HTML HW4

B12901022 廖冠豪

5

By definition of the Hessian we have

$$A_E(\mathbf{w})_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} E_{in}(\mathbf{w})$$

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_i} = \frac{1}{N} \sum_{n=1}^{N} \frac{\exp(-y_n \mathbf{w}^T \mathbf{x}_n)(-y_n x_{n_i})}{1 + \exp(-y_n w^T \mathbf{x}_n)}$$

$$\begin{aligned}
\frac{\partial}{\partial w_j}\left(\frac{\partial E_{in}(\mathbf{w})}{\partial w_i}\right) &= \frac{1}{N} \sum_{n=1}^{N} \frac{(\exp(-y_n \mathbf{w}^T \mathbf{x}_n)(\mathbf{x}_{n_i} \mathbf{x}_{n_j}))(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)) - \exp(-y_n \mathbf{w}^T \mathbf{x}_n)^2 (x_{n_i} x_{n_j})}{(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n))^2} \\
&= \frac{1}{N} \sum_{n=1}^{N} \frac{\exp(-y_n \mathbf{w}^T \mathbf{x}_n)}{(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n))^2} x_{n_i} x_{n_j} \\
&= \frac{1}{N} \sum_{n=1}^{N} \frac{1}{1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)} \frac{1}{1 + \exp(y_n \mathbf{x}^T \mathbf{x}_n)} x_{n_i} x_{n_j} \\
&= \frac{1}{N} \sum_{n=1}^{N} h_t(\mathbf{x}_n) h_t(-\mathbf{x}_n) x_{n_i} x_{n_j}
\end{aligned}$$

(x_{n_i} is the i-th element of the $\mathbf{x}_n$, and is the same as X_{ji}) For a N by d matrix X and a N by N diagonal matrix D, we have

$$(X^T D X)_{ij} = \sum_{n=1}^{N} D_{nn} X_{ni} X_{nj}$$

By comparing the expressions, we can obtain

$$D_{ij} = \begin{cases} 0 & i \neq j \\ \frac{1}{N} h_t(\mathbf{x}_n) h_t(-\mathbf{x}_n) & i = j \end{cases}$$

6

In this problem, we use w_{j_i} to denote the i-th element of $\mathbf{w}_j$, which is the same as W_{ij}
Since E_{in} is minmized with SGD, we know that

$$\mathrm{V}_{ij} = -\frac{\partial}{\partial w_{j_i}} \text{err}(W, \mathbf{x}, y)$$

$$\begin{aligned}
\frac{\partial}{\partial w_{j_i}} \text{err}(W, \mathbf{x}, y) &= -\frac{1}{h_y(\mathbf{x})} \frac{\partial h_y(\mathbf{x})}{\partial w_{j_i}} \\
&= -\frac{1}{h_y(\mathbf{x})} \frac{[\![y = i]\!] \exp\left(\mathbf{w}_y^T \mathbf{x}\right)(\sum_{k=1}^K \exp(\mathbf{w}_k^T \mathbf{x}))x_j - \exp\left(\mathbf{w}_y^T \mathbf{x}\right) \exp\left(\mathbf{w}_i^T \mathbf{x}\right) x_j}{\left(\sum_{k=1}^K \exp\left(\mathbf{w}_k^T \mathbf{x}\right)\right)^2} \\
&= -\frac{1}{h_y(\mathbf{x})} \frac{([\![y = i]\!] - \exp\left(\mathbf{w}_i^T \mathbf{x}\right)) \exp\left(\mathbf{w}_y^T \mathbf{x}\right) x_j}{(\sum_{k=1}^K \exp\left(\mathbf{w}_k^T \mathbf{x}\right))^2} \\
&= -([\![y = i]\!] - \frac{\exp\left(\mathbf{w}_i^T \mathbf{x}\right)}{\sum_{k=1}^K \exp\left(\mathbf{w}_k^T \mathbf{x}\right)}) x_j \\
&= (h_i(\mathbf{x}) - [\![y = i]\!]) x_j
\end{aligned}$$

