University College London

Department of Computer Science

COMP0078: Supervised Learning

Coursework 2

Author:
Mr. Victor Fizesan, Mr. Frederico Wieser

Student Number:
20009434, 18018699

Last Updated:
January 2, 2025

Contents

1 Rademacher Complexity of finite Spaces

1.1 Let $\bar{X} = \max_i X_i$. Show that for any $\lambda > 0$, $\mathbb{E}[\bar{X}] \leq \frac{1}{\lambda} \log \mathbb{E}[e^{\lambda \bar{X}}]$

Jensen's inequality states that for any convex function $f(\cdot)$, we have:

$$f\left(\sum_{i=1}^{n} p_i x_i\right) \leq \sum_{i=1}^{n} p_i f(x_i),$$

where $p_i \geq 0$ and $\sum_{i=1}^{n} p_i = 1$. For random variables, this can be restated as:

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

Now, let $f(x) = \exp(\lambda x)$, where $\lambda > 0$. Since $\exp(\lambda x)$ is a convex function, we can apply Jensen's inequality. Replacing X with the random variable $\bar{X} = \max_i X_i$, we obtain:

$$\exp\left(\lambda \mathbb{E}[\bar{X}]\right) \leq \mathbb{E}\left[\exp(\lambda \bar{X})\right].$$

Taking the natural logarithm on both sides:

$$\lambda \mathbb{E}[\bar{X}] \leq \log \mathbb{E}\left[\exp(\lambda \bar{X})\right].$$

Dividing through by $\lambda > 0$, we arrive at:

$$\mathbb{E}[\bar{X}] \leq \frac{1}{\lambda} \log \mathbb{E}\left[\exp(\lambda \bar{X})\right].$$

This completes the proof.

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

Firstly, observe that:

$$\frac{1}{\lambda} \log \mathbb{E}\left[\exp\left(\lambda \bar{X}\right)\right] = \frac{1}{\lambda} \log \mathbb{E}\left[\exp\left(\lambda \max_i X_i\right)\right].$$

Using the inequality $\max_i \exp(\lambda X_i) \leq \sum_{i=1}^{m} \exp(\lambda X_i)$, it follows that:

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

Since the expectation of a summation is equal to the summation of expectations, we can write:

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

Substituting this into the inequality, we get:

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

Using Hoeffding's lemma, which states that for any centered random variable X_i with values in $[a,b]$,

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

we can bound the sum:

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

Taking the logarithm of both sides:

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

Combining the above results, we conclude that:

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

This completes the proof.

1.3 Choosing λ appropriately

From the result of the previous question, we have:

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

Applying Jensen's inequality directly to $\bar{X}$, we know that:

$$\mathbb{E}\left[\bar{X}\right] = \frac{1}{\lambda}\log\left(\exp\left(\mathbb{E}\left[\lambda\bar{X}\right]\right)\right) \leq \frac{1}{\lambda}\log \mathbb{E}\left[\exp\left(\lambda \bar{X}\right)\right].$$

Combining the two inequalities, we get:

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

To optimize this bound, we minimize the function:

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

Differentiating $\varphi(\lambda)$ with respect to λ, we find:

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

Setting $\varphi'(\lambda) = 0$, we solve for λ:

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

Substituting this value of λ back into the expression for $\varphi(\lambda)$, we have:

$$\varphi(\lambda) = \frac{b-a}{2\sqrt{2\log m}}\log m + \frac{(b-a)\sqrt{2\log m}}{4}.$$

To simplify further, we rewrite the first term as:

$$\frac{b-a}{2\sqrt{2\log m}}\log m = \frac{(b-a)}{2\sqrt{2}}\sqrt{\log m},$$

and combining this with the second term:

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

Thus, the bound on the expected maximum, when an appropriate λ is chosen, is:

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

This concludes the derivation.

1.4 Bound on $\mathcal{R}(S)$

Using Jensen's inequality for the convex function $\exp(\cdot)$, we have:

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

Interchanging the expectation and the maximum:

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

Using the independence of the Rademacher random variables σ_j, the expectation can be written as:

$$\mathbb{E}_\sigma\left[\exp\left(\frac{\lambda}{n}\sum_{j=1}^{n}\sigma_j x_j\right)\right] = \prod_{j=1}^{n}\mathbb{E}_\sigma\left[\exp\left(\frac{\lambda}{n}\sigma_j x_j\right)\right].$$

It is known that for $\sigma_j \sim \{-1, 1\}$, we have:

$$\mathbb{E}_\sigma\left[\exp\left(\frac{\lambda}{n}\sigma_j x_j\right)\right] = \frac{1}{2}\exp\left(\frac{\lambda}{n}\cdot(+1)\,x_j\right) + \frac{1}{2}\exp\left(\frac{\lambda}{n}\cdot(-1)\,x_j\right) = \cosh\left(\frac{\lambda x_j}{n}\right).$$

Using the inequality $\cosh(t) \leq \exp(t^2/2)$, we obtain:

$$\prod_{j=1}^{n}\cosh\left(\frac{\lambda x_j}{n}\right) \leq \prod_{j=1}^{n}\exp\left(\frac{\lambda^2 x_j^2}{2n^2}\right).$$

Simplifying the product:

$$\prod_{j=1}^{n} \exp\left(\frac{\lambda^2 x_j^2}{2n^2}\right) = \exp\left(\frac{\lambda^2}{2n^2}\|\boldsymbol{x}\|_2^2\right).$$

Thus:

$$\exp(\lambda\mathcal{R}(S)) \leq m \exp\left(\frac{\lambda^2}{2n^2}\max_{x\in S}\|x\|_2^2\right).$$

Taking the natural logarithm:

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

Rewriting this as:

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

To minimize the bound, set:

$$\lambda = \frac{n\sqrt{2\log(m)}}{\max_{x\in S}\|x\|_2}.$$

Substituting λ into the terms:

$$\frac{\log(m)}{\lambda} = \frac{\max_{x\in S}\|x\|_2 \cdot \sqrt{\log(m)}}{n}, \quad \frac{\lambda}{2n^2}\max_{x\in S}\|x\|_2^2 = \frac{\max_{x\in S}\|x\|_2 \cdot \sqrt{2\log(m)}}{2n}.$$

Adding these terms:

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

Simplifying we get:

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

as required.

