COMP0078: Supervised Learning **2024/25**

Coursework - ⟨2⟩

Due Date: ⟨**02/01/2025**⟩ *Student(s):* ⟨⟩

Instructions This is a sample template to submit coursework for COMP0078. It is not mandatory to use this template for submissions. This is provided just as a reference.

1.Rademacher Complexity of finite Spaces

1.1. Show that $\mathbb{E}[\bar{X}] \leq \frac{1}{\lambda} \log \mathbb{E}[e^{\lambda X}]$

To show the inequality, we employ the **log-sum-exp trick**, which provides a convenient bound for the maximum of a set of random variables. Specifically, for any $\lambda > 0$, we have:

$$\max_i X_i \leq \frac{1}{\lambda} \log \left(\sum_{i=1}^{m} e^{\lambda X_i} \right).$$

Next, we take the expectation of both sides:

$$\mathbb{E}[\max_i X_i] \leq \mathbb{E} \left[\frac{1}{\lambda} \log \left(\sum_{i=1}^{m} e^{\lambda X_i} \right) \right].$$

Since $\frac{1}{\lambda}$ is a constant, it can be factored out:

$$\mathbb{E}[\max_i X_i] \leq \frac{1}{\lambda} \mathbb{E} \left[\log \left(\sum_{i=1}^{m} e^{\lambda X_i} \right) \right].$$

Now, we apply **Jensen's inequality** for the concave function $\log(x)$. Jensen's inequality states that for any random variable Y and concave function f, we have:

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

Thus, applying this inequality to the logarithm:

$$\frac{1}{\lambda} \mathbb{E}[\log \left(\sum_{i=1}^{m} e^{\lambda X_i} \right)] \leq \frac{1}{\lambda} \log \mathbb{E} \left[\sum_{i=1}^{m} e^{\lambda X_i} \right].$$

By the **linearity of expectation**, we can simplify the expectation of the sum:

$$\mathbb{E} \left[\sum_{i=1}^{m} e^{\lambda X_i} \right] = \sum_{i=1}^{m} \mathbb{E}[e^{\lambda X_i}].$$

Therefore, we obtain the bound:

$$\mathbb{E}[\max_i X_i] \leq \frac{1}{\lambda} \log \sum_{i=1}^{m} \mathbb{E}[e^{\lambda X_i}].$$

Finally, for any collection $X_1, X_2, \ldots, X_m$ of centered random variables:

$$\mathbb{E}[\overline{X}] \leq \frac{1}{\lambda} \log \mathbb{E}[e^{\lambda \overline{X}}],$$

where $\overline{X}$ represents the maximum of the random variables.

1.2. Show that $\frac{1}{\lambda} \log \mathbb{E}[e^{\lambda \bar{X}}] \leq \frac{1}{\lambda} \log m + \frac{\lambda (b-a)^2}{8}$

Given **Hoeffding's lemma**, for a random variable X_i such that $X_i - \mathbb{E}[X_i] \in [a, b]$, we have:

$$\mathbb{E}\left[e^{\lambda (X_i - \mathbb{E}[X_i])}\right] \leq e^{\frac{\lambda^2 (b-a)^2}{8}}, \quad \forall \lambda > 0.$$

Since $\mathbb{E}[X_i] = 0$, the lemma applies directly:

$$\mathbb{E}[e^{\lambda X_i}] \leq e^{\frac{\lambda^2 (b-a)^2}{8}}.$$

Using the result from 1.1, we know:

$$\mathbb{E}[\bar{X}] \leq \frac{1}{\lambda} \log \mathbb{E}\left[\sum_{i=1}^{m} e^{\lambda \overline{X}}\right],$$

where $\bar{X} = \max_{i=1,\ldots,m} X_i$. By the linearity of expectation:

$$\mathbb{E}\left[\sum_{i=1}^{m} e^{\lambda X_i}\right] = \sum_{i=1}^{m} \mathbb{E}[e^{\lambda X_i}].$$

Substituting Hoeffding's lemma for $\mathbb{E}[e^{\lambda X_i}]$:

$$\mathbb{E}\left[\sum_{i=1}^{m} e^{\lambda X_i}\right] \leq m \cdot e^{\frac{\lambda^2 (b-a)^2}{8}}.$$

Taking the logarithm of both sides:

$$\log \mathbb{E}[e^{\lambda \bar{X}}] \leq \log \left(m \cdot e^{\frac{\lambda^2 (b-a)^2}{8}}\right).$$

Simplify the logarithm:

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

Dividing by $\lambda > 0$, we obtain:

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

Therefore, we have shown:

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

1.3. Show that $\mathbb{E}\left[\max_{i=1,\ldots,m} X_i\right] \leq \frac{b-a}{2}\sqrt{2\log(m)}$

Recall the bound we derived in 1.2:

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

where $\overline{X} = \max\{X_1, X_2, \ldots, X_m\}$.
To optimize the bound, we minimize the right-hand side with respect to λ. Taking the derivative with respect to λ and setting it equal to zero:

$$\frac{d}{d\lambda}\left(\frac{1}{\lambda}\log m + \frac{\lambda(b-a)^2}{8}\right) = 0.$$

Thus, we have the equation:

$$-\frac{1}{\lambda^2}\log m + \frac{(b-a)^2}{8} = 0.$$

Solving for λ:

$$\frac{1}{\lambda^2}\log m = \frac{(b-a)^2}{8},$$

$$\lambda^2 = \frac{8\log m}{(b-a)^2},$$

$$\lambda = \frac{\sqrt{8\log m}}{b-a}.$$

Substituting this value of λ into the bound:

$$\mathbb{E}[\max_i X_i] \leq \frac{1}{\lambda}\log m + \frac{\lambda(b-a)^2}{8}.$$

For the first term:

$$\frac{1}{\lambda}\log m = \frac{1}{\frac{\sqrt{8\log m}}{b-a}}\log m = \frac{b-a}{\sqrt{8\log m}}\cdot \log m = \frac{(b-a)\log m}{2\sqrt{2\log m}}$$

For the second term:

$$\frac{\lambda(b-a)^2}{8} = \frac{\sqrt{8\log m}}{b-a}\cdot\frac{(b-a)^2}{8} = \frac{(b-a)\sqrt{8\log m}}{8} = \frac{(b-a)\log m}{2\sqrt{2\log m}}$$

Adding the two terms and combining the results of 1.1 and 1.2, we have shown:

$$\mathbb{E}[\max_i X_i] = \mathbb{E}[\overline{X}] \leq \frac{b-a}{2}\sqrt{2\log m}$$

1.4. Show that $\mathcal{R}(S) \leq \max_{x \in S} \|x\|_2 \frac{\sqrt{2 \log m}}{n}$

To prove this result, we start with the definition of the Rademacher complexity:

$$\mathcal{R}(S) = \mathbb{E}_\sigma \left[\max_{x \in S} \frac{1}{n} \sum_{j=1}^{n} \sigma_j x_j \right],$$

where σ_j are independent Rademacher random variables (i.e., $\sigma_j \in \{-1, 1\}$), $x \in S$ is fixed for the maximum, and S is a finite set with cardinality m.
For a fixed $x_i \in S$, define:

$$X_i = \frac{1}{n} \sum_{j=1}^{n} \sigma_j x_j.$$

Here, X_i is a random variable dependent on the Rademacher variables σ_j, and we note the following properties:

1. X_i is a centered random variable because $\mathbb{E}[\sigma_j] = 0$ implies $\mathbb{E}[X_i] = 0$.

2. By Cauchy-Schwarz inequality, the range of X_i is bounded by:

$$X_i \in \left[-\frac{\|x_i\|_2}{n}, \frac{\|x_i\|_2}{n} \right].$$

   This is because the maximum possible value of $\frac{1}{n} \sum_{j=1}^{n} \sigma_j x_{ij}$ occurs when all $\sigma_j = 1$, giving $X_i = \frac{1}{n} \sum_{j=1}^{n} x_{ij}$. Similarly, the minimum occurs when all $\sigma_j = -1$, giving $X_i = -\frac{1}{n} \sum_{j=1}^{n} x_{ij}$.

