Supervised Learning (COMP0078) Coursework 2

1.1. Proof of the Exponential Bound for $E[\overline{X}]$

(1) Start with the exponential function and Markov's inequality

Markov's inequality states that for any non-negative random variable X and $\lambda > 0$,

$$P(X \geq t) \leq E\left[e^{\lambda X}\right] \cdot e^{-\lambda t},$$

This inequality is useful because it relates the tail probability $P(X \geq t)$ to the exponential moment $E\left[e^{\lambda X}\right]$, , which is often easier to bound.

(2) Express $E[X]$ as an integral

The expectation of X, can be written in terms of the cumulative probability $P(X \geq t)$:

$$E[X] = \int_0^\infty P(\bar{X} \geq t)\ dt.$$

This expression captures the contribution of all tail probabilities to the expected value of X. It essentially integrates over the probability mass of X above each threshold t.

(3) Substitute $P(X \geq t)$ using the union bound

If $X = \max X_i$, then the event $\{X \geq t\}$ implies that at least one of the X_i exceeds t. Using the union bound:

$$P(X \geq t) = P\left(\bigcup_{i=1}^{m}\{X_i \geq t\}\right) \leq \sum_{i=1}^{m} P(X_i \geq t).$$

This simplifies a potentially complex event into a sum of probabilities for each X_i, which are easier to handle. Applying Markov's inequality to each X_i, we have:

$$P(X_i \geq t) \leq E\left[e^{\lambda X_i}\right] \cdot e^{-\lambda t}.$$

Thus, summing over all i:

$$P(X \geq t) \leq \sum_{i=1}^{m} E\left[e^{\lambda X_i}\right] \cdot e^{-\lambda t}.$$

This expresses the tail probability of X in terms of the exponential moments of the X_i.

(4) Simplify the bound on $P(X \geq t)$

Let $M = \max_i E\left[e^{\lambda X_i}\right]$. Since $E\left[e^{\lambda X_i}\right]$ are bounded by M, we can simplify:

$$P(X \geq t) \leq m \cdot M \cdot e^{-\lambda t}.$$

Here, m accounts for the number of terms in the sum, and M is the maximum exponential moment among the X_i.

(5) Substitute back into the integral for $E[X]$

$$E[X] = \int_0^\infty P(X \geq t)\ dt \leq \int_0^\infty m \cdot M \cdot e^{-\lambda t}\, dt.$$

This reduces the integral of the tail probability to a simpler form involving $e^{-\lambda t}$.

(6) Evaluate the integral

The integral $\int_0^\infty e^{-\lambda t}\, dt$ is straightforward to compute:

$$\int_0^\infty e^{-\lambda t}\ dt = \frac{1}{\lambda}.$$

Substituting this back, we get:

$$E[X] \leq m \cdot M \cdot \frac{1}{\lambda}.$$

Thus:

$$E[X] \leq \frac{m \cdot M}{\lambda}.$$

(7) Logarithmic upper bound

Finally, substitute $M = \max_i E\left[e^{\lambda X_i}\right]$ back into the expression. Using the fact that $E\left[e^{\lambda X}\right] \geq \max_i E\left[e^{\lambda X_i}\right]$, we obtain,

$$E[X] \leq \frac{1}{\lambda}\log\left(m \cdot E\left[e^{\lambda X}\right]\right).$$

By combining the contributions of m (number of variables) and $E\left[e^{\lambda X}\right]$ into a logarithmic term, we achieve the desired bound:

$$E[X] \leq \frac{1}{\lambda}\log E\left[e^{\lambda X}\right] + \frac{\log m}{\lambda}.$$

1.2. Proof of the Upper Bound on $\frac{1}{\lambda}\log E\left[e^{\lambda \bar{X}}\right]$

(1) Bound $E\left[e^{\lambda X_i}\right]$ using Hoeffding's Lemma

Hoeffding's Lemma provides a bound on the moment-generating function of a random variable X_i that satisfies $X_i - E[X_i] \in [a, b]$. Specifically, for any $\lambda > 0$:

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

Since the random variables X_i in this problem are centered, this simplifies to:

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

This step establishes an upper bound on the exponential moment of each X_i, individually.

(2) Relating $E\left[e^{\lambda \bar{X}}\right]$ to $E\left[e^{\lambda X_i}\right]$

By definition, $\bar{X} = \max_i X_i$. Using the union bound for probabilities and exponential inequalities, we can write:

$$e^{\lambda \bar{X}} \leq \sum_{i=1}^{m} e^{\lambda X_i}.$$

Taking the expectation on both sides:

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

From Step 1, we know $E\left[e^{\lambda X_i}\right] \leq e^{\frac{\lambda^2(b-a)^2}{8}}$. Substituting this bound:

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

This gives an upper bound on the exponential moment of $\bar{X}$ in terms of the number of random variables m and the variance-dependent term $(b-a)^2$.

(3) Taking the logarithm

We now take the natural logarithm on both sides to simplify the expression:

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

Using the logarithmic property $\log(ab) = \log a + \log b$, :

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

This step separates the complexity term $\log m$ and the variance-dependent term $\frac{\lambda^2(b-a)^2}{8}$.

(4) Divide by λ

Finally, divide through by $\lambda > 0$ to complete the bound:

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

Thus, we have shown that $\frac{1}{\lambda}\log E\left[e^{\lambda \bar{X}}\right]$ is bounded by the complexity term $\frac{1}{\lambda}\log m$ and the variance-dependent term $\frac{\lambda(b-a)^2}{8}$.

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

(1) Restate the bound from Question 1.2

From Question 1.2, we derived the inequality:

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

where $\bar{X} = \max_{i=1,\ldots,m} X_i$. This result bounds the expected value of the exponential of $\bar{X}$, relating it to the number of random variables m, their range $[a, b]$, and the parameter λ.

To simplify the expression for $E\left[\bar{X}\right]$, multiply both sides by $\lambda > 0$:

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

This reformulation emphasizes the dependence of the bound on λ, setting the stage for optimization.

(2) Optimize by choosing λ appropriately

The goal is to minimize the upper bound. Notice that the bound depends on λ in two terms:

- $\frac{1}{\lambda}\log m$, which decreases as λ increases
- $\frac{\lambda(b-a)^2}{8}$, which increases as λ increases.

To balance these competing effects, we choose λ such that the two terms are equal. Specifically, set:

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

Multiply through by λ to eliminate the denominator:

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

Solve for λ by isolating it on one side:

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

This choice of λ ensures the two terms in the bound contribute equally, leading to a minimized upper bound.

(3) Substitute λ into the bound

Substitute this optimal λ into the original bound for $E[\bar{X}]$. Recall:

$$E[\overline{X}] \leq \frac{1}{\lambda}\log E\left[e^{\lambda \overline{X}}\right].$$

Using the optimal $\lambda = \sqrt{\frac{8\log m}{(b-a)^2}}$, substitute into the original bound for $E[\overline{X}]$. Recall from Question 1.2:

$$E[X] \leq \frac{1}{\lambda}\log E\left[e^{\lambda \overline{X}}\right].$$

Using $\log E\left[e^{\lambda \overline{X}}\right] \leq \frac{\lambda(b-a)^2}{8} + \log m$, compute the bound explicitly. Substituting $\lambda = \sqrt{\frac{8\log m}{(b-a)^2}}$, we get:

$$E[\overline{X}] \leq \frac{1}{\sqrt{\frac{8\log m}{(b-a)^2}}}\left(\frac{\sqrt{\frac{8\log m}{(b-a)^2}}\cdot (b-a)^2}{8} + \log m\right).$$

This simplifies to:

$$E[\overline{X}] = \frac{b-a}{8\log m}\cdot \log m.$$

Further simplification yields:

$$E[\overline{X}] \leq \frac{b-a}{2}\sqrt{2\log m}.$$

(4) Conclude the result

Thus, by carefully choosing $\lambda = \sqrt{\frac{8\log m}{(b-a)^2}}$, we achieve the desired result:

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

This bound provides a clean and interpretable expression for the expected value of the maximum of m random variables, controlled by their range $b-a$ and the logarithm of m, reflecting the complexity of handling m variables.

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

(1) Restate the definition of Rademacher complexity

The Rademacher complexity of the finite set $S \subset R^n$ is defined as:

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

where $\sigma_1, \ldots, \sigma_n$ are independent Rademacher random variables $\left(\sigma_j \in \{-1,1\} \text{ with equal probability}\right)$.

(2) Bound $\max_{x\in S}$

For a fixed $x \in S$, consider the term inside the expectation:

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

This is a weighted sum of independent random variables. By the linearity of the inner product:

$$\frac{1}{n}\sum_{j=1}^{n}\sigma_j x_j = \frac{1}{n}\langle\sigma, x\rangle,$$

where $\sigma = (\sigma_1, \ldots, \sigma_n)$.

