Homework #1

CSE 446/546: Machine Learning
Professors Matt Golub & Pang Wei Koh
Due: **Monday** October 21, 2024 11:59pm
A: 62 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. 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/cse446/24au/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 the bias-variance tradeoff?

> Solution:
> Bias is the error introduced by approximating a real world problem with a model. It represents the difference between the average prediction of the model and the correct value we're trying to predict. Variance, on the other hand, is the difference between the model's predictions for different training datasets. The bias-variance tradeoff is the concept in ML that deals with a model's ability to appropriately minimize both as much as possible given that as you minimize one, the other increases, and vice versa.

b. *[2 points]* What *typically* happens to bias and variance when the model complexity increases/decreases?

> Solution:
> As we increase model complexity, bias tends to decrease while variance increases, and vice versa. High variance leads to overfitting, and high bias leads to underfitting.

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.

> Solution:
> False. As you collect more training data from the same distribution, the variance of the predictor tends to decrease, not increase. This is because with more training data, the model's ability to generalize increases since the impact each data point has on the model decreases, resulting in a more stable predictor.

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.

> Solution:
> Validation. The validation sets should be used for hyperparameter tuning because the validation sets provides an unbiased assessment of the model's performance, it helps prevent overfitting to the training data, and it preserves the integrity of the test sets. If the test sets were used for tuning, the evaluation of the model would be overly optimistic. A procedure for hyperparameter tuning is Bayesian optimization, which uses a probabilistic model to search for optimal parameters. The core idea is simply to make smart guesses based on what you've learned from previous attempts till you minimize error as much as possible.

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.

> Solution:
> False. The training error of a function on the training set provides an underestimate of the true error of that function, not an overestimate because the model has been specifically optimized to perform well on the training data - so we say that training error is optimistically biased.

What to Submit:

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

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 $x_1, \ldots, x_5$. 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 $x = 0, 1, 2, \ldots$ 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 λ governing the Poisson distribution in terms of goal counts for the first n games: $x_1, \ldots, x_n$. (Hint: remember that the log of the likelihood has the same maximizer as the likelihood function itself.)

Solution:
MLE of λ:

$$\begin{aligned}
\widehat{\lambda}_{MLE} &= arg_\lambda max\; log\, p_\lambda(x_1, ..., x_n) \\
&= arg_\lambda max\; log \prod_{i=1}^{n} p(x_i) \\
&= arg_\lambda max\; \sum_{i=1}^{n} log\, p(x_i) \\
&= arg_\lambda max\; \sum_{i=1}^{n} log[e^{-\lambda}\frac{\lambda^{x_i}}{x_i!}] \\
&= arg_\lambda max\; \sum_{i=1}^{n} (log(\frac{1}{x_i!}) - \lambda) + log(\lambda)\sum_{i=1}^{n} x_i \\
&= arg_\lambda max\; log(\lambda)\sum_{i=1}^{n} x_i - n\lambda \qquad \text{since } \sum_{i=1}^{n}(log(\frac{1}{x_i!}) \text{ is a constant}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial_\lambda}(log(\lambda)\sum_{i=1}^{n} x_i - n\lambda) &= 0 \\
\frac{\sum_{i=1}^{n} x_i}{\lambda} - n &= 0 \\
\sum_{i=1}^{n} x_i &= n\lambda \\
\widehat{\lambda}_{MLE} &= \frac{\sum_{i=1}^{n} x_i}{n}
\end{aligned}$$

b. *[2 points]* Give a numerical estimate of λ after the first five games. Given this λ, what is the probability that the Reign score exactly 4 goals in their next game?

Solution:
$\lambda \approx \frac{3+7+5+0+2}{5} = \frac{17}{5} = 3.4$

$\mathrm{Poi}(4|\frac{17}{5}) = e^{-\frac{17}{5}} \frac{\frac{17}{5}^4}{4!} = 0.18582$

c. *[2 points]* Suppose the Reign actually 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 exactly 5 goals in their 7th game.

Solution:
$\lambda \approx \frac{3+7+5+0+2+8}{6} = \frac{25}{6}$

$\mathrm{Poi}(5|\frac{25}{6}) = e^{-5} \frac{(\frac{25}{6})^5}{5!} = 0.16226$

What to Submit:

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

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:

$$\widehat{\epsilon}_{\text{train}}(f) = \frac{1}{N_{\text{train}}} \sum_{(x,y)\in S_{\text{train}}} (f(x) - y)^2$$

$$\widehat{\epsilon}_{\text{test}}(f) = \frac{1}{N_{\text{test}}} \sum_{(x,y)\in S_{\text{test}}} (f(x) - y)^2$$

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

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

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

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

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

Solution:
We will begin by proving that $\mathbb{E}_{S_{\text{train}}}[\widehat{\epsilon}\text{train}(f)] = \mathbb{E}_{S_{\text{test}}}[\widehat{\epsilon}\text{test}(f)] = \epsilon(f)$ for a fixed f by math.

