

Homework #1

CSE 446/546: Machine Learning
Professor Pang Wei Koh & Sewoong Oh
Due: **Wednesday, October 22, 2025, 11:59pm**
A: 57 points, B: 35 points

Jiayao Huang

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. \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.

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/cse446/25sp/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.

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 bias-variance tradeoff?
- b. [2 points] What **typically** happens to bias and variance when the model complexity increases/decreases?
- c. [1 point] True or False: A learning algorithm will always generalize better if we use fewer features to represent our data.
- d. [2 points] True or False: Hyperparameters should be tuned on the test set. 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 a-e:** Brief (2-3 sentence) explanation
- **Parts c-e:** True or False

Part a.

Bias is error from over simplistic modeling (underfitting). High-bias models would fail to show certain features from the data.

Variance is error from excessive complexity, making the model over sensitive to small fluctuations in the training data (overfitting).

Bias-variance tradeoff is to find an appropriate model complexity such that the total error from bias and variance is minimal, i.e. a model neither too complex nor too simple to fit the data.

Part b.

As model complexity increases, typically bias decreases and variance increases.

As model complexity decreases, typically bias increases and variance decreases.

Part c. False

Although using fewer features can help generalize data, removing too many features can cause underfitting and hurt generalization by failing to capture important patterns in the data.

Part d. False

We cannot tune hyperparameters on the test set.

Proper procedure: 1) split data into training, validation, and test sets; 2) train models with different hyperparameters on the training set; 3) evaluate their performance on validation set; 4) select hyperparameters with the best performance; 5) do evaluation on the test set.

Part e. False

Training error provides an underestimate of the true error.

Because the model was specifically optimized for the training data. When we test against unseen data whose features might differ from the trained ones, we would get higher true error, especially for complex models.

Maximum Likelihood Estimation (MLE)

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

$$[2, 4, 6, 0, 1].$$

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 λ assigns every non-negative integer $x = 0, 1, 2, \dots$ a probability given by

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

- [5 points]** Derive an expression for the maximum-likelihood estimate of the parameter λ 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.)
- [2 points]** Give a numerical estimate of λ after the first five games. Given this λ , what is the probability that the Reign score 6 goals in their next game?
- [2 points]** Suppose the Reign score 8 goals in their 6th game. Give an updated numerical estimate of λ after six games and compute the probability that the Reign score 6 goals in their 7th game.

What to Submit:

- **Part a:** An expression for the MLE of λ after n games and relevant derivation
- **Parts b-c:** A numerical estimate for λ and the probability that the Reign score 6 next game.

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

Since the data are i.i.d. with respect to Poisson distribution, the likelihood function for n games is

$$L_n(\lambda) = \prod_{i=1}^n e^{-\lambda} \frac{\lambda^{x_i}}{x_i!} = e^{-n\lambda} \lambda^{\sum_{i=1}^n x_i} \left(\prod_{i=1}^n x_i! \right)^{-1}$$

The log likelihood is

$$\ell_n(\lambda) = \log L_n(\lambda) = -n\lambda + \left(\sum_{i=1}^n x_i \right) \ln \lambda - \sum_{i=1}^n \ln(x_i!)$$

Take the derivative with respect to λ and set it to zero

$$\frac{d\ell_n}{d\lambda} = -n + \frac{\sum_{i=1}^n x_i}{\lambda} = 0 \quad \Rightarrow \quad \lambda = \frac{\sum_{i=1}^n x_i}{n} = \bar{x}$$

Therefore, we get the MLE as

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

Part b. From part a, we know that for $x = [2, 4, 6, 0, 1]$, the MLE is

$$\hat{\lambda} = \frac{2 + 4 + 6 + 0 + 1}{5} = \frac{13}{5} = 2.6$$

Using Poisson distribution, the probability of scoring 6 next game is

$$P(X = 6) = Poi(x = 6 | \hat{\lambda} = 2.6) = e^{-\hat{\lambda}} \frac{\hat{\lambda}^x}{x!} \Bigg|_{x=6, \hat{\lambda}=2.6} = e^{-2.6} \cdot \frac{2.6^6}{6!} \approx 0.0319$$

Part c. With the additional game, the dataset becomes $[2, 4, 6, 0, 1, 8]$:

$$\hat{\lambda} = \frac{2 + 4 + 6 + 0 + 1 + 8}{6} = \frac{21}{6} = 3.5$$

Using Poisson distribution, the probability of scoring 6 next game is

$$P(X = 6) = Poi(x = 6 | \hat{\lambda} = 3.5) = e^{-\hat{\lambda}} \frac{\hat{\lambda}^x}{x!} \Bigg|_{x=6, \hat{\lambda}=3.5} = e^{-3.5} \cdot \frac{3.5^6}{6!} \approx 0.0771$$

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_{train} of size N_{train} and a set S_{test} of size N_{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 (for example, using linear least squares regression) using the training set to obtain \hat{f} .