Using the result from Part 1.3, we know:

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

where $b - a$ is the range of the random variables X_i. In this case, $b - a = \frac{2\|x_i\|_2}{n}$ because X_i lies within the interval $\left[-\frac{\|x_i\|_2}{n}, \frac{\|x_i\|_2}{n} \right]$. Substituting this range into the inequality, we have:

$$\mathbb{E} \left[\max_{i=1,\ldots,m} X_i \right] \leq \frac{\|x_i\|_2}{n} \sqrt{2 \log(m)}.$$

Since the Rademacher complexity involves taking the maximum over all $x \in S$, we write:

$$\mathcal{R}(S) = \mathbb{E}_\sigma \left[\max_{x \in S} \frac{1}{n} \sum_{j=1}^{n} \sigma_j x_j \right] = \mathbb{E}_\sigma \left[\max_{i=1,\ldots,m} X_i \right].$$

Taking the maximum $\|x_i\|_2$ over all $x \in S$, we obtain:

$$\mathcal{R}(S) \leq \max_{x \in S} \frac{\|x\|_2}{n} \sqrt{2 \log(m)}.$$

1.5. Prove an upper bound for $\mathcal{R}_S(\mathcal{H})$, where the cardinality of $\mathcal{H}$ appears logarithmically

To prove this result, we begin with the definition of the empirical Rademacher complexity:

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

where:

- $\sigma_i \in \{-1, 1\}$ are independent Rademacher random variables,
- $S = \{x_i\}_{i=1}^n$ is a fixed dataset of n points,
- $\mathcal{H}$ is a finite hypothesis class with cardinality $|\mathcal{H}| < +\infty$.

For any $f \in \mathcal{H}$, define the random variable:

$$X_f = \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i).$$

The supremum over $f \in \mathcal{H}$ corresponds to the maximum over all X_f, since $\mathcal{H}$ is finite:

$$\sup_{f \in \mathcal{H}} X_f = \max_{f \in \mathcal{H}} X_f.$$

Thus, the empirical Rademacher complexity becomes:

$$\mathcal{R}_S(\mathcal{H}) = \mathbb{E}_\sigma \left[\max_{f \in \mathcal{H}} X_f \right].$$

From the result of 1.3, we know that for any m centered random variables $X_1, \ldots, X_m$ bounded in $[a, b]$, the expected maximum satisfies:

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

Here:

- $m = |\mathcal{H}|$, the cardinality of the hypothesis class,
- Each $X_f = \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(x_i)$ is a centered random variable because $\mathbb{E}[\sigma_i] = 0$,
- Each $\sigma_i f(x_i)$ is a random variable that is bounded by $|f(x_i)|$. Thus, the random variables $\sigma_i f(x_i)$ are bounded by $[-|f(x_i)|, |f(x_i)|]$.
- The range of X_f is $\left[-\frac{\|f\|_2}{n}, \frac{\|f\|_2}{n} \right]$ by Cauchy-Schwarz inequality,, where $\|f\|_2 = \sqrt{\sum_{i=1}^{n} f(x_i)^2}$ is the Euclidean norm of the hypothesis evaluated at each point.

Thus, $b - a = \frac{2\|f\|_2}{n}$, and the result of 1.3 gives:

$$\mathbb{E}\left[\max_{f\in\mathcal{H}} X_f\right] \leq \frac{\|f\|_2}{n}\sqrt{2\log(|\mathcal{H}|)}.$$

Using the above bound, we can write:

$$\mathcal{R}_S(\mathcal{H}) = \mathbb{E}_\sigma\left[\sup_{f\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i f(x_i)\right] \leq \frac{\|f\|_2}{n}\sqrt{2\log(|\mathcal{H}|)}.$$

Since the Rademacher complexity involves taking the maximum over all $f \in \mathcal{H}$, we get:

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

2.Bayes Decision Rule and Surrogate Approaches

2.1. Show that the misclassification error corresponds to the expected risk of the 0-1 loss.

The misclassification error can be written as:

$$R(c) = \mathbb{P}_{(x,y)\sim\rho}\left(c(x) \neq y\right),$$

which is the probability that $c(x) \neq y$ for a randomly sampled input-output pair (x, y) from the distribution ρ. This is equivalent to the expected value of the indicator function $1_{c(x)\neq y}$, i.e., the probability that $c(x) \neq y$. The indicator function $1_{c(x)\neq y}$ takes the value 1 if $c(x) \neq y$ and 0 otherwise. Thus, the misclassification error can be expressed as:

$$R(c) = \mathbb{E}_{(x,y)\sim\rho}\left[1_{c(x)\neq y}\right].$$

This is the expected value of the indicator function with respect to the distribution $\rho(x, y)$. The expected value of the indicator function can also be written as an integral over the joint distribution $\rho(x, y)$:

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

which is precisely the definition of the expected risk of the 0-1 loss function.
Therefore, we have shown that the misclassification error $R(c)$ is equal to the expected risk of the 0-1 loss:

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

2.2. Calculate the closed-form minimizer f_* of $\mathcal{E}(f)$.

The expected risk for a surrogate loss function $\ell(f(x), y)$ is given by:

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

1. **Squared Loss:** $\ell(f(x), y) = (f(x) - y)^2$

   The expected risk is given by:

   $$\mathcal{E}(f) = \int_{\mathcal{X}} \left(\int_{\mathcal{Y}} (f(x) - y)^2\, d\rho(y|x) \right) d\rho_{\mathcal{X}}(x),$$

   where $\rho(y|x)$ is the conditional distribution of y given x, and $\rho_{\mathcal{X}}(x)$ is the marginal distribution of x.
   We start by expanding the squared term:

   $$(f(x) - y)^2 = f(x)^2 - 2f(x)y + y^2.$$

   Substituting this into the expected risk:

   $$\mathcal{E}(f) = \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} \left(f(x)^2 - 2f(x)y + y^2 \right)\, d\rho(y|x) \right] d\rho_{\mathcal{X}}(x).$$

   We can separate the terms inside the integral:

   $$\mathcal{E}(f) = \int_{\mathcal{X}} \left[f(x)^2 \int_{\mathcal{Y}} d\rho(y|x) - 2f(x) \int_{\mathcal{Y}} y\, d\rho(y|x) + \int_{\mathcal{Y}} y^2\, d\rho(y|x) \right] d\rho_{\mathcal{X}}(x).$$

   Using the fact that $\int_{\mathcal{Y}} d\rho(y|x) = 1$, we can simplify the first term:

   $$\mathcal{E}(f) = \int_{\mathcal{X}} \left[f(x)^2 - 2f(x)\mathbb{E}[y|x] + \mathbb{E}[y^2|x] \right] d\rho_{\mathcal{X}}(x),$$

   where $\mathbb{E}[y|x] = \int_{\mathcal{Y}} y\, d\rho(y|x)$ and $\mathbb{E}[y^2|x] = \int_{\mathcal{Y}} y^2\, d\rho(y|x)$.
   To minimize the expected risk $\mathcal{E}(f)$, we take the derivative of $\mathcal{E}(f)$ with respect to $f(x)$ and set it equal to zero:

   $$\frac{\partial}{\partial f(x)} \mathcal{E}(f) = \frac{\partial}{\partial f(x)} \left[f(x)^2 - 2f(x)\mathbb{E}[y|x] + \mathbb{E}[y^2|x] \right] = 0.$$

   The derivative of each term is:

   $$\frac{\partial}{\partial f(x)} \left(f(x)^2 \right) = 2f(x),$$

   $$\frac{\partial}{\partial f(x)} \left(-2f(x)\mathbb{E}[y|x] \right) = -2\mathbb{E}[y|x],$$

$$\frac{\partial}{\partial f(x)}\left(\mathbb{E}[y^2|x]\right) = 0 \quad \text{(since this term does not depend on } f(x)\text{)}.$$

Thus, the derivative of the expected risk becomes:

$$2f(x) - 2\mathbb{E}[y|x] = 0.$$

Now, solve for $f(x)$:

$$f(x) = \mathbb{E}[y|x].$$

Hence, the minimizer is:

$$f_*(x) = \mathbb{E}[y|x]$$

2. **Exponential Loss:** $\ell(f(x), y) = \exp(-yf(x))$

$$\mathcal{E}(f) = \int_{\mathcal{X}} \left(\int_{\mathcal{Y}} \exp(-yf(x))\, d\rho(y|x) \right) d\rho_{\mathcal{X}}(x)$$

Since $y \in \{-1, 1\}$, we can write the inner integral as a sum:

$$\int_{\mathcal{Y}} \exp(-yf(x))\, d\rho(y|x) = \exp(-f(x))\rho(y = 1|x) + \exp(f(x))\rho(y = -1|x).$$

