Basic Data Analysis Tutorial

University of the Western Cape
Department of Computer Science
2009 July 24

Jacob Whitehill
jake@mplab.ucsd.edu

1 Introduction

In an undergraduate (BSc, BComm) degree program in computer science, it's easy to lose track of the "science" in "computer science" because a large proportion of courses are devoted to computer programming. In this tutorial, we will run through a few hypothetical but realistic experiments and demonstrate some simple but powerful data analysis tools.

Scientific research can be divided into two categories – *theoretical* and *empirical*. Theoretical research uses mathematics and logic to prove beyond any uncertainty that certain propositions are true. For instance, one can prove that the average-case time complexity of Quicksort is $O(n \log n)$. Empirical research is about drawing conclusions from observations. For example, in biology, based on comparing the DNA of two species, one might conclude that the species likely have common ancestry. Computer science is replete with examples of empirical data analyses. Here are a few examples from various sub-fields.

- **Databases**: You wish to monitor the number of queries submitted to a server as a function of time. For instance, maybe queries go up over certain holidays and down during other periods. Maybe the total number of customers is increasing year-by-year in an exponential fashion.
- **Networking**: How long, on average, does it take a packet to travel from client to server? What is the average packet queue length at a particular router? Which routing protocol delivers packets most consistently?
- **Human-computer interaction**: Which user interface satisfies users more? What other factors are these preferences correlated with?
- **Machine learning**: Aside from the fact that statistics and probability theory form the theoretical foundation of machine learning, there are countless empirical questions as well, e.g.: Which pattern recognition algorithm works better? What is the probability that a randomly selected image will be classified correctly?
- **Architecture**: Which caching strategy (e.g., write-through, write-back, etc.) delivers the highest throughput (processes-per-second)?

2 Notation

For anyone who doesn't know, the "big sigma" notation $\sum_{i=1}^{n} x_i$ translates into the following C code:

```
total = 0;
for (i = 1; i <= n; i++) {
     total += x[i];
}
```

3 First Example: Comparing Two Hardware Caches

We're going to motivate some of the techniques explained in this tutorial with a simple example from computer architecture. Suppose you have just developed a new memory cache on a computer chip, and

you wish to assess how much of a performance boost this gives to your programs. How would you go about answering this question?

The first thing to consider is what kinds of computer programs your target computer is intended to run. Knowing this will help you to design a *test bed* – a set of programs to run and whose running times to measure, both on your "new and improved" cache, and on the "regular" cache to which you're comparing it. Note that programs such as Firefox and Microsoft Word would likely not be good choices, because how long they take to "complete" depends entirely on when the user selects to "Quit". Instead, you would likely choose such programs as (under UNIX) `gcc`, `gzip`, or `sort` because they start and terminate by themselves, and their running times are not affected by user input.

Let's suppose that you write a single shell script to run all of the programs in your test bed consecutively; call this script `run.sh`. Now, for both your "new and improved" cache (which we'll call Cache A) and the "old" cache (we'll call it B), we can run `run.sh`, measure how long it takes the script to terminate, and compare the results between A and B. Hopefully (in our hypothetical scenario), the total running time for A will be lower than that for B.

It turns out that it is better to run `run.sh` not just once for A and B, but many times. In fact, the more times, the better. The reason is, each individual time you measure the running time of the script, there is always a bit of "noise" in the measurement. For one thing, the method with which you measure running time may not be completely precise – if you just look at a stop-watch, for example, there will likely be slight human error during each test run. More likely, you would use more sophisticated timing methods, but even then there will be noise. For instance, the exact order in which "system interrupts" (this will be discussed in Operating Systems) occur will also affect timing. Hence, in order to get a good idea of how long `run.sh` would run for A and B, it is better to run the script multiple times, and average the results.

Let's say we run the script 10 times each for A and B. (This is actually not that many, but otherwise the example will become tedious to typeset.) Each run time for Cache A will be called x_i ($i = 1, 2, \ldots, 20$), and each run time for Cache B will be called y_i. The data are as follows (measured in seconds):

$$x = [264.1, 188.8, 323.4, 156.6, 306.0, 155.0, 210.5, 287.6, 334.0, 124.3]$$
$$y = [365.8, 335.1, 277.3, 267.1, 269.3, 241.2, 281.7, 282.1, 343.5, 338.9]$$

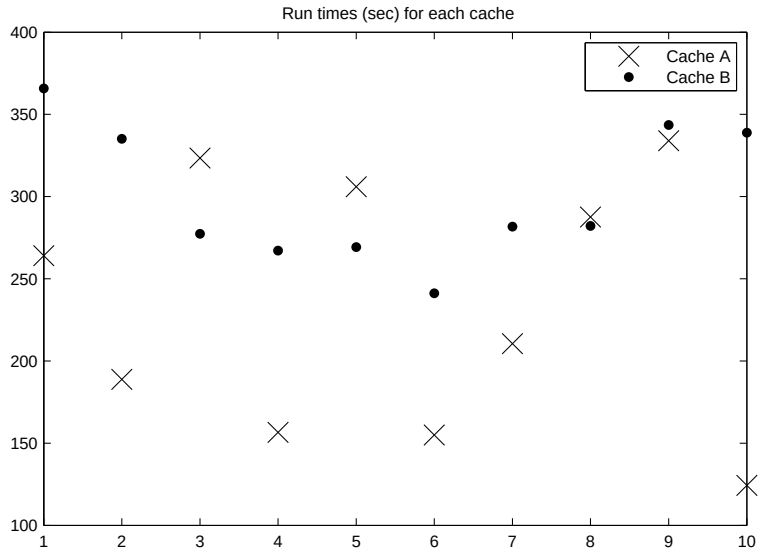
Which cache implementation, A or B, is better, i.e., delivers shorter run times for our test bed? In the sections that follow we'll introduce both visualization and statistical analysis methods to help answer this question.

