HTML

Homework 2

B07502166 魏子翔

1. First, $w_0 = 0$ if the line passes the lucky point.
   Second, if the perceptrons always passes the lucky point, then we can normalize the data points to the unit circle around the lucky point, so the only thing to be decided is the slope of the line, which can be determined by choosing one pair of the neighbor points and letting the line pass between the points.
   So there's N pairs of points to choose, and two possible sign with each choice, then the growth function is $2N$.

2. If we only have 1126 perceptrons, then the VC dimension will be limited by $2^{d_{VC}} \leq 1126$, $d_{VC} \leq \log_2 1126$

3. There's left and right boundary for each of the two intervals, so there's 4 variables.
   3D perceptrons is d+1, which is 4.
   There's only one variable, so is 1.
   We can decide the x and y location of the lower left point, then the x location of the lower right point, finally the y location of the upper right point, so it's 2+1+1=4.
   We can decide the x and y location of the lower left point, then the x location of the lower right point, then the y location of the upper right point, then the last point is determined, so it's 2+1+1=4.

4. Consider $x_1, x_2, x_3, ..., x_{2M}$ with $x_1^T x_1 \leq x_2^T x_2 \leq x_3^T x_3, ... \leq x_{2M}^T x_{2M}$.
   For the results of $h(x_i), h(x_{i+1})$, if there's two consecutive +1 or -1, then we can simply put them into the same interval to increase the freedom of the other x_j, so the (+1, -1, ..., +1, -1) sequence or its inverse are the two most difficult situation.
   For (+1, -1, ..., +1, -1) sequence, we can let $x_1^T x_1$ be in the interval of a_1, b_1, $x_2^T x_2$ in b_1, a_2, $x_3^T x_3$ in a_2, b_2, ..., $x_{2M}^T x_{2M} > b_m$ to achieve the sequence. Similarly, we can set (-1, +1, ..., -1, +1) by letting $x_1^T x_1 < a_1$, $x_2^T x_2$ in a_1, b_1, $x_3^T x_3$ in b_1, a_2, ..., $x_{2M}^T x_{2M}$ in a_m, b_m to achieve that sequence. Hence, $2M$ is not the break point.
   Consider $x_1, x_2, x_3, ..., x_{2M+1}$ with $x_1^T x_1 \leq x_2^T x_2 \leq x_3^T x_3, ... \leq x_{2M+1}^T x_{2M+1}$.
   For the sequence (+1, -1, ..., +1, -1, +1), as the discussion above, we can set the first $2M$ part one by one in the interval of a_1 to b_{2M}, with the -1 in the $2M$ place be larger than b_{2M}, so for the $2M+1$ place element, if we set it to be +1, then we will fail because there's no legal place to place it, which implies that $2M+1$ is a break point.
   Therefore, the VC dimension is $2M$.

5. To be a necessary condition of a $\leq$ sentence, it should be the necessary condition for both $=$ and $<$ sentences. So, the upper four are true only if $d_{vc}(H) =$ or $< d$, but not both, which implies that they are not the necessary condition for the sentence.
   Similar to the example in class, we can find that the final condition is the correct one, also, the "some" version of it will also be true, so the answer is 2.

6. To get the minimal E_{in}, we can set$E'_{in} = 0$ and find the corresponding w.

$$E'_{in}(w) = \frac{2}{N}\sum_{n=1}^{N}(wx_n - y_n) = \frac{2}{N}\left(\sum_{n=1}^{N} wx_n - \sum_{n=1}^{N} y_n\right) = 0$$

$$\Rightarrow \ w\sum_{n=1}^{N} x_n = \sum_{n=1}^{N} y_n$$

$$\Rightarrow \ w = \frac{\sum_{n=1}^{N} y_n}{\sum_{n=1}^{N} x_n}$$

7. a. $L(\lambda) = \prod_{i=1}^{N} \frac{e^{-\lambda}\lambda^{x_i}}{x_i!}$

$$\ln L(\lambda) = \sum_{i=1}^{N} -\lambda + x_i \ln\lambda - \ln x_i!$$

$$\frac{d\ln L(\lambda)}{d\lambda} = -N + \frac{1}{\lambda}\sum_{i=1}^{N} x_i = 0$$

$$\lambda = \frac{1}{N}\sum_{i=1}^{N} x_i = \bar{x}$$

b. $L(\mu) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x_i-\mu)^2}$

$$\ln L(\mu) = \sum_{i=1}^{N} - \ln\sqrt{2\pi} - \tfrac{1}{2}(x_i - \mu)^2$$

$$\frac{d\ln L(\mu)}{d\mu} = \sum_{i=1}^{N}(-x_i + \mu) = 0$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i = \bar{x}$$

c. $L(\mu) = \prod_{i=1}^{N} \frac{1}{2} e^{-|x_i-\mu|}$

$$\ln L(\mu) = \sum_{i=1}^{N} - \ln 2 - |x_i - \mu|$$

$$\frac{d\ln L(\mu)}{d\mu} = \sum_{i=1}^{N} sign(x_i - \mu) = 0$$

We can set $\mu =$ median of x_n to get 0.

