COMP0078: Supervised Learning Coursework 2

Contents

Part I Rademacher Complexity of finite Spaces

1.1

Recall Jensen's inequality, which says for a convex function f,

$$f(\mathbb{E}[Y]) \leq \mathbb{E}[f(Y)]. \tag{1}$$

We can assume $f(x) = e^{\lambda x}$, and we can find that

$$\begin{aligned} & e^{\lambda \mathbb{E}[\bar{X}]} \leq \mathbb{E}[e^{\lambda \bar{X}}], \\ & \xRightarrow{\log} \lambda \mathbb{E}[\bar{X}] \leq \log \mathbb{E}[e^{\lambda \bar{X}}], \\ & \xRightarrow{\lambda > 0} \mathbb{E}[\bar{X}] \leq \frac{1}{\lambda} \log \mathbb{E}[e^{\lambda \bar{X}}]. \end{aligned} \tag{2}$$

■

1.2

By definition $\bar{X} = \max_i X_i$, we have

$$\begin{aligned} & e^{\lambda \bar{X}} \leq \sum_{i=1}^{m} e^{\lambda X_i}, \\ \Rightarrow & \mathbb{E}[e^{\lambda \bar{X}}] \leq \sum_{i=1}^{m} \mathbb{E}[e^{\lambda X_i}]. \end{aligned} \tag{3}$$

Now, we can apply Hoeffding's Lemma with $\mathbb{E}[X_i] = 0$.

$$\begin{aligned} & \mathbb{E}[e^{\lambda X_i}] \leq e^{\lambda^2 (b-a)^2/8}, \\ \Rightarrow & \sum_{i=1}^{m} \mathbb{E}[e^{\lambda X_i}] \leq m e^{\lambda^2 (b-a)^2/8}. \end{aligned} \tag{4}$$

Then, we can combine above two inequalities.

$$\begin{aligned} & \mathbb{E}[e^{\lambda \bar{X}}] \leq m e^{\lambda^2 (b-a)^2/8}, \\ \Rightarrow & \frac{1}{\lambda} \log \mathbb{E}[e^{\lambda \bar{X}}] \leq \frac{1}{\lambda} \log (m e^{\lambda^2 (b-a)^2/8}), \\ \Rightarrow & \frac{1}{\lambda} \log \mathbb{E}[e^{\lambda \bar{X}}] \leq \frac{1}{\lambda} \log m + \lambda \frac{(b-a)^2}{8}. \end{aligned} \tag{5}$$

■

1.3

To summarize what we get from 1.1 and 1.2, we have

$$\mathbb{E}[e^{\lambda \bar{X}}] \leq \frac{1}{\lambda} \log m + \lambda \frac{(b-a)^2}{8}, \tag{6}$$

which is valid for any $\lambda > 0$. We can minimize the RHS by applying AM-GM inequality. Let us assume $A = \frac{1}{\lambda} \log m$ and $B = \lambda \frac{(b-a)^2}{8}$. Then, we have

$$\begin{aligned} & A + B \geq 2\sqrt{AB}, \\ \Rightarrow & \frac{1}{\lambda} \log m + \lambda \frac{(b-a)^2}{8} \geq 2 \sqrt{\frac{1}{\lambda} \log m \times \lambda \frac{(b-a)^2}{8}}, \\ \Rightarrow & \frac{1}{\lambda} \log m + \lambda \frac{(b-a)^2}{8} \geq \frac{b-a}{2} \sqrt{2 \log(m)}, \end{aligned} \tag{7}$$

with equality if $A = B$. In that case, we can find the appropriate λ.

$$\begin{aligned} \frac{1}{\lambda}\log m &= \lambda\frac{(b-a)^2}{8}, \\ \Rightarrow \lambda^2 &= \frac{8\log m}{(b-a)^2}, \\ \Rightarrow \lambda &= \sqrt{\frac{8\log m}{(b-a)^2}}. \end{aligned} \tag{8}$$

Substitute this λ back into the inequality, we have

$$\mathbb{E}\left[\max_{i=1,\ldots,m} X_i\right] \leq \frac{b-a}{2}\sqrt{2\log m}. \tag{9}$$

■

1.4

Let us assume $\bar{X} = \max \frac{1}{n}\sum \sigma_j x_j$. Now, we can apply the relation in 1.1.

$$\mathbb{E}[\bar{X}] \leq \frac{1}{\lambda}\log \mathbb{E}[\exp{(\max \frac{\lambda}{n}\sum_{j=1}^{n}\sigma_j x_j)}]. \tag{10}$$

Then, we have

$$\begin{aligned} \mathbb{E}[\exp{(\max \frac{\lambda}{n}\sum_{j=1}^{n}\sigma_j x_j)}] &\leq \sum_{x\in S}\mathbb{E}[\exp{(\frac{\lambda}{n}\sum_{j=1}^{n}\sigma_j x_j)}], \\ \Rightarrow \mathbb{E}[\exp{(\max \frac{\lambda}{n}\sum_{j=1}^{n}\sigma_j x_j)}] &\leq \sum_{x\in S}\prod_{j=1}^{n}\mathbb{E}[\exp{(\frac{\lambda}{n}\sigma_j x_j)}]. \end{aligned} \tag{11}$$

For $\mathbb{E}[\exp{(\frac{\lambda}{n}\sigma_j x_j)}]$, we can apply Hoeffding's Lemma with $\mathbb{E}[\sigma_j x_j] = 0$ and $X_j \in [-x_j, x_j]$.

$$\mathbb{E}[\exp{(\frac{\lambda}{n}\sigma_j x_j)}] \leq \exp{(\frac{\lambda^2}{8n^2}(2x_j)^2)}. \tag{12}$$

To summary, we get

$$\begin{aligned} \mathbb{E}[\bar{X}] &\leq \frac{1}{\lambda}\log\sum_{x\in S}\prod_{j=1}^{n}\exp{(\frac{\lambda^2}{8n^2}(2x_j)^2)}, \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \frac{1}{\lambda}\log\sum_{x\in S}\exp{(\frac{\lambda^2}{2n^2}\sum_{j=1}^{n}x_j^2)}, \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \frac{1}{\lambda}\log\left(m\max_{x\in S}\exp\left(\frac{\lambda^2}{2n^2}\sum_{j=1}^{n}x_j^2\right)\right), \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \frac{1}{\lambda}\max_{x\in S}\left(\log m + \left(\frac{\lambda^2}{2n^2}\sum_{j=1}^{n}x_j^2\right)\right). \end{aligned} \tag{13}$$

Here we have the similar expression in 1.3. Thus, we can use AM-GM inequality again.

$$\begin{aligned} \mathbb{E}[\bar{X}] &\leq \max_{x\in S}\left(\frac{\log m}{\lambda} + \frac{\lambda}{2n^2}\|x\|_2^2\right), \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \max_{x\in S}2\sqrt{\left(\frac{\log m}{\lambda}\times\frac{\lambda}{2n^2}\|x\|_2^2\right)}, \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \max_{x\in S}\left(\frac{\|x\|_2}{n}\sqrt{2\log m}\right). \end{aligned} \tag{14}$$

