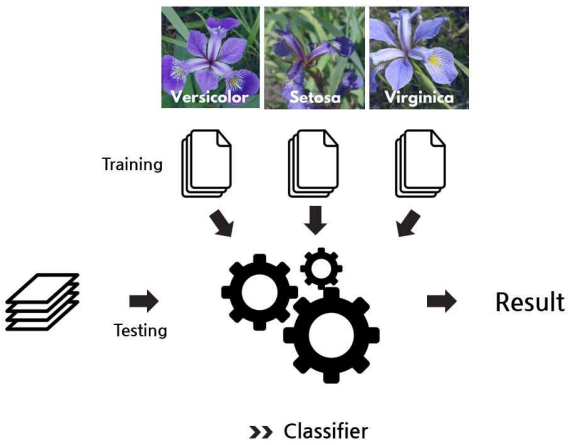
REPORT

Project #1

국민대학교	
학 부	컴퓨터공학부
과 목	비주얼 컴퓨팅 최신기술
담 당 교 수	한광수 교수님
제 출 일	2017년 3월 27일
학 번	20143390
이 름	김선영

· **Project 목표**

Training set과 Testing set으로 나눠진 Iris data set을 사용하여 간단한 인식기를 구현함으로 강의시간에 배운 인식기에 대한지식과 스스로 학습한 인식기에 대한 지식을 평가하는 프로젝트를 진행한다.



각 training 데이터는 3개의 class로 분류가 되어있고(Supervised Learning) 이 데이터를 사용하여 training을 거친 후 test set을 사용하여 구현한 인식기 성능을 시험한다. 1)은 4개의 feature을 가진 data를 사용하여 학습 후 성능을 시험하고 2)는 2개의 feature를 가진 data를 사용하여 학습 후 성능을 하고 2개의 차이점을 기술해본다.

1) The Iris data set is divided into a Training set and a Testing set. The training set has 40 samples for each class, and the testing set has 10 samples for each class.

각 class마다 주어진 40개의 samples을 이용하여 train한 후 10개의 test samples을 이용하여 구현한 분류기의 성능을 확인한다.

a. Assume that each class has a normal distribution, and use the training data to estimate the mean and covariance of each class.

```
mean class 1
[ 4.9975  3.4175  1.4425  0.2525]
covariance class 1
[[ 0.13512179  0.10748077  0.02805769  0.01244231]
 [ 0.10748077  0.15686538  0.01564744  0.00854487]
 [ 0.02805769  0.01564744  0.02507051  0.00642949]
 [ 0.01244231  0.00854487  0.00642949  0.01230128]]

mean class 2
[ 5.99    2.7775  4.31    1.3325]
covariance class 2
[[ 0.28041026  0.10079487  0.20035897  0.06648718]
 [ 0.10079487  0.11666026  0.09561538  0.04767308]
 [ 0.20035897  0.09561538  0.23528205  0.0804359 ]
 [ 0.06648718  0.04767308  0.0804359   0.04327564]]

mean class 3
[ 6.61    2.97    5.5575  2.03  ]
covariance class 3
[[ 0.44194872  0.0854359   0.34453846  0.04687179]
 [ 0.0854359   0.09497436  0.06638462  0.03297436]
 [ 0.34453846  0.06638462  0.35173718  0.04951282]
 [ 0.04687179  0.03297436  0.04951282  0.05548718]]
```

```python
# 평균 벡터를 구하는 함수
def meanVector(dataClass):
    mean = np.sum(dataClass, axis=0)[:4]/40
    return mean

# 공분산 벡터를 구하는 함수
def cov(classData, m):
    covV = np.zeros((4, 4)).astype(float)
    for i in range(0, 4):
        for j in range(0, 4):
            data1 = classData[:,i] - m[i]
            data2 = (classData[:, j] - m[j]).T
            covV[i][j] = np.sum(np.dot(data1, data2))/39
    return covV
```

각 class가 정규분포를 따를 때 확률 분포는 평균과 분산에 의해 결정된다. 평균은 모든 데이터의 합을 데이터의 개수로 나누어 구하고 분산은 $C = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \hat{\mu})(x_k - \hat{\mu})^t$ 과 같은 식을 이용하여 구할 수 있다. 공분산 행렬은 행렬을 이용하여 두 변수가 어떤 상관관계를 가지는지 표현한다. 따라서 공분산 행렬은 대각선 성분을 중심으로 대칭이 되게 된다.