(a) For $\mathbb{E}_{S_{\text{train}}}[\widehat{\epsilon}\text{train}(f)]$, we have ...

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

(b) Similarly, for $\mathbb{E}_{S_{\text{test}}}[\widehat{\epsilon}\text{test}(f)]$, we have ...

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

Thus, we have $\mathbb{E}_{S_{\text{train}}}[\widehat{\epsilon}\text{train}(f)] = \mathbb{E}_{S_{\text{test}}}[\widehat{\epsilon}\text{test}(f)] = \epsilon(f)$.

For $\widehat{f}$, it is important to note that $\widehat{f}$ is independent of the test set, so we can show a similar proof as the one for a fixed f. As a result, we have ..

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

b. *[3 points]* (bias: the train/dev error) Is the above equation true (in general) with regards to the training loss? Specifically, does $\mathbb{E}_{S_{\text{train}}}[\widehat{\epsilon}\text{train}(\widehat{f})]$ equal $\epsilon(\widehat{f})$? If so, why? If not, give a clear argument as to where your previous argument breaks down.

Solution:
The above equation is not true with regards to the training loss ($\mathbb{E}_{S_{\text{train}}}[\widehat{\epsilon}\text{train}(\widehat{f})] \neq \epsilon(\widehat{f})$) because simply put, $\widehat{f}$ is dependent on the training set since the learned function is derived from training data, making it biased towards the training set. Thus, $\epsilon(\widehat{f})$ would likely be significantly higher than $\mathbb{E}_{S_{\text{train}}}[\widehat{\epsilon}\text{train}(\widehat{f}$.

c. *[5 points]* Let $\mathcal{F} = (f_1, f_2, \dots)$ be a collection of functions and let $\widehat{f}_{\text{train}}$ minimize the training error such that $\widehat{\epsilon}_{\text{train}}(\widehat{f}_{\text{train}}) \leq \widehat{\epsilon}_{\text{train}}(f)$ for all $f \in \mathcal{F}$. Show that

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

(Hint: note that

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

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

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

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

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

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

a. *[5 points]* Intuitively, how do you expect the bias and variance to behave for small values of m? What about large values of m?

b. *[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

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

show that

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

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

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

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[a] 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).[b]

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.

[a] 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

[b] 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[1]:

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

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

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

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 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.

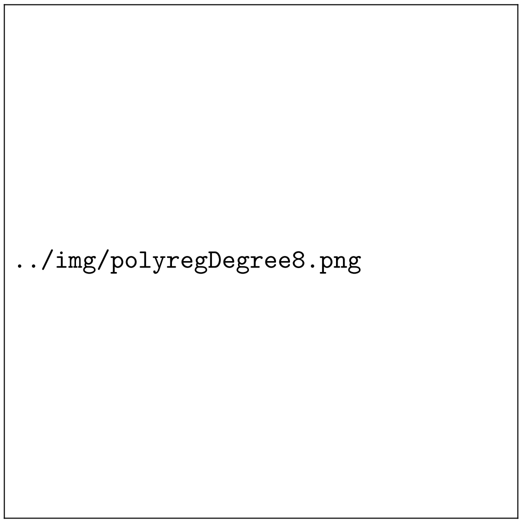


Figure 2: Fit of polynomial regression with $\lambda = 0$ and $d = 8$

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

[1] **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.)

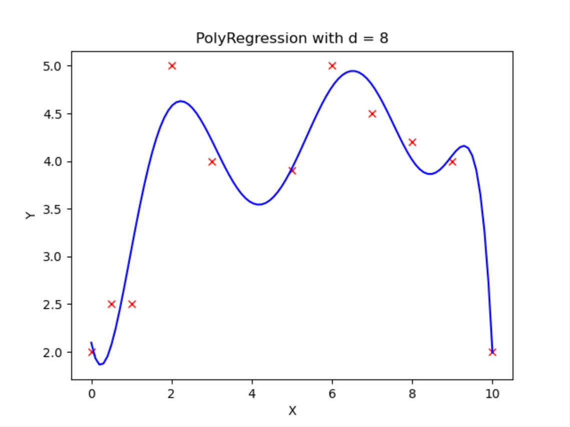
Also, notice that the resulting features will be badly scaled if we use them in raw form. For example, with a polynomial of degree $d = 8$ and $x = 20$, the basis expansion yields $x^1 = 20$ while $x^8 = 2.56 \times 10^{10}$ – an absolutely huge difference in range. Consequently, we will need to standardize 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.

Update: Do **not** standardize the bias terms. Standardizing a column of all ones will result in a variance of zero, which can cause division by zero errors. Instead, first standardize, then add the bias term.

