Machine Learning: Homework #2

Due on April 21, 2021

Professor Xiao Li

Peng Deng

Problem 1

Suppose f maps sample space $\mathcal{X}$ to {0,1} and let $\mathcal{S}$ be n i.i.d training samples. For any $t > 0$, we have the concentration inequality (can be shown by Hoeffding's inequity):

$$\Pr\left[\left|\mathrm{Er}_{\mathrm{in}}\left(f\right) - \mathrm{Er}_{\mathrm{out}}\left(f\right)\right| > t\right] \leq 2e^{-2nt^2}$$

Show that the variance of $\mathrm{Er}_{\mathrm{in}}\left(f\right)$ satisfies $\mathrm{Var}\left[\mathrm{Er}_{\mathrm{in}}\left(f\right)\right] \leq \frac{\log(2e)}{n}$.(**Hint**: use $\mathbb{E}\left[X^2\right] = \int_0^\infty \Pr\left[X^2 > t\right]dt$)

Solution

$$\begin{aligned}
\mathrm{Var}\left[\mathrm{Er}_{\mathrm{in}}\left(f\right)\right] &= \mathrm{E}\left[\left(\mathrm{Er}_{\mathrm{in}}\left(f\right) - \mathrm{E}_{\mathcal{S}\sim\mathcal{D}}\left[\mathrm{Er}_{\mathrm{in}}\left(f\right)\right]\right)^2\right] \\
&= \int_0^\infty \Pr\left[\left(\mathrm{Er}_{\mathrm{in}}\left(f\right) - \mathrm{E}_{\mathcal{S}\sim\mathcal{D}}\left[\mathrm{Er}_{\mathrm{in}}\left(f\right)\right]\right)^2 > t\right] dt \\
&= \int_0^\infty \Pr\left[\left|\mathrm{Er}_{\mathrm{in}}\left(f\right) - \mathrm{E}_{\mathcal{S}\sim\mathcal{D}}\left[\mathrm{Er}_{\mathrm{in}}\left(f\right)\right]\right| > \sqrt{t}\right] dt \\
&\leq \int_0^\infty 2e^{-2nt} dt \\
&= \frac{1}{n} \\
&\leq \frac{\log(2e)}{n}
\end{aligned} \tag{1}$$

Thus, we finished the proof.

Problem 2

Suppose $f \in \mathcal{H}$ maps sample space $\mathcal{X}$ to {0,1} . The Rademacher complexity bound derived in our lecture is by applying McDiarmid's inequality to

$$h(\mathcal{S}) = \sup_{f\in\mathcal{H}}\left[\mathrm{Er}_{\mathrm{out}}\left(f\right) - \mathrm{Er}_{\mathrm{in}}\left(f\right)\right]$$

Try to obtain a generalization bound in terms of the empirical Rademacher complexity $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H})$ by applying McDiarmid's inequality to

$$\Psi(\mathcal{S}) = \sup_{f\in\mathcal{H}}\left[\mathrm{Er}_{\mathrm{out}}(f) - \mathrm{Er}_{\mathrm{in}}(f) - \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H})\right]$$

Hint: You can follow the analysis in slides 6 page $20-21$ and use directly the result in slides 6 page $22-24$ that $\mathbb{E}[h(\mathcal{S})] \leq \mathcal{R}(\mathcal{H})$.

Solution

Firstly, we check the condition in McDiarmid's inequality. Let

$$\begin{aligned}
\mathcal{S} &= \{\boldsymbol{z}_1, \cdots, \boldsymbol{z}_i, \cdots, \boldsymbol{z}_n\} \\
\mathcal{S}' &= \{\boldsymbol{z}_1, \cdots, \boldsymbol{z}_i', \cdots, \boldsymbol{z}_n\}
\end{aligned}$$

Problem 2 continued on next page...

We have

$$\begin{aligned}
\Psi(\mathcal{S}') - \Psi(\mathcal{S}) &= \sup_{f\in\mathcal{H}}\left[\mathrm{Er}_{\mathrm{out}}(f) - \mathrm{Er}_{\mathrm{in}}(f;\mathcal{S}') - \widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{H})\right] - \sup_{f\in\mathcal{H}}\left[\mathrm{Er}_{\mathrm{out}}(f) - \mathrm{Er}_{\mathrm{in}}(f;\mathcal{S}) - \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H})\right] \\
&\le \sup_{f\in\mathcal{H}}\left[\mathrm{Er}_{\mathrm{in}}(f;\mathcal{S}) + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) - \mathrm{Er}_{\mathrm{in}}(f;\mathcal{S}') - \widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{H})\right] \\
&= \sup_{f\in\mathcal{H}}\left[\frac{e\left(f\left(\boldsymbol{x}_i\right),y_i\right) - e\left(f\left(\boldsymbol{x}_i'\right),y_i'\right)}{n} + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) - \widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{H})\right] \\
&\le \sup_{f\in\mathcal{H}}\left[\frac{e\left(f\left(\boldsymbol{x}_i\right),y_i\right) - e\left(f\left(\boldsymbol{x}_i'\right),y_i'\right)}{n}\right] + \sup_{f\in\mathcal{H}}\left[\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) - \widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{H})\right] \\
&= \sup_{f\in\mathcal{H}}\left[\frac{e\left(f\left(\boldsymbol{x}_i\right),y_i\right) - e\left(f\left(\boldsymbol{x}_i'\right),y_i'\right)}{n}\right] + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) - \widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{H}) \\
&\le \frac{1}{n} + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) - \widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{H})
\end{aligned} \tag{2}$$