1.5 Bounding $\mathcal{R}_S(H)$

Since H is finite, we can replace the supremum with a maximum:

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

Using Jensen's inequality:

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

Interchanging the expectation and the maximum, and then bounding the maximum with the sum across all elements of H:

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

Applying Hoeffding's Lemma, we have:

$$\mathbb{E}_\sigma\left[\exp\left(\lambda\frac{1}{n}\sum_{i=1}^{n}\sigma_i f(x_i)\right)\right] \leq \exp\left(\frac{\lambda^2 \max_{f\in H}\|f(x_i)\|_2^2}{2n^2}\right).$$

Substituting this back:

$$\exp\left(\lambda\mathcal{R}_S(H)\right) \leq |H| \cdot \exp\left(\frac{\lambda^2 \max_{f\in H}\|f(x_i)\|_2^2}{2n^2}\right).$$

Taking the logarithm:

$$\lambda\mathcal{R}_S(H) \leq \log|H| + \frac{\lambda^2 \max_{f\in H}\|f(x_i)\|_2^2}{2n^2}.$$

To optimize the bound, set:

$$\lambda = \frac{n\sqrt{2\log|H|}}{\max_{f\in H}\|f(x_i)\|_2}.$$

Substituting λ gives:

$$\mathcal{R}_S(H) \leq \max_{f\in H}\|f(x_i)\|_2 \frac{\sqrt{2\log|H|}}{n}.$$

Thus, the empirical Rademacher complexity depends logarithmically on $|H|$, as required.

2 Bayes Decision Rule & Surrogate Approaches

2.1 $R(c) = \mathbb{E}_{(x,y)\sim\rho}[1_{c(x)\neq y}]$

The misclassification error of a classification rule is defined as the probability $c(x) \neq y$ over the joint. This probability can be expressed as an integral of the indicator function $1_{c(x)\neq y}$ under ρ, since the indicator function assigns a value of 1 to misclassified pairs (x, y) and 0 otherwise. Thus, the misclassification error is equivalently written as

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

The expected risk of a classification rule is defined as

$$\mathbb{E}_{(x,y)\sim\rho}[\ell(c(x), y)] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(c(x), y)\, d\rho(x, y),$$

where ℓ is a general loss function. In the specific case of the 0-1 loss function, defined as

$$\ell(c(x), y) = \mathbf{1}_{c(x)\neq y},$$

the expected risk reduces to

$$\mathbb{E}_{(x,y)\sim\rho}[\mathbf{1}_{c(x)\neq y}] = \int_{\mathcal{X}\times\mathcal{Y}} \mathbf{1}_{c(x)\neq y}\, d\rho(x, y).$$

Comparing this expression with the integral form of the misclassification error, it is evident that

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

Thus, the misclassification error is equivalent to the expected risk under the 0-1 loss, both being expressed as

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

This equivalence demonstrates that the misclassification error can be formally interpreted as the expected risk for the 0-1 loss function.

2.2 Minimizer f^* of $E(f)$

We aim to calculate the closed-form minimiser f^* of $E(f)$ for different loss functions. The expected risk is defined as:

$$E(f) = \int_{X\times Y} \ell(f(x), y)\, d\rho(x, y) = \int_X \left[\int_Y \ell(f(x), y)\, d\rho(y|x)\right] d\rho_X(x),$$

where $\rho(y|x)$ is the conditional distribution of Y given $X = x$, and $\ell(f(x), y)$ is the loss. This reduces to minimizing the inner integral pointwise for fixed x, with:

$$f^*(x) =_f \int_Y \ell(f(x), y)\, d\rho(y|x).$$

Let $p = \rho(y = +1|x)$ and $1 - p = \rho(y = -1|x)$.

2.2.1 Squared Loss

The squared loss is defined as:

$$\ell(f(x), y) = (f(x) - y)^2.$$

The conditional expectation of the squared loss is:

$$\int_Y \ell(f(x), y)\, d\rho(y|x) = p(f(x) - 1)^2 + (1 - p)(f(x) + 1)^2.$$

Expanding and simplifying:

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

Minimizing w.r.t. $f(x)$, we find:

$$f^*(x) = 2p - 1.$$

2.2.2 Exponential Loss

The exponential loss is defined as:

$$\ell(f(x), y) = \exp(-yf(x)).$$

The conditional expectation of the exponential loss is:

$$\int_Y \ell(f(x), y)\, d\rho(y|x) = p\exp(-f(x)) + (1 - p)\exp(f(x)).$$

Minimizing w.r.t. $f(x)$, we solve:

$$p\exp(-f(x)) = (1 - p)\exp(f(x)).$$

Taking the logarithm:

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

2.2.3 Logistic Loss

The logistic loss is defined as:

$$\ell(f(x), y) = \ln(1 + \exp(-yf(x))).$$

The conditional expectation of the logistic loss is:

$$\int_Y \ell(f(x), y)\, d\rho(y|x) = p\ln(1 + \exp(-f(x))) + (1 - p)\ln(1 + \exp(f(x))).$$

Minimizing w.r.t. $f(x)$, we solve:

$$\frac{p}{1 + \exp(f(x))} = \frac{1 - p}{1 + \exp(-f(x))}.$$

This simplifies to:

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

2.2.4 Hinge Loss

The hinge loss is defined as:

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

The conditional expectation of the hinge loss is:

$$\int_Y \ell(f(x), y)\, d\rho(y|x) = p \max(0, 1 - f(x)) + (1 - p)\max(0, 1 + f(x)).$$

Analysing case-by-case:

- If $f(x) > 1$, the hinge loss is minimized at $f^*(x) = 1$.
- If $f(x) < -1$, the hinge loss is minimized at $f^*(x) = -1$.
- If $-1 \leq f(x) \leq 1$, the loss is linear, and the minimizer depends on p.

