Machine Learning HW4

b06502152, 許書銓

Deterministic Noise

1. The answer should be (c).

   Our goal is to minimize the error to obtaing the optimal linear hypothesis.

$$
\begin{aligned}
E_{in} &= \int_0^2 (wx - e^x)^2 dx \\
&= \int_0^2 (w^2x^2 - 2wxe^x + e^{2x}) dx \\
&= \frac{w^2}{8}x^3 - 2w(x-1)e^x + \frac{1}{2}e^{2x} \\
&= \frac{8}{3}w^2 - (2we^2 + 2w) + \frac{1}{2}e^4 - \frac{1}{2} \qquad (1.0)
\end{aligned}
$$

   To minimize E_{in} , we would like to take derivative on equation (1.0).

$$
\frac{8}{3}w - e^2 - 1 = 0 \qquad (1.1)
$$

   Hence,

$$w = \frac{3 + 3e^2}{8} \tag{1.2}$$

The magnitude of deterministic noise would be

$$|e^x - \frac{3 + 3e^x}{8}| \tag{1.3}$$

Learning Curve

2. Tha answer should be (b.)

- $\mathbb{E}_D[E_{in}(A(D))] < \mathbb{E}_D[E_{out}(A(D))]$
- $\mathbb{E}_D[E_{in}(A(D))] = \mathbb{E}_D[E_{out}(A(D))]$

Considering the first and second arguments, we can prove that the arguments would not be always false by giving a example that the argument may be true.

Given that the optimal hypothetsis gor finding the optimal $h*$ for E_{out}.

$$h^* = argmin_{h \in H} \ E_{in}(h) \tag{2.0}$$

The following function will be true for that $h*$ will be the hypothesis which make E_{out} the smallest.

$$E_{out}(h^*) \ \leq \ E_{out}(A(D)) \tag{2.1}$$

Considering that if the condition that $E_{in}(A(D)) = E_{out}(h^*)$ holds, from equation (2.1)

$$E_{in}(A(D)) \ = \ E_{out}(h^*) \ \leq \ E_{out}(A(D)) \tag{2.2}$$

Furthermore, each examples is generated i.i.d from a distribution P.

$$\mathbb{E}_D[E_{in}(A(D))] \;\leq\; \mathbb{E}_D[E_{out}(A(D))] \tag{2.3}$$

Then, it is obviously that the first and second arguments may be true.

- $\mathbb{E}_D[E_{in}(A(D))] > \mathbb{E}_D[E_{out}(A(D))]$

As for disproving the third argument, which is the above one, is always false, we try to use prove the condition by proving the argument will lead to contradiction.

Since $E_{out}(h^*)$ will be the smallest E_{out} over D.

$$E_{out}(h^*) \;=\; \mathbb{E}_D[E_{out}(h^*)] \;\leq\; \mathbb{E}_D[E_{out}(A(D))] \tag{2.4}$$

Since $E_{in}(A(D))$ will be the smallest E_{in} over D.

$$\mathbb{E}_D[E_{in}(A(D))] \;\leq\; \mathbb{E}_D[E_{in}(h^*)] \tag{2.5}$$

Since each examples is generated i.i.d from a distribution P.

$$\mathbb{E}_{\mathbb{D}}[E_{in}(h)] \;=\; \mathbb{E}_{x\sim p}[err(h(x), y)] \;=\; E_{out}(h) \tag{2.6}$$

From equations (2.4), (2.5) and (2.6),

$$\begin{aligned}
\mathbb{E}_D[E_{out}(h^*)] \;&\leq\; \mathbb{E}_D[E_{out}(A(D))] \\
&<\; \mathbb{E}_D[E_{in(A(D))}] \qquad \text{(from the third argument)} \\
&\leq\; \mathbb{E}_D[E_{in}(h^*)] \qquad (2.7)
\end{aligned}$$

Then from equations (2,4), (2.6) and (2.7),

$$E_{out}(h^*) \;=\; \mathbb{E}_D[E_{out}(h^*)] \;<\; \mathbb{E}_D[E_{in}(h^*)] = E_{out}(h^*) \tag{2.8}$$

Then, it is obviously that equation will lead to contradiction. Then, the third argument will alwalys be false.