Notice that

$$(\mathbf{x}_n \cdot \mathbf{u}^T)_{ij} = x_j u_i$$

Hence by comparing the expressions, $\mathbf{u}$ is given by

$$u_i = -h_i(\mathbf{x}_n) + [\![y_n = i]\!]$$

7

For MLR , we know that $\sum_{n=1}^{N} \mathrm{err}(\mathrm{W}, \mathbf{x}, y)$ is minimized at its optimal solution, $(\mathbf{w}_1^\star, \mathbf{w}_2^\star)$. Also

$$\begin{aligned}
\sum_{n=1}^{N} \mathrm{err}(\mathrm{W}, \mathbf{w}, y) &= \sum_{n=1}^{N} \ln\left(\frac{\exp\left(\mathbf{w}_{y_n}^T \mathbf{x}_n\right)}{\exp(\mathbf{w}_1^T \mathbf{x}_n) + \exp(\mathbf{w}_2^T \mathbf{x}_n)}\right) \\
&= \begin{cases} -\sum_{n=1}^{N} \ln\left(1 + \exp((\mathbf{w}_2^T - \mathbf{w}_1^T)\mathbf{x}_n)\right) & \text{if } y_n = 1, \\ -\sum_{n=1}^{N} \ln\left(1 + \exp((\mathbf{w}_1^T - \mathbf{w}_2^T)\mathbf{x}_n)\right) & \text{if } y_n = 2 \end{cases} \\
&= -\sum_{n=1}^{N} \ln\left(1 + \exp(-y_n'(\mathbf{w}_2 - \mathbf{w}_1)^T \mathbf{x}_n)\right)
\end{aligned}$$

For logistic regression

$$E_{\text{in}}(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + \exp(-y_n' \mathbf{w}^T \mathbf{x}_n)\right)$$

By comparing the expressions, we see that $E_{\text{in}}(\mathbf{w})$ is minimized with

$$\mathbf{w}_{lr} = \mathbf{w}_2^\star - \mathbf{w}_1^\star$$

Hence this is the optimal solution to the logistic regression.

8

The linear hypothesis that mimizes E_{in} is the line that passes $(x_1, f(x_1))$ and $(x_2, f(x_2))$. Hence

$$\begin{aligned} g(x) &= f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) \\ &= -2(x_1 + x_2)x + 2x_1x_2 + 1 \end{aligned}$$

For such a hypothesis, $E_{\text{in}} = 0$.
E_{out} is given by

$$\begin{aligned} E_{\text{out}} &= \int_0^1 (g(x) - f(x))^2 \, \mathrm{d}x \\ &= \int_0^1 \left(2x_1x_2 - 2(x_1 + x_2)x + 2x^2\right)^2 \mathrm{d}x \end{aligned}$$

$$\begin{aligned} \mathbb{E}_{\mathcal{D}}(|E_{\text{in}} - E_{\text{out}}|) &= \int_0^1 \int_0^1 |E_{\text{out}}| \, \mathrm{d}x_1 \, \mathrm{d}x_2 \quad (E_{\text{in}} = 0) \\ &= \int_0^1 \int_0^1 \int_0^1 \left(2x_1x_2 - 2(x_1 + x_2)x + 2x^2\right)^2 \mathrm{d}x \, \mathrm{d}x_1 \, \mathrm{d}x_2 \\ &= \frac{2}{15} \end{aligned}$$

9

Let

$$\tilde{\mathrm{X}} = \mathrm{X} + \mathrm{E}$$

Where

$$\mathrm{E} = \begin{bmatrix} | & \dots & | \\ \epsilon & \dots & \epsilon \\ | & \dots & | \end{bmatrix}$$

$$\begin{aligned} \mathrm{X}_h^T \mathrm{X}_h &= \begin{bmatrix} \mathrm{X}^T & \tilde{\mathrm{X}}^T \end{bmatrix} \begin{bmatrix} \mathrm{X} \\ \tilde{X} \end{bmatrix} \\ &= \mathrm{X}^T \mathrm{X} + \tilde{\mathrm{X}}^T \tilde{\mathrm{X}} \\ &= 2\mathrm{X}^T \mathrm{X} + \mathrm{E}^T \mathrm{X} + \mathrm{X}^T \mathrm{E} + \mathrm{E}^T \mathrm{E} \end{aligned}$$