- a. [3 points] (bias: the test error) For all fixed f (before we've seen any data) show that

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

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. [4 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. [8 points] Let $\mathcal{F} = (f_1, f_2, \dots)$ be a collection of functions and let \hat{f}_{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

Part a. For any fixed f , the expectation of the training error is:

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

Since samples are i.i.d. from \mathcal{D} , by linearity of expectation, we have

$$\mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(f)] = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \mathbb{E}_{(x_i, y_i) \sim \mathcal{D}}[(f(x_i) - y_i)^2] = \frac{1}{N_{\text{train}}} \cdot N_{\text{train}} \cdot \epsilon(f) = \epsilon(f)$$

where we use the fact that each term in the sum equals $\epsilon(f)$.

Similarly for test error,

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

Thus,

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

For \hat{f} depending on the training samples, we condition on S_{train} ,

$$\mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(\hat{f}) | S_{\text{train}}] = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbb{E}_{(x_i, y_i) \sim \mathcal{D}}[(\hat{f}(x_i) - y_i)^2 | S_{\text{train}}] = \epsilon(\hat{f})$$

Taking expectation over S_{train} , we get

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

Since S_{train} and S_{test} are independent, the left hand side is exactly $\mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(\hat{f})]$.

Thus

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

The test error is an unbiased estimate of the expected true error of \hat{f} .

Part b. No. In general $\mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(\hat{f})] \neq \epsilon(\hat{f})$.

The previous argument breaks down because \hat{f} is dependent on S_{train} . We choose it to minimize the training error. This creates a dependency that biases the training error as an estimate of true error. When we compute

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

\hat{f} is optimized for this specific S_{train} . The training error would then be smaller than the true error $\epsilon(\hat{f})$,

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

Part c. Using the hint, we express the test error expectation using independence of S_{train} and S_{test} :

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

From previous calculation, we know $\mathbb{E}_{S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(f)] = \epsilon(f)$ and $\mathbb{E}_{S_{\text{train}}}[\mathbf{1}\{\hat{f}_{\text{train}} = f\}] = \mathbb{P}(\hat{f}_{\text{train}} = f)$. Then

$$\mathbb{E}_{S_{\text{train}}, S_{\text{test}}}[\hat{\epsilon}_{\text{test}}(\hat{f}_{\text{train}})] = \sum_{f \in \mathcal{F}} \epsilon(f) \cdot \mathbb{P}(\hat{f}_{\text{train}} = f) \quad (1)$$

Now consider the training error expectation, similarly by linearity

$$\mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}})] = \sum_{f \in \mathcal{F}} \mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(f) \cdot \mathbf{1}\{\hat{f}_{\text{train}} = f\}]$$

Since it's the expectation of a product of two random variables dependent on S_{train} , we have

$$\mathbb{E}_{S_{\text{train}}}[\hat{\epsilon}_{\text{train}}(\hat{f}_{\text{train}})] = \sum_{f \in \mathcal{F}} \mathbb{E}[\hat{\epsilon}_{\text{train}}(f) \mid \hat{f}_{\text{train}} = f] \cdot \mathbb{P}(\hat{f}_{\text{train}} = f) \quad (2)$$

When conditioning on $\hat{f}_{\text{train}} = f$, we would have the smallest training error for each f , which means its training error is lower than overall

$$\mathbb{E}[\hat{\epsilon}_{\text{train}}(f) \mid \hat{f}_{\text{train}} = f] \leq \mathbb{E}[\hat{\epsilon}_{\text{train}}(f)] = \epsilon(f)$$

On both sides, multiplying by $\mathbb{P}(\hat{f}_{\text{train}} = f)$ and summing over f , we get

$$\sum_f \mathbb{E}[\hat{\epsilon}_{\text{train}}(f) \mid \hat{f}_{\text{train}} = f] \cdot \mathbb{P}(\hat{f}_{\text{train}} = f) \leq \sum_f \epsilon(f) \cdot \mathbb{P}(\hat{f}_{\text{train}} = f)$$

Combining equations (1) and (2), we finally get

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

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 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).

