

# Homework #1

CSE 446/546: Machine Learning  
Prof. Hunter Schafer & Prof. Matt Golub  
Due: **Monday, January 29, 2024, 11:59pm**  
**A:** 60 points, **B:** 30 points

Please review all homework guidance posted on the website before submitting to Gradescope. Reminders:

- All code must be written in Python and all written work must be typeset (e.g.  $\text{\LaTeX}$ ).
- Make sure to read the “What to Submit” section following each question and include all items.
- Please provide succinct answers and supporting reasoning for each question. Similarly, when discussing experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. All explanations, tables, and figures for any particular part of a question must be grouped together.
- For every problem involving generating plots, please include the plots as part of your PDF submission.
- When submitting to Gradescope, please link each question from the homework in Gradescope to the location of its answer in your homework PDF. Failure to do so may result in deductions of up to 10% of the value of each question not properly linked. For instructions, see [https://www.gradescope.com/get\\_started#student-submission](https://www.gradescope.com/get_started#student-submission).

Not adhering to these reminders may result in point deductions.

**Important:** By turning in this assignment (and all that follow), you acknowledge that you have read and understood the collaboration policy with humans and AI assistants alike: <https://courses.cs.washington.edu/courses/cse446/24wi/assignments/>. Any questions about the policy should be raised at least 24 hours before the assignment is due. There are no warnings or second chances. If we suspect you have violated the collaboration policy, we will report it to the college of engineering who will complete an investigation. Not adhering to these reminders may result in point deductions.

## Changelog:

- **1/14/2023:** Fixed “What to Submit” for problem A2

## Short Answer and “True or False” Conceptual questions

A1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

- a. [2 points] In your own words, describe what bias and variance are. What is the bias-variance tradeoff?
- b. [2 points] What *typically* happens to bias and variance when the model complexity increases/decreases?
- c. [2 points] True or False: Suppose you’re given a fixed learning algorithm. If you collect more training data from the same distribution, the variance of your predictor increases.
- d. [2 points] Suppose that we are given train, validation, and test sets. Which of these sets should be used for hyperparameter tuning? Explain your choice and detail a procedure for hyperparameter tuning.
- e. [1 point] True or False: The training error of a function on the training set provides an overestimate of the true error of that function.

What to Submit:

- **Parts c, e:** True or False
- **Parts a-e:** Brief (2-3 sentence) explanation justifying your answer.

**Solution:**

**Part a** Bias is the difference between the expectation of the trained model on training data and the true model, variance is the fluctuation of the trained model.

**Part b** When model complexity increases, the bias decreases, but the variance increases. On the other hand, when model complexity decreases, the bias increases, but the variance decreases.

**Part c** False. With a fixed learning algorithm and more training data, the variance of predictor will decrease.

**Part d** The validation set should be used for hyper-parameter tuning. Test data cannot be used for any training session, and the training data is used for training the model parameter. For example, in the regularized regression, we use the validation set to tune the penalty coefficient.

**Part e** False. The training error usually underestimates the true error.

## Maximum Likelihood Estimation (MLE)

A2. You’re the Reign FC manager, and the team is five games into its 2021 season. The numbers of goals scored by the team in each game so far are given below:

$$[3, 7, 5, 0, 2].$$

Let’s call these scores  $x_1, \dots, x_5$ . Based on your (assumed iid) data, you’d like to build a model to understand how many goals the Reign are likely to score in their next game. You decide to model the number of goals scored per game using a *Poisson distribution*. Recall that the Poisson distribution with parameter  $\lambda$  assigns every non-negative integer  $x = 0, 1, 2, \dots$  a probability given by

$$\text{Poi}(x|\lambda) = e^{-\lambda} \frac{\lambda^x}{x!}.$$

- a. [5 points] Derive an expression for the maximum-likelihood estimate of the parameter  $\lambda$  governing the Poisson distribution in terms of goal counts for the first  $n$  games:  $x_1, \dots, x_n$ . (Hint: remember that the log of the likelihood has the same maximizer as the likelihood function itself.)

- b. [2 points] Give a numerical estimate of  $\lambda$  after the first five games. Given this  $\lambda$ , what is the probability that the Reign score exactly 4 goals in their next game?
- c. [2 points] Suppose the Reign actually score 8 goals in their 6th game. Give an updated numerical estimate of  $\lambda$  after six games and compute the probability that the Reign score exactly 5 goals in their 7th game.

**What to Submit:**

- **Part a:** An expression for the MLE of  $\lambda$  after  $n$  games and relevant derivation
- **Part b:** A numerical estimate for  $\lambda$  and the probability that the Reign score 4 goals in their sixth game
- **Part c:** A numerical estimate for  $\lambda$  and the probability that the Reign score 5 goals in their seventh game

**Solution:**

**Part a** The likelihood function is

$$L = \prod_{i=1}^n P(x_i|\lambda) = \prod_{i=1}^n e^{-\lambda} \frac{\lambda^{x_i}}{x_i!}.$$

Thus, the log-likelihood function is

$$\begin{aligned} \log L &= \sum_{i=1}^n [(-\lambda) + \log(\lambda^{x_i}) - \log(x_i!)] \\ &= \sum_{i=1}^n [(-\lambda) + x_i \log(\lambda) - \log(x_i!)] . \end{aligned}$$

$$\frac{\partial \log L}{\partial \lambda} = -n + \sum_{i=1}^n \frac{x_i}{\lambda}.$$