Therefore:

- If $p > 1/2$, the minimum is at $f^*(x) = 1$.
- If $p < 1/2$, the minimum is at $f^*(x) = -1$.
- If $p = 1/2$, any $f(x) \in [-1, 1]$ is optimal.

Hence:

$$f^*(x) = \begin{cases} +1, & \text{if } p > \frac{1}{2}, \\ -1, & \text{if } p < \frac{1}{2}, \\ \text{either } [-1, 1], & \text{if } p = \frac{1}{2}. \end{cases}$$

Or equivalently:

$$f^*(x) = \text{sign}(2p - 1), \text{ with a tie if } p = 1/2.$$

2.3 Bayes Decision Rule Derivation

To minimize $R(c)$, note that $c(x)$ appears inside the integral only through the indicator $\mathbf{1}_{c(x)\neq y}$. Therefore, for each x, we minimize the inner integral:

$$\int_{y\in\{-1,+1\}} \mathbf{1}_{c(x)\neq y}\, \rho(y|x)\, dy.$$

For binary classification ($Y \in \{-1, +1\}$), this corresponds to:

- $\rho(y = -1|x)$, the probability of misclassification if $c(x) = +1$
- $\rho(y = +1|x)$, the probability of misclassification if $c(x) = -1$

The optimal decision for x is thus to assign the label with the highest posterior probability:

$$c^*(x) = \arg \max_{y\in\{-1,+1\}} \rho(y|x).$$

This can be expressed explicitly as:

$$c^*(x) = \begin{cases} +1, & \text{if } \rho(y = +1|x) \geq \rho(y = -1|x), \\ -1, & \text{otherwise.} \end{cases}$$

Equivalently, using $\rho(y = -1|x) = 1 - \rho(y = +1|x)$, we have:

$$c^*(x) = \begin{cases} +1, & \text{if } \rho(y = +1|x) \geq 0.5, \\ -1, & \text{otherwise.} \end{cases}$$

This rule minimizes the probability of misclassification.

2.4 Find d

2.4.1 Squared Loss

For $\ell(f(x), y) = (f(x) - y)^2$, the minimizer is:

$$f_*(x) = 2p - 1.$$

This $f_*(x)$ is Fisher consistent because:

- If $p > \frac{1}{2}$, then $f_*(x) > 0$, and $d(f_*(x)) = \text{sign}(f_*(x)) = +1 = c_*(x)$.
- If $p < \frac{1}{2}$, then $f_*(x) < 0$, and $d(f_*(x)) = \text{sign}(f_*(x)) = -1 = c_*(x)$.
- If $p = \frac{1}{2}$, then $f_*(x) = 0$, and $d(f_*(x))$ can assign either label, consistent with $c_*(x)$.

Thus, the mapping $d(f) = \text{sign}(f)$ ensures Fisher consistency for squared loss.

2.4.2 Exponential Loss

For $\ell(f(x), y) = \exp(-yf(x))$, the minimizer is:

$$f_*(x) = \frac{1}{2} \ln\left(\frac{p}{1-p}\right).$$

This $f_*(x)$ is Fisher consistent because:

- If $p > \frac{1}{2}$, then $\frac{p}{1-p} > 1$, so $\ln(p/(1-p)) > 0$, and $f_*(x) > 0$. Hence, $d(f_*(x)) = \text{sign}(f_*(x)) = +1 = c_*(x)$.
- If $p < \frac{1}{2}$, then $\frac{p}{1-p} < 1$, so $\ln(p/(1-p)) < 0$, and $f_*(x) < 0$. Hence, $d(f_*(x)) = \text{sign}(f_*(x)) = -1 = c_*(x)$.
- If $p = \frac{1}{2}$, then $\ln(p/(1-p)) = 0$, and $f_*(x) = 0$. The mapping $d(f_*(x))$ can assign either label, consistent with $c_*(x)$.

Thus, $d(f) = \text{sign}(f)$ ensures Fisher consistency for exponential loss.

2.4.3 Logistic Loss

For $\ell(f(x), y) = \ln(1 + \exp(-yf(x)))$, the minimizer is:

$$f_*(x) = \ln\left(\frac{p}{1-p}\right).$$

This $f_*(x)$ is Fisher consistent because:

- If $p > \frac{1}{2}$, then $\frac{p}{1-p} > 1$, so $\ln(p/(1-p)) > 0$, and $f_*(x) > 0$. Hence, $d(f_*(x)) = \text{sign}(f_*(x)) = +1 = c_*(x)$.
- If $p < \frac{1}{2}$, then $\frac{p}{1-p} < 1$, so $\ln(p/(1-p)) < 0$, and $f_*(x) < 0$. Hence, $d(f_*(x)) = \text{sign}(f_*(x)) = -1 = c_*(x)$.
- If $p = \frac{1}{2}$, then $\ln(p/(1-p)) = 0$, and $f_*(x) = 0$. The mapping $d(f_*(x))$ can assign either label, consistent with $c_*(x)$.

Thus, $d(f) = \text{sign}(f)$ ensures Fisher consistency for logistic loss.

