(These notes change the notes from class starting at the bottom of page 15, **The Covariance Matrix**)

Notation correction

The sample covariance is denoted by s^2 (and the standard deviation is denoted by s), and is defined as

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

This is also denoted as S_{xx}. Then the covariance would be

$$\mathrm{Cov}(X, Y) = S_{xy}$$

and we wouldn't square this (there was an error in the previous notes).

Normalizing a data set

We have to be careful dealing with data that is badly scaled, For example, if one dimension has all of its data scaled between 0.1 and 0.2, and in another dimension, the data is scaled to between -1000 and 2000, then any algorithm using this data will focus solely on the second variable (dimension) in order to get an accurate model, and will ignore the first variable.

Therefore, we typically want to make the data in each dimension similarly scaled. There are several ways to do this, here are two popular methods:

Scaled to $[0, 1]$

Given data $\{x_1, x_2, \ldots, x_n\}$, we find the minimum and the maximum data point, and call them $x_{\min}, x_{\max}$. Also, define $L = x_{\max} - x_{\min}$. Then every data point is scaled:

$$x_i \quad \rightarrow \quad \frac{x_i - x_{\min}}{L}$$

Notice that if $x_i = x_{\min}$, we get 0. Similarly, if $x_i = x_{\max}$, then we get 1. All the other data points are mapped to between 0 and 1, as desired.

Zero mean, unit STD

Given data $\{x_1, x_2, \ldots, x_n\}$, it is common to normalize the data to have zero mean and unit standard deviation. Therefore, given the sample mean $\bar{x}$ and sample variance s^2 (so the sample standard deviation is s), the new data set is constructed:

$$x_i \quad \rightarrow \quad \frac{x_i - \bar{x}}{s}$$

has a mean of zero and unit variance (and therefore, unit standard deviation).

Matrix notation and Scaling

(*NOTE:* The operations being discussed here wouldn't be defined in a standard linear algebra course. However, they are becoming standard in coding languages, since we need to perform these operations so often when working with data.)

Matrix plus row/col

Suppose our matrix is $p \times n$, where p is the number of points (or number of observations), and n is the number of dimensions (or number of variables).

Let m be a (row) vector consisting of the mean of each column, and let s be a row vector consisting of the standard deviation of the data in each column (therefore, m and s are both row vectors containing n elements).

For matrix arithmetic, assume that if we add a matrix and a row of length n, that means the row is added to every row of the matrix. Similarly, addition of a column would mean that the column is added to every column of the matrix (so the dimension of the row, column vector would need to be consistent with the dimension of the matrix).

For example,

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} - [1\ 1\ 2] = \begin{bmatrix} 0 & 1 & 1 \\ 3 & 4 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} - [1\ 1] = \text{ is not defined}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix}$$

Matrix multiplied/divided by a row/col

Similarly, let's take multiplication or division by an appropriately sized row or column would mean elementwise division to each row or column. For example,

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 8 & 10 & 12 \end{bmatrix}$$

$$\frac{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}{[1\ 2\ 3]} = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 5/2 & 2 \end{bmatrix}$$

Programming note in Matlab: This type of multiplication and division is denoted by a dot then the operation. For example, `A.*b` or `A./b`

Add/subtract, mult/divide a row and a column

If $\mathbf{r}$ is a row vector with n elements and $\mathbf{c}$ is a column vector with m elements, then adding, subtracting, multiplying or dividing them would result is an $m \times n$ matrix, where the (i, j) element would be computing by making the operation between $\mathbf{r}(j)$ and $\mathbf{c}(i)$, or we might visualize this in tabular notation:

	r_1	r_2	$\cdots$	r_n
c_1			$\cdots$	
c_2			$\cdots$	
$\vdots$				
c_m				

so that thet (i, j) entry of the array is made from the appropriate entries.

Scaling Using Matrix Notation

If X is a matrix $(p \times n)$, and $\mathbf{m}$ is the row vector of means (one for each column), $\mathbf{s}$ is a row vector of standard deviations in each column, then we define the standard matrix Z by taking:

$$Z = \frac{X - \mathbf{m}}{\mathbf{s}}$$

Then the covariance matrix of X is given by

$$C = \frac{1}{n-1}(X - \mathbf{m})^T(X - \mathbf{m})$$

And the **correlation matrix** is given by

$$\frac{1}{n-1}Z^T Z$$

Linear Regression

Basic Model

Given a set of p pairs of data:

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_p, y_p)\}$$

we're told that we wish to model the data as:

$$y = mx + b$$

and the problem is to find the slope m and the intercept b. Geometrically, this is finding a line that goes through the data. If we were able to do this **exactly**, then we know that two points determine the line, and so the rest of the data is not needed.

