HTML_2023_HW4

tags: `Personal`

```
Discuss with anonymous and TAs.
Ref: [TA Poy HTML 2023 HW4](https://hackmd.io/@Poy/SJVAvg0Qn)
```

More about Regularization

P1

In order to find the minimum w, we can set the derivative of the optimal solution to be zero, hence, we have

$$
\begin{aligned}
0 &= \frac{\partial}{\partial w}(\frac{1}{N}\sum_{n=1}^{N}(wx_n - y_n)^2 + \frac{\lambda}{N}w^2) \\
&= \frac{2}{N}\sum_{n=1}^{N}x_n(wx_n - y_n) + \frac{2w\lambda}{N} \\
&= \frac{2w}{N}\sum_{n=1}^{N}x_n^2 - \frac{2}{N}\sum_{n=1}^{N}x_n y_n + \frac{2w\lambda}{N} \\
&= \frac{2w}{N}(\sum_{n=1}^{N}x_n^2 + \lambda) - \frac{2}{N}\sum_{n=1}^{N}x_n y_n \\
\frac{2w}{N}(\sum_{n=1}^{N}x_n^2 + \lambda) &= \frac{2}{N}\sum_{n=1}^{N}x_n y_n \\
w(\sum_{n=1}^{N}x_n^2 + \lambda) &= \sum_{n=1}^{N}x_n y_n \\
w^* &= \frac{\sum_{n=1}^{N}x_n y_n}{\sum_{n=1}^{N}x_n^2 + \lambda} \\
\text{or we can use } w^* &= (x^T x + \lambda I)^{-1}x^T y \\
&= (\sum_{n=1}^{N}x_n^2 + \lambda)^{-1} \cdot \sum_{n=1}^{N}x_n y_n \\
&= \frac{\sum_{n=1}^{N}x_n y_n}{\sum_{n=1}^{N}x_n^2 + \lambda} \\
\because C &= (w^*)^2 \\
\therefore C &= (\frac{\sum_{n=1}^{N}x_n y_n}{\sum_{n=1}^{N}x_n^2 + \lambda})^2
\end{aligned}
$$

As a result, we should choose $[a]$ as our solution.

P2

Since the L2-regularized linear regressionin the $\mathcal{Z}$-space is equivalent to regularized linear regression in the $\mathcal{X}$-space, we have

$$
\begin{aligned}
&\min_{\tilde{w}\in\mathbb{R}^{d+1}}\frac{1}{N}\sum_{n=1}^{N}(\tilde{w}^T\Phi(x_n) - y_n)^2 + \frac{\lambda}{N}(\tilde{w}^T\tilde{w}) \\
=&\min_{\tilde{w}\in\mathbb{R}^{d+1}}\frac{1}{N}\sum_{n=1}^{N}(w^T x_n - y_n)^2 + \frac{\lambda}{N}\Omega(w) \\
\therefore &\ \tilde{w}^T\Phi(x_n) = \tilde{w}\Gamma^{-1}(x-u) = \tilde{w}\Gamma^{-1}x = w^T x_n, \tilde{w}^T\tilde{w} = \Omega(w)
\end{aligned}
$$

Since Γ is a diagonal matrix, we know that the inverse of diagonal matrix equals to itself, which means $\Gamma^T = \Gamma$ then we have

$$
\begin{aligned}
w^T &= \tilde{w}\Gamma^{-1} \\
\tilde{w}^T &= w^T\Gamma \\
\tilde{w} &= (w^T\Gamma)^T = \Gamma^T w \\
\therefore \Omega(w) &= \tilde{w}^T\tilde{w} = w^T\Gamma\Gamma w = w^T\Gamma^2 w
\end{aligned}
$$

As a result, we should choose $[b]$ as our solution.

P3

$$
\min_w \frac{1}{N}\sum_{n=1}^{N}\text{err}_{\text{smooth}}(w, x_n, y_n) = \min_w \frac{1}{N}\sum_{n=1}^{N}\text{err}(w, x_n, y_n) + \frac{\lambda}{N}\sum_{n=1}^{N}\Omega(w, x_n)
$$

$$
D_{KL}(P\|Q) = \sum_x P(x)\ln\frac{P(x)}{Q(x)} = \sum_x \frac{1}{2}\ln\frac{\frac{1}{2}}{\text{err}(w, x, y)}
$$

