6140: Programming Assignment 1

Essentially all problems in machine learning involve some form of optimization, whether to fit a model to data (estimation) or to select a prediction based on a model (decision). In a few cases we can find the optimum analytically; in many others we do it numerically. In this assignment we will begin to explore numerical methods for some of the basic estimation problems that we have been learning about.

All programming assignments must be completed in **Python**. You need to make sure your code is **runnable** before you submit it. We will grade your solution based on both **coding style (comment, naming convention, etc.)** and **code**, so make sure you have a clear explanation of your code. **Last, you should not discuss your solutions with other students. Cheating and other acts of academic dishonesty will be referred to OSCCR (office of student conduct and conflict resolution) and the College of Computer Science.**

To submit your solutions, you can log onto the ccs github server, and look for the corresponding homework repository. We will post a guideline of using github in piazza later. Your code solution should be saved in a **single .py** file following the order of the questions. Don't forget to create a **README** file to explain your answer to all non-coding problems. Put **README** and **.py** file in a folder named by your **full name (First name_Last name)** and upload it.

1 Implement Gradient Descent

The simplest and most commonly referred-to method of optimization is gradient descent (see Murphy 8.3.2 and Wikipidia). It's worth understanding its behavior on actual problems. There are many much more sophisticated optimization methods available in Python; you should also benchmark one of those.

1. Implement a basic gradient descent procedure to minimize scalar functions of a vector argument. Write it generically, so that you can easily specify the objective function and the function to compute the gradient. You should be able to specify the initial guess, the step size and the convergence criterion (a threshold such that the algorithm terminates when the objective values on two successive steps is below this value).

2. Test your gradient descent procedure on some functions whose optimal value you know, e.g. a quadratic bowl or the (negative of) Gaussian. Make sure that you try functions of more than one variable and that you try at least one convex function and one very non-convex function

with multiple minima. Discuss the effect of the choice of starting guess, the step size, and the convergence criterion on the resulting solution.

3. Write code to approximate the gradient of a function numerically at a given point using central differences (see the "Finite difference" article in Wikipedia). Verify its behavior on the functions you used in the question above by comparing the analytic and numerical gradients at various points.

4. Compare the behavior of your gradient descent procedure with one the more sophisticated optimizers available in `scipy.optimize` (e.g. `fmin_bfgs`). A good metric for comparison is the number of iterations required to reach convergence. You should instrument your code to keep track of function calls and call Scipy optimizers so that they print this.

2 Linear Basis Function Regression

Let's consider the linear combination of basis function class of regression models. We know how to get analytic solutions for the maximum likelihood weight vector (see Murphy 7.3). We'll use this problem to "benchmark" the gradient descent solution.

We have provided you with a text file (`curvefitting.txt`) that has the 10 data points that were used to generate the plots in Appendix Figure 1. We have also given you code to read this data and some code illustrating how to generate plots, both in Python.

1. Write a procedure for computing the maximum likelihood weight vector given (a) an array of 1-dimensional data points X, (b) a vector of Y values, and (c) the value of M, the maximum order of a simple polynomial basis $\phi_1(x) = x, \ldots, \phi_M(x) = x^M$. Test your solution by replicating the plots in the Appendix Figure 1 and the weight values in Table 2. You should be able to get very close agreement.

2. Now, write functions to compute the sum of squares error (SSE) (and its derivative) for a data set and a hypothesis, specified by the list of M polynomial basis functions and a weight vector. Verify your gradient using the numerical derivative code.

3. Use gradient descent on the SSE function to replicate the graphs in Appendix. Describe your experience with initial guesses, step sizes and convergence thresholds. Compare to using one of the more sophisticated optimizers. Explain your results in terms of the properties of the function being optimized and the properties of the algorithms.

3 Ridge Regression

1. Implement ridge regression both analytically and via gradient descent (See Murphy 7.5). Experiment with different values of λ on the simple data from Appendix Figure 1, for various

values of M. Describe your observations.