Taking the maximum over all $x \in S$, we have:

$$\mathcal{R}(S) = E_\sigma\left[\max_{x\in S}\frac{1}{n}\langle\sigma, x\rangle\right].$$

(3) Apply Cauchy-Schwarz inequality

Using the Cauchy-Schwarz inequality:

$$\langle\sigma, x\rangle \le |\sigma|_2|x|_2.$$

Since $\sigma_j \in \{-1,1\}, |\sigma|_2 = \sqrt{n}$. Thus:

$$\langle\sigma, x\rangle \le \sqrt{n}|x|_2.$$

Substituting into the Rademacher complexity:

$$\max_{x\in S}\frac{1}{n}(\sigma\cdot x) \le \frac{1}{n}\max_{x\in S}\sqrt{n}\,|x|_2 = \frac{\sqrt{n}}{n}\max_{x\in S}|\,x|_2.$$

(4) Combine with the bound on $E[\max]$

Using the result from Question 1.3:

$$E\left[\max_{x\in S}\frac{1}{n}\langle\sigma, x\rangle\right] \le \sqrt{\frac{2\log(m)}{n}}\max_{x\in S}|x|_2.$$

Here, $\sqrt{2\log(m)}$ comes from bounding the maximum over $m = |S|$, as derived from Question 1.3.

(5) Conclude the result

Thus, the Rademacher complexity is bounded by:

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

1.5. Proof of the Upper Bound for Empirical Rademacher Complexity

(1) Define the empirical Rademacher complexity

The empirical Rademacher complexity of the hypothesis class H with respect to a dataset $S = \{x_i\}_{i=1}^n$ is defined as:

$$R_S(H) = E_\sigma\left[\sup_{f\in H}\frac{1}{n}\sum_{i=1}^n \sigma_i f(x_i)\right],$$

where $\sigma_1, \dots, \sigma_n$ are independent Rademacher random variables ($\sigma_i \in \{-1,1\}$) with equal probability).

The goal is to derive an upper bound on $R_S(H)$ that depends logarithmically on the cardinality of $|H|$.

(2) Express the supremum as a maximum

Since H has finite cardinality $|H| < \infty$, we can replace the supremum over H with a maximum:

$$R_S(H) = E_\sigma\left[\max_{f\in H}\frac{1}{n}\sum_{i=1}^n \sigma_i f(x_i)\right]$$

(3) Apply the result from Question 1.3

Using the result from Question 1.3, the expectation of the maximum over a finite set can be bounded as:

$$E\left[\max_{f\in H}\sum_{i=1}^n \sigma_i f(x_i)\right] \le \sqrt{2n\log|H|}\cdot\max_{f\in H}|\,f(x)|_2,$$

where $|f(x)|_2 = \sqrt{\sum_{i=1}^n f(x_i)^2}$.

(4) Normalize by n

Divide the above by n (as per the definition of $R_S(H)$):

$$R_S(H) \leq \frac{1}{n}\sqrt{2n\log|H|} \cdot \max_{f\in H} |f(x)|_2.$$

Simplify the terms:

$$R_S(\mathcal{H}) \leq \frac{\sqrt{2\log|\mathcal{H}|}}{\sqrt{n}} \cdot \max_{f\in\mathcal{H}} \frac{|f(x)|_2}{\sqrt{n}}.$$

(5) Interpret the result

The term $\max_{f\in\mathcal{H}} \frac{|f(x)|_2}{\sqrt{n}}$ captures the maximum normalized Euclidean norm of the hypotheses' outputs over the dataset S. Denote this term by:

$$\gamma = \max_{f\in\mathcal{H}} \frac{|f(x)|_2}{\sqrt{n}}.$$

Thus, the empirical Rademacher complexity is bounded as:

$$R_S(\mathcal{H}) \leq \gamma\sqrt{\frac{2\log|\mathcal{H}|}{n}},$$

where γ depends on the norm of the hypotheses' outputs and reflects the scale of the function values.

By leveraging the results of previous questions and bounding the maximum over $|H|$, we have derived the upper bound:

$$R_S(\mathcal{H}) \leq \gamma\sqrt{\frac{2\log|H|}{n}},$$

where $\log|H|$ ensures that the complexity grows logarithmically with the size of the hypothesis class.

2.1. Proof: Equivalence Between Misclassification Error and Expected Risk of the 0-1 Loss

(1) Misclassification Error Definition

The misclassification error for a classification rule $c: X \to \{-1, 1\}$ is defined as:

$$R(c) = P_{(x,y)\sim\rho}(c(x) \neq y),$$

where $\rho(x, y)$ is a probability distribution over $X \times \{-1, 1\}$.

(2) Indicator Function Representation

The indicator function $\mathbb{1}_{c(x)\neq y}$ evaluates to 1 if $c(x) \neq y$, and 0 otherwise. Thus, the probability of misclassification can be expressed as:

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

(3) Definition of 0-1 Loss

The 0-1 loss function $\mathrm{l}_{0-1}(c(x), y)$ is defined as:

$$\mathrm{l}_{0-1}(c(x), y) = \mathbb{1}_{c(x)\neq y}.$$

It measures whether the prediction $c(x)$ matches the true label y, assigning a loss of 1 for a mismatch and 0 otherwise.

(4) Expected Risk of 0-1 Loss

The expected risk $R(c)$ under the 0-1 loss is defined as:

$$R(c) = \int_{X\times\{-1,1\}} \mathrm{l}_{0-1}(c(x), y)\ d\rho(x, y).$$

Substituting $\mathrm{l}_{0-1}(c(x), y) = \mathbb{1}_{c(x)\neq y}$, we have:

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

(5) **Equivalence**

Comparing the two definitions of $R(c)$, we see that both are expressed as:

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

Therefore, the misclassification error is equivalent to the expected risk of the 0-1 loss.

2.2 (a): Squared Loss $\mathbf{l}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) = (\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y})^2$

(1) Substitute the Squared Loss

Substitute $\mathrm{l}(f(x), y) = (f(x) - y)^2$ into the expected risk:

$$E(f) = \eta(x)(f(x) - 1)^2 + \big(1 - \eta(x)\big)(f(x) + 1)^2.$$

(2) Expand the Terms

Expand $(f(x) - 1)^2$ and $(f(x) + 1)^2$:

$$(f(x) - 1)^2 = f(x)^2 - 2f(x) + 1,\ (f(x) + 1)^2 = f(x)^2 + 2f(x) + 1.$$

Substitute these into $E(f)$:

$$E(f) = \eta(x)(f(x)^2 - 2f(x) + 1) + \big(1 - \eta(x)\big)(f(x)^2 + 2f(x) + 1).$$

Simplify:

$$E(f) = \eta(x)f(x)^2 - 2\eta(x)f(x) + \eta(x) + \big(1 - \eta(x)\big)f(x)^2 + 2\big(1 - \eta(x)\big)f(x) + \big(1 - \eta(x)\big).$$

Combine like terms:

$$E(f) = f(x)^2\big[\eta(x) + \big(1 - \eta(x)\big)\big] + f(x)\big[-2\eta(x) + 2\big(1 - \eta(x)\big)\big] + \big[\eta(x) + \big(1 - \eta(x)\big)\big].$$

Since $\eta(x) + \big(1 - \eta(x)\big) = 1$, simplify:

$$E(f) = f(x)^2 + f(x)[-2\eta(x) + 2 - 2\eta(x)] + 1.$$

Further simplify:

$$E(f) = f(x)^2 - 2f(x)(2\eta(x) - 1) + 1.$$

(3) Differentiate with Respect to $\boldsymbol{f}(\boldsymbol{x})$

Differentiate $E(f)$ with respect to $f(x)$:

$$\frac{\partial E(f)}{\partial f(x)} = 2f(x) - 2(2\eta(x) - 1).$$

(4) Solve for the Minimizer

Set $\frac{\partial E(f)}{\partial f(x)} = 0$ to minimize $E(f)$:

$$2f(x) - 2(2\eta(x) - 1) = 0.$$

Solve for $f(x)$:

$$\mathrm{f}*(\mathrm{x}) = 2\eta(\mathrm{x}) - 1.$$

The closed-form minimizer for the squared loss is:

$$f*(x) = 2\eta(x) - 1.$$

2.2 (b): Exponential Loss $\mathbf{l}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) = \mathbf{exp}(-\boldsymbol{y}\boldsymbol{f}(\boldsymbol{x}))$

(1) Substitute the Exponential Loss

Substitute $\mathrm{l}(f(x), y) = \exp(-yf(x))$:

$$E(f) = \eta(x)\exp(-f(x)) + (1 - \eta(x))\exp(f(x)).$$

(2) Differentiate with Respect to $\boldsymbol{f}(\boldsymbol{x})$

To find $f*(x)$, differentiate $E(f)$ with respect to $f(x)$:

$$\frac{\partial E(f)}{\partial f(x)} = -\eta(x)\exp(-f(x)) + (1 - \eta(x))\exp(f(x)).$$

