과목명	데이터마이닝
담당교수	황두성 교수님
과제명	중간고사 대체 과제
학과	소프트웨어학과
학번	32151671
이름	박민혁
제출일자	2021 – 04 - 29

Problem 1.

Implement the following algorithm. Given a dataset $S = \{(x_i, y_i) \mid i = 1, 2, \ldots, n$ and $y_i = -1$ or $+ 1\}$ and learning rate $\eta \in R$:

procedure train_model (S, η, w, b)

$w = 0; b = 0;$

$R = \max_{1 \leq i \leq n} \| x_i \|_2$

repeat

for $i = 1$ to l do

if $y_i(< w \cdot x_i > +b) \leq 0$ then

$w = w + \eta y_i x_i$

$b = b + \eta y_i R^2$

end if

end for

until $y_j(< w \cdot x_j > +b) > 0, \forall j$

Given parameters w and b, and unknown instance x.

Procedure predict(w, b, x)

If $(< w \cdot x > +b) > 0$ then

    return +1

else

    return -1

end if

1.1) Learn a model for prob1_data.tra and evaluate it in terms of accuracy.

```
import pandas as pd
import numpy as np



def predict(w, b, x):
    if (np.dot(w, x) + b) > 0:
        return 1
    else:
        return -1



def train_model(train_data, l):
    start = 10
    data = np.array(train_data)

    x, y = data[:, 1:], data[:, 0]
    b = 0
    w = [0] * len(x[0])
    R = 0

    for i in range(data.shape[0]):
        tmp = np.sqrt(x[i][0] ** 2 + x[i][1] ** 2)
        if R < tmp:
            R = tmp

    for i in range(start):
        for j in range(len(data)):
            if y[j] * (np.dot(w, x[j].T) + b) <= 0:
                w = w + l * y[j] * x[j]
                b = b + l * y[j] * (R ** 2)

    return w, b
```

```
def predict_accuracy(w, b, x, y):

    pred = []
    for i in range(len(x)):
        pred.append(predict(w, b, x[i]))

    accuracy = 0
    for i in range(len(x)):
        if y[i] == pred[i]: accuracy += 1

    return accuracy / len(x)


train_data = pd.read_csv('./dm_data2/prob1_data.tra')
train_data = np.array(train_data)

x, y = train_data[:, 1:], train_data[:, 0]
w, b = train_model(train_data, 0.01)

print('train accuracy : {0}'.format(predict_accuracy(w, b, x, y)))
```

Problem01-1

```
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem01-1.py
train accuracy : 0.962

Process finished with exit code 0
```

1.2) Test the model for prob1_data.tes in terms of accuracy.

```
test_data = pd.read_csv('./dm_data2/prob1_data.tes')
test_data = np.array(test_data)

x, y = test_data[:, 1:], test_data[:, 0]
w, b = train_model(train_data, 0.01)

print('Test accuracy : {0}'.format(predict_accuracy(w, b, x, y)))
```

Problem01-1

```
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem01-1.py
Test accuracy : 0.97

Process finished with exit code 0
```

Problem 2.

Perceptrons can simulate any logical circuits such as OR, AND, NOT, NAND, and XOR. This means that we can design any combinatorial circuit by integrating perceptrons that act as logical circuits.

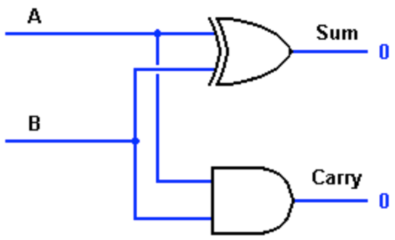


Figure 1: An example of half adder

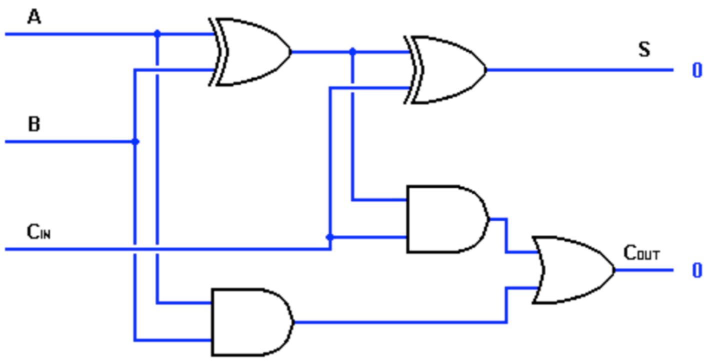


Figure 2: An example of full adder

2.1) Write the truth table of a half adder(Figure 1)

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2.2) Implement and test a half adder by combining the perceptrons that simulism logical circuits.

```
import numpy as np

def AND(x1, x2):
    x = np.array([x1, x2, 1])
    w = np.array([0.5, 0.5, -0.7])

    temp = np.dot(x, w.T)

    if temp <= 0:
        return 0
    else:
        return 1


def OR(x1, x2):
    x = np.array([x1, x2, 1])
    w = np.array([0.5, 0.5, -0.2])

    temp = np.dot(x, w.T)

    if temp <= 0:
        return 0
    else:
        return 1


def NAND(x1, x2):
    x = np.array([x1, x2, 1])
    w = np.array([-0.5, -0.5, 0.7])

    temp = np.dot(x, w.T)

    if temp <= 0:
        return 0
    else:
        return 1
```

```
def XOR(x1, x2):
    nand_gate = NAND(x1, x2)
    or_gate = OR(x1, x2)
    xor_gate = AND(nand_gate, or_gate)

    return xor_gate


for i in [0, 1]:
    for j in [0, 1]:
        print('(A : {0}, B : {1}) (Sum : {2}, Carry : {3})'.format(i, j, XOR(i, j), AND(i, j)))
```