Then, let's consider $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) - \widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{H})$

$$\begin{aligned}
&\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) - \widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{H}) \\
&= \mathrm{E}_{\boldsymbol{\epsilon}}\left[\sup_{f\in\mathcal{H}}\frac{1}{n}\left[\epsilon_1 f\left(\boldsymbol{x}_1\right) + \cdots + \epsilon_i f\left(\boldsymbol{x}_i\right) + \cdots + \epsilon_n f\left(\boldsymbol{x}_n\right)\right]\right] - \mathrm{E}_{\boldsymbol{\epsilon}}\left[\sup_{f\in\mathcal{H}}\frac{1}{n}\left[\epsilon_1 f\left(\boldsymbol{x}_1\right) + \cdots + \epsilon_i f\left(\boldsymbol{x}_i'\right) + \cdots + \epsilon_n f\left(\boldsymbol{x}_n\right)\right]\right] \\
&= \mathrm{E}_{\boldsymbol{\epsilon}}\left[\sup_{f\in\mathcal{H}}\frac{1}{n}\left[\epsilon_1 f\left(\boldsymbol{x}_1\right) + \cdots + \epsilon_i f\left(\boldsymbol{x}_i\right) + \cdots + \epsilon_n f\left(\boldsymbol{x}_n\right)\right] - \sup_{f\in\mathcal{H}}\frac{1}{n}\left[\epsilon_1 f\left(\boldsymbol{x}_1\right) + \cdots + \epsilon_i f\left(\boldsymbol{x}_i'\right) + \cdots + \epsilon_n f\left(\boldsymbol{x}_n\right)\right]\right] \\
&\le \mathrm{E}_{\boldsymbol{\epsilon}}\left[\sup_{f\in\mathcal{H}}\frac{\epsilon_i}{n}\left[f\left(\boldsymbol{x}_i\right) - f\left(\boldsymbol{x}_i'\right)\right]\right] \\
&= \left[\sup_{f\in\mathcal{H}}\frac{1}{n}\left[f\left(\boldsymbol{x}_i\right) - f\left(\boldsymbol{x}_i'\right)\right]\right]\cdot\frac{1}{2} + \left[\sup_{f\in\mathcal{H}}\frac{-1}{n}\left[f\left(\boldsymbol{x}_i\right) - f\left(\boldsymbol{x}_i'\right)\right]\right]\cdot\frac{1}{2} \\
&\le \frac{1}{2n} + \frac{1}{2n} \\
&= \frac{1}{n}
\end{aligned} \tag{3}$$

Back to equation (2), we have

$$\begin{aligned}
\Psi(\mathcal{S}') - \Psi(\mathcal{S}) &\le \frac{1}{n} + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) - \widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{H}) \\
&\le \frac{1}{n} + \frac{1}{n} \\
&= \frac{2}{n}
\end{aligned} \tag{4}$$

Due to symmetric, we also have

$$\Psi(\mathcal{S}) - \Psi(\mathcal{S}') \le \frac{2}{n} \tag{5}$$

Thus, we have

$$\left|\Psi(\mathcal{S}) - \Psi(\mathcal{S}')\right| \le \frac{2}{n} \tag{6}$$

Thus, the condition in McDiarmid's inequality is satisfied. Apply McDiarmid's inequality to $\Psi(\mathcal{S})$, we have

$$\Pr\left[\Psi(\mathcal{S}) - \mathrm{E}\left[\Psi(\mathcal{S})\right] \ge t\right] \le e^{\frac{-nt^2}{2}}, \quad \forall t > 0 \tag{7}$$

Let $\delta = e^{\frac{-nt^2}{2}}$, we then have with probability at least 1-δ

$$\Psi\left(\mathcal{S}\right) \leq \mathrm{E}\left[\Psi\left(\mathcal{S}\right)\right] + \sqrt{\frac{2\log\left(\frac{1}{\delta}\right)}{n}} \tag{8}$$

Since

$$\Psi(\mathcal{S}) = \sup_{f\in\mathcal{H}}\left[\mathrm{Er}_{\mathrm{out}}(f) - \mathrm{Er}_{\mathrm{in}}(f) - \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H})\right] \geq \mathrm{Er}_{\mathrm{out}}(f) - \mathrm{Er}_{\mathrm{in}}(f) - \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}), \quad \forall f \in \mathcal{H} \tag{9}$$