Noisy Virtual Examples

3. The answer should be (d).

Here, we denote that

$$X_h = \begin{bmatrix} X^T \tilde{X}^T \end{bmatrix}^T = \begin{bmatrix} X \\ \tilde{X} \end{bmatrix} \tag{3.0}$$

$$X_h^T = \begin{bmatrix} X^T \tilde{X}^T \end{bmatrix} \tag{3.1}$$

Hence, the expectation value will be

$$\begin{aligned} \mathbb{E}(X_h^T X_h) &= \mathbb{E}(\begin{bmatrix} X^T \tilde{X}^T \end{bmatrix} \begin{bmatrix} X \\ \tilde{X} \end{bmatrix}) \\ &= \mathbb{E}(X^T X + (\tilde{X}^T \tilde{X})) \\ &= \mathbb{E}(X^T X + ((X+\epsilon)^T (X+\epsilon))) \\ &= (X^T X + X^T X) \;+\; \mathbb{E}(2X\epsilon) \;+\; \mathbb{E}(\epsilon^2) \\ &= 2X^T X + 0 + N\sigma^2 I \\ &= 2X^T X + N\sigma^2 I_{d+1} \end{aligned} \tag{3.2}$$

4. The answer should be (e).

From the equation (3.0) and (3.1), we can directly write down the expected value

$$\begin{aligned}
\mathbb{E}(X_h^T y_h) &= \mathbb{E}(\begin{bmatrix} X^T & \tilde{X}^T \end{bmatrix} \begin{bmatrix} y \\ y \end{bmatrix}) \\
&= X^T y + \mathbb{E}((X+\epsilon)y) \\
&= 2X^T y + 0 \\
&= 2X^T y + \mathbb{E}(\epsilon y) \qquad (4.0) \\
&= 2X^T y + 0 \\
&= 2X^T y \qquad (4.1)
\end{aligned}$$

Regularization

5. The answer should be (d).

- If $\lambda = 0$, then

$$\begin{aligned}
v = (Z^T Z)^{-1} Z^T y &= (Q^T X^T X Q)^{-1} Z^T y \\
&= (Q^T Q \Gamma Q Q^T)^{-1} Q^T X^T y \\
&= (\Gamma)^{-1} Q^T X^T y \qquad (5.0)
\end{aligned}$$

- If $\lambda > 0$, then

$$u = (Z^T Z + \lambda I)^{-1} Z^T y = (\Gamma + \lambda I)^{-1} Q^T X^T y \qquad (5.1)$$

$$\frac{u_i}{v_i} = \frac{(\gamma_i + \lambda)^{-1}}{\gamma_i^{-1}} = \frac{\gamma_i}{(\gamma_i + \lambda)} \tag{5.2}$$

6. The answer should be (a.)

   The optimal solutioin $w*$ is the solution of

$$\min_{w \in \mathbb{R}} \frac{1}{N} \Sigma_{n=1}^{N} (wx_n - y_n)^2 + \frac{\lambda}{N} w^2 \tag{6.0}$$

   Hence, $w*$ can be derived by derivation of the equation (6.0) .

$$\frac{2}{N} \Sigma_{n=1}^{N} (wx_n - y)x_n + \frac{2\lambda}{N} w = 0$$

$$\Sigma_{n=1}^{N} wx_n^2 - \Sigma_{n=1}^{N} x_n y + \lambda w = 0$$

$$w = (\frac{\Sigma_{n=1}^{N} x_n y}{\Sigma_{n=1}^{N} x_n^2 + \lambda}) \tag{6.1}$$

   With $C = w^2$, we have

$$w = (\frac{\Sigma_{n=1}^{N} x_n y}{\Sigma_{n=1}^{N} x_n^2 + \lambda})^2 \tag{6.2}$$