Therefore

$$\begin{aligned} \mathbb{E}[\mathrm{X}_h^T \mathrm{X}_h] &= \mathbb{E}[2\mathrm{X}^T \mathrm{X} + \mathrm{E}^T \mathrm{X} + \mathrm{X}^T \mathrm{E} + \mathrm{E}^T \mathrm{E}] \\ &= 2\mathrm{X}^T \mathrm{X} + \mathbb{E}[\mathrm{E}^T]\mathrm{X} + \mathrm{X}^T \mathbb{E}[\mathrm{E}] + \mathbb{E}[\mathrm{E}^T \mathrm{E}] \end{aligned}$$

Since ϵ is generated i.i.d. from a normal distribution with variance σ^2, we have

$$\begin{aligned} \mathbb{E}[\mathrm{E}^T] &= \mathrm{O}_{(d+1)\times N} \\ \mathbb{E}[\mathrm{E}] &= \mathrm{O}_{N\times(d+1)} \\ \mathbb{E}[\mathrm{E}^T \mathrm{E}] &= N\sigma^2 \mathrm{I}_{d+1} \end{aligned}$$

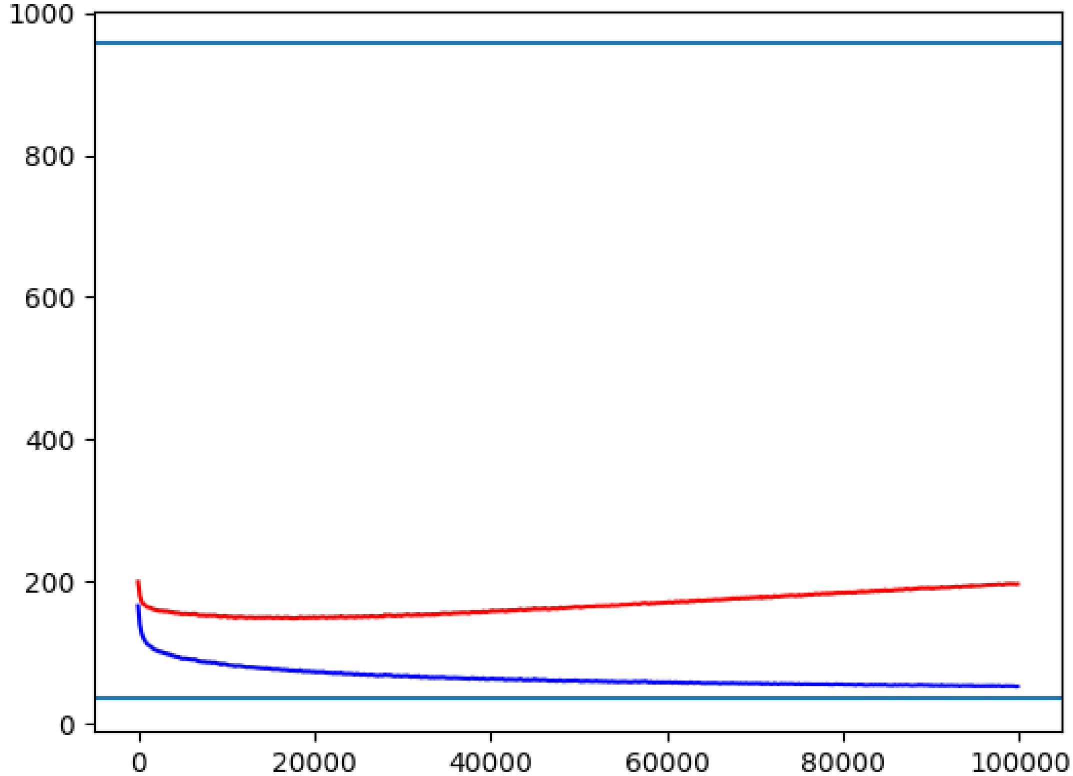
Hence

$$\mathbb{E}[\mathrm{X}_h^T \mathrm{X}_h] = 2\mathrm{X}^T \mathrm{X} + N\sigma^2 \mathrm{I}_{d+1}$$

So $(\alpha, \beta) = (2, N)$

10

Figure:



The horizontal light-blue lines at top and bottom are the average $E_{\text{out}}(\mathbf{w}_{lin})$ and $E_{\text{in}}(\mathbf{w}_{lin})$ respectively.

