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 c-e:** True or False
- **Parts a-e:** Brief (2-3 sentence) explanation

Answer:
Part a:
Bias is the difference between the average predictions and true values

Variance is the variability of our predictions i.e. how spread out the model predictions are

The goal of a machine learning algorithm is usually to achieve low bias and low variance, which results in a good prediction performance. Bias and variance are inversely proportional to each other. To find the the optimal algorithm to the solution, a trade off needs to be made between the error introduced by the bias and the variance.

Part b:
As model complexity increase, variance increase and bias decreases

As model complexity decrease, variance decreases and bias increases

Part c:
False. Having fewer features may lead to underfitting which is the scenario of high bias and low variance.

Part d:
False. The test set should only be used for testing. The dataset is split into train, validation and test sets. A model is trained on the training set and the hyperparameters of the model are then tuned on the validation set. The test set is then evaluated to determine the final performance score of the model.

Part e:
False. The model is trained on the training set which is the same data that the training error obtained from therefore, it is usually lower than the true error.

Maximum Likelihood Estimation (MLE)

A2. You're the Reign FC manager, and the team is five games into its 2021 season. The 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

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

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.)

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?

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.

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.

Answers:
Part a:

$$\begin{aligned}
Poi(x|\lambda) &= e^{-\lambda}\frac{\lambda^x}{x!} \\
Ł(\lambda|x_1, \ldots, x_n) &= \prod_{i=1}^{n} Poi(x_i|\lambda) \\
&= \prod_{i=1}^{n} e^{-\lambda}\frac{\lambda^{x_i}}{x_i!} \\
log Ł(\lambda|x_1, \ldots, x_n) &= \sum_{i=1}^{n} \log(e^{-\lambda}\frac{\lambda^{x_i}}{x_i!}) \\
&= \sum_{i=1}^{n}(-\lambda + x_i \log\lambda - \log(x_i!)) \\
\log Ł(\lambda|x_i, \ldots, x_n) &\propto \sum_{i=1}^{n}(-\lambda + x_i \log\lambda) \\
&= -n\lambda + \sum_{i=1}^{n} x_i \log\lambda
\end{aligned}$$

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\lambda}\log Ł(\lambda|x_i, \ldots, x_n) &= -n + \frac{\sum_{i=1}^{n} x_i}{\lambda} \overset{\text{set}}{=} 0 \\
\implies \widehat{\lambda} &= \frac{\sum_{i=1}^{n} x_i}{n}
\end{aligned}$$

Part b:

After first 5 games,

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

$$\begin{aligned}\mathbb{P}(x_6 = 6) &= e^{-2.6}\frac{2.6^6}{6!} \\ &\approx 0.0319\end{aligned}$$

Part c:

After 6 games,

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

$$\begin{aligned}\mathbb{P}(X_7 = 6) &= e^{-3.5}\frac{3.5^6}{6!} \\ &\approx 0.0771\end{aligned}$$

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 $\{1, \ldots, n\}$ 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 $x_1, \ldots, x_n$ be independent, uniformly distributed on the continuous domain $[0, \theta]$ for some θ. What is the Maximum likelihood estimate for θ?

What to Submit:

- An expression for the MLE of θ after n games and relevant derivation.

Answer:

$$f(x|\theta) = \begin{cases} \frac{1}{\theta} & 0 \leq x \leq \theta \\ 0 & elsewhere \end{cases}$$

Let $x_1 \leq x_2 \cdots \leq x_n$. Then the likelihood function is given by:

$$\begin{aligned} Ł(\theta|x) &= \prod_{i=1}^{n} \frac{1}{\theta} \\ &= \theta^{-n} \end{aligned}$$

$$\text{subject to } 0 \leq x_1 \text{ and } \theta \geq x_n \ \ ,0 \ \ elsewhere$$

$$\begin{aligned} &\frac{\mathrm{d}}{\mathrm{d}\theta} Ł(\theta|x) = -\frac{n}{\theta} < 0 \\ \implies &Ł(\theta|x) = \theta^{-n} \text{is a decreasing function for } \theta_n \end{aligned}$$

