CS4220 Exercise

timo9madrid7

November 2020

1 Week 1

1.1 Decision Theory

- Exercise 1.1
  1. 0.5
  2. (1, 2)
  3. (0.5, 1)
  4. 1
- Exercise 1.2
  1. $$p(x|w_1) = \frac{1}{\sqrt{\pi}}e^{-x^2} \tag{1}$$
     $$p(x|w_2) = \frac{1}{\sqrt{\pi}}e^{-(x-1)^2} \tag{2}$$
     In exe 1.1, it calculated
     $\frac{p(x|w_1)}{p(x|w_2)} = 1$,
     and now it calculates
     $\frac{p(x|w_1)}{p(x|w_2)} = 2 \rightarrow x = \frac{1-ln(0.5)}{2} \approx 0.1534$
  2. $\frac{p(x|w_1)p(w_1)}{p(x|w_2)p(w_2)} = 2 \rightarrow x \in (0.5, 1)$
- Exercise 1.3
  1. $x = 3 \rightarrow w_2$
     $x = -0.5 \rightarrow w_1$
     $x = +0.5 \rightarrow w_1$
  2. $p(\mathbf{x}|w_1) = p(\mathbf{x}|w_1) \rightarrow \mathbf{x} = 0.8$
- Exercise 1.4

1. $x = 3 \rightarrow w_2$
   $x = -0.5 \rightarrow w_1$
   $x = +0.5 \rightarrow w_1$
2. $x = \frac{2}{3}$

- Exercise 1.5
  $\frac{1}{2} \times \frac{1}{2} \times 1 \times 0.2 = 0.05$

1.2 Prtools for Python

1. Exercise 1.6

   (a)

   ```
   np.random.seed(0)
   x = np.random.random([10,3])
   for i in x:
       print(i)
   ```

   (b)
   mean(x): the mean of all data
   mean(x, axis=0): the mean of row
   mean(x, axis=1): the mean of column

2. Exercise 1.7
   done

3. Exercise 1.8
   done

4. Exercise 1.9
   done

5. Exercise 1.10
   what does myClassifier(myData) do?

6. Exercise 1.11

   (a)
   Zero for 2.3, 2.4 and 2.5; negative for 2.6

   (b)
   correlation is positive now

7. Exercise 1.12

```
from scipy.stats import multivariate_normal
   def gauss_fun(X, Y):
      mux = [0]
      muy = [0]
      sx = [1]
      sy = [3]
      rho = [0]
      d = np.dstack([X, Y])
      z = None
      for i in range(len(mux)):
         mean = [mux[i], muy[i]]
         # Extract covariance matrix
         cov = [[sx[i] * sx[i], rho[i] * sx[i] *
             sy[i]], [rho[i] * sx[i] * sy[i], sy[i] *
             sy[i]]]
         gaussian = multivariate_normal(mean = mean,
             cov = cov)
         z_ret = gaussian.pdf(d)
         if z is None:
            z = z_ret
         else:
            z += z_ret
      return z

   x = np.linspace(-7, 7, 100)
   y = np.linspace(-7, 7, 100)
   X, Y = np.meshgrid(x, y)
   Z = gauss_fun(X, Y)

   plt.contour(X, Y, Z, colors='black');
```

8. Exercise 1.13

```
mySample = pr.numpy.random.randn(1000, 2)
myData = pr.prdataset(mySample)
myClassifier = pr.gaussm()
myClassifier.train(myData)
pr.scatterd(myData)
pr.plotm(myClassifier)
```

9. Exercise 1.14
   Gaussian Quadratic Decision Boundary: ellipsoid, parabolas, hyperbolas, pairs of lines
   figures 2.7 - 2.9 are brilliant
   Gaussian Linear Decision Boundary: hyper-planes

$$g_i(x) = -\frac{1}{2}X^T\Sigma_i^{-1}X+\frac{1}{2}X^T\Sigma_i^{-1}\mu_i-\frac{1}{2}\mu_i^T\Sigma_i^{-1}\mu_i+\frac{1}{2}\mu_i^T\Sigma_i^{-1}X+lnP(w_i)+c_i \tag{3}$$

equal co-variance matrices $\rightarrow$ no quadratic term

$$g_i(x) = X^T\sigma^{-1}\mu_i + w_{i0} \tag{4}$$

Diagonal co-variance matrix with equal elements $\rightarrow \Sigma = \sigma^2 I$

$$g_i(x) = \frac{1}{\sigma^2}\mu_i^T X + w_{i0} \tag{5}$$

Decision Boundary becomes,

$$g_{ij}(x) \equiv g_i(x) - g_j(x) = (\mu_i - \mu_j)(x - x_0) = 0 \tag{6}$$

where $x_0 = \frac{1}{2}(\mu_i + \mu_j) - \sigma^2 ln(\frac{p(w_i)}{p(w_j})\frac{\mu_i-\mu_j}{||\mu_i-\mu_j||^2}$