By combining equation (8) and (9), we obtain

$$\begin{aligned} \forall f \in \mathcal{H}, \quad \mathrm{Er}_{\mathrm{out}}(f) &\leq \mathrm{Er}_{\mathrm{in}}(f) + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) + \Psi(\mathcal{S}) \\ &\leq \mathrm{Er}_{\mathrm{in}}(f) + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) + \mathrm{E}\left[\Psi\left(\mathcal{S}\right)\right] + \sqrt{\frac{2\log\left(\frac{1}{\delta}\right)}{n}} \end{aligned} \tag{10}$$

Then, we consider the term $\mathrm{E}\left[\Psi\left(\mathcal{S}\right)\right]$ as follow

$$\begin{aligned} \mathrm{E}\left[\Psi\left(\mathcal{S}\right)\right] &= \mathrm{E}_{\mathcal{S}\sim i.i.d.\mathcal{D}}\left[\sup_{f\in\mathcal{H}}\left[\mathrm{Er}_{\mathrm{out}}(f) - \mathrm{Er}_{\mathrm{in}}(f) - \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H})\right]\right] \\ &= \mathrm{E}_{\mathcal{S}}\left[\sup_{f\in\mathcal{H}}\left[\mathrm{Er}_{\mathrm{out}}(f) - \mathrm{Er}_{\mathrm{in}}(f)\right] - \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H})\right] \\ &= \mathrm{E}_{\mathcal{S}}\left[h\left(\mathcal{S}\right) - \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H})\right] \\ &= \mathrm{E}_{\mathcal{S}}\left[h\left(\mathcal{S}\right)\right] - \mathrm{E}_{\mathcal{S}}\left[\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H})\right] \\ &\leq \mathcal{R}\left(\mathcal{H}\right) - \mathcal{R}\left(\mathcal{H}\right) \\ &= 0 \end{aligned} \tag{11}$$

By combining equation 10 and 11, we have

$$\begin{aligned} \forall f \in \mathcal{H}, \quad \mathrm{Er}_{\mathrm{out}}(f) &\leq \mathrm{Er}_{\mathrm{in}}(f) + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) + \mathrm{E}\left[\Psi\left(\mathcal{S}\right)\right] + \sqrt{\frac{2\log\left(\frac{1}{\delta}\right)}{n}} \\ &\leq \mathrm{Er}_{\mathrm{in}}(f) + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) + \sqrt{\frac{2\log\left(\frac{1}{\delta}\right)}{n}} \end{aligned} \tag{12}$$

Above all, we obtain the generalization bound in terms of the empirical Rademacher complexity $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H})$. For any $\delta > 0$, with probability at least $1-\delta$, we have the following generalization bound:

$$\forall f \in \mathcal{H}, \quad \mathrm{Er}_{\mathrm{out}}(f) \leq \mathrm{Er}_{\mathrm{in}}(f) + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}) + \sqrt{\frac{2\log\left(\frac{1}{\delta}\right)}{n}} \tag{13}$$

Problem 3

Toy polynomial regression using ℓ_2 -regularization and cross-validation.
Suppose that we have the underlying model

$$y = x^2 + \varepsilon \tag{14}$$

We collect $n = 10$ data points $\{(x_i, y_i)\}_{i=1}^n$; see the visualization in Figure 1. You can download the data from blackboard. Now, suppose we are going to fit all the data using 8 -th order polynomials:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \ldots + \theta_8 x^8. \tag{15}$$

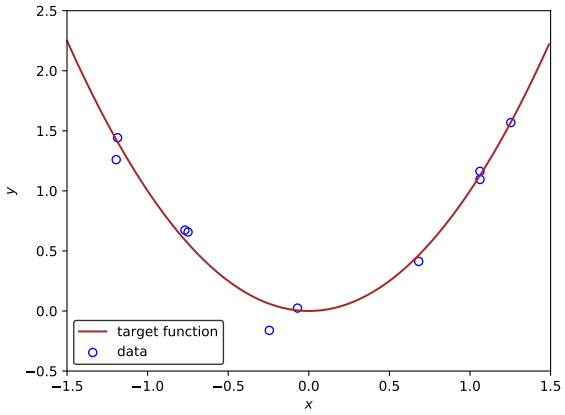

Figure 1 Visualization of data

(a1) Denote the $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_8) \in \mathbb{R}^9$ as the parameter. We have the following linear model

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta}$$

Specify $\boldsymbol{y}$ and $\boldsymbol{X}$ using the training data $\{(x_i, y_i)\}_{i=1}^n$

(a2) Furthermore, we can formulate the following least squares

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\mathbb{R}^9}{\operatorname{argmin}} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 \tag{16}$$

Calculate $\widehat{\boldsymbol{\theta}}$ defined in (16) and plot the fitted curve in Figure 1 (limit x -axis from -1.5 to 1.5 and limit y -axis from -0.5 to 2.5). You can download the code used to generate Figure 1 from blackboard.

(a3) Using the test data set, calculate the test error $\|\boldsymbol{X}_{\text{test}}\widehat{\boldsymbol{\theta}} - \boldsymbol{y}_{\text{test}}\|_2$ of $\widehat{\boldsymbol{\theta}}$ defined in (16).