Set  $\frac{\partial \log L}{\partial \lambda} = 0$ , and we have

$$\hat{\lambda}_{MLE} = \frac{\sum_{i=1}^n x_i}{n}.$$

**Part b** With the equation from part a and the first five score data, we have the estimation of  $\lambda$ , that is,

$$\hat{\lambda}_{MLE} = \frac{3 + 7 + 5 + 0 + 2}{5} = 3.4.$$

Thus, the probability that the Reign score 4 goals in their sixth game is

$$P(x = 4|\lambda = 3.4) = e^{-3.4} \frac{3.4^4}{4!} = 0.1858$$

**Part c** The updated estimation of  $\lambda$  after the sixth score is

$$\hat{\lambda}_{MLE} = \frac{3 + 7 + 5 + 0 + 2 + 8}{6} = \frac{25}{6} = 4.1667.$$

Thus, the probability that the Reign score 5 goals in their 7th game is

$$P(x = 5|\lambda = \frac{25}{6}) = e^{-\frac{25}{6}} \frac{(\frac{25}{6})^5}{5!} = 0.1623.$$

## Overfitting

B1. Suppose we have  $N$  labeled samples  $S = \{(x_i, y_i)\}_{i=1}^N$  drawn i.i.d. from an underlying distribution  $\mathcal{D}$ . Suppose we decide to break this set into a set  $S_{\text{train}}$  of size  $N_{\text{train}}$  and a set  $S_{\text{test}}$  of size  $N_{\text{test}}$  samples for our training and test set, so  $N = N_{\text{train}} + N_{\text{test}}$ , and  $S = S_{\text{train}} \cup S_{\text{test}}$ . Recall the definition of the true least squares error of  $f$ :

$$\epsilon(f) = \mathbb{E}_{(x,y) \sim \mathcal{D}}[(f(x) - y)^2],$$

where the subscript  $(x, y) \sim \mathcal{D}$  makes clear that our input-output pairs are sampled according to  $\mathcal{D}$ . Our training and test losses are defined as:

$$\begin{aligned}\hat{\epsilon}_{\text{train}}(f) &= \frac{1}{N_{\text{train}}} \sum_{(x,y) \in S_{\text{train}}} (f(x) - y)^2 \\ \hat{\epsilon}_{\text{test}}(f) &= \frac{1}{N_{\text{test}}} \sum_{(x,y) \in S_{\text{test}}} (f(x) - y)^2\end{aligned}$$

We then train our algorithm using the training set to obtain  $\hat{f}$ .

- a. [2 points] (bias: the test error) For all fixed  $f$  (before we've seen any data) show that

$$\mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(f)] = \mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(f)] = \epsilon(f),$$

where  $\mathbb{E}_{S_{\text{train}}}$  indicates we are computing the expectation over all possible sets  $S_{\text{train}}$  and  $\mathbb{E}_{S_{\text{test}}}$  indicates we are computing the expectation over all possible sets  $S_{\text{test}}$ .

Use a similar line of reasoning to show that the test error is an unbiased estimate of our true error for  $\hat{f}$ . Specifically, show that:

$$\mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(\hat{f})] = \epsilon(\hat{f})$$

- b. [3 points] (bias: the train/dev error) Is the above equation true (in general) with regards to the training loss? Specifically, does  $\mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(\hat{f})]$  equal  $\epsilon(\hat{f})$ ? If so, why? If not, give a clear argument as to where your previous argument breaks down.
- c. [5 points] Let  $\mathcal{F} = (f_1, f_2, \dots)$  be a collection of functions and let  $\hat{f}_{\text{train}}$  minimize the training error such that  $\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}}) \leq \hat{\epsilon}_{\text{train}}(f)$  for all  $f \in \mathcal{F}$ . Show that

$$\mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}})] \leq \mathbb{E}_{S_{\text{train}}, S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(\hat{f}_{\text{train}})].$$

(Hint: note that

$$\begin{aligned}\mathbb{E}_{S_{\text{train}}, S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(\hat{f}_{\text{train}})] &= \sum_{f \in \mathcal{F}} \mathbb{E}_{S_{\text{train}}, S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(f) \mathbf{1}\{\hat{f}_{\text{train}} = f\}] \\ &= \sum_{f \in \mathcal{F}} \mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(f)] \mathbb{E}_{S_{\text{train}}}[\mathbf{1}\{\hat{f}_{\text{train}} = f\}] \\ &= \sum_{f \in \mathcal{F}} \mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(f)] \mathbb{P}_{S_{\text{train}}}(\hat{f}_{\text{train}} = f)\end{aligned}$$

where the second equality follows from the independence between the train and test set.)

### What to Submit:

- **Part a:** Proof
- **Part b:** Brief Explanation (3-5 sentences)

• **Part c:** Proof

**Solution:**

**Part a** Because  $f$  is a fixed function before we have seen any data,  $f(x)$  is independent of the  $S_{\text{train}}$  and  $S_{\text{test}}$ .

$$\begin{aligned}\mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(f)] &= \mathbb{E}\left[\frac{1}{N_{\text{train}}} \sum_{(x,y) \in S_{\text{train}}} (f(x) - y)^2\right] \\ &= \frac{1}{N_{\text{train}}} \mathbb{E}\left[\sum_{(x,y) \in S_{\text{train}}} (f(x) - y)^2\right] \\ &= \frac{1}{N_{\text{train}}} N_{\text{train}} \mathbb{E}_{(x,y) \sim \mathbb{D}}(f(x) - y)^2 \\ &= \mathbb{E}_{(x,y) \sim \mathbb{D}}(f(x) - y)^2 = \epsilon(f)\end{aligned}$$

