6.867: Homework 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.

You will find a zip file with some useful code and data in the Stellar Materials page. You can do these assignments in any computational system that you are used to. We recommend Matlab or the Pylab/Numpy/Scipy/Matplotlib cluster of packages and we'll try to provide help for those two systems. If you use anything else, you're on your own...

The solution you turn in should satisfy: (1) A single pdf (2) Typeset. Do not handwrite solutions. You want to make it easy for your classmates (and us) to read. You are free to use any typesetting software you like: LaTeX, Word, etc.

You will be turning in a single, readable "paper" (a single PDF file) with your solutions. In the Stellar Materials page we have provided some sample solutions from a previous year. The content in these examples is not directly relevant, but note that there are coherent explanations and nice illustrations of results in figures and tables. You should write your paper as if it is going to be read by someone who has taken a class in machine learning but has not read this assignment handout.

We will be emulating the process of submitting papers for publication to a conference. We will be using an actual conference review system (CMT) to have these papers peer reviewed (by the other students). This means that your answers have to be readable and understandable to your peers and, where possible, interesting. Note that when explanations are called for, you will need to convince the reviewers that you understand what you're talking about. The course staff will serve as the Program Committee for the conference and make all final decisions. **Piazza will be used to answer questions related to the process.**

Grading process and due dates

- We *strongly* encourage you to do this assignment in groups of two students, though you may do it individually if you wish. Post on Piazza if you need help finding a partner.
- You submit your paper via the CMT site by 11:59 PM on Thursday, September 28.
- Each student who was an author or co-author on a submission will be assigned 3 papers to review.

- Reviews must be entered on the CMT site by 11:59PM on Tuesday, October 3.
- All reviews will be made visible shortly thereafter, and students will have a two day period in which to enter rebuttals of their reviews into CMT if they wish.

Grading rubric

Your paper must be anonymous (no identifying information should appear in the PDF file). If it is not, it will automatically receive a 20% deduction, and will be graded by a grumpy staff member.

The paper must be no more than 6 pages long in a font no smaller than 10 point. It should include whatever tables, graphs, plots, etc., are necessary to demonstrate your work and conclusions. *It should not include code.*

Each of the four parts of the assigment will be graded on a scale from 0 to 5 (where 0 is failing and 5 is an A) on two aspects:

- **Content:** Did the solution answer the questions posed? Were the answers correct? Were the experiments well-designed or examples well chosen?
- **Clarity:** Were the results written up clearly? Were the plots labeled appropriately and described well? Did the plots support the points in the paper? Did the discussion in the paper illuminate the plots?

As a reviewer, you will be asked to provide a score for each section, and at least two paragraphs of feedback, per review, explaining things that were done well and things that could have been improved upon.

Your overall score for this assignment will be:

- **80%**: The average of all 8 scores on your assignment given by all three reviewers.
- **20%**: A score for the quality of your reviews. This will be full credit, by default. But we will skim reviews and examine some carefully and may reduce this grade for review commentary that is sloppy or wrong.

The course staff will spot-check submissions and reviews, paying careful attention to cases where there were rebuttals. The staff will act as the program committee and determine a final score. Our overall goals in this process are:

- To motivate you to work seriously on the problems and learn something about the machine learning material in the process
- To engage you in thinking critically and learning from other students' solutions to the problems

We will arrange to give full credit to anyone who submits a serious and careful solution to the problems and who gives evidence of having read carefully the solutions they were assigned and who writes thoughtful reviews of them.

The following questions are the points that your paper should cover in order to receive full credit. Your presentation should roughly follow the order of these questions so that your reviewers can see what you're doing. The number in the parenthesis to the right of the question title roughly indicates the relative weights for each question.

1 Implement Gradient Descent (35)

The simplest and most commonly referred-to method of optimization for differentiable cost function is gradient descent (see Bishop section 5.2.4, Murphy section 8.3.2, and Wikipedia). It's worth understanding its behavior on actual problems. In this question, you will implement two of its most important forms: batch gradient descent and stochastic gradient descent (SGD).

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 (or learning rate) and the convergence criterion (e.g., a threshold such that the algorithm terminates when the difference in objective function value on two successive steps is below this threshold or when the norm of the gradient is below this threshold). For this question, you can use a fixed step size in your implementation. Test your gradient descent procedure on two functions with the parameters provided by us in `parametersp1.txt` (for data files, we also provide basic reading scripts for MATLAB and Python): the negative of a Gaussian function and a quadratic bowl. Discuss (and illustrate) the effect of the choice of starting guess, the step size, and the convergence criterion on the resulting solution, as well as how the norm of the gradient evolves through the iteration.