Thus, the expected risk becomes:

$$\mathcal{E}(f) = \int_{\mathcal{X}} \left[\exp(-f(x))\rho(y = 1|x) + \exp(f(x))\rho(y = -1|x)\right] d\rho_{\mathcal{X}}(x).$$

To minimize $\mathcal{E}(f)$, we take the derivative of $\mathcal{E}(f)$ with respect to $f(x)$ and set it to zero:

$$\frac{d}{df(x)}\mathcal{E}(f) = \frac{d}{df(x)}\left[\int_{\mathcal{X}} \left(\exp(-f(x))\rho(y = 1|x) + \exp(f(x))\rho(y = -1|x)\right) d\rho_{\mathcal{X}}(x)\right]$$

Applying the chain rule, the derivatives are:

$$\frac{d}{df(x)}\left(\exp(-f(x))\right) = -\exp(-f(x)),$$

$$\frac{d}{df(x)}\left(\exp(f(x))\right) = \exp(f(x)).$$

Thus, the derivative of the expected risk becomes:

$$\frac{d}{df(x)}\mathcal{E}(f) = -\exp(-f(x))\rho(y = 1|x) + \exp(f(x))\rho(y = -1|x).$$

To minimize, we set the derivative equal to zero:

$$-\exp(-f(x))\rho(y = 1|x) + \exp(f(x))\rho(y = -1|x) = 0.$$

This simplifies to:

$$\exp(f(x))\rho(y=-1|x) = \exp(-f(x))\rho(y=1|x).$$

Multiplying both sides by $\exp(f(x))$ to eliminate the negative exponent:

$$\exp(2f(x))\rho(y=-1|x) = \rho(y=1|x).$$

Now, take the natural logarithm of both sides:

$$2f(x) = \log\left(\frac{\rho(y=1|x)}{\rho(y=-1|x)}\right).$$

Thus, the minimizer for the exponential loss is:

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

3. **Logistic Loss:** $\ell(f(x), y) = \log(1+\exp(-yf(x)))$

   Given the logistic loss function, the expected risk is:

$$\mathcal{E}(f) = \int_{\mathcal{X}}\left(\int_{\mathcal{Y}}\log(1+\exp(-yf(x)))\,d\rho(y|x)\right)d\rho_{\mathcal{X}}(x),$$

   Since $y \in \{-1, 1\}$, the sum over y becomes:

$$\int_{\mathcal{Y}}\log(1+\exp(-yf(x)))\,\rho(y|x) = \log(1+\exp(-f(x)))\rho(y=1|x)+\log(1+\exp(f(x)))\rho(y=-1|x).$$

   Thus, the expected risk becomes:

$$\mathcal{E}(f) = \int_{\mathcal{X}}\left[\log(1+\exp(-f(x)))\rho(y=1|x) + \log(1+\exp(f(x)))\rho(y=-1|x)\right]d\rho_{\mathcal{X}}(x).$$

   Now, we take the derivative of the expected risk $\mathcal{E}(f)$ with respect to $f(x)$. We will differentiate the two terms $\log(1+\exp(-f(x)))$ and $\log(1+\exp(f(x)))$ separately.

   Derivative of $\log(1+\exp(-f(x)))$**:**

   The derivative of $\log(1+\exp(-f(x)))$ with respect to $f(x)$ is:

$$\frac{d}{df(x)}\log(1+\exp(-f(x))) = -\frac{\exp(-f(x))}{1+\exp(-f(x))}.$$

   Derivative of $\log(1+\exp(f(x)))$**:**

   The derivative of $\log(1+\exp(f(x)))$ with respect to $f(x)$ is:

$$\frac{d}{df(x)}\log(1+\exp(f(x))) = \frac{\exp(f(x))}{1+\exp(f(x))}.$$

Thus, the derivative of the expected risk becomes:

$$\frac{d}{df(x)}\mathcal{E}(f) = -\frac{\exp(-f(x))}{1+\exp(-f(x))}\rho(y=1|x) + \frac{\exp(f(x))}{1+\exp(f(x))}\rho(y=-1|x).$$

To minimize the expected risk, we set the derivative equal to zero:

$$-\frac{\exp(-f(x))}{1+\exp(-f(x))}\rho(y=1|x) + \frac{\exp(f(x))}{1+\exp(f(x))}\rho(y=-1|x) = 0.$$

We multiply only the first term by $\frac{\exp(f(x))}{\exp(f(x))}$, which is equivalent to multiplying by 1. Simplifying the first term:

$$-\frac{1}{1+\exp(f(x))}\rho(y=1|x) + \frac{\exp(f(x))}{1+\exp(f(x))}\rho(y=-1|x) = 0.$$

Now, to eliminate the denominators, we multiply both sides by $(1+\exp(f(x)))$:

$$-\rho(y=1|x) + \exp(f(x))\rho(y=-1|x) = 0.$$

Rearranging the equation:

$$\exp(f(x))\rho(y=-1|x) = \rho(y=1|x).$$

Taking the natural logarithm of both sides:

$$f(x) = \log\frac{\rho(y=1|x)}{\rho(y=-1|x)}.$$

Thus, the minimizer is:

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

4. **Hinge Loss:** $\ell(f(x), y) = \max(0, 1 - yf(x))$

   We want to minimize the expected risk, which is given by:

$$\mathcal{E}(f) = \int_{\mathcal{X}}\left[\int_{\mathcal{Y}}\max(0, 1-yf(x))\,d\rho(y|x)\right]d\rho_{\mathcal{X}}(x).$$

   Since $y \in \{-1, 1\}$, we can rewrite the expected risk as a sum over the two possible values of y:

$$\mathcal{E}(f) = \int_{\mathcal{X}}\left[\max(0, 1-f(x))\rho(y=1|x) + \max(0, 1+f(x))\rho(y=-1|x)\right]d\rho_{\mathcal{X}}(x).$$

   The hinge loss function is piecewise, so we need to handle the cases separately.
   The derivative of the hinge loss $\ell(f(x), y) = \max(0, 1-yf(x))$ with respect to $f(x)$ is:

   - $f(x) \geq 1$ for $y = 1$ or $f(x) \leq -1$ for $y = -1$. In this case, the hinge loss is zero, and the derivative is zero.

- $f(x) \in (-1, 1)$ for both $y = 1$ and $y = -1$. In this case, the hinge loss is $1 - yf(x)$, and the derivative is $-y$.

The derivative of the expected risk becomes:

$$\frac{d}{df(x)}\mathcal{E}(f) = -\rho(y = 1|x) \cdot I(1 - f(x) > 0) + \rho(y = -1|x) \cdot I(1 + f(x) > 0),$$

where $I(\cdot)$ is the indicator function, which is 1 if the condition is true and 0 otherwise. To minimize the expected risk, we set the derivative equal to zero:

$$-\rho(y = 1|x) \cdot I(1 - f(x) > 0) + \rho(y = -1|x) \cdot I(1 + f(x) > 0) = 0.$$

This condition implies that the gradient is zero when the conditional probabilities are balanced.
Thus, the minimizer of the expected risk for the hinge loss is:

$$f_*(x) = \text{sign}\left(\rho(y = 1|x) - \rho(y = -1|x)\right).$$

2.3. The Bayes Decision Rule

The Bayes decision rule c_* minimizes the misclassification error $R(c)$ over all possible decision rules $c : \mathcal{X} \to \{-1, 1\}$. The misclassification error is given by:

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

For each $x \in \mathcal{X}$, the Bayes decision rule $c_*(x)$ is defined as:

$$c_*(x) = \arg \min_{c(x)\in\{-1,1\}} \mathbb{P}(c(x) \neq y|x).$$

Equivalently, $c_*(x)$ assigns the label that maximizes the posterior probability:

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

Thus, the Bayes decision rule is:

$$c_*(x) = \text{sign}\big(\rho(y = 1|x) - \rho(y = -1|x)\big).$$

2.4. Fisher Consistency of the Surrogate Frameworks

The surrogate framework is Fisher consistent if there exists a map $d : \mathbb{R} \to \{-1, 1\}$ such that:

$$R(c^*(x)) = R(d(f^*(x))),$$

where $c^*(x)$ is the Bayes decision rule (from 2.3) and $f^*(x)$ is the minimizer of the surrogate risk $\mathcal{E}(f)$ (from 2.2).