Problem02-2

```
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem02-2.py
(A : 0, B : 0) (Sum : 0, Carry : 0)
(A : 0, B : 1) (Sum : 1, Carry : 0)
(A : 1, B : 0) (Sum : 1, Carry : 0)
(A : 1, B : 1) (Sum : 0, Carry : 1)

Process finished with exit code 0
```

2.3) Write the truth table of a full adder(Figure 2)

A	B	Carry in	Carry out	Sum
0	0	0	0	0
0	1	0	0	1
1	0	0	1	0
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

2.4) Implement and test a full adder by combining the perceptrons that simulism logical circuits.

```
for i in [0, 1]:
    for j in [0, 1]:
        for k in [0, 1]:
            print('(A : {0}, B : {1}, Carry in : {2}) (Carry out : {3}, Sum : {4})'.format(j, k, i,

   OR(AND(XOR(j, k), i),

        AND(j, k)),

        XOR(XOR(j, k), i)))
```

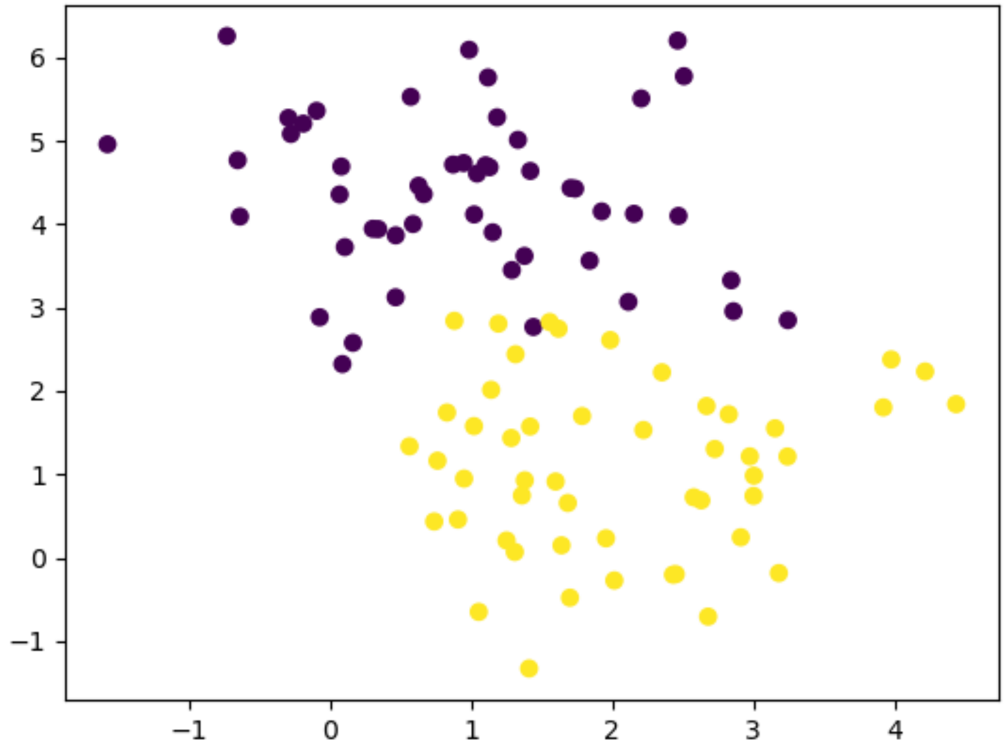
```
Problem02-4
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/dm_data2/Problem02-4.py
(A : 0, B : 0, Carry in : 0) (Carry out : 0, Sum : 0)
(A : 0, B : 1, Carry in : 0) (Carry out : 0, Sum : 1)
(A : 1, B : 0, Carry in : 0) (Carry out : 0, Sum : 1)
(A : 1, B : 1, Carry in : 0) (Carry out : 1, Sum : 0)
(A : 0, B : 0, Carry in : 1) (Carry out : 0, Sum : 1)
(A : 0, B : 1, Carry in : 1) (Carry out : 1, Sum : 0)
(A : 1, B : 0, Carry in : 1) (Carry out : 1, Sum : 0)
(A : 1, B : 1, Carry in : 1) (Carry out : 1, Sum : 1)
```

Problem 3.

A 2-class classification is given in Figure 3 from the training set(prob3_data.tra)

3.1) Read and plot the data and count the number of data per class.

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

data = pd.read_csv('./dm_data2/prob3_data.tra')
scatter = plt.scatter(data['X1'], data['X2'], c=data['# cls'])
print(data[['X1', 'X2']].groupby(data['# cls']).count())
plt.show()
```




```
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem03-1.py
       X1  X2
# cls
0      50  50
1      50  50

