CIS 419/519 Applied Machine Learning
Homework 2

Due: February 12, 2019 11:59pm

Instructions

Read all instructions in this section thoroughly.

Collaboration: Make certain that you understand the collaboration policy described on the course website.

You should feel free to talk to other members of the class in doing the homework. I am more concerned that you learn how to solve the problem than that you demonstrate that you solved it entirely on your own. You may *discuss* the homework to understand the problems and the mathematics behind the various learning algorithms, but you must **write down your solutions yourself**. In particular, **you are not allowed to share problem solutions or your code with any other students.**

We take plagiarism and cheating very seriously, and will be using automatic checking software to detect academic dishonesty, so please don't do it.

You are also prohibited from posting any part of your solution to the internet, even after the course is complete. Similarly, please don't post this PDF file or the homework skeleton code to the internet.

Formatting: This assignment consists of two parts: a problem set and program exercises.

For the problem set, you must write up your solutions electronically and submit it as a single PDF document. We will not accept handwritten or paper copies of the homework. Your problem set solutions must use proper mathematical formatting.

All solutions must be typeset using LaTeX; handwritten solutions of any part of the assignment (including figures or drawings) are not allowed.

Your solutions to the programming exercises must be implemented in python, following the precise instructions included in Part 2. Portions of the programing exercise will be graded automatically, so it is imperative that your code follows the specified API. A few parts of the programming exercise asks you to create plots or describe results; these should be included in the same PDF document that you create for the problem set.

Homework Template and Files to Get You Started: The programming exercises are based around the following ipython notebook: https://www.seas.upenn.edu/~cis519/spring2020/homework/hw2_skeleton.ipynb.

Citing Your Sources: Any sources of help that you consult while completing this assignment (other students, textbooks, websites, etc.) ***MUST*** be noted in the your PDF file. This includes anyone you briefly discussed the homework with. If you received help from the following sources, you do not need to cite it: course instructor, course teaching assistants, course lecture notes, course textbooks or other readings.

Submitting Your Solution: You will submit this assignment through **both** Gradescope and Canvas; submission will open approximately one week before the due date.

Both the PDF with your written responses and your python code must be submitted to **both** Canvas and Gradescope. Why do we require that you submit to both sites? Gradescope has better automatic unit testing functionality, and Canvas allows us to run plagiarism detection software. Unfortunately this means that we need you to submit the assignment in two places so we can get the best of both worlds.

All of the files should be uploaded directly; there should not be any directory structure.

CIS 519 ONLY Problems: Some parts of the assignment will be marked as "[CIS 519 ONLY]". Only students enrolled in CIS 519 are required to complete these problems. However, we do encourage students in CIS 419 to read through these problems, although you are not required to complete them.

All homeworks will receive a percentage grade, but CIS 519 homeworks will be graded out of a different total number of points than CIS 419 homeworks. Students in CIS 419 choosing to complete CIS 519 ONLY exercises will not receive any credit for answers to these questions (i.e., they will not count as extra credit nor will they compensate for points lost on other problems).

Philosophy Behind This Homework In lecture recently, we have focused on the fundamentals of machine learning with basic models defined by a cost function and fit via gradient descent. Before we return to more applied work, it's essential to understand how these learning algorithms can be implemented, and so we'll be focusing on this aspect over the next two assignments, covering several ML algorithms. Once finished, you'll know in detail how to implement cost functions, gradient descent with convergence tests, basis expansion, regularization, and the effect of varying parameter choices and learning configurations (e.g., different types of regularization) on the resulting models. Establishing these foundations now will allow us to have later homeworks focus much more on applications.

PART I: PROBLEM SET

Your solutions to the problems will be submitted as a single PDF document. Be certain that your problems are well-numbered and that it is clear what your answers are.

1 Gradient Descent (10 pts)

Let k be a counter for the iterations of gradient descent, and let α_k be the learning rate for the k^{th} step of gradient descent.

a.) In one sentence, what are the implications of using a constant value for α_k in gradient descent?

b.) In another sentence, what are the implications for setting α_k as a function of k?

2 Linear Regression (10 pts) [CIS 519 ONLY]

[Adapted from an exercise by Hastie, et al.]