1. Squared Loss: $\ell(f(x), y) = (f(x) - y)^2$

The expected risk for the squared loss is:

$$\mathcal{E}(f) = \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} (f(x) - y)^2 \, d\rho(y|x) \right] d\rho_{\mathcal{X}}(x).$$

The minimizer $f^*(x)$ of the expected risk is $f^*(x) = \mathbb{E}[y|x]$.
The decision rule corresponding to $f^*(x)$ is:

$$c^*(x) = \text{sign}(f^*(x)) = \text{sign}(\mathbb{E}[y|x]).$$

For Fisher consistency, we need to find a map $d(f(x))$ such that:

$$R(c^*(x)) = R(d(f^*(x))).$$

Since $f^*(x)$ is the conditional expectation $\mathbb{E}[y|x]$, the map $d(f(x))$ is:

$$d(f(x)) = \text{sign}(f(x)).$$

Thus, the surrogate framework is Fisher consistent for squared loss, with $d(f(x)) = \text{sign}(f(x))$.

2. Exponential Loss: $\ell(f(x), y) = \exp(-yf(x))$

The expected risk for the exponential loss is:

$$\mathcal{E}(f) = \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} \exp(-yf(x)) \, d\rho(y|x) \right] d\rho_{\mathcal{X}}(x).$$

The minimizer $f^*(x)$ of the expected risk is:

$$f^*(x) = \frac{1}{2} \log \frac{\rho(y = 1|x)}{\rho(y = -1|x)}.$$

The decision rule corresponding to $f^*(x)$ is:

$$c^*(x) = \text{sign}(f^*(x)) = \text{sign}\left(\frac{1}{2} \log \frac{\rho(y = 1|x)}{\rho(y = -1|x)} \right).$$

For Fisher consistency, we need to find $d(f(x))$ such that:

$$R(c^*(x)) = R(d(f^*(x))).$$

Since $f^*(x) = \frac{1}{2} \log \frac{\rho(y=1|x)}{\rho(y=-1|x)}$, the map $d(f(x))$ is:

$$d(f(x)) = \text{sign}(f(x)).$$

Thus, the surrogate framework is Fisher consistent for the exponential loss, with $d(f(x)) = \text{sign}(f(x))$.

3. Logistic Loss: $\ell(f(x), y) = \log(1 + \exp(-yf(x)))$

The expected risk for the logistic loss is:

$$\mathcal{E}(f) = \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} \log(1 + \exp(-yf(x)))\, d\rho(y|x) \right] d\rho_{\mathcal{X}}(x).$$

The minimizer $f^*(x)$ of the expected risk is:

$$f^*(x) = \log \frac{\rho(y = 1|x)}{\rho(y = -1|x)}.$$

The decision rule corresponding to $f^*(x)$ is:

$$c^*(x) = \text{sign}(f^*(x)) = \text{sign}\left(\log \frac{\rho(y = 1|x)}{\rho(y = -1|x)} \right).$$

For Fisher consistency, we need to find $d(f(x))$ such that:

$$R(c^*(x)) = R(d(f^*(x))).$$

Since $f^*(x) = \log \frac{\rho(y=1|x)}{\rho(y=-1|x)}$, the map $d(f(x))$ is:

$$d(f(x)) = \text{sign}(f(x)).$$

Thus, the surrogate framework is Fisher consistent for the logistic loss, with $d(f(x)) = \text{sign}(f(x))$.

4. Hinge Loss: $\ell(f(x), y) = \max(0, 1 - yf(x))$

The expected risk for the hinge loss is:

$$\mathcal{E}(f) = \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} \max(0, 1 - yf(x))\, d\rho(y|x) \right] d\rho_{\mathcal{X}}(x).$$

The minimizer $f^*(x)$ of the expected risk is:

$$f^*(x) = \text{sign}\left(\rho(y = 1|x) - \rho(y = -1|x)\right).$$

The decision rule corresponding to $f^*(x)$ is:

$$c^*(x) = \text{sign}(f^*(x)) = \text{sign}\left(\rho(y = 1|x) - \rho(y = -1|x)\right).$$

Since $f^*(x) = \text{sign}\left(\rho(y = 1|x) - \rho(y = -1|x)\right)$, the decision rule is already based on the sign of the difference in conditional probabilities, so the map $d(f(x))$ is:

$$d(f(x)) = \text{sign}(f(x)).$$

Thus, the surrogate framework is Fisher consistent for the hinge loss, with $d(f(x)) = \text{sign}(f(x))$.

Conclusion

For each of the four loss functions (squared loss, exponential loss, logistic loss, and hinge loss), the surrogate framework is Fisher consistent, and the corresponding map $d(f(x))$ is:

$$d(f(x)) = \text{sign}(f(x)).$$

2.5. Comparison Inequality for Least Square Surrogates

2.5.1. Show $|R(\textbf{sign}(f)) - R(\textbf{sign}(f_*))| = \int_{\mathcal{X}_f} |f_*(x)| d\rho_{\mathcal{X}}(x)$.

The misclassification risk for a decision rule $c(x)$ is given by:

$$R(c) = \int_{\mathcal{X}} 1_{\{c(x) \neq y\}} \, d\rho(x, y),$$

where $1_{\{c(x)\neq y\}}$ is the indicator function that is 1 when the decision $c(x)$ does not match the true label y, and 0 otherwise.
For the decision rule $\text{sign}(f(x))$, the misclassification risk is:

$$R(\text{sign}(f)) = \int_{\mathcal{X}} 1_{\{\text{sign}(f(x)) \neq y\}} \, d\rho(x, y),$$

which measures the probability of making a misclassification using $\text{sign}(f(x))$.
Similarly, for $\text{sign}(f^*(x))$, the misclassification risk is:

$$R(\text{sign}(f^*)) = \int_{\mathcal{X}} 1_{\{\text{sign}(f^*(x)) \neq y\}} \, d\rho(x, y).$$

The difference in misclassification risks is:

$$|R(\text{sign}(f)) - R(\text{sign}(f^*))| = \int_{\mathcal{X}} \left(1_{\{\text{sign}(f(x)) \neq y\}} - 1_{\{\text{sign}(f^*(x)) \neq y\}}\right) d\rho(x, y).$$

The difference between the two indicator functions is nonzero only when $\text{sign}(f(x)) \neq \text{sign}(f^*(x))$. Thus, the set of points where the two decision rules disagree is:

$$X_f = \{x \in \mathcal{X} \mid \text{sign}(f(x)) \neq \text{sign}(f^*(x))\}.$$

Hence, the difference in risks becomes:

$$|R(\text{sign}(f)) - R(\text{sign}(f^*))| = \int_{X_f} \left(1_{\{\text{sign}(f(x)) \neq y\}} - 1_{\{\text{sign}(f^*(x)) \neq y\}}\right) d\rho(x, y).$$

The term $1_{\{\text{sign}(f(x)) \neq y\}} - 1_{\{\text{sign}(f^*(x)) \neq y\}}$ simplifies as follows:

- It is equal to 1 when $\text{sign}(f(x)) \neq \text{sign}(f^*(x))$ and y agrees with one of the two signs (i.e., either $\text{sign}(f(x))$ or $\text{sign}(f^*(x))$ is correct).
- It is 0 otherwise.

Thus, we have shown that:

$$|R(\text{sign}(f)) - R(\text{sign}(f^*))| = \int_{X_f} |f^*(x)| \, d\rho_{\mathcal{X}}(x),$$

where $X_f = \{x \in \mathcal{X} \mid \text{sign}(f(x)) \neq \text{sign}(f^*(x))\}$.