Process finished with exit code 0
```

3.2 Define the classification problem based on mathematical notation.

$P = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathrm{R}^d, y_i \in \{1,2,\cdots,c\}\}$

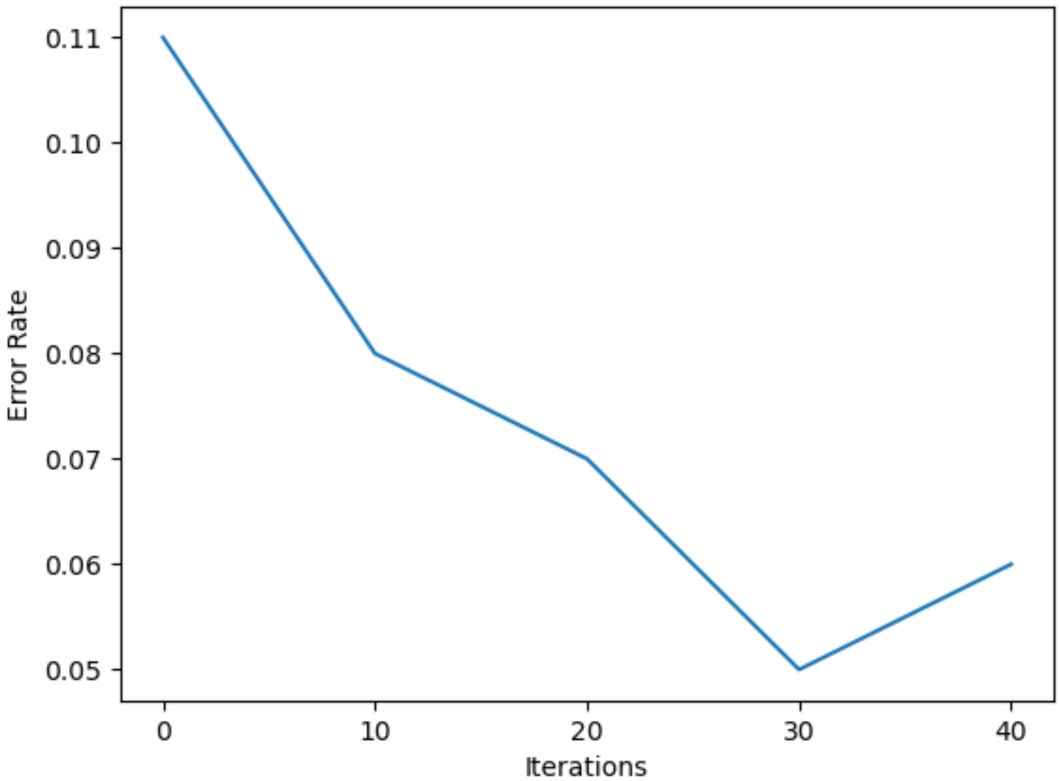
Let $X \equiv \mathrm{R}^d$ and $Y \equiv \{1,2,\cdots,c\}$

$y_i \in \{1,2\}$, $y_i \in \{0,1\}$, $y_i \in \{-1,+1\}$

3.3 Model a linear classifier using the perceptron algorithm and plot the error graph per 10 iterations

```
def perceptron(train_data, l):
    start = 50
    data = np.array(train_data)
    x, y = data[:, 1:], data[:, 0]
    w = [0] * len(x[0])
    b = 0
    error = []

    for i in range(start):
        for j in range(len(data)):
            update = l * (y[j] - predict(w, b, x[j]))
            w += update * x[j]
            b += update

        if i % 10 == 0:
            error.append(1 - predict_accuracy(w, b, x, y))

    return w, b, error

train_data = pd.read_csv('./dm_data2/prob3_data.tra')
train_data = np.array(train_data)

x, y = train_data[:, 1:], train_data[:, 0]
w, b, error = perceptron(train_data, 0.1)
```

```
X = np.arange(0, 50, 10)

plt.plot(X, error)
plt.xticks(np.arange(0, 50, 10))
plt.xlabel('Iterations')
plt.ylabel('Error Rate')
plt.show()
```



3.4) Explain about your architecture and write it with weights and bias.

```
Learning Rate : 0.1

Start : 50
```

```
Problem03-3
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem03-3.py
w : [ 0.2312822 -0.8768658], b : 2.600000000000001

Process finished with exit code 0
```

3.5) Test your model for both the training and test sets(prob3_data.tes)

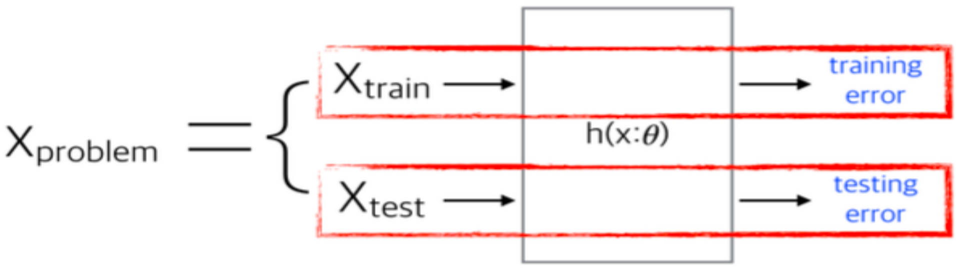
```
test_data = pd.read_csv('./dm_data2/prob3_data.tes')
test_data = np.array(test_data)

train_data = pd.read_csv('./dm_data2/prob3_data.tra')
train_data = np.array(train_data)

test_x, test_y = test_data[:, 1:], test_data[:, 0]
train_x, train_y = train_data[:, 1:], train_data[:, 0]

test_w, test_b, test_error = perceptron(test_data, 0.1)
train_w, train_b, train_error = perceptron(train_data, 0.1)

print('Train Accuracy : {0}'.format(predict_accuracy(train_w, train_b, train_x, train_y)))
print('Test Accuracy : {0}'.format(predict_accuracy(test_w, test_b, test_x, test_y)))
```

```
Problem03-5
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem03-5.py
Train Accuracy : 0.92
Test Accuracy : 0.96

