0416025 呂翊愷

圖形識別 Assignment #1

Methods you have implemented

I. Bayesian classifier (use Gaussian pdfs with maximum-likelihood estimation)

先將原始dataset的順序隨機排序，然後利用cross validation將dataset分出training data和testing data後，將training data中每個類別的mean和covariance算出來，得到每個類別的maximum likelihood。接著在testing data的部分，一筆一筆資料處理，將該筆testing data每個類別的Gaussian pdf算出來，和該類別的機率(priori probability)相乘後得到Discriminant function，將每個類別的Discriminant function比大小，最大值的類別即為該testing data的預測類別。

$$\hat{\boldsymbol{\mu}}_{ML} = \frac{1}{N}\sum_{k=1}^{N}\boldsymbol{x}_k \qquad \hat{\Sigma}_{ML} = \frac{1}{N}\sum_{k=1}^{N}(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})^T$$

$$p(\boldsymbol{x} \mid \omega_i) = \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^l |\Sigma_i|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i)\right)$$

$$g_i(\boldsymbol{x}) = \ln[\, p(\boldsymbol{x} \mid \omega_i) P(\omega_i)]$$

$$g_{ij}(\boldsymbol{x}) = g_i(\boldsymbol{x}) - g_j(\boldsymbol{x})$$

II. Naïve-Bayes classifier

先將原始dataset的順序隨機排序，然後利用cross validation將dataset分出training data和testing data後，將training data中每個類別的每個attribute的mean和variance算出來。接著在testing data的部分，一筆一筆資料處理，將該筆testing data每個類別的每個attribute的pdf算出來後相乘，再和該類別的機率(priori probability)相乘後得到posteriori probability，將每個類別的posteriori probability比大小，最大值的類別即為該testing data的預測類別。

$$p(x \mid \omega_i) = \frac{1}{\sqrt{2\pi}\,\sigma_i}\exp\left(-\frac{1}{2\sigma_i^2}(x - \mu_i)^2\right) \qquad p(\boldsymbol{x} \mid \omega_i) = \prod_{j=1}^{l} p(x_j \mid \omega_i)$$

III. Linear classifier (by Perceptron Algorithm)

先將原始dataset的順序隨機排序，然後利用cross validation將dataset分出training data和testing data後，為了找出decision boundary，設一個向量w依序和每筆training data所形成的向量x相乘，如果結果大於0就分到第一類，小於等於0就分到第二類。如果預測出來的類別和實際的類別y不同，就更新w為w = w + y * x，之後就如此反覆用w繼續對之後的training data預測分類，由於計算量過於龐大，因此我設定分類達到10次後即停止，而效果和分類100次的效果差不多，估計是因為w更新10次後就漸漸收斂，之後的更新也不會變動太大。接著在testing data的部分，一筆一筆資料處理，將training data時最後得到的w和testing data所形成的向量x相乘，如果結果大於0就分到第一類，小於等於0就分到第二類。

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} \qquad \text{with} \quad \boldsymbol{w} \leftarrow \begin{bmatrix} \boldsymbol{w} \\ w_0 \end{bmatrix} \qquad \boldsymbol{x} \leftarrow \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta_t\, \boldsymbol{x}(t)\delta_{\boldsymbol{x}(t)}$$

where

$$\delta_{\boldsymbol{x}} = \begin{cases} +1 & \text{if } \boldsymbol{x} \text{ is misclassified and } \boldsymbol{x} \in \omega_1 \\ -1 & \text{if } \boldsymbol{x} \text{ is misclassified and } \boldsymbol{x} \in \omega_2 \\ 0 & \text{if } \boldsymbol{x} \text{ is classified correctly} \end{cases}$$

IV. Confusion matrix

根據testing data被classifier預測的類別和實際類別，將confusion matrix算出來，再用pyplot將confusion matrix陣列顯示出來。

V. ROC curve

將預測testing data過程中的所有threshold記錄下來，並將threshold由大排到小，依序用這些threshold再度將testing data再分類一次，如果該testing data的g值大於等於threshold就分到第一類，否則就分到第二類。根據testing data被預測的類別和實際類別，將結果分成TN, FP, FN, TP四種，並算出PD和FA，最後將所有的PD和FA用pyplot畫出曲線圖。

VI. AUC

由於AUC是ROC curve線下的面積，所以我將每個ROC curve上前後的點，以計算梯形的面積的方式來估計兩點間的線下面積，最後再加總起來，得到AUC的估計值。

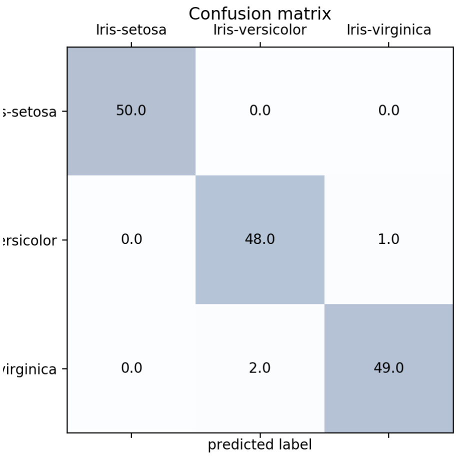
Experiments you have done, and the results

