Stat 406 Spring 2008 Homework 3

1 Deriving the offset term

Let

$$J(\mathbf{w}, w_0) = (\mathbf{y} - \mathbf{X}\mathbf{w} - w_0\mathbf{1}_n)^T(\mathbf{y} - \mathbf{X}\mathbf{w} - w_0\mathbf{1}_n) \tag{1}$$

By solving $\frac{\partial}{\partial w_0} J(\mathbf{w}, w_0) = 0$, show that

$$\hat{w}_0 = \overline{y} - \overline{\mathbf{x}}^T\mathbf{w} \tag{2}$$

where $\overline{y} = \frac{1}{n}\sum_{i=1}^n y_i$ and $\overline{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i$.

2 Least squares using SVD

Let $\mathbf{w}$ be a solution of $\mathbf{X}\mathbf{w} = \mathbf{y}$. Let $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ be the SVD of $\mathbf{X}$. Prove that $\mathbf{w} = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^T\mathbf{y}$.

3 Multivariate linear regression

Multivariate linear regression is just like "regular" linear regression, except the output is a vector. Hence we replace the weight vector with a weight matrix:

$$\mathbf{y}_i = \mathbf{W}^T\mathbf{x}_i + \boldsymbol{\epsilon}_i \tag{3}$$

where $\mathbf{x}_i$ is a column vector of p inputs (covariates), $\mathbf{y}_i$ is a column vector of q outputs (responses), and $\mathbf{W}$ is a $p \times q$ matrix. We assume the noise is uncorrelated, $\boldsymbol{\epsilon}_i \sim \mathcal{N}(0, I_q)$.

1. Consider the objective minimizes

$$J(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^n ||\mathbf{e}_i||^2 \tag{4}$$

where $\mathbf{e}_i = \mathbf{y}_i - \mathbf{W}^T\mathbf{x}_i$ is the vector of residuals on training case i. Show that the minimal least squares estimator is given by

$$\hat{\mathbf{W}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{5}$$

where $\mathbf{Y}$ is a matrix whose columns are $\mathbf{y}_1$ to $\mathbf{y}_n$, and $\mathbf{X}$ is a matrix whose rows are $\mathbf{x}_1^T$ to $\mathbf{x}_n^T$. Hint: show that the objective decomposes into q independent univariate least squares problems. You may state the univariate result without proof.

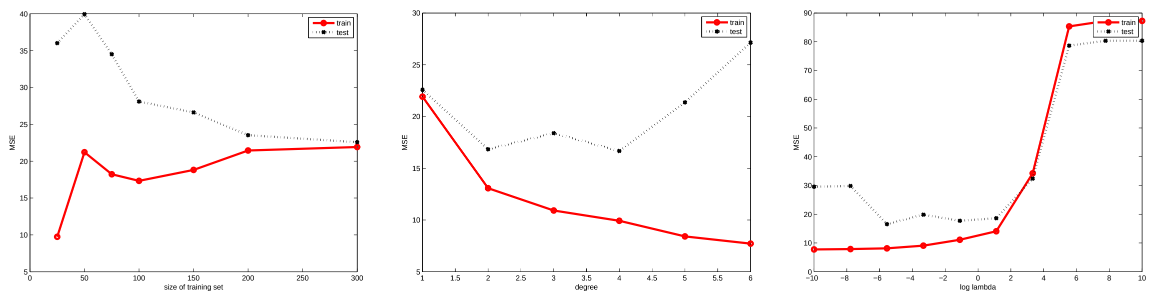
2. If the input $\mathbf{x}_i$ is transformed through a set of basis functions,$\boldsymbol{\phi}(\mathbf{x}_i)$, we can write

$$\hat{\mathbf{W}} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{Y} \tag{6}$$

where

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\mathbf{x}_1)^T \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}_n)^T \end{pmatrix} \tag{7}$$

is the modified design matrix. Consider the following example. $x \in \{0, 1\}$ and $\boldsymbol{\phi}(0) = (1, 0)^T$ and $\boldsymbol{\phi}(1) = (0, 1)^T$ (thus we encode the binary input as a 2-dimensional column vector). The response is also a 2-dimensional column vector (so $p = q = 2$). The dataset is