(b1) Since we know that the underlying model in (14) is quadratic, while the fitting model in (15) is polynomial of order 8, we must have overfitting, which you can see from question (a2) and (a3). One way to prevent overfitting is regularization. Instead of using (16), we formulate the following ℓ_2 -regularized least squares

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\mathbb{R}^9}{\operatorname{argmin}} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2 \tag{17}$$

However, one difficulty to implementing (17) is determining the regularization parameter λ. A too large λ leads to underfitting, while a too small λ(e.g. $\lambda = 0$) results in overfitting. Suppose we set the set of candidates of λ as

$$\begin{bmatrix} 10^{-5} & 10^{-4} & 10^{-3} & 10^{-2} & 10^{-1} & 0.3 & 0.5 & 0.8 & 1 & 2 & 5 & 10 & 15 & 20 & 50 & 100 \end{bmatrix}$$

using **5 -fold cross-validation** to select the regularization parameter λ, and plot the validation error versus the value of λ, where error is the y -axis and λ is the x -axis (set the x -axis to log-scale).

(b2) Based on the result in (b1), set $\lambda = 0.01, 0.1, 0.8$, and 5 in (17) and solve for the corresponding $\widehat{\boldsymbol{\theta}}$, respectively. Plot the fitted curve using the former four choices of λ in Figure 1 , you have to draw four figures separately (limit x -axis from - 1.5 to 1.5 and limit y -axis from -0.5 to 2.5).

(b3) Using the test data set, calculate the test error $\|\boldsymbol{X}_{\text{test}}\widehat{\boldsymbol{\theta}} - \boldsymbol{y}_{\text{test}}\|_2$ of each of $\widehat{\boldsymbol{\theta}}$ obtained in (b2).

Solution

Subproblem (a1)

We can specify $\boldsymbol{y}$ and $\boldsymbol{X}$ as follow.

$$\boldsymbol{y} = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \end{pmatrix}^\top$$
$$\boldsymbol{X} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^8 \\ 1 & x_2 & x_2^2 & \cdots & x_2^8 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^8 \end{pmatrix} \tag{18}$$

Subproblem (a2)

According to the closed form solution of least squares, we have

$$\widehat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{19}$$

Then, we can plot the fitted curve as Figure 2.

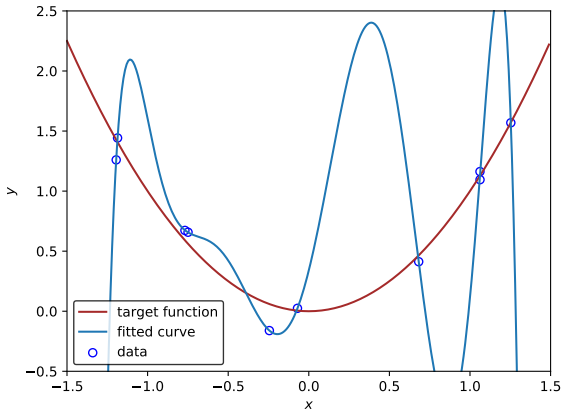

Figure 2 Fitted curve

As we can see from Figure 2, when we use 8-th order polynomials, we can almost fit all the 10 points. However, the fitted curve is far away from the target function, which means it is overfitted. Although the in-sample error is small (fit the training data very well), the out of sample error is large, which means the model's generalization ability is not good.

Subproblem (a3)

By using Python, we can compute the test error as follow:

$$\|\boldsymbol{X}_{\text{test}}\widehat{\boldsymbol{\theta}} - \boldsymbol{y}_{\text{test}}\|_2 = 106.40 \tag{20}$$

As we can see from equation 20, the test error is very large, which means the 8-th order polynomial model is overfitted.

Subproblem (b1)

We know that the ℓ_2-regularized least squares has the closed form solution as follow:

$$\widehat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{21}$$

By using 5-fold cross validation, we separate the training dataset into 5 small datasets, and each small dataset has 2 training data. Every iteration, we choose one small dataset as validation dataset, and the left small datasets as training datasets. Then, we average the validation errors among 5 iterations. The figure of validation error versus the value of λ is showed as Figure 3.

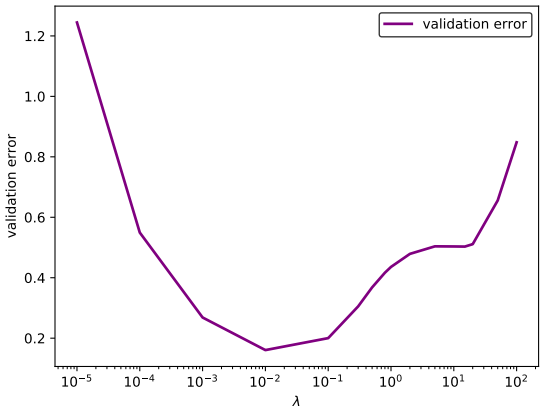

Figure 3 The validation error vs λ

As we can see from Figure 3, when $\lambda = 10^{-2}$, we can get the smallest validation error.

Subproblem (b2)

We set $\lambda = 0.01$, 0.1, 0.8 and 5 separately, and use the original training dataset to train the model. Then, we plot the fitted curve with different λ as Figure 4.

