Homework #1

CSE 446/546: Machine Learning
Profs. Kevin Jamieson and Ludwig Schmidt
Due: Friday 22nd April, 2022 11:59pm
60 points(A)/30 points(B)

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 10% of the points for each problem improperly tagged**. For instructions, see `https://www.gradescope.com/get_started#student-submission`.
- 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.
- **The coding problems are available in a .zip file on the course website**, with some starter code. All coding questions in this class will have starter code. **Before attempting these problems make sure your coding environment is working**. See instructions in README file in the .zip file.
- 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 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. *[2 points]* True or False: Suppose you're given a fixed learning algorithm. If you collect more training data from the same distribution, the variance of your predictor increases.

d. *[2 points]* Suppose that we are given train, validation and test sets. Which of these sets should be used for hyperparameter tuning? Explain your choice and detail a procedure for hyperparameter tuning.

e. *[1 point]* True or False: The training error of a function on the training set provides an overestimate of the true error of that function.

What to Submit:

- a). Bias is the error between the expected value and true value.
  Variance represents the dispersion of data. Higher variance shows the data are more spread from its mean.
  Bias-variance tradeoff, increase bias can reduce variance and vice versa. So, find a balance point that minimize the sum of bias and variance.
- b). When the model complexity increases, bias goes down and variance goes up. When the model complexity decrease, bias goes up and variance goes down.
- c). True, more train data will reduce learning error, which included both bias and variance.
- d). Never train on test data. Divide train data into pieces and use cross validation to find hyperparameter.
- e). False, the model may over fitted.

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.)

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:** Likelihood function: $L_n(\lambda) = \Pi_{i=1}^n e^{-\lambda}\frac{\lambda^{x_i}}{x_i!}$
  Log-Likelihood function: $l_n(\lambda) = Log(L_n(\lambda)) = \sum_{i=1}^n log(e^{-\lambda}\frac{\lambda^{x_i}}{x_i!}) = \sum_{i=1}^n(\lambda - x_i log(\lambda) + log(x_i!))$
  Set $\frac{d}{d\lambda}l_n(\lambda) = 0 = \sum_{i=1}^n(1 - \frac{x_i}{\lambda})$
  $\hat{\lambda}_{MLE} = \frac{1}{n}\sum_{i=1}^n x_i$
- **Parts b:** $\hat{\lambda}_{MLE} = \frac{1}{5}\sum_{i=1}^5 x_i = \frac{13}{5}$
  $Poi(6|2.6) = 3.19\%$
- **Parts c:** $\hat{\lambda}_{MLE} = \frac{1}{6}\sum_{i=1}^6 x_i = \frac{21}{6}$
  $Poi(6|3.5) = 7.71\%$

Polynomial Regression

Relevant Files[1]

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

A3. *[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.

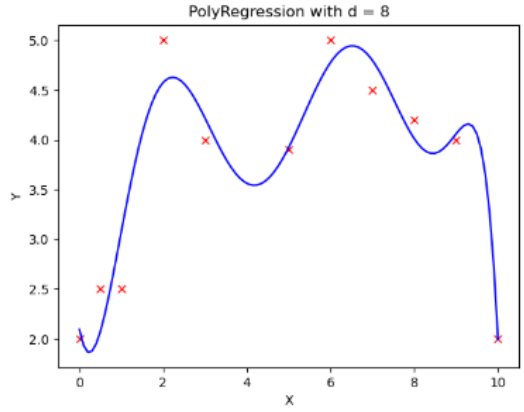
Implement regularized polynomial regression in `polyreg.py`. You may implement it however you like, using gradient descent or a closed-form solution. However, I would recommend the closed-form solution since the data sets are small; for this reason, we've included an example closed-form implementation 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.