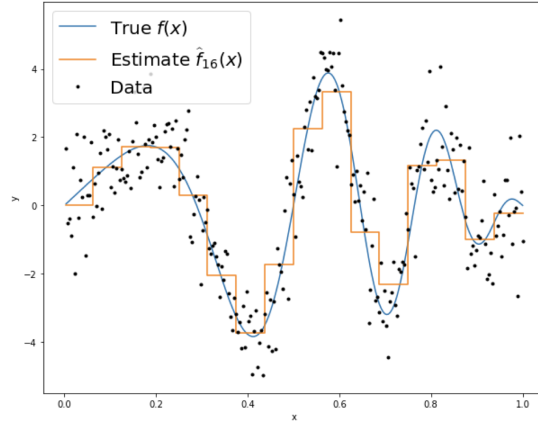


Figure 1: Step function estimator with $n = 256$, $m = 16$, and $\sigma^2 = 1$.

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 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 , σ^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).

What to Submit:

- **Part C:** Proof
- **Part D:** Derivation of minimal error with respect to m . 1-2 sentences about scaling of m with parameters.

Part a.

For small m , intuitively bias is **small** because each step covers a narrow interval; variance is **large** because each estimate provides limited data approximation which makes the estimator sensitive.

For large m , bias is **large** because each step covers a wide interval; variance is **small** because the average of the observations is more representative.

Part b. For x_i in the j -th interval $\left(\frac{(j-1)m}{n}, \frac{jm}{n}\right]$, the estimator is

$$\hat{f}_m(x_i) = c_j = \frac{1}{m} \sum_{k=(j-1)m+1}^{jm} y_k$$

Then we take the expectation over it,

$$\mathbb{E}[\hat{f}_m(x_i)] = \mathbb{E}[c_j] = \frac{1}{m} \sum_{k=(j-1)m+1}^{jm} \mathbb{E}[y_k] = \frac{1}{m} \sum_{k=(j-1)m+1}^{jm} f(x_k) = \bar{f}^{(j)}$$

where the second equality we use that $\mathbb{E}[y_k] = f(x_k)$ and $\mathbb{E}[\epsilon_k] = 0$.

Then the bias at x_i is

$$\mathbb{E}[\hat{f}_m(x_i)] - f(x_i) = \bar{f}^{(j)} - f(x_i)$$

We know the n points are partitioned into $K = n/m$ intervals of size m . The sum over all n points can be written as a double sum by first over blocks $j = 1, \dots, n/m$, then over the m points within each block,

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

Thus the average bias-squared over all n points becomes

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

Part c. For x_i in the j -th interval $\left(\frac{(j-1)m}{n}, \frac{jm}{n}\right]$, $\hat{f}_m(x_i) = c_j$ and $\mathbb{E}[\hat{f}_m(x_i)] = \bar{f}^{(j)}$, so

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

Within each j , the inner sum has m identical terms $(c_j - \bar{f}^{(j)})^2$. By linearity of expectation,

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

Since $c_j = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} y_i$ and $\bar{f}^{(j)} = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} f(x_i)$, inside the expectation we have

$$c_j - \bar{f}^{(j)} = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} (y_i - f(x_i)) = \frac{1}{m} \sum_{i=(j-1)m+1}^{jm} \epsilon_i$$

Thus

$$\mathbb{E}[(c_j - \bar{f}^{(j)})^2] = \mathbb{E} \left[\left(\frac{1}{m} \sum_{i=(j-1)m+1}^{jm} \epsilon_i \right)^2 \right] = \frac{1}{m^2} \sum_{i=(j-1)m+1}^{jm} \mathbb{E}[\epsilon_i^2] = \frac{1}{m^2} \cdot m \cdot \sigma^2 = \frac{\sigma^2}{m}$$

where $\mathbb{E}[\epsilon_i^2] = (\mathbb{E}[\epsilon_i])^2 + \text{Var}[\epsilon_i] = 0 + \sigma^2 = \sigma^2$.