I. Bayesian classifier (use Gaussian pdfs with maximum-likelihood estimation)

經過多次的測試，發現Bayesian classifier在k = 4下的交叉驗證表現最穩，準確率最高，因此以下測試結果皆以k = 4做測試。此外，由於Ionosphere dataset的前兩個feature會使準確率較低，因此將前兩個feature拿掉。

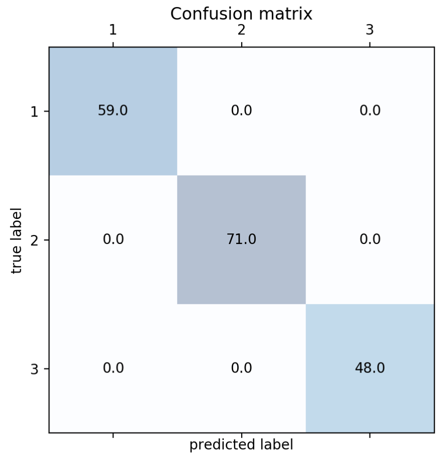
Iris dataset

```
accuracy: 0.980000
```



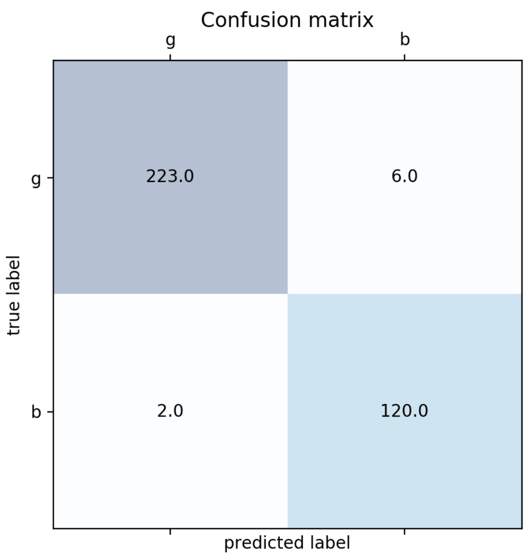
Wine dataset

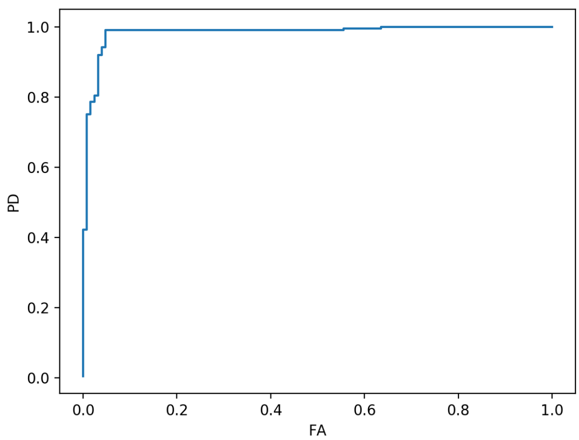
```
accuracy: 1.000000
```



Ionosphere dataset

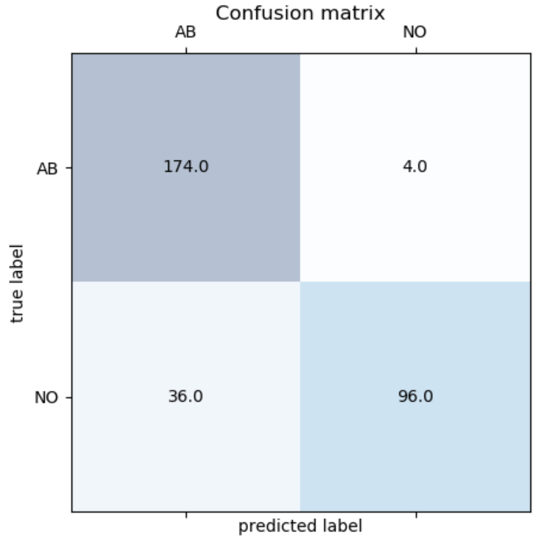
```
accuracy: 0.977208
AUC: 0.984233
```