Process finished with exit code 0
```

Problem 4.

A regression problem is given in Figure 4(reg4_data.tra).

4.1) Define the regression problem based on mathematical notation.



Training data: $\mathbf{x}_i \in R^d$ for i = 1,2,··· ,n_1

Testing data: $\mathbf{x}_i \in R^d$ for i = 1,2,··· ,n_2

Learning goal: Find a set of groups or classes such that *a predefined measure* is satisfied.

$$h_\theta(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = \sum_{i=0}^{2} \theta_i x_i$$

$$(1\ x_1\ x_2\ ) \times \begin{matrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{matrix} = x'\theta^t$$

$$argmin\, J(\theta) = \frac{1}{2} \sum_{i=0}^{46} (h_\theta(x_i) - t_i)^2$$

$$where\ t_i \cong h_\theta(x_i)$$

4.2) Describe the least square function J(θ) for a linear regression.

Least Mean Squre step 1: set θ an initial value step 2: change θ to make J(θ) smaller $\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ step 3: repeat step 2 until we converge to a minimum value of θ

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} \sum_{i=1}^{n} (h_\theta(x_i) - y_i)^2$$

$$= \left[\sum_{i=1}^{n} (h_\theta(x_i) - y_i)\right] \frac{\partial}{\partial \theta_j} (h_\theta(x_i) - y_i)$$

$$= \sum_{i=1}^{n} (h_\theta(x_i) - y_i) \frac{\partial}{\partial \theta_j} \left(\sum_{j=0}^{d} \theta_j \cdot x_{ij} - y_i\right)$$

$$= \underbrace{\sum_{i=1}^{n} (h_\theta(x_i) - y_i) x_{ij}}_{Error_i}$$

$f(x,y) = x^2 + xy + y^2$

$\frac{\partial}{\partial x} f(x,y) = 2x + 1 = 0$

$\frac{\partial}{\partial y} f(x,y) = y + 2y = 0$

$\therefore x = 0, \quad y = 0$

$f(0,0) = 0$

$f(x) = 3x^2 - x + 5$

$\frac{d}{dx} f(x) = 6x - 1$

$\therefore x = \frac{1}{6}$

$f(0,0) = \frac{59}{12}$

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$= \theta_j - \alpha \sum_{i=1}^{n} \left(h_\theta(x_i) - y_i\right) x_{ij}$$

$$= \theta_j + \alpha \cdot \sum_{i=1}^{n} \left(y_i - h_\theta(x_i)\right) x_{ij}$$

$$\theta_j = \theta_j + \alpha \left(y_i - h_\theta(x_i)\right) x_{ij}$$

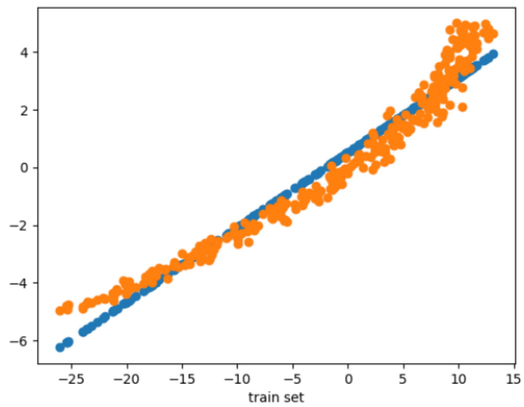
4.3) Based on the least square algorithm, model a linear regressor.

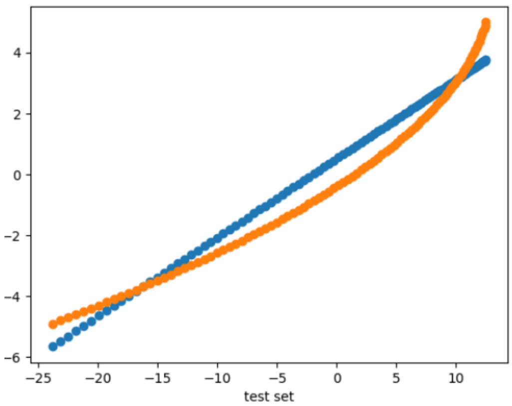
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

train_data = pd.read_csv('./dm_data2/reg4_data.tra')
test_data = pd.read_csv('./dm_data2/reg4_data.tes')

train_data = np.array(train_data)
test_data = np.array(test_data)

train_x, train_y = train_data[:, 0].reshape(-1, 1), train_data[:, 1].reshape(-1, 1)
test_x, test_y = test_data[:, 0].reshape(-1, 1), test_data[:, 1].reshape(-1, 1)

model = LinearRegression()
model.fit(train_x, train_y)
```

4.4) Based on the least square error, evaluate your model for both the training and test sets(reg4_data.tes).

```
print("Train : {0}".format(model.score(train_x, train_y)))
print("Test : {0}".format(model.score(test_x, test_y)))
```

```
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem04-3.py
Train : 0.9477734820436993
Test : 0.9491537839454265

Process finished with exit code 0
```





Problem 5.

A 2-class classification problem is plotted in Figure 5 for the training set(prob5_moon.tra). Answer the following questions for the test set(prob5_moon.tes).