$$
\begin{aligned}
w_{lr} &= \underset{w}{\text{argmin}}\sum_{i=1}^{n} D_{KL}(P(Y_i)\|P(\hat{Y}_i) \\
&= \underset{w}{\text{argmin}}\sum_{i=1}^{n} y_i\ln\frac{y_i}{p_i} + (1-y_i)\ln\frac{(1-y_i)}{(1-p_i)} \\
&= \underset{w}{\text{argmin}}\sum_{i=1}^{n} y_i(\ln y_i - \ln p_i) + (1-y_i)(\ln(1-y_i) - \ln(1-p_i)) \\
&= \underset{w}{\text{argmin}}\sum_{i=1}^{n} -y_i\ln p_i - (1-y_i)\ln(1-p_i) + (y_i\ln y_i + (1-y_i)\ln(1-y_i)) \\
&= \underset{w}{\text{argmin}}\sum_{i=1}^{n} y_i\ln p_i + (1-y_i)\ln(1-p_i)) \\
&= \underset{w}{\text{argmin}}\sum_{i=1}^{n} H(P(Y_i)\|P(\hat{Y}_i) \\
\therefore \Omega(w, x) &= D_{KL}(P_u\|P_h)
\end{aligned}
$$

As a result, we should choose $[a]$ as our solution.

Validation

P4

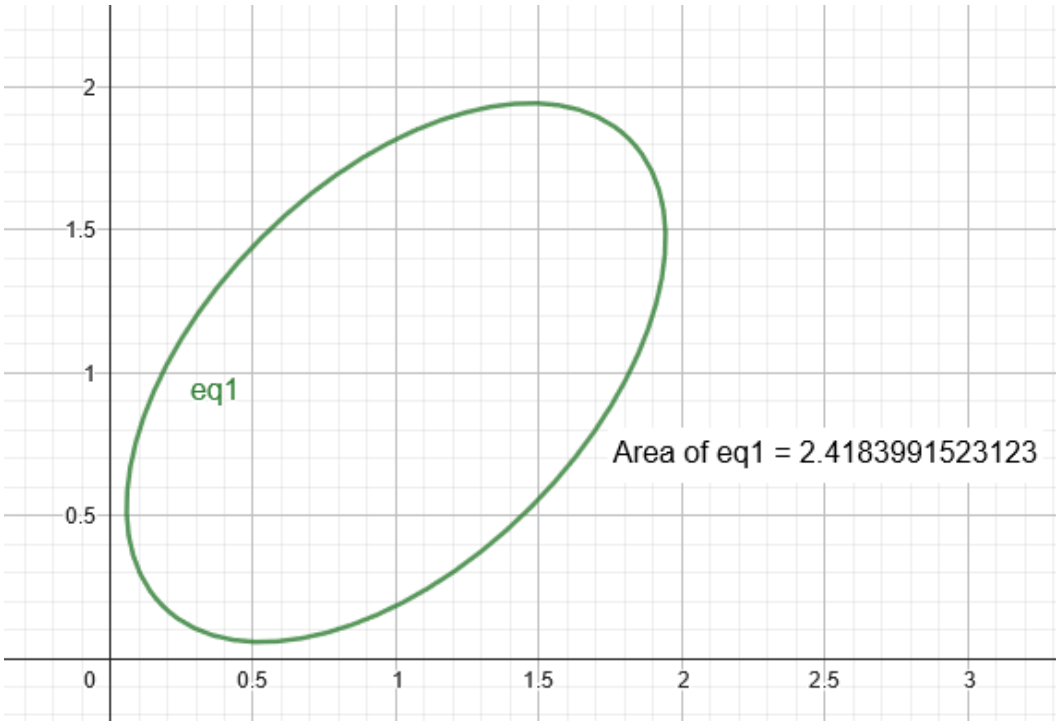
Since constant hypothesis $h(x) = w_0$, we only need to take y into the consideration. By selecting two examples for predicting another one example, we can obtain the equation

$$
\begin{aligned}
E_{loocv} &= \frac{1}{3}\left((y_1 - \frac{y_2+1}{2})^2 + (y_2 - \frac{y_1+1}{2})^2 + (1 - \frac{y_1+y_2}{2})^2\right) \\
&= \frac{1}{2}(y_1^2 + y_2^2 - y_1y_2 - y_1 - y_2 + 1)
\end{aligned}
$$

Since we are finding $E_{loocv} \leq \frac{1}{3}$, we can rewrite the equation to be

$$
\begin{aligned}
&(\frac{x+1}{2}-y)^2+(\frac{y+1}{2}-x)^2+(\frac{x+y}{2}-1)^2=1 \\
\Rightarrow\ &\frac{1}{2}(x^2+y^2-xy-x-y+1)=\frac{1}{3}
\end{aligned}
$$

Drawing the equation with `GeoBeBra`, we can simply get the area, which is $\approx 2.4183 \approx \frac{\pi}{3\sqrt{3}}$



As a result, we should choose $[b]$ as our solution.