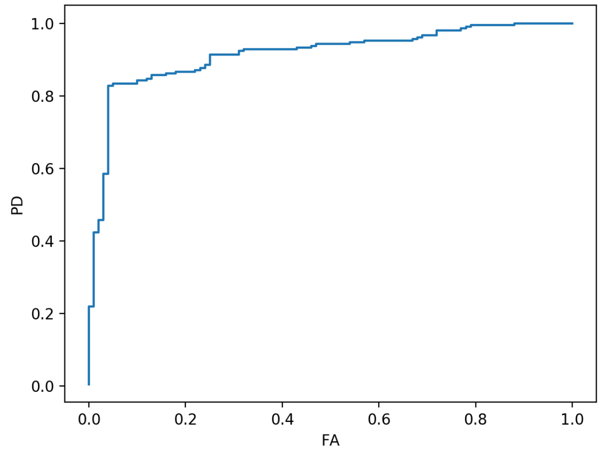




Vertebral dataset

```
accuracy: 0.870968
AUC: 0.915952
```



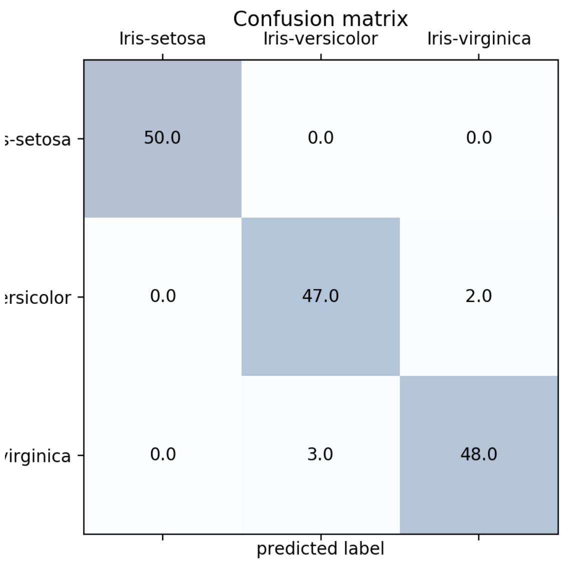


II. Naïve-Bayes classifier

經過多次的測試，發現Naïve-Bayes classifier在k = 10下的交叉驗證表現最穩，準確率最高，因此以下測試結果皆以k = 10做測試。此外，由於Ionosphere dataset的前兩個feature會使準確率較低，因此將前兩個feature拿掉。

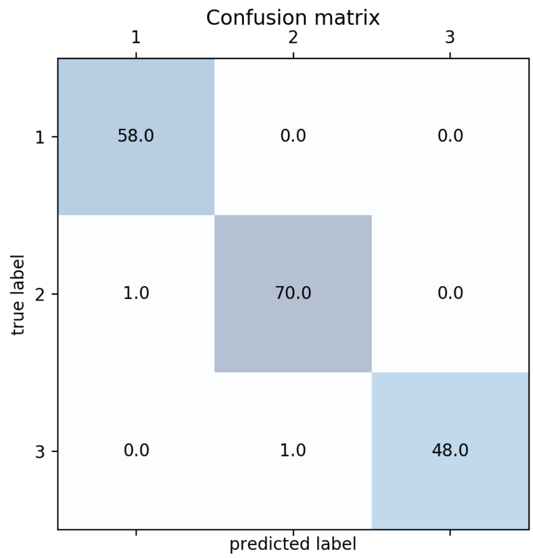
Iris dataset

```
accuracy: 0.966667
```