   Tip: For a negative Gaussian function[1] of
   $$f(x) = -\frac{10^4}{\sqrt{(2\pi)^n|\Sigma|}} \exp\left[-\frac{1}{2}(x-u)^T\Sigma^{-1}(x-u)\right],$$
   the derivative can be written as
   $$\frac{\partial f(x)}{\partial x} = -f(x)\Sigma^{-1}(x-u).$$
   For a quadratic bowl of
   $$f(x) = \frac{1}{2}x^TAx - x^Tb,$$

[1]Here the negative Gaussian function merely serves as a test case for your gradient descent procedure. It is a well-known smooth, multi-dimensional function where the unique, global optimum is well-defined.

(with A being a positive definite matrix), its derivative is

$$\frac{\partial f(x)}{\partial x} = Ax - b.$$

Hint: We recommend that you do not use the norm of the gradient criteria for the Gaussian function because of numerical issues. Optionally, change the 10,000 multiplier to 1 and see how that affects the size of the gradient, your step-size and convergence criteria.

2. The gradient function may not always look as simple and clean as the ones provided above. A common way to check if one's gradient evaluation is correct or not is to use the central difference approximation (see the "Finite difference" article in Wikipedia) to numerically evaluate the gradient at various points. Write code to approximate the gradient of a function numerically at a given point using this method. Verify the gradient values on the functions you used in the question above by comparing the closed-form and numerical gradients at various points. Discuss the effect of changing the difference step (or "δ") on the accuracy of the gradient evaluation.

3. In machine learning and statistical estimation context, batch gradient descent uses samples from the full training set for each parameter update. Thus, each iteration would be slow for a large dataset. In addition, the native form of batch gradient descent does not provide a way to incorporate new data efficiently (i.e., in an online setting). Stochastic gradient descent (SGD) addresses these problems by using stochastic approximation on the gradient term (See Bishop 3.1.3, 5.2.4 and Wikipedia). In this question, we will use both batch gradient descent and SGD on a least square fitting problem.

   The dataset is provided in `fittingdatap1_x.txt` and `fittingdatap1_y.txt`. Each row of X and y represents a single data sample pair $(x^{(i)}, y^{(i)})$ and your goal is to find a coefficient vector θ that minimizes the least square error of $J(\theta; X, y) = \|X\theta - y\|^2$, which can be written as a sum as $J(\theta) = \sum_i (x^{(i)T}\theta - y^{(i)})^2 = \sum_i J(\theta; x^{(i)}, y^{(i)})$. In this question, you don't need to augment X with 1's as we did in the lecture. Notice that $J(\theta)$ is natively a function over the whole dataset and there is a closed-form solution to this problem (Bishop Equation (3.15)), which would be useful for checking your answers.

   a) Use batch gradient descent on $J(\theta)$ with a fixed step size. This would be the case where the gradient of the cost function for the entire training dataset is used in each parameter update.

   b) In contrast to batch gradient descent, which uses all training data in every parameter update, SGD performs a parameter update based on a single training example:

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta J(\theta_t; x^{(i)}, y^{(i)}),$$

where η_t is the learning rate for each iteration $t = 1, 2, ...$ and $J(\theta_t; x^{(i)}, y^{(i)}) = (x^{(i)T}\theta_t - y^{(i)})^2$ is the objective function in terms of a single data sample. Write a general procedure that performs SGD on this dataset. You should first derive the point-wise (with respect to a single data sample) gradient function, $\nabla_\theta J(\theta_t; x^{(i)}, y^{(i)})$. Then write a procedure to perform parameter update by iterating through the samples in the dataset for some number of rounds until a stopping criterion is reached (such as convergence of the full objective function over all data points).