P5

$$
\begin{aligned}
&\because E_{out}(h) = \underset{x\sim P}{\mathcal{E}}[h(x) \neq f(x)] \\
&\therefore \underset{\mathcal{D}_{val}\sim\mathcal{P}^K}{\text{Variance}}\left[E_{val}(h)\right] \\
&= \underset{(x,y)\sim\mathcal{P}}{\text{Variance}}\left[\frac{1}{K}\sum_{n=1}^{K}\text{err}(h(x_n),y_n)\right] \\
&= \frac{1}{K^2}\underset{(x,y)\sim\mathcal{P}}{\text{Variance}}\left[\sum_{n=1}^{K}\text{err}(h(x_n),y_n)\right] \\
&= \frac{1}{K^2}\underset{(x,y)\sim\mathcal{P}}{\text{Variance}}\left[\text{err}(h(x_1),y_1)+\text{err}(h(x_2),y_2)+\cdots+\text{err}(h(x_K),y_K)\right] \\
&= \frac{1}{K^2}(K\cdot\underset{(x,y)\sim\mathcal{P}}{\text{Variance}}\left[\text{err}(h(x),y)\right]) \\
&= \frac{1}{K}\underset{(x,y)\sim\mathcal{P}}{\text{Variance}}\left[\text{err}(h(x),y)\right] \\
&\therefore \square = \frac{1}{K}
\end{aligned}
$$

Since the examples (x, y) are genetated from the i.i.d. distribution, the covariance the between examples should be zero.

$$
\begin{aligned}
\because \quad & \text{Covariance}\left[err(h(x_i),y_i), err(h(x_j),y_j)\right] = 0, \forall i,j \\
\therefore \text{Variance}\,(A+B) &= \text{Variance}\,(A) + \text{Variance}\,(B) + 2\cdot\text{Covariance}\,(A,B) \\
&= \text{Variance}\,(A) + \text{Variance}\,(B) + 0
\end{aligned}
$$

As a result, we should choose $[d]$ as our solution.

P6

Since single data set in a binary classification has N positive examples and N negative examples, we can perform leave-one-out validation by selecting one positive or one negative example into the validation data set.
Once we select a positive example as a validation data set, there are $N-1$ positive examples and N negative examples in the training data set, and the binary classification algorithm $A_{majority}$ will return negative to the training data set for the reason that it predicts the majority class. However, $A_{majority}$ will return positive to the validation data set since there is only one positive example, and vice versa.
As a result, we know that $A_{majority}$ will always return a different classification between the training data set and the validation data set, which means $E_{loocv}(A_{majority}) = 1$.

As a result, we should choose $[d]$ as our solution.

P7

The threshold of the decision stump model can be represented as

$$\theta \in \{-1\} \cup \{\frac{x_i + x_{i+1}}{2}\} : 1 \leq i \leq N-1 \text{ and } x_i \neq x_{i+1}$$

Since x is generated by a uniform distribution in $[-1, +1]$, and $y = \text{sign}(x)$, it's trivial that $E_{in} = 0$ when $\theta = 0$, and the threshold θ with the lowest E_{in} in the decision stump model will be trained in the middle of the smallest positive value and the largest negative value. Therefore, the trained $\theta = \frac{x_j + x_{j+1}}{2}$, where x_j is the largest negative value, and x_{j+1} is the smallest positive value.

If the x_j is selected as the validation data set, then $\theta = \frac{x_{j-1} + x_{j+1}}{2}$, where x_{j-1} is the second largest negative value. Both θ and the x_j are between x_{j-1} and x_{j+1}, then we have

- if $\theta \geq x_j \Rightarrow h(x_j) = -1$, which means the validation data set will be classified correctly.
- if $\theta < x_j \Rightarrow h(x_j) = +1$, which means the validation data set will be classified incorrectly.

And vice versa, if x_{j+1} is selected as the validation data set, then $\theta = \frac{xj + x_{j+2}}{2}$, where x_{j+2} is the second smallest positive value. Both θ and the x_{j+1} are between x_j and x_{j+2}, then we have

- if $\theta \geq x_{j+1} \Rightarrow h(x_{j+1}) = -1$, which means the validation data set will be classified incorrectly.
- if $\theta < x_{j+1} \Rightarrow h(x_{j+1}) = +1$, which means the validation data set will be classified correctly.

Since only x_j and x_{j+1} will be classified incorrectly by the threshold θ, the tightest upper bound of the leave-one-out validation error to the decision stump model is equal to $\max(E_{loocv}) = 2/N$.

As a result, we should choose $[c]$ as our solution.

Support Vector Machine

P8

Since the examples are ordered, the hard-margin SVM without transformation can be considered as a decision stump model when the separation line is orthogonal to the coordinate of the examples. It's trivial that the perfect separation line must be the middle of x_M and x_{M+1}, therefore, the hypothesis can be represented as

$$g_{svm}(x) = \text{sign}(w^T x + b) = 0 = x - \frac{x_M + x_{M+1}}{2}$$

Therefore, the largest margin is equal to $\frac{1}{2}(x_{M+1} - x_M)$

As a result, we should choose $[e]$ as our solution.