What makes the problem more difficult is that we'll assume that the data does not lie exactly on a line, and so in that sense, the problem has no solution.

Since the data does not lie on a line, no matter what m, b we choose, there will be an error between the y that is predicted by the model (let's use $\hat{y}$ for that), and the y coming from the data- We'll call this the **residual**:

$$\epsilon_i = y - \hat{y} = y_i - (mx_i + b)$$

Now we want to construct an **error function** (or **loss function**). Given an error function, we should then be able to solve the problem mathematically by minimizing the function.

An error function should be a function that maps our data to the non-negative real numbers. In our case, we wouldn't want to simply add up the residuals, since they may be negative and positive. To make them all positive, we'll define the error for the i^{th} data point as:

$$\epsilon_i^2 = (y_i - \hat{y}_i^2) = (y_i - mx_i - b)^2$$

Then overall, our error function E will be the **sum of the squared errors** (SSE), and the unknowns in the error function are the m, b:

$$E(m, b) = \sum_{i=1}^{p} (y_i - mx_i - b)^2$$

Finding the minimum of E

From calculus, we know how to find the minimum of E- The candidates for the minimum occur when the partial derivatives of E are all equal to zero (or, we set the gradient of E to zero):

$$\frac{\partial E}{\partial m} = 0 \qquad \frac{\partial E}{\partial b} = 0$$

Also, we might notice that E is a sum of errors, and the derivative of a sum is the sum of the derivatives. Therefore, it may be easiest to compute the partial derivatives term-by-term. For example, the i^{th} term in E is

$$(y_i - mx_i - b)^2$$

Therefore, the partial derivatives with respect to m and b (respectively) are the following (use the chain rule):

$$\frac{\partial}{\partial m}(y_i - mx_i - b)^2 = 2(y_i - mx_i - b)(-x_i)$$

and

$$\frac{\partial}{\partial b}(y_i - mx_i - b)^2 = 2(y_i - mx_i - b)(-1)$$

Following through with this last derivative, let's compute E_b and set it equal to zero.

$$E_b = \sum_{i=1}^{p} 2(y_i - mx_i - b)(-1) = 2\sum_{i=1}^{p}(-y_i) + 2m\sum_{i=1}^{p} x_i + b\sum_{i=1}^{p} 2 = 0$$

$$m\sum_{i=1}^{p} x_i + b \cdot p = \sum_{i=1}^{p} y_i$$

Similarly, we can show that solving for $E_m = 0$ will give us:

$$m\sum_{i=1}^{p} x_i^2 + b\sum_{i=1}^{p} x_i = \sum_{i=1}^{p} x_i y_i$$

and that gives us two equations in m, b:

$$\begin{aligned} m\sum_{i=1}^{p} x_i^2 + b\sum_{i=1}^{p} x_i &= \sum_{i=1}^{p} x_i y_i \\ m\sum_{i=1}^{p} x_i + b \cdot p &= \sum_{i=1}^{p} y_i \end{aligned}$$

Solving for m, b

From the equation for $\partial E/\partial b$, we can say that

$$b = \bar{y} - m\bar{x}$$

Using this expression for b, go to the other equation and substitute:

$$m\sum_{i=1}^{n} x_i^2 + (\bar{y} - m\bar{x})\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i$$

Now we can also start to solve for m:

$$m\left(\sum_{i=1}^{n} x_i^2 - \bar{x}\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} x_i y_i - \bar{y}\sum x_i$$

