Machine Learning 6.867 - Pset 1

October 1, 2015

1 Implement Gradient Descent

1.1 Implementation

We implemented a basic gradient descent function in MATLAB. In this function, the user can specify the objective function and the corresponding gradient function. The user also inputs an initial guess, the constant step size, and the convergence threshold. The algorithm works as follows: starting at the initial guess, we move to a new point by taking a step (of the specified step size) in the direction of the gradient. We repeat the process until the difference in objective value between two subsequent iterations is less than the specified threshold.

1.2 Testing Gradient Descent

We test the gradient descent method on the following functions:

1. $f(x_1, x_2) = 3x_1^2 + 2x_1x_2 + x_2^2 - 4x_1 + 5x_2$
   By completing the square we find that the optimal solution is: $x_1 = 2.25, x_2 = -4.75$, and the optimal value is: $f(x_1, x_2) = -16.375$.

2. $f(x) = -\frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$
   This is the negative of a Gaussian pdf, with $\mu = 0$ and $\sigma = 1$. The optimal solution is $x = 0$ and the optimal value is: $f(x) = -\frac{1}{\sqrt{2\pi}}$.

3. $f(x) = x^3 - 10x$.
   This is not a convex function. There is one local minimum: $x = 1.8257$, $f(x) = -12.17$, but the global minimum is negative infinity.

The basic gradient descent method works well on the first two functions. For most specifications of the initial guess and step size, the procedure converges to the right answer. Increasing the convergence threshold to very large values leads to imprecise results, and decreasing it to very small values results in more iterations until convergence. In some cases, very large or very small step sizes may result in non-convergence. For the third function which is non-convex, selecting an initial guess $x_0 \geq -1.8257$ leads us to the incorrect, local minimum.

1.3 Numerical Gradient

We can also numerically evaluate the gradient at a given point. The partial derivative at x_i is approximated by the formula below, where $\mathbf{h}$ is the vector of all zeros except for the ith component that is ϵ. We apply this numerical gradient in place of the analytic one in the original three functions. For sufficiently small ϵ, the results are almost identical.

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \frac{f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}-\mathbf{h})}{2\epsilon} \tag{1}$$

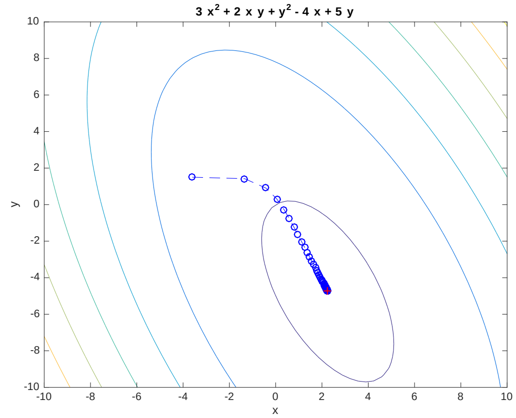


Figure 1: Contour plot of the first function. The gradient descent method moves in steps (blue dots) towards the local/global optimum (red cross).

1.4 Comparing with Sophisticated Methods

As the final step to evaluate our implementation of gradient descent, we compare our method against state-of-the-art gradient descent packages in MATLAB. Compared against `fminunc`, our implementation is dramatically slower. For example, given the same convergence criterion, it takes 35 iterations for our method to converge, while `fminunc` converges after 8 iterations. We think the disadvantage is due to the constant step size in our method. Since the step size does not adapt while the algorithm is running, our method is likely to undershoot or overshoot the optimal solution. An improvement could be made using backtracking, exact line-search, or other techniques to adaptively update the step size.