2.5.2. Show $\int_{\mathcal{X}_f} |f_*(x) - f(x)| \, d\rho_{\mathcal{X}}(x) \leq \sqrt{\mathbb{E}[(f(x) - f_*(x))^2]}$.

We first define the set X_f, which is the set of points where the signs of $f(x)$ and $f^*(x)$ differ:

$$X_f = \{x \in \mathcal{X} \mid \text{sign}(f(x)) \neq \text{sign}(f^*(x))\}.$$

Since $f(x)$ and $f^*(x)$ have different signs in X_f, we can use the triangle inequality and the properties of absolute value to show that:

$$|f^*(x) - f(x)| \geq |f^*(x)|.$$

This holds because the absolute difference between $f^*(x)$ and $f(x)$ is always greater than or equal to the absolute value of $f^*(x)$ when their signs differ.
Now, we integrate both sides of the inequality $|f^*(x) - f(x)| \geq |f^*(x)|$ over X_f:

$$\int_{X_f} |f^*(x)| \, d\rho_{\mathcal{X}}(x) \leq \int_{X_f} |f^*(x) - f(x)| \, d\rho_{\mathcal{X}}(x).$$

Recall that:

$$\mathbb{E}[|f(x) - f^*(x)|^2] = \int_{\mathcal{X}} |f(x) - f^*(x)|^2 \, d\rho_{\mathcal{X}}(x).$$

By Jensen's inequality for the square root function, we know that:

$$\sqrt{\mathbb{E}[|f(x) - f^*(x)|^2]} \geq \int_{\mathcal{X}} |f(x) - f^*(x)| \, d\rho_{\mathcal{X}}(x).$$

Thus, we obtain:

$$\int_{X_f} |f^*(x) - f(x)| \, d\rho_{\mathcal{X}}(x) \leq \sqrt{\mathbb{E}[|f(x) - f^*(x)|^2]}.$$

Finally, combining the results from the previous steps, we obtain the desired inequality:

$$\int_{X_f} |f^*(x)| \, d\rho_{\mathcal{X}}(x) \leq \int_{X_f} |f^*(x) - f(x)| \, d\rho_{\mathcal{X}}(x) \leq \sqrt{\mathbb{E}[|f(x) - f^*(x)|^2]}.$$

2.5.3. Show $\mathcal{E}(f) - \mathcal{E}(f_*) = \mathbb{E}[|f(x) - f_*(x)|^2]$.

For square loss, the expected risks are defined as:

$$\mathcal{E}(f) = \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} (f(x) - y)^2 \, d\rho(y|x) \right] d\rho_{\mathcal{X}}(x),$$

$$\mathcal{E}(f^*) = \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} (f^*(x) - y)^2 \, d\rho(y|x) \right] d\rho_{\mathcal{X}}(x).$$

Now subtract $\mathcal{E}(f^*)$ from $\mathcal{E}(f)$:

$$\mathcal{E}(f) - \mathcal{E}(f^*) = \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} (f(x) - y)^2 - (f^*(x) - y)^2 \, d\rho(y|x) \right] d\rho_{\mathcal{X}}(x).$$

Expand the difference between $(f(x) - y)^2$ and $(f^*(x) - y)^2$:

$$(f(x) - y)^2 - (f^*(x) - y)^2 = f(x)^2 - f^*(x)^2 - 2y(f(x) - f^*(x)).$$

Thus, the integral becomes:

$$\int_{\mathcal{X}} \left[\int_{\mathcal{Y}} \left(f(x)^2 - f^*(x)^2 - 2y(f(x) - f^*(x)) \right) d\rho(y|x) \right] d\rho_{\mathcal{X}}(x).$$

We now compute the expectations with respect to the distribution $\rho(y|x)$. Recall that $\int_{\mathcal{Y}} \rho(y|x)\, dy = 1$, and $\mathbb{E}[y|x]$ denotes the expected value of y conditioned on x. The expression then becomes:

$$\mathcal{E}(f) - \mathcal{E}(f^*) = \int_{\mathcal{X}} \left[f(x)^2 - f^*(x)^2 - 2(f(x) - f^*(x))\mathbb{E}[y|x] \right] d\rho_{\mathcal{X}}(x).$$

Since $f^*(x) = \mathbb{E}[y|x]$, we substitute this into the expression:

$$\mathcal{E}(f) - \mathcal{E}(f^*) = \int_{\mathcal{X}} \left[f(x)^2 - f^*(x)^2 - 2(f(x) - f^*(x))f^*(x) \right] d\rho_{\mathcal{X}}(x).$$

Next, we expand and simplify:

$$\mathcal{E}(f) - \mathcal{E}(f^*) = \int_{\mathcal{X}} \left[f(x)^2 - f^*(x)^2 - 2f(x)f^*(x) + 2f^*(x)^2 \right] d\rho_{\mathcal{X}}(x)$$

We can get:

$$\mathcal{E}(f) - \mathcal{E}(f^*) = \int_{\mathcal{X}} \left[f(x)^2 - 2f(x)f^*(x) + f^*(x)^2 \right] d\rho_{\mathcal{X}}(x).$$

This expression is now a perfect square:

$$\mathcal{E}(f) - \mathcal{E}(f^*) = \int_{\mathcal{X}} \left[(f(x) - f^*(x))^2 \right] d\rho_{\mathcal{X}}(x).$$

Thus, we have:

$$\mathcal{E}(f) - \mathcal{E}(f^*) = \mathbb{E}[|f(x) - f^*(x)|^2].$$

3.Kernel Perceptron (Handwritten Digit Classification)

3.1 One-vs-All Method

The **One-vs-All (OvA)** method generalizes a binary classifier, such as the Kernel Perceptron, to handle k-class classification problems by training k separate binary classifiers. For each class $j \in \{1, \ldots, k\}$, the training examples belonging to class j are treated as the positive class (labeled as $+1$), while the examples from all other classes ($\neq j$) are treated as the negative class (labeled as -1). Using this binary labeling scheme, a separate binary classifier is trained for each class j, resulting in k binary classifiers.

For a new input x_i, the prediction is made by computing the confidence scores for all k classifiers:

$$G_j(x_i) = \sum_{l=1}^{m} \alpha_l^{(j)} K(x_l, x_i),$$

where $\alpha_l^{(j)}$ are the learned weights for the support vectors corresponding to class j, $K(x_l, x_i)$ is the kernel function, and x_l are the support vectors. The input is then assigned to the class j^* with the highest confidence score:

$$j^* = \arg \max_{j \in \{1,\ldots,k\}} G_j(x_i).$$

This approach converts the multi-class problem into k binary classification problems.

3.2 One-vs-All Kernel Perceptron Implementation

The implementation of the One-vs-All Kernel Perceptron is summarized below:

Algorithm 1 One-vs-All Kernel Perceptron

Input: $\{(x_1, y_1), \ldots, (x_m, y_m)\} \subset \mathbb{R}^n \times \{-1, +1\}$, kernel function K, number of classes k, number of epochs T

Output: Trained Classifier with Weight matrix α

Initialize: $\alpha = \mathbf{0}^{k \times m}$

for *epoch* $t = 1$ *to* T **do**
  for *each sample* x_i *in* $\{x_1, \ldots, x_m\}$ **do**
    for *class* $j = 1$ *to* k **do**
      Compute confidence for class j:
      $G_j(x_i) = \sum_{l=1}^{m} \alpha_l^{(j)} K(x_l, x_i)$
    Make prediction:
    $\hat{y}_i = \arg\max_{j \in \{1,\ldots,k\}} G_j(x_i)$
    if $\hat{y}_i \neq y_i$ **then**
      Update:
      $\alpha_i^{(\hat{y}_i)} \leftarrow \alpha_i^{(\hat{y}_i)} - 1$
      $\alpha_i^{(y_i)} \leftarrow \alpha_i^{(y_i)} + 1$

Representation of $w(\cdot)$

The weight function $w(\cdot)$ is represented implicitly:

- α_i stores the contribution of each support vector.
- Support vectors x_i and their corresponding labels are stored for each binary classifier.
- The kernel function $K(x_i, \cdot)$ computes similarity between x_i and input points.

For each class c, we store $\alpha_i^{(c)}$, support vectors, and their binary labels.

Evaluation of $w(\cdot)$

During training, the decision function for class c is defined as:

$$h_c(x) = \sum_{i=1}^{m} \alpha_i^{(c)} K(x_i, x),$$

where $K(x_i, x)$ is the kernel function that measures similarity between the training sample x_i and the input x.

For efficient computation, the kernel matrix K is precomputed during training, where $K_{ij} = K(x_i, x_j)$ stores the kernel values between all pairs of training samples. This allows efficient reuse of precomputed values, avoiding repeated computation during training and prediction.

To classify a new sample x_{new} in the testing set: 1. Compute $w_c(x_{\text{new}})$ for each class c by evaluating the kernel values $K(x_{\text{new}}, x_j)$ for $j = 1, \ldots, n$, reusing the training weights α_c. 2. The decision function for each class c is:

$$w_c(x_{\text{new}}) = \sum_{j=1}^{n} \alpha_j^{(c)} K(x_j, x_{\text{new}}).$$

3. Assign the class c^* with the highest decision function value:

$$c^* = \arg\max_c w_c(x_{\text{new}}).$$

Adding Terms During Training

When a mistake occurs ($\hat{y}_t \neq y_t$), the model updates:

$$\alpha_t^{(c)} = \alpha_t^{(c)} + y_t,$$

and adds x_t to the list of support vectors for class c. This ensures only misclassified examples influence the decision boundary.