P9

Based on the subjections of $\min(w, b) = \frac{1}{2} w^T w$, we have

$$
\begin{aligned}
(w^T x_n + b) &\geq 1 \qquad \text{for } y_n = +1 \\
-(w^T x_n + b) &\geq 1126 \quad \text{for } y_n = -1
\end{aligned}
$$

$$
y = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}^T \begin{bmatrix} 0 & 4 \\ 2 & 0 \\ -1 & 0 \\ 0 & 0 \end{bmatrix} + b = \begin{bmatrix} +1 \\ -1 \\ +1 \\ +1 \end{bmatrix} \Rightarrow \begin{bmatrix} (4w_2 + b) \geq 1 \\ -(2w_1 + b) \leq 1126 \\ (-w_1 + b) \geq 1 \\ (+b) \geq 1 \end{bmatrix}
$$

By drawing all data on the graph and from the subjections above, we know that the ratio of positive margin versus negative margin can represent as $r_m = \frac{y_n=+1}{y_n=-1} = \frac{1}{1126}$, hence, the hyperplane can be represented by the thinnest gap $-(w^T[2,0] + b = 1126)$ and $(w^T[0,0] + b = 1)$, which implies that $w^T x + b = 0 \Rightarrow -\frac{1127}{2}x + 1 = 0$, which means $w = (-\frac{1127}{2}, 0), b = 1$.

As a result, we should choose $[e]$ as our solution.

P10

Since $\alpha = 1, b = 0, h(x) = \text{sign}(\sum_{n=1}^{N} y_n K(x_n, x))$, then we have $E_{in}(h) = \frac{1}{N}\sum_{i=1}^{N}[h(x_i) \neq y_i]$. If $E_{in}(\hat{h}) = 0$, all predictions should be classified correctly, and there is no classification error, which means

$$
\left[\text{sign}\left(\sum_{n=1}^{N} y_n K(x_n, x_i)\right) = y_i\right], \forall i \overset{\text{rewrite}}{\Longrightarrow} \left[\left(\sum_{n=1}^{N} y_n K(x_n, x_i)\right) \cdot y_i > 0\right], \forall i
$$

To sum up the equation above, we have

$$
\begin{aligned}
\sum_{i=1}^{N}\left(y_i \sum_{n=1}^{N} y_n K(x_n, x_i)\right) &= \left[y_1^2 + y_1\left(y_2 K(x_2, x_1) + \cdots + y_N K(x_N, x_1)\right)\right] \\
&+ \left[y_2^2 + y_2\left(y_1 K(x_1, x_2) + \cdots + y_N K(x_N, x_2)\right)\right] \\
&+ \cdots \\
&+ \left[y_N^2 + y_N\left(y_1 K(x_1, x_N) + \cdots + y_{N-1} K(x_{N-1}, x_N)\right)\right] > 0
\end{aligned}
$$

Since $\gamma > 0, \|x_n \neq x_m\| \geq \epsilon, \forall n \neq m$, we have

$$
\begin{aligned}
K(x_n, x_m) = \exp(-\gamma\|x_n - x_m\|^2) \leq \exp(-\gamma\epsilon^2) := \zeta \\
\text{if } n = i, K(x_n, x_i) = \exp(-\gamma\|x_n - x_i\|^2) = \exp(-\gamma \cdot 0) = 0
\end{aligned}
$$

Taking ζ back to the equation as the upper bound, then we get

$$
\begin{aligned}
0 &< \sum_{i=1}^{N}\left(y_i\sum_{n=1}^{N}y_nK(x_n,x_i)\right) < \sum_{i=1}^{N}\left(y_i\sum_{n=1}^{N}y_n\exp(-\gamma\epsilon^2)\right) \\
&= \left[y_1^2+y_2^2+\cdots+y_N^2\right] \\
&+ \underbrace{\left[y_1\left(\underbrace{y_2\exp(-\gamma\epsilon^2)+\cdots+y_N\exp(-\gamma\epsilon^2)}_{N-1}\right)+\cdots+y_N\left(\underbrace{y_1\exp(-\gamma\epsilon^2)+\cdots+y_{N-1}\exp(-\gamma\epsilon^2)}_{N-1}\right)\right.}_{N}
\end{aligned}
$$

And we know that

$$
\begin{aligned}
&\forall n=m,\, y_m=(+1)\vee(-1)\Rightarrow y_i^2=1,\forall i \\
&\forall n\neq m,\, -1<y_ny_m<1 \Rightarrow \text{assumed that all } y_ny_m=-1
\end{aligned}
$$

Then, we have

$$
\begin{aligned}
0 &< \sum_{i=1}^{N}\left(y_i\sum_{n=1}^{N}y_n\exp(-\gamma\epsilon^2)\right) \\
&< \sum_{i=1}^{N}y_i^2+\sum_{i=1}^{N}(-1)\times(N-1)\exp(-\gamma\epsilon^2) \\
&= N+N\times[-(N-1)\exp(-\gamma\epsilon^2)] \\
&= N-N(N-1)\exp(-\gamma\epsilon^2) \\
1 &> (N-1)\exp(-\gamma\epsilon^2) \\
\frac{1}{N-1} &> \exp(-\gamma\epsilon^2) \\
\ln(N-1) &< \gamma\epsilon^2 \\
\gamma &> \frac{\ln(N-1)}{\epsilon^2}
\end{aligned}
$$

As a result, we should choose $[d]$ as our solution.

