Homework #1

CSE 446/546: Machine Learning
Profs. Jamie Morgenstern and Simon Du
Due: **Wednesday** October 20, 2021 11:59pm
A: 59 points, **B:** 30 points

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

- 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 *[5 points]* . For instructions, see `https://www.gradescope.com/get_started#student-submission`.
- Please recall that B problems, indicated in $\boxed{\text{boxed text}}$, are only graded for 546 students, and that they will be weighted at most 0.2 of your final GPA (see website for details). In Gradescope there is a place to submit solutions to A and B problems separately. You are welcome to create just a single PDF that contains answers to both, submit the same PDF twice, but associate the answers with the individual questions in Gradescope.
- If you collaborate on this homework with others, you must indicate who you worked with on your homework. Failure to do so may result in accusations of plagiarism.
- For every problem involving code, please include the code as part of your PDF for the PDF submission *in addition to* submitting your code to the separate assignment on Gradescope created for code. Not submitting all code files will lead to a deduction of *[1 point]* .

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?

Solution: Bias and variance are both components of the true total error . More precisely, they are part of the learning error. Bias measures how far on average our estimation is from the true parameter or function we are interested in.Variance, on the other hand, gives a measure of variability. If we were to change our training set, it tells us how much we should expect our estimation to change.

As we can see from the explanation above, ideally we would like to have lower bias and lower variance. We want our estimation to equal to the true value on average and we want our estimation to be robust to changes in training set. However, there is a fundamental tradeoff between bias and variance. Lowering bias will lead to an increase in variance and vice versa. For example, if a model uses the training data more then we should expect it to have a lower bias. However, using the training data more means that the variance is higher because the estimation focuses too heavily on the training set and will not generalize well. It can be easily seen that too much focus on training set means that the estimation is not robust to changes in the training set. Therefore, this makes choosing complexity more difficult because the true total error is not monotonic and so we can say that we can simply increase or decrease complexity to reduce true total error.

---

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

Solution:

Typically, (squared) bias decreases and variance increases when the model complexity increases. As the model becomes more complex, it uses the training data more and is able to adapt to more complicated underlying structures. Hence, there is a decrease in bias but an increase in variance. As model complexity decreases, the bias increases and the variance decreases.

---

c. *[1 point]* True or False: A learning algorithm will always generalize better if we use fewer features to represent our data.

Solution:

False. This has to be with the bias variance trade off. The true total error is not a monotonic function of the features. This is because increasing the feature space can increase the complexity of the learned estimator. We know that when the complexity increases (fewer features), the variance decreases but the bias increases. So it is not necessarily true that the the true total error will decrease because the bias increase can be greater than the variance decrease. If we already have a lot of features, then the decrease in variance resulting from decreasing futures can outweigh the increase in bias. However if we have too few features, the decrease in features may lead to a greater increase in bias compared to the decrease in variance resulting in worse generalization. There is going to be an optimal level of features that we should use and if we are below this, a decrease will worsen the generalization. If we are above this, decreasing features will help generalize better.

---

d. *[2 points]* True or False: Hyperparameters should be tuned on the test set. Explain your choice and detail a procedure for hyperparameter tuning.

Solution:

False. The test set should be untouched until model is ready to be applied to it. The test set error would not give us an unbiased estimate of true error otherwise.

A typical approach to selecting a hyperparameter is using cross validation. However, this cannot be done on the test set. We will discuss the procedure of k- fold cross validation because of the computation cost of leave-one-out cross validation. Let λ be the hyperparameter we are interested in tuning. For the k- fold cross validation, we will first divide the training data into k equal parts $D_1, \ldots, D_k$. Then for each i, we will learn classifier $f_{D/Di}$ using data points not in D_i. Estimate error of f_{D/D_i} on validation set D_i:

$$\text{error}_{D_i} = \frac{1}{|D_i|} \sum_{(x_j, y_j) \in D_i} (y_j - f_{D/D_i}(x_j))^2$$

The **k-fold cross validation error** is the average over the data splits:

$$\text{error}_{k-\text{fold}} = \frac{1}{k} \sum_{i=1}^{k} \text{error}_{D_i}$$

We repeat the above procedure for a range of values of λ and then we pick the value of λ that gives the smallest k-fold cross validation error. Even though k-fold cross validation is much faster to compute than leave-one-out validation it is more biased as we are using much less data.