The red curve is the average $E_{\text{out}}(\mathbf{w}_t)$, and the dark-blue curve is the average $E_{\text{in}}(\mathbf{w}_t)$.

The horizontal axis represents t in the SGD process.

Findings:

First we can see that compared with the average values of linear regression, the E_{in} of SGD is slightly larger, and the E_{out} of SGD is significantly lower. This implies that SGD is probably a better approach compared with directly computing the weight vector in this case.

We see that starting from about 200 at $t = 0$, $E_{\text{in}}(\mathbf{w}_t)$ decreases monotonically as t increases. This is as expected, since the model can fit the $N = 64$ training data vectors better after more iterations.

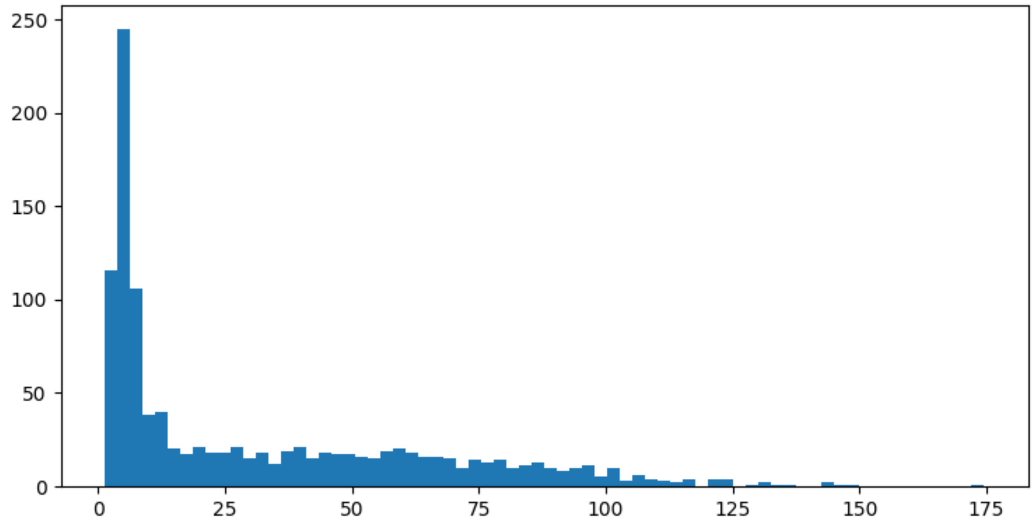
We also see that as t increases, $E_{\text{out}}(\mathbf{w}_t)$ first decreases slightly, after reaching its minimum at small t (around 0), it begins to increase slowly. At the end of iterations $t = 100000$, $E_{\text{out}}(\mathbf{w}_t)$ is approximately at the same level as when the iteration started. A possible cause for this phenomenon is that at the very beginning of the SGD process, E_{out} decreases because some information about the data set is learned by the algorithm. But as t increases, the weight vector gets more biased by the $N = 64$ training data vectors, and its ability to classify the entire data set is compromised, hence the increasing E_{out}. This also means that a large number of iterations may not be meaningful or beneficial in this learning problem, since most iterations result in the increase of E_{out}. Moreover, there is no improve in E_{out} at the end of the 100000 iterations compared with at the beginning, when little "learning" is done.

Code snapshot:

```python
import numpy as np
import matplotlib.pyplot as plt


#### parse data
def parse_data_line(line):
    parts = line.strip().split()
    y = int(parts[0])
    vector = []
    vector.append(1)
    for part in parts[1:]:
        _, value = part.split(":")
        vector.append(float(value))
    return y, np.array(vector)


def parse_data_file():
    file_path = "data.txt"
    matrix = []
    y_values = []
    with open(file_path, "r") as file:
        for line in file:
            y, vector = parse_data_line(line)
            y_values.append(y)
            matrix.append(vector)
    return np.array(y_values), np.array(matrix)


y, matrix = parse_data_file()
#### end parse

e_lin_in = []
e_lin_out = []
e_t_in = np.zeros(500)
e_t_out = np.zeros(500)

def sgd(random_indicies):
    w = np.zeros(13)
    for j in range(1, 100001):
        ind = random_indicies[np.random.randint(0, 64)]
        vec = matrix[ind, :].transpose()
        w = np.subtract(w, 0.02 * (np.dot(vec, w) - y[ind]) * vec)
        if j % 200 == 0:
            err = np.subtract(y, np.matmul(matrix, w))
            e_in = 0
            for k in random_indicies:
                e_in += err[k] ** 2
            e_out = np.linalg.norm(err) ** 2
            # print(e_out)
            e_out -= e_in
            # print(e_out)
            # for j in range(err.shape[0]):
            #     if j in random_indicies:
```