2.4.4 Hinge Loss

For $\ell(f(x), y) = \max(0, 1 - yf(x))$, the minimizer is:

$$f_*(x) = \begin{cases} +1, & \text{if } p > \frac{1}{2}, \\ -1, & \text{if } p < \frac{1}{2}, \\ \text{any value in } [-1, 1], & \text{if } p = \frac{1}{2}. \end{cases}$$

This $f_*(x)$ is Fisher consistent because:

- If $p > \frac{1}{2}$, then $f_*(x) = +1$, and $d(f_*(x)) = \text{sign}(f_*(x)) = +1 = c_*(x)$.
- If $p < \frac{1}{2}$, then $f_*(x) = -1$, and $d(f_*(x)) = \text{sign}(f_*(x)) = -1 = c_*(x)$.
- If $p = \frac{1}{2}$, then $f_*(x) \in [-1, 1]$. The mapping $d(f_*(x))$ can assign either label, consistent with $c_*(x)$.

Thus, $d(f) = \text{sign}(f)$ ensures Fisher consistency for hinge loss.

2.4.5 Conclusion

For all surrogate frameworks, the mapping $d(f) = \text{sign}(f)$ ensures that:

$$R(c_*(x)) = R(d(f_*(x))),$$

proving Fisher consistency.

2.5 Prove Comparison Inequality

2.5.1 Identity for $|R(\text{sign}(f)) - R(\text{sign}(f_*))|$

Definition of the set $\mathcal{X}_f$. We define

$$\mathcal{X}_f \;=\; \Big\{x \in \mathcal{X} :\; \text{sign}[f(x)] \neq \text{sign}[f_*(x)]\Big\},$$

In other words, the set of all points x for which $f(x)$ and $f_*(x)$ disagree in sign.

To prove the relevant identity, recall the 0–1 risk of any classifier c is

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

For clarity let $\mathbf{1}\{c(x) \neq y\} := \mathbf{1}_{c(x)\neq y}$. Hence

$$R(\text{sign}(f)) \;=\; \int \mathbf{1}\{\text{sign}[f(x)] \neq y\}\, d\rho(x,y), \quad R(\text{sign}(f_*)) \;=\; \int \mathbf{1}\{\text{sign}[f_*(x)] \neq y\}\, d\rho(x,y).$$

Subtracting,

$$R(\text{sign}(f)) - R(\text{sign}(f_*)) \;=\; \int \Big[\mathbf{1}\{\text{sign}[f(x)] \neq y\} \;-\; \mathbf{1}\{\text{sign}[f_*(x)] \neq y\}\Big] d\rho(x,y).$$

If $\text{sign}[f(x)] = \text{sign}[f_*(x)]$, these two indicators coincide (either both 0 or both 1), so their difference is 0. Therefore, the integrand is non-zero only on $\mathcal{X}_f = \{x : \text{sign}[f(x)] \neq \text{sign}[f_*(x)]\}$. This shows

$$R(\text{sign}(f)) - R(\text{sign}(f_*)) \;=\; \int_{x\in\mathcal{X}_f} \Big[\mathbf{1}\{\text{sign}[f(x)] \neq y\} - \mathbf{1}\{\text{sign}[f_*(x)] \neq y\}\Big] d\rho(x,y).$$

At each $x \in \mathcal{X}_f$, by definition $\text{sign}[f(x)] \neq \text{sign}[f_*(x)]$. A case analysis on the labels $y \in \{-1,+1\}$ reveals that the above difference $\mathbf{1}\{\text{sign}[f(x)] \neq y\} - \mathbf{1}\{\text{sign}[f_*(x)] \neq y\}$ telescopes to either $+(2\eta(x)-1)$ or $-(2\eta(x)-1)$, where $\eta(x) = \rho(Y = +1 \mid X = x)$. Recall $f_*(x) = 2\,\eta(x) - 1$. In absolute value, this is precisely $|f_*(x)|$.

Hence, upon taking absolute values,

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

as required.

2.5.2 Bounding $\int_{\mathcal{X}_f} |f_*(x)|\, d\rho_{\mathcal{X}}(x)$ in terms of $\sqrt{\mathbb{E}[(f - f_*)^2]}$

By definition,

$$\mathcal{X}_f \;=\; \Big\{x : \text{sign}[f(x)] \neq \text{sign}[f_*(x)]\Big\}.$$

so exactly two possibilities arise, on this set:

- $f_*(x) > 0$ but $f(x) \le 0$. Then

$$|f_*(x) - f(x)| = (f_*(x)) - (f(x)) \;\ge\; f_*(x) = |f_*(x)|.$$

- $f_*(x) < 0$ but $f(x) \ge 0$. Then

$$|f_*(x) - f(x)| \;=\; -(f_*(x)) + f(x) \;\ge\; |f_*(x)|.$$

Hence for all $x \in \mathcal{X}_f$,

$$|f_*(x)| \;\le\; |f_*(x) - f(x)|.$$

Integrating,

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

Note that $\mathcal{X}_f \subseteq \mathcal{X}$, so

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

The Cauchy–Schwarz inequality states that, for any random variable Z,

$$\int |Z(x)|\, d\rho_{\mathcal{X}}(x) \;\le\; \sqrt{\int Z(x)^2\, d\rho_{\mathcal{X}}(x)}.$$

We apply this with $Z(x) = f_*(x) - f(x)$. Thus

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

Combining both steps completes the proof.