---

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 will be less than the true error, because when calculating the training error the same data is being used to fit the method and assess its error. A fitting method can adapt to the training set, and so training error is an underestimate of the true error which is based on the test set.

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 $x_1, \ldots, x_5$. 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: Let $D = \{x_1, \ldots, x_n\}$. We are interested in λ that maximizes $P(D|\lambda)$. We will denote $P(D|\lambda) = L(D;\lambda)$ as the likelihood function. Then:

$$L(D;\lambda) = \text{Poi}(D|\lambda) = \prod_{i=1}^{n} \text{Poi}(x_i|\lambda)$$

where we have used the assumption of iid data to write it as a product. Recall: the log of the likelihood has the same maximizer as the likelihood function. Therefore, we will find the log of the likelihood function:

$$\log L(D;\lambda) = \log \prod_{i=1}^{n} \text{Poi}(x_i|\lambda) = \sum_{i=1}^{n} \log \text{Poi}(x_i|\lambda)$$

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

$$= \sum_{i=1}^{n}\Big(-\lambda + \log(\lambda^{x_i}) - \log(x_i!)\Big)$$

$$= \sum_{i=1}^{n}\Big(-\lambda + x_i\log(\lambda) - \log(x_i!)\Big)$$

$$= -n\lambda + \log(\lambda)\sum_{i=1}^{n} x_i - \sum_{i=1}^{n}\log(x_i!)$$

Recall: $\lambda_{MLE} = \arg\max_\lambda \log L(D;\lambda)$.

$$\frac{d\log L(D;\lambda)}{d\lambda} = 0$$

$$\implies -n + \frac{\sum_{i=1}^{n} x_i}{\lambda} = 0$$

$$\implies \boxed{\lambda_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i}$$

We can easily check that the second order condition is also satisfied such that λ_{MLE} is truly the $\arg\max$.

$$\frac{d^2 \log L(D;\lambda)}{d\lambda^2} = -\frac{\sum_{i=1}^n x_i}{\lambda^2} \leq 0$$

---

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 6 goals in their next game?

Solution:

From above,

$$\lambda_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

After the first five games, $D = \{2, 4, 6, 0, 1\}$. So, we have:

$$\lambda_{MLE} = \frac{1}{5}\sum_{i=1}^{5} x_i$$

$$= \frac{1}{5}\Big(2+4+6+0+1\Big)$$

$$\implies \boxed{\lambda_{MLE} = \frac{13}{5}}$$

Given this λ, the probability that the Reign score 6 goals in their next game is the following:

$$\text{Poi}(6|\lambda_{MLE}) = e^{-\lambda_{MLE}}\frac{\lambda_{MLE}^6}{6!}$$

$$= e^{-\frac{13}{5}} \times \left(\frac{13}{5}\right)^6 \times \frac{1}{720}$$

$$= \boxed{0.0319}.$$

---

c. *[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.

Solution:

Now, $D = \{2, 4, 6, 0, 1, 8\}$. So,

$$\lambda_{MLE} = \frac{1}{6}\sum_{i=1}^{6} x_i$$

$$= \frac{1}{6}\Big(2 + 4 + 6 + 0 + 1 + 8\Big)$$

$$= \frac{21}{6}$$

$$\implies \boxed{\lambda_{MLE} = \frac{7}{2}}$$

$$\text{Poi}(6|\lambda_{MLE}) = e^{-\lambda_{MLE}}\frac{\lambda_{MLE}^6}{6!}$$

$$= e^{-\frac{7}{2}} \times \left(\frac{7}{2}\right)^6 \times \frac{1}{720}$$

$$= \boxed{0.0771}.$$

A3. *[10 points]* *(Optional Background)* In World War 2, the Allies attempted to estimate the total number of tanks the Germans had manufactured by looking at the serial numbers of the German tanks they had destroyed. The idea was that if there were n total tanks with serial numbers $\{1, \dots, n\}$ then its reasonable to expect the observed serial numbers of the destroyed tanks constituted a uniform random sample (without replacement) from this set. The exact maximum likelihood estimator for this so-called *German tank problem* is non-trivial and quite challenging to work out (try it!). For our homework, we will consider a much easier problem with a similar flavor.

Let $x_1, \dots, x_n$ be independent, uniformly distributed on the continuous domain $[0, \theta]$ for some θ. What is the Maximum likelihood estimate for θ?

