

# Homework #1

CSEP 546: Machine Learning

Prof. Byron Boots

Due: **Monday** November 1st, 2021 11:59pm

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.
- Please typeset your submission in a pdf file. Failure to do so may result in a points deduction.
- 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](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.
- 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

1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

- a. *[2 points]* 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:** True or False with brief (2-3 sentence) explanation

## Maximum Likelihood Estimation (MLE)

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

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

Let’s call these scores  $x_1, \dots, x_5$ . Based on your (assumed iid) data, you’d like to build a model to understand how many goals the Reign are likely to score in their next game. You decide to model the number of goals scored per game using a *Poisson distribution*. Recall that the Poisson distribution with parameter  $\lambda$  assigns every non-negative integer  $x = 0, 1, 2, \dots$  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  $\lambda$  governing the Poisson distribution in terms of goal counts for the first  $n$  games:  $x_1, \dots, x_n$ . (Hint: remember that the log of the likelihood has the same maximizer as the likelihood function itself.)
- b. *[2 points]* Give a numerical estimate of  $\lambda$  after the first five games. Given this  $\lambda$ , what is the probability that the Reign score 6 goals in their next game?

**What to Submit:**

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

## Polynomial Regression

Relevant Files<sup>1</sup>

---

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

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

3. [10 points] Recall that polynomial regression learns a function  $h_{\theta}(x) = \theta_0 + \theta_1x + \theta_2x^2 + \dots + \theta_dx^d$ , where  $d$  represents the polynomial's highest degree. We can equivalently write this in the form of a linear model with  $d$  features

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

using the basis expansion that  $\phi_j(x) = x^j$ . Notice that, with this basis expansion, we obtain a linear model where the features are various powers of the single univariate  $x$ . We're still solving a linear regression problem, but are fitting a polynomial function of the input.

Implement regularized polynomial regression in `polyreg.py`. You may implement it however you like, using gradient descent or a closed-form solution. However, I would recommend the closed-form solution since the data sets are small; for this reason, we've included an example closed-form implementation of linear regression in `linreg_closedform.py`. Please 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  $\lambda$
- `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. In `fit()`, after you perform the polynomial feature expansion, you should standardize the values with the same degrees by subtracting their mean and divide the result with their standard deviation. You'll need to apply the same standardization transformation in `predict()` before you apply it to new data. After standardization, be sure to add bias term before the column with lowest degree (i.e. add a column of ones to the left most of resulting matrix). You can achieve this with `pad()` function in `numpy`.

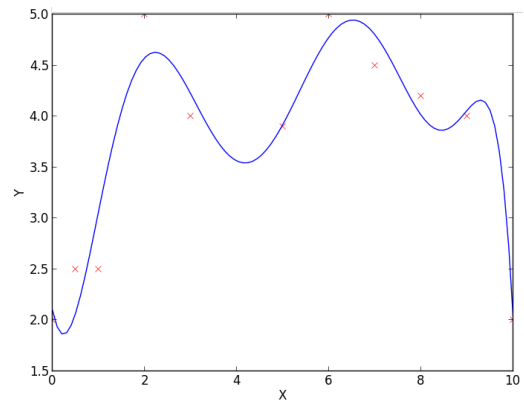


Figure 1: Fit of polynomial regression with  $\lambda = 0$  and  $d = 8$

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 additional plot to support your analysis).

## What to Submit:

- **Write-up:** The description of resulting effect of increasing regularization.
- **Code:** Implement all functions with NotImplementedError in `polreg.py` and submit on on Gradescope

## Ridge Regression on MNIST

4. In this problem we will implement a regularized least squares classifier for the MNIST data set. The task is to classify handwritten images of numbers between 0 to 9.

You are **NOT** allowed to use any of the pre-built classifiers in `sklearn`. Feel free to use any method from `numpy` or `scipy`. **Remember:** if you are inverting a matrix in your code, you are probably doing something wrong (Hint: look at `scipy.linalg.solve`).