7. The answer should be (d).

   Since the estimation is the optimal solution of

$$\min_{w \in \mathbb{R}} \frac{1}{N} \Sigma_{n=1}^{N} (y - y_n)^2 + \frac{2K}{N} \Omega(y) \tag{7.0}$$

   The optimal solution fulfills that make the derivative's of equation (7.0) . Assume $\Omega = (ay + b)^2$

$$\frac{2}{N}\,\Sigma_{n=1}^{N}\,(y-y_n)+\frac{2K}{N}\,(2ay+2b)=0$$

$$\Sigma_{n=1}^{N}\,(y-y_n)+K\,(2ay+2b)=0$$

$$\Sigma_{n=1}^{N}\,y+2aKy=\Sigma_{n=1}^{N}\,y_n-2bK$$

$$Ny+2aKy=\Sigma_{n=1}^{N}\,y_n-2bK$$

$$y=\frac{\Sigma_{n=1}^{N}\,y_n-2bK}{N+2aK} \tag{7.1}$$

By comparin the coefficients of estimation

$$y=\frac{\Sigma_{n=1}^{N}\,y_n+K}{N+2K} \tag{7.2}$$

and the equation (7.1). we realize that a = 1 and b = -0.5.

Hence,

$$\Omega(y)=(y-0.5)^2 \tag{7.3}$$

8. The answer should be (b).

From the scaling transformatio itself, $\tilde{w}=\Gamma w$, hance we can rewrite the equation into,

$$\min_{\tilde{w}\in\mathbb{R}^{d+1}}\ \frac{1}{N}\Sigma_{n=1}^{N}(\tilde{w}^T\Phi(x_n)-y_n)^2+\frac{\lambda}{N}(\tilde{w}^T\tilde{w}) \tag{8.0}$$

$$\min_{\tilde{w}\in\mathbb{R}^{d+1}}\ \frac{1}{N}\Sigma_{n=1}^{N}(w^T\Gamma^T\Gamma^{-1}x_n-y_n)^2+\frac{\lambda}{N}(w^T\Gamma^T\Gamma w) \tag{8.1}$$

$$\min_{\tilde{w}\in\mathbb{R}^{d+1}}\ \frac{1}{N}\Sigma_{n=1}^{N}(w^Tx_n-y_n)^2+\frac{\lambda}{N}(w^T\Gamma^2 w) \tag{8.1}$$

Hence, $\Omega(w)=w^T\Gamma^2w$.

9. The answer should be (b).

Let $X'=[x_1,x_2,\ldots,x_n,\tilde{x_1},\tilde{x_2},\ldots,\tilde{x_k}]^T=\begin{bmatrix}X\\ \tilde{X}\end{bmatrix}$, $y'=[y_1,y_2,\ldots,y_n,\tilde{y_1},\tilde{y_2},\ldots,\tilde{y_k}]^T=\begin{bmatrix}y\\ \tilde{y}\end{bmatrix}$. Hence, we can rewrite the original equation to

$$\min_{w} \frac{1}{N+K} \Sigma_{n=1}^{N+K} (w^T x'_n - y'_n)^2 \tag{9.0}$$

$$\min_{w} \frac{1}{N+K} ||X'w - y||^2 \tag{9.1}$$

Hence, the optimal solution of the equation (9.1) is,

$$\begin{aligned} w &= ((X')^T X')^{-1} (X')^T y \\ &= (X^T X + \tilde{X}^T \tilde{X}) \quad (X^T y + \tilde{X}^T \tilde{y}) \end{aligned} \tag{9.2}$$

Now, consider the problem "scale regularization"

$$\min_{w \in \mathbb{R}^{d+1}} \frac{1}{N} \Sigma_{n=1}^{N} (w^T x_n - y_n)^2 + \frac{\lambda}{N} \Sigma_{i=0}^{d} \beta_i w_i^2 \tag{9.3}$$

the optimal solution of the equation (9.3) is.