Wine dataset

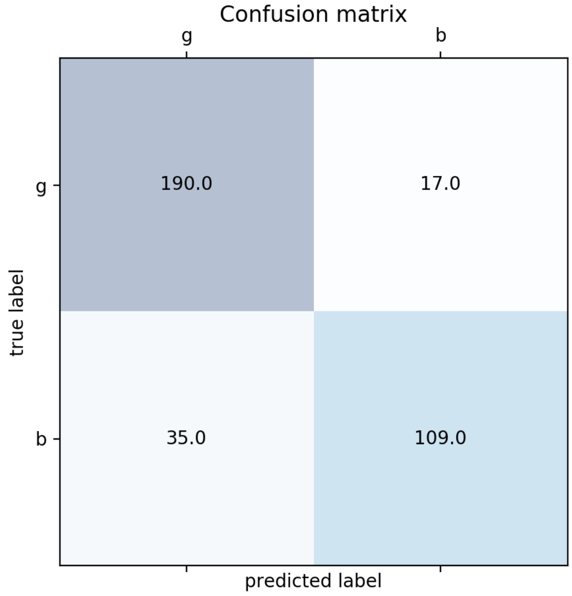
```
accuracy: 0.988764
```

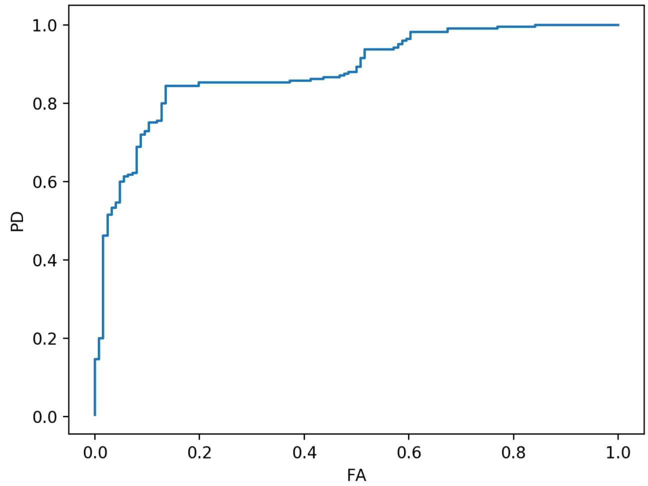


0416025 呂翊愷

Ionosphere dataset

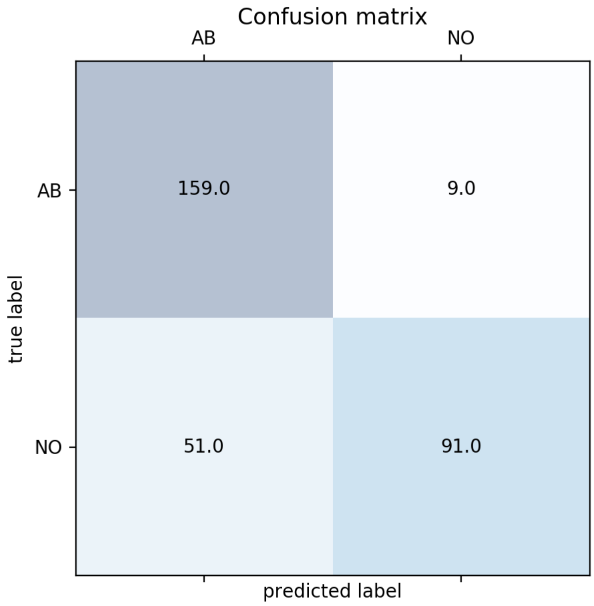
```
accuracy: 0.851852
AUC: 0.883457
```

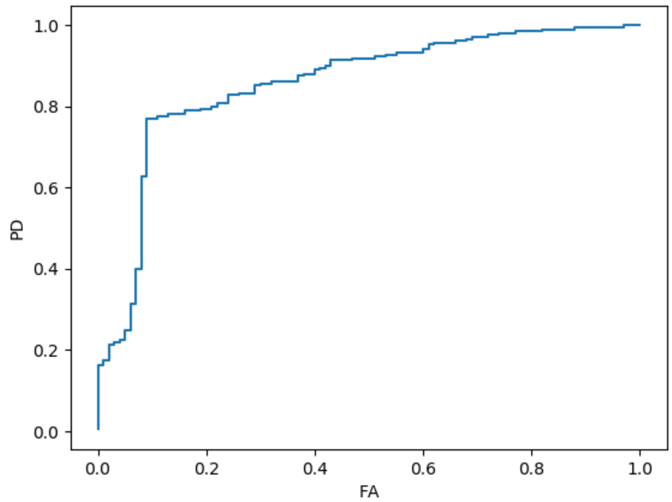




Vertebral dataset

```
accuracy: 0.806452
AUC: 0.857190
```



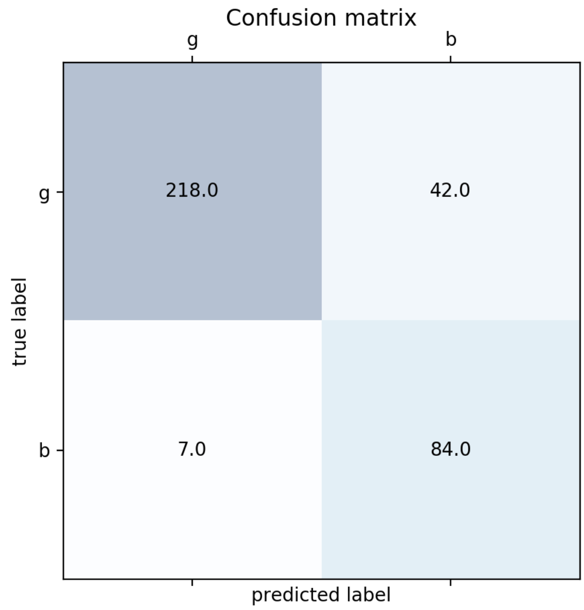


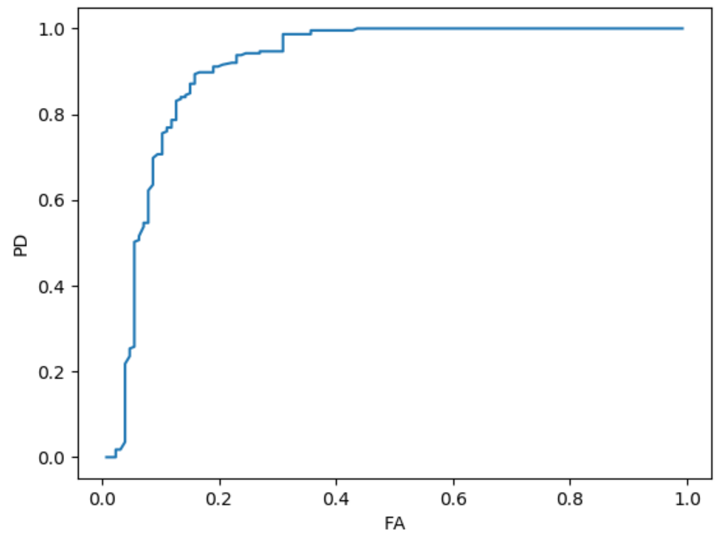
III. Linear classifier (by Perceptron Algorithm)

經過多次的測試，發現Linear classifier在k = 10下的交叉驗證表現最穩，準確率最高，因此以下測試結果皆以k = 10做測試。此外，由於Ionosphere dataset的前兩個feature會使準確率較低，因此將前兩個feature拿掉。為了讓執行時間不要太久，經過測試發現w值更新10次，就會有不錯的準確率，因此設定w值最多只能更新10次，否則如果dataset並非seperable，程式將無止盡跑下去。