Tip: In order to have a guaranteed convergence for SGD, the *learning rate schedule* η_t for t can be chosen to satisfy the Robbins-Monro conditions:

$$\sum_{t=1}^{\infty} \eta_t = \infty, \sum_{t=1}^{\infty} \eta_t^2 < \infty.$$

An example of this is to choose $\eta_t = (\tau_0 + t)^{-\kappa}$ for constants $\tau_0 \geq 0$ and $\kappa \in (0.5, 1)$.

c) Compare the behavior of the two implementations in terms of accuracy and number of evaluations of the point-wise gradient, $\nabla_\theta J(\theta_t; x^{(i)}, y^{(i)})$. That is, each iteration of batch gradient descent takes n (the number of total samples) evaluations of the point-wise gradients, whereas each iteration of SGD takes one evaluation of it.

d) (Optional) If you plot out the full objective function vs. iteration number, you may have noticed that SGD does not strictly descent at every iteration and may have large "jumps". A third in-between method is often used to take the best of both worlds. This is the mini-batch gradient descent where it uses a subset of training samples for parameter update in a stochastic setting and can smooth out these large "jumps". You may choose to try out mini-batch gradient descent on this dataset. Does the convergence become "smoother" as you would expect compare to the case of SGD?

2 Linear Basis Function Regression (25)

Let's consider the linear combination of basis function class of regression models. We know how to get closed-form solutions for the maximum likelihood weight vector (see Bishop equation 3.15, Murphy section 7.2 and equation 7.16). We'll use this problem to "benchmark" the gradient descent solution.

We have provided you with a text file (`curvefittingp2.txt`) that has 11 data points for you to fit. The function we used for generating these data is $y(x) = \cos(\pi x) + 1.5\cos(2\pi x)$, with a little added noise. We have also given you code to read this data and some code illustrating how to generate plots, both in Python and Matlab.

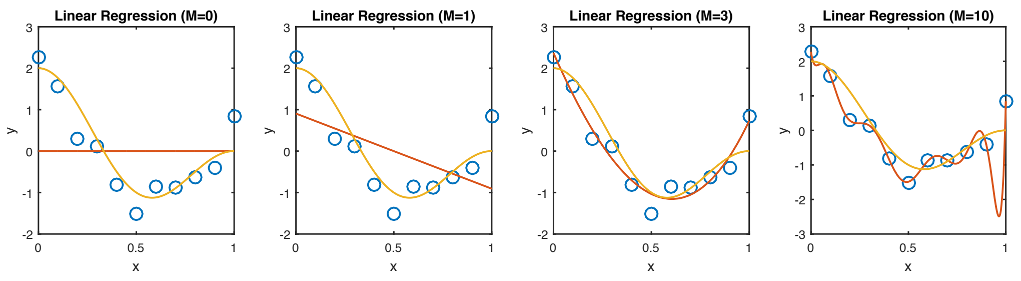

Figure 1: Plots of polynomials having various orders M, shown as red curves, fitted to the data set shown with blue circles. The yellow curve shows the function $\cos(\pi x) + 1.5\cos(2\pi x)$ used to generate the data.

1. Assume that we do not know the real basis for these points and want to try out a polynomial fitting. 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 $\phi_0(x) = 1, \phi_1(x) = x, \ldots, \phi_M(x) = x^M$. Test your solution by replicating the plots above.

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 batch gradient descent on the SSE function on some values of M. Describe your experience with initial guesses, step sizes and convergence thresholds. Compare with using SGD on the same data. Explain your results in terms of the properties of the function being optimized and the properties of the algorithms.

4. Assuming that you know the basis functions were cosine functions rather than polynomials, such that $\phi_1(x) = \cos(\pi x), \phi_2(x) = \cos(2\pi x), \ldots, \phi_M(x) = \cos(M\pi x)$, but you do not know which cosines you should use. Perform the maximum likelihood weight vector calculation for the cosine basis functions for the first eight $\phi_M(x)$. How does your weight vector compare to the true values we used to generate the data? (If you have prior beliefs on certain property (e.g., sparsity) of your regression coefficients, you can enforce it with regularization techniques, which we will explore in Q4. You don't need to implement it for this question.) Optional: add $\phi_0(x) = 1$

3 Ridge Regression (20)

1. Implement ridge regression (see Bishop equation 3.27, 3.28); the closed-form solution for the optimal solution of ridge regression is easy to calculate so do not use gradient descent for 3.1

and 3.2. Use the polynomial basis from 2.1 for 3.1 and 3.2. Experiment with different values of λ on the simple data from the previous question (`curvefittingp2.txt`), for various values of M. Describe your observations.

2. We have given you three additional data sets: `regressA_train.txt`, `regressB_train.txt` and `regress_validate.txt`. First, use `regressA_train.txt` as a training data set and `regressB_train.txt` as a test data set. Then, use `regressB_train.txt` as a training data set and `regressA_train.txt` as a test data set. Use `regress_validate.txt` as the validation set for both cases. For Matlab and Python, you can use the provided scripts to read the data sets. Each text file contains data in the following format: the fist row contains x data, and the second row contains y data: each row contains the data with %e format as %0.18e %0.18e %0.18e ... %0.18e where a space exists between the values. 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? Explain.