$$\frac{2}{N} (X^T X w - X^T y) + \frac{2\lambda}{N} (\beta w) = 0$$

$$(X^T X + \lambda \beta) w = X^T y$$

$$w = (X^T X + \lambda \beta) X^T y \tag{9.4}$$

Comparing the coefficients in equation (9.4) and equation (9.3).

$$\tilde{X} = \sqrt{\lambda} \sqrt{\beta} I, \quad \tilde{y} = 0 \tag{9.5}$$

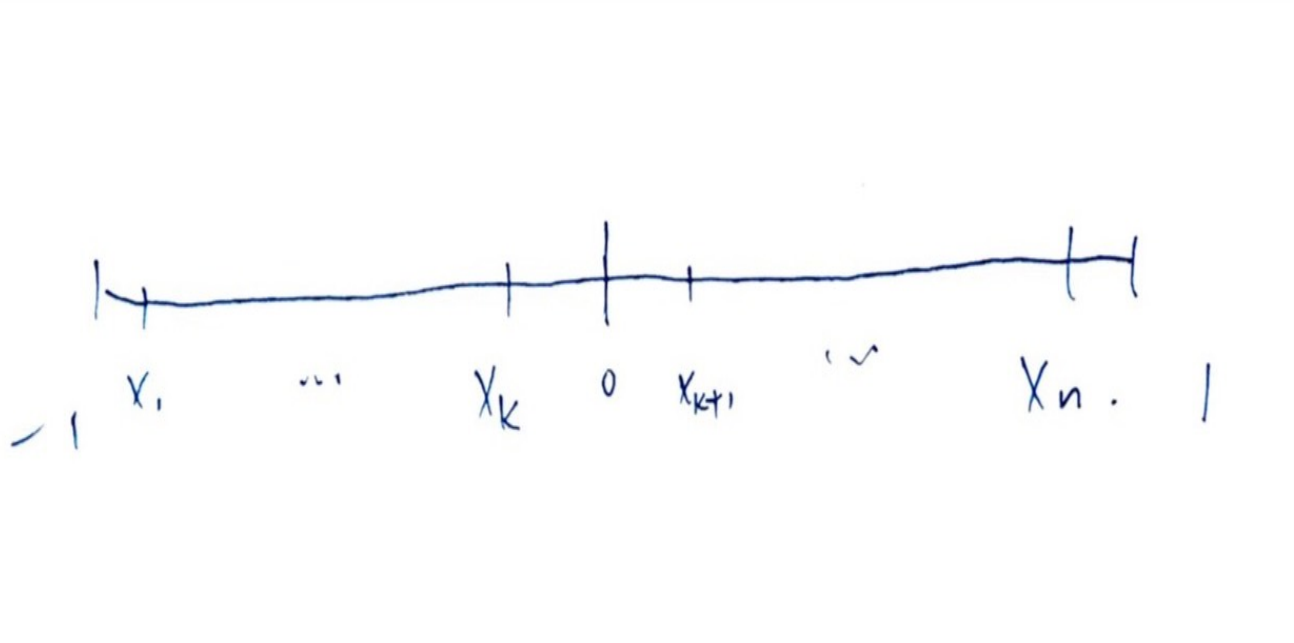
Leave-one-out

10. The answer should be (e).

Since the constant classifier will always predict the majority class, the class of the one chosen to be validation set will leave as the minority class. Hence, the e_n will always to be 1.

$$E_{loocv} = \frac{1}{N} \Sigma_{n=1}^{N} e_n = \frac{1}{N} * N * 1 = 1 \tag{10.0}$$

11. The answer should be (c).



From the upper picture, we can consider the problem into cases

- If we choose x_k or x_{k+1} as validation data

  In this scenario, if x_k is choosed as validation data, then θ in the decision stump may fall in the right of x_k, which may lead to the mislabeling in validation data. So does x_{k+1} is choosed.