■

1.5

The empirical Rademacher complexity measures the maximum correlation of the Rademacher variables σ_i with the outputs $f(x_i)$, maximized over the hypothesis class $\mathcal{H}$:

$$\mathcal{R}_S(\mathcal{H}) = \mathbb{E}_\sigma \left[\sup_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i) \right]. \tag{15}$$

Let us assume $\bar{X} = \sup_{f\in\mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i)$. We need to go through the same process as 1.4. First, we use Jensen's inequality to get:

$$\mathbb{E}[\sup_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i)] \leq \frac{1}{\lambda} \log \mathbb{E}[\exp(\sup_{f \in \mathcal{H}} \frac{\lambda}{n} \sum_{i=1}^{n} \sigma_i f(x_i))]. \tag{16}$$

Then, we have:

$$\begin{aligned} \mathbb{E}[\exp(\sup_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i))] &\leq \sum_{f \in \mathcal{H}} \mathbb{E}[\exp(\frac{\lambda}{n} \sum_{i=1}^{n} \sigma_i f(x_i))], \\ \Rightarrow \mathbb{E}[\exp(\sup_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i))] &\leq \sum_{f \in \mathcal{H}} \prod_{i=1}^{n} \mathbb{E}[\exp(\frac{\lambda}{n} \sigma_i f(x_i))]. \end{aligned} \tag{17}$$

To apply Hoeffding's Lemma, we have $X_i \in [-f(x_i), f(x_i)]$ and $\mathbb{E}[\sigma_i f(x_i)] = 0$. In that case, we have:

$$\mathbb{E}\left[\exp\left(\frac{\lambda}{n} \sigma_i f(x_i)\right)\right] \leq \exp\left(\frac{\lambda^2}{8n^2} (2f(x_i))^2\right). \tag{18}$$

Then, we can combine all the steps together.

$$\begin{aligned} \mathbb{E}[\bar{X}] &\leq \frac{1}{\lambda} \log \sum_{f \in \mathcal{H}} \prod_{i=1}^{n} \exp\left(\frac{\lambda^2}{8n^2} (2f(x_i))^2\right), \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \frac{1}{\lambda} \log \sum_{f \in \mathcal{H}} \exp\left(\frac{\lambda^2}{2n^2} \sum_{i=1}^{n} f(x_i)^2\right), \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \frac{1}{\lambda} \log \left(\sup_{f \in \mathcal{H}} |\mathcal{H}| \exp\left(\frac{\lambda^2}{2n^2} \sum_{i=1}^{n} f(x_i)^2\right)\right), \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \frac{1}{\lambda} \sup_{f \in \mathcal{H}} \left(\log |\mathcal{H}| + \left(\frac{\lambda^2}{2n^2} \sum_{i=1}^{n} f(x_i)^2\right)\right). \end{aligned} \tag{19}$$

Now, we can apply AM-GM inequality again.

$$\begin{aligned} \mathbb{E}[\bar{X}] &\leq \sup_{f \in \mathcal{H}} \left(\frac{\log |\mathcal{H}|}{\lambda} + \frac{\lambda}{2n^2} \sum_{i=1}^{n} f(x_i)^2\right), \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \sup_{f \in \mathcal{H}} 2\sqrt{\left(\frac{\log |\mathcal{H}|}{\lambda} \times \frac{\lambda}{2n^2} \sum_{i=1}^{n} f(x_i)^2\right)}, \\ \Rightarrow \mathbb{E}[\bar{X}] &\leq \sup_{f \in \mathcal{H}} \frac{\sqrt{2 \log |\mathcal{H}|}}{n} \sqrt{\sum_{i=1}^{n} f(x_i)^2}. \end{aligned} \tag{20}$$

Thus, we find the upper bound of empirical Rademacher complexity:

$$\mathcal{R}_S(\mathcal{H}) \leq \sup_{f \in \mathcal{H}} \frac{\sqrt{2 \log |\mathcal{H}|}}{n} \sqrt{\sum_{i=1}^{n} f(x_i)^2}. \tag{21}$$

■

Part II Bayes Decision Rule and Surrogate Approaches

2.1

The quality of a classification rule can be measured by the misclassification error

$$R(c) = \mathbb{P}_{(x,y)\sim\rho}(c(x) \neq y). \tag{22}$$

The probability of an event E (here, $c(x) \neq y$) under a distribution $\rho(x, y)$ is given by:

$$\mathbb{P}_{(x,y)\sim\rho}(E) = \int_E d\rho(x, y). \tag{23}$$

According to the condition in the question, the misclassification can be defined by a 0-1 loss function $\mathbf{1}_{y\neq y'}$. In that case, only $y \neq y'$ situations are counted. Thus, we can express our probability as:

$$\mathbb{P}_{(x,y)\sim\rho}(c(x) \neq y) = \int_{\mathcal{X}\times\mathcal{Y}} \mathbf{1}_{c(x)\neq y}(x, y) d\rho(x, y), \tag{24}$$

where $\mathbf{1}_{c(x)\neq y}(x, y)$ is written as:

$$\mathbf{1}_{c(x)\neq y}(x, y) = \begin{cases} 1 & \text{if } c(x) \neq y \\ 0 & \text{if } c(x) = y \end{cases} \tag{25}$$

Finally, we show that

$$R(c) = \int_{\mathcal{X}\times\mathcal{Y}} \mathbf{1}_{c(x)\neq y} d\rho(x, y). \tag{26}$$

■

2.2

To find the minimizer, we can follow the hints in the question:

$$\mathcal{E}(f) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x), y) d\rho(x, y) = \int_{\mathcal{X}} \left(\int_{\mathcal{Y}} \ell(f(x), y) d\rho(y \mid x) \right) d\rho_{\mathcal{X}}(x). \tag{27}$$

where we can take the derivative of the empirical risk with respect to the function f.

$$\begin{aligned} &\frac{\partial \mathcal{E}(f)}{\partial f} = 0, \\ \Rightarrow &\int_{\mathcal{Y}} \frac{\partial \ell(f(x), y)}{\partial f} d\rho(y \mid x) = 0, \\ \Rightarrow &\sum_{\mathcal{Y}\in\{-1,1\}} \frac{\partial \ell(f(x), y)}{\partial f} \rho(y \mid x) = 0. \end{aligned} \tag{28}$$

We can use the last line to find the minimizers for different loss functions.

a) squared loss

Substitute the squared loss function into the equation we derived above.

$$\begin{aligned} &\sum_{\mathcal{Y}\in\{-1,1\}} \frac{\partial (f(x) - y)^2}{\partial f} \rho(y \mid x) = 0, \\ \Rightarrow &\sum_{\mathcal{Y}\in\{-1,1\}} 2(f(x) - y)\rho(y \mid x) = 0, \\ \Rightarrow &(f(x) - 1)\rho(y = 1 \mid x) + (f(x) + 1)\rho(y = -1 \mid x) = 0, \\ \Rightarrow &f(x) = \frac{\rho(y = 1 \mid x) - \rho(y = -1 \mid x)}{\rho(y = 1 \mid x) + \rho(y = -1 \mid x)}, \end{aligned} \tag{29}$$

where we notice that $\rho(y = 1 \mid x) + \rho(y = -1 \mid x) = 1$. Thus, we find the minimizer for squared loss function.