Similarly, we have

$$\begin{aligned}\mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(f)] &= \frac{1}{N_{\text{test}}} \mathbb{E}\left[\sum_{(x,y) \in S_{\text{test}}} (f(x) - y)^2\right] \\ &= \frac{1}{N_{\text{test}}} N_{\text{test}} \mathbb{E}_{(x,y) \sim \mathbb{D}}(f(x) - y)^2 \\ &= \mathbb{E}_{(x,y) \sim \mathbb{D}}(f(x) - y)^2 = \epsilon(f)\end{aligned}$$

Again, because that  $\hat{f}(x)$  is independent of  $S_{\text{test}}$ , we have

$$\begin{aligned}\mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(\hat{f})] &= \frac{1}{N_{\text{test}}} \mathbb{E}\left[\sum_{(x,y) \in S_{\text{test}}} (\hat{f}(x) - y)^2\right] \\ &= \frac{1}{N_{\text{test}}} N_{\text{test}} \mathbb{E}_{(x,y) \sim \mathbb{D}}(\hat{f}(x) - y)^2 \\ &= \mathbb{E}_{(x,y) \sim \mathbb{D}}(\hat{f}(x) - y)^2 = \epsilon(\hat{f})\end{aligned}$$

**Part b** No. Because  $\hat{f}(x)$  is trained on the  $S_{\text{train}}$ , and thus, it is dependent on the  $S_{\text{train}}$ , making it different from sampling from  $\mathbb{D}$ .

**Part c** Based on the hint, we have

$$\mathbb{E}_{S_{\text{train}}, S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(\hat{f}_{\text{train}})] = \sum_{f \in \mathcal{F}} \mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(f)] \mathbb{P}_{S_{\text{train}}}(\hat{f}_{\text{train}} = f).$$

Then, because  $\mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(f)] = \mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(f)]$  and  $\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}}) \leq \hat{\epsilon}_{\text{train}}(f)$ , we have

$$\begin{aligned}\mathbb{E}_{S_{\text{train}}, S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(\hat{f}_{\text{train}})] &= \sum_{f \in \mathcal{F}} \mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(f)] \mathbb{P}_{S_{\text{train}}}(\hat{f}_{\text{train}} = f) \\ &= \sum_{f \in \mathcal{F}} \mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(f)] \mathbb{P}_{S_{\text{train}}}(\hat{f}_{\text{train}} = f) \\ &\geq \mathbb{E}_{S_{\text{train}}} \hat{\epsilon}(\hat{f}_{\text{train}}) \sum_{f \in \mathcal{F}} \mathbb{P}_{S_{\text{train}}}(\hat{f}_{\text{train}} = f) \\ &= \mathbb{E}_{S_{\text{train}}} \hat{\epsilon}(\hat{f}_{\text{train}}).\end{aligned}$$

## Bias-Variance tradeoff

B2. For  $i = 1, \dots, n$  let  $x_i = i/n$  and  $y_i = f(x_i) + \epsilon_i$  where  $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$  for some unknown  $f$  we wish to approximate at values  $\{x_i\}_{i=1}^n$ . We will approximate  $f$  with a step function estimator. For some  $m \leq n$  such that  $n/m$  is an integer define the estimator

$$\hat{f}_m(x) = \sum_{j=1}^{n/m} c_j \mathbf{1}\left\{x \in \left(\frac{(j-1)m}{n}, \frac{jm}{n}\right]\right\} \quad \text{where} \quad c_j = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} y_i.$$

Note that  $x \in \left(\frac{(j-1)m}{n}, \frac{jm}{n}\right]$  means  $x$  is in the open-closed interval  $\left(\frac{(j-1)m}{n}, \frac{jm}{n}\right]$ .

Note that this estimator just partitions  $\{1, \dots, n\}$  into intervals  $\{1, \dots, m\}, \{m+1, \dots, 2m\}, \dots, \{n-m+1, \dots, n\}$  and predicts the average of the observations within each interval (see Figure 1).



Figure 1: Step function estimator with  $n = 256$ ,  $m = 16$ , and  $\sigma^2 = 1$ .

By the bias-variance decomposition at some  $x_i$  we have

$$\mathbb{E} \left[ (\hat{f}_m(x_i) - f(x_i))^2 \right] = \underbrace{(\mathbb{E}[\hat{f}_m(x_i)] - f(x_i))^2}_{\text{Bias}^2(x_i)} + \underbrace{\mathbb{E} \left[ (\hat{f}_m(x_i) - \mathbb{E}[\hat{f}_m(x_i)])^2 \right]}_{\text{Variance}(x_i)}$$

- [5 points] Intuitively, how do you expect the bias and variance to behave for small values of  $m$ ? What about large values of  $m$ ?
- [5 points] If we define  $\bar{f}^{(j)} = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} f(x_i)$  and the *average bias-squared* as

$$\frac{1}{n} \sum_{i=1}^n (\mathbb{E}[\hat{f}_m(x_i)] - f(x_i))^2,$$

show that

$$\frac{1}{n} \sum_{i=1}^n (\mathbb{E}[\hat{f}_m(x_i)] - f(x_i))^2 = \frac{1}{n} \sum_{j=1}^{n/m} \sum_{i=(j-1)m+1}^{jm} (\bar{f}^{(j)} - f(x_i))^2$$

- c. [5 points] If we define the *average variance* as  $\mathbb{E} \left[ \frac{1}{n} \sum_{i=1}^n (\hat{f}_m(x_i) - \mathbb{E}[\hat{f}_m(x_i)])^2 \right]$ , show (both equalities)

$$\mathbb{E} \left[ \frac{1}{n} \sum_{i=1}^n (\hat{f}_m(x_i) - \mathbb{E}[\hat{f}_m(x_i)])^2 \right] = \frac{1}{n} \sum_{j=1}^{n/m} m \mathbb{E}[(c_j - \bar{f}^{(j)})^2] = \frac{\sigma^2}{m}$$

- d. [5 points] By the Mean-Value theorem we have that

$$\min_{i=(j-1)m+1, \dots, jm} f(x_i) \leq \bar{f}^{(j)} \leq \max_{i=(j-1)m+1, \dots, jm} f(x_i)$$

Suppose  $f$  is  $L$ -Lipschitz<sup>a</sup> so that  $|f(x_i) - f(x_j)| \leq \frac{L}{n} |i - j|$  for all  $i, j \in \{1, \dots, n\}$  for some  $L > 0$ .