2.5.3 Exact Relation: $\mathcal{E}(f) - \mathcal{E}(f_*) = \mathbb{E}[(f - f_*)^2]$

We have

$$\mathcal{E}(f) \;=\; \mathbb{E}\Big[(f(X) - Y)^2\Big], \quad \mathcal{E}(f_*) \;=\; \mathbb{E}\Big[(f_*(X) - Y)^2\Big].$$

Thus

$$\mathcal{E}(f) - \mathcal{E}(f_*) \;=\; \mathbb{E}\Big[(f(X) - Y)^2 - (f_*(X) - Y)^2\Big].$$

Write $(a-b)^2 = a^2 - 2ab + b^2$, giving

$$(f(X) - Y)^2 - (f_*(X) - Y)^2 \;=\; \Big[f(X)^2 - f_*(X)^2\Big] \;-\; 2\Big[f(X)Y - f_*(X)Y\Big].$$

Taking expectation and using $f_*(x) = \mathbb{E}[Y \mid X = x]$ implies

$$\mathbb{E}[\,f(X)\,Y\,] \;=\; \mathbb{E}_X\Big[f(X)\,\mathbb{E}[Y \mid X]\Big] \;=\; \mathbb{E}\Big[f(X)\,f_*(X)\Big].$$

Furthermore we can deduce that:

$$\mathbb{E}[f_*(X)Y] = \mathbb{E}_X[f_*(X)\mathbb{E}[Y \mid X]] = \mathbb{E}[f_*(X)f_*(X)] = \mathbb{E}[f_*(X)^2]$$

Hence

$$\mathcal{E}(f) - \mathcal{E}(f_*) \;=\; \mathbb{E}\Big[f(X)^2 - 2f_*(X)f_*(X) + f_*(X)^2\Big] \;=\; \mathbb{E}\Big[(f(X) - f_*(X))^2\Big]$$

as required.

Overall Conclusion (Comparison Inequality).

$$|R(\operatorname{sign}(f)) - R(\operatorname{sign}(f_*))| \;=\; \int_{\mathcal{X}_f} |f_*(x)|\, d\rho_{\mathcal{X}}(x) \;\leq\; \sqrt{\mathbb{E}\Big[(f(X) - f_*(X))^2\Big]} \;=\; \sqrt{\mathcal{E}(f) - \mathcal{E}(f_*)}.$$

Hence

$$0 \;\leq\; R(\operatorname{sign}(f)) - R(\operatorname{sign}(f_*)) \;\leq\; \sqrt{\mathcal{E}(f) - \mathcal{E}(f_*)}.$$

3 Kernel perceptron

3.1 One-vs-Rest

In a dataset with k classes, a One-versus-Rest kernel perceptron creates k binary classifiers. Each classifier distinguishes one class c_i $(i = 1,\ldots,k)$ from all others, labeling c_i as $+1$ and the rest as -1. For a new input x, each binary classifier computes a real-valued confidence score through its kernel expansion:

$$\text{confidence}_i(x) = \sum_{j=1}^{m} \alpha_j K(x_j, x)$$

where α_j are the learned weights and K is the kernel function. The final prediction is the class whose classifier produced the highest confidence score.

3.2 Implementation of One-vs-Rest

Our implementation transforms the binary kernel perceptron into a multi-class classifier using the One-vs-Rest approach. For a k-class problem, we maintain k separate weight vectors, each distinguishing one class from all others. These weight vectors are implicitly represented through a K×m matrix α of coefficients, where m is the number of training examples.

Algorithm 1 One-vs-Rest Kernel Perceptron

Require: Training data $\{(x_1, y_1), ..., (x_m, y_m)\} \in (\mathbb{R}^n, \{1, ..., K\})^m$
Ensure: Trained model parameters α
  Initialize $\alpha = 0^{K \times m}$ ▷ K × m matrix of zeros
  for epoch = 1 to 5 **do**
    for t = 1 to m **do**
      for k = 1 to K **do**
        $y_t^k \leftarrow 2 \cdot \mathbb{1}(y_t = k) - 1$ ▷ +1 for class k, -1 otherwise
        Compute score: $C_k(x_t) = \sum_{i=1}^{m} \alpha_i^{(k)} K(x_i, x_t)$
        if $y_t^k C_k(x_t) \leq 0$ **then**
          $\alpha_t^{(k)} \leftarrow \alpha_t^{(k)} + y_t^k$ ▷ Update if misclassified
        end if
      end for
    end for
  end for
  return α

The kernel expansion $\sum_{i=0}^{m} \alpha_i K(x_i, \cdot)$ is represented implicitly through the α coefficients and stored training points. During prediction, we evaluate this expansion by computing the kernel matrix K between test and training points, then multiplying with α to obtain confidence scores for each class. When misclassifications occur during training, the expansion is updated by modifying the relevant α coefficients: incrementing for positive examples and decrementing for negative ones.

Five epochs were chosen based on empirical analysis showing convergence of test errors to 2-3% after 3-4 epochs, with diminishing returns beyond this point. This provides a good balance

between model performance and computational efficiency.

3.3 Basic Results with Polynomial Kernel

We evaluated the polynomial kernel perceptron with kernel $K(x_i, x_t) = (x_i \cdot x_t)^d$ over 20 random 80-20 train-test splits, varying the degree d from 1 to 7. The results demonstrate a clear bias-variance tradeoff: training error decreases monotonically as model complexity increases, while test error follows a convex pattern.

Figure 1 shows the test error drops sharply from 9.36% (d=1) to 4.47% (d=2), then gradually decreases to a minimum of 3.16% (d=6) before increasing again. The optimal range appears to be d=4-6, where test errors stabilize around 3.2-3.3% with consistent standard deviations (±0.38-0.40%), suggesting these degrees provide sufficient complexity while avoiding overfitting.

d	Mean Train Error Rates (%)	Mean Test Error Rates (%)
1	7.7097±0.2457	9.3548±1.6674
2	1.5878±0.1406	4.4677±0.6874
3	0.6722±0.0988	3.7016±0.4322
4	0.3482±0.0757	3.3360±0.3834
5	0.3038±0.0839	3.3280±0.4069
6	0.2245±0.0690	3.1586±0.4048
7	0.1795±0.0617	3.7285±1.5295

Figure 1: Polynomial kernel perceptron train and test error rates (%) for different polynomial degrees, d.