For this exercise, just replace σ^2 by c

10. Exercise 1.15

```
myData = pr.gendats((10,10), dim=2, delta=1)
labelData = pr.labeld(myData)

myLDC = pr.ldc()
myQDC = pr.qdc()

myLDC.train(myData)
myQDC.train(myData)

pr.scatterd(myData)
pr.plotc(myLDC)
pr.plotc(myQDC)
```

(a)
hyper-planes.

(b)
it is curve.

(c)
ldc uses a line as decision boundary, whereas qdc uses a curve as decision boundary.

(d)
When there are two samples for each class, the number of the samples is not enough to support QDC estimator. When there is only one sample for each class, it is not enough to support LDC estimator. Hence, for QDC, the quantity of estimating samples should be more than two, while for LDC, the quantity of estimating samples should be more than one.

11. Exercise 1.16

(a)

```
set1 = pr.numpy.random.randn(10,2)
label1 = np.zeros((1,10))
set2 = pr.numpy.random.randn(10,2)
label2 = np.ones((1,10))

setAll = np.vstack([set1,set2])
labelAll = np.vstack([label1.T,label2.T])

myData = pr.prdataset(setAll, labelAll)

myLDC = pr.ldc()
myQDC = pr.qdc()

myLDC.train(myData)
myQDC.train(myData)

pr.scatterd(myData)
pr.plotc(myLDC)
pr.plotc(myQDC)

errorLDC = pr.testc(myLDC(myData))
errorQDC = pr.testc(myQDC(myData))

print("errorLDC:", errorLDC*20)
print("errorQDC:", errorQDC*20)
```

(b)

```
labelLDC = myData*myLDC*pr.labeld()
labelQDC = myData*myQDC*pr.labeld()

errLDC = 20 - np.sum(labelLDC == labelAll)
errQDC = 20 - np.sum(labelQDC == labelAll)

print("errLDC:", errLDC)
print("errQDC:", errQDC)
```

12. Exercise 1.17

```
a = pr.gendats((20,20), dim=1, delta=8)

h = 0.8
```

```
a = pr.prdataset(+a)
w = pr.parzenm(a, h)

pr.scatterd(a)
pr.plotm(w, gridsize=100)
```

To get an honest estimate of the log-likelihood, we have to evaluate the log-likelihood on a test set

13. Exercise 1.18

    (a)

```
a = pr.gendats((20,20), dim=1, delta=8)
a = pr.prdataset(+a)
h =  [0.01,0.05,0.1,0.25,0.5,1,1.5,2,3,4,5]
LL = np.zeros(len(h))
for i in range(len(h)):
    w = pr.parzenm(a, h[i])
    LL[i] = np.sum(np.log(+(w(a))))

plt.plot(h,LL)
plt.ylabel("Log-likelihood")
plt.xlabel("h")
plt.show()
```

    (b)

```
pr.scatterd(a)
pr.plotm(pr.parzenm(a, h[np.argmax(LL)]))
```

14. Exercise 1.19

```
import warnings
warnings.filterwarnings("ignore")

# Split into trn and tst, both 50%
[dataTrain, dataTest] = pr.gendat(a, 0.5, seed=1)

# Array of h to try
h = [0.01,0.05,0.1,0.25,0.5,1,1.5,2,3,4,5]

LL_train = np.zeros(len(h))
LL_test = np.zeros(len(h))

for i in range(len(h)):
    w = pr.parzenm(dataTrain, h[i])
    LL_train[i] = np.sum(np.log(+(w(dataTrain))))
    LL_test[i] = np.sum(np.log(+(w(dataTest))))

plt.plot(h, LL_train, "b-*")
plt.plot(h, LL_test, 'g-*')
plt.ylabel("Log-likelihood")
plt.xlabel("h")
plt.legend("train", "test")
plt.show()
```

15. Exercise 1.20
    The quadratic, linear and Fisher discriminant are not affected, the nearest mean and the k-NN are. It is because in the first classifiers a co variance matrix $\hat{\Sigma}$ is estimated. Therefore automatically the scale of the feature values is estimated in these classifiers.
    In many cases it is an advantage, because it means that you don't need to optimize the scaling of the features. The disadvantage is that the training set should be large enough to make the estimation of the scale reliable.