$\therefore Ł(\theta|x)$is maximised at $\theta = x_n$ MLE of θ is given by

$$\widehat{\theta} = x_n$$

Since $\widehat{\epsilon}_{train}(\widehat{f}_{train}) \leq \widehat{\epsilon}_{train}(f)$,

$$\begin{aligned}
\mathbb{E}_{train}[\widehat{\epsilon}_{train}(\widehat{f}_{train})] &= \sum_{f \in \mathbb{F}} \mathbb{E}_{train}[\widehat{\epsilon}_{train}(\widehat{f}_{train}|f)]\mathbb{P}_{train}(\widehat{f}_{train} = f) \\
&\leq \sum_{f \in \mathbb{F}} \mathbb{E}_{train}[\widehat{\epsilon}_{train}(f|f)]\mathbb{P}_{train}(\widehat{f}_{train} = f) \\
&\leq \sum_{f \in \mathbb{F}} \mathbb{E}_{train}[\widehat{\epsilon}_{train}]\mathbb{P}_{train}(\widehat{f}_{train} = f)
\end{aligned}$$

Since we know that f is fixed from part a $\implies \mathbb{E}_{train}[\widehat{\epsilon}_{train}(\widehat{f}_{train})] = \mathbb{E}_{test}[\widehat{\epsilon}_{test}(\widehat{f}_{train})]$,

$$\begin{aligned}
\therefore \mathbb{E}_{train}[\widehat{\epsilon}_{train}(\widehat{f}_{train})] &\leq \sum_{f \in \mathbb{F}} \mathbb{E}_{train}[\widehat{\epsilon}_{train}]\mathbb{P}_{train}(\widehat{f}_{train} = f) \\
&\leq \sum_{f \in \mathbb{F}} \mathbb{E}_{test}[\widehat{\epsilon}_{test}]\mathbb{P}_{train}(\widehat{f}_{train} = f) \\
&\leq \mathbb{E}_{train,test}[\widehat{\epsilon}_{test}(\widehat{f}_{train})] \quad \square
\end{aligned}$$

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.

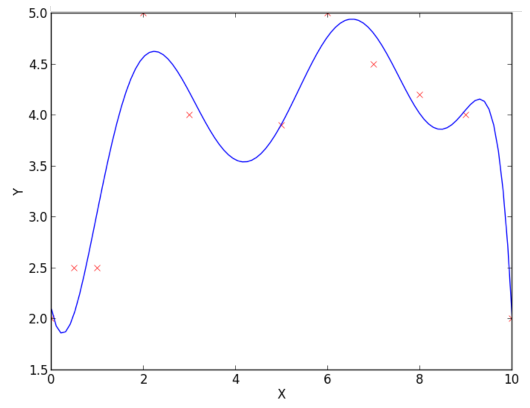


Figure 1: Fit of polynomial regression with $\lambda = 0$ and $d = 8$

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 of regularization, and in 1-2 sentences, describe the resulting effect on the function (you may also provide an