P11

Applying the feature transform ϕ to the Gaussian kernel, then we have

$$
\begin{aligned}
\|\phi(x)-\phi(x^{'})\|^2 &= <\phi(x)-\phi(x^{'}),\phi(x)-\phi(x^{'})> \\
&= <\phi(x),\phi(x)>-2<\phi(x),\phi(x^{'})>+<\phi(x^{'}),\phi(x^{'})> \\
&= \phi(x)^T\phi(x)-\phi(x)^T\phi(x^{'})-\phi(x^{'})^T\phi(x)+\phi(x^{'})^T\phi(x^{'}) \\
&= K(x,x)-2K(x,x^{'})+K(x^{'},x^{'}) \\
&= 2-2\exp(-\gamma\|x-x^{'}\|^2) \\
&\because \exp(-\gamma\|x-x^{'}\|^2)\in(0,1] \\
\therefore 2 &> \|\phi(x)-\phi(x^{'})\|^2 \\
\sqrt{2}\approx 1.5 &> \|\phi(x)-\phi(x^{'})\|
\end{aligned}
$$

Therefore, the tightest upper bound for the distane in the $\mathcal{Z}$-space is 1.5.

As a result, we should choose $[d]$ as our solution.

Experiments with Regularized Logistic Regression

P12

Since L2-regularized logistic regression `-s 0` in liblinear solves

$$w = \min_w \frac{1}{2} w^T w + C \sum_{i=1}^{N} \log(1 + \exp(-y_i w^T x_i))$$

and we solve

$$w_\lambda = \operatorname*{argmin}_w \frac{\lambda}{N} \|w\|^2 + \frac{1}{N} \sum_{n=1}^{N} \ln(1 + \exp(-y_n w^T \Phi_4(x_n)))$$

Then we know that

$$\frac{1}{2} = \frac{\lambda}{N}, C = \frac{1}{N} \Rightarrow C = \frac{1}{2\lambda}$$

The parameter `-c cost : set the parameter C (default 1)` should be set to $C = \frac{1}{2\lambda}$.

```python
import numpy as np
from itertools import combinations_with_replacement
from liblinear.liblinearutil import *

def transformation(args, x):
    x_origin = x[:, 1:] # remove x_{n=0}=1
    transformed_x = x.copy()
    combination_tuples = list()
    iterable = list(i for i in range(1, args['dimension']))
    for r in range(1, args['Q']):
        combination_tuples.extend(list(combinations_with_replacement(iterable, r+1)))

    for combination in range(len(combination_tuples)):
        x_temp = 1
        for j in combination_tuples[combination]:
            x_temp = x_origin[:,j-1].reshape(-1, 1) * x_temp
        transformed_x = np.hstack((
            transformed_x, x_temp
            ))
    return transformed_x
```

```
def read_file(args, filename):
    data = np.loadtxt(filename, dtype=float)
    x = data[:,:-1]
    x = np.c_[np.ones(len(x)), x] # x_{n=0}=1
    y = data[:,-1]
    args['dimension'] = x.shape[1]
    return x, y
```

```
def main(args):
    x_train, y_train = read_file(args, args['filename_train'])
    x_test, y_test = read_file(args, args['filename_test'])
    x_train = transformation(args, x_train)
    x_test = transformation(args, x_test)

    E_out = list()
    for lamb in args['lambda']:
        C = 1/(2*lamb)
        prob = problem(y_train, x_train)
        param = parameter('-s 0 -c {} -e 0.000001 -q'.format(C))
        m = train(prob, param)
        p_label, p_acc, p_val = predict(y_test, x_test, m)
        E_out.append(round(np.mean(y_test != p_label), 6))

    min_E_out = min(E_out)
    min_E_out_index = [i for i, v in enumerate(E_out) if v == min_E_out]
    print("E_out: ", E_out)
    print("min E_out: ", min(E_out))
    print("Choice: ", chr(97+max(min_E_out_index)))
```

```
if __name__ == '__main__':
    args = {
        'dimension': 0,
        'filename_test': "hw4_test.dat",
        'filename_train': "hw4_train.dat",
        'lambda': [10**(-6), 10**(-3), 10**(0), 10**3, 10**6],
        'Q': 4,
    }

    main(args)
```

```
NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numl
  def csr_to_problem_jit(l, x_val, x_ind, x_rowptr, prob_val, prob_ind, prob_rowptr):
Accuracy = 77.4% (387/500) (classification)
Accuracy = 82.2% (411/500) (classification)
Accuracy = 84.6% (423/500) (classification)
Accuracy = 85.8% (429/500) (classification)
Accuracy = 81.2% (406/500) (classification)
E_out:  [0.226, 0.178, 0.154, 0.142, 0.188]
min E_out:  0.142
Choice:  d
```

Therefore, the best $\log_{10}(\lambda^*) = 3$

As a result, we should choose $[d]$ as our solution.