각 class에 대한 mean과 covariance는 위의 그림의 데이터와 같다.

b. Determine the decision boundaries.

$$g_i(x) = x^t V_i x + v_i^t x + v_{i0}$$

where $$V_i = -\frac{1}{2}\Sigma_i^{-1}$$

$$v_i = \Sigma_i^{-1}\mu_i,$$

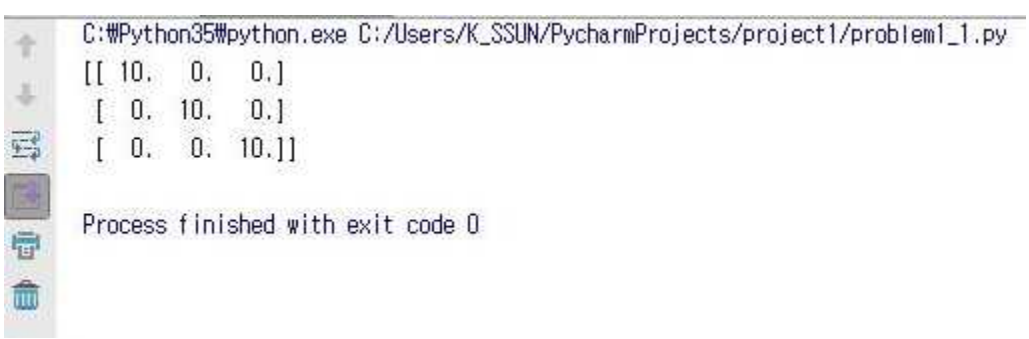
and $$v_{i0} = -\frac{1}{2}\mu_i^t \Sigma_i^{-1}\mu_i - \frac{1}{2}\ln|\Sigma_i| + \ln P(w_i)$$

각 class의 discriminant function값을 구하여 최대값을 갖는 class로 분류한다. 각 class가 각기 다른 covariance를 가질 때 discriminant function는 위의 공식을 이용하여 구할 수 있다.

```
# class를 결정하는 함수
def dicision(test):
    temp1 = calDicision(test, mVClass1, covV1)
    temp2 = calDicision(test, mVClass2, covV2)
    temp3 = calDicision(test, mVClass3, covV3)

    return compare(temp1, temp2, temp3)

# decision boundary 계산
def calDicision(test, m, c):
    com1 = np.dot(np.dot(test.T, -0.5 * lin.inv(c)), test)
    com2 = np.dot(np.dot(lin.inv(c), m).T, test)
    com3 = -0.5 * np.dot(np.dot(m, lin.inv(c)), m.T) - 0.5 * math.log(lin.det(c))

    return com1 + com2 + com3

# 3개 숫자 대소비교
def compare(temp1, temp2, temp3):
    return (3 if (temp2 < temp3) else 2) if (temp1 < temp2) else ( 3 if (temp1 < temp3) else 1)
```

위의 공식을 활용하여 discriminant function값을 비교하는 함수를 구현하고 test sample을 대입하였을 때 가장 큰 값을 갖는 class로 분류한다.

c. Classify the testing data set and construct the confusion matrix.