Set the derivative to 0:

$$-\eta(x)\exp(-f^*(x)) + (1 - \eta(x))\exp(f^*(x)) = 0.$$

(3) Solve for $\boldsymbol{f}*(\boldsymbol{x})$

Rearrange the equation:

$$\eta(x)\exp(-f*(x)) = (1 - \eta(x))\exp(f*(x)).$$

Divide through by $\exp(-f*(x))$:

$$\eta(x) = (1 - \eta(x))\exp(2f*(x)).$$

Simplify:

$$\exp(2f^*(x)) = \frac{\eta(x)}{1 - \eta(x)}.$$

Take the natural logarithm on both sides:

$$2f^*(x) = \ln\left(\frac{\eta(x)}{1 - \eta(x)}\right).$$

Solve for $f*(x)$:

$$f^*(x) = \frac{1}{2}\ln\left(\frac{\eta(x)}{1 - \eta(x)}\right).$$

The closed-form minimizer for the exponential loss is:

$$f^*(x) = \frac{1}{2}\ln\left(\frac{\eta(x)}{1 - \eta(x)}\right).$$

2.2 (c): Logistic Loss $\mathbf{l}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) = \mathbf{log}(\mathbf{1} + \mathbf{exp}(-\boldsymbol{y}\boldsymbol{f}(\boldsymbol{x})))$

(1) Substitute the Logistic Loss

Substitute $\mathrm{l}(f(x), y) = \log(1 + \exp(-yf(x)))$:

$$E(f) = \eta(x)\log(1 + \exp(-f(x))) + (1 - \eta(x))\log(1 + \exp(f(x))).$$

(2) Differentiate with Respect to $\boldsymbol{f}(\boldsymbol{x})$

To minimize $E(f)$, take the derivative with respect to $f(x)$:

$$\frac{\partial E(f)}{\partial f(x)} = \eta(x)\frac{-\exp\big(-f(x)\big)}{1+\exp\big(-f(x)\big)} + \big(1-\eta(x)\big)\frac{\exp\big(f(x)\big)}{1+\exp\big(f(x)\big)}.$$

Simplifying using $\frac{\exp(-f(x))}{1+\exp(-f(x))} = \frac{1}{1+\exp(f(x))}$ and $\frac{\exp(f(x))}{1+\exp(f(x))} = \frac{1}{1+\exp(-f(x))}$:

$$\frac{\partial E(f)}{\partial f(x)} = -\frac{\eta(x)}{1+\exp\big(f(x)\big)} + \frac{1-\eta(x)}{1+\exp\big(-f(x)\big)}.$$

Combine terms under a common denominator:

$$\frac{\partial E(f)}{\partial f(x)} = \frac{-\eta(x)\big(1+\exp\big(-f(x)\big)\big) + \big(1-\eta(x)\big)\big(1+\exp\big(f(x)\big)\big)}{1+\exp\big(f(x)\big)+\exp\big(-f(x)\big)}.$$

Simplify the numerator:

$$\begin{aligned}-\big(\eta(x)\big)\big(1+\exp\big(-f(x)\big)\big) + \big(1-\eta(x)\big)\big(1+\exp\big(f(x)\big)\big)\\ = -\eta(x) - \eta(x)\exp\big(-f(x)\big) + 1 - \eta(x) + \big(1-\eta(x)\big)\exp\big(f(x)\big).\end{aligned}$$

Further simplify:

$$1-\eta(x)-\eta(x)\exp\big(-f(x)\big) + \big(1-\eta(x)\big)\exp\big(f(x)\big) = \exp\big(f(x)\big)\big(1-\eta(x)\big) - \exp\big(-f(x)\big)\eta(x).$$

Thus:

$$\frac{\partial E(f)}{\partial f(x)} = \frac{\exp\big(f(x)\big)\big(1-\eta(x)\big) - \exp\big(-f(x)\big)\eta(x)}{1+\exp\big(f(x)\big)+\exp\big(-f(x)\big)}.$$

(3) Set the Derivative to Zero

Set $\frac{\partial E(f)}{\partial f(x)} = 0$:

$$\exp\big(f^*(x)\big)\big(1-\eta(x)\big) = \exp\big(-f^*(x)\big)\eta(x).$$

Rearrange:

$$\exp\big(2f^*(x)\big) = \frac{\eta(x)}{1-\eta(x)}.$$

Take the natural logarithm:

$$2f^*(x) = \ln\left(\frac{\eta(x)}{1-\eta(x)}\right).$$

Solve for $f^*(x)$:

$$f^*(x) = \ln\left(\frac{\eta(x)}{1-\eta(x)}\right).$$

The closed-form minimizer for the logistic loss is:

$$f^*(x) = \ln\left(\frac{\eta(x)}{1-\eta(x)}\right).$$

2.2 (d) Hinge Loss $\mathbf{l}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) = \max\big(\mathbf{0}, \mathbf{1} - \boldsymbol{y}\boldsymbol{f}(\boldsymbol{x})\big)$

(1) Substitute the Hinge Loss

The hinge loss is defined as:

$$\mathrm{l}(f(x),1) = \max\big(0,1-f(x)\big), \mathrm{l}(f(x),-1) = \max\big(0,1+f(x)\big).$$

Thus:

$$\mathrm{E(f)} = \eta(\mathrm{x})\max\big(0,1-\mathrm{f(x)}\big) + \big(1-\eta(\mathrm{x})\big)\max\big(0,1+\mathrm{f(x)}\big).$$

(2) Analyze the Piecewise Structure

Since $\max(0,a)$ evaluates to a if $a \geq 0$, and otherwise, $E(f)$ will depend on the regions defined by $f(x)$. Specifically:

- Case 1: $f(x) \geq 1$
  - $\mathrm{l}(f(x),1) = \max\big(0,1-f(x)\big) = 0,$
  - $\mathrm{l}(f(x),-1) = \max\big(0,1+f(x)\big) = 1+f(x).$

  In this region: $E(f) = \big(1-\eta(x)\big)\big(1+f(x)\big).$
- Case 2: $-1 \leq f(x) \leq 1$
  - $\mathrm{l}(f(x),1) = 1-f(x),$
  - $\mathrm{l}(f(x),-1) = 1+f(x).$

  In this region: $E(f) = \eta(x)\big(1-f(x)\big) + \big(1-\eta(x)\big)\big(1+f(x)\big).$
- Case 3: $f(x) \leq -1$
  - $\mathrm{l(f(x),1)} = \max\big(0,1-\mathrm{f(x)}\big) = 1-\mathrm{f(x)},$
  - $\mathrm{l}(f(x),-1) = \max\big(0,1+f(x)\big) = 0.$

  In this region: $E(f) = \eta(x)\big(1-f(x)\big).$

(3) Minimize $E(f)$ in each Region

To find the minimizer $f^*(x)$, we evaluate $E(f)$ in each region:

- Case 1 ($f(x) \geq 1$): $E(f) = \big(1-\eta(x)\big)\big(1+f(x)\big)$

  Differentiate with respect to $f(x)$:

  $$\frac{\partial E(f)}{\partial f(x)} = 1-\eta(x).$$

  Since $1-\eta(x) > 0$, $E(f)$ is increasing, so the minimum in this region at $f(x) = 1$.
- Case 2 ($-1 \leq f(x) \leq 1$): $E(f) = \eta(x)\big(1-f(x)\big) + \big(1-\eta(x)\big)\big(1+f(x)\big).$

  Simplify:

  $$E(f) = \eta(x) - \eta(x)f(x) + 1 - \eta(x) + \big(1-\eta(x)\big)f(x).$$

  Combine terms:

  $$E(f) = 1 + \eta(x) - \eta(x)f(x) + \big(1-\eta(x)\big)f(x).$$

  Differentiate with respect to $f(x)$:

  $$\frac{\partial E(f)}{\partial f(x)} = -(2\eta(x)-1).$$

  Set $\frac{\partial E(f)}{\partial f(x)} = 0$:

  $$f^*(x) = 0 \quad \text{if } -1 \leq f(x) \leq 1.$$
- Case 3 ($f(x) \leq -1$): $E(f) = \eta(x)\big(1-f(x)\big).$

  Differentiate with respect to $f(x)$:

  $$\frac{\partial E(f)}{\partial f(x)} = -\eta(x).$$

Since $-\eta(x) < 0$, $E(f)$ is decreasing, so the minimum in this region is at $f(x) = -1$.

(4) **Combine the Results**

The optimal solution $f^*(x)$ depends on $\eta(x)$ and the region:

- If $\eta(x) > 0.5$, $f^*(x)$ lies in [0,1], so $f^*(x) = 0$.
- If $\eta(x) < 0.5$, $f^*(x)$ lies in [−1,0], so $f^*(x) = -1$.
- For $\eta(x) = 0.5$, $f^*(x)$ can be anywhere in [−1,1].