$$f_* = \rho(y = 1 \mid x) - \rho(y = -1 \mid x). \tag{30}$$

b) exponential loss

Substitute the exponential loss function into the equation we derived above.

$$\begin{aligned}
&\sum_{\mathcal{Y}\in\{-1,1\}} \frac{\partial \exp(-yf(x))}{\partial f}\rho(y \mid x) = 0,\\
\Rightarrow &\sum_{\mathcal{Y}\in\{-1,1\}} -y\exp(-yf(x))\rho(y \mid x) = 0,\\
\Rightarrow &-\exp(-f(x))\rho(y=1 \mid x) + \exp(f(x))\rho(y=-1 \mid x) = 0,\\
\Rightarrow &\exp(2f(x)) = \frac{\rho(y=1 \mid x)}{\rho(y=-1 \mid x)}.
\end{aligned} \tag{31}$$

where we take logarithm on both sides. Thus, we find the minimizer for exponential loss function.

$$f_* = \frac{1}{2}\log\frac{\rho(y=1 \mid x)}{\rho(y=-1 \mid x)}. \tag{32}$$

c) logistic loss

Substitute the logistic loss function into the equation we derived above.

$$\begin{aligned}
&\sum_{\mathcal{Y}\in\{-1,1\}} \frac{\partial \log(1+\exp(-yf(x)))}{\partial f}\rho(y \mid x) = 0,\\
\Rightarrow &\sum_{\mathcal{Y}\in\{-1,1\}} \frac{-y}{e^{yf(x)}+1}\rho(y \mid x) = 0,\\
\Rightarrow &\frac{-1}{e^{f(x)}+1}\rho(y=1 \mid x) + \frac{1}{e^{-f(x)}+1}\rho(y=-1 \mid x) = 0,\\
\Rightarrow &\frac{e^{f(x)}+1}{e^{-f(x)}+1} = \frac{\rho(y=1 \mid x)}{\rho(y=-1 \mid x)}.\\
\Rightarrow &e^{f(x)} = \frac{\rho(y=1 \mid x)}{\rho(y=-1 \mid x)}.
\end{aligned} \tag{33}$$

where we take logarithm on both sides. Thus, we find the minimizer for logistic loss function.

$$f_* = \log\frac{\rho(y=1 \mid x)}{\rho(y=-1 \mid x)}. \tag{34}$$

d) hinge loss

For hinge loss, we cannot differentiate it at start. In that case, we can discuss it in different conditions. Substitute the hinge loss function into the equation without differentiation.

$$\begin{aligned}
&\sum_{\mathcal{Y}\in\{-1,1\}} \max(0, 1-yf(x))\rho(y \mid x),\\
\Rightarrow &\max(0,1-f(x))\rho(y=1 \mid x) + \max(0,1+f(x))\rho(y=-1 \mid x),
\end{aligned} \tag{35}$$

where we can split it into three different conditions.

$$\begin{array}{rl}
(1+f(x))\rho(y=-1 \mid x) & f(x) \geq 1\\
(1-f(x))\rho(y=1 \mid x) + (1+f(x))\rho(y=-1 \mid x) & -1 < f(x) < 1\\
(1-f(x))\rho(y=1 \mid x) & f(x) \leq -1
\end{array} \tag{36}$$

Now, we take the derivative with respect to $f(x)$.

$$\begin{array}{rl}
\rho(y=-1 \mid x) & f(x) \geq 1\\
-\rho(y=1 \mid x) + \rho(y=-1 \mid x) & -1 < f(x) < 1\\
-\rho(y=1 \mid x) & f(x) \leq -1
\end{array} \tag{37}$$

The form of gradient show that the global minima occurs at $f(x) = \pm 1$. This can be deduced from the situation 1 and 3. Substitute $f(x) = \pm 1$ back to our expression for empirical risk.

$$\mathcal{E}(f_*) \propto \min(2\underbrace{\rho(y=-1 \mid x)}_{f_*=1}, 2\underbrace{\rho(y=1 \mid x)}_{f_*=-1}). \tag{38}$$

This shows that the hinge loss minimizer $f_*(x)$ depends on the balance of $\rho(y=1 \mid x)$ and $\rho(y=-1 \mid x)$. Thus, we can express it as

$$f_*(x) = \begin{cases} 1 & \text{if } \rho(y=1 \mid x) \geq \rho(y=-1 \mid x) \\ -1 & \text{if } \rho(y=1 \mid x) < \rho(y=-1 \mid x) \end{cases} \tag{39}$$

The critical point is where $\rho(y=1 \mid x) = \rho(y=-1 \mid x) = \frac{1}{2}$, indicating uncertainty in the labels. In that case, we can choose the equality belong to either $f_* = 1$ or $f_* = -1$.

2.3

From the definition, we can express $R(c)$ as:

$$R(c) = \int_{\mathcal{X}\times\mathcal{Y}} \mathbf{1}_{c(x)\neq y} d\rho(x,y) = \int_{x\in\mathcal{X}} \left(\int_{y\in\mathcal{Y}} \mathbf{1}_{c(x)\neq y} d\rho(y \mid x) \right) d\rho_{\mathcal{X}}(x). \tag{40}$$

This is the misclassification error. To minimize this expression, we can consider an optimal situation that every label is correct. In that case, we choose $c(x)$ to classify x as the more likely label:

$$c_*(x) = \begin{cases} 1 & \text{if } \rho(y=1 \mid x) \geq \rho(y=-1 \mid x) \\ -1 & \text{if } \rho(y=1 \mid x) < \rho(y=-1 \mid x). \end{cases} \tag{41}$$

We can also regard $\mathbf{1}_{c(x)\neq y}$ as a loss function. Surprisingly, it has the similar function as the hinge loss where we derived in 2.2 (d). It gives us the same results for $c_*(x)$.

2.4

We need to find a map $d : \mathbb{R} \to \{-1, 1\}$ such that $R(c_*(x)) = R(d(f_*(x)))$ where f_* is the corresponding minimizer for the loss function in 2.2. Combining what we prove in 2.3, this means $d(f_*(x))$ correctly recovers the Bayes optimal decision rule. Now, let us go through different loss functions.

a) squared loss For squared loss, the minimizer is

$$f_* = \rho(y=1 \mid x) - \rho(y=-1 \mid x), \tag{42}$$

To find the suitable mapping, we can check the probability.

$$\begin{aligned} f_*(x) \geq 0 \quad \text{if} \quad \rho(y=1 \mid x) \geq \rho(y=-1 \mid x), \\ f_*(x) < 0 \quad \text{if} \quad \rho(y=1 \mid x) < \rho(y=-1 \mid x). \end{aligned} \tag{43}$$

In that case, we can define the mapping to $\{-1, 1\}$ based on the sign of $f_*(x)$.

$$d(f_*(x)) = \begin{cases} 1 & \text{if } f_*(x) \geq 0 \\ -1 & \text{if } f_*(x) < 0. \end{cases} \tag{44}$$

b) exponential loss

The minimizer for exponential loss function is

$$f_* = \frac{1}{2} \log \frac{\rho(y=1 \mid x)}{\rho(y=-1 \mid x)}, \tag{45}$$

which has the following property.