Show that the average bias-squared is  $O(\frac{L^2 m^2}{n^2})$ . Using the expression for average variance above, the total error behaves like  $O(\frac{L^2 m^2}{n^2} + \frac{\sigma^2}{m})$ . Minimize this expression with respect to  $m$ .

Does this value of  $m$ , and the total error when you plug this value of  $m$  back in, behave in an intuitive way with respect to  $n$ ,  $L$ ,  $\sigma^2$ ? That is, how does  $m$  scale with each of these parameters? It turns out that this simple estimator (with the optimized choice of  $m$ ) obtains the best achievable error rate up to a universal constant in this setup for this class of  $L$ -Lipschitz functions (see Tsybakov's *Introduction to Nonparametric Estimation* for details).<sup>b</sup>

#### What to Submit:

- **Part a:** 1-2 sentences
- **Part b:** Proof
- **Part c:** Proof
- **Part d:** Derivation of minimal error with respect to  $m$ . 1-2 sentences about scaling of  $m$  with parameters.

**Solution: Part a** With small  $m$ , the bias will be small, but the variance will be large; while with large  $m$ , the bias will be large, but the variance will be small.

**Part b** Because  $y_i = f(x_i) + \epsilon_i$  and  $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ , we have  $\mathbb{E}y_i = f(x_i)$ . Thus,

$$\begin{aligned} \mathbb{E}\hat{f}_m(X_i) &= \mathbb{E} \sum_{j=1}^{n/m} \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} y_i \mathbf{1}\{x \in \left(\frac{(j-1)m}{n}, \frac{jm}{n}\right]\} \\ &= \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} f(x_i) \\ &= \bar{f}^{(j)} \end{aligned}$$

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n (\mathbb{E}[\hat{f}_m(x_i)] - f(x_i))^2 &= \frac{1}{n} \sum_{j=1}^{n/m} \sum_{i=(j-1)m+1}^{jm} (\mathbb{E}[\hat{f}_m(x_i)] - f(x_i))^2 \\ &= \frac{1}{n} \sum_{j=1}^{n/m} \sum_{i=(j-1)m+1}^{jm} (\bar{f}^{(j)} - f(x_i))^2 \end{aligned}$$

**Part c**

$$\begin{aligned}
\mathbb{E} \left[ \frac{1}{n} \sum_{i=1}^n (\hat{f}_m(x_i) - \mathbb{E}[\hat{f}_m(x_i)])^2 \right] &= \frac{1}{n} \sum_{j=1}^n (\hat{f}_m(x_j) - \mathbb{E}[\hat{f}_m(x_j)])^2 \\
&= \frac{1}{n} n \text{Var}(\hat{f}_m(x_i)) \\
&= \frac{1}{m^2} m \text{Var}(y_i) \\
&= \frac{\sigma^2}{m}
\end{aligned}$$

Also, from part b, we know that  $\bar{f}^{(j)} = \mathbb{E}\hat{f}_m(X_i)$ , and thus,

$$\begin{aligned}
\frac{1}{n} \sum_{j=1}^{n/m} m \mathbb{E}[(c_j - \bar{f}^{(j)})^2] &= \frac{1}{n} \sum_{j=1}^{n/m} \sum_{i=(j-1)m+1}^{jm} \mathbb{E}[(c_j - \bar{f}^{(j)})^2] \\
&= \frac{1}{n} \sum_{j=1}^{n/m} \sum_{i=(j-1)m+1}^{jm} (\hat{f}_m(x_i) - \mathbb{E}[\hat{f}_m(x_i)])^2 \\
&= \frac{\sigma^2}{m}
\end{aligned}$$

In summary, we have

$$\mathbb{E} \left[ \frac{1}{n} \sum_{i=1}^n (\hat{f}_m(x_i) - \mathbb{E}[\hat{f}_m(x_i)])^2 \right] = \frac{1}{n} \sum_{j=1}^{n/m} m \mathbb{E}[(c_j - \bar{f}^{(j)})^2] = \frac{\sigma^2}{m}$$

**Part d** Given that,

$$\min_{i=(j-1)m+1, \dots, jm} f(x_i) \leq \bar{f}^{(j)} \leq \max_{i=(j-1)m+1, \dots, jm} f(x_i)$$

Thus,

$$(\bar{f}^{(j)} - f(x_i))^2 \leq \left[ \max_{i=(j-1)m+1, \dots, jm} f(x_i) - \min_{i=(j-1)m+1, \dots, jm} f(x_i) \right]^2$$

Given that

$$|f(x_i) - f(x_j)| \leq \frac{L}{n} |i - j|$$

We have

$$\begin{aligned}
(\bar{f}^{(j)} - f(x_i))^2 &\leq \left[ \max_{i=(j-1)m+1, \dots, jm} f(x_i) - \min_{i=(j-1)m+1, \dots, jm} f(x_i) \right]^2 \\
&\leq \frac{L^2}{n^2} \left[ \arg \max_{i=(j-1)m, \dots, jm} f(x_i) - \arg \min_{i=(j-1)m, \dots, jm} f(x_i) \right]^2 \\
&\leq \frac{L^2(m-1)^2}{n^2}
\end{aligned}$$



