CS340 Fall 2006: Homework 2

Out Mon 18 Sep, back Mon 25 Sep

1 Bayes rule

After your yearly checkup, the doctor has bad news and good news. The bad news is that you tested positive for a serious disease, and that the test is 99% accurate (i.e., the probability of testing positive given that you have the disease is 0.99, as is the probability of tetsing negative given that you don't have the disease). The good news is that this is a rare diseas, striking only one in 10,000 people. What are the chances that you actually have the disease? (Show your calculations as well as giving the final result.)

2 VC dimension

What is the VC dimension of the hypothesis class that defines positive points as lying inside a triangle? (Assume all points are in the plane R^2.) This is similar to the rectangle example studied in class, but now there are 3 sides. Give a concise proof of your result (pictures may help!). Hint 1: to show the VC dimension of a class is at least d, we must simply find *some* shattered set of size d. In order to show the VC dimension is at most d. we must show that *no* set of size $d + 1$ is shattered. Hint 2: consider points arranged on a circle.

3 Linear regression

We will use linear regression to predict the fuel consumption (miles per gallon) of various cars. (We will cover linear regression later in the class; in this homework, all of the necessary functions are given to you, so you can just use them as a "black box".) The data set is available from the UCI machine learning repository `http://www.ics.uci.edu/~mlearn/MLRepository.html`. It is included in the file mpg.data; a description of the fields is in mpg.names. We will use the 8th column (mpg) as the value to be predicted, and use the first 7 columns as input features (covariates). We will use the first 300 examples as training data, and the remaining examples as test data. The function `getData` loads the data, as follows

```
function [Xtrain, ytrain, Xtest, ytest] = getData()
  data = load('mpg.dat');
  N = size(data,1);
  x = data(:, 1:7);
  y = data(:,8);
  [n,d] = size(x);
  Xtrain = x(1:300,:); ytrain = y(1:300);
  Xtest = x(301:end,:); ytest = y(301:end);
```

We will fit a function of the form

$$f(x; w) = w_0 + \sum_{i=1}^{p} w_i z_i$$

where w_0 is an offset or "bias" term and z_i is a "standardized" version of the original features

$$z_i = \frac{x_i - \mu_i}{\sigma_i}$$

where μ_i and σ_i are the mean and standard deviation of feature i. The provided function `train_ls` estimates these parameters from data (ls stands for least squares):

```
function [w, mu, sigma] = train_ls(X, y)
```

Once we have "trained" the model, we can evaluate its performance by computing the mean squared error (mse) on any data set

$$mse = \frac{1}{N}\sum_{i=1}^{N}(f(x_i; w) - y_i)^2$$

where y_i is the true output and $f(x_i; w)$ is our predicted output. The provided function `testLinearPredictor` does this:

```
function [mse] = testLinearPredictor(x, y, w, mu, sigma)
```

Answer the following questions.

1. Write a script to train the model on subsets of the original training data. Use the following sizes: [10 20 50 100 200 300]. For example, to train on the first 50 examples, use

   ```
   [w, mu, sigma] = train_ls(Xtrain(1:50,:), ytrain(1:50))
   ```

   Plot the mse evaluated on the training set vs training set size. Plot the mse evaluated on the test set vs training set size. Turn in your code and plots.

2. Does the training error increase or decrease as the training set gets larger? Why?

3. Does the test error increase or decrease as the training set gets larger? Why?

4 Polynomial regression

We will now replace the original features with an expanded set of features (this is called **basis function expansion**). In particular, we will add quadratic terms

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1,p} \\ & \vdots & & \\ x_{N1} & x_{N2} & \cdots & x_{N,p} \end{pmatrix} \rightarrow \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1,p} & x_{11}^2 & x_{12}^2 & \cdots & x_{1,p}^2 \\ & \vdots & & & & & & \\ x_{N1} & x_{N2} & \cdots & x_{N,p} & x_{N1}^2 & x_{N2}^2 & \cdots & x_{Np}^2 \end{pmatrix}$$

In general, one would also add cross-terms such as $x_{ni}x_{nj}$. For simplicity, we will only consider powers of individual terms. The provided function `degexpand(X,deg)` will replace each row of X with all powers up to degree deg. For example,

```
x = [1 2 3;
     2 0.5 1.5;
     0 1 2];
degexpand(x, 2)
ans =
    1.0000    2.0000    3.0000    1.0000    4.0000    9.0000
    2.0000    0.5000    1.5000    4.0000    0.2500    2.2500
         0    1.0000    2.0000         0    1.0000    4.0000
```

To train a degree 3 model (using all the training data), you can use

```
[w, mu, sigma] = train_ls(degexpand(Xtrain, 3), ytrain);
```

1. Write a script to train models of degrees $d = 1, \ldots, 6$ (using all 300 training examples). Compute the mse on the training set and test set. Plot the error vs degree. Turn in your code and plots.
2. Does the training error increase or decrease as the degree gets larger? Why?
3. Does the test error increase or decrease as the degree gets larger? Why?
4. What order polynomial do you think is best?

5 Cross validation

In the previous section, we chose the polynomial order based on the error rate computed on a single test set. But this is a noisy estimate of the true generalization error. A better approach is to use K-fold cross validation, in which you compute the error rate on different subsets ("folds") of the data (training on all the rest), and average the results. For example, with $K = 10$, you train on 90% of the data and test on 10%, and repeat this 10 times, for different partitions into training/ test.

1. Implement 10-fold cross validation, and plot the cross-validated mse vs degree. Include error bars. Turn in your code and plots.
2. Does cross validation with small k tend to choose models that are too simple or too complex?