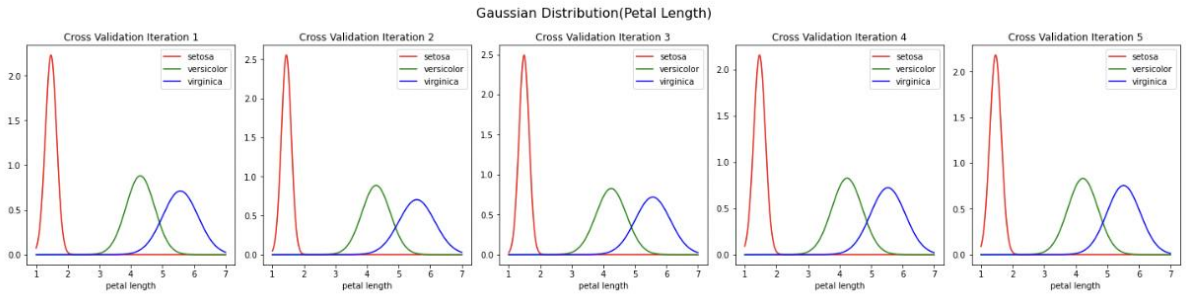

```
precision : 0.5808870564133721
recall : 0.5676551226551226
```

Sepal width feature 를 사용한 classifier 의 성능은 precision 이 약 0.58, recall 이 약 0.57 이다.

- **Petal Length feature 를 사용한 classifier**

Gaussian distribution 의 parameter 인 mean 과 standard deviation 을 구해 petal length likelihood 를 gaussian distribution 으로 모델링한다.

구한 petal length 의 gaussian distribution 은 다음과 같다.



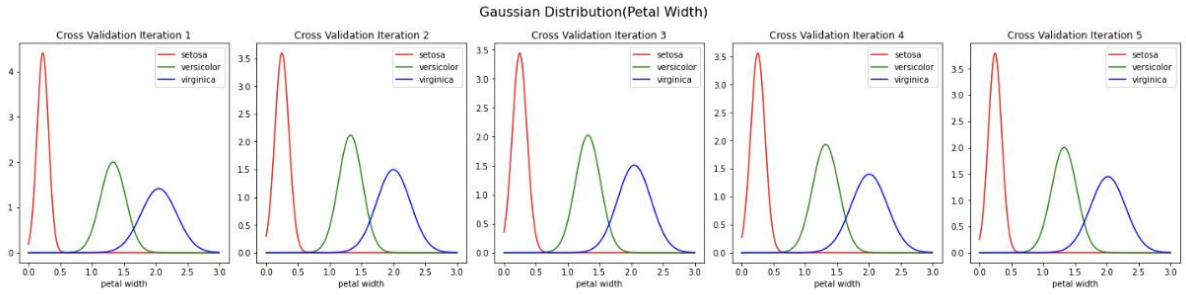

```
precision : 0.9561279461279462
recall : 0.9540909090909091
```

Petal length feature 를 사용한 classifier 의 성능은 precision 이 약 0.96, recall 이 약 0.95 이다.

- **Petal Width feature 를 사용한 classifier**

Gaussian distribution 의 parameter 인 mean 과 standard deviation 을 구해 petal width likelihood 를 gaussian distribution 으로 모델링한다.

구한 petal width 의 gaussian distribution 은 다음과 같다.




```
precision : 0.9562626262626261
recall : 0.9523232323232322
```

Petal width feature 를 사용한 classifier 의 성능은 precision 이 약 0.96, recall 이 약 0.95 이다.

5.

5 번을 위해 4 번에서 5-fold cross validation 할 때 계산한 gaussian distribution 을 저장하였다.

```
#5번을 위해 5-fold cross validation할 때 계산한 gaussian distribution을 저장한다.
se_sl_gaussian = []
se_sw_gaussian = []
se_pl_gaussian = []
se_pw_gaussian = []
ve_sl_gaussian = []
ve_sw_gaussian = []
ve_pl_gaussian = []
ve_pw_gaussian = []
vi_sl_gaussian = []
vi_sw_gaussian = []
vi_pl_gaussian = []
vi_pw_gaussian = []
```

5 번에서는 저장된 gaussian distribution 을 사용하였다.

Sepal length, Sepal width, Petal length, Petal width feature 들은 서로 independent 하다고 가정하였다. 따라서 각각의 gaussian 을 모두 곱하고 마지막에 prior 를 곱하였다. Evidence 는 normalize 역할만 하고 대소 비교에는 영향을 주지 않으므로 생략하였다.

```
#evidence는 normalize 역할만 하고 대소 비교에는 영향을 주지 않으므로 생략한다.
se_prob = se_sl_gaussian[i][sl_idx] * se_sw_gaussian[i][sw_idx] * se_pl_gaussian[i][pl_idx] * se_pw_gaussian[i][pw_idx] * se_prior
ve_prob = ve_sl_gaussian[i][sl_idx] * ve_sw_gaussian[i][sw_idx] * ve_pl_gaussian[i][pl_idx] * ve_pw_gaussian[i][pw_idx] * ve_prior
vi_prob = vi_sl_gaussian[i][sl_idx] * vi_sw_gaussian[i][sw_idx] * vi_pl_gaussian[i][pl_idx] * vi_pw_gaussian[i][pw_idx] * vi_prior
```

```
precision : 0.9609764309764308
recall : 0.9606565656565657
```

4 개의 feature 를 모두 사용한 classifier 의 성능은 precision 이 약 0.96, recall 이 약 0.96 이다. Multivariate gaussian model 을 사용하니 성능이 좋은 것을 알 수 있다.

Procedure of execution

이번 과제는 jupyter notebook 을 이용해 python 으로 구현하였다.

HW2_1.ipynb 는 Part B 의 1 번에 대한 코드이다.

HW2_2.ipynb 는 Part B 의 2 번에 대한 코드이다.

HW2_3.ipynb 는 Part B 의 3 번에 대한 코드이다.

HW2_4_5.ipynb 는 Part B 의 4 번과 Part B 의 5 번에 대한 코드이다.

네 파일 모두 실행 할 때 iris.data 파일이 "./iris/iris.data" 경로에 위치해야 한다.