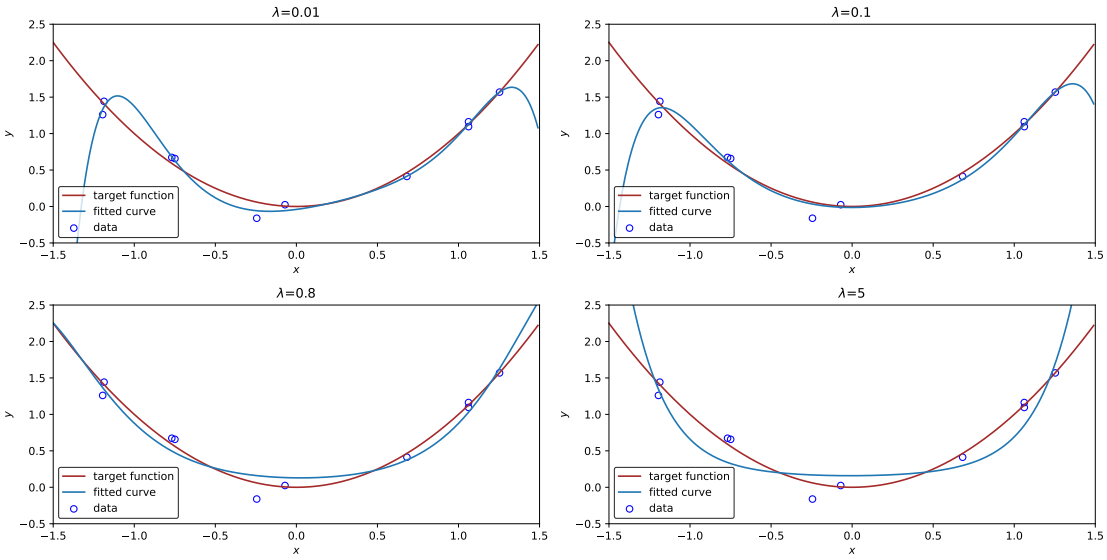

Figure 4 The fitted curve with different λ

As we can see from Figure 4, when λ =0.01 and 0.1, it seems that the model is overfitted, and when $\lambda = 5$, it seems that the model is underfitted. Finally, when λ =0.8, the model seems good and the fitted figure is close to the target function.

Subproblem (b3)

Using the test data set, we can calculate the test error $\|\boldsymbol{X}_{\text{test}}\widehat{\boldsymbol{\theta}} - \boldsymbol{y}_{\text{test}}\|_2$ of each of $\widehat{\boldsymbol{\theta}}$ obtained in (b2) as follow:

$$\begin{aligned} \lambda = 0.01 : \quad & \|\boldsymbol{X}_{\text{test}}\widehat{\boldsymbol{\theta}} - \boldsymbol{y}_{\text{test}}\|_2 = 12.46 \\ \lambda = 0.1 \ : \quad & \|\boldsymbol{X}_{\text{test}}\widehat{\boldsymbol{\theta}} - \boldsymbol{y}_{\text{test}}\|_2 = 5.21 \\ \lambda = 0.8 \ : \quad & \|\boldsymbol{X}_{\text{test}}\widehat{\boldsymbol{\theta}} - \boldsymbol{y}_{\text{test}}\|_2 = 0.42 \\ \lambda = 5 \quad : \quad & \|\boldsymbol{X}_{\text{test}}\widehat{\boldsymbol{\theta}} - \boldsymbol{y}_{\text{test}}\|_2 = 3.65 \end{aligned} \tag{22}$$

The Python code to solve Problem 3 is as follow

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#a1------------
data = pd.read_csv("Q3_training_data.csv", header=0, index_col=0)
x_train = np.array(data.iloc[0,:]).reshape(-1, 1)
y_train = np.array(data.iloc[1,:]).reshape(-1, 1)

#a2------------
x_sim = np.arange(-1.5, 1.5, 0.01)
y_sim = x_sim ** 2

plt.figure(1)
plt.plot(x_sim, y_sim, color="brown", label="target function")
plt.scatter(x_train, y_train, marker='o', c='', edgecolors='b', label="data")
plt.xlim(-1.5, 1.5)
plt.ylim(-0.5, 2.5)
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.legend(loc="best", edgecolor="black")
plt.savefig("hw_tex/tupian/original.pdf")

order = 8
n_train = 10
x_train_new = np.ones(n_train).reshape(-1,1)
new_col = np.ones(n_train).reshape(-1,1)
for i in range(1,order+1):
    new_col = new_col * x_train
    x_train_new = np.hstack((x_train_new, new_col))
y_train_new = y_train
theta_hat = np.dot(np.dot(np.linalg.inv(np.dot(x_train_new.T, x_train_new)), x_train_new.T),
                                                y_train_new)

x_sim = np.arange(-1.5, 1.5, 0.01)
y_sim = np.zeros(len(x_sim))
for i in range(9):
    y_sim = y_sim + theta_hat[i] * (x_sim**i)

plt.plot(x_sim, y_sim, label="fitted curve")
plt.legend(loc="best", edgecolor="black")
plt.savefig("hw_tex/tupian/fitted.pdf")