3.4 Cross-Validation for Polynomial Kernel

Using 5-fold cross-validation over 20 runs, the optimal polynomial degree d^* consistently falls between 5 and 6, with low variance across runs. The model achieves strong training performance while maintaining good generalization on test data (mean error $\sim$ 3.4%). These cross-validation results reinforce our earlier findings that moderate-degree polynomials provide optimal performance for the digit classification task.

Run	d*	Train Error (%)	Test Error (%)
1	6	0.2420	3.6559
2	7	0.2286	3.2258
3	7	0.2420	3.0645
4	6	0.2017	3.1720
5	4	0.4706	3.3871
6	6	0.0941	3.0645
7	6	0.3227	3.5484
8	5	0.4168	3.6559
9	6	0.1076	3.1183
10	4	0.3630	3.4409
11	6	0.2958	3.5484
12	5	0.2017	3.6022
13	5	0.1882	3.9785
14	7	0.1748	3.2258
15	5	0.2689	3.5484
16	5	0.1748	2.7957
17	5	0.1479	2.5806
18	6	0.1210	3.8710
19	5	0.1613	3.4409
20	5	0.2958	3.1183
----	----	----------	----------
μ	5.55	0.2360	3.3522
σ	0.86	0.0987	0.3380

Figure 2: Cross-validation results over 20 runs for One-Vs-Rest.

3.5 Confusion Matrix

The confusion matrix reveals error patterns that align with visual similarities between digits. The most frequent confusions occur between structurally similar pairs like ’3’-’5’, ’5’-’8’, and ’4’-’9’, with some pairs showing notable variability (>1% standard deviation) across different data splits.

Most cells show very low error rates (<0.5%), indicating the model is generally robust. The asymmetric nature of some confusions is notable – for instance, ’3’ is more often mistaken for ’5’ (2.3%) than ’5’ for ’3’ (1.6%), suggesting certain digit representations are more distinctive than others.

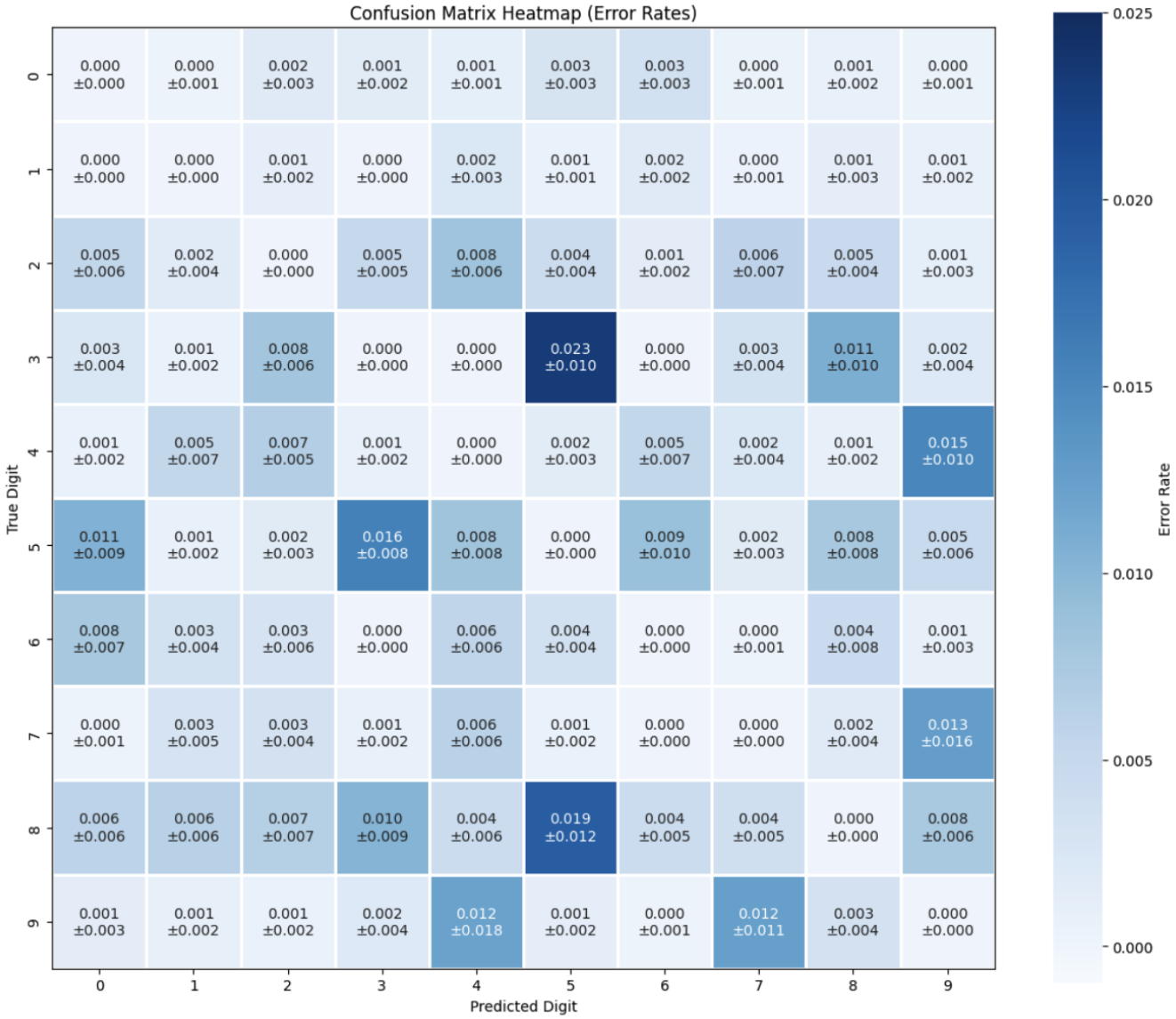

Figure 3: Confusion matrix of error rates (%) between true and predicted digits.