2 Linear Basis Function Regression

2.1 MLE for Polynomial Basis Function Regression

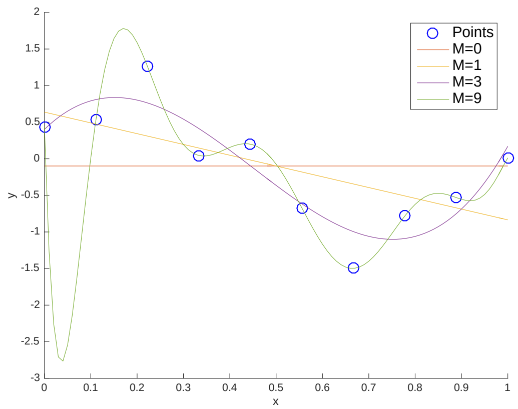
We implemented a maximum likelihood estimator for a linear regression with polynomial basis functions, using the formula $\mathbf{w} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$. The replicated weight vectors are close to what is reported in Table 1.1, and the plots we generated in Figure 2a match the ones in Bishop well:

2.2 Using SSE

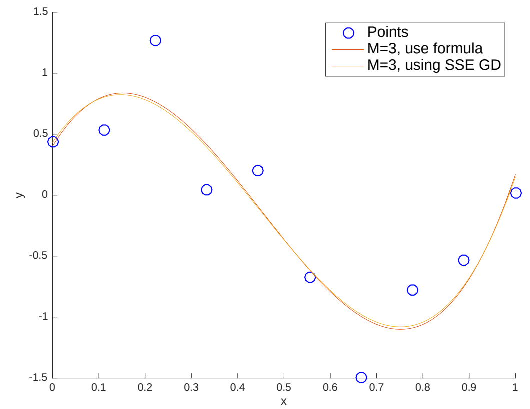
Alternatively, the maximum likelihood weight vector can be found using numerical gradient descent on the SSE. We first implement code to calculate the SSE: $(\mathbf{Y} - \mathbf{\Phi w})'(\mathbf{Y} - \mathbf{\Phi w})$. The derivative is $-2\mathbf{\Phi}(\mathbf{Y} - \mathbf{\Phi w})$, or can be computed numerically using the finite differencing method we implemented in Problem 1. We found that the two match.

2.3 Use Gradient Descent on SSE to find MLE

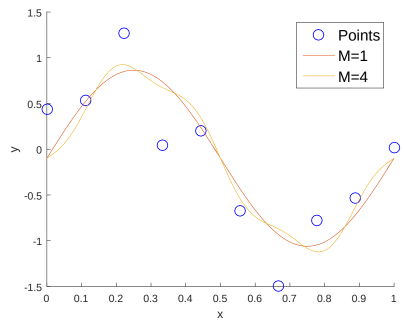
To get MLE for $\mathbf{w}$, we feed the SSE as the objective function in the gradient descend, $\mathbf{w}$ as the variable to be optimized over. With some smart initial guess, step size, and reasonably small convergence threshold, our $\mathbf{w}$ from gradient descent match the analytic solution using the formula largely (but not exactly). The comparison can be found in Figure 2b. Some bad choice of step size results in non-convergence or extremely large estimates.



(a) Replication of Bishop 1.4, using polynomial basis function up to degree M.



(b) Comparing the estimates using MLE equation or numeric gradient descent on SSE.



(c) Using sine basis function.

Figure 2

2.4 Sine Basis Function

We can also use sine basis functions, with $\phi_1(x) = \sin(2\pi x), \ldots, \phi_M(x) = \sin(2\pi M x)$. The estimated plots are in Figure 2c. We observe that the estimated function is not only periodic at period equal to one, for larger values of M it is even periodic within a period! Potential disadvantage is that the data is assumed to be periodic.

