Numerical Methods For Estimation
6.867 HW 1

Anonymous

September 28, 2016

1 Gradient Descent

1.1 Evaluation of Parameters

In this section, we implement a basic generic gradient descent algorithm. The goal of gradient descent is to optimize some differential cost function, by repeatedly adjusting our 'guess' for the optimizing value. Intuitively, this is achieved by iteratively 'moving' our guess in the direction of the greatest rate of decrease of the error function, scaled by a learning rate, towards some minimum value.

We implemented basic batch gradient descent with a random initialization, and adjusted to parameters of learning rate and convergence threshold to see how the performance of the algorithm varied when tested on the negative Gaussian function, and the quadratic bowl.

First, we evaluate the effect of initial guess on the convergence of the algorithm. Given several random initializations, we found that the algorithm always converged to roughly the same values. The negative Gaussian is a smooth function with a global minimum at the mean value, so there were no local minima in which random initializations could get stuck. Similarly, the Quadratic Bowl had reliable convergence. This is as illustration in Figure 1 below:

INSERT FIGURE OF RANDOM INITIALIZATIONS.

Secondly, we evaluated the effect of learning rate/step size on the performance of the algorithm. The learning rate, η is a multiplicative factor which controls the extent to which we move in the direction of the gradient.

The following plots shows how the convergence value varies with $\log \eta$, as well as how the number of iterations needed for convergence varied with $\log \eta$. This was run for a convergence criteria that the successive difference in the error function be at most $1e10$.

!!INCLUDE PLOTS FROM NOTEBOOK

Finally, we examined the effect of convergence criteria on the resulting solution. We considered a solution converged within a threshold, t, if the difference between successive evaluations of the error function was at most t.

1.2 Gradient Approximation

The functions used to test the implementations if the previous section all had simple closed form solutions, but in the cases where we wish to perform gradient descent on an objective function without such a simple derivative, we have to approximate the gradient at each point.

We approximate this gradient using a numerical method of "central difference", which involves evaluatindg the function at two points near the target point,at distances $\delta/2$ and $-\delta/2$ from the target point.

We then implemented the method of central differences, given by the following formula:

$$f'(x) = \frac{1}{\delta}[f(x + \frac{\delta}{2}) + f(x - \frac{\delta}{2})]$$

We tested our implementation of approximate gradient against the closed form solutions for the negative Gaussian and for the Quadratic bowl, for different values of δ to test its accuracy.

!!INSERT PLOTS

1.3 Stochastic Gradient Descent

In this section, we explore a modification to the basic gradient descent algorithm. In batch gradient descent, the gradient in the update is calculated over the entire set, which may be a costly operation. A variation used is stochastic gradient descent, in which the gradient is calculated over one point only, which is randomly chosen, so that in expectation, the gradient calculated accurately reflects the gradient over the entire dataset.

First, we ran batch gradient descent on the squared error function, using a fixed learning rate of $\eta =$?? and convergence was CCC.

We found that this algorithm converged to a minimum value of ≈ 91 in X iterations, as shown in the figure below.

Second, we implemented stochastic gradient descent, using a stopping criteria of convergence of the full objective function over all data points below a threshold, t.

We ran stochastic gradient descent until the same condition of convergence as batch gradient descent and obtained the following result, using a learning rate of $TODO$:

Comparing the performance of each algorithm on the *fittingdatap1x.txt* and *fittingdatap1y.txt* data sets, we see that WHAT DO WE SEE??

2 Linear Basis Function Regression

2.1 Implementation

In this problem, we implement regression on linear basis functions. The goal of regression is to predict $y \in \mathbb{R}$ given a data set $D = x \in \mathbb{R}^d$. We perform linear regression on functions which are not linear in the input variables by taking a linear combination of a fixed set of non-linear functions

of the input variables, known as basis function. In this problem we explore first a polynomial basis, with varying orders, M. The polynomial basis functions consist of $\{\phi_0(x) = x^0, ..., \phi_M(x) = x^M\}$ for 1-D data points. We implemented this regression and tested on the dataset *curvefittingp2.txt* to replicate the following plots.

!!INSERT THE REQUIRED PLOTS

2.2 Evaluation of Weight Vector

In order to evaluate the performance of the weight vector corresponding to a polynomial basis on a given dataset, we compute the sum of squares error, given by

$$\sum_{n=1}^{n} = \frac{1}{2}\{y_n - w^T\phi(x_n)^2\}$$

We also implemented a function to calculate the derivate of the least squares function, in order to perform gradient descent with least squares as the error function.

The derivative of the least squares function is given by

$$A$$

We tested our implementation of the least squares derivative against the approximation of the numerical gradient and obtained a GOOOD FIT>???

!!INSERT RESULTS!!

I need results to fill in this section because I don't know how things really worked.

3 Ridge Regression

In this section, we implement ridge regression (Bishop equation 3.27, 3.28). Ridge regression aims to mitigate the effect of overfitting by adding a regularization term that causes weights in $\mathbf{w}$ to shrink to zero, unless there is substantial evidence in the data against this.

The closed form solution for the sum-of-squares error function with the sum of squares regularizer is given by:

$$w = (\lambda I + \phi^T\phi)^{-1}\phi^T y$$

We implemented this solution, and modified the values of λ and M, with M+1 is the order of the polynomial to which we fit the data, in order to visualise how these parameters affect the fit of a simple set of data.

As expected, we found that all things constant, the higher the degree of the polynomial, the more suspiciously accurate the weight vector, $\mathbf{w}$. This was most pronounced for low values of λ, where the squared error in the data was more strongly emphasized than the norm of the weight vector.

Following this test on a simple dataset to visualize the effect of these parameters on over-fitting, we ran ridge regression on larger data sets. These data sets are split into three files - training

data, validation data, and test data, each of which serve a special purpose in evaluating a set of parameters: $\{w, \lambda, M\}$.

First, for a fixed M, λ, we used the training data to estimate the weight vectors, and repeated this process for different combinations of M, λ. For every set of parameters, we evaluated the performance on the validation data, and used the error of the validation data as an indicator of the performance on the test data. Then, we evaluated the error of the actual test data, relative to the performance on the validation data, to determine how well the parameters generalized from the validation data to the test data.

We found ....

4 Sparsity and LASSO

In the previous section, we studied regularization with a regularizer of the form

$$\frac{1}{2}\sum_{n=q}^{N}\{t_n - w^T\phi(x_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}\|w_j\|^q$$

where q=2. In this section, we will consider to study to effects of regularization, but this time, we will use $q = 1$ to give the special case of the lasso.

A larger value of λ tends to drive the values of some of the co-efficients w_j down to zero. This is called a *sparse* model.

We used the sci-kit implementation of LASSO to find the weights of the training