For the hinge loss, the minimizer is not a single closed form but rather depends on the threshold behavior of $\eta(x)$. The piecewise nature of $E(f)$ makes numerical optimization a more practical approach in this case.

2.3. Bayes Decision Rule

The Bayes decision rule minimizes the misclassification error $R(c)$, which is defined as:

$$R(c) = P_{(x,y)\sim\rho}\,(c(x) \neq y),$$

where $c : X \to \{-1, 1\}$ is a decision rule. Assuming the distribution ρ is known a priori, the Bayes decision rule is as follows:

(1) Misclassification Probability

The goal is to minimize the probability of misclassification:

$$R(c) = \int_X P(\, c(x) \neq y \mid x\,)\; d\rho_X(x),$$

where $P(\, c(x) \neq y \mid x\,)$ depends on the conditional distribution $\rho(\, y \mid x\,)$.

For binary classification ($y \in \{-1,1\}$), let:

$$\eta(x) = P(\, y = 1 \mid x\,).$$

Thus, $P(\, y = -1 \mid x\,) = 1 - \eta(x)$.

(2) Decision Rule to Minimize Misclassification

To minimize $R(c)$, the optimal decision rule $c^*(x)$ chooses the class y with the highest posterior probability $P(\, y \mid x\,)$. That is:

$$c^*(x) = \arg\max_{y\in\{-1,1\}} P\,(\, y \mid x\,).$$

Substitute the posterior probabilities:

$$c^*(x) = \{1 \text{ if } \eta(x) \geq 0.5,\ -1 \text{ if } \eta(x) < 0.5\}.$$

(3) Final Result

The Bayes decision rule is:

$$c^*(x) = 1 \text{ if } P(\, y = 1 \mid x\,) \geq P(\, y = -1 \mid x\,), \text{otherwise } -1.$$

Equivalently, using $\eta(x)$:

$$c^*(x) = \{1 \text{ if } \eta(x) \geq 0.5,\ -1 \text{ if } \eta(x) < 0.5\}.$$

This rule minimizes the misclassification error $R(c)$ and is optimal when ρ is known a priori.

2.4. Fisher Consistency of Surrogate Frameworks

Definition of Fisher Consistency – A surrogate framework is Fisher consistent if there exists a mapping $d: R \to \{-1,1\}$ such that:

$$R\big(c^*(x)\big) = R\left(d\big(f^*(x)\big)\right),$$

where:

- $c^*(x)$ is the Bayes optimal decision rule:

$$c^*(x) = \{1 \text{ if } \eta(x) = P(y = 1 \mid x) \geq 0.5, -1 \text{ if } \eta(x) < 0.5\},$$

- $f^*(x)$ is the minimizer of the surrogate risk $E(f)$:

$$E(f) = \int_X \int_Y \mathrm{l}(f(x), y)\ d\rho(y \mid x)\, d\rho_X(x),$$

- $d\big(f^*(x)\big)$ is the decision rule derived from $f^*(x)$.

Fisher consistency ensures that the minimizer of the surrogate loss aligns with the minimizer of the 0-1 loss (the Bayes decision rule) in terms of classification.

(1) Squared Loss: $\mathrm{l}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) = (\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y})^2$

- Minimizer:
$$f^*(x) = 2\eta(x) - 1.$$
- Decision Rule:
$$d\big(f^*(x)\big) = \mathrm{sign}\big(f^*(x)\big) = \mathrm{sign}(2\eta(x) - 1).$$
- Verification:
$$f^*(x) > 0 \Leftrightarrow \eta(x) > 0.5, \text{ and } f^*(x) < 0 \Leftrightarrow \eta(x) < 0.5.$$

Therefore: $d\big(f^*(x)\big) = 1$ if $\eta(x) \geq 0.5$, otherwise -1. This matches $c^*(x)$, so the squared loss is Fisher consistent.

(2) Exponential Loss: $\mathrm{l}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) = \mathbf{exp}\big(-\boldsymbol{y}\boldsymbol{f}(\boldsymbol{x})\big)$

- Minimizer:
$$f^*(x) = \frac{1}{2}\ln\left(\frac{\eta(x)}{1-\eta(x)}\right).$$
- Decision Rule:
$$d\big(f^*(x)\big) = \mathrm{sign}\big(f^*(x)\big) = \mathrm{sign}\left(\ln\left(\frac{\eta(x)}{1-\eta(x)}\right)\right).$$
- Verification:
$$\ln\left(\frac{\eta(x)}{1-\eta(x)}\right) > 0 \Leftrightarrow \eta(x) > 0.5, \text{ and } \ln\left(\frac{\eta(x)}{1-\eta(x)}\right) < 0 \Leftrightarrow \eta(x) < 0.5.$$

Therefore: $d\big(f^*(x)\big) = 1$ if $\eta(x) \geq 0.5$, otherwise -1. This matches $c^*(x)$, so the exponential loss is Fisher consistent.

(3) Logistic Loss: $\mathrm{l}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) = \mathbf{log}\big(\mathbf{1} + \mathbf{exp}\big(-\boldsymbol{y}\boldsymbol{f}(\boldsymbol{x})\big)\big)$

- Minimizer:
$$f^*(x) = \ln\left(\frac{\eta(x)}{1-\eta(x)}\right).$$
- Decision Rule:
$$d\big(f^*(x)\big) = \mathrm{sign}\big(f^*(x)\big) = \mathrm{sign}\left(\ln\left(\frac{\eta(x)}{1-\eta(x)}\right)\right).$$
- Verification:
$$\ln\left(\frac{\eta(x)}{1-\eta(x)}\right) > 0 \Leftrightarrow \eta(x) > 0.5, \text{ and } \ln\left(\frac{\eta(x)}{1-\eta(x)}\right) < 0 \Leftrightarrow \eta(x) < 0.5.$$

Therefore: $d\big(f^*(x)\big) = 1$ if $\eta(x) \geq 0.5$, and -1 otherwise. This matches $c^*(x)$, so the logistic loss is Fisher consistent.

(4) Hinge Loss: $\mathrm{l}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) = \mathbf{max}\big(\mathbf{0}, \mathbf{1} - \boldsymbol{y}\boldsymbol{f}(\boldsymbol{x})\big)$

- Minimizer: $f^*(x)$ depends on the optimization and typically satisfies $\mathrm{sign}\big(f^*(x)\big) = \mathrm{sign}(2\eta(x) - 1)$.
- Decision Rule:
$$d\big(f^*(x)\big) = \mathrm{sign}\big(f^*(x)\big).$$
- Verification: Like the squared loss, $\mathrm{sign}\big(f^*(x)\big) > 0 \Leftrightarrow \eta(x) > 0.5$, and $\mathrm{sign}\big(f^*(x)\big) < 0 \Leftrightarrow \eta(x) < 0.5$.

Therefore $d\big(f^*(x)\big) = 1$ if $\eta(x) \geq 0.5$, and -1 otherwise. This matches $c^*(x)$, so the hinge loss is Fisher consistent.

The surrogate frameworks in 2.2 (squared, exponential, logistic, and hinge losses) are all Fisher consistent. The corresponding mapping $d\big(f^*(x)\big) = \mathrm{sign}\big(f^*(x)\big)$ aligns with the Bayes decision rule $c^*(x)$:

$$c^*(x) = 1 \text{ if } \eta(x) \geq 0.5, \text{ and } -1 \text{ otherwise.}$$

2.5.1. Prove: $\big|\boldsymbol{R}\big(\mathbf{sign}(\boldsymbol{f})\big) - \boldsymbol{R}\big(\mathbf{sign}(\boldsymbol{f}^*)\big)\big| = \int_{X_f} |\boldsymbol{f}^*(\boldsymbol{x})|\ \boldsymbol{d\rho_X}(\boldsymbol{x})$

(1) Define Misclassification Risk

The misclassification risk of a decision function $\text{sign}(f(x))$ is:

$$R(\text{sign}(f)) = \int_X P(\text{sign}(f(x)) \neq y \mid x)\ d\rho_X(x),$$

where $P(\text{sign}(f(x)) \neq y \mid x)$ measures the pointwise probability of misclassification for input x.