Therefore,

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

Now we show that both the equalities hold.

Part d. We bound the average bias-squared using the Lipschitz condition.

By Mean-Value Theorem, for any i in interval j , let i_0 be the index where $f(x_{i_0}) = \bar{f}^{(j)}$, then

$$|\bar{f}^{(j)} - f(x_i)| \leq |f(x_{i_0}) - f(x_i)| \leq \frac{L}{n} |i_0 - i| \leq \frac{L}{n} \cdot m$$

Then taking square on both sides gives

$$(\bar{f}^{(j)} - f(x_i))^2 \leq \frac{L^2 m^2}{n^2}$$

Summing over all n points and dividing by n , we have

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

Thus the average bias-squared is $O\left(\frac{L^2 m^2}{n^2}\right)$.

Now we minimize the total error $E(m) = O\left(\frac{L^2 m^2}{n^2} + \frac{\sigma^2}{m}\right)$ with respect to m .

Let $E(m) = A \frac{L^2 m^2}{n^2} + B \frac{\sigma^2}{m}$ with $A, B > 0$, then we take the derivative and set it to 0.

$$\frac{dE}{dm} = \frac{2AL^2 m}{n^2} - \frac{B\sigma^2}{m^2} = 0 \Rightarrow \frac{2AL^2 m}{n^2} = \frac{B\sigma^2}{m^2} \Rightarrow \frac{2AL^2 m^3}{n^2} = B\sigma^2$$

Then,

$$m^3 = \frac{B\sigma^2 n^2}{2AL^2} \Rightarrow m^* = \left(\frac{B}{2A}\right)^{1/3} \left(\frac{\sigma^2 n^2}{L^2}\right)^{1/3}$$

which means $m^* \propto \left(\frac{\sigma^2 n^2}{L^2}\right)^{1/3}$.

Plugging back to the total error we get

$$E^* = O\left(\frac{L^2}{n^2} \cdot \frac{\sigma^{4/3} n^{4/3}}{L^{4/3}} + \frac{\sigma^2}{\sigma^{2/3} n^{2/3} L^{-2/3}}\right) = O\left(\sigma^{4/3} L^{2/3} n^{-2/3} + \sigma^{4/3} L^{2/3} n^{-2/3}\right) = O\left(\frac{\sigma^{4/3} L^{2/3}}{n^{2/3}}\right)$$

Scaling: m grows as $n^{2/3}$, decreases as $L^{2/3}$ increases, and increases as $\sigma^{2/3}$ increases.

This can be translated as, larger data allows better partitioning, higher noise requires more smoothing of a function, and smoother functions allow larger intervals.

Polynomial Regression

Relevant Files¹

- **polyreg.py**
- **linreg_closedform.py**
- **plot_polyreg_univariate.py**
- **plot_polyreg_learningCurve.py**

A3. [10 points] 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.

Implement regularized polynomial regression in **polyreg.py**. You may implement it however you like, using gradient descent or a closed-form solution. However, I would recommend the closed-form solution since the data sets are small; for this reason, we've included an example closed-form implementation of 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). You are also welcome to build upon your implementation from the previous assignment, but you must follow the API below. Note that all matrices are actually 2D numpy arrays in the implementation.

- **__init__(degree=1, regLambda=1E-8)**: constructor with arguments of d and λ
- **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.

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 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. Recall that standardization is the process of transforming each feature in our dataset to have mean 0 and variance 1. The typical way to do this is using the Z-Score transformation, defined as

$$\frac{x_i^{(j)} - \mu_i}{\sigma_i}$$

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

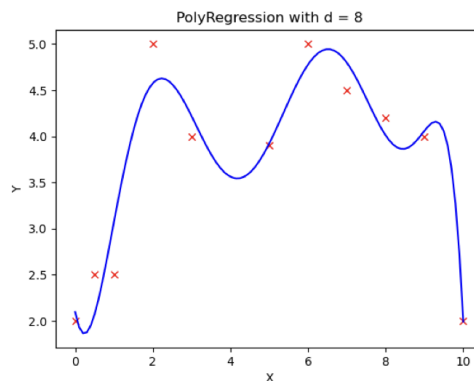


Figure 2: Fit of polynomial regression with $\lambda = 0$ and $d = 8$

where μ_i denotes the mean of feature i and σ_i denotes its standard deviation.