$$\begin{aligned} f_*(x) \geq 0 \quad \text{if} \quad \rho(y=1 \mid x) \geq \rho(y=-1 \mid x), \\ f_*(x) < 0 \quad \text{if} \quad \rho(y=1 \mid x) < \rho(y=-1 \mid x). \end{aligned} \tag{46}$$

Again, we can find the mapping as

$$d(f_*(x)) = \begin{cases} 1 & \text{if } f_*(x) \geq 0 \\ -1 & \text{if } f_*(x) < 0. \end{cases} \tag{47}$$

c) logistic loss

The minimizer for logistic loss is similar to the one for exponential loss.

$$f_* = \log \frac{\rho(y=1 \mid x)}{\rho(y=-1 \mid x)}. \tag{48}$$

We can also map this based on the probability like what we did before.

$$d(f_*(x)) = \begin{cases} 1 & \text{if } f_*(x) \geq 0 \\ -1 & \text{if } f_*(x) < 0. \end{cases} \tag{49}$$

d) hinge loss

The minimizer for hinge loss is

$$f_*(x) = \begin{cases} 1 & \text{if } \rho(y=1 \mid x) \geq \rho(y=-1 \mid x), \\ -1 & \text{if } \rho(y=1 \mid x) < \rho(y=-1 \mid x), \end{cases} \tag{50}$$

which can be directly mapped to $\{-1, 1\}$ based on the sign of $f_*(x)$.

$$d(f_*(x)) = \begin{cases} 1 & \text{if } f_*(x) \geq 0 \\ -1 & \text{if } f_*(x) < 0. \end{cases} \tag{51}$$

Summary

From above calculations, we find that all the surrogate frameworks in problem 2.2 are Fisher consistent. Furthermore, we can write a general expression of mapping as:

$$d(f_*(x)) = \begin{cases} 1 & \text{if } f_*(x) \geq 0 \\ -1 & \text{if } f_*(x) < 0. \end{cases} \tag{52}$$

2.5

2.5.1

Based on the definition of misclassification error, we can express $R(\operatorname{sign}(f))$ as:

$$R(\operatorname{sign}(f)) = \int_{x \in \mathcal{X}} \left(\int_{y \in \mathcal{Y}} \mathbf{1}_{\operatorname{sign}(f) \neq y} d\rho(y \mid x) \right) d\rho_{\mathcal{X}}(x). \tag{53}$$

At the same time, the error difference is zero if $\operatorname{sign}(f) = \operatorname{sign}(f_*)$. In that case, we can simplify the integration for non-zero error difference only. It can be defined as $\mathcal{X}_f = \{x \in \mathcal{X} \mid \operatorname{sign}(f(x)) \neq \operatorname{sign}(f_*(x))\}$. Thus, we can express the error difference as:

$$\begin{aligned} |R(\operatorname{sign}(f)) - R(\operatorname{sign}(f_*))| &= \left| \int_{x \in \mathcal{X}_f} \left(\int_{y \in \mathcal{Y}} \mathbf{1}_{\operatorname{sign}(f) \neq y} - \mathbf{1}_{\operatorname{sign}(f_*) \neq y} d\rho(y \mid x) \right) d\rho_{\mathcal{X}}(x) \right|, \\ \Rightarrow |R(\operatorname{sign}(f)) - R(\operatorname{sign}(f_*))| &= \int_{x \in \mathcal{X}_f} \left| \left(\sum_{y \in \mathcal{Y}} \left(\mathbf{1}_{\operatorname{sign}(f) \neq y} - \mathbf{1}_{\operatorname{sign}(f_*) \neq y} \right) \rho(y \mid x) \right) \right| d\rho_{\mathcal{X}}(x). \end{aligned} \tag{54}$$

This can be further expanded as

$$\int_{x \in \mathcal{X}_f} \left| \left(\mathbf{1}_{\operatorname{sign}(f) \neq 1} - \mathbf{1}_{\operatorname{sign}(f_*) \neq 1} \right) \rho(y=1 \mid x) + \left(\mathbf{1}_{\operatorname{sign}(f) \neq -1} - \mathbf{1}_{\operatorname{sign}(f_*) \neq -1} \right) \rho(y=-1 \mid x) \right| d\rho_{\mathcal{X}}(x). \tag{55}$$

Due to the condition of our question, the above equation only has two values:

$$\begin{aligned} &\int_{x \in \mathcal{X}_f} |\rho(y=1 \mid x) - \rho(y=-1 \mid x)| \, d\rho_{\mathcal{X}}(x) \text{ if } \operatorname{sign}(f) = -1 \text{ and } \operatorname{sign}(f_*) = 1, \\ &\int_{x \in \mathcal{X}_f} |-\rho(y=1 \mid x) + \rho(y=-1 \mid x)| \, d\rho_{\mathcal{X}}(x) \text{ if } \operatorname{sign}(f) = 1 \text{ and } \operatorname{sign}(f_*) = -1. \end{aligned} \tag{56}$$

Recall the expression we derived in problem 2.2. The minimizer for lest square is

$$f_* = \rho(y = 1 \mid x) - \rho(y = -1 \mid x), \tag{57}$$

which gives the same value for our error difference. Thus, we have

$$|R(\text{sign}(f)) - R(\text{sign}(f_*))| = \int_{\mathcal{X}_f} |f_*(x)| \, d\rho_{\mathcal{X}}(x). \tag{58}$$

■

2.5.2

Let us start with the first inequality. In $\mathcal{X}_f$, we know that the signs for f and f_* are different. This further implies:

$$|f_*(x)| \leq |f_*(x) - f(x)|, \tag{59}$$

as $\text{sign}(f(x)) \neq \text{sign}(f_*(x))$, the "distance" between them (in terms of sign disagreement) is determined by:

$$|f_*(x)| + |f(x)|, \tag{60}$$

which is the sum of their magnitudes. Taking the integral of both sides with respect to $\rho_{\mathcal{X}}(x)$, we get:

$$\int_{\mathcal{X}_f} |f_*(x)| \, d\rho_{\mathcal{X}}(x) \leq \int_{\mathcal{X}_f} |f_*(x) - f(x)| \, d\rho_{\mathcal{X}}(x). \tag{61}$$

For the second inequality, we apply the Cauchy-Schwarz inequality to the integral:

$$\left(\int_{\mathcal{X}_f} |f_*(x) - f(x)| \, d\rho_{\mathcal{X}}(x)\right)^2 \leq \int_{\mathcal{X}_f} 1^2 d\rho_{\mathcal{X}}(x) \cdot \int_{\mathcal{X}_f} |f_*(x) - f(x)|^2 \, d\rho_{\mathcal{X}}(x). \tag{62}$$