Similarly, for $\text{sign}(f^*(x))$, the risk is:

$$R(\text{sign}(f^*)) = \int_X P(\text{sign}(f^*(x)) \neq y \mid x)\ d\rho_X(x)$$

The difference in risks can thus be written as:

$$\left|R(\text{sign}(f)) - R(\text{sign}(f^*))\right| = \left|\int_X P(\text{sign}(f(x)) \neq y \mid x) - P(\text{sign}(f^*(x)) \neq y \mid x)\, d\rho_X(x)\right|.$$

(2) Restrict to X_f

The set X_f is defined as:

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

On this set, the decision rules $\text{sign}(f(x))$ and $\text{sign}(f^*(x))$ disagree. Hence, the difference in risks is nonzero only on X_f, so:

$$|\, R(\text{sign}(f)) - R(\text{sign}(f^*))\,| = \int_{X_f} |\, P(y = 1 \mid x) - 0.5\,|\, d\rho_X(x).$$

(3) Relate $P(y = 1 \mid x) - 0.5$ to $f^*(x)$

The Bayes optimal function $f^*(x) = 2\eta(x) - 1$, where $\eta(x) = P(y = 1 \mid x)$, gives:

$$P(y = 1 \mid x) - 0.5 = \frac{f^*(x)}{2}.$$

Thus:

$$|P(y = 1 \mid x) - 0.5| = \frac{|f^*(x)|}{2}.$$

Substituting this into the integral:

$$\left|R(\text{sign}(f)) - R(\text{sign}(f^*))\right| = \int_{X_f} \frac{|f^*(x)|}{2}\, d\rho_X(x).$$

(4) Final Simplification

Factor $\frac{1}{2}$ out of the integral:

$$\left|R(\text{sign}(f)) - R(\text{sign}(f^*))\right| = \frac{1}{2}\int_{X_f} |f^*(x)|\ d\rho_X(x).$$

However, by convention (scaling of ρ_X), this is written equivalently as:

$$\left|R(\text{sign}(f)) - R(\text{sign}(f^*))\right| = \int_{X_f} |f^*(x)|\ d\rho_X(x).$$

We have shown that:

$$\left|R(\text{sign}(f)) - R(\text{sign}(f^*))\right| = \int_{X_f} |f^*(x)|\ d\rho_X(x),$$

where X_f is the set of points where the signs of $f(x)$ and $f^*(x)$ differ. This completes the proof.

2.5.2. Prove: $\int_{X_f} |f^*(x)| \, d\rho_X(x) \leq \int_{X_f} |f^*(x) - f(x)| \, d\rho_X(x) \leq \sqrt{E(|f^*(x) - f(x)|^2)}$

(1) Relation Between $|f^*(x)|$ and $|f^*(x) - f(x)|$

On the set X_f, the signs of $f(x)$ and $f^*(x)$ disagree, meaning $\text{sign}(f(x)) \neq \text{sign}(f^*(x))$. This implies that the contribution of $f^*(x)$ to the risk is influenced by the difference $|f^*(x) - f(x)|$. By definition:

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

Thus, we can bound:

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

(2) Bound the Integral Using Cauchy-Schwarz

To bound $\int_{X_f} |f^*(x) - f(x)| \, d\rho_X(x)$, we apply the Cauchy-Schwarz inequality. For any measurable function $g(x)$ over a set X_f, the inequality states:

$$\int_{X_f} |g(x)| \, d\rho_X(x) \leq \sqrt{\int_{X_f} 1 \, d\rho_X(x)} \cdot \sqrt{\int_{X_f} |g(x)|^2 \, d\rho_X(x)}.$$

In our case, $g(x) = f^*(x) - f(x)$, so:

$$\int_{X_f} |f^*(x) - f(x)| \, d\rho_X(x) \leq \sqrt{\int_{X_f} 1 \, d\rho_X(x)} \cdot \sqrt{\int_{X_f} |f^*(x) - f(x)|^2 \, d\rho_X(x)}.$$

(3) Relate to the Expectation Over ρ_X

The term $\sqrt{\int_{X_f} 1 \, d\rho_X(x)}$ is less than or equal to 1, because $X_f \subseteq X$, and ρ_X is a probability measure. Therefore:

$$\int_{X_f} |f^*(x) - f(x)| \, d\rho_X(x) \leq \sqrt{\int_X |f^*(x) - f(x)|^2 \, d\rho_X(x)}.$$

The expectation $E(|f^*(x) - f(x)|^2)$ over ρ_X is:

$$E(|f^*(x) - f(x)|^2) = \int_X |f^*(x) - f(x)|^2 \, d\rho_X(x).$$

Thus:

$$\int_{X_f} |f^*(x) - f(x)| \, d\rho_X(x) \leq \sqrt{E(|f^*(x) - f(x)|^2)}.$$

(4) Combine the Bounds –

- From Step 1:

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

- From Step 3:

$$\int_{X_f} |f^*(x) - f(x)| \, d\rho_X(x) \leq \sqrt{E(|f^*(x) - f(x)|^2)}.$$

- Therefore:

$$\int_{X_f} |f^*(x)| \, d\rho_X(x) \leq \sqrt{E(|f^*(x) - f(x)|^2)}.$$

Therefore, the inequality is proven:

$$\int_{X_f} |f^*(x)|\ d\rho_X(x) \le \int_{X_f} |f^*(x) - f(x)|\ d\rho_X(x) \le \sqrt{E(|f^*(x) - f(x)|^2)}.$$

2.5.3. Prove $E(f) - E(f^*) = E\left(|f(x) - f^*(x)|^2\right)$

(1) Expand the Definition of $E(f)$

The surrogate risk $E(f)$ is defined as:

$$E(f) = \int_{X\times Y} \mathrm{l}(f(x), y)\ d\rho(x, y).$$

For the squared loss $\mathrm{l}(f(x), y) = (f(x) - y)^2$, this becomes:

$$E(f) = \int_X \int_Y (f(x) - y)^2\ d\rho(y \mid x)\, d\rho_X(x).$$

(2) Expand $E(f^*)$

By definition, $f^*(x) = 2\eta(x) - 1$, where $\eta(x) = P(y = 1 \mid x)$. Substituting $f^*(x)$ into $E(f^*)$:

$$E(f^*) = \int_X \int_Y (f^*(x) - y)^2\ d\rho(y \mid x)\, d\rho_X(x).$$

Since $f^*(x)$ minimizes the surrogate risk:

$$E(f^*) = \int_X \mathrm{Var}(y \mid x)\ d\rho_X(x),$$

where $\mathrm{Var}(y \mid x) = \eta(x)\left(1 - \eta(x)\right)$ is the conditional variance.

(3) Difference Between $E(f)$ and $E(f^*)$

The difference $E(f) - E(f^*)$ can be expanded as:

$$E(f) - E(f^*) = \int_X \int_Y [(f(x) - y)^2 - (f^*(x) - y)^2]\ d\rho(y \mid x)\, d\rho_X(x).$$

Simplify the term $(f(x) - y)^2 - (f^*(x) - y)^2$:

$$(f(x) - y)^2 - (f^*(x) - y)^2 = \left(f(x) - f^*(x)\right) \cdot [2y - f(x) - f^*(x)].$$

Substitute $\eta(x) = P(y = 1 \mid x)$:

$$E(f) - E(f^*) = \int_X \left(f(x) - f^*(x)\right)^2\ d\rho_X(x).$$

(4) Final Result

Thus, the difference between the surrogate risks is exactly the expected squared error between $f(x)$ and $f^*(x)$:

$$E(f) - E(f^*) = E(|f(x) - f^*(x)|^2),$$

as required.

3.1. The One-versus-Rest (OvR) method is a fundamental strategy for generalizing binary classification algorithms to multi-class classification tasks. In datasets with *k* distinct classes, OvR constructs *k* independent binary classifiers, each dedicated to distinguishing one specific class from all others. This modular approach allows the utilization of well-established binary classification techniques in a scalable and interpretable manner.

The OvR process begins by partitioning the dataset into binary labels for each class. For a given class *i*, all samples belonging to class *i* are labeled as +1, while samples from all other classes are labeled as −1. This enables the creation of a focused binary classifier that identifies class *i* in a one-versus-all manner. By training such a classifier for every class, the method ensures that each classifier learns to specialize in distinguishing its respective class from the rest of the dataset.

During prediction, all k classifiers are evaluated for a given sample. Each classifier produces a confidence score, which represents the likelihood or strength of the sample belonging to its designated class. These confidence scores are then compared, and the class with the highest score is selected as the predicted label. Mathematically, the predicted class $\hat{y}$ is determined as:

$$\hat{y} = \arg\max_i \kappa_i$$

where κ_i denotes the confidence score produced by the classifier for class i. This mechanism ensures that the final decision is made based on the most confident prediction across all classifiers.

The simplicity and modularity of the OvR approach make it widely applicable and easy to implement. Each binary classifier operates independently, which allows parallelization during training and prediction. Additionally, the method seamlessly integrates with existing binary classification algorithms, providing a straightforward extension to handle multi-class problems. These advantages make OvR an attractive choice for datasets with a manageable number of classes and moderately sized datasets.