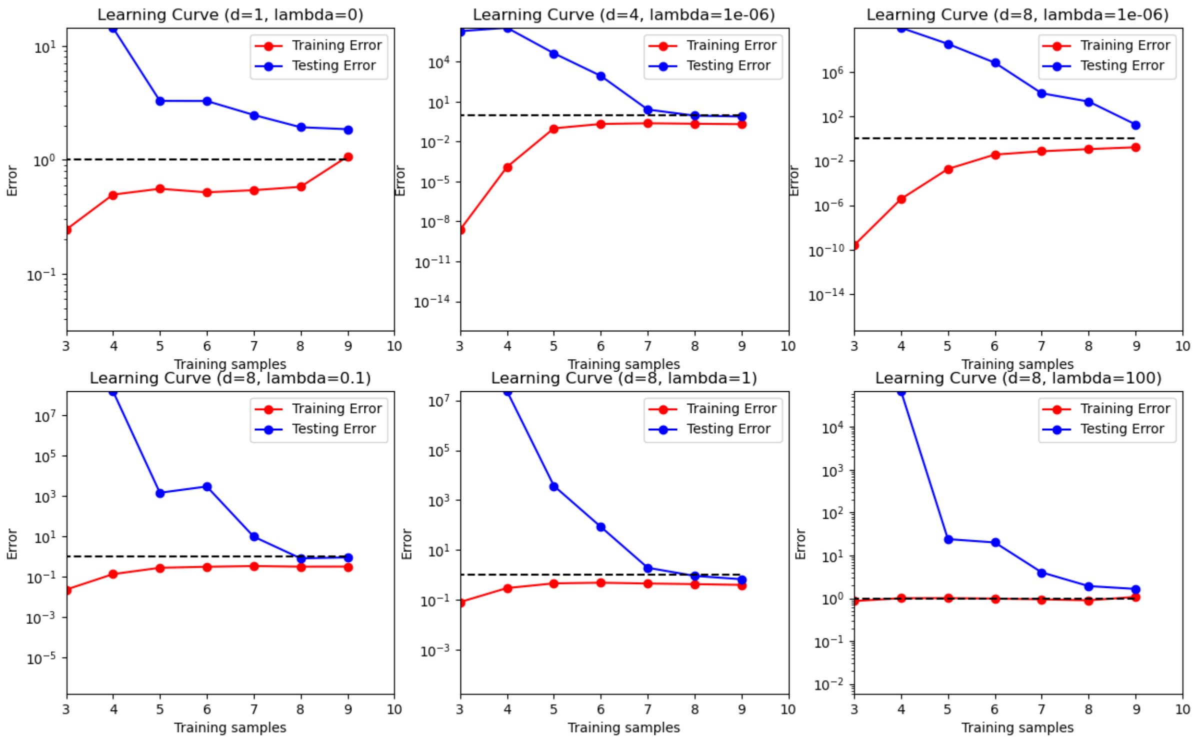
[1]**Bold text** indicates files or functions that you will need to complete; you should not need to modify any of the other files.

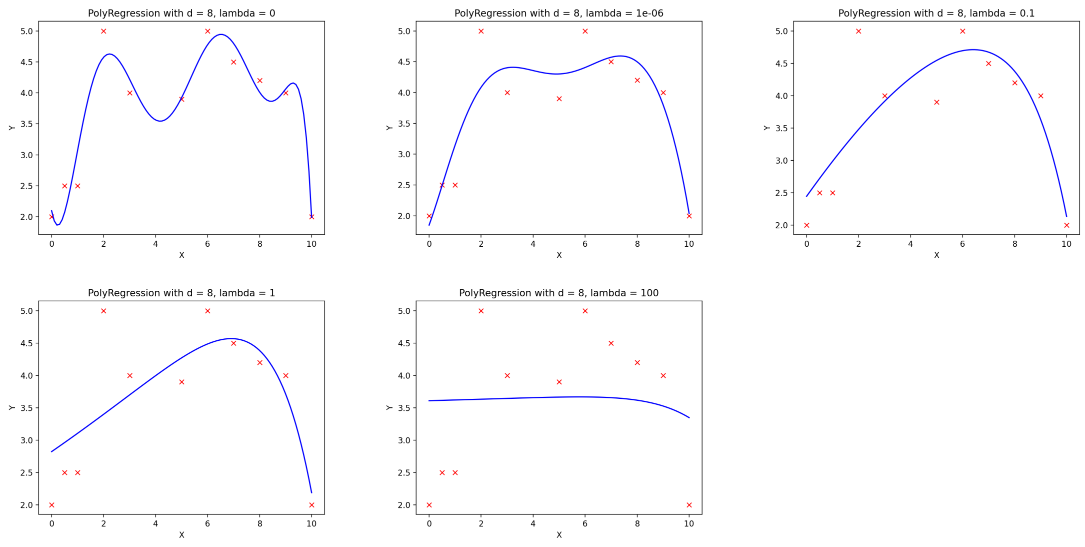
additional plot to support your analysis).

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

Answers:
Increasing regularization flattens the curve and leads to higher training error





Administrative

A5.

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

I spent about 20-30hrs working on it spread out over multiple days