Advanced Statistical Inference
Loss minimization in vector form

1 Aims

- To become familiar with vector and matrix operations in Matlab.
- Implementing a cross-validation procedure for model selection.

2 Vectors and matrices

Load the olympics data (see instructions in previous lab if you've forgotten how). Create two vectors `x` and `t` equal to the first and second columns of `male100`:

```
x = male100(:,1);
t = male100(:,2);
```

To check that this has all worked correctly, plot the data:

```
plot(x,t,'b.','markersize',25);
```

It should look like the plot we've seen many times in the lectures.

2.1 Least squares in matrix form

In the lectures, we derived the following expression for the least squares solution in vector form:

$$\widehat{\mathbf{w}} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{t} \quad (1)$$

where

$$\mathbf{X} = \begin{bmatrix} x_1^0 & x_1^1 \\ x_2^0 & x_2^1 \\ \vdots & \vdots \\ x_N^0 & x_n^1 \end{bmatrix}$$

Our first step is to create $\mathbf{X}$. This can be done in many ways. Perhaps the easiest is with the following command:

```
X = [x.^0, x.^1];
```

Where

```
x.^k
```

is the vector `x` raised to the power k (remember the '.' operation from last time). We saw the square brackets last time when we defined a vector. We can use them to join any sets of variables into a vector/matrix as long as the sizes are consistent. In this case, we are OK because the two elements both have the same number of columns and both have one row. Inspect `X` to make sure it looks OK.

As an aside, test yourself by working out what the following command would do:


```
X = [(x.^0)',(x.^1)'];
```

(`a'` is the transpose of `a`)

Given `X`, we can easily compute Equation 1 with the following command:

```
w_hat = inv(X'*X)*X'*t;\\
```

Write a function that computes `w_hat` for a given `X` and `t`.

To fit higher order polynomials, we need to add extra columns to `X`. We could do this individually, or using a `for` loop. For example, the following code would add columns to `X` up to the third order terms:

```
K = 3;
X = [x.^0];
for k = 1:K
X = [X, x.^k];
end
```

Notice how we can keep iteratively extending `X`.

To make predictions, at a column vector of test points, `x_test`, we need to create `X_test` and then multply it by `w_hat`. For example

```
x_test = [2012;2016];
X_test = [x_test.^0];
for k = 1:K
X_test = [X_test, x_test.^k];
end
predictions = X_test*w_hat;\\
```

Write a function that, when given `x` and `x_test`, computes `w_hat` and makes predictions at `x_test`

You may find it interesting to test your code with different datasets. The following code will generate data from a third order polynomial:

```
x = rand(50,1);
x = sort(x);
x_test = rand(200,1);
x_test = sort(x_test);
noise = 0.5;
t = 5x.^3 - x.^2 + x + randn(50,1)*noise;
tt = 5x_test.^3 - x_test.^2 + x_test + randn(200,1)*noise;
```

You can compute the loss on the test data using:

```
di = (predictions - tt).^2;
meanerr = mean(di);
```

3 Cross-validation

Write a script that will perform LOO cross-validation

Some hints (you can generate data using the code above – you won't need to define `x_test` or `tt`)

- You'll need to write a loop that removes the data-points one by one, find `w_hat` on the reduced dataset and then compare the prediction on the removed data-point and compare it to the true value.
- To create a copy of `X` with the nth data-point (row) removed:

```
trainX = X;
trainX(n,:) = [];
traint = t;
traint(n) = [];
```

- To create a copy of the nth row of `X`

```
testX = X(n,:);
testt = t(n);
```

- You can use your previous function for finding `w_hat` and pass it `trainX` instead of X.
- To compute the error on a prediction (on, say, the nth fold):

```
err(n) = (testX*w_hat - testt)^2
```

- To find the average error:

```
mean_err = mean(err);
```

- Plot the average error for different polynomial orders. Does your code predict the correct one?