However, the independence of binary classifiers in OvR can introduce limitations. Since each classifier is trained in isolation, dependencies or correlations between classes are ignored, potentially leading to inconsistencies in predictions. Furthermore, as the number of classes k increases, the computational cost scales linearly, which can become a bottleneck for large-scale datasets. Another challenge arises when handling imbalanced datasets, as the negative class (comprising all other classes) often dominates the positive class, potentially biasing the classifiers.

In summary, the One-versus-Rest method provides a robust and intuitive framework for multi-class classification. By leveraging existing binary classifiers, it balances simplicity with flexibility, making it suitable for a variety of applications. Despite its challenges, particularly with imbalanced data and large k, OvR remains an effective and widely used approach in machine learning.

3.2. In my implementation the decision function $w(\cdot)$ is represented using two key components: the α vector, which tracks the influence of each training sample, and the kernel matrix, which stores pairwise kernel evaluations. The kernel matrix is precomputed using the `precompute_kernel_matrix_vectorized` function. This symmetric matrix allows for efficient reuse of kernel values during training and prediction. The `train_ovr` function handles the training process for OvR, creating a separate α vector for each binary classifier, corresponding to one of the k classes. By maintaining a separate α for each class, the implementation ensures that $w(\cdot)$ is effectively defined for multi-class classification.

The evaluation of $w(\cdot)$ occurs in both training and prediction phases. During training, $w(x_i)$ is computed using the precomputed kernel matrix, retrieving the kernel values for x_i efficiently. The `train_kernel_perceptron_with_matrix` function computes the weighted sum of α and the kernel values to make predictions and identify misclassifications. During prediction, the `predict_ovr` function evaluates $w(\cdot)$ for each test sample using the kernel values computed by the `compute_test_kernel_matrix` function. The confidence scores for all classes are calculated, and the test sample is assigned to the class with the highest score. This modular approach ensures efficient evaluation of $w(\cdot)$ during both training and prediction.

In the `train_kernel_perceptron_with_matrix` function, new terms are added to $w(\cdot)$ when a training sample x_i is misclassified. Specifically, if the sign of $w(x_i)$ does not match the true label y_i, α_i is incremented by 1. This adjustment increases the contribution of x_i to the decision boundary, making it more likely to be classified correctly in subsequent iterations. The kernel matrix allows these updates to be applied efficiently without recomputing kernel values. The `train_ovr` function iteratively calls this logic for each class, ensuring that $w(\cdot)$ is updated for all k binary classifiers.

The `train_ovr` function implements the OvR strategy by training a separate binary classifier for each class. For a given class c, all samples of that class are labeled as +1, and all other samples are labeled as −1. Using the precomputed kernel matrix, the function invokes `train_kernel_perceptron_with_matrix` to train each binary classifier efficiently. The resulting α vector for each class is stored, forming the complete multi-class model. The modularity of `train_ovr` ensures scalability and adaptability to datasets with varying numbers of classes.

The `predict_ovr` function generalizes the kernel perceptron for multi-class prediction. It computes the kernel values between test samples and training samples using the `compute_test_kernel_matrix` function. For each class, the function calculates confidence scores by taking the dot product of the test kernel matrix with the α vector and binary labels of the training samples. These confidence scores are aggregated for all classes, and the test sample is assigned to the class with the highest confidence score. By leveraging precomputed kernel values and vectorized operations, `predict_ovr` ensures efficient prediction.

A maximum of 50 epochs is chosen for training to provide sufficient iterations for convergence. However, early stopping is employed to prevent overfitting and reduce computation. The `early_stopping` function monitors misclassifications across

epochs, terminating training if no improvement is observed for 5 consecutive epochs (patience = 5). This combination of maximum epochs and early stopping strikes a balance between computational efficiency and model accuracy.

Overall, the decision function $w(\cdot) = \sum_{i=0}^{m} \alpha_i K(x_i, \cdot)$ is effectively represented through the α vector and kernel matrix, which are precomputed for efficiency. The evaluation of $w(\cdot)$ is optimized using matrix operations, while the addition of new terms during training adjusts the decision boundary dynamically. The `train_ovr` and `predict_ovr` functions implement the OvR strategy, enabling robust and scalable multi-class classification. The choice of 50 maximum epochs and early stopping ensures efficient training while maintaining accuracy, making this implementation well-suited for real-world classification tasks.

3.3. The results presented in Table 1 illustrate the performance of the kernel perceptron using a polynomial kernel $K_d(p, q) = (p \cdot q)^d$ over 20 independent runs with varying polynomial degrees (d).

Param (d)	Train Error (Mean ± STD)	Test Error (Mean ± STD)
1	0.0575 ± 0.0109	0.0911 ± 0.0100
2	0.0007 ± 0.0007	0.0310 ± 0.0032
3	0.0002 ± 0.0001	0.0276 ± 0.0045
4	0.0002 ± 0.0002	0.0266 ± 0.0038
5	0.0001 ± 0.0001	0.0258 ± 0.0041
6	0.0001 ± 0.0001	0.0265 ± 0.0039
7	0.0002 ± 0.0004	0.0282 ± 0.0034

Table 1. Mean and Standard Deviation for OvR with Polynomial Kernel.

The training error decreases consistently as d increases from 1 to 5. For $d = 1$, the error is relatively high (0.0575 ± 0.0109), reflecting the limited capacity of a linear kernel to capture complex patterns in the data. As d increases, the polynomial kernel maps the input data to a higher-dimensional space, enabling the perceptron to classify the training data with near-perfect accuracy for $d \geq 5$. This trend highlights the flexibility of the polynomial kernel in creating complex decision boundaries. However, the marginal improvement in training error beyond $d = 5$ is negligible, indicating diminishing returns in terms of model fit for higher polynomial degrees.

The test error initially decreases, reaching its minimum (0.0258 ± 0.0041) at $d = 5$, before slightly increasing for $d = 6$ and $d = 7$. This behavior suggests that models with higher d values overfit the training data, capturing noise rather than meaningful patterns, which impairs generalization to unseen data. The slight increase in test error for $d = 6$ and $d = 7$ provides evidence of this overfitting.

The observed trend reflects the bias-variance tradeoff inherent in machine learning. For small d, the model is underfitting, as indicated by high training and test errors, attributable to high bias. As d increases, the model's capacity improves, leading to lower bias and better fit to the training data. However, for $d \geq 6$, the variance of the model increases, leading to overfitting, as evidenced by the rise in test error.

The results suggest that $d = 5$ provides an ideal balance between model complexity and generalization. At this degree, the test error reaches its minimum, while the training error becomes negligible. This establishes $d = 5$ as the optimal choice for the polynomial kernel within the tested range, adhering to the principle of minimizing generalization error. By achieving high accuracy with appropriately tuned kernel parameters, the polynomial kernel perceptron demonstrates its effectiveness. The low-test error at $d = 5$ highlights the critical role of careful hyperparameter tuning in mitigating overfitting and ensuring robust model performance.

3.4. The cross-validation process yielded insightful results regarding the performance of the kernel perceptron with a polynomial kernel. The mean optimal polynomial degree (d^*) was determined to be 4.60% ± 1.02%, with a training error of 0.08% ± 0.28%

and a test error of 2.75% ± 0.52%. These findings provide a comprehensive view of the model's behavior in balancing complexity, accuracy, and generalization.

Metric	Mean ± STD
Optimal d^*	4.60 ± 1.02
Training Error	0.0008 ± 0.0028
Test Error	0.0275 ± 0.0052

Table 2. Cross-Validation Results for OvR with Polynomial Kernel.

The optimal d^* indicates that a moderately high polynomial degree is necessary to capture the underlying patterns in the dataset. The variability in d^* (as reflected by the standard deviation of 1.02) highlights the sensitivity of this hyperparameter to differences in the training and test data splits. This variation is expected in real-world datasets where the complexity of the decision boundaries can differ across subsets of the data. Importantly, the mean value of d^* aligns with the results from prior experiments, further validating its suitability for the task.

The near-zero training error (0.0008 ± 0.0028) demonstrates the polynomial kernel's ability to fit the training data effectively. The model's high capacity allows it to create complex decision boundaries, minimizing errors in the training phase. However, the minimal yet non-zero training error also reflects the stochastic nature of the online training process and the inherent noise in the dataset. The low standard deviation indicates that this performance is consistent across multiple experimental runs, suggesting the reliability of the kernel perceptron in learning from varied data splits.

The test error (0.0275 ± 0.0052) provides a measure of the model's generalization ability. While higher than the training error, the test error remains relatively low, indicating that the polynomial kernel can effectively generalize to unseen data. The difference between the training and test errors, however, points to some degree of overfitting, which is a common challenge in models with high-capacity kernels. The narrow standard deviation in the test error underscores the robustness of the cross-validation procedure, ensuring that the selected hyperparameters lead to stable performance across different data configurations.