P13

We use the same code above, then update the `main` function. The updated functions show below.

```
def main(args):
    x_train, y_train = read_file(args, args['filename_train'])
    x_test, y_test = read_file(args, args['filename_test'])
    x_train = transformation(args, x_train)
    x_test = transformation(args, x_test)

    E_in = list()
    for lamb in args['lambda']:
        C = 1/(2*lamb)
        prob = problem(y_train, x_train)
        param = parameter('-s 0 -c {} -e 0.000001 -q'.format(C))
        m = train(prob, param)
        p_label, p_acc, p_val = predict(y_train, x_train, m)
        E_in.append(round(np.mean(y_train != p_label), 6))

    min_E_in = min(E_in)
    min_E_in_index = [i for i, v in enumerate(E_in) if v == min_E_in]
    print("E_in: ", E_in)
    print("min E_in: ", min(E_in))
    print("Choice: ", chr(97+max(min_E_in_index)))
```

```
NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numl
  def csr_to_problem_jit(l, x_val, x_ind, x_rowptr, prob_val, prob_ind, prob_rowptr):
Accuracy = 100% (200/200) (classification)
Accuracy = 100% (200/200) (classification)
Accuracy = 100% (200/200) (classification)
Accuracy = 96% (192/200) (classification)
Accuracy = 76% (152/200) (classification)
E_in:  [0.0, 0.0, 0.0, 0.04, 0.24]
min E_in:  0.0
Choice:  c
```

Since $E_{in}(w_{\log_{10}(\lambda^*)=-6}) = E_{in}(w_{\log_{10}(\lambda^*)=-3}) = E_{in}(w_{\log_{10}(\lambda^*)=0}) = 0$, by selecting the largest λ, therefore, the best $\log_{10}(\lambda^*) = 0$

As a result, we should choose $[c]$ as our solution.

P14

We use the same code above, then add a `train_test_split` function and update the `main` function and the parameter of args. The updated functions show below.

```
import numpy as np
import random
from itertools import combinations_with_replacement
from liblinear.liblinearutil import *

def train_test_split(args, x, y):
    rng = np.random.default_rng()
    idx = np.arange(len(x))
    rng.shuffle(idx)
    x_train = x[idx[:args['split']]]
    y_train = y[idx[:args['split']]]
    x_test = x[idx[args['split']:]]
    y_test = y[idx[args['split']:]]
    return x_train, y_train, x_test, y_test
```

```
def main(args):
    x_train, y_train = read_file(args, args['filename_train'])
    x_train = transformation(args, x_train)
    lamb_list = list()
    E_val_list = list()
    for i in range(args['repeat_time']):
        x_train_split, y_train_split, x_test_split, y_test_split = train_test_split(a
        args['size'] = x_train_split.shape[0]

        E_val = list()
        for lamb in args['lambda']:
            C = 1/(2*lamb)
            prob = problem(y_train_split, x_train_split)
            param = parameter('-s 0 -c {} -e 0.000001 -q'.format(C))
            m = train(prob, param)
            p_label, p_acc, p_val = predict(y_test_split, x_test_split, m)
            E_val.append(round(np.mean(y_test_split != p_label), 6))

        min_E_val = min(E_val)
        min_E_val_index = [i for i, v in enumerate(E_val) if v == min_E_val]
        lamb_list.append(max(min_E_val_index))
        E_val_list.append(min_E_val)

    best_lambda = args['lambda'][max(set(lamb_list), key = lamb_list.count)]
    print("best_lambda: ", best_lambda)
```

```
if __name__ == '__main__':
    args = {
        'dimension': 0,
        'filename_test': "hw4_test.dat",
        'filename_train': "hw4_train.dat",
        'lambda': [10**(-6), 10**(-3), 10**(0), 10**3, 10**6],
        'Q': 4,
        'repeat_time': 256,
        'seed': 1126,
        'size': 0,
        'split': 120
    }

    main(args)
```

```
best_lambda:  1000
```

As a result, we should choose $[d]$ as our solution.