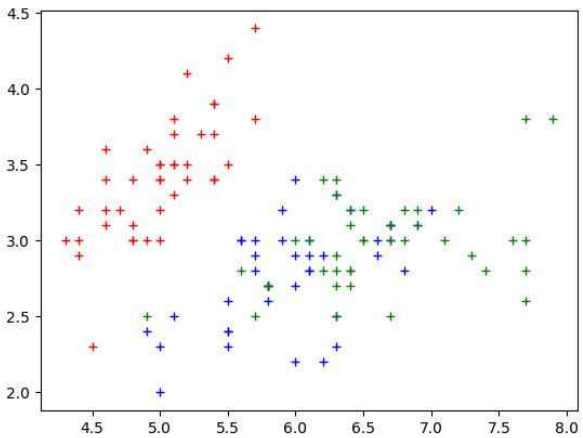


각 test sample의 분류 결과는 위의 그림과 같다. 각 test sample들은 해당하는 class로 잘 분류 됐음을 알 수 있다.

2) In this problem we will use the first 2 features of the Iris data.

이번 문제에서는 feature를 2개로 제한하여 분류기를 구현한다. 그래프는 matplotlib를 사용하여 그렸다.

a. Plot the training data samples




```
# train set
trainX1 = train1[:, 0]; trainY1 = train1[:, 1]
trainX2 = train2[:, 0]; trainY2 = train2[:, 1]
trainX3 = train3[:, 0]; trainY3 = train3[:, 1]
plt.plot(trainX1, trainY1, 'r+')
plt.plot(trainX2, trainY2, 'b+')
plt.plot(trainX3, trainY3, 'g+')
```

+(붉은 +기호)는 class1의 training data samples을 나타내고 +(푸른 +기호)는 class2의 training data samples을, +(초록 +기호)는 class3의 training data samples을 나타낸다.

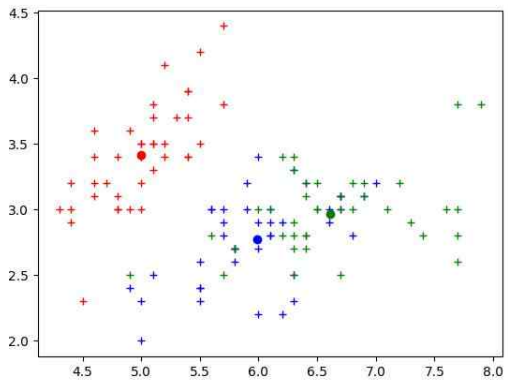
b. Estimate the mean and covariance of each class.

```
mean class 1
[ 4.9975  3.4175]
covariance class 1
[[ 0.13512179  0.10748077]
 [ 0.10748077  0.15686538]]

mean class 2
[ 5.99    2.7775]
covariance class 2
[[ 0.28041026  0.10079487]
 [ 0.10079487  0.11666026]]

mean class 3
[ 6.61  2.97]
covariance class 3
[[ 0.44194872  0.0854359 ]
 [ 0.0854359   0.09497436]]
```

```
# 평균 벡터를 구하는 함수
def meanVector(dataClass):
    mean = np.sum(dataClass, axis=0)[:2]/40
    return mean

# 공분산 벡터를 구하는 함수
def cov(classData, m):
    covV = np.zeros((2, 2)).astype(float)
    for i in range(0, 2):
        for j in range(0, 2):
            data1 = classData[:,i] - m[i]
            data2 = (classData[:, j] - m[j]).T
            covV[i][j] = np.sum(np.dot(data1, data2))/39
    return covV
```

각 class의 mean과 covariance는 위의 데이터와 같다.

c. Plot the means and the contours for which the Mahalanobis distance = 2.




```
# train set
trainX1 = train1[:, 0]; trainY1 = train1[:, 1]
trainX2 = train2[:, 0]; trainY2 = train2[:, 1]
trainX3 = train3[:, 0]; trainY3 = train3[:, 1]
plt.plot(trainX1, trainY1, 'r+')
plt.plot(trainX2, trainY2, 'b+')
plt.plot(trainX3, trainY3, 'g+')

# mean
plt.plot(mVClass1[0], mVClass1[1], 'ro')
plt.plot(mVClass2[0], mVClass2[1], 'bo')
plt.plot(mVClass3[0], mVClass3[1], 'go')
```