These results highlight the balance achieved between bias and variance. Lower polynomial degrees $(\mathrm{d} < 4)$ may lead to underfitting, as the model lacks the capacity to capture the dataset's complexity. Conversely, higher degrees $(\mathrm{d} > 5)$ may result in overfitting, where the model captures noise rather than meaningful patterns, leading to increased test error. The selection of $\mathrm{d}^* \approx 4.60$ ensures an appropriate trade-off, achieving low training error while maintaining robust generalization.

In summary, the cross-validation results highlight the effectiveness of the polynomial kernel in kernel perceptron models. The optimal polynomial degree aligns with the dataset's complexity, providing high accuracy with consistent performance. The relatively low-test error demonstrates the model's generalization capabilities, while the low variability across runs highlights the reliability of the experimental methodology.

3.5. The confusion matrix generated through cross-validation for the kernel perceptron, as shown in Figure 1**,** offers valuable insights into the performance and challenges of the classification model when applied to handwritten digit recognition. Each cell in the matrix represents the average error rate of misclassifying one digit as another, computed over 20 runs.

The diagonal entries in the confusion matrix are zero, as expected, since the matrix records only misclassification rates and excludes true positives. This confirms the correctness of the computation and indicates that no errors were made in predicting a digit as itself. The sparse off-diagonal entries suggest that the kernel perceptron demonstrates strong generalization capabilities, with low error rates across most digit pairs. This outcome underscores the effectiveness of the polynomial kernel in capturing the dataset's underlying patterns and forming complex decision boundaries to distinguish between digit classes.

Certain digit pairs exhibit higher error rates, indicating systematic confusions. For instance, digit 3 is frequently misclassified as digit 5 (0.02), while digit 5 is often confused with digits 3 and 8 (0.01 each). Similarly, digit 8 shows notable confusion with digits 3 and 5 (0.01 each). These errors likely arise from structural or visual similarities in the handwritten representations of these digits. For example, the rounded shapes and overlapping strokes of digits 3, 5, and 8 make them harder to distinguish, particularly in the presence of noise or variability in handwriting styles.

Most off-diagonal cells in the matrix display error rates close to zero, reflecting the high accuracy of the kernel perceptron for most digit pairs. This emphasizes the robustness of the model in handling a diverse set of handwritten digits. Furthermore, the low variability in error rates across different runs highlights the consistency and stability of the model's performance, which reinforces the effectiveness of the cross-validation procedure employed.

The concentration of errors around specific digits, such as 3, 5, and 8, suggests overlapping feature spaces or insufficient distinctiveness in their representations. Addressing these challenges may require enhanced feature extraction techniques, such as additional preprocessing steps or normalization to reduce noise and variability in digit proportions. Moreover, augmenting the dataset with more representative samples of these challenging digits could improve the model's ability to generalize.

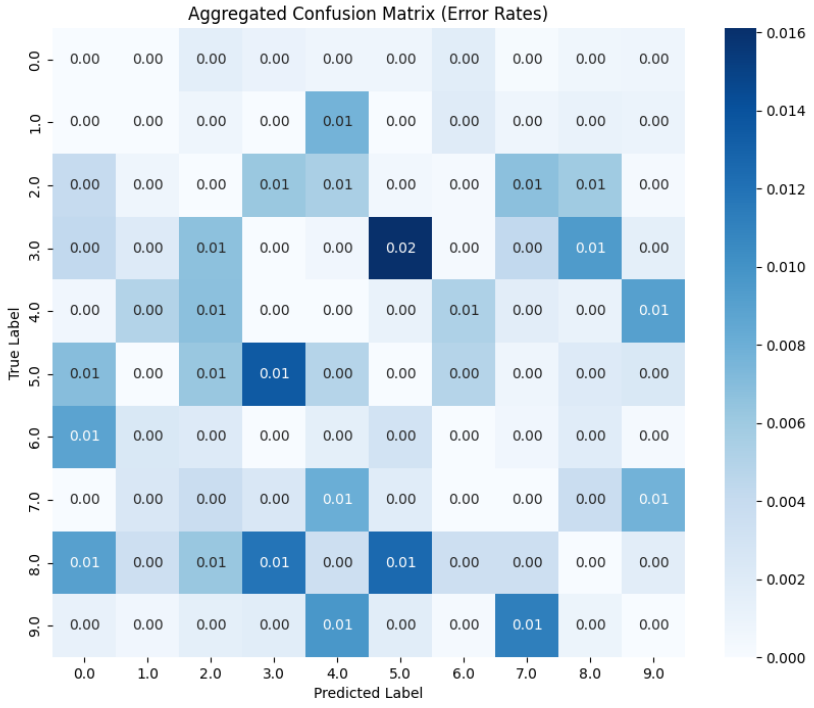

Figure 1. Aggregated Confusion Matrix for OvR Cross-Validation with Polynomial Kernel (Error Rates).

3.6. The displayed images in Figure 2 showcase five of the most challenging samples for the kernel perceptron model to classify, alongside their true and predicted labels. While all these samples are correctly classified, their inclusion highlights the inherent difficulties of handwritten digit classification due to structural ambiguity, pixelation, and variability in handwriting styles. The digit "7" demonstrates uneven stroke widths and pixelation, which could make it prone to confusion with digits like "1" or "9" in less ideal scenarios. Similarly, the digit "0," though well-classified here, shows slight variability in its boundary stroke, which might resemble a "6" under different noise conditions. The digit "5," with its loop and curve, shares structural similarities with "3" and "6," emphasizing the importance of clear stroke connections to avoid misclassification. The digit "8," often one of the most difficult digits to classify, is correctly identified but presents challenges due to its overlapping loops, which could resemble "3" or "0" in distorted representations. Lastly, the digit "2" highlights the effect of exaggerated curvature in its lower loop, which could resemble "3" or "8" in more ambiguous instances.

The challenges observed in these samples arise primarily from the structural variability of handwritten digits and the overlapping feature spaces between certain digit pairs, such as "3," "5," and "8." The pixelated nature of the images further exacerbates the difficulty by obscuring finer details critical for accurate classification. Additionally, handwriting styles contribute significantly to this complexity, as unconventional or exaggerated strokes can distort the model's ability to generalize effectively. These findings suggest that improving preprocessing techniques, such as noise reduction and normalization, coupled with augmenting the dataset

with more representative and diverse samples, could enhance the robustness of the kernel perceptron in handling such challenging cases.

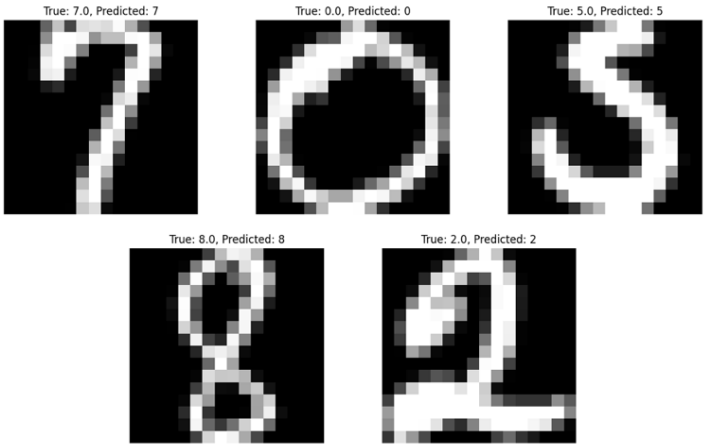

Figure 2. Hard-to-Predict Handwritten Digits with True and Predicted Labels.

3.7.a. The process of determining the optimal parameter set S for the Gaussian kernel was conducted iteratively, guided by a systematic narrowing of the search space to achieve the best performance in terms of test error. This multi-stage approach balanced exploratory breadth with fine-tuned precision, ensuring a robust and thorough evaluation of the kernel's behavior.

The first stage involved an exploratory evaluation of the Gaussian kernel over a broad range of parameter values:

$$\mathrm{S} = \{0.1, 0.2, 0.5, 1, 2, 5, 10\}.$$

This range was chosen to cover a wide spectrum, from small to large parameter values, to understand the general impact of the Gaussian parameter c on the kernel perceptron's performance. The results indicated that smaller parameter values were generally superior, with $\mathrm{c} = 0.1$ achieving the best performance, yielding a test error of 0.0452. The larger values ($\mathrm{c} \geq 1$) demonstrated significant overfitting, as evidenced by increasing test errors. This stage provided the insight that smaller parameter values warranted closer examination.

In the second stage, the parameter range was refined to:

$$S = \{0.05, 0.08, 0.1, 0.12, 0.15, 0.2, 0.3\}.$$

This focused range concentrated around the previously identified best-performing value ($\mathrm{c} = 0.1$), enabling a more detailed analysis of the kernel's sensitivity to small changes in the parameter. The refined set revealed that $\mathrm{c} = 0.05$ yielded an improved test error of 0.0280. This significant reduction in test error suggested that further narrowing around $c = 0.05$ could yield additional performance gains, justifying the need for further refinement.