Thus,

$$\frac{1}{n} \sum_{j=1}^{n/m} \sum_{i=(j-1)m+1}^{jm} (\bar{f}^{(j)} - f(x_i))^2 \leq \frac{1}{n} \frac{n}{m} m \frac{L^2(m-1)^2}{n^2} \sim O\left(\frac{L^2 m^2}{n^2}\right)$$

Given the bias-variance decomposition, we have the total error behaves like  $O(\frac{L^2 m^2}{n^2} + \frac{\sigma^2}{m})$ . Then, we can get the derivative in terms of  $m$ , and get

$$\frac{\partial}{\partial m} \frac{L^2 m^2}{n^2} + \frac{\sigma^2}{m} = \frac{2L^2 m}{n^2} - \frac{\sigma^2}{m^2} = 0$$

Thus,

$$m = \left(\frac{\sigma^2 n^2}{2L^2}\right)^{1/3}.$$

We can plug the  $m$  back in the total error, and the total error behaves like  $O(\frac{L^{2/3} \sigma^{4/3}}{n^{2/3}} + \frac{L^{2/3} \sigma^{2/3}}{n^{2/3}})$ , which is equal to

$$O\left(\frac{L^{2/3} \sigma^{4/3}}{n^{2/3}}\right).$$

Both the value of  $m$  and total error behave in an intuitive way because:

When  $n$  increases, meaning that the sample size is increasing,  $m$  will increase, leading to a simpler model, and the total error will decrease.

When  $L$  increases, meaning that the function is changing more dramatically,  $m$  will decrease, leading to a more complex model, and the total error will increase.

When  $\sigma$  increases, meaning that the noise is increasing,  $m$  will increase, leading to a simpler model, and the total error will increase.

---

<sup>a</sup>A function is  $L$ -Lipschitz if there exists  $L \geq 0$  such that  $\|f(x_i) - f(x_j)\| \leq L\|x_i - x_j\|$ , for all  $x_i, x_j$

<sup>b</sup>This setup of each  $x_i$  deterministically placed at  $i/n$  is a good approximation for the more natural setting where each  $x_i$  is drawn uniformly at random from  $[0, 1]$ . In fact, one can redo this problem and obtain nearly identical conclusions, but the calculations are messier.

# Polynomial Regression

Relevant Files<sup>1</sup>:

- `polyreg.py`
- `linreg_closedform.py`
- `plot_polyreg_univariate.py`
- `plot_polyreg_learningCurve.py`

A3. Recall that polynomial regression learns a function  $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_d x^d$ , where  $d$  represents the polynomial's highest degree. We can equivalently write this in the form of a linear model with  $d$  features

$$h_{\theta}(x) = \theta_0 + \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \dots + \theta_d \phi_d(x) , \quad (1)$$

using the basis expansion that  $\phi_j(x) = x^j$ . Notice that, with this basis expansion, we obtain a linear model where the features are various powers of the single univariate  $x$ . We're still solving a linear regression problem, but are fitting a polynomial function of the input.

- a. **[8 points] Implement regularized polynomial regression in `polyreg.py`.** You may implement it however you like, using gradient descent or a closed-form solution. However, we would recommend the closed-form solution since the data sets are small; for this reason, we've included an example closed-form implementation of regularized linear regression in `linreg_closedform.py` (you are welcome to build upon this implementation, but make CERTAIN you understand it, since you'll need to change several lines of it). Note that all matrices are 2D NumPy arrays in the implementation.

- `__init__(degree=1, regLambda=1E-8)`: constructor with arguments of  $d$  and  $\lambda$
- `fit(X,Y)`: method to train the polynomial regression model
- `predict(X)`: method to use the trained polynomial regression model for prediction
- `polyfeatures(X, degree)`: expands the given  $n \times 1$  matrix  $X$  into an  $n \times d$  matrix of polynomial features of degree  $d$ . Note that the returned matrix will not include the zero-th power.

Note that the `polyfeatures(X, degree)` function maps the original univariate data into its higher order powers. Specifically,  $X$  will be an  $n \times 1$  matrix ( $X \in \mathbb{R}^{n \times 1}$ ) and this function will return the polynomial expansion of this data, a  $n \times d$  matrix. Note that this function will **not** add in the zero-th order feature (i.e.,  $x_0 = 1$ ). You should add the  $x_0$  feature separately, outside of this function, before training the model.



Figure 2: Fit of polynomial regression with  $\lambda = 0$  and  $d = 8$

By not including the  $x_0$  column in the matrix `polyfeatures()`, this allows the `polyfeatures` function to be more general, so it could be applied to multi-variate data as well. (If it did add the  $x_0$  feature, we'd

---

<sup>1</sup>**Bold text** indicates files or functions that you will need to complete; you should not need to modify any of the other files.

end up with multiple columns of 1's for multivariate data.)

Also, notice that the resulting features will be badly scaled if we use them in raw form. For example, with a polynomial of degree  $d = 8$  and  $x = 20$ , the basis expansion yields  $x^1 = 20$  while  $x^8 = 2.56 \times 10^{10}$  – an absolutely huge difference in range. Consequently, we will need to standardize the data before solving linear regression. Standardize the data in `fit()` after you perform the polynomial feature expansion. You'll need to apply the same standardization transformation in `predict()` before you apply it to new data.