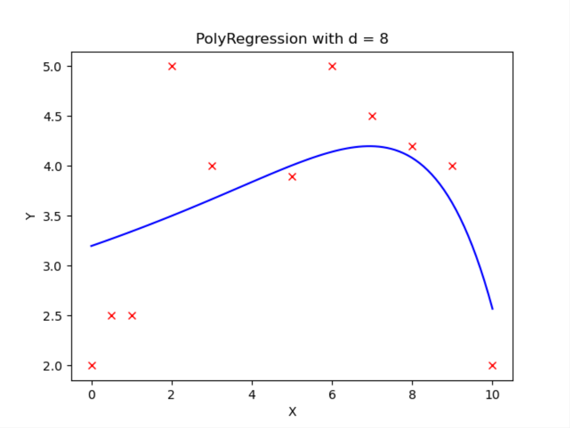
b. *[2 points]* **Run `plot_polyreg_univariate.py` to test your implementation, which will plot the learned function.** In this case, the script fits a polynomial of degree $d = 8$ with no regularization $\lambda = 0$. From the plot, we see that the function fits the data well, but will not generalize 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).

Solution:
Increasing the regularization, decreased the model's accuracy (due to oversimplification), but increased its generalizability. Rather than closely fitting the data, the more regularized graph was now a lot smoother, following the trend of the data.

Before: $\lambda = 0$



After: $\lambda = 5$



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.

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 of each array should return the training error (or testing error) for learning with $i + 1$ training instances. **You should skip calculating the errors for when** $i = 0$, as learning curves are typically displayed starting with two or more instances. Additionally, training your model on a single data point may result in divide-by-zero or singular matrix errors, depending on your implementation. It is a good exercise to identify why (and when) these errors arise!

When computing the learning curves, you should learn on `Xtrain`$[0{:}i]$ for $i = 2, 3, \ldots, \text{numInstances}(\texttt{Xtrain})$, 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

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

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:

../img/polyregLearningCurves.png

Figure 3: Learning curves for various values of d and λ. The blue lines represent the testing error, while the red lines the training error.

Notice the following:

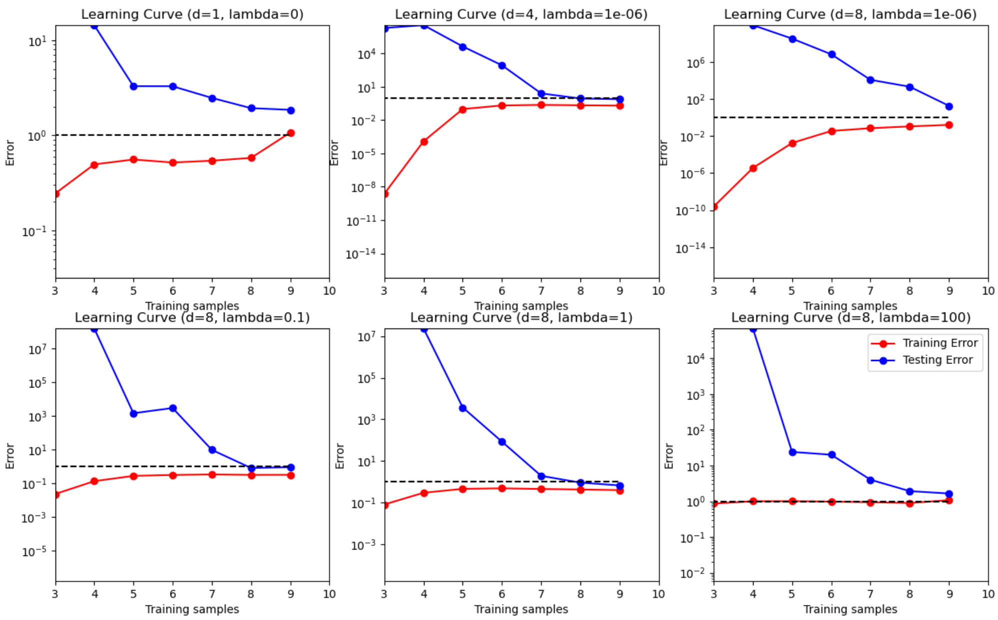
- The y-axis is using a log-scale and the ranges of the y-scale are all different for the plots. The dashed black line indicates the $y = 1$ line as a point of reference between the plots.
- The plot of the unregularized model with $d = 1$ shows poor training error, indicating a high bias (i.e., it is 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.

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.

Solution:



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

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`).

`../img/mnist_images.png`

Figure 4: Sample images from the MNIST data set.