11

Figure:



The average value of $E_{in}^{sqr}(\mathbf{w}_{lin}) - E_{in}^{sqr}(\mathbf{w}_{poly})$ is 32.45148423777838

This figure is the histogram for $E_{in}^{sqr}(\mathbf{w}_{lin}) - E_{in}^{sqr}(\mathbf{w}_{poly})$, the horizontal axis is the value of $E_{in}^{sqr}(\mathbf{w}_{lin}) - E_{in}^{sqr}(\mathbf{w}_{poly})$, and the vertical axis is the number of times a value happens in the 1126 experiments (call this "frequency" in the following context).

We see that $E_{in}^{sqr}(\mathbf{w}_{lin}) - E_{in}^{sqr}(\mathbf{w}_{poly})$ mostly fall between 0 and 10, as the frequency is high in this range. The frequency drops rapidly at about $E_{in}^{sqr}(\mathbf{w}_{lin}) - E_{in}^{sqr}(\mathbf{w}_{poly}) = 10$, after the rapid drop, $E_{in}^{sqr}(\mathbf{w}_{lin}) - E_{in}^{sqr}(\mathbf{w}_{poly})$ decreases slowly, the largest value is slightly less than 150.

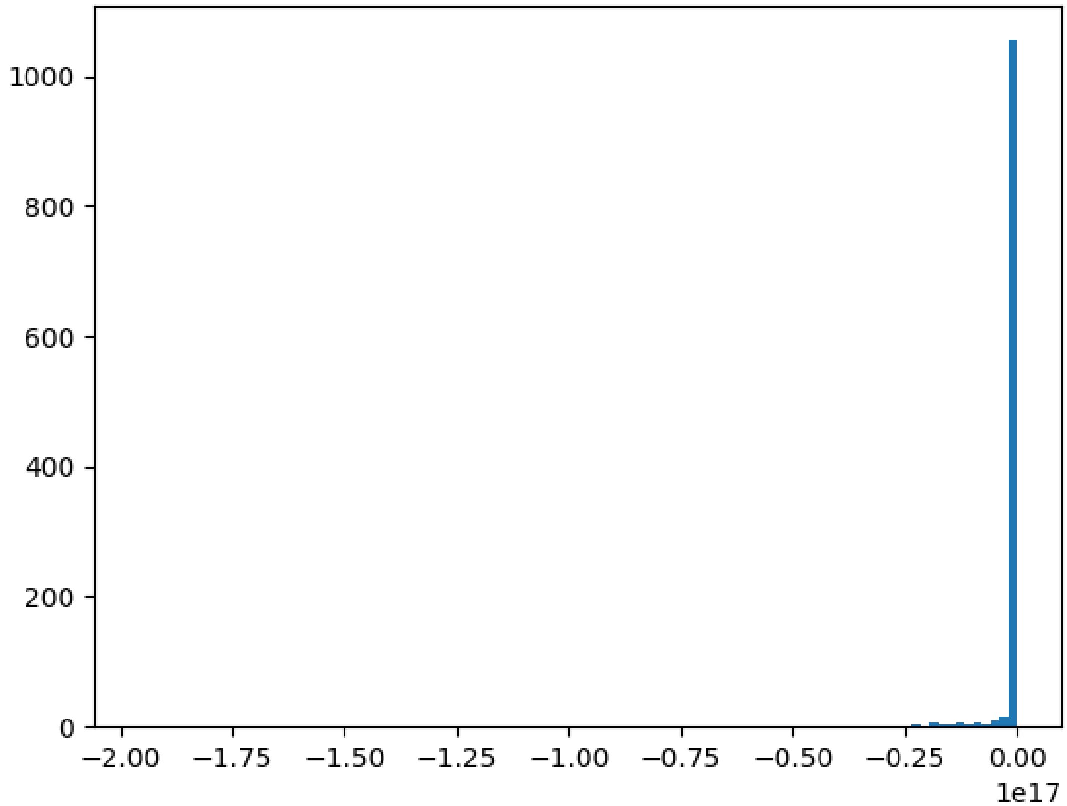
The histogram and the average value of $E_{in}^{sqr}(\mathbf{w}_{lin}) - E_{in}^{sqr}(\mathbf{w}_{poly})$ implies that the homogeneous polynomial transform does indeed result in a E_{in} gain. (which is in fact a decrease in E_{in}, but can be seen as a gain in the ability to fit the training set judging by E_{in}). Furthermore, compared with the average value of $E_{\text{in}}(\mathbf{w}_{lin})$ in P10, which is about 35 (see the light-blue horizontal line at the bottom), the E_{in} gain (≈ 32.5) is quite significant. This further shows that the polynomial transform reduces E_{in} effectively. This is because by allowing a more complicated model, the classifier can fit the $N = 64$ training data vectors better.