#a3------------
data_test = pd.read_csv("Q3_test_data.csv", header=0, index_col=0)
x_test = np.array(data_test.iloc[0,:]).reshape(-1, 1)
y_test = np.array(data_test.iloc[1,:]).reshape(-1, 1)

n_test = 10
x_test_new = np.ones(n_test).reshape(-1,1)
```

```
new_col = np.ones(n_test).reshape(-1,1)
for i in range(1,order+1):
    new_col = new_col * x_test
    x_test_new = np.hstack((x_test_new, new_col))
y_test_new = y_test
test_error = np.linalg.norm(np.dot(x_test_new, theta_hat) - y_test_new, ord=2)
print("test error:", test_error)

#b1------------
lambda_candidate = [1e-5, 1e-4, 1e-3, 1e-2, 0.1, 0.3, 0.5, 0.8, 1, 2, 5, 10, 15, 20, 50, 100
                                                ]
k = 5
s_size = n_train / k
error_list = []
for lambda_i in lambda_candidate:
    error = 0
    for val_i in range(k):
        x_val = x_train_new[[val_i*2, val_i*2+1], :]
        y_val = y_train_new[[val_i*2, val_i*2+1], :]
        x_train_val = np.vstack((x_train_new[0:val_i*2, :], x_train_new[val_i*2+2:, :]))
        y_train_val = np.vstack((y_train_new[0:val_i*2, :], y_train_new[val_i*2+2:, :]))
        I = np.identity(x_train_val.shape[1])
        theta_hat = np.dot(np.dot(np.linalg.inv(np.dot(x_train_val.T, x_train_val)+lambda_i*
                                                    I), x_train_val.T), y_train_val)
        error  = error + np.linalg.norm(np.dot(x_val, theta_hat) - y_val, ord=2)

    error = error / k
    error_list.append(error)

plt.figure(2)
plt.semilogx(lambda_candidate, error_list, linewidth=2, color="purple", label="validation
                                                error")
plt.xlabel("$\lambda$")
plt.ylabel("validation error")
plt.legend(loc="best", edgecolor="black")
plt.savefig("hw_tex/tupian/validation.pdf")

#b2------------
lambda_candidate = [0.01, 0.1, 0.8, 5]
figure_num = 1
plt.figure(3, figsize=(20, 20))

test_error_list = []
for lambda_i in lambda_candidate:
    plt.subplot(2, 2, figure_num)
    figure_num = figure_num + 1

    x_sim = np.arange(-1.5, 1.5, 0.01)
    y_sim = x_sim ** 2
    plt.plot(x_sim, y_sim, color="brown", label="target function")
    plt.scatter(x_train, y_train, marker='o', c='', edgecolors='b', label="data")


    I = np.identity(x_train_new.shape[1])
    theta_hat = np.dot(np.dot(np.linalg.inv(np.dot(x_train_new.T, x_train_new)+lambda_i*I),
                                                x_train_new.T), y_train_new)
    x_sim = np.arange(-1.5, 1.5, 0.01)
    y_sim = np.zeros(len(x_sim))
    for i in range(9):
        y_sim = y_sim + theta_hat[i] * (x_sim**i)
    plt.plot(x_sim, y_sim, label="fitted curve")
```

```
    plt.xlim(-1.5, 1.5)
    plt.ylim(-0.5, 2.5)
    plt.xlabel("$x$")
    plt.ylabel("$y$")
    plt.title("$\lambda$={}".format(lambda_i))
    plt.legend(loc="lower left", edgecolor="black")
    plt.tight_layout()

    test_error = np.linalg.norm(np.dot(x_test_new, theta_hat) - y_test_new, ord=2)
    test_error_list.append(test_error)

plt.savefig("hw_tex/tupian/lambda.pdf")