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:

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

Description:

As the regularization parameter λ increases, the learned polynomial function becomes smoother and less flexible. While the model does not match the training data perfectly, it would generalize better to unseen data and reduce overfitting in some sense.

Plots:

The plots are shown in Figure 3. The upper-left plot corresponds to the model before regularization ($\lambda = 0$), while the remaining plots display how the model behaves as λ increases.

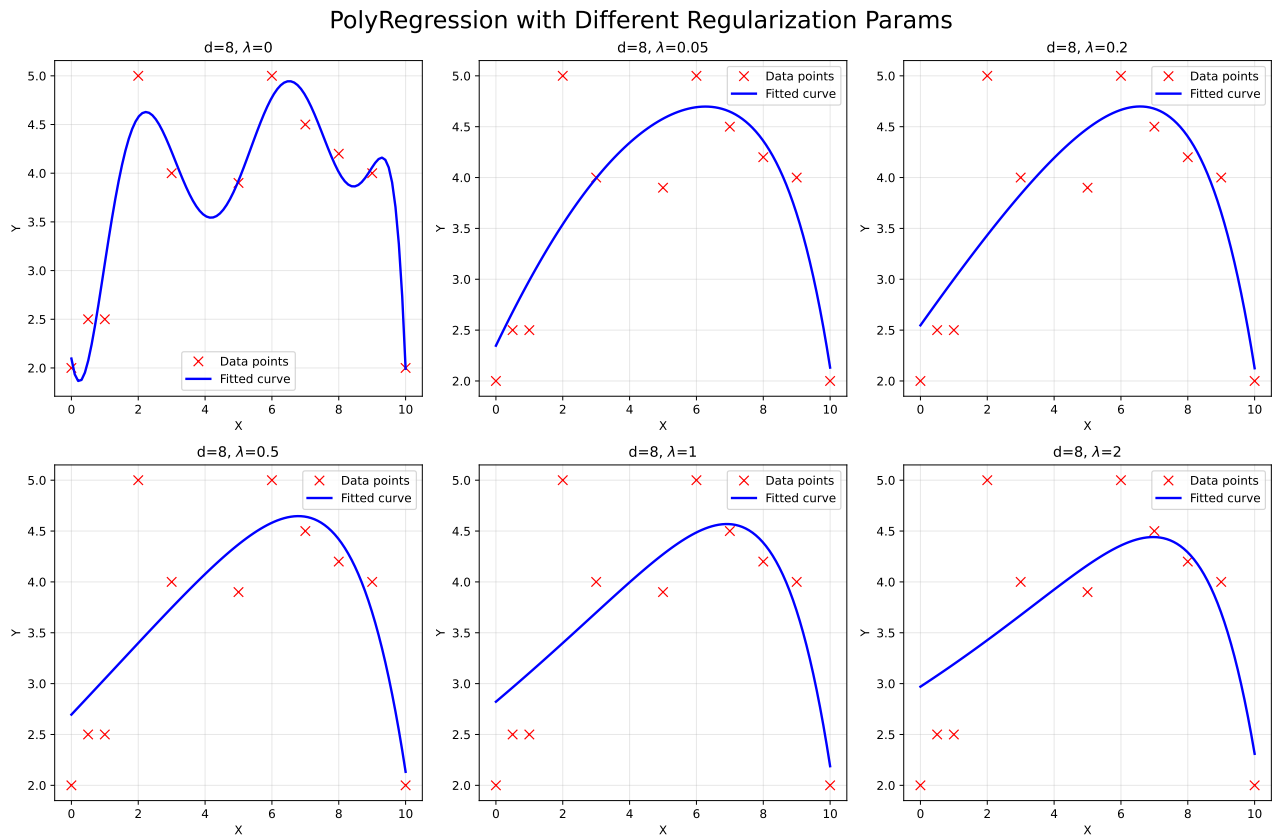


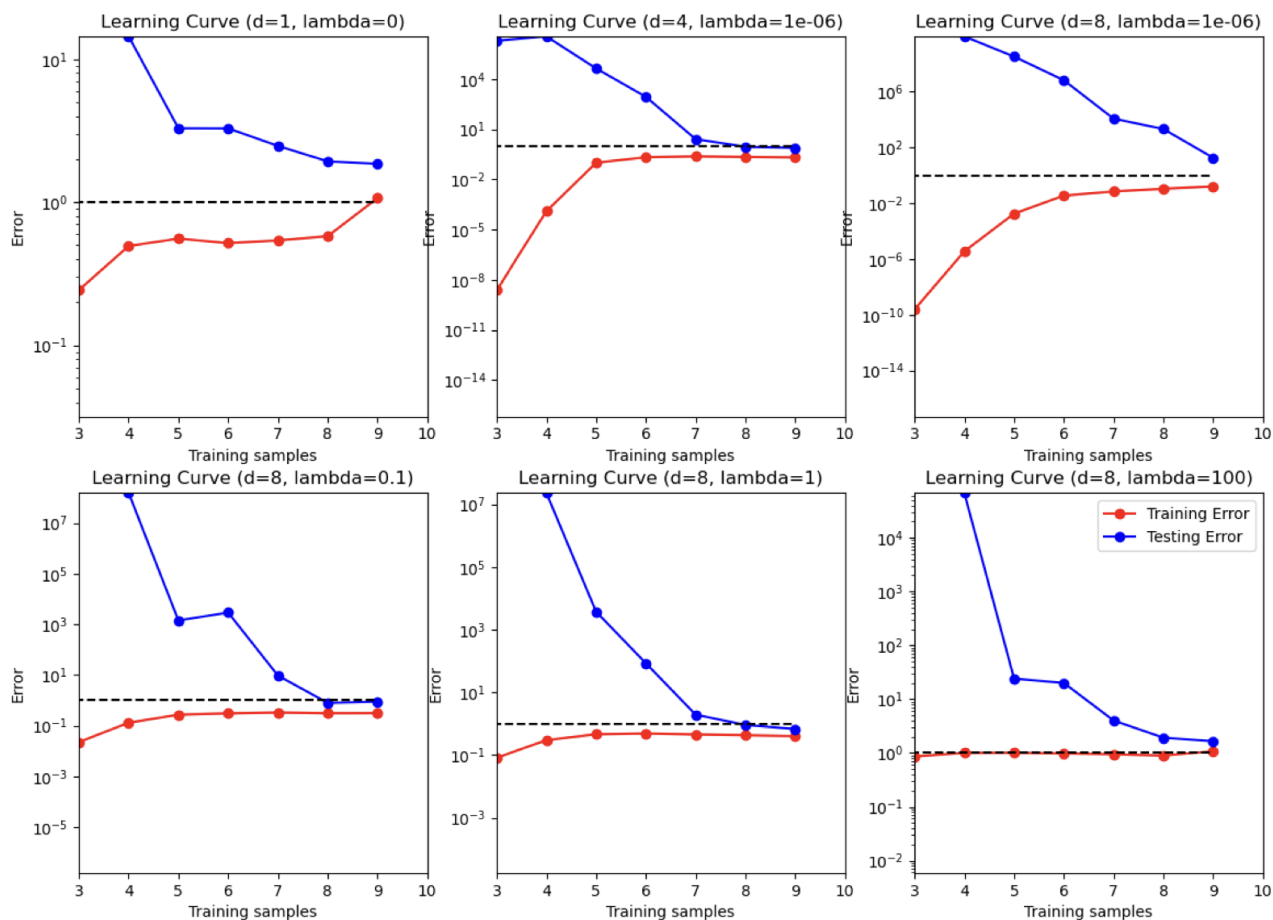
Figure 3: PolyRegression with Different Regularization Parameters

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 λ . 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 λ 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 λ 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 λ via cross-validation to achieve the best bias-variance tradeoff.

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

Plots.

Figure 4 displays the learning curves with different parameters in λ and d .