5.1) Model and evaluate a logistic classifier for the training and test sets

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

train_data = pd.read_csv('./dm_data2/prob5_moons.tra')
test_data = pd.read_csv('./dm_data2/prob5_moons.tes')

train_data = np.array(train_data)
test_data = np.array(test_data)

train_x, train_y = train_data[:, 1:], train_data[:, 0].reshape(-1, 1)
test_x, test_y = test_data[:, 1:], test_data[:, 0].reshape(-1, 1)

model = LogisticRegression()
model.fit(train_x, train_y)

print('Train Accuracy : {0}'.format(model.score(train_x, train_y)))
print('Test Accuracy : {0}'.format(model.score(test_x, test_y)))
```

```
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem05-1.py
Train Accuracy : 0.8817635270541082
Test Accuracy : 0.8888888888888888
```

5.2) Model and evaluate a k-nearest neighbor for the training and test sets(k = 1, 3, 5)

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

train_data = pd.read_csv('./dm_data2/prob5_moons.tra')
test_data = pd.read_csv('./dm_data2/prob5_moons.tes')

train_data = np.array(train_data)
test_data = np.array(test_data)

train_x, train_y = train_data[:, 1:], train_data[:, 0].reshape(-1, 1)
test_x, test_y = test_data[:, 1:], test_data[:, 0].reshape(-1, 1)

for i in range(1, 6, 2):
    model = KNeighborsClassifier(n_neighbors=i)  # 1, 3, 5
    model.fit(train_x, train_y)
    print('K = {0}'.format(i))
    print('Train Accuracy : {0}'.format(model.score(train_x, train_y)))
    print('Test Accuracy : {0}'.format(model.score(test_x, test_y)))
```

```
K = 1
Train Accuracy : 1.0
Test Accuracy : 1.0
K = 3
Train Accuracy : 1.0
Test Accuracy : 1.0
K = 5
Train Accuracy : 1.0
Test Accuracy : 1.0
```

5.3) Model and evaluate a linear discriminant model for the training and test sets. Use LinearDiscriminantAnalysis from sklearn

```
import numpy as np
import pandas as pd
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

train_data = pd.read_csv('./dm_data2/prob5_moons.tra')
test_data = pd.read_csv('./dm_data2/prob5_moons.tes')

train_data = np.array(train_data)
test_data = np.array(test_data)

train_x, train_y = train_data[:, 1:], train_data[:, 0].reshape(-1, 1)
test_x, test_y = test_data[:, 1:], test_data[:, 0].reshape(-1, 1)

model = LinearDiscriminantAnalysis()
model.fit(train_x, train_y)

print('Train Accuracy : {0}'.format(model.score(train_x, train_y)))
print('Test Accuracy : {0}'.format(model.score(test_x, test_y)))
```

```
Train Accuracy : 0.8797595190380761
Test Accuracy : 0.8888888888888888

Process finished with exit code 0
```

5.4) Model and evaluate a quadratic discriminant model for the training and t est sets. Use QuadraticDiscriminantAnalysis from sklearn.

```
import numpy as np
import pandas as pd
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

train_data = pd.read_csv('./dm_data2/prob5_moons.tra')
test_data = pd.read_csv('./dm_data2/prob5_moons.tes')

train_data = np.array(train_data)
test_data = np.array(test_data)

train_x, train_y = train_data[:, 1:], train_data[:, 0].reshape(-1, 1)
test_x, test_y = test_data[:, 1:], test_data[:, 0].reshape(-1, 1)

model = QuadraticDiscriminantAnalysis()
model.fit(train_x, train_y)

print('Train Accuracy : {0}'.format(model.score(train_x, train_y)))
print('Test Accuracy : {0}'.format(model.score(test_x, test_y)))
```

```
Train Accuracy : 0.875751503006012
Test Accuracy : 0.8888888888888888

Process finished with exit code 0
```

5.5) Compare all the learned models in terms of training and test error, model architecture, etc.

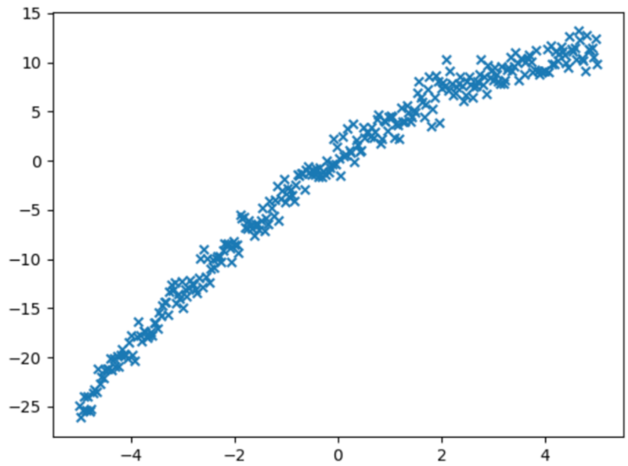


Figure 4: A toy example for regression

Models of logistic regression differ from linear models in the relationship between dependent and independent variables. The first difference is that when applied to binomial data, the result of the dependent variable y is bounded by range[0,1], and the second difference is that the distribution of the conditional probability (P(y|x)) follows a binomial distribution instead of a normal one.

K-nearest neighbor algorithm can be easily implemented by calculating the distance between the test data and all stored data, but requires large computations for large training sets.

The quadratic discriminant analysis assumes that the independent variable x is real and that the probability distribution is multivariate normal. The location and shape of the distribution of x can vary from class to class.

In linear discriminant analysis, only expected value vectors depend on the class and covariance matrices are estimated in common.

Problem 6.

Consider the 3-class categories in Figure 6 for the data(prob_bayes.tra and prob_bayes.tes). Assume that the underlying distributions are Gaussian.