Here we need to extend the domain from $\mathcal{X}_f$ to $\mathcal{X}$ which will only increase the value of integral. Thus, we have

$$\begin{aligned} &\int_{\mathcal{X}_f} |f_*(x) - f(x)| \, d\rho_{\mathcal{X}}(x) \leq \sqrt{\int_{\mathcal{X}} 1^2 d\rho_{\mathcal{X}}(x) \cdot \int_{\mathcal{X}} |f_*(x) - f(x)|^2 \, d\rho_{\mathcal{X}}(x)}, \\ \Rightarrow &\int_{\mathcal{X}_f} |f_*(x) - f(x)| \, d\rho_{\mathcal{X}}(x) \leq \sqrt{\int_{\mathcal{X}} |f_*(x) - f(x)|^2 \, d\rho_{\mathcal{X}}(x)}, \\ \Rightarrow &\int_{\mathcal{X}_f} |f_*(x) - f(x)| \, d\rho_{\mathcal{X}}(x) \leq \sqrt{\mathbb{E}\left(|f(x) - f_*(x)|^2\right)}. \end{aligned} \tag{63}$$

Combining two inequalities together, we then prove

$$\int_{\mathcal{X}_f} |f_*(x)| \, d\rho_{\mathcal{X}}(x) \leq \int_{\mathcal{X}_f} |f_*(x) - f(x)| \, d\rho_{\mathcal{X}}(x) \leq \sqrt{\mathbb{E}\left(|f(x) - f_*(x)|^2\right)}. \tag{64}$$

■

2.5.3

The expected surrogate risk is:

$$\mathcal{E}(f) = \int_{\mathcal{X}} \int_{\mathcal{Y}} (f(x) - y)^2 \rho(y \mid x) d\rho_{\mathcal{X}}(x). \tag{65}$$

Then, we can express the difference as

$$\begin{aligned} &\mathcal{E}(f) - \mathcal{E}(f_*) = \int_{\mathcal{X}} \int_{\mathcal{Y}} \left((f(x) - y)^2 - (f_*(x) - y)^2\right) d\rho(y \mid x) d\rho_{\mathcal{X}}(x), \\ \Rightarrow &\mathcal{E}(f) - \mathcal{E}(f_*) = \int_{\mathcal{X}} \int_{\mathcal{Y}} \left(f(x)^2 - f_*(x)^2 - 2y(f(x) - f_*(x))\right) d\rho(y \mid x) d\rho_{\mathcal{X}}(x), \\ \Rightarrow &\mathcal{E}(f) - \mathcal{E}(f_*) = \int_{\mathcal{X}} \left(f(x)^2 - f_*(x)^2 - 2(f(x) - f_*(x)) \sum_{\mathcal{Y}} y\rho(y \mid x)\right) d\rho_{\mathcal{X}}(x). \end{aligned} \tag{66}$$

Here, we notice that $f_*(x) = \sum_{\mathcal{Y}} y\rho(y \mid x)$. Thus, we can further express above equation as

$$\begin{aligned}
&\mathcal{E}(f) - \mathcal{E}(f_*) = \int_{\mathcal{X}} \left(f(x)^2 - f_*(x)^2 - 2(f(x) - f_*(x))f_*(x)\right) d\rho_{\mathcal{X}}(x), \\
\Rightarrow &\mathcal{E}(f) - \mathcal{E}(f_*) = \int_{\mathcal{X}} \left(f(x)^2 - 2f(x)f_*(x) + f_*(x)^2\right) d\rho_{\mathcal{X}}(x), \\
\Rightarrow &\mathcal{E}(f) - \mathcal{E}(f_*) = \int_{\mathcal{X}} \left(f(x) - f_*(x)\right)^2 d\rho_{\mathcal{X}}(x), \\
\Rightarrow &\mathcal{E}(f) - \mathcal{E}\left(f_*\right) = \mathbb{E}\left(\left|f(x) - f_*(x)\right|^2\right).
\end{aligned} \tag{67}$$

■

Part III Kernel perceptron (Handwritten Digit Classification)

3.1. Introduce 'One-versus-Rest'

One-vs-rest (OvR for short, also referred to as One-vs-All or OvA) is a method for using binary classification for multi-class classification. From the code, the general approach for One-vs-Rest classification with a kernel perceptron can be outlined as follows:

Step	**Description**
Input	$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\} \in (\mathbb{R}^n, \{1, \ldots, k\})^m$ where k is the number of classes.
Kernel Function	Polynomial Kernel: $K_d(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x} \cdot \boldsymbol{z})^d$.
Initialization	Initialize the weight matrix: $\boldsymbol{W} = \boldsymbol{0}$, where $\boldsymbol{W} \in \mathbb{R}^{k \times m}$.
Prediction	For a given instance $\boldsymbol{x}_t$, compute the confidence scores: $$\kappa_j(\boldsymbol{x}_t) = \sum_{i=1}^{m} \alpha_{j,i} K(\boldsymbol{x}_i, \boldsymbol{x}_t), \quad \forall j \in \{1, \ldots, k\}.$$ Predict the class: $$\hat{y}_t = \arg\max_j \kappa_j(\boldsymbol{x}_t).$$
Update Rule	For each class j: $$\text{If } \hat{y}_t \neq y_t :$$ $$\alpha_{y,t} \leftarrow \alpha_{y,t} + 1.$$ $$\alpha_{\hat{y}_t,t} \leftarrow \alpha_{\hat{y}_t,t} - 1.$$
Training Loop	For each epoch: 1. For each training sample $(\boldsymbol{x}_t, y_t)$: • Perform Prediction and Update steps. 2. Count the number of misclassifications.
Testing	For a test instance $\boldsymbol{x}_t$: 1. Compute the confidence scores: $\kappa_j(\boldsymbol{x}_t)$, $\forall j$. 2. Predict the class: $\hat{y}_t = \arg\max_j \kappa_j(\boldsymbol{x}_t)$. 3. Compute test error rate as the number of incorrect predictions divided by the test size.
Output	Final weight matrix $\boldsymbol{W}$, training error, and test error rates.

Table 1: One-versus-Rest Kernel Perceptron (Training and Testing Process)

The method trains k binary classifiers(in the code, k=3), where k is the number of classes. For each class j, a separate perceptron is trained to distinguish class j (labeled as $+1$) from all other classes (labeled as -1). Specifically in class j, we add 1 for $j = y$ and minus 1 for $j \neq \hat{y}$. During training, the perceptron iterates through all training samples, computes a confidence score for each class, and updates the weights for misclassified points. Specifically, if the confidence score for the true class is not positive, the weight associated with that class is increased, while weights for incorrect predictions are penalized. In the prediction phase, the confidence scores for all k classifiers are calculated for a given test instance, and the class with the maximum confidence score is selected as the final prediction. The model's performance is evaluated by measuring the test error rate, defined as the ratio of misclassified samples to the total test set size.