If N is odd, then median of x_n, which can let those x_i smaller than μ be negative, and the larger ones be positive, with the median itself be 0. So the summation of them will be $-1 \times \frac{N-1}{2} + 1 \times \frac{N-1}{2} + 0 = 0$.

If N is even, then median of x_n will give the average of $x_{N/2}$ and $x_{N/2+1}$ if we sort it first, which let the summation be $-1 \times \frac{N}{2} + 1 \times \frac{N}{2} = 0$.

d. $L(\theta) = \prod_{i=1}^{N} (1-\theta)^{x_i - 1}\theta$

$$\ln L(\theta) = \sum_{i=1}^{N}(x_i - 1)\ln(1-\theta) + \ln\theta$$

$$\frac{d\ln L(\theta)}{d\theta} = \sum_{i=1}^{N}(x_i - 1)\tfrac{-1}{1-\theta} + \tfrac{1}{\theta} = \sum_{i=1}^{N} \tfrac{1-\theta-x_i\theta+\theta}{\theta(1-\theta)} = \sum_{i=1}^{N} \tfrac{1-x_i\theta}{\theta(1-\theta)} = 0$$

$$\theta = \frac{N}{\sum_{i=1}^{N} x_i} = \tfrac{1}{\bar{x}}$$

8. $E_{in} = \dfrac{1}{N}\sum_{n=1}^{N} \ln \dfrac{2+2|y_n w^T x_n|}{1+y_n w^T x_n + |y_n w^T x_n|}$

$$\nabla \tilde{E}_{in}(w) = \frac{1}{N}\sum_{n=1}^{N} \frac{1+y_n w^T x_n + |y_n w^T x_n|}{2+2|y_n w^T x_n|} \times$$
$$\frac{(1+y_n w^T x_n + |y_n w^T x_n|)2sign(w)(y_n x_n) - (2+2|y_n w^T x_n|)(y_n x_n + sign(w)(y_n x_n))}{(1+y_n w^T x_n + |y_n w^T x_n|)^2}$$
$$= \frac{1}{N}\sum_{n=1}^{N} \frac{2sign(w) y_n w^T x_n (y_n x_n) - 2y_n x_n - 2|y_n w^T x_n|(y_n x_n)}{(2+2|y_n w^T x_n|)(1+y_n w^T x_n + |y_n w^T x_n|)}$$
$$= -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n x_n}{(1+|y_n w^T x_n|)(1+y_n w^T x_n + |y_n w^T x_n|)}$$

9. $\nabla E_{in}(w) = \frac{1}{N}(2X^T X w - 2b)$
$\nabla^2 E_{in}(w) = \frac{2}{N} X^T X$

10. Follow the result in the previous problem and in the class, we can calculate u as follow.

$$u = -(\nabla^2 E_{in}(w_0))^{-1} \nabla E_{in}(w_0) = -(\frac{2}{N}X^T X)^{-1}\frac{2}{N}(X^T X w_0 - X^T y)$$
$$= -(X^T X)^{-1}(X^T X w_0 - X^T y) = -(w_0 - (X^T X)^{-1} X^T y) = (X^T X)^{-1} X^T y - w_0$$

$w_1 = w_0 + u = w_0 + (X^T X)^{-1} X^T y - w_0 = (X^T X)^{-1} X^T y$
$\nabla E_{in}(w_1) = (X^T X)^{-1} X^T y - w_1 = (X^T X)^{-1} X^T y - (X^T X)^{-1} X^T y = 0$ So only need 1 iteration.

11. $4(2N)^2 \exp(-\frac{1}{8} 0.05^2 N) \leq 0.1 \;\Rightarrow\; N \sim 89000 \;\Rightarrow\;$ 100000 is enough.

12. If $\tau = 0$, then $\min(|\theta|, 0.5)$ is the range of error, while $1 - \min(|\theta|, 0.5)$ is the range of correct. So if we consider τ, then the probability of the points that no flip are $1-\tau$, with the flip ones are τ. Therefore, $\min(|\theta|, 0.5)(1-\tau)$ is the error of non-flip ones, plus $(1 - \min(|\theta|, 0.5))\tau$, which equals to $\min(|\theta|, 0.5)(1-2\tau) + \tau$.