6.1) Write a procedure to calculate the discriminate function for a given Gaussian distribution and prior probability.

$$P = \{(x_i, y_i) \mid x_i \in \mathbb{R}^d,\ y_i \in V\} \text{ and } V = \{0, 1\}$$

$$y \sim \text{Bernoulli}(\phi)$$

$$x \mid y = 0 \sim N(\mu_0, \Sigma)$$

$$x \mid y = 1 \sim N(\mu_1, \Sigma)$$

$$P(y) = \phi^{y}(1-\phi)^{1-y}$$

$$\Rightarrow P(x \mid \mu_0, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_0)^t \Sigma^{-1} (x-\mu_0)\right)$$

$$\Rightarrow P(x \mid \mu_1, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_1)^t \Sigma^{-1} (x-\mu_1)\right)$$

$$P(x \mid y) = \frac{P(x, y)}{P(y)}$$

$$\begin{aligned} l(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} P(x_i, y_i, \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{n} P(x_i \mid y_i ; \phi, \mu_0, \mu_1, \Sigma)\, P(y_i ; \phi) \end{aligned}$$

$$\phi = \frac{1}{n}\sum_{i=0}^{n} 1(y_i = 1), \quad \mu_0 = \frac{\sum_{i=0}^{n} 1(y_i = 0)\, x_i}{\sum_{i=0}^{n} 1(y_i = 0)}$$

$$\mu_1 = \frac{\sum_{i=0}^{n} 1(y_i = 1)\, x_i}{\sum_{i=0}^{n} 1(y_i = 1)} \qquad \Sigma = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu_{y_i})(x_i - \mu_{y_i})^t$$

$$p(x|\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^t\Sigma^{-1}(x-\mu)\right)$$

$$E(x) = \int_x x\,p(x|\mu,\Sigma)\,dx = \mu$$

$$\mathrm{cov}(x) = E\left[(x-\mu)(x-\mu)^t\right] = \Sigma$$

6.2) Estimate mean and variance for each class.

```
import numpy as np
import pandas as pd

train_data = pd.read_csv('./dm_data2/prob_bayes.tra')
test_data = pd.read_csv('./dm_data2/prob_bayes.tes')

train_data['X1X2'] = train_data['X1'] * train_data['X2']

grouped = train_data.groupby(train_data['# cls'])
mean_x1, var_x1 = grouped['X1'].mean(), grouped['X1'].var()
mean_x2, var_x2 = grouped['X2'].mean(), grouped['X2'].var()
mean_x1x2, var_x1x2 = grouped['X1X2'].mean(), grouped['X1X2'].var()

sigma = np.array([np.eye(2)] * 3)
mean = np.array([np.zeros(2)] * 3)

for i in range(len(mean_x1)):
    sigma[i, 0, 0] = var_x1[i]
    sigma[i, -1, -1] = var_x2[i]
    sigma[i, 0, 1] = mean_x1x2[i] - (mean_x1[i] * mean_x2[i])
    sigma[i, 1, 0] = mean_x1x2[i] - (mean_x1[i] * mean_x2[i])

    mean[i, 0] = mean_x1[i]
    mean[i, -1] = mean_x2[i]

for i in range(len(mean)):
    print('class {0} : (x1, x2) mean = ({1}, {2})'.format(i, mean[i][0], mean[i][1]))

for i in range(len(sigma)):
    print('class {0} : var = {1}'.format(i, sigma[i]))
```

```
class 0 : (x1, x2) mean = (0.9529215300000001, 4.383678520000001)
class 1 : (x1, x2) mean = (1.9343147599999995, 0.8241768500000002)
class 2 : (x1, x2) mean = (-1.6216559100000003, 2.84897229)
class 0 : var = [[ 0.48765571 -0.0167833 ]
 [-0.0167833   0.49934725]]
class 1 : var = [[ 0.46646269 -0.036757  ]
 [-0.036757    0.4138193 ]]
class 2 : var = [[0.53823626 0.04241193]
 [0.04241193 0.51406914]]
```

6.3) Determine the train and test accuracy and check out how many data eac h Gaussian classifier misses.

```
import numpy as np
import pandas as pd
import scipy.stats as sp


def cal_mean_var(train_data):
    train_data['X1X2'] = train_data['X1'] * train_data['X2']
    grouped = train_data.groupby(train_data['# cls'])
    mean_x1, var_x1 = grouped['X1'].mean(), grouped['X1'].var()
    mean_x2, var_x2 = grouped['X2'].mean(), grouped['X2'].var()

    sigma = np.eye(3)
    mean = np.array([np.zeros(2)] * 3)

    for i in range(len(mean_x1)):
        mean[i, 0], mean[i, -1] = mean_x1[i], mean_x2[i]

    return mean, sigma


def predict(data_x, data_y, mean, sigma):
    pred_zero = (sp.multivariate_normal.pdf(data_x, mean=mean[0], cov=sigma[0, :2], allow_
singular=True))
    pred_first = (sp.multivariate_normal.pdf(data_x, mean=mean[1], cov=sigma[1, :2], allow_s
ingular=True))
    pred_second = (sp.multivariate_normal.pdf(data_x, mean=mean[2], cov=sigma[2, :2], allo
w_singular=True))
```

```python
pred_y = []

    for i, j, k in zip(pred_zero, pred_first, pred_second):
        if i == max(i, j, k):
            pred_y.append(0)
        elif j == max(i, j, k):
            pred_y.append(1)
        elif k == max(i, j, k):
            pred_y.append(2)

    accuracy = 0

    for i in range(len(pred_y)):
        if pred_y[i] != data_y[i]:
            accuracy += 1

    return accuracy


train_data = pd.read_csv('./dm_data2/prob_bayes.tra')
test_data = pd.read_csv('./dm_data2/prob_bayes.tes')

train_x, train_y = train_data[['X1', 'X2']], train_data['# cls']
test_x, test_y = test_data[['X1', 'X2']], test_data['# cls']

mean, sigma = cal_mean_var(train_data)

print('Train misses count : {0}'.format(predict(train_x, train_y, mean, sigma)))
print('Test misses count : {0}'.format(predict(test_x, test_y, mean, sigma)))
```

```
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem06-3.py
Train misses count : 200
Test misses count : 40

Process finished with exit code 0
```