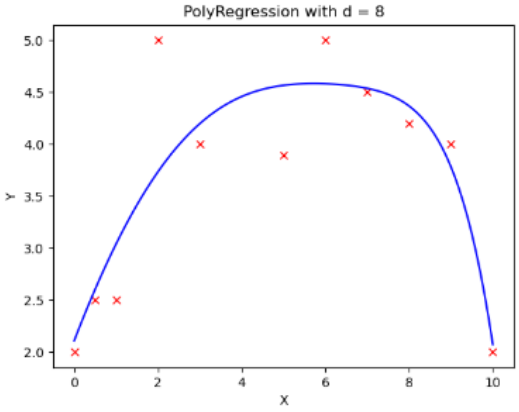
Run `test_polyreg_univariate.py` to test your implementation, which will plot the learned function. In this case, the script fits a polynomial of degree $d = 8$ with no regularization $\lambda = 0$. From the plot, we see that the function fits the data well, but will not generalize well to new data points.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

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

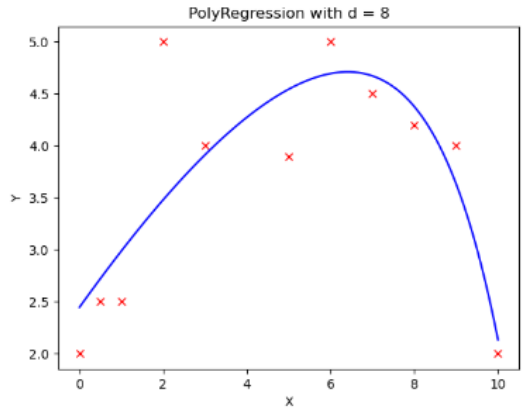
What to Submit:



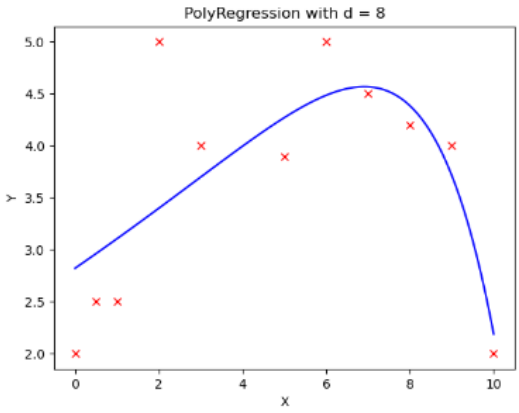
$\lambda = 0$



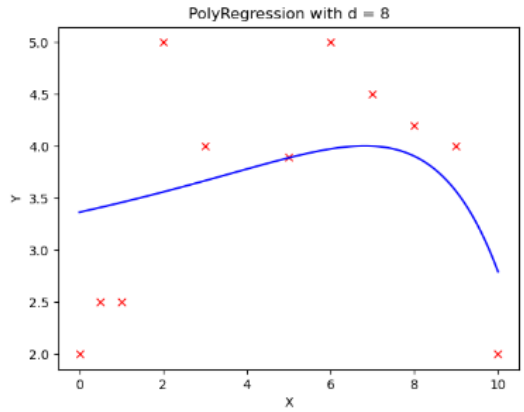
$\lambda = 0.01$



$\lambda = 0.1$



$\lambda = 1$



$\lambda = 10$

As λ increasing, the polynomial regression function becomes flat and fit the data bad.

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 $i = 1, \ldots,$ numInstances(`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