Choice of Epochs

We selected **epoch = 5** after observing that additional epochs beyond this point resulted in negligible improvements in model accuracy during preliminary experiments.

This choice balances computational efficiency and training time while ensuring the model converges sufficiently without unnecessary iterations. For this dataset and model, limiting to 5 epochs achieves a good trade-off between performance and resource usage.

3.3 One-vs-All Kernel Perceptron with Polynomial Kernel

The table below shows the mean training and test error rates across 20 runs for the polynomial kernel perceptron with degrees $d = 1, \ldots, 7$. **Results Table:**

d	Mean Train Error Rates (%)	Mean Test Error Rates (%)
1	7.5518 ± 0.2944	9.7581 ± 1.8428
2	0.8739 ± 0.2877	3.8331 ± 1.5810
3	0.2360 ± 0.1838	3.1102 ± 0.6437
4	0.0820 ± 0.1692	2.8118 ± 0.7367
5	0.0471 ± 0.1261	2.7151 ± 0.6295
6	0.0350 ± 0.1401	2.7903 ± 0.6084
7	0.0289 ± 0.1170	2.8226 ± 0.4026

From the results, we observed that:

- **Training Errors:** The training errors decrease significantly as the polynomial degree increases. For $d = 1$, the training error is relatively high at 7.55%. However, as d increases, the training error approaches extremely small, reaching 0.0289% for $d = 7$. This trend suggests that higher polynomial degrees provide a better fit to the training data.
- **Test Errors:** The test errors initially decrease with increasing polynomial degree, reaching the lowest point at $d = 5$ with a mean error of 2.7151%. Beyond $d = 5$, the test error begins to slightly increase, suggesting overfitting as the model becomes too complex and starts to memorize the training data.
- **Model Complexity:** While higher polynomial degrees achieve lower training errors, the slight increase in test errors for $d = 6$ and $d = 7$ reflects overfitting. Therefore, $d = 5$ appears to be the optimal degree, balancing model complexity and generalization performance.

Based on these results, the optimal polynomial degree for the kernel perceptron is $d = 5$. At this degree, the model achieves the lowest test error rate (2.7151%) while maintaining a low training error rate (0.0471%).

3.4 Optimal $d*$ for Polynomial Kernel via Cross-Validation

The table below shows the selected degree d^*, mean training error rates, and mean test error rates across 20 runs for the polynomial kernel perceptron.

Results Table:

Run	d*	Train Error Rates (%)	Test Error Rates (%)
1	4	0.1882	2.957
2	4	0.1479	2.3118
3	6	0.0269	2.4194
4	6	0.0403	3.4409
5	4	0.121	2.4194
6	6	0.0269	3.0108
7	6	0.0403	3.5484
8	4	0.0538	2.5806
9	6	0.0269	2.7419
10	3	0.2286	3.0645
11	6	0.0403	2.6344
12	6	0.0134	3.4409
13	6	0.0403	2.5269
14	6	0.0269	3.2258
15	6	0.0269	2.6882
16	6	0.0403	2.7957
17	6	0.0134	2.7419
18	4	0.0672	2.7957
19	6	0.0403	2.9032
20	6	0.0134	3.0108

Metric	Mean	Standard Deviation
Selected Degree (d^*)	5.30	0.95
Train Error Rates (%)	0.0618	0.0006
Test Error Rates (%)	2.8629	0.0034

From the results, we observed the following:

- **Selected Degree (d^*):** The selected degree d^* varies across runs, with $d^* = 6$ being the most frequently selected, followed by $d^* = 4$. This indicates that higher polynomial degrees generally perform better on this dataset.
- **Training Errors:** The training errors are consistently low, with a mean error rate of approximately 0.0618%. This shows that the perceptron effectively fits the training data, even for higher polynomial degrees.
- **Test Errors:** The test errors vary slightly across runs, with a mean error rate of approximately 2.8629%. The results indicate good generalization performance, though there are cases where the test error increases for higher d^*, suggesting potential overfitting.
- **Model Complexity:** The frequent selection of $d^* = 6$ as the best-performing degree suggests that the model benefits from higher polynomial degrees. However, some evidence of overfitting is observed, as test errors increase in certain runs despite low training errors.

To conclude, the cross-validation results demonstrate that the polynomial kernel perceptron achieves low error rates on both training and testing sets. While $d^* = 6$ is the most frequently selected degree, $d^* = 5$ or $d^* = 4$ may provide a more balanced trade-off between training and testing performance, reducing the risk of overfitting.

3.5 Confusion Matrix Averaged Across 20 Runs

The table below shows the mean confusion matrix with standard deviations, computed over 20 runs, for the polynomial kernel perceptron using the optimal degree d^* determined via cross-validation. Here, the rows represent the true labels, and the columns represent the predicted labels.

Final Confusion Matrix (Mean ± Std):

	0	1	2	3	4	5	6	7	8	9
0	0.000 ± 0.563	0.049 ± 0.117	0.223 ± 0.289	0.229 ± 0.234	0.145 ± 0.194	0.181 ± 0.268	0.164 ± 0.262	0.032 ± 0.097	0.143 ± 0.234	0.096 ± 0.147
1	0.000 ± 0.000	0.000 ± 0.634	0.076 ± 0.152	0.018 ± 0.080	0.253 ± 0.448	0.000 ± 0.000	0.269 ± 0.270	0.039 ± 0.116	0.078 ± 0.155	0.039 ± 0.118
2	0.408 ± 0.330	0.167 ± 0.255	0.000 ± 1.279	0.517 ± 0.605	0.721 ± 0.656	0.115 ± 0.300	0.272 ± 0.354	0.753 ± 0.674	0.393 ± 0.518	0.052 ± 0.158
3	0.129 ± 0.347	0.185 ± 0.336	0.639 ± 0.617	0.000 ± 1.970	0.087 ± 0.270	1.894 ± 1.217	0.059 ± 0.178	0.339 ± 0.358	0.926 ± 0.630	0.174 ± 0.411
4	0.086 ± 0.205	0.548 ± 0.555	0.556 ± 0.574	0.058 ± 0.173	0.000 ± 1.242	0.170 ± 0.360	0.610 ± 0.500	0.257 ± 0.335	0.118 ± 0.306	0.533 ± 0.491
5	0.734 ± 1.029	0.106 ± 0.252	0.380 ± 0.466	0.976 ± 0.763	0.425 ± 0.624	0.000 ± 2.206	0.798 ± 0.556	0.172 ± 0.298	0.461 ± 0.688	0.381 ± 0.463
6	0.858 ± 0.665	0.148 ± 0.256	0.242 ± 0.446	0.033 ± 0.142	0.528 ± 0.578	0.301 ± 0.355	0.000 ± 1.142	0.000 ± 0.000	0.230 ± 0.328	0.058 ± 0.176
7	0.028 ± 0.124	0.175 ± 0.325	0.252 ± 0.366	0.214 ± 0.293	1.010 ± 0.725	0.059 ± 0.256	0.000 ± 0.000	0.000 ± 1.026	0.408 ± 0.416	0.660 ± 0.658
8	0.745 ± 0.723	0.464 ± 0.479	0.585 ± 0.551	1.031 ± 1.022	0.663 ± 0.825	1.363 ± 1.276	0.354 ± 0.529	0.250 ± 0.411	0.000 ± 2.054	0.141 ± 0.373
9	0.123 ± 0.247	0.031 ± 0.137	0.209 ± 0.393	0.186 ± 0.286	1.545 ± 1.167	0.091 ± 0.289	0.000 ± 0.000	0.931 ± 0.835	0.121 ± 0.242	0.000 ± 1.692

From the output matrix we found that:

Misclassification Patterns

- **Digit 8**:
  - Has a relatively high confusion rate with digit 5 ($1.363 \pm 1.276\%$) and digit 3 ($1.031 \pm 1.022\%$).
  - This suggests that digits 8 and 5 as well as 8 and 3 share similar visual features, such as loops or curves, making them hard to distinguish.
- **Digit 5**:
  - Shows notable confusion with digit 8 ($1.363 \pm 1.276\%$) and digit 3 ($0.976 \pm 0.763\%$).
  - This is expected since 5 and 8 can look similar in certain handwritten styles, particularly with poorly written samples.
- **Digit 9**:
  - Has a moderate confusion rate with digit 7 ($0.931 \pm 0.835\%$), which is likely due to the top loops in digit 9 resembling a slanted 7.