6.4) When the variance is I (identity matrix), evaluate the train and test accuracy.

```
import numpy as np
import pandas as pd
import scipy.stats as sp

def cal_mean_var(train_data):
    train_data['X1X2'] = train_data['X1'] * train_data['X2']
    grouped = train_data.groupby(train_data['# cls'])
    mean_x1, var_x1 = grouped['X1'].mean(), grouped['X1'].var()
    mean_x2, var_x2 = grouped['X2'].mean(), grouped['X2'].var()

    sigma = np.eye(2)
    mean = np.array([np.zeros(2)] * 3)

    for i in range(len(mean_x1)):
        mean[i, 0], mean[i, -1] = mean_x1[i], mean_x2[i]

    return mean, sigma

def predict(data_x, data_y, mean, sigma):

    pred_zero = (sp.multivariate_normal.pdf(data_x, mean=mean[0], cov=sigma))
    pred_first = (sp.multivariate_normal.pdf(data_x, mean=mean[1], cov=sigma))
    pred_second = (sp.multivariate_normal.pdf(data_x, mean=mean[2], cov=sigma))

    pred_y = []

    for i, j, k in zip(pred_zero, pred_first, pred_second):
        if i == max(i, j, k):
            pred_y.append(0)
        elif j == max(i, j, k):
            pred_y.append(1)
        elif k == max(i, j, k):
            pred_y.append(2)

    accuracy = 0
```

```
for i in range(len(pred_y)):
        if pred_y[i] != data_y[i]:
            accuracy += 1

    return accuracy / len(pred_y)

train_data = pd.read_csv('./dm_data2/prob_bayes.tra')
test_data = pd.read_csv('./dm_data2/prob_bayes.tes')

train_x, train_y = train_data[['X1', 'X2']], train_data['# cls']
test_x, test_y = test_data[['X1', 'X2']], test_data['# cls']

mean, _ = cal_mean_var(train_data)
sigma = np.eye(2)

print('Train Accuracy : {0}'.format(predict(train_x, train_y, mean, sigma)))
print('Test Accuracy : {0}'.format(predict(test_x, test_y, mean, sigma)))
```

```
/Users/m1nh/.conda/envs/Desktop/bin/python /Users/m1nh/Desktop/minh/DKU/데이터마이닝/Problem06-4.py
Train Accuracy : 0.01
Test Accuracy : 0.05

Process finished with exit code 0
```

6.5) Compare your results for two models in terms of accuracy and the number of misclassified data.

If p(x|y) is multivariate Gaussian with the same Σ, then p(y|x) follows a logistic function.
But p(y|x) being a logistic function does not imply p(x|y) is multivariate gaussian.
This shows that GDA makes stronger modeling assumptions about the data than does logistic regression.
Specifically, when p(x|y) is indeed Gaussian, then GDA is asymptotically efficient.
By making significantly weaker assumptions, logistic regression is more robust and less sensitive to incorrect modeling assumptions.
Choose logistic regression when the data is non-Gaussian, or apply GDA.

Problem 7.

Given a dataset $D = \{(x_i, y_i)|i = 1, . . . , n\}$, consider the fitted values that result from perform- ing linear regression without a bias. In this setting, the x_i's fitted value takes the form

$$\hat{y} = x_i \hat{\beta},$$

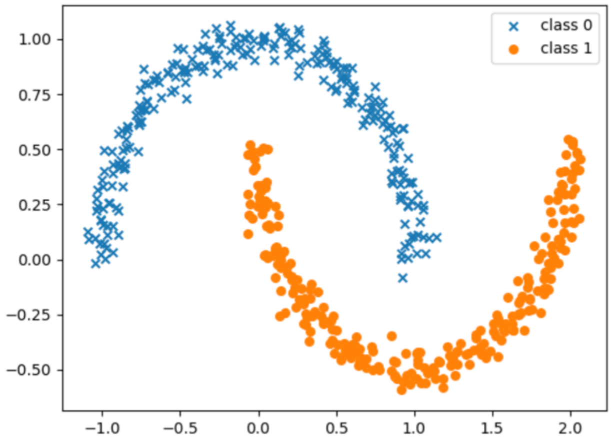


Figure 5: A toy example for 2-class classification

where

$$\hat{\beta} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}.$$

When we can write

$$\hat{y}_i = \sum_{j=1}^{n} \alpha_j y_j$$

What is α_j?

$\hat{y} = x_i \hat{B}_i$

Where

$$\hat{B} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2} \quad \Rightarrow \hat{y}_i = \sum_{i=j}^{n} x_j y_j$$

$\therefore x_i \hat{B}$

$\Rightarrow \hat{y}_i = x_i \hat{B}$

$$\Rightarrow \frac{x_i (x_1 y_1 + x_2 y_2 + \cdots + x_n y_n)}{x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

$$\therefore x_j = \frac{x_i x_j}{\sum_{i=1}^{n} x_i^2}$$

Problem 8.