- If we choose other points than x_k or x_{k+1} as validation data

  In this scenario, if data other than above two points chosen as validation data, let's say x_1 for instance, then θ in decision stump may fall between x_k and x_{k+1}. Hence, the validatioin data will alwalys predict in the same way as its label y (sign value).

Considering above two cases, we can conclude that the tightest upper bond of the leave-one-out error of the decision stump is $\frac{2}{N}$.

12. The answer should be (e).

We first compute the square error of loocv of constant model.

- x_1 is choose to be validation part, and then the square would be $(0-\frac{1}{2}(2+0))^2=1$
- x_2 is choose to be validation part, and then the square would be $(2-\frac{1}{2}(0+0))^2=4$
- x_3 is choose to be validation part, and then the square would be $(0-\frac{1}{2}(0+2))^2=1$

Hence, the square of the constant model is $1/3(1+4+1)=2$

Then, we compute the square error of loocv of linear model.

- x_2 is choose to be validation part, and then the square would be $(0 - \frac{2}{\rho+3}(-3-3))^2 = \frac{144}{(\rho+3)^2}$
- x_2 is choose to be validation part, and then the square would be $(2-0)^2 = 4$
- x_3 is choose to be validation part, and then the square would be $(0 - \frac{2}{\rho-3}(-3-3))^2 = \frac{144}{(\rho-3)^2}$

From the problem itself, we know that two square error will tie. Thus,

$$2 = \frac{1}{3}\left(\frac{144}{(\rho+3)^2} + 4 + \frac{144}{(\rho-3)^2}\right) \tag{12.0}$$

$$\rho = \sqrt{81 + 36\sqrt{6}} \tag{12.1}$$

13. The answer should be (d).

Since D_{val} is choose k examples from the distribution of $P(\mathbf{x}, y)$, the variance will be

$$\begin{aligned}
\text{Variance}_{D_{val}\sim P^K}[E_{val}(h)] &= \text{Variance}_{D_{val}\sim P^K}[\frac{\Sigma_{i=1}^K err(h(x_i), y_i)}{K}] \\
&= \frac{1}{K^2}\text{Variance}_{D_{val}\sim P^K}[\Sigma_{i=1}^K err(h(x_i), y_i)] \\
&= \frac{\Sigma_{i=1}^K \text{Var}_{D_{val}\sim P^K}[err(h(x_i), y_i)] + \Sigma\Sigma_{i\neq j}\text{Cov}(err(h(x_i), y_i), err(h(x_j), y_j))}{K^2}
\end{aligned} \tag{13.0}$$

Since that err(h(x), y) is from i.i.d examples,

$$Cov(err(h(x_i), y_i), err(h(x_j), y_j)) = 0 \tag{13.1}$$

Hence, from equation (13.0) and (13.1) ,

$$\text{Variance}_{D_{val}\sim P^K}[E_{val}(h)] = \frac{1}{K}\text{Var}_{D_{val}\sim P^K}[err(h(x_i), y_i)] \tag{13.2}$$

Learning Principles

14. The answer should be (c).

$$\varepsilon_{y_1,y_2,y_3,y_4}\left(\min_{w\in\mathbb{R}^{2+1}} E_{in}(w)\right) = \min_{w\in\mathbb{R}^{2+1}} \varepsilon(E_{in}(w)) = \frac{2}{16}\frac{1}{4} = \frac{2}{64} \tag{14.0}$$

Among the 16 combinations, two of them are not linear seperable. Furthermore, each of them make 2 labels wrong.