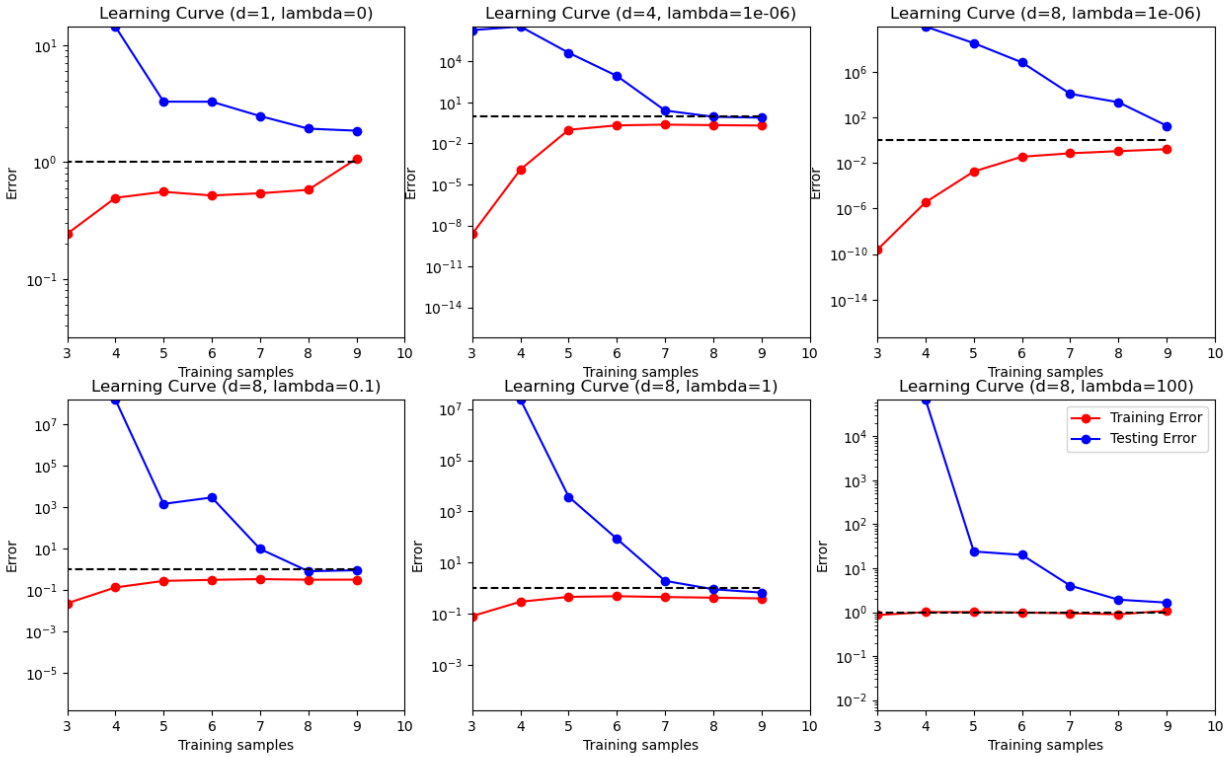
$$\frac{1}{n}\sum_{i=1}^{n}(h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2 \quad . \tag{2}$$

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:



Ridge Regression on MNIST

Relevant Files[2]:

- **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`).

Each example has features $x_i \in \mathbb{R}^d$ (with $d = 28 * 28 = 784$) and label $z_j \in \{0, \ldots, 9\}$. 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). Checkout figure **??** 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.

a. *[10 points]* In this problem we will choose 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 $\arg\max_{j=0,\ldots,9} e_{j+1}^T \widehat{W}^T x_i$. Note that if $W = \begin{bmatrix} w_1 & \ldots & w_k \end{bmatrix}$ then

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

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

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

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

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 $Y \in \{0, ..., k-1\}^n$, 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 $\arg\max_{j=0,...,9} e_j^T W^T x_i'$ 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 `main` function provided in zip file. **What is the training and testing error?** Note that they should both be about 15%.

Once you finish this problem question, you should have a very powerful classifier for handwritten digits! 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 $\widehat{W}$
  The question given us:

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

  And

$$\begin{aligned} \frac{\delta}{\delta w_j} \sum_{j=1}^{k} \left[\|X w_j - Y e_j\|^2 + \lambda \|w_j\|^2 \right] &= \frac{\delta}{\delta w_j} \sum_{j=1}^{k} \left[(X w_j - Y e_j)^T (X w_j - Y e_j) + \lambda w_j^T w_j \right] \\ &= 2 \sum_{j=1}^{k} \left[w_j^T (X^T X + \lambda I) - y_i^T X \right] \\ &= \sum_{j=1}^{k} \left[(X^T X + \lambda I) w_j - X^T y_i \right]^T \\ &= 0 \end{aligned}$$

  Thus,

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

- **Part B:** Values of training and testing errors
  Train Error: 14.805%
  Test Error: 14.66%
- **Code** on Gradescope through coding submission

Administrative

A6.

a. *[2 points]* About how many hours did you spend on this homework? There is no right or wrong answer :)
5-6 hours, I dropped this class before, and I retake it this quarter.