Ionosphere dataset

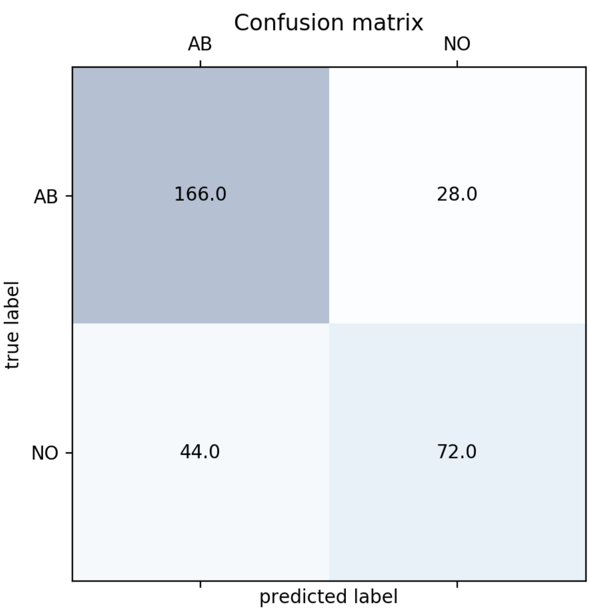
```
accuracy: 0.860399
AUC: 0.899453
```

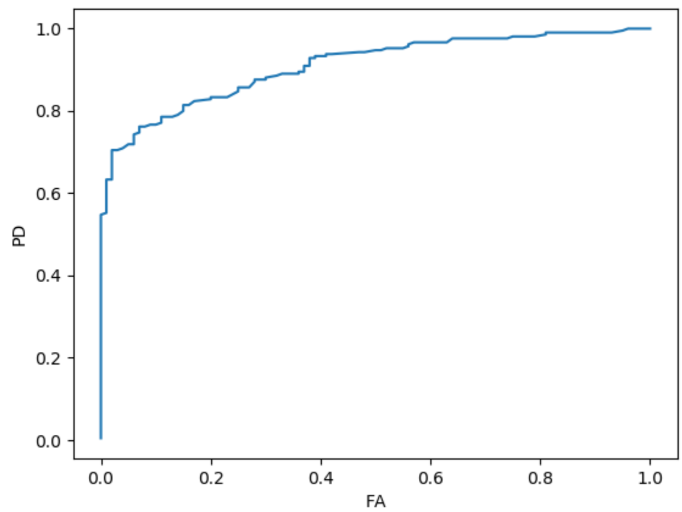




Vertebral dataset

```
accuracy: 0.767742
AUC: 0.909595
```





0416025 呂翊愷

Analysis - Are the results what you expect? Why?

比較這三個分類器，我發現準確率從高到低依序為Bayesian classifier, Naïve-Bayes classifier, Linear classifier，其中最高的是Bayesian classifier，其準確率有時甚至還達到100%，而最低的是Linear classifier，對於這樣的結果其實我並不意外。有些dataset不見得能找出一條線來完美區分兩類data，總是會有一些data和別類的data混合在一起，而linear classifier對於分類這種data會不好區分，造成準確率較低，但整體而言準確率也算是高。

另外，我發現在不同分類器下不同dataset的準確率高低排序相當一致，依序為Wine, Iris, Ionosphere, Vertebral，而我認為原因可能跟dataset本身的attribute分佈有關。如果dataset中不同類別的attribute分佈差很多的話，則dataset就會容易被分類。反之，如果dataset中不同類別的attribute分佈差太少的話，則dataset在分類時就容易被誤判。

0416025 呂翊愷

Code

I. Bayesian classifier (use Gaussian pdfs with maximum-likelihood estimation)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
import statistics as stat
import random
from scipy.stats import norm
from scipy.stats import multivariate_normal
import math

def maxLikelihood(data):
    data = pd.DataFrame(data)
    newData = data.drop("variety", axis=1)

    mu = newData.sum()
    mu = mu.values.reshape([newData.shape[1], 1])
    mu = mu / newData.shape[0]

    cov = np.zeros([newData.shape[1], newData.shape[1]])

    for index, row in newData.iterrows():
        xk = row.values.reshape((newData.shape[1], 1))
        vector = xk - mu

        for i in range(newData.shape[1]):
            for j in range(newData.shape[1]):
                cov[i][j] += vector[i] * vector[j]

    cov = cov / newData.shape[0]

    return mu, cov

def gi(data, mu, cov, P):
    data = pd.DataFrame(data)
    newData = data.drop("variety", axis=0)

    x = np.zeros([newData.shape[0], 1])
    for i in range(len(x)):
        x[i] = newData.ix[i]


    detCov = np.linalg.det(cov)

    expPow = -1 / 2 * np.dot(np.transpose(x - mu),
np.dot(np.linalg.inv(cov), x - mu))
    div = math.pow(2 * math.pi, len(dataVariety)) * abs(detCov)
    pdfVal = 1 / math.sqrt(div) * math.exp(expPow)

    gi = pdfVal * P
```

```
    # gi = math.log(pdfVal * P)

    return gi


# load iris dataset from csv
dataset = pd.read_csv('./iris.csv', names=['sepal.length', 'sepal.width',
'petal.length', 'petal.width', 'variety'], skiprows=0)