3.2. Implement OvR

Following the table in question and the one we proposed in 3.1, we can implement the 'One-versus-Rest'. To figure the update procure of the weight, we can start with the initialization of the weight. It is a matrix of $\#Class \times \#Data$ with the element being zero at start. As we go through the column i, we can compare the prediction value with the real value by $\hat{y}_t = \arg\max_i \sum_{i=1}^{m} \alpha_{i,j} K(\boldsymbol{x}_j, \boldsymbol{x}_t),$. For the right prediction, there is no update. For the mistakes, we can update our classifier via increasing the weight of the right label and decreasing the weights of the rest. For example, we update our weight in ith step. The summation is calculated as:

$$\boldsymbol{W} \cdot K(x_i, \cdot) = \begin{array}{c} \begin{array}{cccc} & i & i+1 & \end{array} \\ \begin{pmatrix} \cdots & 1 & 0 & \cdots \\ \cdots & 1 & 0 & \cdots \\ \cdots & -1 & 0 & \cdots \\ \cdots & \vdots & \vdots & \cdots \\ \cdots & 0 & 0 & \cdots \end{pmatrix} \end{array} \times \begin{array}{c} i \\ \begin{pmatrix} K(x_i, x_1) \\ K(x_i, x_2) \\ \cdots \\ \cdots \\ K(x_i, x_n) \end{pmatrix} \end{array}. \tag{68}$$

Then, we take the maximum value of the summation as our prediction. For the right value, we do not update our weight matrix. For the wrong prediction, we increase the correct weight and decrease the predicted weight. This can be illustrated as

$$\boldsymbol{W} = \begin{array}{c} \begin{array}{cccc} & i & i+1 & \end{array} \\ \begin{pmatrix} \cdots & 1 & 0 & \cdots \\ \cdots & 1 & 0 & \cdots \\ \cdots & -1 & 0 & \cdots \\ \cdots & \vdots & \vdots & \cdots \\ \cdots & 0 & 0 & \cdots \end{pmatrix} \end{array} \rightarrow \begin{array}{c} \begin{array}{cccc} & i & i+1 & \end{array} \\ \begin{pmatrix} \cdots & 2 & 0 & \cdots \\ \cdots & 1 & 0 & \cdots \\ \cdots & -1 & 0 & \cdots \\ \cdots & \vdots & \vdots & \cdots \\ \cdots & -1 & 0 & \cdots \end{pmatrix} \end{array}, \tag{69}$$

where we assign the blue position as right value and the position as predicted value. Since it is a mistake, we add 1 on blue and minus 1 on red to update.

To find the optimal epochs number, we perform 20 runs of One-versus-Rest method with each run contains 20 maximum epochs. The algorithm will break if the change in error is small enough. Here is the last run of our method and we plot the error against epoch for each degree. It is clear that most algorithm converges between 8 to 16 epochs.

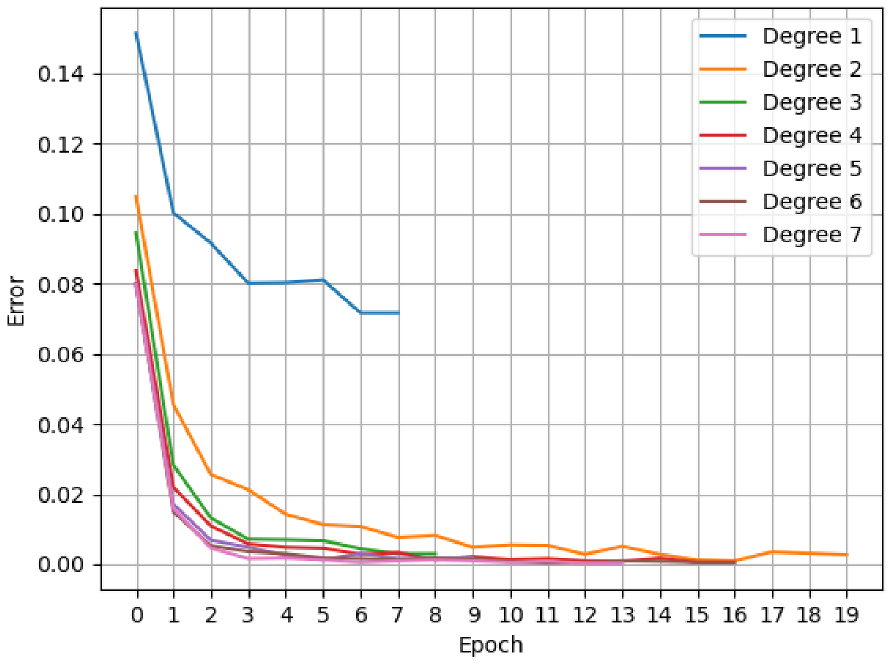

Figure 1: The Error vs Epoch in the last run

In that case, we take the mean value and exact 1 epoch which is 13 for the rest test.

3.3. Basic results

Here is the result for 20 runs with fixed 13 epoch.

Degree	Train Error(%)	Test Error(%)
1	6.0151 ± 0.2258	9.2339 ± 1.7044
2	0.4121 ± 0.0989	3.8602 ± 0.5724
3	0.1889 ± 0.0723	3.3226 ± 0.4573
4	0.0874 ± 0.0565	3.3306 ± 0.4949
5	0.0598 ± 0.0418	3.1667 ± 0.4157
6	0.0424 ± 0.0346	3.0591 ± 0.2757
7	0.0430 ± 0.0367	2.9355 ± 0.3196

Table 2: Train and Test Errors for Different Degrees

There is no significant rise in test error as the degree increases, suggesting the kernel perceptron handles higher polynomial degrees well without overfitting. Degree 7 achieves the best test error (2.9355%). However, degrees 3–5 also provide competitive performance with slightly higher test errors but much simpler models.

3.4. Cross-validation results

We perform 20 runs with each run splitting the train data set into 5 parts. Then, we perform 5-fold cross-validation to find the best degree for our kernel. Here are the results.

Run	d	Train Error(%)	Test Error(%)	Run	d	Train Error(%)	Test Error(%)
1	4	0.1008	4.1935	11	6	0.0504	4.1398
2	5	0.1848	3.9785	12	7	0.0672	4.1935
3	5	0.1680	4.1398	13	4	0.0336	4.1398
4	6	0.0336	3.8710	14	5	0.1512	4.3011
5	5	0.0000	3.8710	15	3	0.0840	3.9785
6	5	0.1176	4.1398	16	5	0.1176	3.8172
7	5	0.0000	3.8172	17	4	0.0504	3.8710
8	4	0.1008	4.0323	18	4	0.1008	3.6022
9	3	0.3193	4.0323	19	7	0.0000	3.7097
10	5	0.0672	3.9247	20	3	0.1008	3.9785

Table 3: Train and Test Errors for Optimal Degree in 20 runs

And we can find the mean value of the optimal degree and the overall test error as:

$$\text{Degree: } 4.75 \pm 1.1347 \qquad \text{Test error(\%): } 3.9866 \pm 0.1744 \tag{70}$$

Our final result 4.75 is consistent with our discussion in 3.3 and the test error is acceptable. Thus, we can choose from 3 to 7 for our kernel function.

3.5. Confusion matrix

Here we use the definition given in the question to calculate the confusion matrix. During test, we find that a specific digit might not appear in the test. In that case, we replace the NaN value with the zeros.

Following 3.4, we calculate the confusion matrix with the optimal degree in each run and take the mean value. Here is the plot.