Code snapshot:

```python
def poly_regression(random_indicies):
    sub_x = poly_matrix[random_indicies, :]
    sub_y = y[random_indicies]
    x_dagger = np.linalg.pinv(sub_x)
    w_lin = np.matmul(x_dagger, sub_y)
    err = np.subtract(y, np.matmul(poly_matrix, w_lin))
    e_in = 0
    for j in random_indicies:
        e_in += err[j] ** 2
    return e_in / 64


def linear_regression(random_indicies):
    sub_x = matrix[random_indicies, :]
    sub_y = y[random_indicies]
    x_dagger = np.linalg.pinv(sub_x)
    w_lin = np.matmul(x_dagger, sub_y)
    err = np.subtract(y, np.matmul(matrix, w_lin))
    e_in = 0
    for j in random_indicies:
        e_in += err[j] ** 2
    return e_in / 64


for i in range(1126):
    print(i)
    np.random.seed(i)
    random_indicies = np.random.choice(matrix.shape[0], size = 64, replace = False)
    e_in_lin = linear_regression(random_indicies)
    e_in_poly = poly_regression(random_indicies)
    e_lin_in.append(e_in_lin)
    e_poly_in.append(e_in_poly)
#

# random_indicies = np.random.choice(matrix.shape[0], size = 64, replace = False)
# e_in, e_out = linear_regression(random_indicies)
# e_lin_in.append(e_in)
# e_lin_out.append(e_out)
# sgd(random_indicies);

# print(e_t_in)
# print(e_t_out)

# print(e_poly_in)
# print(e_lin_in)

de = []
for i in range(0, len(e_poly_in)):
    de.append(e_lin_in[i] - e_poly_in[i])
plt.hist(de, bins = 30)
print(np.mean(de))
plt.show()
#
```

12

Figure:



The average value of $E_{out}^{sqr}(\mathbf{w}_{lin}) - E_{out}^{sqr}(\mathbf{w}_{poly})$ is -1051028513192239.2

This figure is the histogram of $E_{out}^{sqr}(\mathbf{w}_{lin}) - E_{out}^{sqr}(\mathbf{w}_{poly})$, the horizontal axis is the value of $E_{out}^{sqr}(\mathbf{w}_{lin}) - E_{out}^{sqr}(\mathbf{w}_{poly})$, and the vertical axis is the number of times a value happens in the 1126 experiments. We see that $E_{out}^{sqr}(\mathbf{w}_{poly}) - E_{out}^{sqr}(\mathbf{w}_{lin})$ is huge, which implies that $E_{out}^{sqr}(\mathbf{w}_{poly})$ is far greater than $E_{out}^{sqr}(\mathbf{w}_{lin})$.

Together with the result in P11, we can see that adopting the polynomial transform in this problem makes classifying performance a lot worse. This is because the polynomial transform allows a more complicated model, and although E_{in} is decreased, the classifying performance is worse due to the hazard of overfitting.