Solution: Let $x_1, \dots, x_n$ be independent, uniformly distributed on the continuous domain $[0, \theta]$ for some θ. The pdf of the uniform distribution is given by:

$$f(x|\theta) = \begin{cases} \frac{1}{\theta} & \text{if } 0 \le x \le \theta \\ 0 & \text{otherwise} \end{cases}$$

In this example, $D = \{x_1, \dots, x_n\}$. Then, we can write the likelihood function as the following:

$$L(D; \theta) = f(D|\theta) = f(x_1, \dots, x_n|\theta)$$

Recall:

$$\theta_{MLE} = \arg\max_{\theta} L(D; \theta) = \arg\max_{\theta} \log L(D; \theta)$$

Using the fact that $x_1, \dots, x_n$ are iid,

$$L(D; \theta) = f(x_1|\theta) \dots f(x_n|\theta)$$

Notice: If for at least one x_i it is the case that $x_i > \theta$, then $L(D; \theta) = 0$. When calculating MLE we have choice over θ to maximize the likelihood and so we can always pick θ such that $x_i \le \theta \; \forall \; i$ and this will give us a strictly positive likelihood. Thus,

$$L(D; \theta) = f(x_1|\theta) \dots f(x_n|\theta) = \frac{1}{\theta^n}$$

Then:

$$\log L(D; \theta) = -\log(\theta^n) = -n \log(\theta)$$

Differentiating with respect to θ:

$$\frac{d \log L(D; \theta)}{d\theta} = -\frac{n}{\theta} < 0$$

So, the likelihood function is decreasing in θ. Therefore, we need to pick the smallest θ possible to maximize the likelihood function. The smallest feasible θ is going to be the maximum out of the x_is because any θ less than that will make the likelihood function equal to zero. This is because if $x_{\max} > \theta$ then $f(x_{\max}|\theta) = 0$. Thus,

$$\boxed{\theta_{MLE} = \max\{x_1, \dots, x_n\}}.$$

Notice: this is consistent with our initial restriction that $x_i \le \theta \; \forall \; i$.

Overfitting

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 (for example, using linear least squares regression) using the training set to obtain $\widehat{f}$.

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

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

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:

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

Solution: We want to show the following:

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

Notice:

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

Then:

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

since $\{(x_i, y_i)\}_{i=1}^n$ drawn i.i.d from an underlying distribution $\mathcal{D}$, the (x_j, y_j)s in the training set must also be drawn from $\mathcal{D}$. f is fixed and so it is independent of the training set. Thus,

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

Similarly,

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

since $\{(x_i, y_i)\}_{i=1}^n$ drawn i.i.d from an underlying distribution $\mathcal{D}$, the (x_j, y_j)s in the test set must also be drawn from $\mathcal{D}.f$ is fixed and so it is independent of the test set Then:

$$\begin{aligned}
\mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(f)] &= \frac{1}{N_{\text{train}}} \times N_{\text{train}} \times \epsilon(f) \\
&= \epsilon(f)
\end{aligned}$$

Thus:

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

Since, $\widehat{f}$ is obtained using the training set it is independent of the test set. Therefore, from the point of view of the test set, $\widehat{f}$ is fixed. We had already shown above that for all fixed f,

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

Given that $\widehat{f}$ is fixed for the test set, the above result gives us:

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

---

b. *[4 points]* (bias: the train/dev error) Is the above equation true (in general) with regards to the training loss? Specifically, does $\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(\widehat{f})]$ equal $\epsilon(\widehat{f})$? If so, why? If not, give a clear argument as to where your previous argument breaks down.

Solution: No, the above equation is not true in general, that is, $\mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f})]$ might not equal $\epsilon(\widehat{f})$. We used the assumption that $\widehat{f}$ is obtained using only the training set so $\widehat{f}$ is independent of our test and can be considered as fixed from its point of view. If for some reason we use the test set to obtain $\widehat{f}$ this will no longer hold true. The test set will not be independent of $\widehat{f}$ and cannot be considered fixed. If we obtain $\widehat{f}$ using the test then the test error will not be an unbiased estimate of our true error for $\widehat{f}$.