Each example has features $x_i \in \mathbb{R}^d$ (with $d = 28 * 28 = 784$) and label $z_j \in \{0, \ldots, 9\}$. **Since images are represented as 784-dimensional vectors, you can visualize a single example x_i with `imshow` after reshaping it to its original 28×28 image shape** (and noting that the label z_j is accurate). Checkout figure 4 for some sample images. We wish to learn a predictor $\widehat{f}$ that takes as input a vector in $\mathbb{R}^d$ and outputs an index in $\{0, \ldots, 9\}$. We define our training and testing classification error on a predictor f as

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

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

We will use one-hot encoding of the labels: for each observation (x, z), the original label $z \in \{0, \ldots, 9\}$ 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.

Update: Do **not** standardize the data for A5. Normally you would but our tests expect the original data.

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

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

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

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

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

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

Solution:
Using the expression that was derived in the question prompt, we find ...

$$\begin{aligned}
\widehat{W} &= \text{argmin}_{W \in \mathbb{R}^{d\times k}} \sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2 \\
&= \text{argmin}_{W \in \mathbb{R}^{d\times k}} \sum_{j=1}^{k} \left[\|X w_j - Y e_j\|_2^2 + \lambda \|w_j\|_2^2 \right]
\end{aligned}$$

From here, since e_j is the j-th standard basis vector in $\mathbb{R}^k$, the rest follows as ...

$$\begin{aligned}
\text{argmin}_{W \in \mathbb{R}^{d\times k}} \sum_{j=1}^{k} \left[\|X w_j - Y e_j\|_2^2 + \lambda \|w_j\|_2^2 \right] &= \text{argmin}_{W \in \mathbb{R}^{d\times k}} (XW - Y)^T (XW - Y) + \lambda W^T W \\
&= \text{argmin}_{W \in \mathbb{R}^{d\times k}} (W^T X^T X W - 2 Y^T X W + Y^T Y + \lambda W^T W) \\
&= \frac{\partial}{\partial_W} (W^T X^T X W - 2 Y^T X W + Y^T Y + \lambda W^T W) \\
&= 2 X^T X W - 2 X^T Y + 2 \lambda W = 0
\end{aligned}$$

$$(X^T X + \lambda I) W = X^T Y$$

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

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 $Y \in \{0, ..., k-1\}^n$, 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 ith 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 ith example from X'.

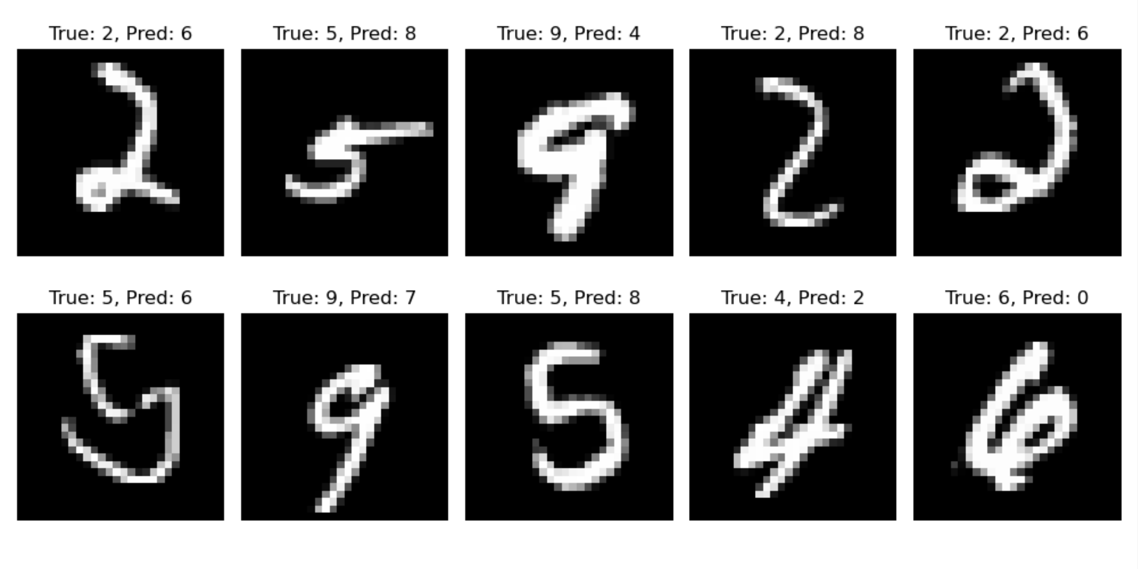
- Using the functions you coded above, train a model to estimate $\widehat{W}$ on the MNIST training data with $\lambda = 10^{-4}$, and make label predictions on the test data. This behavior is implemented in the `main` function provided in a zip file.

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

> Solution:
> Training Error: 14.805%
> Testing Error: 14.66%

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?

Solution:



I noticed a lot of the drawings that were misclassified look similar to multiple numbers or at the least have similar features to other numbers (i.e. curves, structure). There seems to be a significant impact of whether a number has a loop or uses multiple lines as well.

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.

Administrative

A6.

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

Solution:
Around 4-5 hours.