3 Ridge Regression

3.1 Implementation

Ridge regression is the particular case of regularized least squares with a quadratic regularizer term. The error function that we aim to minimize over is given by:

$$\frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\phi(\mathbf{x}_n))^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{2}$$

The closed-form solution of this problem is well-known, and can be derived by setting the gradient of (2) equal to zero. The optimal solution for $\mathbf{w}$ is provided by Bishop (2006), page 145:

$$\mathbf{w}_{ridge} = (\lambda\mathbf{I} + \Phi^T\Phi)^{-1}\Phi^T\mathbf{t} \tag{3}$$

We coded this method in MATLAB and tested our program using data from Bishop Figure 1.4, varying the parameters of λ and M. For the extreme cases, we observed that if $\lambda \leq 0.0001$, then $\mathbf{w}_{ridge} \approx \mathbf{w}_{OLS}$, and if $\lambda \geq 100$, then $\mathbf{w}_{ridge} \approx \mathbf{0}$.

3.2 Model Selection

To optimize parameter values for λ and M, we build our models using training data and then compare out-of-sample performance on validation data. For this example, we performed a grid search over the ranges: $\lambda = \{0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$, $M = \{0, 1, 2, 3, 4, 5\}$. We found that the model with $\lambda =$ TODO and $M =$ TODO yields the lowest MSE on validation data, so we select these to be the final parameter values. This model leads to MSE = TODO for the test data, which is the TODO lowest overall. In general, we observe that models with MSE for the validation set tend to yield low MSE on the test set.

4 Sparsity and LASSO

An alternative approach is to use the L_1 norm regularizer in the objective for regularized least squares. The error function that we aim to minimize over is given by:

$$\frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\phi(\mathbf{x}_n))^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j| \tag{4}$$

The LASSO is widely used in practice because it leads to solutions which are relatively sparse, i.e. the resulting prediction vector $\mathbf{w}$ tends to have few non-zero components.

To test this method and its sparsity properties, we consider a regression problem with 5 training points and 500 testing points in $\mathbb{R}^{12}$. The data is generated according to $f(x) = \mathbf{w}^T\phi(\mathbf{x}) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$ and $\mathbf{w} \in \mathbb{R}^{12}$ is a sparse vector with only two non-zero components. For this problem, we assume that $\mathbf{w}$ is unknown, and we estimate this vector with ridge regression in Section 4.1 and with LASSO in Section 4.2. Let λ_2 denote the regularizer term for the L_2 norm and let λ_1 denote the regularizer term for the L_1 norm. Figure 3 shows plots of the estimated function with different regularizers.

4.1 Ridge Regression Gradient Descent

Applying our method for gradient descent to the ridge regression problem on the training dataset of 12 points, we compute the vector of weights $\mathbf{w}$ and MSE on the testing dataset for $\lambda_2 = 0.1$. We obtain:

$$\begin{aligned}\mathbf{w} &= (-0.23, 0.13, 0.15, 0.19, 0.11, -0.04, -0.19, -0.31, -0.42, -0.49, -0.43, -0.20, 0.19)\\ \text{MSE} &= 179.7919.\end{aligned}$$

If we disable the L_2 regularizer (set $\lambda_2 = 0$), we obtain:

$$\begin{aligned}\mathbf{w} &= (-0.35, 0.09, 0.09, 0.08, -0.03, -0.17, -0.30, -0.42, -0.55, -0.68, -0.66, -0.37, 0.17),\\ \text{MSE} &= 446.4282.\end{aligned}$$

4.2 LASSO Gradient Descent

Similarly, we apply our method for gradient descent to the LASSO problem on the on the training dataset of 12 points. For $\lambda_1 = 0.1$, we find:

$$\begin{aligned}\mathbf{w} &= (-0.18, 0.00, 0.00, 0.26, 0.00, -0.00, -0.13, -0.13, -0.23, -0.70, -0.66, -0.00, 0.24),\\ \text{MSE} &= 111.0630.\end{aligned}$$