#b3------------
print()
print("-"*20)
for i, test_error in enumerate(test_error_list):
    print("lambda={}, test error={}".format(lambda_candidate[i], test_error))
```

Problem 4

The logistic regression developed in class is designated for Binary classification. How about when number of classes $K > 2$? The key idea is to assign each class $\ell = 1, \ldots, K$ a weight vector $\boldsymbol{\theta}^\ell$. Let $\boldsymbol{\Theta} = \left[\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^K\right] \in \mathbb{R}^{d\times K}$ and $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be the training data. The model for estimating the a-posteriori of y_i is given by

$$\Pr\left[y_i = \ell \mid \boldsymbol{\Theta}, \boldsymbol{x}_i\right] = \frac{\exp\left(\left\langle \boldsymbol{\theta}^\ell, \boldsymbol{x}_i \right\rangle\right)}{\sum_{j=1}^K \exp\left(\left\langle \boldsymbol{\theta}^j, \boldsymbol{x}_i \right\rangle\right)}$$

It is clear that $\Pr\left[y_i = \ell \mid \boldsymbol{\Theta}, \boldsymbol{x}_i\right]$ sum to 1 over ℓ. Using the reasoning of log-likelihood, we can formulate the learning problem as

$$\widehat{\boldsymbol{\Theta}} = \underset{\boldsymbol{\Theta}\in\mathbb{R}^{d\times K}}{\operatorname{argmin}} -\frac{1}{n}\sum_{i=1}^n\sum_{\ell=1}^K 1_{\{y_i=\ell\}} \log\left(\frac{\exp\left(\left\langle \boldsymbol{\theta}^\ell, \boldsymbol{x}_i \right\rangle\right)}{\sum_{j=1}^K \exp\left(\left\langle \boldsymbol{\theta}^j, \boldsymbol{x}_i \right\rangle\right)}\right) \tag{23}$$

where $1_{\{y_i=\ell\}}$ is the indicator function defined as

$$1_{\{y_i=\ell\}} = \begin{cases} 1, & y_i = \ell \\ 0, & O.W \end{cases}$$

MNIST dataset consists of 10 digits from 0 to 9 (i.e., $K = 10$); see Figure 5. Download the training and test dataset from blackboard. Each MNIST image is of dimension 28×28. We vectorize each image and this leads to $\boldsymbol{x}_i \in \mathbb{R}^{784}$ (i.e., $d = 28 \times 28 = 784$). The training data consists of $n = 2800$ digital images, while the test data consists of 200 images (each row represents one image).
Learning task: Apply the multinomial logistic regression (23) to this MNIST dataset and the settings are:

- Using accelerated gradient descent algorithm.
- Set the stepsize to $\mu_k = \mu = 0.01$.
- Start the algorithm at the origin.
- Set the total number of iterations to 500 .
- Plot the training accuracy versus iteration number.

Problem 4 continued on next page...

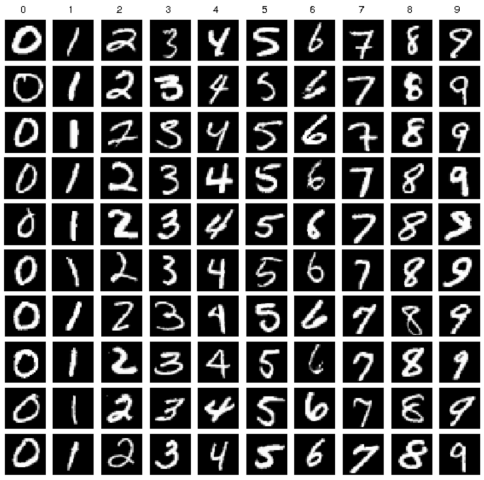

Figure 5 MNIST dataset

- Plot the test accuracy versus iteration number.

The classification rule is defined as the most natural one

$$\underset{j}{\operatorname{argmax}}\ \boldsymbol{e}_j^\top \widehat{\boldsymbol{\Theta}}^\top \boldsymbol{x}_i$$

Solution

The loss function is defined as follow

$$\mathcal{L}(\boldsymbol{\Theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{\ell=1}^{K} 1_{\{y_i=\ell\}} \log\left(\frac{\exp\left(\left\langle \boldsymbol{\theta}^\ell, \boldsymbol{x}_i\right\rangle\right)}{\sum_{j=1}^{K}\exp\left(\left\langle \boldsymbol{\theta}^j, \boldsymbol{x}_i\right\rangle\right)}\right) \tag{24}$$

Then, we can calculate the gradient due to the loss function in the matrix format as follow:

$$\frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}} = \frac{1}{n}\cdot \boldsymbol{X}^\top (\boldsymbol{Z} - \boldsymbol{Y}) \tag{25}$$

where

$$\boldsymbol{X} = \begin{pmatrix} \cdots & \boldsymbol{x}_1 & \cdots \\ \cdots & \boldsymbol{x}_2 & \cdots \\ & \vdots & \\ \cdots & \boldsymbol{x}_n & \cdots \end{pmatrix} \Longrightarrow \boldsymbol{X} \in \mathbb{R}^{n\times d}$$

$$\boldsymbol{Y} = \begin{pmatrix} \cdots & \boldsymbol{y}_1 & \cdots \\ \cdots & \boldsymbol{y}_2 & \cdots \\ & \vdots & \\ \cdots & \boldsymbol{y}_n & \cdots \end{pmatrix} \Longrightarrow \boldsymbol{Y} \in \mathbb{R}^{n\times K} \tag{26}$$

($\boldsymbol{y}_i$ is the one-hot presentation of labels)

$$\boldsymbol{Z}_{ij} = \frac{\exp\left(\left\langle \boldsymbol{x}_i, \boldsymbol{\theta}^j\right\rangle\right)}{\sum_{k=1}^{K}\exp\left(\left\langle \boldsymbol{x}_i, \boldsymbol{\theta}^k\right\rangle\right)} \Longrightarrow \boldsymbol{Z} \in \mathbb{R}^{n\times K}$$

