CSE446 Machine Learning, Spring 2013: Homework 1

Due: Friday, April 19^{th}, beginning of class

Instructions There are 4 written questions on this assignment, plus a fifth coding question. Submit both your written answers (as a .txt or a .pdf) and your implementation to the Dropbox at `https://catalyst.uw.edu/collectit/assignment/darylh/26829/108640`

1 Probability [15 points]

You're on vacation with your family, and you're incredibly bored. To pass the time, you and your cousin start to play a game. Your cousin collects 10 plastic cups, and turns them all upside-down on the floor. Underneath one of the cups, she hides a penny, but you don't know where. Now it is your turn to guess where the penny is hidden. Your cousin tells you to guess which cup is hiding the penny, and that she will then flip over one of the other cups that is not hiding the penny. Once this happens, she gives you the option of changing your guess to another cup.

1. *(5 points)* What is the probability that you have selected the cup hiding the penny? What is the probability that any single cup that has not been flipped is hiding the penny? What should you do? Please provide a brief (< 1 paragagraph) explanation of the reasoning behind your answer.

   As you are working out the probability of winning if you change your guess, your cousin grows impatient. Taunting you with a, "Here, let me help you out," she flips another cup, and another, all of them empty, until there are only two cups unflipped: your original choice and one other.

2. *(2 points)* Express the probability that the cup you originally selected hides the penny as a function of k the number of cups your cousin flips.

3. *(3 points)* Express the probability that any single cup that has not been flipped hides the penny, as a function of k the number of cups that your cousin flips.

4. *(5 points)* Generalize your answers to parts b and c. Suppose you start the game with n cups, and your cousin flips k of them. What is the probability that the original cup hides the penny, in terms in terms of n and k? What is the probability that any single cup that has not been flipped hides the penny, in terms of n and k?

2 MLE [15 points]

In DNA, also known as the Code of Life, there exist four different possible bases: adenine (abbreviated A), cytosine (C), guanine (G) and thymine (T). Now, you are given an organism which has a set of unknown DNA base frequencies. Let p_A, p_C, p_T, and p_G be those unknown frequencies. Assume that you obtain a strand of DNA and you want to infer the unknown frequencies. Let n_A, n_C, n_T, n_G the corresponding number of bases that you observe for A,C,T and G.

Please, be a frequentist and in analogy with the thumbtack example that Carlos explained in the class, derive the maximum likelihood estimates of the unknown parameters p_A, p_C, p_T, and p_G .

(Hint: You might find it useful to use the fact that $p_A + p_C + p_T + p_G = 1$.)

3 Linear Regression and LOOCV [20 points]

In class you learned about using cross validation as a way to estimate the true error of a learning algorithm. A solution that provides an almost unbiased estimate of this true error is *Leave-One-Out Cross Validation* (LOOCV), but it can take a really long time to compute the LOOCV error. In this problem you will derive an algorithm for efficiently computing the LOOCV error for linear regression using the *Hat Matrix*. [1] (This is the *cool trick* alluded to in the slides!)

Assume that there are n given training examples, $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$, where each input data point X_i, has d real valued features. The goal of regression is to learn to predict Y from X. The *linear* regression model assumes that the output Y is a *linear* combination of the input features plus Gaussian noise with weights given by β.

We can write this in matrix form by stacking the data points as the rows of a matrix X so that x_{ij} is the j-th feature of the i-th data point. Then writing Y, β and ϵ as column vectors, we can write the matrix form of the linear regression model as:

$$Y = X\beta + \epsilon$$

where:

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}, X = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ x_{21} & x_{22} & \ldots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nd} \end{bmatrix}, \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \end{bmatrix}, \quad \text{and } \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

Assume that ϵ_i is normally distributed with variance σ^2. We saw in class that the maximum likelihood estimate of the model parameters β (which also happens to minimize the sum of squared prediction errors) is given by the *Normal equation*:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Define $\hat{Y}$ to be the vector of predictions using $\hat{\beta}$ if we were to plug in the original training set X:

$$\begin{aligned} \hat{Y} &= X\hat{\beta} \\ &= X(X^T X)^{-1} X^T Y \\ &= HY \end{aligned}$$

where we define $H = X(X^T X)^{-1} X^T$ (H is often called the *Hat Matrix*).