Low Misclassification Rates

- Digits 0 and 1 generally have low confusion rates with other digits (e.g., confusion rates are consistently below 0.5%).
- This is likely because their shapes (a closed loop for 0 and a vertical line for 1) are visually distinct.

High Variability in Confusion

- Some confusion rates have relatively high standard deviations:
  - For instance, digit 5 misclassified as 8 ($1.363 \pm 1.276\%$) has a significant variability, indicating inconsistency in the model's performance across different test samples or runs.
  - Digit 8 misclassified as 3 ($1.031 \pm 1.022\%$) also shows similar variability.

3.6 Hard-to-predict Samples: Top 5

We obtained the five hardest-to-predict samples in the dataset with highest misclassification rates over 20 runs using the optimal d^* each time, along with their true and predicted labels.

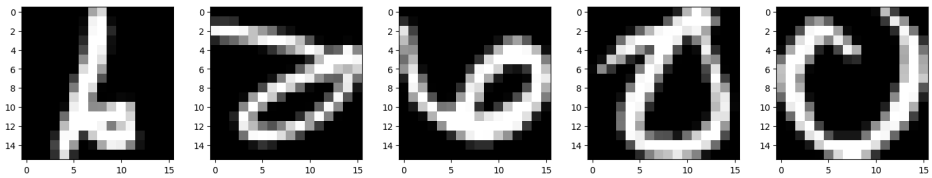

Figure 1.1. Visualization of the 5 hardest-to-predict samples: true label is 1, 0, 6, 0, 5 from left to right.

We observed that:

- **Sample 1:** True Label = 1, Predicted Label = 6.
  - The vertical line structure of the digit "1" could resemble part of a "6," especially given the pixelated and noisy nature of the image.
- **Sample 2:** True Label = 0, Predicted Label = 3.
  - The curvature of the "0" in the pixelated image might have been misinterpreted as the top half of a "3."
- **Sample 3:** True Label = 6, Predicted Label = 0.
  - The circular structure of the "6" overlaps visually with the structure of a "0," making it challenging for the classifier to differentiate.
- **Sample 4:** True Label = 0, Predicted Label = 5.

  - The "0" could have been confused with the lower half of the "5" due to overlapping visual patterns in pixelated images.

- **Sample 5:** True Label = 5, Predicted Label = 0.
  - The curved structure of the "5" could resemble a "0" when viewed in a low-resolution, pixelated format.

3.7 One-vs-All Kernel Perceptron with Gaussian Kernel

3.7.1 Selection of the Set S for the Parameter c

To determine a reasonable set S of values for c, we performed initial experiments to identify a range of c values that minimize the test error while capturing the trends around the optimal region.

We started with a coarse grid search, testing c values across a wide range, including $\{0.01, 0.10, 0.50, 1.00, 5.00, 10.00, 50.00\}$. The results indicated that test errors were relatively low for very small c values ($c = 0.01$) and began increasing for $c \geq 5$, highlighting potential overfitting or underfitting issues for large c.

Based on these observations, we refined our search to a narrower range around the region of interest and tested a new set of c values $\{0.005, 0.01, 0.015, 0.02, 0.05, 0.1, 0.3, 0.5, 1\}$. The refined results indicated that the lowest test error was achieved at $c = 0.01$, with optimal performance concentrated in c-value range ($[0.005, 0.02]$).

The finalized set S we chose is:

$$S = \{0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.5\}.$$

This set ensures a diverse exploration of the parameter space, focusing on the region of interest, while covering diverse scales of c for valid evaluation.

3.7.2 One-vs-All Kernel Perceptron with Gaussian Kernel

The table below summarizes the mean train and test error rates for different c values in the set $S = \{0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.5\}$. Each cell contains the mean $\pm$ standard deviation of the error rates computed over 20 runs.

c	Mean Train Error Rates (%)	Mean Test Error Rates (%)
0.001	6.8621 ± 0.0294	8.7876 ± 0.0283
0.005	0.3865 ± 0.0014	3.3226 ± 0.0051
0.01	0.0746 ± 0.0005	2.8199 ± 0.0046
0.02	0.0229 ± 0.0001	2.5860 ± 0.0043
0.05	0.0430 ± 0.0009	3.9005 ± 0.0051
0.1	0.0161 ± 0.0003	5.3602 ± 0.0046
0.5	0.0020 ± 0.0017	6.7581 ± 0.0063

Observations on the Results:

- **Train Error Trends:** The train error rates decrease as c increases from 0.001 to 0.02, indicating better fit to the training data. However, after $c = 0.02$, the train error rates fluctuates slightly.
- **Test Error Trends:** The test error rates are lowest for $c = 0.01$ and $c = 0.02$, indicating optimal performance. Beyond this range ($c > 0.02$), the test error rates increase, likely due to overfitting as the model becomes more sensitive to small-scale variations in the data.
- **Overall:** The Gaussian kernel achieves good performance for small c values ($c \leq 0.02$), but larger values of c lead to poor generalization on the test set.

3.7.3 Optimal c for Gaussian Kernel via Cross-Validation

The table below shows the selected parameter c^*, mean training error rates, and mean test error rates across 20 runs for the Gaussian kernel perceptron.

Results Table:

Run	c*	Train Error Rates (%)	Test Error Rates (%)
1	0.02	0.0269	2.8495
2	0.02	0.0538	2.3656
3	0.02	0.0269	1.4516
4	0.02	0.0403	3.2258
5	0.01	0.0807	3.0108
6	0.02	0.0269	3.172
7	0.02	0.0134	3.0108
8	0.02	0.0134	2.4194
9	0.02	0.0134	2.2043
10	0.01	0.121	3.172
11	0.02	0.0269	2.0968
12	0.02	0.0134	2.6882
13	0.02	0.0269	2.2581
14	0.02	0.0269	3.172
15	0.02	0.0403	2.957
16	0.02	0.0521	2.3656
17	0.02	0.0134	2.6344
18	0.02	0.0269	2.7419
19	0.02	0.0403	2.7419
20	0.02	0.0134	3.1183

Metric	Mean	Standard Deviation
Selected Parameter (c^*)	0.02	0.004
Train Error Rates (%)	0.0323	0.0003
Test Error Rates (%)	2.6828	0.0045

From the results, we observed the following:

- **Selected Parameter (c^*):** The selected parameter c^* is consistently 0.02 across most runs, with occasional selections of $c^* = 0.01$. This indicates that a narrower Gaussian kernel (higher c) performs well for this dataset.
- **Training Errors:** The training errors are very low, with a mean error rate of $0.0323\% \pm 0.0003\%$, demonstrating that the Gaussian kernel perceptron effectively fits the training data.
- **Test Errors:** The test errors vary slightly, with a mean error rate of $2.6828\% \pm 0.0045\%$. The low variability in test errors suggests stable generalization performance for the selected kernel parameter.
- **Model Complexity:** The consistent selection of $c^* = 0.02$ highlights the effectiveness of this kernel width in balancing training fit and generalization. Smaller kernel widths ($c^* = 0.01$) lead to slightly higher test errors in some runs, suggesting potential underfitting.

To conclude, the Gaussian kernel perceptron achieves low error rates and stable generalization across runs. The parameter $c^* = 0.02$ is optimal for this dataset, providing a balance between effective training and minimal test error rates.

3.7.4 Gaussian Kernel vs. Polynomial Kernel

We compares the One-vs-All Kernel Perceptron results with Gaussian and polynomial kernels, focusing on key metrics:

Training Error Rates

- **Polynomial Kernel:** Training errors decrease with d, reaching $0.0289\% \pm 0.1170$ at $d = 7$, indicating effective data fitting.
- **Gaussian Kernel:** Training errors reach 0.002% for $c \geq 0.5$, showing perfect fitting for larger c.
- **Comparison:** Both kernels achieve very low training errors, but the Gaussian kernel achieves perfect fitting with fewer parameters.

Test Error Rates

- **Polynomial Kernel:** Lowest test error is $2.7151\% \pm 0.6295$ at $d = 5$, with errors increasing for $d > 5$ due to overfitting.
- **Gaussian Kernel:** Best test error is $2.8199\% \pm 0.0046$ at $c = 0.01$, increasing for larger c.
- **Comparison:** Both achieve similar optimal test errors, but the polynomial kernel generalizes slightly better at moderate complexity ($d = 5$) and Gaussian kernel performs slightly better in cross-validation.