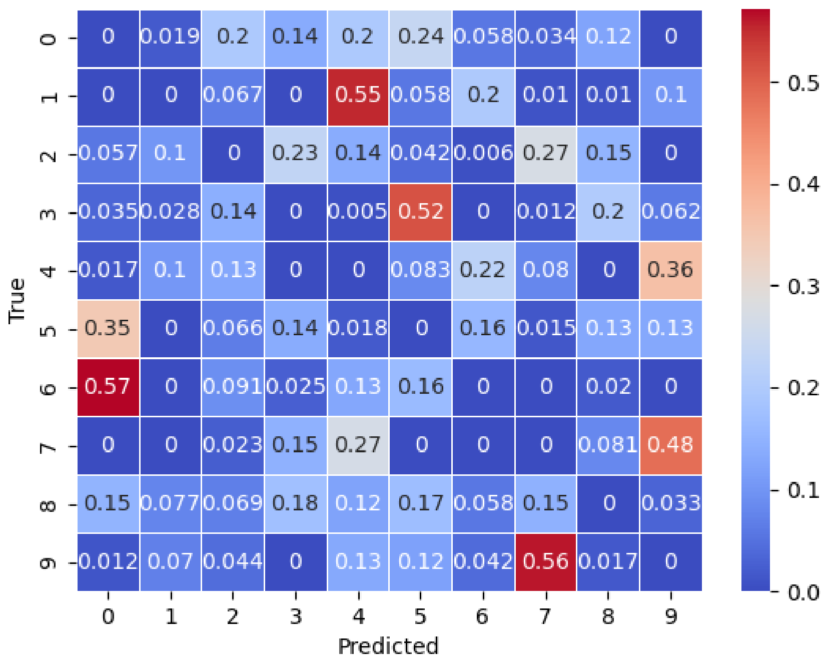

Figure 2: confusion Matrix

We can find from the plot that digits 1, 5, 6 and 9 are more likely to be misclassified in this test set. Digit 1 is often recognised as 4. Digits 5 and 6 are often recognised as 0. Digit 3 is recognised as 5 and digit 7 is recognised as 9.

3.6. Hard-to-predict samples

In this study, we conducted 50 runs on our dataset using a kernel perceptron model with $d = 5$ as the degree of the polynomial kernel. In each run, we count number of times digit a was mistaken for digit b. This repeated procedure aims to improve accuracy. By ranking the numbers from the counting process, we find the five hardest-to-predict samples.

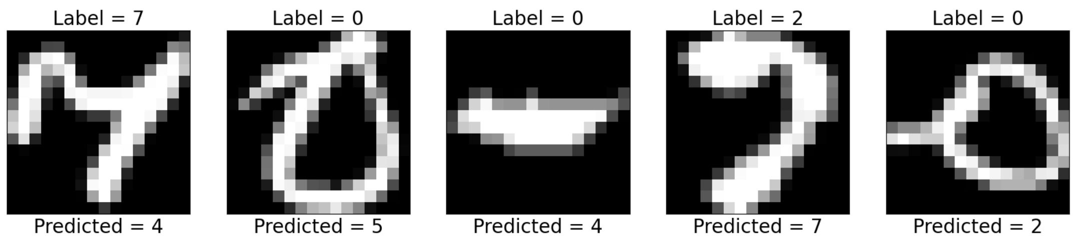

Figure 3: Top 5 hardest to predict images in the dataset

From the above figure, it is not surprising that we gained some insights from the confusion matrix. At the same time, it becomes evident that certain digits are particularly challenging to classify correctly. These misclassifications might arise due to the following reasons:

- The true label itself is incorrect.
- The features of the sample are inherently ambiguous or overlapping with those of other classes, making classification difficult.

- The dataset lacks sufficient diversity or representation for specific digit classes, leading to biased predictions.
- The kernel function or the model parameters might not capture the optimal decision boundaries for these challenging cases.

3.7. Gaussian kernel

(a) Find suitable set S

To find a suitable set for c value, we need to check the property of the Gaussian kernel first. In this question, we have

$$K_c(\boldsymbol{p}, \boldsymbol{q}) = e^{-c\|\boldsymbol{p}-\boldsymbol{q}\|^2}, \tag{71}$$

where the input values p and q are the feature of the image x_i and x_j. Due to the update algorithm, we want the closed x_i and x_j with large kernel value and far away pairs with small kernel value. In that case, we should choose c value between 0 and 1. In our first attempt, we start with 2^i for i from -10 to 0. To match with the previous experiment, we split this into 7 parts. For efficiency, we take 1 run for each value of c and track the test error. Here are the results.

c	train error(%)	test error(%)
0.0010	3.0250	7.0968
0.0031	0.3630	5.5914
0.0098	0.0538	3.2258
0.0312	0.0269	3.8172
0.0992	0.0000	4.6774
0.3150	0.0000	6.1828
1.0000	0.0000	5.9677

(a) Set S and its error for the first attempt

c	train error(%)	test error(%)
0.0028	0.0807	4.3011
0.0041	0.1479	3.4946
0.0062	0.1479	3.6559
0.0093	0.0269	3.3333
0.0139	0.0807	3.4946
0.0209	0.0000	2.8495
0.0312	0.0269	3.4946

(b) Set S and its error for the second attempt

Table 4: Set S and its error

From Table 4(a), we find that the test error falls when c decreases at large c. However, the test error increases when c approaches zero. We gain a minimum at $c = 0.0098$. In that case, we should go for a second run with a different range from 0.0031 to 0.0312. Table 4(b) satisfies our exception as we find a smaller error when we further split the range. In later question, we will use the range from 0.0093 to 0.0312 as set S.

(b) Basic results for Gaussian kernel

Here is the basic results for

c	Train Error	Test Error
0.0110	0.0558 ± 0.0406	3.4194 ± 0.2872
0.0131	0.0471 ± 0.0235	3.4462 ± 0.2993
0.0156	0.0504 ± 0.0318	3.3172 ± 0.2319
0.0186	0.0430 ± 0.0302	3.4274 ± 0.2486
0.0221	0.0376 ± 0.0178	3.4113 ± 0.2754
0.0263	0.0255 ± 0.0147	3.7016 ± 0.3734
0.0312	0.0222 ± 0.0172	3.7742 ± 0.2745

Table 5: Basic results for Gaussian kernel

In the basic results, we have a optimal c value at 0.0156 with the lowest test error.

(c) Cross-validation results for Gaussian kernel

Here is the cross-validation results for Gaussian kernel.

And we can find the mean value of the optimal c value and the overall test error as:

$$\text{C value: } 0.0146 \pm 0.0030 \quad \text{Test error(\%): } 3.8253 \pm 0.2436 \tag{72}$$

This result is consistent with what we get in (b) as 0.0156 lies in the range of the best c value.