---

c. *[8 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

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

(Hint: note that

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

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

Solution: We want to show that:

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

The hint shows that:

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

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

$$\begin{aligned}
\mathbb{E}_{\text{train,test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f}_{\text{train}})] &= \sum_{f\in\mathcal{F}} \mathbb{E}_{\text{test}}[\widehat{\epsilon}_{\text{test}}(f)]\mathbb{P}_{\text{train}}(\widehat{f}_{\text{train}} = f) \\
&= \sum_{f\in\mathcal{F}} \mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(f)]\mathbb{P}_{\text{train}}(\widehat{f}_{\text{train}} = f) \qquad \text{(using part a of this question)}
\end{aligned}$$

We know that $\widehat{\epsilon}_{\text{train}}(\widehat{f}_{\text{train}}) \leq \widehat{\epsilon}_{\text{train}}(f)$ for all $f \in \mathcal{F}$ which implies:

$$\mathbb{E}_{\text{train}}[\widehat{\epsilon}_{\text{train}}(\widehat{f}_{\text{train}})] \leq E_{\text{train}}[\widehat{\epsilon}_{\text{train}}(f)]$$

Plugging this back into $\mathbb{E}_{\text{train,test}}[\widehat{\epsilon}_{\text{test}}(\widehat{f}_{\text{train}})]$:

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

where the last equality uses the fact that $\sum_{f\in\mathcal{F}} \mathbb{P}_{\text{train}}(\widehat{f}_{\text{train}} = f) = 1$. Thus, we have:

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

Polynomial Regression

Relevant Files[1]

- **polyreg.py**
- `linreg_closedform.py`
- `test_polyreg_univariate.py`
- `test_polyreg_learningCurve.py`
- `data/polydata.dat`

A4. *[10 points]* 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.

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.

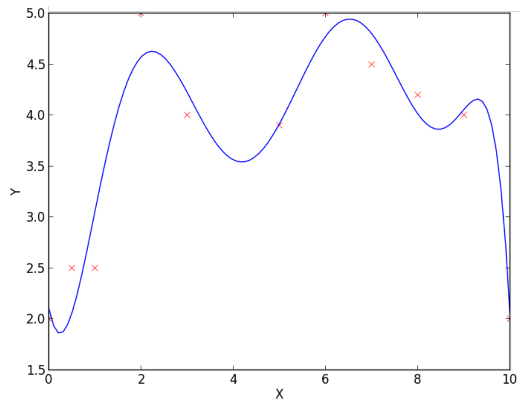


Figure 1: Fit of polynomial regression with $\lambda = 0$ and $d = 8$

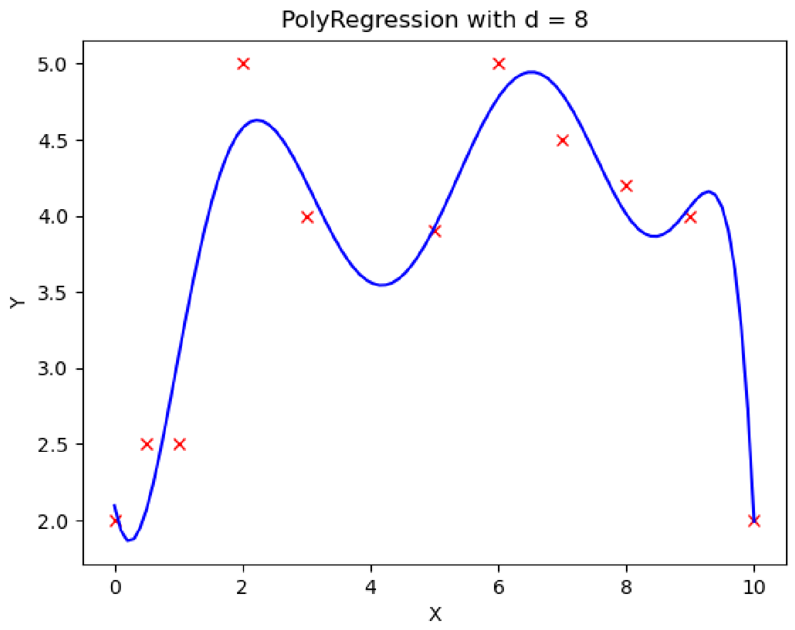
Run `test_polyreg_univariate.py` to test your implementation, which will plot the learned function. In this case, the script fits a polynomial of degree $d = 8$ with no regularization $\lambda = 0$. From the plot, we see that the function fits the data well, but will not generalize well to new data points. Try increasing the amount