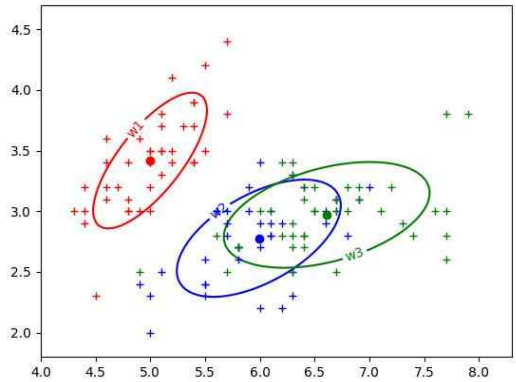
왼쪽 그림에서 ●(붉은 ●기호)는 class1의 mean을 나타내고 ●(푸른 ●기호)는 class2의 mean을, ●(초록 ●기호)는 class3의 mean을 나타낸다.

Mahalanobis distance는 공분산 확률을 이용하여 데이터가 얼마나 일어나기 힘든 일인지 즉 평균과 표준편차를 고려했을 때 얼마나 중심에서 멀리 떨어져 있는지를 나타낸다. 그 식은 아래와 같다.

$$(\underline{x}-\underline{\mu}_i)^T \Sigma^{-1} (\underline{x}-\underline{\mu}_i) = k$$

위의 식에서 k를 이항하여 우항을 0으로 만든 뒤 Mahalanobis distance를 구하는 함수를 구현하였다. 아래와 같다.

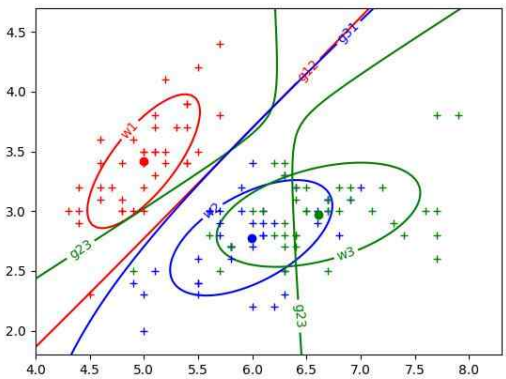
```
# Mahalanobis distance = 2
def mahalanobis(x, y, m, c):
    covT = lin.inv(c)
    return (x-m[0])*(covT[0][0]*(x-m[0])+covT[1][0]*(y-m[1])) + (y-m[1])*(covT[0][1]*(x-m[0])+covT[1][1]*(y-m[1])) - 2
```




```
# plot
x = np.linspace(4, 8.3, 100)
y = np.linspace(1.8, 4.7, 100)
X,Y = np.meshgrid(x,y)

plt.figure()

# mahalanobis = 2
w1 = plt.contour(X, Y, mahalanobis(X,Y, mVClass1, covV1), 0, colors='red')
plt.clabel(w1, fontsize=10, inline=1,fmt = 'w1')
w2 = plt.contour(X, Y, mahalanobis(X,Y, mVClass2, covV2), 0, colors='blue')
plt.clabel(w2, fontsize=10, inline=1,fmt = 'w2')
w3 = plt.contour(X, Y, mahalanobis(X,Y, mVClass3, covV3), 0, colors='green')
plt.clabel(w3, fontsize=10, inline=1,fmt = 'w3')
```

위의 그림은 각 class의 중심에서 Mahalanobis distance = 2인 그래프를 나타낸 것이다.

d. Determine the decision boundaries and plot them.



class 1,2를 구분하는 g12(붉은 선), class 2,3을 구분하는 g23(초록선), class3, 1을 구분하

는 g31(푸른 선)은 위의 그림과 같다.

$$g_i(x) = x^t V_i x + v_i^t x + v_{i0}$$

where $V_i = -\frac{1}{2}\Sigma_i^{-1}$

$$v_i = \Sigma_i^{-1}\mu_i,$$

and $v_{i0} = -\frac{1}{2}\mu_i^t \Sigma_i^{-1}\mu_i - \frac{1}{2}\ln|\Sigma_i| + \ln P(w_i)$

$g(x) = g_1(x) - g_2(x)$, decide w_1 if $g(x)>0$.