# # load wine dataset from csv
# names=['variety', 'alco', 'malic', 'ash', 'alcal', 'mag', 'total',
'flav', 'nonflav', 'proan', 'color', 'hue', 'OD', 'proline']
# dataset = pd.read_csv('./wine.csv', names=['variety', 'alco', 'malic',
'ash', 'alcal', 'mag', 'total',
#                                                 'flav', 'nonflav',
'proan', 'color', 'hue', 'OD', 'proline'], skiprows=0)

# # load ionos dataset from csv
# dataset = pd.read_csv('./ionosphere.csv', names=['c', 'd', 'e', 'f',
'g',
#                                                 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
#                                                       'u', 'v', 'w',
'x', 'y', 'z', 'ab', 'bc', 'cd', 'de', 'ef',
#                                                        'fg', 'gh', 'hi',
'variety'], skiprows=0)

# # load vertebral dataset from csv
# names=['a', 'b', 'c', 'd', 'e', 'f', 'variety']
# dataset = pd.read_csv('./vertebral.csv', names=['a', 'b', 'c', 'd',
'e', 'f', 'variety'], skiprows=0)



dataVariety = dataset['variety'].unique()


# randomize the iris data
randomData = dataset.sample(frac=1)
datasize = randomData.shape[0]

KFold = 4
success = 0
confusion_mat = np.zeros([len(dataVariety), len(dataVariety)])
g_unsort = []
testIdX = []
for i in range(KFold):
    # split the train data and test data
    trainData = randomData[:int(datasize * i / KFold) + int(datasize * (i
+ 1) / KFold):]
    testData = randomData[int(datasize * i / KFold):int(datasize * (i +
1) / KFold)]

    mu = np.zeros((len(dataVariety), trainData.shape[1] - 1, 1))
    cov = np.zeros((len(dataVariety), trainData.shape[1] - 1,
trainData.shape[1] - 1))
```

```
    prob = []

    # train data
    for i in range(len(dataVariety)):
        classData = trainData[trainData['variety'] == dataVariety[i]]
        mu[i], cov[i] = maxLikelihood(classData)
        prob.append(classData.shape[0] / trainData.shape[0])

    # test data
    for index, row in testData.iterrows():
        g = np.zeros([len(dataVariety)])
        for i in range(len(dataVariety)):
            g[i] = gi(row, mu[i], cov[i], prob[i])
        g_unsort.append(g[0] - g[1])
        testIdX.append(index)
        predict = np.argmax(g)


        if (dataVariety[predict] == row["variety"]):
            success += 1

        real = -1
        for i in range(len(dataVariety)):
            if dataVariety[i] == row["variety"]:
                real = i
        confusion_mat[predict][real] += 1


print("accuracy: %f" %(success / dataset.shape[0]))

# draw confusion matrix
fig, axis = plt.subplots(figsize=(5, 5))
axis.matshow(confusion_mat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confusion_mat.shape[0]):
    for j in range(confusion_mat.shape[1]):
        axis.text(x=j, y=i, s=confusion_mat[i,j], va='center',
ha='center')

tick_marks = np.arange(len(dataVariety))
plt.xticks(tick_marks, dataVariety, rotation=0)
plt.yticks(tick_marks, dataVariety)

plt.xlabel('predicted label')
plt.ylabel('true label')
plt.title('Confusion matrix')
plt.show()



if (len(dataVariety) == 2):
    # draw ROC curve
    g_sort = sorted(g_unsort, reverse=True)
    TPR = []
    FPR = []
    labels = np.zeros([dataset.shape[0]])
```

```
    predPosProb = np.zeros([dataset.shape[0]])


    for i in range(len(g_sort)):
        TP = 0
        FP = 0
        FN = 0
        TN = 0

        threshold = g_sort[i]
        for j in range(len(g_unsort)):
            if g_unsort[j] >= threshold and dataset.ix[testIdX[j],
"variety"] == dataVariety[0]:
                TP += 1
                predPosProb[testIdX[j]] += 1
            elif g_unsort[j] >= threshold and dataset.ix[testIdX[j],
"variety"] == dataVariety[1]:
                FP += 1
                predPosProb[testIdX[j]] += 1
            elif g_unsort[j] < threshold and dataset.ix[testIdX[j],
"variety"] == dataVariety[0]:
                FN += 1
            elif g_unsort[j] < threshold and dataset.ix[testIdX[j],
"variety"] == dataVariety[1]:
                TN += 1
        TPR.append(TP / (TP + FN))
        FPR.append(FP / (FP + TN))


    plt.xlabel('FA')
    plt.ylabel('PD')
    plt.plot(FPR, TPR)
    plt.show()