3.6 Hard-to-Predict Samples

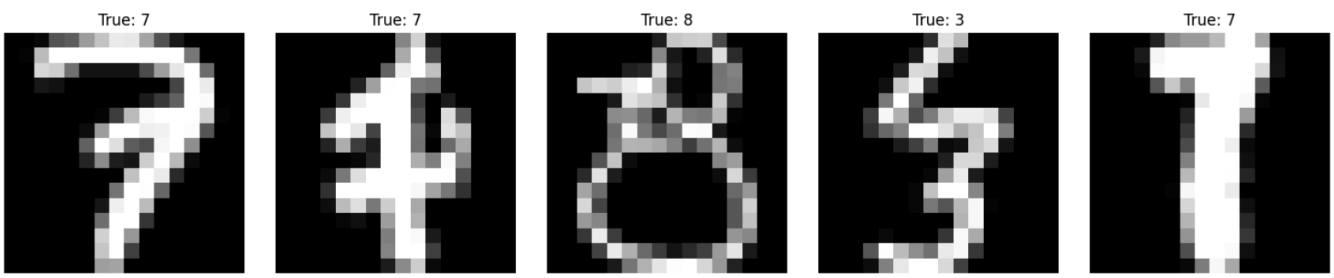

Figure 4: A size-5 subset of the hardest-to-predict images

The misclassified digits reveal interesting patterns about the kernel perceptron's limitations, with distinctly high error rates (97-100%) across multiple train-test splits. The misclassifications make sense given the visual ambiguity in these examples. Two '7's with unusual styling are confused for '9' and '4', while a slightly distorted '8' is seen as '5'. A poorly formed '3' is frequently classified as '8' and '5', and a straight '7' is mistaken for '8' and '1'. These errors occur when writing styles deviate significantly from typical examples, key distinguishing features are ambiguous, or pixelation obscures important structural details.

The consistency in misclassification across multiple runs suggests these examples lie in particularly challenging regions of the feature space, where the decision boundaries learned by the kernel perceptron remain consistently incorrect.

3.7 Gaussian Kernel Analysis

3.7.1 Selecting Gaussian Kernel Parameters

For the Gaussian kernel $K(x_i, x_t) = e^{-c\|x_i - x_t\|}$, we conducted preliminary experiments to determine an appropriate search space for the width parameter c. We observed that: small values ($c < 10^{-3}$) led to under-fitting, with the kernel approaching a constant function; and large values ($c > 5$) caused over-fitting, making the kernel too localized.

Based on these observations, we selected a refined search space:

$$S = \{0.001, 0.005, 0.010, 0.015, 0.020, 0.050, 0.100\}$$

This range provides good coverage of the transition between under-fitting and over-fitting regimes while maintaining computational efficiency.

3.7.2 Training and Testing with Gaussian Kernel

The Gaussian kernel models are trained and tested for $c \in S$. The results are averaged over 20 runs.

c	Mean Train Error Rates (%)	Mean Test Error Rates (%)
0.001	7.3501±0.3125	8.1667±1.5464
0.005	1.1831±0.1214	4.3683±1.1453
0.01	0.3872±0.0928	3.2796±0.3929
0.015	0.2212±0.0759	3.2231±0.4592
0.02	0.1566±0.0526	3.1210±0.3636
0.05	0.0497±0.0427	3.9409±0.3244
0.1	0.0282±0.0199	5.1371±0.3931

Figure 5: Mean train and test errors sets for each value of c in S

3.7.3 Cross-Validation with Gaussian Kernel

Cross-validation is performed over $c \in S$ to select the optimal c^*. The retrained model is evaluated on train and test data, reporting means and standard deviations.

Run	c*	Train Error (%)	Test Error (%)
1	0.02	0.1748	3.1183
2	0.015	0.2286	3.1183
3	0.015	0.3227	2.9570
4	0.02	0.0941	2.6882
5	0.015	0.2017	3.0108
6	0.015	0.1882	3.0108
7	0.015	0.2286	3.2258
8	0.02	0.1613	3.9785
9	0.02	0.2017	3.7634
10	0.02	0.2823	4.7312
11	0.02	0.1479	3.1720
12	0.02	0.1613	3.2258
13	0.01	0.2151	3.6559
14	0.015	0.3227	3.1720
15	0.02	0.1882	4.0323
16	0.02	0.2286	3.1183
17	0.015	0.1882	3.4946
18	0.015	0.2151	3.2796
19	0.015	0.3227	3.5484
20	0.02	0.1613	3.3333
----	----	----------	----------
μ	0.02	0.2118	3.3817
σ	0.00	0.0598	0.4558

Figure 6: Results over 20 runs of the Gaussian kernel method using c^*

3.7.4 Comparison of Kernels

The Gaussian kernel achieves consistently lower test error values across non-optimised values of c than the polynomial kernel does for different values of d, with minimal variance around the optimal parameter $c^* = 0.02 \pm 0.00$, demonstrating stronger generalization capabilities than the polynomial kernel where $d^* = 5.55 \pm 0.86$.