4 Sparsity and LASSO (20)

So far, we have been working with problems where the closed-form solution is very easy and thus gradient descent might be used only for computational reasons, e.g. big data sets. Let's now consider some problems where the closed-form solutions don't exist.

This problem provides a guide for estimating sparse weights with the LASSO estimator. For this section, use the following dataset: `lasso_train.txt`, `lasso_validate.txt` and `lasso_test.txt`. The format is the same as `regressA_train.txt`.

The dataset was generated according to $y = w_{\text{true}}^T \phi(x) + \varepsilon$, where $x, y \in \mathbb{R}$, $w \in \mathbb{R}^{13}$ and ε is a small noise. The feature vector is given by

$$\phi(x) = (x, \sin(0.4\pi x \times 1), \sin(0.4\pi x \times 2), \ldots, \sin(0.4\pi x \times 12)) \in \mathbb{R}^{13}.$$

Use this basis $\phi(x)$ in this problem. The true parameter w_{true}^T is provided in `lasso_true_w.txt`, where the 13 row entries are ordered consistently with the above equation (i.e., the first entry 0.0 is the true coefficient of x, and the second entry 0.0 is the true coefficient of $\sin(0.4\pi x \times 1)$, etc).

The goal is to estimate the weights w from the training dataset. If you look at `lasso_true_w.txt`, you will see that only four entries of w_{true}^T are non-zero. But without knowing it, we do not know which components are non-zero. Assume that w_{true}^T is unknown.

1. In ridge regression we used a quadratic regularizer (the sum of squares of the weights (L_2 norm)). Bishop's equation (3.29) shows a generalization to other exponents. In particular,

using $q = 1$ (the sum of the absolute values of the weights (L_1 norm)) is called the "LASSO". Specifically, we minimize the following objective function for LASSO:

$$\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - w^T\phi(x^{(i)})\right)^2 + \lambda\sum_{j=1}^{M}|w_j| \tag{1}$$

By using LASSO, estimate the weights w from the training dataset. Try different values of λ to get a good sparse estimation. Compare it with the results by the ridge regression for this data set. For example, plot `W_true`, the estimated w by the ridge regression (L_2 regularizer) and the LASSO (L_1 regularizer). Do you get sparse estimation from LASSO?

Implementation note: we can implement LASSO in several different ways: e.g., subgradient method and proximal gradient method (or projected gradient descent method). While proximal gradient method has a fast asymptotic convergence rate, you are free to choose any implementation. As those concepts are not taught in class or recitation, you can also simply use some publicly available implementation (e.g., `lasso` command in Matlab or `Lasso` from `sklearn.linear_model` in Python).

Figure 2 and Figure 3 provides some reference illustrations of this dataset. The purpose of these figures are only for dataset illustration (i.e., you are free to choose your own way of reporting the results). Also, keep in mind that the results will be different depending on different optimization algorithm, regularization coefficients, random initialization, etc.