Each example has features  $x_i \in \mathbb{R}^d$  (with  $d = 28 * 28 = 784$ ) and label  $z_j \in \{0, \dots, 9\}$ . You can visualize a single example  $x_i$  with `imshow` after reshaping it to its original  $28 \times 28$  image shape (and noting that the label  $z_j$  is accurate). We wish to learn a predictor  $\hat{f}$  that takes as input a vector in  $\mathbb{R}^d$  and outputs an index in  $\{0, \dots, 9\}$ . We define our training and testing classification error on a predictor  $f$  as

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

We will use one-hot encoding of the labels: for each observation  $(x, z)$ , the original label  $z \in \{0, \dots, 9\}$  is mapped to the standard basis vector  $e_{z+1}$  where  $e_i$  is a vector of size  $k$  containing all zeros except for a 1 in the  $i^{\text{th}}$  position (positions in these vectors are indexed starting at one, hence the  $z + 1$  offset for the digit labels). We adopt the notation where we have  $n$  data points in our training objective with features  $x_i \in \mathbb{R}^d$  and label one-hot encoded as  $y_i \in \{0, 1\}^k$ . Here,  $k = 10$  since there are 10 digits.

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

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

Note that  $\|W\|_F$  corresponds to the Frobenius norm of  $W$ , i.e.  $\|W\|_F^2 = \sum_{i=1}^d \sum_{j=1}^k W_{i,j}^2$ . To classify a point  $x_i$  we will use the rule  $\operatorname{arg} \max_{j=0, \dots, 9} e_{j+1}^T \widehat{W}^T x_i$ . Note that if  $W = [w_1 \ \dots \ w_k]$  then

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

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

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

- b. *[10 points]*

- Implement a function `train` that takes as input  $X \in \mathbb{R}^{n \times d}$ ,  $Y \in \{0, 1\}^{n \times k}$ ,  $\lambda > 0$  and returns  $\widehat{W} \in \mathbb{R}^{d \times k}$ .
- Implement a function `one_hot` that takes as input  $Y \in \{0, \dots, k-1\}^n$ , and returns  $Y \in \{0, 1\}^{n \times k}$ .
- Implement a function `predict` that takes as input  $W \in \mathbb{R}^{d \times k}$ ,  $X' \in \mathbb{R}^{m \times d}$  and returns an  $m$ -length vector with the  $i$ th entry equal to  $\arg \max_{j=0, \dots, k-1} e_j^T W^T x'_i$  where  $x'_i \in \mathbb{R}^d$  is a column vector representing the  $i$ th example from  $X'$ .
- Using the functions you coded above, train a model to estimate  $\widehat{W}$  on the MNIST training data with  $\lambda = 10^{-4}$ , and make label predictions on the test data. This behavior is implemented in `main` function provided in zip file. **What is the training and testing error?** Note that they should both be about 15%.

#### What to Submit:

- **Part A:** Derivation of expression for  $\widehat{W}$
- **Part B:** Values of training and testing errors
- **Code** Implement all functions with `NotImplementedError` in `ridge_regression.py` and submit on on Gradescope

5. [10 points] We just fit a classifier that was linear in the pixel intensities to the MNIST data. For classification of digits the raw pixel values are very, very bad features: it's pretty hard to separate digits with linear functions in pixel space. The standard solution to this is to come up with some transform  $h : \mathbb{R}^d \rightarrow \mathbb{R}^p$  of the original pixel values such that the transformed points are (more easily) linearly separable. In this problem, you'll use the feature transform:

$$h(x) = \cos(Gx + b).$$

where  $G \in \mathbb{R}^{p \times d}$ ,  $b \in \mathbb{R}^p$ , and the cosine function is applied elementwise. We'll choose  $G$  to be a *random* matrix, with each entry sampled i.i.d. from a Gaussian with mean  $\mu = 0$  and variance  $\sigma^2 = 0.1$ , and  $b$  to be a random vector sampled i.i.d. from the uniform distribution on  $[0, 2\pi]$ . The big question is: *how do we choose  $p$ ?* Using cross-validation, of course!

Randomly partition your training set into proportions 80/20 to use as a new training set and validation set, respectively. Using the `train` function you wrote before, train a  $\widehat{W}^p$  for different values of  $p$  and plot the classification training error and validation error on a single plot with  $p$  on the  $x$ -axis. Be careful, your computer may run out of memory and slow to a crawl if  $p$  is too large ( $p \leq 6000$  should fit into 4 GB of memory that is a minimum for most computers, but if you're having trouble you can set  $p$  in the several hundreds). You can use the same value of  $\lambda$  as in corresponding A problem but feel free to study the effect of using different values of  $\lambda$  and  $\sigma^2$  for fun.

#### What to Submit:

- **Write-up:** Plot of train, validation, and testing errors as a function of  $p$ .
- **Code** Implement all functions with `NotImplementedError` in `ridge_regression_cos.py` and submit on on Gradescope

## Administrative

6. [2 points] About how many hours did you spend on this homework? There is no right or wrong answer :)