- b. [2 points] Run `plot_polyreg_univariate.py` to test your implementation, which will plot the learned function. In this case, the script fits a polynomial of degree  $d = 8$  with no regularization  $\lambda = 0$ . From the plot, we see that the function fits the data well, but will not generalize well to new data points. Try increasing the amount of regularization, and in 1-2 sentences, describe the resulting effect on the function (you may also provide an additional plot to support your analysis).

### What to Submit:

- **Part a: Code** on Gradescope through coding submission.
- **Part b:** 1-2 sentence description of the effect of increasing regularization.
- **Part b:** Plots before and after increase in regularization.

### Solution:

**Part a** Please see the code on Gradescope.

**Part b** As you can see from  $\lambda = 0$  to  $\lambda = 4$ , the function is simpler. Thus, increasing the regularization increases the bias, making the model less complex.

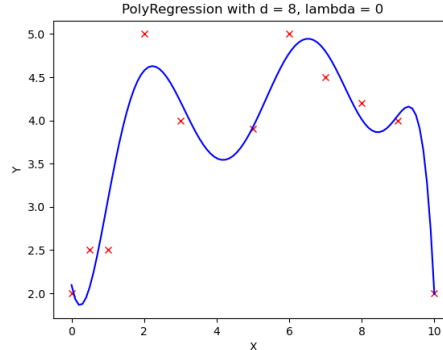


Figure 3: Fit of polynomial regression with  $\lambda = 0$  and  $d = 8$

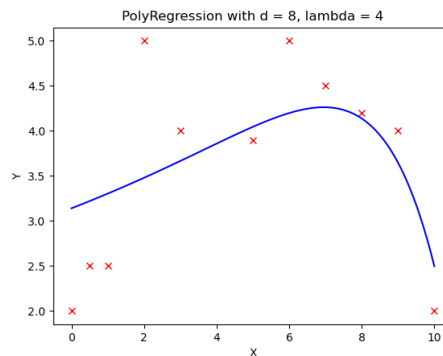


Figure 4: Fit of polynomial regression with  $\lambda = 4$  and  $d = 8$

A4. [10 points] In this problem we will examine the bias-variance tradeoff through learning curves. Learning curves provide a valuable mechanism for evaluating the bias-variance tradeoff.

**Implement the `learningCurve()` function in `polyreg.py` to compute the learning curves for a given training/test set.** The `learningCurve(Xtrain, ytrain, Xtest, ytest, degree, regLambda)` function should take in the training data (`Xtrain, ytrain`), the testing data (`Xtest, ytest`), and values for the polynomial degree  $d$  and regularization parameter  $\lambda$ . The function should return two arrays, `errorTrain` (the array of training errors) and `errorTest` (the array of testing errors). The  $i^{th}$  index (start from 0) of each array should return the training error (or testing error) for learning with  $i + 1$  training instances. Note that the  $0^{th}$  index actually won't matter, since we typically start displaying the learning curves with two or more instances.

When computing the learning curves, you should learn on `Xtrain[0:i]` for  $i = 1, \dots, \text{numInstances}(\text{Xtrain})$ , each time computing the testing error over the **entire** test set. There is no need to shuffle the training data, or to average the error over multiple trials – just produce the learning curves for the given training/testing sets with the instances in their given order. Recall that the error for regression problems is given by

$$\frac{1}{n} \sum_{i=1}^n (h_{\theta}(\mathbf{x}_i) - y_i)^2 . \quad (2)$$

Once the function is written to compute the learning curves, run the `plot_polyreg_learningCurve.py` script to plot the learning curves for various values of  $\lambda$  and  $d$ . You should see plots similar to the following:

**Notice the following:**

- The y-axis is using a log-scale and the ranges of the y-scale are all different for the plots. The dashed black line indicates the  $y = 1$  line as a point of reference between the plots.
- The plot of the unregularized model with  $d = 1$  shows poor training error, indicating a high bias (i.e., it is a standard univariate linear regression fit).
- The plot of the (almost) unregularized model ( $\lambda = 10^{-6}$ ) with  $d = 8$  shows that the training error is low, but that the testing error is high. There is a huge gap between the training and testing errors caused by the model overfitting the training data, indicating a high variance problem.
- As the regularization parameter increases (e.g.,  $\lambda = 1$ ) with  $d = 8$ , we see that the gap between the training and testing error narrows, with both the training and testing errors converging to a low value. We can see that the model fits the data well and generalizes well, and therefore does not have either a high bias or a high variance problem. Effectively, it has a good tradeoff between bias and variance.
- Once the regularization parameter is too high ( $\lambda = 100$ ), we see that the training and testing errors are once again high, indicating a poor fit. Effectively, there is too much regularization, resulting in high bias.

**Submit plots for the same values of  $d$  and  $\lambda$  shown here.** Make absolutely certain that you understand these observations, and how they relate to the learning curve plots. In practice, we can choose the value for  $\lambda$  via cross-validation to achieve the best bias-variance tradeoff.