13~20
```python
import numpy as np

def read_data(filename):
    X, y = [], []
    with open(filename, "r") as f:
        for line in f:
            line = line[:-1]
            line = list(map(float, line.split()))
            X.append(line[:-1])
            y.append(int(line[-1]))
    return np.array(X), np.array(y)

def sign(x):
    if np.sign(x) == 1:
        return 1
    else:
        return -1

def one_dim_solver(N, tau=0, gen_data=True, X=None, Y=None):
    err_in_min_list = []
    it = 10000 if gen_data else 1
    for i in range(it):
        err_in_list = [0]*N
        theta_list = [-100,]
        check_list = [0]*N
        if gen_data:
            np.random.seed(i)
            X = np.random.random(N) - 0.5
            X = np.sort(X)
            Y = []
            for i in range(N):
                if np.random.random(1) < tau:
                    Y.append(-sign(X[i]))
                else:
                    Y.append(sign(X[i]))
        else:
            idx = X.argsort()
            X = X[idx]
            Y = Y[idx]

        for i in range(N):
            if sign(X[i]-theta_list[0]) != Y[i]:
                err_in_list[0] += 1
            else:
                check_list[i] = 1
        for i in range(1, N):
            theta_list.append((X[i]+X[i-1])/2)
            err_in_list[i] = err_in_list[i-1]
            if (sign(X[i-1]-theta_list[i]) == Y[i-1]) and check_list[i-1] == 0:
                err_in_list[i] -= 1
            elif (sign(X[i-1]-theta_list[i]) != Y[i-1]) and check_list[i-1] == 1:
                err_in_list[i] += 1

        err_in_min = err_in_list[0]
        theta_min = theta_list[0]
        for i in range(1, N):
            if (err_in_list[i] == err_in_min and theta_list[i] <
                theta_min) or err_in_list[i] < err_in_min:
                err_in_min = err_in_list[i]
                theta_min = theta_list[i]

        err_in_list = N - np.array(err_in_list)
        err_in_min_1 = err_in_list[0]
        theta_min_1 = theta_list[0]
        for i in range(1, N):
            if (err_in_list[i] == err_in_min_1 and theta_list[i] <
                -1*theta_min_1) or err_in_list[i] < err_in_min_1:
                err_in_min_1 = err_in_list[i]
                theta_min_1 = theta_list[i]

        if (err_in_min_1 < err_in_min) or (err_in_min_1 ==
            err_in_min and -1*theta_min_1 < theta_min):
            if not gen_data:
                return -1, theta_min_1, err_in_min_1/N
            err_in_min_list.append(
                1-(min(abs(theta_min_1), 0.5)*(1-2*tau)+tau)-err_in_min_1/N)

        else:
            if not gen_data:
                return 1, theta_min, err_in_min/N
            err_in_min_list.append(
                min(abs(theta_min), 0.5)*(1-2*tau)+tau-err_in_min/N)

    print(np.mean(err_in_min_list))

def multi_dim_solver(X, Y, c):
    err_in_min = 1 if c else 0
    for i in range(X.shape[1]):
        s, theta, err_in = one_dim_solver(
```

```python
            X.shape[0], gen_data=False, X=X[:, i], Y=Y)
        if c:
            if err_in < err_in_min:
                err_in_min = err_in
                best = [i, s, theta]
        else:
            if err_in > err_in_min:
                err_in_min = err_in
                best = [i, s, theta]
    return best, err_in_min

def multi_dim_tester(X, Y, best, err_in):
    X = X[:, best[0]]
    err_out = 0
    for i in range(len(X)):
        if best[1]*sign(X[i]-best[2]) != Y[i]:
            err_out += 1
    return err_out/len(X)

if __name__ == "__main__":
    one_dim_solver(2, 0)        #P13
    one_dim_solver(128, 0)      #P14
    one_dim_solver(2, 0.2)      #P15
    one_dim_solver(128, 0.2)    #P16
    X, Y = read_data("hw2_train.dat")
    X_test, Y_test = read_data("hw2_test.dat")
    best, err_in_b = multi_dim_solver(X, Y, True)
    print(err_in_b)             #P17
    err_out_b = multi_dim_tester(X_test, Y_test, best, err_in_b)
    print(err_out_b)            #P18
    best, err_in_w = multi_dim_solver(X, Y, False)
    err_out_w = multi_dim_tester(X_test, Y_test, best, err_in_w)
    print(err_in_w-err_in_b)    #P19
    print(err_out_w-err_out_b) #P20
```