    # calculate AUC
    auc = 0
    for i in range(len(TPR) - 1):
        auc += (TPR[i] + TPR[i + 1]) * (FPR[i + 1] - FPR[i]) / 2
    print("AUC: %f" %(auc))
```

II. Naïve-Bayes classifier

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
import statistics as stat
import random
from scipy.stats import norm
import math

def pdf(x, mean, var):
    pdfVal = 1 / math.sqrt(2 * math.pi * var) * math.exp(-pow((x - mean),
2) / (2 * var))
```

```
    return pdfVal

# load iris dataset from csv
names=['sepal.length', 'sepal.width', 'petal.length', 'petal.width',
'variety']
dataset = pd.read_csv('./iris.csv', names=['sepal.length', 'sepal.width',
'petal.length', 'petal.width', 'variety'], skiprows=0)


# # load wine dataset from csv
# names=['variety', 'alco', 'malic', 'ash', 'alcal', 'mag', 'total',
'flav', 'nonflav', 'proan', 'color', 'hue', 'OD', 'proline']
# dataset = pd.read_csv('./wine.csv', names=['variety', 'alco', 'malic',
'ash', 'alcal', 'mag', 'total',
#                                                 'flav', 'nonflav',
'proan', 'color', 'hue', 'OD', 'proline'], skiprows=0)


# # load ionos dataset from csv
# names=['c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't',
#                                                        'u', 'v', 'w',
'x', 'y', 'z', 'ab', 'bc', 'cd', 'de', 'ef',
#                                                        'fg', 'gh', 'hi',
'variety']
# dataset = pd.read_csv('./ionosphere.csv', names=['c', 'd', 'e', 'f',
'g',
#                                                 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
#                                                        'u', 'v', 'w',
'x', 'y', 'z', 'ab', 'bc', 'cd', 'de', 'ef',
#                                                        'fg', 'gh', 'hi',
'variety'], skiprows=0)


# # load vertebral dataset from csv
# names=['a', 'b', 'c', 'd', 'e', 'f', 'variety']
# dataset = pd.read_csv('./vertebral.csv', names=['a', 'b', 'c', 'd',
'e', 'f', 'variety'], skiprows=0)

dataVariety = dataset['variety'].unique()

# randomize the input data
randomData = dataset.sample(frac=1)
datasize = randomData.shape[0]

KFold = 10
success = 0
confusion_mat = np.zeros([len(dataVariety), len(dataVariety)])
g = []
testIdX = []
for i in range(KFold):
    # split the train data and test data
    trainData = randomData[:int(datasize * i / KFold) + int(datasize * (i
+ 1) / KFold):]
```

```
    testData = randomData[int(datasize * i / KFold):int(datasize * (i +
1) / KFold)]

    # train data
    mean = np.zeros((len(dataVariety), trainData.shape[1] - 1))
    var = np.zeros((len(dataVariety), trainData.shape[1] - 1))

    prob = []

    for i in range(len(dataVariety)):
        classData = trainData[trainData['variety'] == dataVariety[i]]
        classData = classData.drop("variety", axis=1)
        prob.append(classData.shape[0] / trainData.shape[0])
        for j in range(classData.shape[1]):
            classDataAttr = classData.iloc[:, j]
            mean[i][j] = stat.mean(classDataAttr)
            var[i][j] = stat.variance(classDataAttr)

    # test data
    for index, row in testData.iterrows():
        pdfVal = np.zeros((len(dataVariety), testData.shape[1] - 1))
        post = np.zeros([len(dataVariety)])
        for i in range(len(dataVariety)):
            multiPdf = 1
            newj = 0
            for j in range(testData.shape[1]):
                if names[j] == "variety":
                    continue
                pdfVal[i][newj] = pdf(row.ix[j], mean[i][newj], var[i]
[newj])
                multiPdf = multiPdf * pdfVal[i][newj]
                newj = newj + 1
            post[i] = prob[i] * multiPdf
        g.append(post[0] - post[1])
        testIdX.append(index)
        predict = np.argmax(post)

        if (dataVariety[predict] == row["variety"]):
            success += 1

        real = -1
        for i in range(len(dataVariety)):
            if dataVariety[i] == row["variety"]:
                real = i
        confusion_mat[predict][real] += 1

accuracy = success / dataset.shape[0]
print("accuracy: %f" %(accuracy))
# print(confusion_mat)


# draw confusion matrix
fig, axis = plt.subplots(figsize=(5, 5))
axis.matshow(confusion_mat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confusion_mat.shape[0]):
    for j in range(confusion_mat.shape[1]):
```

```
        axis.text(x=j, y=i, s=confusion_mat[i,j], va='center',
ha='center')

tick_marks = np.arange(len(dataVariety))
plt.xticks(tick_marks, dataVariety, rotation=0)
plt.yticks(tick_marks, dataVariety)

plt.xlabel('predicted label')
plt.ylabel('true label')
plt.title('Confusion matrix')
plt.show()


if (len(dataVariety) == 2):
    # draw ROC curve
    g_sort = sorted(g, reverse=True)
    TPR = []
    FPR = []
    labels = np.zeros([dataset.shape[0]])
    predPosProb = np.zeros([dataset.shape[0]])

    for i in range(len(g_sort)):
        TP = 0
        FP = 0
        FN = 0
        TN = 0

        threshold = g_sort[i]
        for j in range(len(g)):
            if g[j] >= threshold and dataset.ix[testIdX[j], "variety"] ==
dataVariety[0]:
                TP += 1
                predPosProb[testIdX[j]] += 1
                labels[testIdX[j]] = 1
            elif g[j] >= threshold and dataset.ix[testIdX[j], "variety"]
== dataVariety[1]:
                FP += 1
                predPosProb[testIdX[j]] += 1
                labels[testIdX[j]] = 0
            elif g[j] < threshold and dataset.ix[testIdX[j], "variety"]
== dataVariety[0]:
                FN += 1
                labels[testIdX[j]] = 1
            elif g[j] < threshold and dataset.ix[testIdX[j], "variety"]
== dataVariety[1]:
                TN += 1
                labels[testIdX[j]] = 0
        TPR.append(TP / (TP + FN))
        FPR.append(FP / (FP + TN))


    plt.xlabel('FA')
    plt.ylabel('PD')
    plt.plot(FPR, TPR)
    plt.show()
```

```
    auc = 0
    for i in range(len(TPR) - 1):
        auc += (TPR[i] + TPR[i + 1]) * (FPR[i + 1] - FPR[i]) / 2
    print("AUC: %f" %(auc))
```