**Note: your learning curves slightly differ from ones above depending on whether you use ‘`np.linalg.solve`’ or directly implement the closed-form solution. Both solutions are correct.**

**What to Submit:**

- **Plots** (or single plot with many subplots) of learning curves for  $(d, \lambda) \in \{(1, 0), (4, 10^{-6}), (8, 10^{-6}), (8, 0.1), (8, 1), (8, 100)\}$ .
- **Code** on Gradescope through coding submission

**Solution:** Please see the plot below.

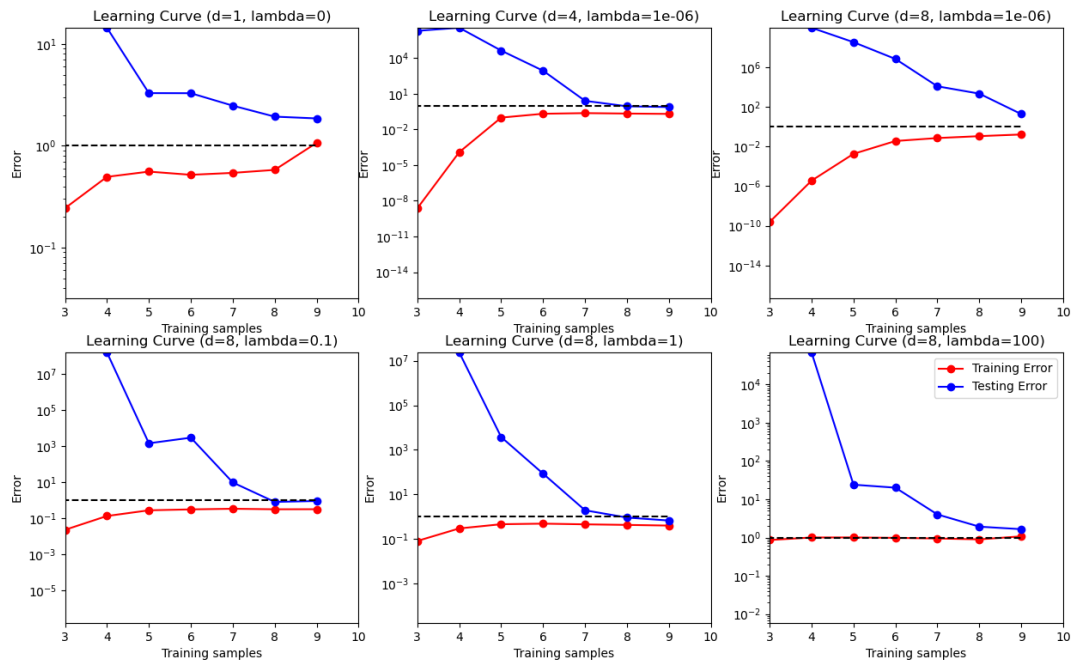


Figure 5: Learning Curves

# Ridge Regression on MNIST

**Relevant Files** (you should not need to modify any of the other files for this part):

- `ridge_regression.py`

A5. In this problem, we will implement a regularized least squares classifier for the MNIST data set. The task is to classify handwritten images of numbers between 0 to 9.

You are **NOT** allowed to use any of the pre-built classifiers in `sklearn`. Feel free to use any method from `numpy` or `scipy`. **Remember:** if you are inverting a matrix in your code, you are probably doing something wrong (Hint: look at `scipy.linalg.solve`).



Figure 6: Sample images from the MNIST data set.

Each example has features  $x_i \in \mathbb{R}^d$  (with  $d = 28 * 28 = 784$ ) and label  $z_j \in \{0, \dots, 9\}$ . **Since images are represented as 784-dimensional vectors, you can visualize a single example  $x_i$  with `imshow` after reshaping it to its original  $28 \times 28$  image shape** (and noting that the label  $z_j$  is accurate). Checkout figure 6 for some sample images. We wish to learn a predictor  $\hat{f}$  that takes as input a vector in  $\mathbb{R}^d$  and outputs an index in  $\{0, \dots, 9\}$ . We define our training and testing classification error on a predictor  $f$  as

$$\hat{\epsilon}_{\text{train}}(f) = \frac{1}{N_{\text{train}}} \sum_{(x,z) \in \text{Training Set}} \mathbf{1}\{f(x) \neq z\}$$
$$\hat{\epsilon}_{\text{test}}(f) = \frac{1}{N_{\text{test}}} \sum_{(x,z) \in \text{Test Set}} \mathbf{1}\{f(x) \neq z\}$$

We will use one-hot encoding of the labels: for each observation  $(x, z)$ , the original label  $z \in \{0, \dots, 9\}$  is mapped to the standard basis vector  $e_{z+1}$  where  $e_i$  is a vector of size  $k$  containing all zeros except for a 1 in the  $i^{\text{th}}$  position (positions in these vectors are indexed starting at one, hence the  $z + 1$  offset for the digit labels). We adopt the notation where we have  $n$  data points in our training objective with features  $x_i \in \mathbb{R}^d$  and label one-hot encoded as  $y_i \in \{0, 1\}^k$ . Here,  $k = 10$  since there are 10 digits.

- a. **[10 points]** In this problem we will choose a linear classifier to minimize the regularized least squares objective:

$$\widehat{W} = \operatorname{argmin}_{W \in \mathbb{R}^{d \times k}} \sum_{i=1}^n \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2$$

Note that  $\|W\|_F$  corresponds to the Frobenius norm of  $W$ , i.e.  $\|W\|_F^2 = \sum_{i=1}^d \sum_{j=1}^k W_{i,j}^2$ . To classify a point  $x_i$  we will use the rule  $\arg \max_{j=0,\dots,9} e_{j+1}^T \widehat{W}^T x_i$ . Note that if  $W = [w_1 \ \dots \ w_k]$  then

$$\begin{aligned} \sum_{i=1}^n \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2 &= \sum_{j=1}^k \left[ \sum_{i=1}^n (e_j^T W^T x_i - e_j^T y_i)^2 + \lambda \|W e_j\|_2^2 \right] \\ &= \sum_{j=1}^k \left[ \sum_{i=1}^n (w_j^T x_i - e_j^T y_i)^2 + \lambda \|w_j\|_2^2 \right] \\ &= \sum_{j=1}^k [\|X w_j - Y e_j\|^2 + \lambda \|w_j\|_2^2] \end{aligned}$$

where  $X = [x_1 \ \dots \ x_n]^T \in \mathbb{R}^{n \times d}$  and  $Y = [y_1 \ \dots \ y_n]^T \in \mathbb{R}^{n \times k}$ . **Show that**

$$\widehat{W} = (X^T X + \lambda I)^{-1} X^T Y$$

b. [9 points]