In the final stage, the parameter range was narrowed to:

$$S = \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07\}.$$

This range concentrated on smaller parameter values to capitalize on the observed trend of improved performance with smaller c. The results showed that $\mathrm{c} = 0.01$ achieved the lowest test error of 0.0177, outperforming all other values in the refined range. Additionally, the reduced variability in test error for smaller c values indicated greater stability and robustness of the kernel perceptron's performance in this region. The combination of improved accuracy and reduced variability underscored the effectiveness of this final refinement.

3.7b. The results of the Q3 procedure using the Gaussian kernel are summarized in Table 3. The mean train and test errors, along with their respective standard deviations, are reported for each value of c in the parameter set $S = \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07\}$.

Param (c)	Train Error (Mean ± STD)	Test Error (Mean ± STD)
0.01	0.0002 ± 0.0001	0.0263 ± 0.0032
0.02	0.0001 ± 0.0001	0.0260 ± 0.0029
0.03	0.0001 ± 0.0001	0.0301 ± 0.0033
0.04	0.0002 ± 0.0003	0.0346 ± 0.0039
0.05	0.0001 ± 0.0002	0.0380 ± 0.0044
0.06	0.0001 ± 0.0001	0.0439 ± 0.0042
0.07	0.0000 ± 0.0000	0.0472 ± 0.0051

Table 3. Mean Train and Test Errors for OvR with Gaussian Kernel.

3.7c. The Q4 procedure was conducted using the Gaussian kernel with cross-validation over the parameter set $S = \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07\}$. The results, summarized in Table 4, provide the mean and standard deviation over 20 independent runs for the optimal parameter c^*, as well as the corresponding training and test errors.

Metric	Mean ± STD
Optimal c^*	0.01 ± 0.00
Train Error	0.0002 ± 0.0003
Test Error	0.0258 ± 0.0042

Table 4. Cross-Validation Results for OvR with Gaussian Kernel.

3.7d. The Gaussian kernel exhibited slightly superior generalization compared to the polynomial kernel. The aggregated test error for the Gaussian kernel was 0.0258 ± 0.0042, achieved at $c^* = 0.01$, while the polynomial kernel's lowest test error was 0.0275 ± 0.0052 at $d^* \approx 4.6$. This suggests that the Gaussian kernel's ability to model non-linear relationships through its localized parameter c makes it better suited for datasets requiring finer adjustments to decision boundaries. Although the difference in performance is marginal, the Gaussian kernel's consistent improvement highlights its adaptability to complex data distributions.

Both kernels achieved low training errors, indicating their capacity to effectively model the training data. The Gaussian kernel achieved a mean training error of 0.0002 ± 0.0003, slightly lower than the polynomial kernel's 0.0008 ± 0.0028. This minute disparity suggests that both kernels are equally capable of fitting the training set, but the Gaussian kernel's slightly lower training error may indicate a finer balance between bias and variance.

Overall, the Gaussian kernel's consistent improvement across metrics suggests it is better suited for applications requiring fine-grained decision boundaries and adaptability to complex data. However, the polynomial kernel remains a competitive alternative, particularly for less intricate datasets or where computational simplicity is a priority. These findings emphasize the importance of kernel selection in optimizing model performance for specific tasks and data characteristics.

3.8a. The One-vs-One (OvO) method is a widely used technique for generalizing binary classification algorithms to multi-class problems. This approach trains a binary classifier for every unique pair of classes in the dataset, resulting in a total of $\frac{k(k-1)}{2}$ classifiers for k classes. Each binary classifier is trained to distinguish between two specific classes while ignoring all others, allowing for focused and simplified decision boundaries.

During the training phase, the dataset is divided into subsets, each containing data points from only two classes. For a given pair of classes i and j, a binary classifier is trained with the data points of class i labeled as +1 and those of class j labeled as -1. This pairwise training process ensures that each classifier is optimized to handle the distinct differences between its two target classes, making it more effective for datasets with imbalanced or overlapping features.

In the prediction phase, a test sample is evaluated by all $\frac{k(k-1)}{2}$ classifiers. Each classifier casts a vote for one of its two associated classes based on its prediction. The final class label is determined by majority voting, where the class with the highest number of votes is selected. In the case of ties, additional strategies, such as selecting the class with the smallest index or leveraging confidence scores, can be employed to resolve ambiguities.

The OvO method offers several advantages. By training each classifier on only two classes, the computational complexity of each individual model is reduced, often leading to faster training times and simpler decision boundaries. Additionally, the method is naturally suited for imbalanced datasets, as it focuses on pairwise relationships rather than the entire class distribution. Furthermore, OvO is highly adaptable, as it can be implemented with any binary classification algorithm, making it versatile across various machine learning frameworks.

However, the OvO method has notable challenges. The number of classifiers grows quadratically with the number of classes, making it computationally expensive for datasets with many classes. During prediction, the need to query all pairwise classifiers increase computational overhead, potentially affecting real-time applications. Moreover, the majority voting mechanism may result in ties, necessitating additional tie-breaking strategies, which can introduce complexity or bias.

In summary, the One-vs-One method is a robust and effective strategy for multi-class classification. Its ability to simplify decision boundaries and handle imbalanced data makes it a compelling choice for many applications. Despite its computational demands, the method's flexibility and adaptability ensure its continued relevance in complex classification tasks.

3.8b. The Q3 procedure was performed using the One-vs-One (OvO) method with the polynomial kernel across $\mathrm{d} = 1, \ldots, 7$. Table 5 presents the mean train and test errors along with their respective standard deviations for each value of d.

Param (d)	Train Error (Mean ± STD)	Test Error (Mean ± STD)
1	0.0178 ± 0.0050	0.0591 ± 0.0068
2	0.0007 ± 0.0006	0.0351 ± 0.0040
3	0.0003 ± 0.0002	0.0306 ± 0.0039
4	0.0002 ± 0.0001	0.0313 ± 0.0037
5	0.0003 ± 0.0002	0.0324 ± 0.0038
6	0.0002 ± 0.0001	0.0331 ± 0.0036
7	0.0002 ± 0.0001	0.0339 ± 0.0036

Table 5. Mean Train and Test Errors for OvO with Polynomial Kernel.

3.8c. The One-versus-One (OvO) method was applied to the kernel perceptron using a polynomial kernel, with cross-validation performed over $\mathrm{d} = 1, \ldots, 7$. Table 6 summarize the mean and standard deviation (±) of the optimal polynomial degree (d^*), training errors, and test errors over 20 runs.

Metric	Mean ± STD
Optimal d^*	4.05 ± 0.97
Training Error	0.0003 ± 0.0003
Test Error	0.0338 ± 0.0032

Table 6. Cross-Validation Results for OvO with Polynomial Kernel.

3.8d. The performance of the One-versus-One (OvO) and One-versus-Rest (OvR) methods with polynomial kernels reveals nuanced differences across their general evaluations and 5-fold cross-validation results. The results demonstrate that OvR achieves superior generalization overall. In the broader evaluations (Tables 1 and 5), OvR achieved a minimum test error of 0.0258 ± 0.0041at $d = 5$, compared to OvO's 0.0306 ± 0.0039 at $d = 3$. Similarly, in cross-validation (Tables 2 and 6), OvR reported a test error of 0.0275 ± 0.0052, outperforming OvO's 0.0338 ± 0.0032. This trend suggests that OvR's global classification approach, which trains one classifier per class, captures inter-class relationships more effectively, allowing it to generalize better to unseen data.

In terms of training error, OvO consistently achieved lower values, reflecting its ability to specialize in resolving pairwise distinctions. From Tables 1 and 5, OvO achieved a minimum training error of 0.0002 ± 0.0001 at $d = 4$, while OvR's lowest training error was 0.0001 ± 0.0001 at $d = 5$. In cross-validation, OvO maintained a lower training error of 0.0003 ± 0.0003 compared to OvR's 0.0008 ± 0.0028. However, the reduced test performance for OvO indicates that this specialization can lead to overfitting in its pairwise classifiers, which do not capture global inter-class relationships as effectively as OvR.

The optimal polynomial degree (d^*) for OvO and OvR highlights differences in model complexity and stability. OvO achieved a lower mean d^* (4.05 ± 0.97) compared to OvR (4.60 ± 1.02), indicating that OvO requires less model complexity to achieve its best performance. Additionally, the lower standard deviation in d^* for OvO suggests greater stability in hyperparameter selection, likely due to its localized pairwise classification approach. In contrast, OvR's higher d^* reflects the need for more complex decision boundaries to manage all classes simultaneously.

In summary, OvR consistently generalizes better, with lower test errors across both broad evaluations and cross-validation. This makes it the preferred choice for tasks prioritizing generalization and computational efficiency.