To conclude, the linear regression with the polynomial transform in this problem results in a bad learning model. (compared with direct linear regression)

Code snapshot:

```python


#### end parse

e_lin_in = []
e_poly_in = []

def poly_regression(random_indicies):
    sub_x = poly_matrix[random_indicies, :]
    sub_y = y[random_indicies]
    x_dagger = np.linalg.pinv(sub_x)
    w_lin = np.matmul(x_dagger, sub_y)
    err = np.subtract(y, np.matmul(poly_matrix, w_lin))
    e_in = 0
    e_out = np.linalg.norm(err) ** 2
    print(e_out / 8128)
    for j in random_indicies:
        e_in += err[j] ** 2
    e_out -= e_in
    return e_out / 8128


def linear_regression(random_indicies):
    sub_x = matrix[random_indicies, :]
    sub_y = y[random_indicies]
    x_dagger = np.linalg.pinv(sub_x)
    w_lin = np.matmul(x_dagger, sub_y)
    err = np.subtract(y, np.matmul(matrix, w_lin))
    e_in = 0
    e_out = np.linalg.norm(err) ** 2
    print(e_out / 8128)
    for j in random_indicies:
        e_in += err[j] ** 2
    e_out -= e_in
    return e_out / 8128


for i in range(1126):
    print(i)
    np.random.seed(i ** 2)
    random_indicies = np.random.choice(matrix.shape[0], size = 64, replace = False)
    e_out_lin = linear_regression(random_indicies)
    e_out_poly = poly_regression(random_indicies)
    e_lin_in.append(e_out_lin)
    e_poly_in.append(e_out_poly)


de = []
for i in range(0, len(e_poly_in)):
    de.append(e_lin_in[i] - e_poly_in[i])
plt.hist(de, bins = 100)
print(np.mean(de))
plt.show()
```

NORMAL main P12.py utf-8 python Bot 87:1

13

Notice that

$$\begin{aligned}
\operatorname{sign}\left(w_0 + \sum_{i=1}^{d} w_i x_i\right) &= \operatorname{sign}\left(\exp\left(w_0 + \sum_{i=1}^{d} w_i x_i\right) - 1\right) \\
&= \operatorname{sign}\left(-e^{-w_0} + \prod_{i=1}^{d} (\exp(x_i))^{w_i}\right)
\end{aligned}$$

Therefore we see that $h_{\mathbf{w}}(\mathbf{x}) = \tilde{h}_{\mathbf{u}}(\mathbf{x}')$ if we let

$$\begin{aligned}
u_0 &= -e^{-w_0} \\
u_i &= w_i \quad i = 1, 2, \dots, d \\
x_i' &= \exp(x_i) - 1
\end{aligned}$$

Hence if there exist n data vectors shattered by $\mathcal{H}_1$, we can find a corresponding set of n data vectors that is shattered by $\mathcal{H}_2$, which implies that $d_{vc}(\mathcal{H}_1) \leq d_{vc}(\mathcal{H}_2)$.

We can also see that $h_{\mathbf{w}}(\mathbf{x}') = \tilde{h}_{\mathbf{u}}(\mathbf{x})$ if we let

$$\begin{aligned}
x_i' &= \ln(1 + |x_i|) \\
w_0 &= \begin{cases} -\ln(-u_0) & u_0 < 0 \\ 1 & u_0 \geq 0 \end{cases} \\
w_i &= \begin{cases} u_i & u_0 < 0 \\ 0 & u_0 \geq 0 \end{cases} \quad i = 1, 2, \dots, d
\end{aligned}$$

(Observe that $\tilde{h}_{\mathbf{u}}(\mathbf{x}) = +1$ if $u_0 \geq 0$)
Hence if there exist n data vectors shattered by $\mathcal{H}_2$, we can find a corresponding set of n data vectors that is shattered by $\mathcal{H}_1$, whcih implies that $d_{vc}(\mathcal{H}_2) \leq d_{vc}(\mathcal{H}_1)$.
Since $d_{vc}(\mathcal{H}_1) \leq d_{vc}(\mathcal{H}_2)$ and $d_{vc}(\mathcal{H}_2) \leq d_{vc}(\mathcal{H}_1)$, $d_{vc}(\mathcal{H}_2) = d_{vc}(\mathcal{H}_1)$, and the statement is disproved.