각 class의 discriminant function을 구한 뒤 빼주어 g12, g23, g31을 구한다.

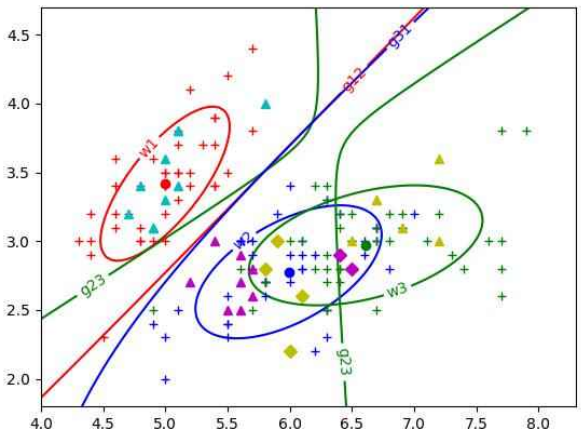
```
# decision boundary를 결정하는 함수
def decisionBoundary(x, y, m1, c1, m2, c2):
    cT1 = lin.inv(c1)
    com1 = -0.5*((cT1[0][0]*x+cT1[1][0]*y)*x + (cT1[1][0]*x+cT1[1][1]*y)*y)
    com2 = x*(cT1[0][0]*m1[0]+cT1[0][1]*m1[1]) + y*(cT1[1][0]*m1[0]+cT1[1][1]*m1[1])
    com3 = -0.5*(m1[0]*(cT1[0][0]*m1[0]+cT1[1][0]*m1[1])+m1[1]*(cT1[0][1]*m1[0]+cT1[1][1]*m1[1])) ₩
          -0.5*math.log(lin.det(c1))
    g1 = com1 + com2 + com3

    cT2 = lin.inv(c2)
    com1 = -0.5 * ((cT2[0][0] * x + cT2[1][0] * y) * x + (cT2[1][0] * x + cT2[1][1] * y) * y)
    com2 = x * (cT2[0][0] * m2[0] + cT2[0][1] * m2[1]) + y * (cT2[1][0] * m2[0] + cT2[1][1] * m2[1])
    com3 = -0.5 * (m2[0] * (cT2[0][0] * m2[0] + cT2[1][0] * m2[1]) + m2[1] * (cT2[0][1] * m2[0] + cT2[1][1] * m2[1])) ₩
          - 0.5 * math.log(lin.det(c2))
    g2 = com1 + com2 + com3

    return g1 - g2
```

```
g12 = plt.contour(X, Y, decisionBoundary(X, Y, mVClass1, covV1, mVClass2, covV2), 0, colors='red')
plt.clabel(g12, fontsize=10, inline=1,fmt = 'g12')

g23 = plt.contour(X, Y, decisionBoundary(X, Y, mVClass2, covV2, mVClass3, covV3), 0, colors='green')
plt.clabel(g23, fontsize=10, inline=1,fmt = 'g23')

g31 = plt.contour(X, Y, decisionBoundary(X, Y, mVClass3, covV3, mVClass1, covV1), 0, colors='blue')
plt.clabel(g31, fontsize=10, inline=1,fmt = 'g31')
```

Test sample을 각 decision boundary $g_{i,j}$에 대입했을 때 값이 0보다 크면 class j에 속하지 않는 data임을 알 수 있다. 따라서 decision boundary에 2번 test sample을 대입하면 test sample이 어떤 class에 속하게 되는지 알 수 있다.

각 decision boundary는 각 feature에 대한 2차식을 가지므로 쌍곡선을 가진다.

e. Add the test data set to the plot.