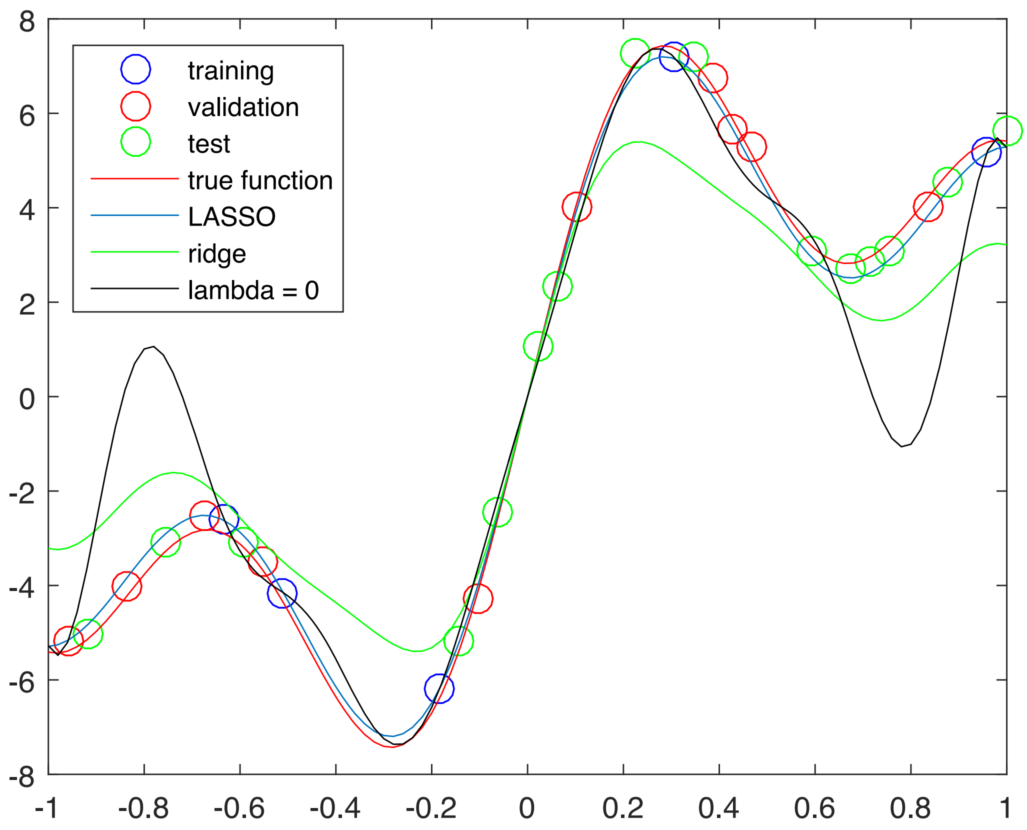

Figure 2: Plots of estimated function with different regularizers.

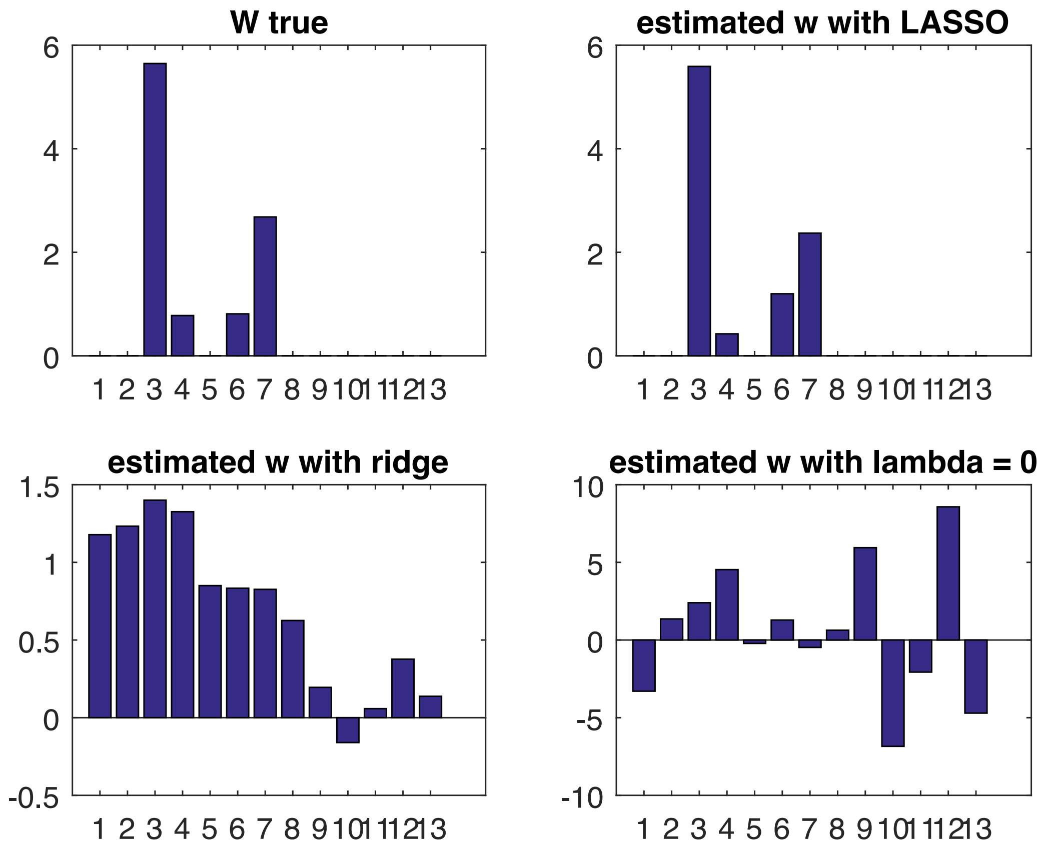

Figure 3: Estimated weights with different regularizers.