(a) (b) (c)

Figure 1: MSE vs (a) training set size, (b) polynomial degree, (c) size of ridge penalty. Solid Red = training, dotted black = test.

x	y
0	$(-1,-1)^T$
0	$(-1,-2)^T$
0	$(-2,-1)^T$
1	$(1,1)^T$
1	$(1,2)^T$
1	$(2,1)^T$

Compute $\hat{\mathbf{W}}$ from the above data.

4 Linear, polynomial and ridge regression on Boston housing data (Matlab)

We will use linear regression to predict house prices, using the famous **Boston housing dataset**, described at `http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`. There are 506 records. We will use first 13 features as inputs, $\mathbf{x}$, and the 14th feature, median house price, as the output y. All features are continuous, except feature 4, which is binary. However, we will treat this like any other continuous variable.

1. Load the `housing.data` file. We will use the first 300 cases for training and the remaining 206 cases for testing. However, the records seem to be sorted in some kind of order. To eliminate this, we will shuffle the data before splitting into a training/ test set. So we can all compare results, let use the following convention:

Listing 1: :

```
data = load('housing.data');
x = data(:, 1:13);
y = data(:,14);
[n,d] = size(x);
seed = 2; rand('state',seed); randn('state', seed);
perm = randperm(n);  % remove any possible ordering fx
x = x(perm,:); y = y(perm);
Ntrain = 300;
Xtrain = x(1:Ntrain,:); ytrain = y(1:Ntrain);
Xtest = x(Ntrain+1:end,:); ytest = y(Ntrain+1:end);
```

2. Now extract the first n records of the training data, for $n \in \{25, 50, 75, 100, 150, 200, 300\}$. For each such training subset, standardize it, and fit a linear regression model using least squares. (Remember to include an offset term.) Then standardize the whole test set in the same way. Compute the mean squared error on the training subset and on the whole test set. Plot MSE versus training set size. You should get a plot like Figure 1(a). Turn in your plot and code. Explain why the test error decreases as n increases, and why the train error *increases* as n increases. Why do the curves eventually meet?

   As a debugging aid, here are the regression weights I get when I train on the first 25 cases (the first term is the offset, w_0):

```
26.11   -0.58   3.02    -0.38  -0.39   1.19   4.72 ...
-4.30   -4.82   0.63    -3.25  -0.21   -0.27  -1.16
```

3. We will now replace the original features with an expanded set of features based on higher order terms. (We will ignore **interaction terms**.) For example, a quadratic expansion gives

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1,d} \\ & \vdots & & \\ x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{pmatrix} \rightarrow \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1,d} & x_{11}^2 & x_{12}^2 & \cdots & x_{1,d}^2 \\ & \vdots & & & & & & \\ x_{n1} & x_{n2} & \cdots & x_{n,d} & x_{n1}^2 & x_{n2}^2 & \cdots & x_{n,d}^2 \end{pmatrix} \tag{8}$$

   The provided function `degexpand(X,deg)` will replace each row of X with all powers up to degree deg. Use this function to train (by least squares) models with degrees 1 to 6. Use all the the training data. Plot the MSE on the training and test sets vs degree. You should get a plot like Figure 1(b). Turn in your plot and code. Explain why the test error decreases and then increases with degree, and why the train error decreases with degree.

4. Now we will use ridge regression to regularize the degree 6 polynomial. Fit models using ridge regression with the following values for λ:

   ```
   lambdas = [0 logspace(-10,10,10)];
   ```

   Use all the training data. Plot the MSE on the training and test sets vs $\log_{10}(\lambda)$. You should get a plot like Figure 1(c). Turn in your plot and code. Explain why the test error goes down and then up with increasing λ, and why the train error goes up with increasing λ.