P15

We use the same code above, then update the `main` function. The updated functions show below.

```
def main(args):
    x_train, y_train = read_file(args, args['filename_train'])
    x_train = transformation(args, x_train)
    x_test, y_test = read_file(args, args['filename_test'])
    x_test = transformation(args, x_test)
    lamb_list = list()
    E_val_list = list()
    E_out_list = list()
    for i in range(args['repeat_time']):
        x_train_split, y_train_split, x_test_split, y_test_split = train_test_split(a
        args['size'] = x_train_split.shape[0]

        model_list = list()
        E_val = list()
        for lamb in args['lambda']:
            C = 1/(2*lamb)
            prob = problem(y_train_split, x_train_split)
            param = parameter('-s 0 -c {} -e 0.000001 -q'.format(C))
            m = train(prob, param)
            p_label, p_acc, p_val = predict(y_test_split, x_test_split, m)
            E_val.append(round(np.mean(y_test_split != p_label), 6))
            model_list.append(m)

        min_E_val = min(E_val)
        min_E_val_index = [i for i, v in enumerate(E_val) if v == min_E_val]
        lamb_list.append(max(min_E_val_index))
        E_val_list.append(min_E_val)
        # predict test dataset
        p_label, p_acc, p_val = predict(y_test, x_test, model_list[max(min_E_val_inde
        E_out_list.append(round(np.mean(y_test != p_label), 6))

    best_lambda = args['lambda'][max(set(lamb_list), key = lamb_list.count)]
    print("best_lambda: ", best_lambda)
    print("E_out: ", np.mean(E_out_list))
```

```
best_lambda:  1000
E_out:  0.169109375
```

As a result, we should choose $[c]$ as our solution.

P16

We use the same code above, then update the `main` function. The updated functions show below.

```
def main(args):
    x_train, y_train = read_file(args, args['filename_train'])
    x_train = transformation(args, x_train)
    x_test, y_test = read_file(args, args['filename_test'])
    x_test = transformation(args, x_test)
    lamb_list = list()
    E_val_list = list()
    E_out_list = list()
    for i in range(args['repeat_time']):
        x_train_split, y_train_split, x_test_split, y_test_split = train_test_split(a
        args['size'] = x_train_split.shape[0]

        model_list = list()
        E_val = list()
        for lamb in args['lambda']:
            C = 1/(2*lamb)
            prob = problem(y_train_split, x_train_split)
            param = parameter('-s 0 -c {} -e 0.000001 -q'.format(C))
            m = train(prob, param)
            p_label, p_acc, p_val = predict(y_test_split, x_test_split, m)
            E_val.append(round(np.mean(y_test_split != p_label), 6))
            model_list.append(m)

        min_E_val = min(E_val)
        min_E_val_index = [i for i, v in enumerate(E_val) if v == min_E_val]
        lamb_list.append(max(min_E_val_index))
        E_val_list.append(min_E_val)

        # rebuild the model from raw training dataset
        prob = problem(y_train, x_train)
        param = parameter('-s 0 -c {} -e 0.000001 -q'.format(1/(2*(args['lambda'][max
        m = train(prob, param)
        # predict from raw testing dataset
        p_label, p_acc, p_val = predict(y_test, x_test, m)
        E_out_list.append(round(np.mean(y_test != p_label), 6))

    best_lambda = args['lambda'][max(set(lamb_list), key = lamb_list.count)]
    print("best_lambda: ", best_lambda)
    print("E_out: ", np.mean(E_out_list))
```

```
best_lambda:  1000
E_out:  0.15015624999999996
```

As a result, we should choose $[b]$ as our solution.

P17

We use the same code above, then add a `KFold` function and update the `main` function and the parameter of args. The updated functions show below.

```
def KFold(args, x, y):
    rng = np.random.default_rng()
    idx = np.arange(len(x))
    rng.shuffle(idx)
    x_split = np.array([x[idx[i*args['fold_size']:(i+1)*args['fold_size']]] for i in r
    y_split = np.array([y[idx[i*args['fold_size']:(i+1)*args['fold_size']]] for i in r
    return x_split, y_split
```

```
def main(args):
    x_train, y_train = read_file(args, args['filename_train'])
    x_train = transformation(args, x_train)
    x_test, y_test = read_file(args, args['filename_test'])
    x_test = transformation(args, x_test)
    args['size'] = x_train.shape[0]
    args['fold_size'] = int(args['size'] / args['fold'])

    E_cv_list = list()
    for i in range(args['repeat_time']):
        E_cv = list()
        x_split, y_split = KFold(args, x_train, y_train)

        for lamb in args['lambda']:
            C = 1/(2*lamb)
            E_val = list()
            for fold in range(args['fold']):
                x_valid_fold = x_split[fold]
                y_valid_fold = y_split[fold]
                x_train_fold = np.vstack(x_split[j] for j in range(args['fold']) if j
                y_train_fold = np.hstack(y_split[j] for j in range(args['fold']) if j

                prob = problem(y_train_fold, x_train_fold)
                param = parameter('-s 0 -c {} -e 0.000001 -q'.format(C))
                m = train(prob, param)
                p_label, p_acc, p_val = predict(y_valid_fold, x_valid_fold, m)
                E_val.append(round(np.mean(y_valid_fold != p_label), 6))
            E_cv.append(np.mean(E_val))
        E_cv_list.append(min(E_cv))

    print("E_cv: ", np.mean(E_cv_list))
```

```
if __name__ == '__main__':
    args = {
        'dimension': 0,
        'filename_test': "hw4_test.dat",
        'filename_train': "hw4_train.dat",
        'fold': 5,
        'fold_size': 0, # size of each fold
        'lambda': [10**(-6), 10**(-3), 10**(0), 10**3, 10**6],
        'Q': 4,
        'repeat_time': 256,
        'seed': 1126,
        'size': 0,
        'split': 120
    }

    main(args)
```

```
E_cv:  0.12927734375
```