The probabilistic assumption follows Bernoulli probability distribution $p(y|\mathbf{x} : \theta) = h_\theta(\mathbf{x})^y(1 - h_\theta(\mathbf{x}))^{1-y}$ where $h_\theta(\mathbf{x}) = g(\theta^t\mathbf{x})$ and $g(z) = \frac{1}{1+e^{-z}}$. There exist n training examples which were generated independently, so we can apply the likelihood of θ. It is easy to maximize the log likelihood.

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{n} h_\theta(\mathbf{x}_i)^{y_i}(1 - h_\theta(\mathbf{x}_i))^{1-y_i} \\ \ell(\theta) &= \log L(\theta) \\ &= \prod_{i=1}^{n} \left\{ y_i \log h_\theta(\mathbf{x}_i) + (1 - y_i)\log(1 - h_\theta(\mathbf{x}_i)) \right\} \end{aligned}$$

Derive $\frac{\partial}{\partial\theta_j}\ell(\theta)$ for a single example $(\mathbf{x}, y)$.

$$\ell(\theta) = y \log h_\theta(\mathbf{x}) + (1 - y)\log(1 - h_\theta(\mathbf{x}))$$

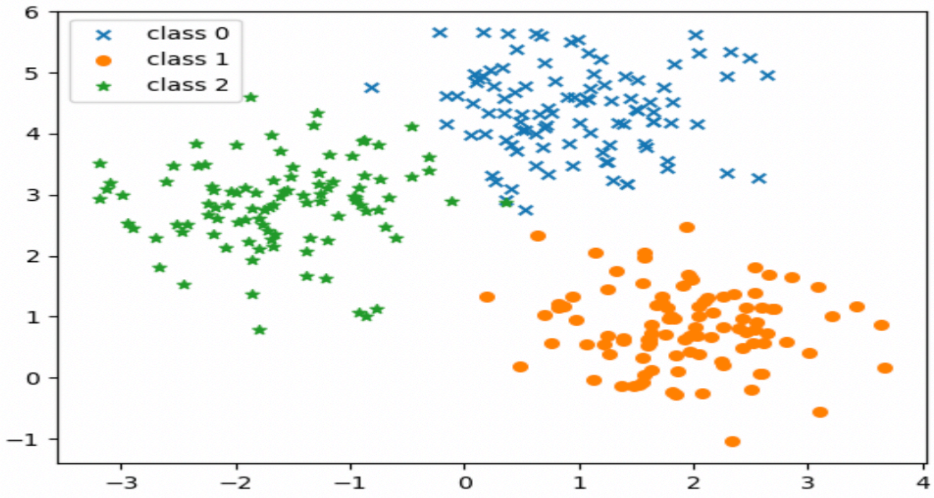


Figure 6: 3-class classification

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = \frac{\partial}{\partial \theta_j} \left\{ y \log h_\theta(x) + (1-y) \log(1-h_\theta(x)) \right\}$$

$$\Rightarrow y \frac{1}{h_\theta(x)} \frac{\partial}{\partial \theta} h_\theta(x) + (1-y) \frac{1}{1-h_\theta(x)} \frac{\partial}{\partial \theta} (1-h_\theta(x))$$

$$\Rightarrow y \frac{1}{g(\theta^t x)} \frac{\partial}{\partial \theta} g(\theta^t x) + (1-y) \frac{1}{1-g(\theta^t x)} \frac{\partial}{\partial \theta} (1-g(\theta^t x))$$

$$\Rightarrow \left\{ y \frac{1}{g(\theta^t x)} - (1-y) \frac{1}{1-g(\theta^t x)} \right\} \frac{\partial}{\partial \theta_j} g(\theta_t x)$$

$$\Rightarrow \left\{ y \frac{1}{g(\theta^t x)} - (1-y) \frac{1}{1-g(\theta^t x)} \right\} \frac{\partial}{\partial \theta_j} g(\theta_t x)$$

$$\Rightarrow \left\{ y \frac{1}{g(\theta^t x)} - (1-y) \frac{1}{1-g(\theta^t x)} \right\} g(g(\theta^t x))(1-g(\theta^t x)) \frac{\partial}{\partial \theta_j} \theta_t x$$

$$\Rightarrow \left[(y(1-g(\theta^t x)) - (1-y) g(\theta^t x) \right] x_j$$

$$\Rightarrow (y - h_\theta(x)) x_j$$

Problem 9.

The logistic function is as follows: $g(z) = \frac{1}{1+ e^{-z}}$

9.1) Find the first derivative of logistic function.

$$g(z) = \frac{1}{1+e^{-z}} = \text{Sigmoid}(z)$$

$$\Rightarrow \frac{d}{dz}\text{Sigmoid}(z) = \frac{d}{dz}(1+e^{-z})^{-1} = (-1)\frac{1}{(1+e^{-z})^2}\cdot\frac{d}{dz}(1+e^{-z})$$

$$\Rightarrow (-1)\cdot\frac{1}{(1+e^{-z})^2}(0+e^{-z})\frac{d}{dz}(-z) = (-1)\frac{1}{(1+e^{-z})^2}e^{-z}(-1)$$

$$\Rightarrow \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{(1+e^{-z})}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$$

$$\Rightarrow \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1-\frac{1}{1+e^{-z}}\right)$$

$$\Rightarrow \text{Sigmoid}(z)(1-\text{Sigmoid}(z))$$

9.2) Plot g(z).

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


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


x = np.arange(-10.0, 10.0, 0.1)
y = sigmoid(x)

plt.plot(x, y)
plt.ylim(-0.1, 1.1)
plt.show()
```