Run	**c**	**Train Error(%)**	**Test Error(%)**	**Run**	**d**	**Train Error(%)**	**Test Error(%)**
1	0.0156	0.0672	3.8172	11	0.0110	0.0672	3.6022
2	0.0156	0.0000	3.6559	12	0.0186	0.0000	3.9785
3	0.0156	0.0504	3.9785	13	0.0186	0.0000	3.9247
4	0.0131	0.0336	3.5484	14	0.0186	0.0336	4.1935
5	0.0131	0.1176	3.7634	15	0.0186	0.1008	3.8710
6	0.0131	0.0000	3.6559	16	0.0110	0.0336	4.1398
7	0.0110	0.0336	4.0860	17	0.0131	0.0840	4.1398
8	0.0131	0.0672	3.3871	18	0.0131	0.0000	3.8710
9	0.0186	0.0504	3.76343	19	0.0186	0.0504	4.0323
10	0.0110	0.0504	3.8172	20	0.0110	0.0336	3.2796

Table 6: Train and Test Errors for Optimal c value in 20 runs

(d) Gaussian kernel versus Polynomial kernel

The polynomial kernel achieved the best test error (2.9355%) at $d = 7$, while the Gaussian kernel's lowest test error was slightly higher ($3.3172 \pm 0.2319\%$) at $c = 0.0156$. Cross-validation results showed the polynomial kernel had greater variability in its optimal degree (4.75 ± 1.1347) compared to the Gaussian kernel's more stable c value (0.0146 ± 0.0030).

While the polynomial kernel minimizes error when tuned precisely, it is sensitive to parameter selection and may overfit. In contrast, the Gaussian kernel delivers competitive performance with greater robustness and stability, making it a preferred choice when consistent generalization is critical.

3.8. One-versus-One

(a) Research and explain 'One-versus-One'

One-versus-One(OvO) splits a multi-class classification dataset into binary classification problems. Different from OvR, it splits the dataset into one dataset for each class versus every other class. For a k class, we have $\frac{k(k-1)}{2}$ classifiers. Here are detailed OvO algorithm.

Step	Description
Input	$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\} \in (\mathbb{R}^n, \{1, \ldots, k\})^m$ where k is the number of classes.
Initialization	For each class pair (i, j), initialize the weight vector: $$\boldsymbol{W}_{ij} = \boldsymbol{0}, \quad \forall (i,j) \in \{1, \ldots, k\} \times \{1, \ldots, k\},\ i < j.$$ This creates $\binom{k}{2} = \frac{k(k-1)}{2}$ binary classifiers.
Prediction	For a given instance $\boldsymbol{x}_t$, compute the confidence scores for each binary classifier: $$\kappa_{ij}(\boldsymbol{x}_t) = \sum_{t=1}^{m} \alpha_{ij,t} K(\boldsymbol{x}_t, \boldsymbol{x}_t), \quad \forall (i,j).$$ Each binary classifier (i, j) votes for either class i or j: $$v_{ij}(\boldsymbol{x}_t) = \begin{cases} i & \text{if } \kappa_{ij}(\boldsymbol{x}_t) > 0, \\ j & \text{otherwise.} \end{cases}$$ The final prediction is determined by majority voting: $$\hat{y}_t = \arg\max_i \text{Votes for } i.$$
Update Rule	For each binary classifier (i, j): $$\text{If } y_t = i \text{ and } \kappa_{ij}(\boldsymbol{x}_t) \leq 0: \quad \alpha_{ij,t} \leftarrow \alpha_{ij,t} + 1.$$ $$\text{If } y_t = j \text{ and } \kappa_{ij}(\boldsymbol{x}_t) \geq 0: \quad \alpha_{ij,t} \leftarrow \alpha_{ij,t} - 1.$$
Training Loop	For each epoch: 1. For each training sample $(\boldsymbol{x}_t, y_t)$: • Identify all relevant binary classifiers (i, j) where $y_t \in \{i, j\}$. • Perform Prediction and Update steps for those classifiers. 2. Count the number of misclassifications for all classifiers.
Testing	For a test instance $\boldsymbol{x}_t$: 1. Compute $\kappa_{ij}(\boldsymbol{x}_t)$ for all binary classifiers. 2. Use majority voting to predict the class: $$\hat{y}_t = \arg\max_i \text{Votes for } i.$$ 3. Compute the test error rate as the number of incorrect predictions divided by the test size.
Output	Final weight matrices $\boldsymbol{W}_{ij}$ for all binary classifiers, training error, and test error rates.

Table 7: One-versus-One Kernel Perceptron (Training and Testing Process)

(b) Basic results for 'One-versus-One'

Here are the basic results for OvO.

Degree	Train Error(%)	Test Error(%)
1	1.7841 ± 0.5607	5.8091 ± 0.4183
2	0.0867 ± 0.0539	3.6882 ± 0.2140
3	0.0692 ± 0.0419	3.5941 ± 0.2276
4	0.0699 ± 0.0393	3.5457 ± 0.4286
5	0.0618 ± 0.0192	3.7070 ± 0.3372
6	0.0491 ± 0.0122	3.6183 ± 0.3388
7	0.0471 ± 0.0178	3.8548 ± 0.3288

Table 8: OvO: Train and Test Errors for Different Degrees

The optimal degree is 4 with lowest test error 3.5457.

(c) Cross-validation results for 'One-versus-One'

Here is the cross-validation results for OvO.

Run	d	Train Error(%)	Test Error(%)	Run	d	Train Error(%)	Test Error(%)
1	5	0.0504	3.9247	11	3	0.0672	4.1935
2	4	0.0504	3.9247	12	3	0.0672	3.7097
3	3	0.0840	4.2473	13	5	0.0672	4.5161
4	3	0.0336	3.9247	14	5	0.0504	4.3548
5	4	0.0336	3.8172	15	5	0.0504	4.2473
6	3	0.0672	4.7312	16	3	0.0672	3.6022
7	5	0.1008	4.1935	17	4	0.0504	3.8710
8	3	0.0336	3.7097	18	3	0.1176	3.5484
9	4	0.0336	3.7634	19	4	0.0672	3.8710
10	3	0.0672	3.7634	20	5	0.0000	4.1398

Table 9: OvO: Train and Test Errors for Optimal degree in 20 runs

And we can find the mean value of the Optimal degree and the overall test error as:

$$\text{Best degree: } 3.8500 \pm 0.8529 \quad \text{Test error(\%): } 4.0027 \pm 0.3058 \tag{73}$$

(d) Compare 'One-versus-One' and 'One-versus-Rest'

Here are the results from cross-validation via polynomial kernel.

Algorithm	Degree	Test Error(%)
OvR	4.75 ± 1.1347	3.9866 ± 0.1744
OvO	3.8500 ± 0.8529	4.0027 ± 0.3058

Table 10: Comparison of OvR and OvO

The comparison between the One-versus-Rest (OvR) and One-versus-One (OvO) methods using the polynomial kernel is summarized in Table 10. Both methods achieve comparable test errors, with OvR yielding a test error of $3.9866 \pm 0.1744\%$ and OvO slightly higher at $4.0027 \pm 0.3058\%$. While the OvR method requires slightly higher average degree (4.75 ± 1.1347) compared to OvO (3.8500 ± 0.8529), the test errors overlap within their respective standard deviations, indicating similar performance. The degree difference is expected as OvO designs a simpler classification than OvR, which yields to a lower degree of polynomial kernel function.

However, the OvO method is computationally more expensive due to the requirement of $\binom{k}{2}$ classifiers compared to k classifiers in OvR. This results in significantly longer training and testing times for OvO under the same setup, particularly as the number of classes increases. Thus, while OvO provides a more granular classification approach, OvR may be more practical for larger datasets or when computational resources are limited.