Suppose we have a sample of n pairs (x_i, y_i) drawn i.i.d. from the following distribution:
$x_i \in X$, the set of instances. Each $x_i \in \mathbb{R}$
$y_i = f(x_i) + \epsilon_i$, where $f()$ is the regression function
$\epsilon_i \sim G(0, \sigma^2)$, a Gaussian with mean 0 and variance σ^2
We can construct an estimator for $f()$ that is *linear* in the in the *labels* (the y_i's) instead of the features:

$$f(x) = \sum_{i=1}^{n} l_i(x; X) y_i \ , \tag{1}$$

where the weights $l_i(x; X)$ do not depend on the y_i's, but do depend on the entire training set X. In other words, we are re-writing $f()$ so that it is some weighted sum of the training labels (the y_i's), and those weights $l_i(x; X)$ are based on some function of the x we're predicting and the entire training set X.

Show that linear regression is a member of this class of estimators by explicitly describing the weights $l_i(x_0; X)$ for linear regression (i.e., give an equation for $l_i()$).

(Hint: this will involve manipulating the equations we covered in lecture to transform them into the form of the estimator above.)

PART II: PROGRAMMING EXERCISES

4 Polynomial Regression (35 pts)

In this exercise, you will modify a linear regression implementation that we provide to fit a polynomial model and explore the bias/variance tradeoff. This exercise is based around the ipython notebook linked in the instructions.

4.1 Implementing Polynomial Regression (25 pts)

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

$$h_{\boldsymbol{\theta}}(x) = \theta_0 \phi_0(x) + \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \ldots + \theta_d \phi_d(x) \ , \tag{2}$$

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 the `PolynomialRegression` class. You must implement it using gradient descent, building on top of an example linear regression implementation that we have

provided, `LinearRegression`. (Make CERTAIN that you understand this implementation, since you'll need to change several lines of it. Note that this implementation just runs gradient descent for a set number of iterations; it does not test for convergence. You can run the test script `test_linreg()` to train an example linear regression model.) Your implementation of polynomial regression should stop gradient descent when it converges.

Your implementation of polynomial regression must follow the API below. Note that all matrices are actually Pandas data frames 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 until it converges
- `predict(X)`: method to use the trained polynomial regression model for prediction
- `polyfeatures(X, degree)`: expands the given $n \times 1$ data frame X into an $n \times d$ data frame of polynomial features of degree d. Note that the returned data frame 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 $n \times 1$ Pandas data frame ($X \in \mathbb{R}^{n \times 1}$) and this function will return the polynomial expansion of this data, a $n \times d$ data frame. 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 data frame returned by `polyfeatures()`, this allows the `polyfeatures` function to be more general, so it could be applied to multivariate 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. You may not use the sklearn utilities for data standardization; you must do implement the computations and transformations yourself.

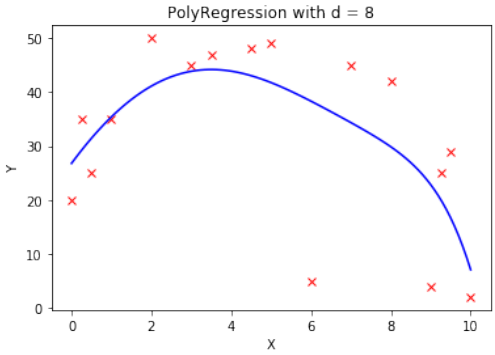

Figure 1: Fit of polynomial regression with $\lambda = 0$ and $d = 8$

Run `test_polyreg_univariate()` 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 $\lambda = 0$. Try increasing the amount of regularization, and examine the resulting effect on the function. You will need to adjust your gradient descent learning rate and convergence test until gradient descent works well.

4.2 Choosing the optimal λ (10 pts)

We can use cross-validation to choose the optimal λ. Modify your `PolynomialRegression` to include this functionality by allowing `__init()` to take in an optional boolean argument `tuneLambda` and an optional array `regLambdaValues`. For example,

$$\texttt{regLambdaValues} = [0, 0.001, 0.003, 0.006, 0.01, 0.03, 0.006, 0.1, 0.3, 0.6, 1, 3, 10]$$

When `tuneLambda` is true, the `fit()` should tune `self.regLambda` automatically to one of the values in `regLambdaValues`. You may have your implementation use any method you like to tune lambda, and you may use any of the sklearn tuning or cross-validation utilities in your implementation; you are not required to implement these yourself.

Rerun the earlier test function or write your own to explore the effectiveness of automatically tuning lambda. Include a pair of plots of the univariate data in your PDF writeup; one showing the fit before regularization and one showing the fit after auto-tuning regularization.