As mentioned above, $\hat{\beta}$, also minimizes the sum of squared errors:

$$\text{SSE} = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

Now recall that the Leave-One-Out Cross Validation score is defined to be:

$$\text{LOOCV} = \sum_{i=1}^{n} (Y_i - \hat{Y}_i^{(-i)})^2$$

where $\hat{Y}^{(-i)}$ is the estimator of Y after removing the i-th observation (i.e., it minimizes $\sum_{j \neq i} (Y_j - \hat{Y}_j^{(-i)})^2$).

1. *(3 points)* What is the time complexity of computing the LOOCV score naively? (The naive algorithm is to loop through each point, performing a regression on the $n-1$ remaining points at each iteration.)

   Hint: The complexity of matrix inversion is $O(k^3)$ for a $k \times k$ matrix [2].

[1] Unfortunately, such an efficient algorithm may not be easily found for other learning methods.

[2] There are faster algorithms out there but for simplicity we'll assume that we are using the naive $O(k^3)$ algorithm.

2. *(1 point)* Write $\hat{Y}_i$ in terms of H and Y.

3. *(5 points)* Show that $\hat{Y}^{(-i)}$ is also the estimator which minimizes SSE for Z where

$$Z_j = \begin{cases} Y_j, & j \neq i \\ \hat{Y}_i^{(-i)}, & j = i \end{cases}$$

4. *(1 point)* Write $\hat{Y}_i^{(-i)}$ in terms of H and Z. By definition, $\hat{Y}_i^{(-i)} = Z_i$, but give an answer that is analagous to *2*.

5. *(5 points)* Show that $\hat{Y}_i - \hat{Y}_i^{(-i)} = H_{ii}Y_i - H_{ii}\hat{Y}_i^{(-i)}$, where H_{ii} denotes the i-th element along the diagonal of H.

6. *(5 points)* Show that

$$LOOCV = \sum_{i=1}^{n} \left(\frac{Y_i - \hat{Y}_i}{1 - H_{ii}} \right)^2$$

What is the algorithmic complexity of computing the LOOCV score using this formula?

Note: We see from this formula that the diagonal elements of H somehow indicate the impact that each particular observation has on the result of the regression.

4 Regularization Constants [15 points]

We have discussed the importance of regularization as a technique to avoid overfitting our models. For linear regression, we have mentioned both LASSO (which uses the L_1 norm as a penalty), and ridge regression (which uses the squared L_2 norm as a penalty). In practice, the scaling factor of these penalties has a significant impact on the behavior of these methods, and must often be chosen empirically for a particular dataset. In this problem, we look at what happens when we choose our regularization factor poorly.

4.1 LASSO regression

Recall that the loss function to be optimized under LASSO regression is:

$$E_L = \sum_{i=1}^{n} (Y_i - (\hat{\beta}_0 + X_i\hat{\beta}))^2 + \lambda \|\hat{\beta}\|_1$$

where

$$\lambda \|\hat{\beta}\|_1 = \lambda \sum_{i=1}^{d} |\hat{\beta}_i| \tag{1}$$

and λ is our regularization constant.

1. Suppose our λ is much too small; that is,

$$\sum_{i=1}^{n} (Y_i - X\hat{\beta})^2 + \lambda \|\beta\|_1 \approx \sum_{i=1}^{n} (Y_i - X\hat{\beta})^2$$

How will this affect the magnitude of:

(a) *(1 point)* The error on the training set?

(b) *(1 point)* The error on the testing set?

(c) *(1 point)* $\hat{\beta}$?

(d) *(1 point)* The number of nonzero elements of $\hat{\beta}$?

2. Suppose instead that we overestimated on our selection of λ. What do we expect to be the magnitude of:

   (a) *(1 point)* The error on the training set?

   (b) *(1 point)* The error on the testing set?

   (c) *(1 point)* $\hat{\beta}$?

   (d) *(1 point)* The number of nonzero elements of $\hat{\beta}$?