15. The answer should be (a).

$$E_{out}(g_c) = 1 - p = p\epsilon_+ + (1-p)\epsilon_- \tag{15.0}$$

$$1 - \epsilon_- = p\epsilon_+ - p\epsilon_- + p$$

$$p = \frac{1-\epsilon_-}{1+\epsilon_+-\epsilon_-} \tag{15.1}$$

Expeiments with Regularized Logistic Regression

The L2-regularized logistic regression defines in LIBLINEAR is as,

$$min_w\ w^T w/2 + C\sum log(1+exp(-y_i w^T x_i)) \tag{16.0}$$

However, in the problem set, regularized logistic regression is defined as,

$$min_w\ \frac{\lambda}{N} w^T w + \frac{1}{N}\sum log(1+exp(-y_i w^T x_i)) \tag{16.1}$$

Thus, as we compare the coefficients, we realized that $C = \frac{1}{2\lambda}$.

16. The answer should be (b).

```
#for question 16
```

```
import numpy as np
import math
import sys

# add the path of liblinear package
LIB_LINEAR_PATH = "./liblinear-2.42/python"
sys.path.append(LIB_LINEAR_PATH)
from liblinearutil import *
from sklearn.preprocessing import PolynomialFeatures

#main
data_in = np.genfromtxt("hw4_train.dat.txt")

X = data_in[:, : -1]
d = X.shape[1]
y = data_in[:, -1]

poly = PolynomialFeatures(2)
X_poly = poly.fit_transform(X)

data_out = np.genfromtxt("hw4_test.dat.txt")

X_out = data_out[:, : -1]
y_out = data_out[:, -1]

X_poly_out = poly.fit_transform(X_out)

C = [1/(2*math.pow(10, -4)), 1/(2*math.pow(10, -2)), 1/(2*math.pow(10, 0)),
1/(2*math.pow(10, 2)),1/(2*math.pow(10, 4))]
acc = 0
acc_idx = 0

for i in range(5):
    model = train(y, X_poly, '-s 0 -c ' + str(C[i]) + ' -e 0.000001')
    p_labs, p_acc, p_vals = predict(y_out, X_poly_out, model)
    if(p_acc[0] >= acc):
        acc = p_acc[0]
        acc_idx = i

print(acc_idx)
```

17. The answer should be (a).

```
#for question 17
import numpy as np
import math
import sys

# add the path of liblinear package
LIB_LINEAR_PATH = "./liblinear-2.42/python"
```

```
sys.path.append(LIB_LINEAR_PATH)
from liblinearutil import *
from sklearn.preprocessing import PolynomialFeatures

#main
data_in = np.genfromtxt("hw4_train.dat.txt")

X = data_in[:, : -1]
d = X.shape[1]
y = data_in[:, -1]

poly = PolynomialFeatures(2)
X_poly = poly.fit_transform(X)

acc = 0
acc_idx = 0
C = [1/(2*math.pow(10, -4)), 1/(2*math.pow(10, -2)), 1/(2*math.pow(10, 0)),
1/(2*math.pow(10, 2)), 1/(2*math.pow(10, 4))]

for i in range(5):
    model = train(y, X_poly, '-s 0 -c ' + str(C[i])+ ' -e 0.000001')
    p_labs, p_acc, p_vals = predict(y, X_poly, model)
    #print(p_labs)
    if(p_acc[0] >= acc):
        acc = p_acc[0]
        acc_idx = i

print(acc_idx)
```

18. The answer should be (e).

```
#for question 18
import numpy as np
import math
import sys

# add the path of liblinear package
LIB_LINEAR_PATH = "./liblinear-2.42/python"
sys.path.append(LIB_LINEAR_PATH)
from liblinearutil import *
from sklearn.preprocessing import PolynomialFeatures

#main
data_in = np.genfromtxt("hw4_train.dat.txt")

X = data_in[:, : -1]
y = data_in[:, -1]

poly = PolynomialFeatures(2)
X_poly = poly.fit_transform(X)
```

```
X_train = X_poly[0:120, :]
y_train = y[0:120]
X_val = X_poly[120:, :]
y_val = y[120: ]

acc = 0
acc_idx = 0
C = [1/(2*math.pow(10, -4)), 1/(2*math.pow(10, -2)), 1/(2*math.pow(10, 0)),
1/(2*math.pow(10, 2)), 1/(2*math.pow(10, 4))]
for i in range(5):
    model = train(y_train, X_train, '-s 0 -c ' + str(C[i])+ ' -e 0.000001')
    p_labs, p_acc, p_vals = predict(y_val, X_val, model, '-q')
    if(p_acc[0] >= acc):
        acc = p_acc[0]
        acc_idx = i

#print(acc_idx)
data_out = np.genfromtxt("hw4_test.dat.txt")
X_out = data_out[:, : -1]
y_out = data_out[:, -1]
X_out_poly = poly.fit_transform(X_out)

model = train(y_train, X_train, '-s 0 -c ' + str(C[acc_idx]) + ' -e 0.000001')

p_labs, p_acc, p_vals = predict(y_out, X_out_poly, model, '-q')
print(1-p_acc[0]/100)
```