The polynomial kernel exhibits higher test error variance and a more sensitive dependency on its degree d, making optimal parameter selection more challenging than the Gaussian kernel, which seems to be more resilient to choosing non-optimal hyperparameters. Specifically, the Gaussian kernel achieves a mean training error of $0.2118 \pm 0.0598\%$ and a mean test error of $3.3817 \pm 0.4558\%$. By comparison, the polynomial kernel achieves competitive results with $d^* = 5.55 \pm 0.86$, yielding a mean training error of $0.2360 \pm 0.0987\%$ and a mean test error of $3.3522 \pm 0.3380\%$. This implies that the mean test error rate for polynomial kernels is lower, and therefore better, than that of the Gaussian kernel.

After reviewing these results we may say that the polynomial kernel is better in this case, when not including the errors which makes the performances fairly similar, but for other problems, we

may find it harder to search for these optimal values d, which can be a limitation of the method. On the other hand, the Gaussian kernel seems to be as performant but does not require as much fine-tuning.

The polynomial kernel is still a good choice for simpler tasks or when interpretability is important, but the Gaussian kernel is more flexible and stable.

3.8 One-vs-One Generalization

3.8.1 Explanation of One-vs-One Method

In a dataset with k classes, a One-versus-One approach creates $\binom{k}{2} = \frac{k(k-1)}{2}$ binary classifiers, each trained to distinguish between a pair of classes. For digit classification with 10 classes, this means training 45 binary classifiers, where each specializes in separating two specific digits.

Algorithm 2 One-vs-One Kernel Perceptron

Require: Training data $\{(x_1, y_1), ..., (x_m, y_m)\} \in (\mathbb{R}^n, \{1, ..., K\})^m$
Ensure: Trained model parameters α
  Initialize $\alpha = 0^{N \times m}$ ▷ N = K(K-1)/2 classifiers
  for epoch = 1 to 5 **do**
    for t = 1 to m **do**
      for j = 1 to N **do**
        Compute vote: $sign(\sum_{i=1}^{m} \alpha_i^{(j)} K(x_i, x_t))$ ▷ Each classifier votes
      end for
      $\hat{y}_t \leftarrow$ prediction from maximum voting
      for j = 1 to N **do**
        if classifier j involves y_t AND $\hat{y}_t \neq y_t$ **then**
          Update $\alpha^{(j)}$ based on whether y_t is positive/negative class
        end if
      end for
    end for
  end for
  return α

For prediction, each binary classifier votes for one class, and the final prediction is made by majority voting. This approach often provides better accuracy than One-vs-Rest as each classifier focuses on a simpler binary problem, though at the cost of training more classifiers.

3.8.2 Training and Testing with Polynomial Kernel

Using the One-vs-One method, the polynomial kernel models are trained and tested for $d = 1, \ldots, 7$. The results are averaged over multiple runs.

d	Mean Train Error Rates (%)	Mean Test Error Rates (%)
1	4.0844±0.7428	6.7957±0.7964
2	0.6897±0.2263	3.9812±0.5479
3	0.1889±0.0861	3.4194±0.3584
4	0.0908±0.0747	3.2876±0.4411
5	0.0652±0.0458	3.3011±0.3871
6	0.0518±0.0585	3.5161±0.4899
7	0.0356±0.0266	3.5753±0.3564

Figure 7: One-versus-One polynomial kernel perceptron train and test error rates (%) for different polynomial degrees, d.

3.8.3 Cross-Validation with Polynomial Kernel

Cross-validation is used to select the optimal d^* for the One-vs-One method. The retrained models are evaluated on train and test data.

Run	d*	Train Error (%)	Test Error (%)
1	4	0.0672	2.6882
2	4	1.5864	4.3548
3	5	0.0403	3.0108
4	4	0.0538	3.1720
5	4	0.1613	3.8172
6	5	0.0807	3.7097
7	6	0.0403	3.1183
8	3	0.0672	2.7419
9	4	0.1076	3.4409
10	6	0.0538	3.9247
11	6	0.0672	3.6559
12	3	0.3899	4.4086
13	4	0.0269	3.1183
14	3	0.4571	4.0323
15	4	0.0269	3.0645
16	4	0.0538	3.3871
17	4	0.0941	3.1720
18	3	0.2420	3.2796
19	5	0.1344	3.4946
20	3	0.1882	3.7097
----	----	----------	----------
μ	4.20	0.1970	3.4651
σ	0.98	0.3387	0.4713

Figure 8: One-versus-One cross-validation results over 20 runs

3.8.4 Comparison with One-vs-Rest

The One-vs-One and One-vs-Rest classification strategies with polynomial kernels exhibit broadly similar generalization capabilities, achieving test error minima around 3.2% to 3.3% at optimal degrees, $d^* \approx 4$–6. One-vs-One reaches a test error of $3.4651 \pm 0.4713\%$ at $d^* = 4.20 \pm 0.98$ with a training error of $0.1970 \pm 0.3387\%$. By contrast, One-vs-Rest achieves a slightly lower test error of $3.3522 \pm 0.3380\%$ at $d^* = 5.55 \pm 0.86$, though with a higher training error of $0.2360 \pm 0.0987\%$. These results highlight competitive performance between the two methods, with minor differences in their error profiles, although the One-vs-One method does have the slight advantage. They also seem to have a large difference in what is the optimal d^*, with the mean One-vs-One being much lower than the One-vs-Rest.

The choice between these methods often depends on practical considerations. One-vs-One requires $\frac{k(k-1)}{2}$ binary classifiers, which can be computationally expensive for large k, but it handles class imbalances more effectively. Conversely, One-vs-Rest only trains k classifiers, making it more computationally efficient, albeit more sensitive to skewed data distributions. Both methods are highly viable, with the final decision often guided by dataset characteristics and implementation constraints, rather than stark performance differences.