4.3 Sparsity Properties of LASSO

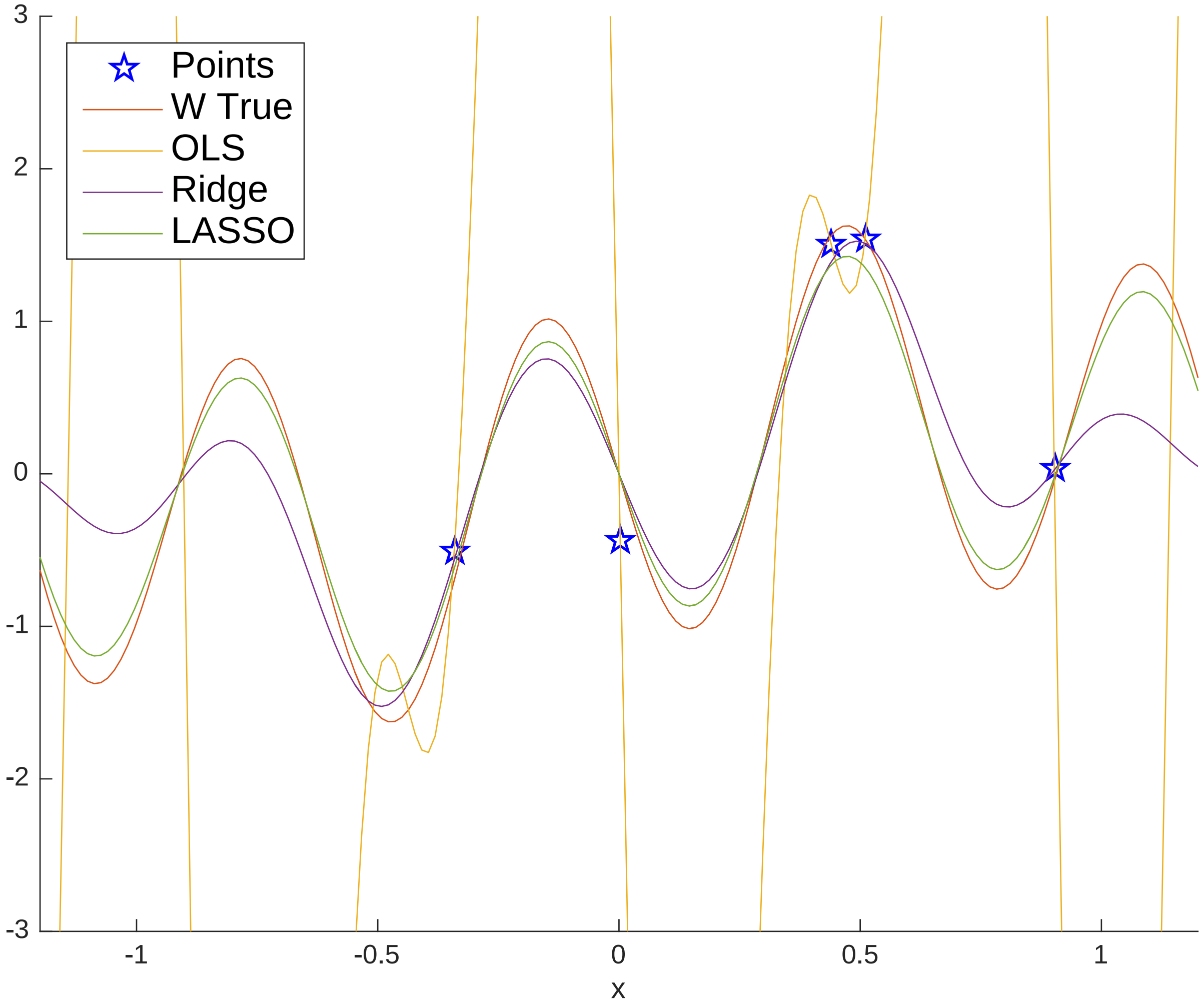


Figure 3: Plots of estimated function with different regularizers
OLS: $\lambda_1 = \lambda_2 = 0$, Ridge: $\lambda_2 = 0.1$, LASSO: $\lambda_1 = 0.1$.