19. The answer should be (d).

In this problem, we use the fact that the optimal solution will be $\lambda = 10^{-2}$, computing in prob. 18.

```
#for question 19
import numpy as np
import math
import sys

# add the path of liblinear package
LIB_LINEAR_PATH = "./liblinear-2.42/python"
sys.path.append(LIB_LINEAR_PATH)
from liblinearutil import *
from sklearn.preprocessing import PolynomialFeatures

#main
data_in = np.genfromtxt("hw4_train.dat.txt")

X = data_in[:, : -1]
y = data_in[:, -1]

poly = PolynomialFeatures(2)
X_poly = poly.fit_transform(X)
```

```
X_train = X_poly[0:120, :]
y_train = y[0:120]
X_val = X_poly[120:, :]
y_val = y[120: ]

C = [1/(2*math.pow(10, -4)), 1/(2*math.pow(10, -2)), 1/(2*math.pow(10, 0)),
1/(2*math.pow(10, 2)), 1/(2*math.pow(10, 4))]
model = train(y, X_poly, '-s 0 -c ' + str(C[1])+ ' -e 0.000001')

data_out = np.genfromtxt("hw4_test.dat.txt")
X_out = data_out[:, : -1]
y_out = data_out[:, -1]
X_out_poly = poly.fit_transform(X_out)

p_labs, p_acc, p_vals = predict(y_out, X_out_poly, model, '-q')
print(1-p_acc[0]/100)
```

20. The answer should be (c).

```
#for question 20
import numpy as np
import math
import sys

# add the path of liblinear package
LIB_LINEAR_PATH = "./liblinear-2.42/python"
sys.path.append(LIB_LINEAR_PATH)
from liblinearutil import *
from sklearn.preprocessing import PolynomialFeatures

#main
data_in = np.genfromtxt("hw4_train.dat.txt")

X = data_in[:, : -1]
y = data_in[:, -1]

poly = PolynomialFeatures(2)
X_poly = poly.fit_transform(X)

err_min = 1
C = [1/(2*math.pow(10, -4)), 1/(2*math.pow(10, -2)), 1/(2*math.pow(10, 0)),
1/(2*math.pow(10, 2)), 1/(2*math.pow(10, 4))]
for i in range(5):
    err = 0
    for j in range(5):
        X_train_1 = X_poly[0:40*j, :]
        y_train_1 = y[0:40*j]
        X_train_2 = X_poly[40*j+40: , :]
        y_train_2 = y[40*j+40:]
        X_train = X_train_1
```

```
        X_train = np.insert(X_train_2, 0, values=X_train_1, axis=0)
        y_train = np.insert(y_train_2, 0, values=y_train_1, axis=0)

        X_val = X_poly[40*j:40*j+40, :]
        y_val = y[40*j:40*j+40]


        model = train(y_train, X_train, '-s 0 -c ' + str(C[i])+ ' -e 0.000001')
        p_labs, p_acc, p_vals = predict(y_val, X_val, model, '-q')
        err += (1-p_acc[0]/100)

    err /= 5
    if(err <= err_min):
        err_min = err

print(err_min)
```