What we would like to do is to show that this equation leads us to the following:

$$m = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{S_{xy}}{S_{xx}}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Hint: You can also write that

$$\sum_{i=1}^{n} x_i = n\bar{x}$$

Then show by expanding out the square and simplifying, that

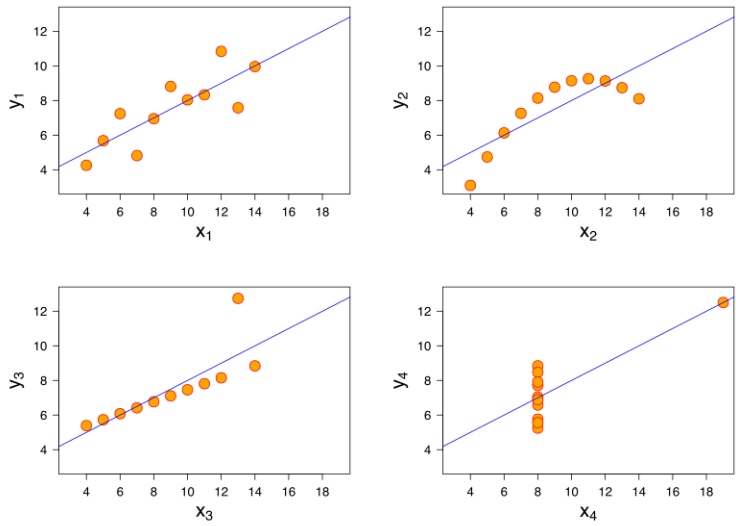
$$\sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2 = \sum_{i=1}^{n} x_i^2 - \bar{x}\sum_{i=1}^{n} x_i$$

Should you be using linear regresstion?

You should only be using linear regression if you have a reason to believe that there is a linear relationship in your data. An example showing the importance of looking at your data and looking at the residuals is given by Anscombe's quartet[1]. The quartet consists of four

[1]Figure from Wikipedia.

datasets that all have the same line of best fit, but are very different sets. The bottom two problems have outliers that change the underlying relationship by a significant amount.



Confidence intervals

Suppose you have a dataset with variables x and y, and you want to fit a linear model:

$$y = \beta_0 + \beta_1 x + \epsilon$$

To construct bootstrap confidence intervals for the parameters of a line of best fit, you can follow these steps:

1. **Fit the Original Model**:

   Fit the linear model to your original dataset to obtain the estimates of the parameters $\hat{\beta}_0$ and $\hat{\beta}_1$.

2. **Bootstrap Resampling**:

   Perform the following steps a large number of times (e.g., 1000 or more iterations):

   - Resample your dataset with replacement to create a bootstrap sample of the same size as the original dataset.
   - Fit the linear model to this bootstrap sample to obtain new estimates of the parameters, $\hat{\beta}_0^*$ and $\hat{\beta}_1^*$.
   - Save the estimates in a vector or matrix.

3. **Construct Confidence Intervals**:

- For each parameter we have a vector of estimates from the last step. Sort them.
- To construct a 95% confidence interval, find the 2.5th percentile and the 97.5th percentile of the sorted bootstrap estimates for each parameter. These percentiles serve as the lower and upper bounds of the confidence interval.

Here is some Matlab pseudo-code to generate a bootstrap confidence intervals for the slope and intercept.

```
% Original data
x = [your x data];
y = [your y data];

% Number of bootstrap samples
B = 1000;

% Fit the original linear model
p = polyfit(x, y, 1);
beta_0 = p(2);
beta_1 = p(1);

% Initialize arrays to store bootstrap estimates
beta_0_star = zeros(B, 1);
beta_1_star = zeros(B, 1);

n = length(x);

% Bootstrap procedure
for b = 1:B
    % Sample with replacement
    indices = randi(n, n, 1);
    x_star = x(indices);
    y_star = y(indices);

    % Fit model to bootstrap sample
    p_star = polyfit(x_star, y_star, 1);
    beta_0_star(b) = p_star(2);
    beta_1_star(b) = p_star(1);
end

% Compute confidence intervals
beta_0_ci = quantile(beta_0_star, [0.025, 0.975]);
beta_1_ci = quantile(beta_1_star, [0.025, 0.975]);
```

```
% Display results
fprintf('95%% Confidence interval for beta_0:\n');
fprintf(' [%f, %f]\n', beta_0_ci(1), beta_0_ci(2));
fprintf('95%% Confidence interval for beta_1:\n'
fprintf(' [%f, %f]\n', beta_1_ci(1), beta_1_ci(2));
```

Explanation

- **polyfit** is used to fit a linear model $y = \beta_0 + \beta_1 x$.
- **randi** is used to generate a vector of random indices for resampling with replacement.
- **quantile** computes the desired percentiles to form the confidence intervals.
- The confidence intervals are then displayed for both the intercept (β_0) and slope (β_1). Replace `[your x data]` and `[your y data]` with your actual data vectors.