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

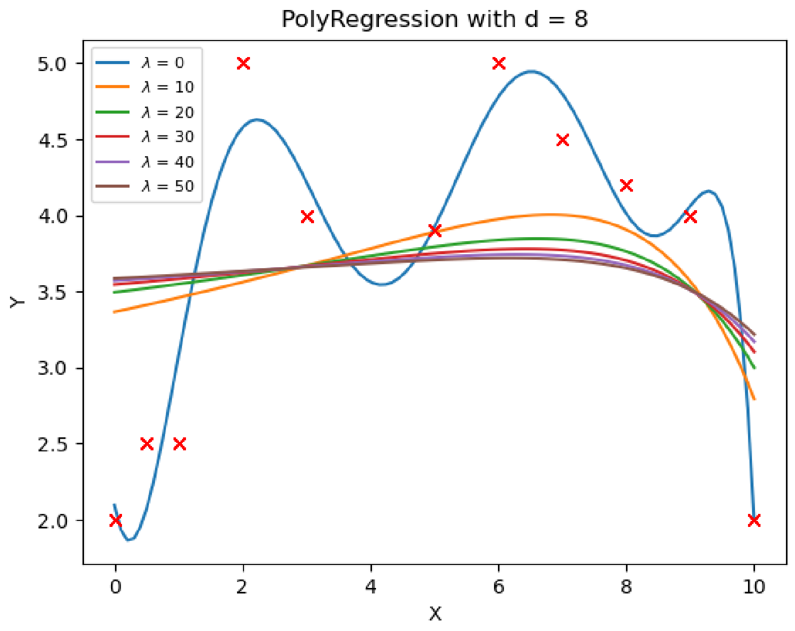
of regularization, and in 1-2 sentences, describe the resulting effect on the function (you may also provide an additional plot to support your analysis).

Solution:

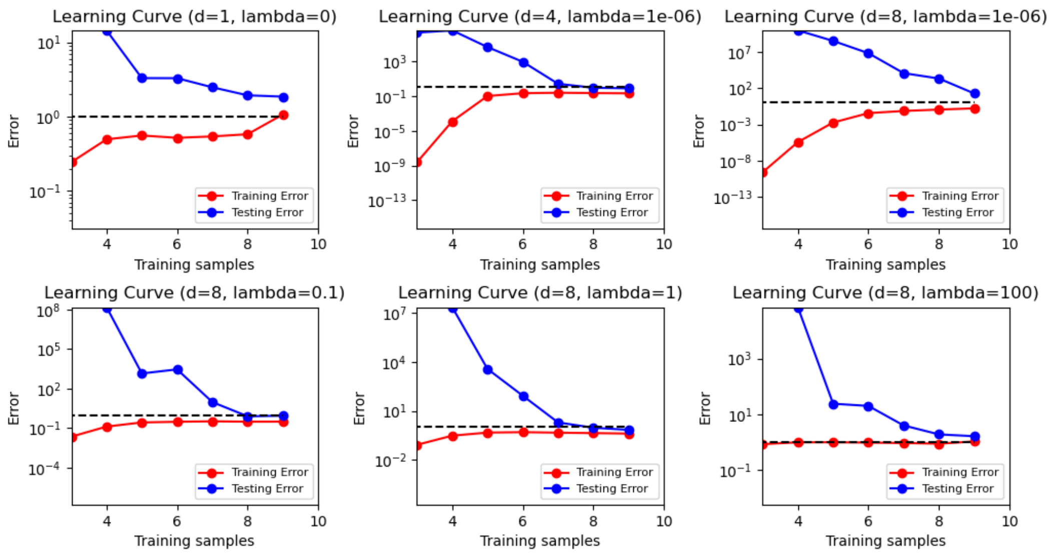
Before Regularization ($\lambda = 0$):



To find the effect of regularization, we plotted the function for a number of values of λ. The following figure demonstrates the effect of regularization:



We know that the point of regularization is to penalize large coefficient values. If we increase the penalty, we should expect to see smaller and smaller magnitudes of the weights. Note that the offset term is not regularized, since this just affects the height of the function, not its complexity. So increasing λ results in smoother functions. The point of the regularization is to make the function simpler and if we let $\lambda \to \infty$ we should get a straight line at the offset term which is the simplest function, corresponding to a constant.
The following figure contains the generated learning curves:



Administrative

A5.

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

   7-8 hours