By using Python, we implement the accelerated gradient descent algorithm with $\mu_k = \mu = 0.01$, and set the initial point as origin. Besides, we fixed the iteration number to be 500. The figure of training accuracy versus iteration number and test accuracy versus iteration number is showed as Figure 6.

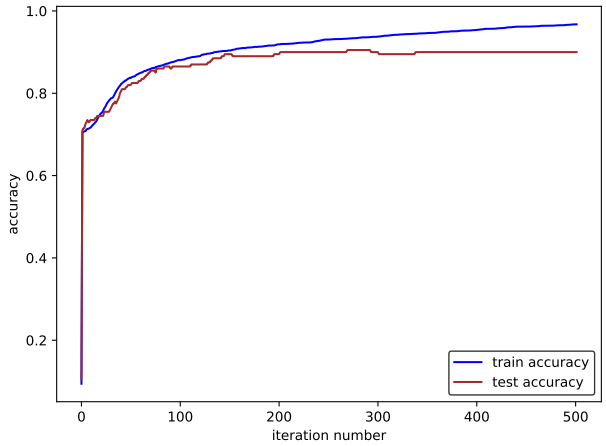

Figure 6 The training accuracy and test accuracy curve

As we can see from Figure 6, the training accuracy and test accuracy increase very fast at the beginning. Then consinuously increases with the increase of iteration number, but the speed is becoming slower. Finally, the train accuracy will reach 96.75% at iteration 500, and the test accuracy will be constant when it reaches to 90%.

The Python code to solve Problem 4 is as follow

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#training data
training_data = pd.read_csv("Q4_MNIST_train.csv", header=0, index_col=0)
x = np.array(training_data.iloc[:, 0])
y = np.array(training_data.iloc[:, 1])
K = 10 #number of class

for i in range(len(x)):
    x_i = x[i][1:-1].split(',')
    for j in range(len(x_i)):
        x_i[j] = float(x_i[j])
    x_i = np.array(x_i).reshape(1,-1)
    if i==0:
        X_train = x_i
    else:
        X_train = np.vstack((X_train, x_i))

for i in range(len(y)):
    l = int(y[i][1:-1])
    y_i = np.zeros(K)
    y_i[l-1] = 1
    if i == 0:
        y_train = y_i
    else:
        y_train = np.vstack((y_train, y_i))

n = X_train.shape[0]
dim = X_train.shape[1]
```

```
#testing data
test_data = pd.read_csv("Q4_MNIST_test.csv", header=0, index_col=0)
x = np.array(test_data.iloc[:, 0])
y = np.array(test_data.iloc[:, 1])

for i in range(len(x)):
    x_i = x[i][1:-1].split(',')
    for j in range(len(x_i)):
        x_i[j] = float(x_i[j])
    x_i = np.array(x_i).reshape(1,-1)
    if i==0:
        X_test = x_i
    else:
        X_test = np.vstack((X_test, x_i))

for i in range(len(y)):
    l = int(y[i][1:-1])
    y_i = np.zeros(K)
    y_i[l-1] = 1
    if i == 0:
        y_test = y_i
    else:
        y_test = np.vstack((y_test, y_i))

#calculate gradient
def cal_grad(theta):
    Z = np.dot(X_train, theta)
    exp_Z = np.exp(Z)
    sum_Z = np.sum(exp_Z, axis=1).reshape(-1, 1)
    vector_Z = exp_Z / sum_Z

    grad = np.dot(X_train.T, vector_Z-y_train)
    grad = grad / n
    return grad

def cal_accuracy(theta, X, y):
    total_num = X.shape[0]
    y_predict = np.argmax(np.dot(X, theta), axis=1)
    y = np.argmax(y, axis=1)
    correct_num = np.sum(y_predict == y)
    return correct_num / total_num

def plot_convergence(train_accuracy, test_accuracy):
    iteration_num = len(train_accuracy)
    iterate = range(iteration_num)

    plt.figure(1)
    plt.plot(iterate, train_accuracy, color="blue", label="train accuracy")
    plt.plot(iterate, test_accuracy, color="brown", label="test accuracy")
    plt.xlabel("iteration number")
    plt.ylabel("accuracy")
    plt.legend(loc="best", edgecolor="black")
    plt.tight_layout()
    plt.savefig("hw_tex/tupian/accuracy.pdf")

def AGD(initial):

    mu = 0.01 # step size
    theta_minus = initial
    theta_k = initial
```

```
    num_iteration = 0

    train_accuracy = []
    test_accuracy = []

    train_accuracy.append(cal_accuracy(theta_k, X_train, y_train))
    test_accuracy.append(cal_accuracy(theta_k, X_test, y_test))
    while num_iteration <= 500:

        beta_k = (num_iteration-1)/(num_iteration+2)
        w = theta_k + beta_k * (theta_k - theta_minus)

        theta_minus = theta_k
        theta_k = w - mu * cal_grad(w)

        train_accuracy.append(cal_accuracy(theta_k, X_train, y_train))
        test_accuracy.append(cal_accuracy(theta_k, X_test, y_test))

        num_iteration  = num_iteration + 1

    plot_convergence(train_accuracy, test_accuracy)
    print("train accuracy at the last step:", train_accuracy[-1])
    print("test accuracy at the last step:", test_accuracy[-1])
theta_initial = np.zeros((dim, K))
AGD(theta_initial)
```