4.2 Ridge Regression

Recall that the loss function to be optimized under ridge regression is now:

$$E_R = \sum_{i=1}^{n} (Y_i - (\hat{\beta}_0 + X_i\hat{\beta}))^2 + \lambda \|\hat{\beta}\|_2^2$$

where

$$\lambda \|\hat{\beta}\|_2^2 = \lambda \sum_{i=1}^{d} (\hat{\beta}_i)^2 \tag{2}$$

If λ is too small, then the misbehavior of ridge regression is similar to that of LASSO in the previous section. Let's suppose λ has been set too high, and compare the behavior of ridge regression to LASSO.

To make this comparison, let's look at what happens to a particular feature weight $\hat{\beta}_i$. Since $\hat{\beta}$ is the solution that minimizes our error function E, it must be the case that $\frac{\partial E}{\partial \hat{\beta}_i} = 0$.

1. *(2 points)* Suppose we use the LASSO loss function E_L as defined in the previous section. Let's ignore the first term (which corresponds to the Sum Squared Error of our prediction), and calculate the partial derivative of the penalty term in Equation 1 with respect to $\hat{\beta}_i$. This can be thought of as the direction that LASSO "pushes" us away from the SSE solution. *Note that the absolute value is not differentiable at $\hat{\beta}_i = 0$, so you only need to answer for the case that $\hat{\beta}_i \neq 0$.*

2. *(2 points)* Instead, suppose we use the ridge regression loss function E_R from above. Again, ignore the SSE term and calculate the partial derivative with respect to $\hat{\beta}_i$ of Equation 2 to find how ridge regression "pushes" us away from the SSE solution.

3. *(3 points)* Comparing these two derivatives, for what values of $\hat{\beta}_i$ will their behaviors differ? What does this mean for our estimate of $\hat{\beta}_i$ when λ is very large?

5 Programming Question [35 Points]

Download the training data set "musicdata.txt" and the test data set "musictestdata.txt" from the website under Homework 1. Store your data in your working directory and read in the files with:

```
trainingdata <- read.table("musicdata.txt")
testdata <- read.table("musictestdata.txt")
```

Recall that this stores the data as a dataframe and not a matrix. The data consist of 91 variables concerning commercial songs released between 1922 and 2011. The first variable (the song's release year) is the response variable that we are trying to predict based on the remaining 90 input variables which consist of various measures of timbre. There are 1000 songs in the training data and 300 in the test data and it represents a fraction of the full data set which consists of over a million songs.

As there are a considerable number of input variables overfitting is a serious issue. In order to avoid this, implement the coordinate descent LASSO algorithm found on Murphy p.441 [3] . Your function [4] should accept a scalar value of λ, a vector-valued response variable ($\mathbf{y}$), and a matrix of input variables ($\mathbf{X}$) and it should output a vector of coefficient values ($\hat{\mathbf{w}}$) – don't forget the intercept.

Define "converging" as the change in any coefficient between one iteration and the next is no larger than 10^{-6}. (That is, $||\hat{\mathbf{w}}_{old} - \hat{\mathbf{w}}_{new}||_\infty < 10^{-6}$)

Once you have implemented a LASSO solver function run the solver using 100 different values of λ ranging from $\lambda = 100,000$ to $\lambda = 2,600,000$ [5].

In your analysis, include:

1. *(10 points)* All of your code
2. *(10 points)* The regularization paths (in one plot) for the coefficients for input variables 14 through 19 – use $\log(\lambda)$ instead of λ
3. *(5 points)* A plot of $\log(\lambda)$ against the squared error in the training data.
4. *(5 points)* A plot of $\log(\lambda)$ against the squared error in the test data.
5. *(3 points)* A plot of λ against the number of nonzero coefficients.
6. *(2 points)* A brief commentary on the task of selecting λ

[3] In the text Murphy discusses both a hard and a soft threshold, use the soft threshold as described in the referenced algorithm

[4] Do NOT use any pre-existing LASSO solvers or coordinate descent functions in R!

[5] These values were selected to be large enough that loops shouldn't take too long, but small enough that not all coefficients are zero