2. We have given you three additional data sets: there are two training data sets and one validation set. In general, we want to use the training data to optimize parameters and then use the performance of those parameters on the validation data to choose among models (e.g. values of M and values of λ), this is called "model selection". Show some values of M and λ, and show the effect on the fit in the test and validation data. Which values work best for each of the training data sets? Explain.

4 Appendix

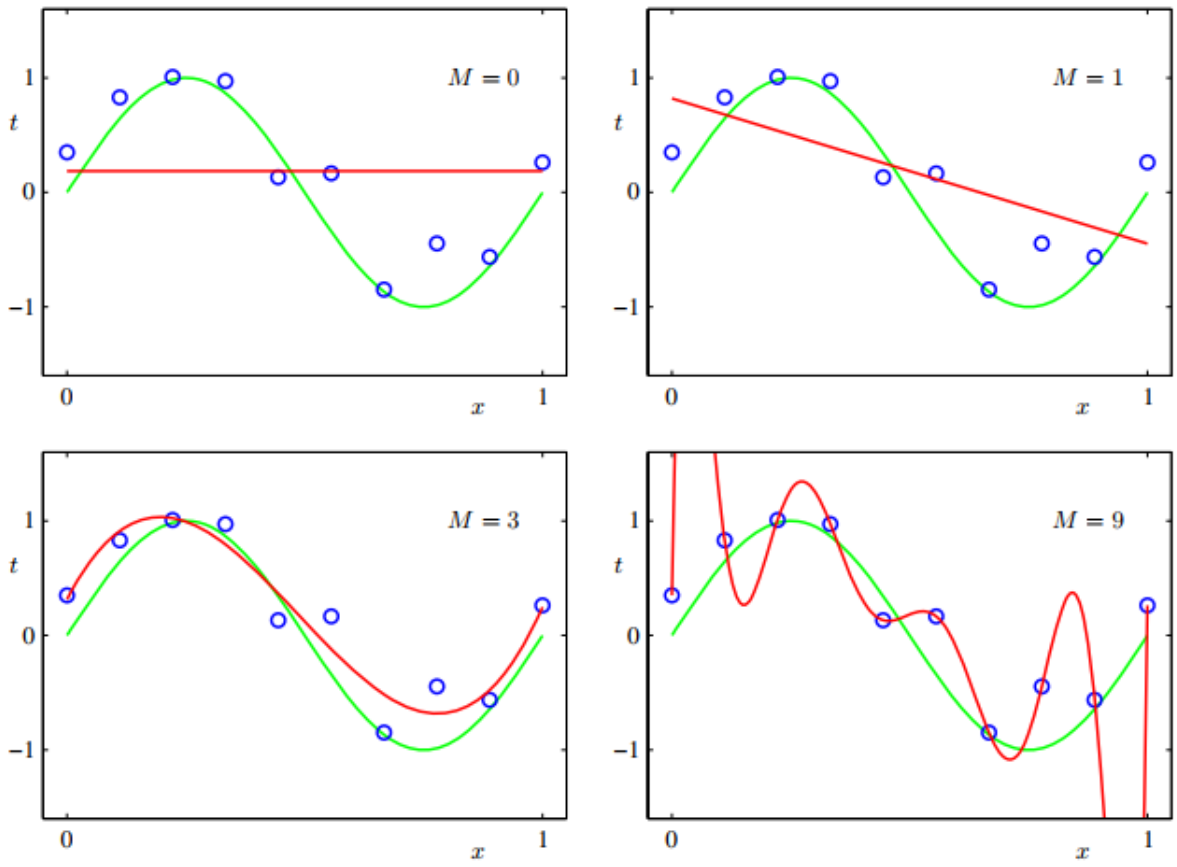

Figure 1: Plots of polynomials having various orders *M*, shown as red curves. (borrow from Bishop's book Figure 1.4)

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Figure 2: Table of the coefficients W for polynomials of various order. Observe how the typical magnitude of the coefficients increases dramatically as the order of the polynomial increases. (borrow from Bishop's book Table 1.1)