As a result, we should choose $[a]$ as our solution.

P18

Since L1-regularized logistic regression `-s 6` in liblinear solves

$$w = \min_w \sum |w_j| + C \sum_{i=1}^{N} \log(1 + \exp(-y_i w^T x_i))$$

and we solve

$$w_\lambda = \operatorname*{argmin}_w \frac{\lambda}{N} \|w\|_1 + \frac{1}{N} \sum_{n=1}^{N} \ln(1 + \exp(-y_n w^T \Phi_4(x_n))$$

Then we know that

$$1 = \frac{\lambda}{N}, C = \frac{1}{N} \Rightarrow C = \frac{1}{\lambda}$$

The parameter `-c cost : set the parameter C (default 1)` should be set to $C = \frac{1}{\lambda}$.

We use the same code from `P12`, then update parameter `C` and `param` in the `main` function. The updated functions show below.

```
C = 1/(lamb)
param = parameter('-s 6 -c {} -e 0.000001 -q'.format(C))
```

```
NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numl
  def csr_to_problem_jit(l, x_val, x_ind, x_rowptr, prob_val, prob_ind, prob_rowptr):
Accuracy = 76.8% (384/500) (classification)
Accuracy = 83.6% (418/500) (classification)
Accuracy = 84.6% (423/500) (classification)
Accuracy = 68% (340/500) (classification)
Accuracy = 49.2% (246/500) (classification)
E_out:  [0.232, 0.164, 0.154, 0.32, 0.508]
min E_out:  0.154
Choice:  c
```

As a result, we should choose $[c]$ as our solution.

P19

We use the same code above, then update the `main` function. The updated functions show below.

```
def main(args):
    x_train, y_train = read_file(args, args['filename_train'])
    x_test, y_test = read_file(args, args['filename_test'])
    x_train = transformation(args, x_train)
    x_test = transformation(args, x_test)
    args['dimension'] = x_train.shape[1]

    model_list = list()
    E_out = list()
    for lamb in args['lambda']:
        C = 1/(lamb)
        prob = problem(y_train, x_train)
        param = parameter('-s 6 -c {} -e 0.000001 -q'.format(C))
        m = train(prob, param)
        model_list.append(np.array([m.w[i] for i in range(args['dimension']) if np.abs
        p_label, p_acc, p_val = predict(y_test, x_test, m)
        E_out.append(round(np.mean(y_test != p_label), 6))

    min_E_out = min(E_out)
    min_E_out_index = [i for i, v in enumerate(E_out) if v == min_E_out]
    print("|m.w[i] <= 10**(-6)|: ", len(model_list[min_E_out_index[0]]))
```

```
NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numl
  def csr_to_problem_jit(l, x_val, x_ind, x_rowptr, prob_val, prob_ind, prob_rowptr):
Accuracy = 76.8% (384/500) (classification)
Accuracy = 83.6% (418/500) (classification)
Accuracy = 84.6% (423/500) (classification)
Accuracy = 68% (340/500) (classification)
Accuracy = 49.2% (246/500) (classification)
|m.w[i] <= 10**(-6)|:  960
```

As a result, we should choose $[e]$ as our solution.

P20

We use the same code above, then update parameter `C` and `param` in the `main` function. The updated functions show below.

```
C = 1/(2*lamb)
param = parameter('-s 0 -c {} -e 0.000001 -q'.format(C))
```

```
NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numl
  def csr_to_problem_jit(l, x_val, x_ind, x_rowptr, prob_val, prob_ind, prob_rowptr):
Accuracy = 77.4% (387/500) (classification)
Accuracy = 82.2% (411/500) (classification)
Accuracy = 84.6% (423/500) (classification)
Accuracy = 85.8% (429/500) (classification)
Accuracy = 81.2% (406/500) (classification)
|m.w[i] <= 10**(-6)|:  1
```

As a result, we should choose $[a]$ as our solution.