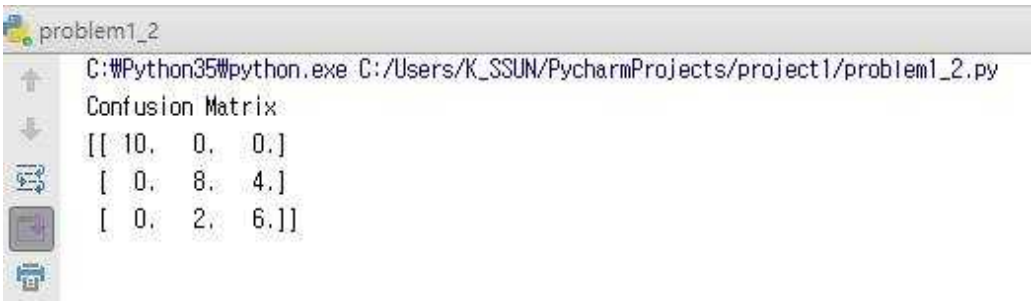

```
for i in test1, test2, test3:
    for j in i :
        if(j[2] == 1):
            result = decision(j[0], j[1])
            confusionM[result - 1, int(j[2]) - 1] += 1
            if(result == int(j[2])):
                plt.plot(j[0], j[1], 'c^')
            else:
                plt.plot(j[0], j[1], 'cD')
        elif(j[2] == 2):
            result = decision(j[0], j[1])
            confusionM[result - 1, int(j[2]) - 1] += 1
            if (result == int(j[2])):
                plt.plot(j[0], j[1], 'm^')
            else:
                plt.plot(j[0], j[1], 'mD')
        else:
            result = decision(j[0], j[1])
            confusionM[result - 1, int(j[2]) - 1] += 1
            if (result == int(j[2])):
                plt.plot(j[0], j[1], 'y^')
            else:
                plt.plot(j[0], j[1], 'yD')
```

```
# decision boundary함수를 이용하여 class 결정
def decision(x, y):

    return (3 if decisionBoundary(x, y, mVClass3, covV3, mVClass1, covV1) > 0 else 1) #
        if (decisionBoundary(x, y, mVClass1, covV1, mVClass2, covV2) > 0)  #
        else (2 if decisionBoundary(x, y, mVClass2, covV2, mVClass3, covV3) > 0 else 3)
```

각 decision boundary에 test sample을 대입하여 값을 비교하여 class를 분류하였다.

위의 그림에서 ▲(사이안 ▲기호)는 class1의 test sample중 class1로 분류된 데이터를 나타내고 ▲(마젠타 ▲기호)는 class2의 test sample중 class2로 분류된 데이터를, ▲(노란 ▲기호)는 class3의 test sample중 class3로 분류된 데이터를 나타낸다. ◆(마젠타 ◆기호)는 class2의 test sample중 class2로 분류되지 않은(class3으로 분류됨) 데이터를 나타내고 ◆(노란 ◆기호)는 class3의 test sample중 class3으로 분류되지 않은(class2로 분류됨) 데이터를 나타낸다.

f. Classify the test data set and construct the confusion matrix. Compare the results with the previous problem.



1)의 결과값과 비교하였을 때 해당하는 class로 분류되지 않은 data가 있음을 알 수 있다.

· **Project 결과**

1)의 결과값과 2)의 결과값을 비교하였을 때 2)에서는 class2인데 class3으로 분류된 sample data가 2개, class3인데 class2로 분류된 sample data가 4개 있는 것을 알 수 있다. 1)과 2)의 차이점은 학습하는 feature의 개수인데 feature 4개를 가지고 학습했을 때와 달리 feature 2개만을 가지고 학습하는 경우에는 정확도가 떨어짐을 알 수 있다. 따라서 같은 sample개수를 가지고 학습을 하는 경우 더 많은 feature을 가지는 data로 학습하였을 때 정확도가 더 높음을 알 수 있다.