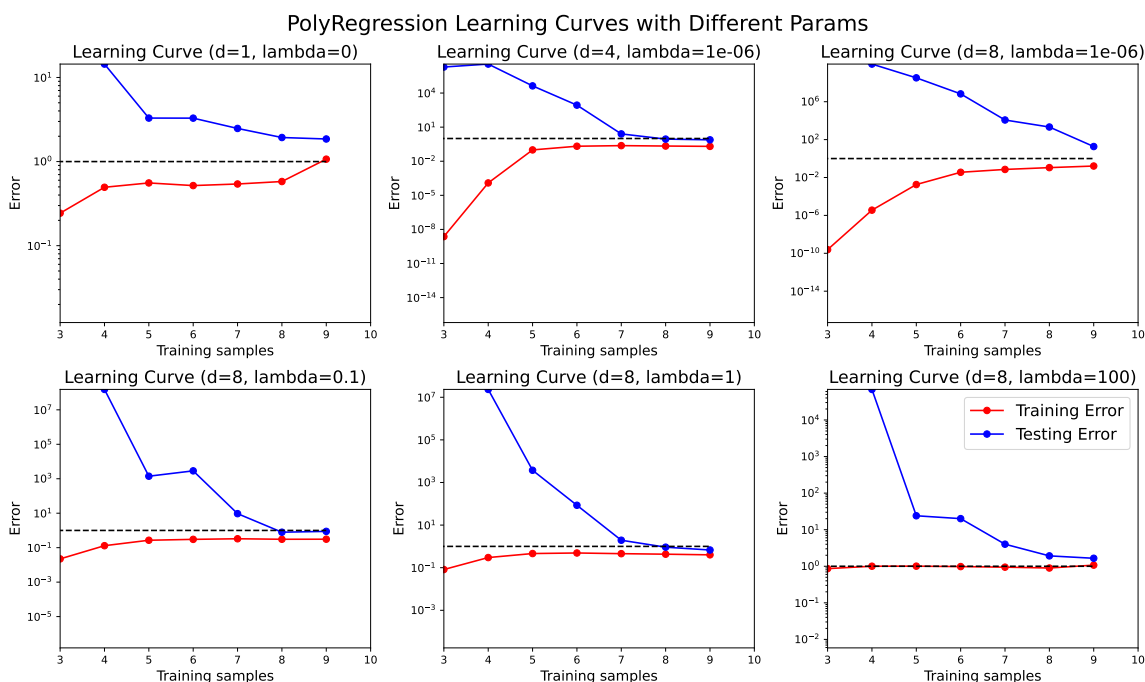


Figure 4: PolyRegression Learning Curves with Different Parameters

Ridge Regression on MNIST

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`).

Each example has features $x_i \in \mathbb{R}^d$ (with $d = 28 * 28 = 784$) and label $z_j \in \{0, \dots, 9\}$. You can visualize a single example x_i with `imshow` after reshaping it to its original 28×28 image shape (and noting that the label z_j is accurate). 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^{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 $W = \begin{bmatrix} w_1 & \dots & w_k \end{bmatrix}$ then

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

where $X = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^T \in \mathbb{R}^{n \times d}$ and $Y = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^T \in \mathbb{R}^{n \times k}$. Show that

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

- b. [10 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 `main` function provided in zip file. **What is the training and testing error?** Note that they should both be about 15%.

What to Submit:

- **Part A:** Derivation of expression for \widehat{W}
- **Part B:** Values of training and testing errors
- **Code** on Gradescope through coding submission

Part A. We want to minimize W with

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

From the derivation of Frobenius norm, we know that

$$\|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2 = \sum_{j=1}^k [\|X w_j - Y e_j\|_2^2 + \lambda \|w_j\|_2^2]$$

Then it becomes that we need to minimize w_j on the right hand side.

Let $b = Y e_j$ (since it has nothing to do with w_j), then the terms inside the sum become

$$\begin{aligned} L(w_j) &= (X w_j - b)^T (X w_j - b) + \lambda w_j^T w_j \\ &= w_j^T X^T X w_j - 2b^T X w_j + b^T b + \lambda w_j^T w_j \\ &= w_j^T (X^T X + \lambda I) w_j - 2(X^T b)^T w_j + b^T b \end{aligned}$$

Now we take the gradient with respect to w_j and set it to zero,

$$\begin{aligned} \nabla_{w_j} L(w_j) &= 2(X^T X + \lambda I) w_j - 2X^T b = 0 \\ \Rightarrow (X^T X + \lambda I) w_j &= X^T b \end{aligned}$$

where we use the fact that $\nabla_w (w^T A w) = 2A w$, $\nabla_w (x^T w) = x$ and that $X^T X + \lambda I$ is symmetric. Then taking the inverse of $X^T X + \lambda I$ on the left of both sides gives

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

Finally, we sum all j from 1 to k , it's the same as stacking columns of the matrix. This gives us

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

That's what we want.

Part B. The values of training and testing errors are:

Ridge Regression Problem

Train Error: 14.805%

Test Error: 14.66%