- Implement a function `train` that takes as input  $X \in \mathbb{R}^{n \times d}$ ,  $Y \in \{0,1\}^{n \times k}$ ,  $\lambda > 0$  and returns  $\widehat{W} \in \mathbb{R}^{d \times k}$ .
- Implement a function `one_hot` that takes as input  $Y \in \{0, \dots, k-1\}^n$ , and returns  $Y \in \{0,1\}^{n \times k}$ .
- Implement a function `predict` that takes as input  $W \in \mathbb{R}^{d \times k}$ ,  $X' \in \mathbb{R}^{m \times d}$  and returns an  $m$ -length vector with the  $i$ th entry equal to  $\arg \max_{j=0,\dots,9} e_j^T W^T x'_i$  where  $x'_i \in \mathbb{R}^d$  is a column vector representing the  $i$ th example from  $X'$ .
- Using the functions you coded above, train a model to estimate  $\widehat{W}$  on the MNIST training data with  $\lambda = 10^{-4}$ , and make label predictions on the test data. This behavior is implemented in the `main` function provided in a zip file.

c. [1 point] **What are the training and testing errors of the classifier trained as above?**

d. [2 points] **Using matplotlib's `imshow` function, plot any 10 samples from the test data whose labels are incorrectly predicted by the classifier. Notice any patterns?**

Once you finish this problem question, you should have a powerful handwritten digit classifier! Curious to know how it compares to other models, including the almighty *Neural Networks*? Check out the **linear classifier (1-layer NN)** on the [official MNIST leaderboard](#). (The model we just built is actually a 1-layer neural network: more on this soon!)

**What to Submit:**

- **Part a:** Derivation of expression for  $\widehat{W}$
- **Part b:** Code on Gradescope through coding submission
- **Part c:** Values of training and testing errors
- **Part d:** Display of 10 images whose labels are incorrectly predicted by the classifier. 1-2 sentences reasoning why.

**Solution: Part a** As provided by the hint, we have

$$\begin{aligned} \sum_{i=1}^n \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2 &= \sum_{j=1}^k [\|X w_j - Y e_j\|^2 + \lambda \|w_j\|_2^2] \\ &= \sum_{j=1}^k [(X w_j - Y e_j)^T (X w_j - Y e_j) + e_j^T w_j]. \end{aligned}$$

Let

$$\frac{\partial}{\partial w_j} \left[ \sum_{j=1}^k [(Xw_j - Ye_j)^T (Xw_j - Ye_j) + \frac{1}{2} w_j^2] \right] = 2(X^T X w_j) - 2(X^T y_j) + w_j = 0.$$

We have

$$(X^T X + \lambda I) w_j = X^T y_j$$

$$w_j = (X^T X + \lambda I)^{-1} X^T y_j$$

Thus,

$$\widehat{W} = (X^T X + \lambda I)^{-1} X^T Y$$

**Part b** Please see the code on Gradescope.

**Part c** The train error is 14.805%, and the test error is 14.66%.

**Part d** The 10 images are shown below. As we can see, some handwriting is ambiguous or has rare patterns, which may not be covered in the training dataset or may not be appropriately incorporated into the model.

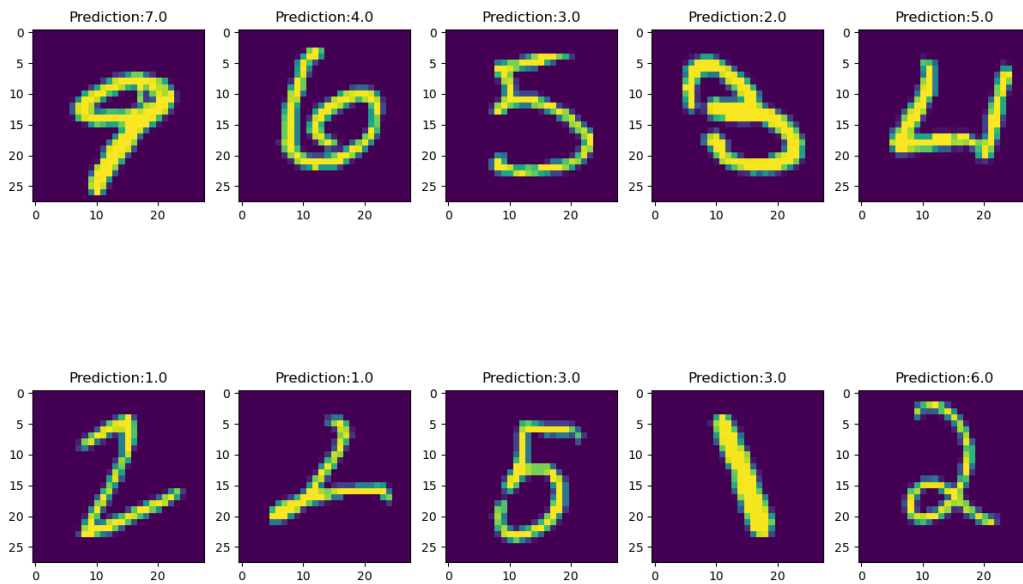


Figure 7: Mis-classified plots

## Administrative

A6.

- a. [2 points] About how many hours did you spend on this homework? There is no right or wrong answer :)

**Solution:** I spent 30 hours on this homework.