Optimal Parameters and Generalization

- **Polynomial Kernel:** $d^* = 6$ is most frequent, with $d = 4$ offering a good balance between complexity and generalization.
- **Gaussian Kernel:** $c = 0.02$ is optimal, balancing flexibility and generalization.
- **Comparison:** Polynomial kernel benefits from moderately high degrees, while the Gaussian kernel is sensitive to c and requires careful tuning.

Model Complexity

- **Polynomial Kernel:** Complexity increases with d, with overfitting for $d > 5$.
- **Gaussian Kernel:** High c leads to overfitting and achieves low complexity for smaller c.
- **Comparison:** Polynomial kernel is more stable at moderate degrees, while the Gaussian kernel is efficient for well-tuned c.

3.8 One-vs-One Kernel Perceptron for Binary Classifier

3.8.1 One-vs-One: Definition and Implementation

The **One-vs-One (OvO)** method generalizes a binary classifier to handle k-class classification problems by training $\binom{k}{2} = \frac{k(k-1)}{2}$ binary classifiers. For each pair of classes $(j_1, j_2) \in \{1, \ldots, k\}$, a binary classifier is trained using only the training examples belonging to these two classes. The examples of class j_1 are labeled as $+1$, while the examples of class j_2 are labeled as -1. This process is repeated for all possible class pairs, resulting in $\binom{k}{2}$ classifiers.

For a new input x_i, each of the $\binom{k}{2}$ classifiers makes a prediction, effectively voting for one of the two classes. The input is then assigned to the class with the majority vote:

$$j^* = \arg \max_{j \in \{1,\ldots,k\}} \text{Vote}_j(x_i),$$

where $\text{Vote}_j(x_i)$ is the number of times class j is predicted across all classifiers involving class j.

The implementation of the One-vs-One Kernel Perceptron is below:

Algorithm 2 One-vs-One Kernel Perceptron

Input: $\{(x_1, y_1), \ldots, (x_m, y_m)\} \subset \mathbb{R}^n \times \{-1, +1\}$, kernel function K, number of classes k, number of epochs T

Output: Weight matrix α for each classifier

Initialize: $\alpha_{(j_1,j_2)} = \mathbf{0}^m$ for each class pair (j_1, j_2)

for *each class pair* (j_1, j_2) **do**

  Extract training data for classes j_1 and j_2 Re-label samples: $y_i = +1$ if $y_i = j_1$, $y_i = -1$ if $y_i = j_2$ Train a binary Kernel Perceptron for (j_1, j_2) **for** *epoch* $t = 1$ *to* T **do**

    for *each sample* x_i *in* $\{x_1, \ldots, x_m\}$ **do**

      Compute confidence:

      $G_{(j_1,j_2)}(x_i) = \sum_{l=1}^{m} \alpha_l^{(j_1,j_2)} K(x_l, x_i)$ **if** *prediction* $\neq$ *true label* **then**

        Update $\alpha_{(j_1,j_2)}$

for *each test sample* x_i **do**

  Compute votes for all k classes using $\binom{k}{2}$ classifiers Assign x_i to the class with the majority vote

3.8.2 One-vs-One Kernel Perceptron with Polynomial Kernel

The table below shows the mean train and test error rates, along with their standard deviations, for the One-vs-One method using the polynomial kernel across 20 runs for each degree d.

Degree	Mean Train Error Rates (%)	Mean Test Error Rates (%)
1	9.3358 ± 0.2502	10.1747 ± 0.9133
2	4.7472 ± 0.1793	6.8199 ± 0.6107
3	2.1760 ± 0.2140	5.5134 ± 0.4882
4	1.1428 ± 0.2258	5.3925 ± 0.6006
5	0.7946 ± 0.1787	5.1425 ± 0.4725
6	0.5963 ± 0.1273	4.9086 ± 0.5193
7	0.4383 ± 0.1179	4.9274 ± 0.5219

From the results, we observe that as the polynomial degree d increases, the train error consistently decreases, indicating that the model becomes more expressive and fits the training data better. For test errors, there is a general decreasing trend up to $d = 6$, with the lowest test error observed at $d = 6$. We can however observe that $d = 7$ provides the best balance between test and train error in model fitting for this dataset.

3.8.3 Optimal d^* for Polynomial Kernel using OvO

The table below shows the selected degree d^*, mean training error rates, and mean test error rates across 20 runs for the polynomial kernel perceptron using the 'One-versus-One' method.

Results Table:

Run	d*	Train Error Rates (%)	Test Error Rates (%)
1	7	0.4571	4.1935
2	7	0.3496	4.5699
3	7	0.3764	4.1935
4	7	0.4974	5.2151
5	6	0.5109	4.5161
6	7	0.4840	4.9462
7	7	0.4382	4.8387
8	7	0.6857	4.9462
9	7	0.3092	4.4086
10	7	0.6857	5.3763
11	7	0.3361	4.6774
12	7	0.3764	5.0538
13	7	0.4033	4.5161
14	6	0.4840	6.2151
15	6	0.5378	5.3226
16	7	0.4571	4.0110
17	6	0.3630	5.1613
18	7	0.3496	4.7312
19	7	0.5243	4.3548
20	7	0.2286	5.4839

Metric	Mean	Standard Deviation
Selected Degree (d^*)	6.65	0.48
Train Error Rates (%)	0.4423	0.1128
Test Error Rates (%)	4.8414	0.4750

From the results, we observed that:

- **Selected Degree (d^*):** The selected degree d^* is consistently close to 7, with a mean of 6.85 ± 0.48, indicating that higher polynomial degrees generally yield better performance.

- **Training Errors:** The training errors are consistently low, averaging $0.4423\% \pm 0.1128$, suggesting that the model fits the training data effectively.
- **Test Errors:** The test errors demonstrate stable generalization, with a mean error rate of $4.8414\% \pm 0.475$. While some variability exists, the model performs reliably across different runs.
- **Model Complexity:** The frequent selection of higher degrees (e.g., $d = 7$) as optimal reflects the dataset's potential benefit from higher-order polynomial transformations. However, the moderate variability in test errors highlights the importance of balancing model complexity and generalization.

3.8.4 One-vs-All vs. One-vs-One

To compare the results of the One-versus-One (OvO) and One-versus-All (OvA) methods with the polynomial kernel, we mainly looked at the 1. Train and test error rates, 2. The selected optimal degree d^*, and 3. General model behavior.

1. Train Error Rates

- **One-versus-All:** OvA achieves lower training errors, with the smallest error at $d = 7$ $(0.0289\% \pm 0.1170)$, indicating strong training performance for higher degrees.
- **One-versus-One:** OvO also achieves low training errors, with a mean of $0.4423\% \pm 0.1128$ across runs, though slightly higher than OvA.
- **Comparison:** OvA is better at minimizing training errors due to its binary classifier per class approach, which increases flexibility.

2. Test Error Rates

- **One-versus-All:** OvA achieves the lowest test error at $d = 5$ $(2.7151\% \pm 0.6295)$, but errors increase for $d > 5$, indicating overfitting.
- **One-versus-One:** OvO test errors remain stable across higher degrees, with the lowest mean error $(4.8414\% \pm 0.4750)$ for $d^* \approx 7$.
- **Comparison:** OvA outperforms OvO in minimizing test errors but is more prone to overfitting at higher degrees.

3. Selected Optimal Degree (d^*)

- **One-versus-All:** The optimal degree varies across runs, with $d = 4$ and $d = 6$ most common (5.30 ± 0.95).
- **One-versus-One:** OvO consistently selects higher degrees, with $d^* \approx 7$ (6.65 ± 0.48), reflecting the need for more complex boundaries.
- **Comparison:** OvO prefers higher degrees, while OvA balances complexity and generalization at moderate degrees.

4. Model Complexity and Generalization

- **One-versus-All:** OvA requires training k binary classifiers. The time complexity is proportional to $k \times m^2$ for training, where m is the number of training samples. While OvA generalizes well at moderate degrees ($d = 5$), higher degrees increase computational costs and risk overfitting.
- **One-versus-One:** OvO involves $\frac{k(k-1)}{2}$ pairwise classifiers, significantly increasing training time compared to OvA. The time complexity scales as $\frac{k(k-1)}{2} \times m^2$. However, OvO shows stable generalization for higher degrees ($d^* \approx 7$), making it suitable for datasets requiring complex decision boundaries.
- **Comparison:** OvA is computationally more efficient, especially for large k, but risks overfitting at high degrees. OvO, while more computationally intensive, is better at handling complex classification problems due to its robustness at higher degrees.