III. Linear classifier (by Perceptron Algorithm)

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

def sign(z):
    if z > 0:
        return 1
    else:
        return -1

def dot(w, y, x):
    for i in range(len(w)):
        w[i] = w[i] + y[0] * x[i]
    return w

# # load ionos dataset from csv
# dataset = pd.read_csv('./ionosphere.csv', names=['c', 'd', 'e', 'f',
'g',
#                                                 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
#                                                       'u', 'v', 'w',
'x', 'y', 'z', 'ab', 'bc', 'cd', 'de', 'ef',
#                                                       'fg', 'gh', 'hi',
'variety'], skiprows=0)


# load vertebral dataset from csv
dataset = pd.read_csv('./vertebral.csv', names=['a', 'b', 'c', 'd', 'e',
'f', 'variety'], skiprows=0)

dataVariety = dataset['variety'].unique()

# randomize ionos data
randomData = dataset.sample(frac=1)
datasize = randomData.shape[0]

KFold = 10
success = 0
confusion_mat = np.zeros([len(dataVariety), len(dataVariety)])
g = []

for i in range(KFold):
    # split the train data and test data
    trainData = randomData[:int(datasize * i / KFold) + int(datasize * (i
+ 1) / KFold):]
```

```
    testData = randomData[int(datasize * i / KFold):int(datasize * (i +
1) / KFold)]

    w = np.zeros((trainData.shape[1]))

    error = 1
    iterator = 0

    while error != 0 and iterator < 10:
        error = 0
        for index, row in trainData.iterrows():
            arr = np.array(row[:trainData.shape[1] - 1])
            x = np.concatenate((np.array([1.]), arr))
            if row['variety'] == dataVariety[0]:
                y = np.array([1.])
            elif row['variety'] == dataVariety[1]:
                y = np.array([-1.])
            if sign(np.dot(w, x)) != y:
                w = dot(w, y, x)
                error += 1
                iterator += 1


    for index, row in testData.iterrows():
        arr = np.array(row[:trainData.shape[1] - 1])
        x = np.concatenate((np.array([1.]), arr))
        if row['variety'] == dataVariety[0]:
            real = 0
            y = np.array([1.])
        elif row['variety'] == dataVariety[1]:
            real = 1
            y = np.array([-1.])

        g.append(np.dot(w, x))

        predict = sign(np.dot(w, x))
        if predict == y:
            success += 1

        if predict == 1:
            confusion_mat[0][real] += 1
        elif predict == -1:
            confusion_mat[1][real] += 1

accuracy = success / dataset.shape[0]
print("accuracy: %f" %(accuracy))

# draw confusion matrix
fig, axis = plt.subplots(figsize=(5, 5))
axis.matshow(confusion_mat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confusion_mat.shape[0]):
    for j in range(confusion_mat.shape[1]):
        axis.text(x=j, y=i, s=confusion_mat[i,j], va='center',
ha='center')

tick_marks = np.arange(len(dataVariety))
```

```
plt.xticks(tick_marks, dataVariety, rotation=0)
plt.yticks(tick_marks, dataVariety)

plt.xlabel('predicted label')
plt.ylabel('true label')
plt.title('Confusion matrix')
plt.show()


if (len(dataVariety) == 2):
    # draw ROC curve
    g_sort = sorted(g, reverse=True)
    TPR = []
    FPR = []
    labels = np.zeros([dataset.shape[0]])
    predPosProb = np.zeros([dataset.shape[0]])

    for i in range(len(g_sort)):
        TP = 0
        FP = 0
        FN = 0
        TN = 0
        for index, row in dataset.iterrows():
            arr = np.array(row[:dataset.shape[1] - 1])
            x = np.concatenate((np.array([1.]), arr))
            g2 = np.dot(w, x)
            if g2 >= g_sort[i] and row['variety'] == dataVariety[0]:
                TP += 1
                predPosProb[index] += 1
                labels[index] = 1
            elif g2 >= g_sort[i] and row['variety'] == dataVariety[1]:
                FP += 1
                predPosProb[index] += 1
                labels[index] = 0
            elif g2 < g_sort[i] and row['variety'] == dataVariety[0]:
                FN += 1
                labels[index] = 1
            else:
                TN += 1
                labels[index] = 0
        TPR.append(TP / (TP + FN))
        FPR.append(FP / (FP + TN))

    plt.xlabel('FA')
    plt.ylabel('PD')
    plt.plot(FPR, TPR)
    plt.show()

    auc = 0
    for i in range(len(TPR) - 1):
        auc += (TPR[i] + TPR[i + 1]) * (FPR[i + 1] - FPR[i]) / 2
    print("AUC: %f" %(auc))
```