4 Visualizing Data

The main reason to collect data is to discover some *trend* among them. Maybe the trend is that data in set x are smaller, bigger, more varied, less varied, faster-growing, slower-growing, etc., than the data in set y. This is especially useful for *time series*, i.e., data that were collected over different points in time. We're going to start with a simpler example, however, namely the series of repeated measurements of run times for Cache A and Cache B, as described in the previous example.

Various software packages, some free and some not, offer simple to sophisticated graphing capabilities. Some free packages include Octave, Gnuplot, and Python's SciPy package. The graphs in this tutorial were created using Matlab, but I also present code for Octave.

Let's visualize the cache run-time data. The plot of run times is shown below:



Perhaps the first thing that jumps out is that the run times for Cache B are much more "consistent" than those for Cache A, which are all over the place. If you look carefully, it also appears that the average run time for Cache A is slightly lower than for Cache B, i.e., Cache A is faster on average.

Visualizing your data is often extremely useful because it suggests to you a first place to start further analysis. The fact that Cache A's data points look "spread out" is an indication that we might want to compare the *variance*, which we'll discuss shortly, of both data series. Visualizing may also reveal a trend right-off – for instance, in a later example during this tutorial we'll find a more subtle trend among the data that would have been difficult to identify if the graph didn't point us in the right direction first.

For the cache example, however, it seems that visualization alone won't quite resolve the question of which cache implementation is faster. Cache A looks a little faster, on average, but we'll need to conduct a proper statistical analysis to be sure. This will be introduced in the next section.

5 Simple Summary Statistics

Suppose you've collected some set of N data points. *Summary statistics* provide a mean of succinctly characterizing data so that one can more easily find trends and compare different datasets than by analyzing the entire sets of data. Two of the most common summary statistics are the *mean* and *standard deviation*.

5.1 Mean

The mean of a set of data $\{x_i\}$ is

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

where x_i is the ith data point. Intuitively, the mean $\overline{x}$ is the *average* of the data points.

5.2 Variance

The variance of a set of data $\{x_i\}$ is

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

Intuitively, the variance measures the average distance, squared, of each data point from the mean. Why use *square* distance? Because otherwise, if one data point is, say, 5 *less* than the mean $\bar{x}$, and another data point is 5 *more* than the mean, then these distances would "cancel out." Since the variance is an

estimate of how "varied" the data is, we don't want this cancelling to occur. Hence, we square each distance so that each term in the summation is positive. Note that one could conceivably use the absolute value $|x_i - \overline{x}|$ instead of $(x_i - \overline{x})^2$ instead; the problem with absolute value is that it is not *differentiable*, which can be a problem in some cases. Hence, we use square distance (deviation) instead. Why divide by $N-1$ instead of N? Because using the former will give us an *unbiased* estimate of the variance. We won't worry about this now, but keep in mind for large data set sizes (i.e., large N) that dividing by N and dividing by $N-1$ will give almost exactly the same answer.

5.3 Standard Deviation

The standard deviation of a set of data $\{x_i\}$ is simply the square-root of the variance, i.e., $\sqrt{s^2} = s$. You can think of the standard deviation as roughly measuring the average distance of each data point from the mean, but since the square root is taken around the entire sum instead of around each square-distance, this is of course only approximate.

When reporting basic characteristics of a particular data set, one usually reports the standard deviation. In other cases, such as adding together several Gaussian random variables, it may be simpler to work with variances instead.

5.4 Example

Let's calculate the mean, variance, and standard deviation of the cache performance data, i.e., x and y as given above.

$$\begin{aligned}
\overline{x} &= \frac{1}{10}\left(264.1 + 188.8 + 323.4 + 156.6 + 306.0 + 155.0 + 210.5 + 287.6 + 334.0 + 124.3\right) = 235.0\text{ms} \\
\overline{y} &= \frac{1}{10}\left(365.8 + 335.1 + 277.3 + 267.1 + 269.3 + 241.2 + 281.7 + 282.1 + 343.5 + 338.9\right) = 300.2\text{ms} \\
s_x^2 &= \frac{1}{9}\left((264.1 - 235.0)^2 + (188.8 - 235.0)^2 + (323.4 - 235.0)^2 + \ldots\right) = 5978.4(\text{ms})^2 \\
s_x &= \sqrt{s_x^2} = 77.3\text{ms} \\
s_y^2 &= \ldots = 1735.5(\text{ms})^2 \\
s_y &= \sqrt{s_y^2} = 41.7\text{ms}
\end{aligned}$$

where s_x^2 and s_y^2 refer to the variance of data sets x and y, respectively.

By comparing the two means $\overline{x}$ and $\overline{y}$, we see that mean (average) of Cache A (the x data) was lower than that of Cache B; hence, it's faster. However, we also see that the variance of A is much higher than that of B. This means that the run times from Cache A will vary much more widely from their mean than for Cache B.

Whenever you collect a set of data that are inherently related to each other – e.g., a set of marks of fellow students in the same class, temperature readings from the same location, counts of the number of goals scored by the same player, etc. – it is a good idea to report both the mean and the standard deviation of these data. E.g., "We measured run times for both Cache A and Cache B over 10 trials. The mean and standard deviation of the run times for Cache A were 235.0sec and 77.3sec, and for Cache B they were 300.2sec and 41.7sec, respectively. (Keep in mind that the units for the mean and standard deviation are the same as for the individual data themselves.)

6 Second Example: Disk Fragmentation

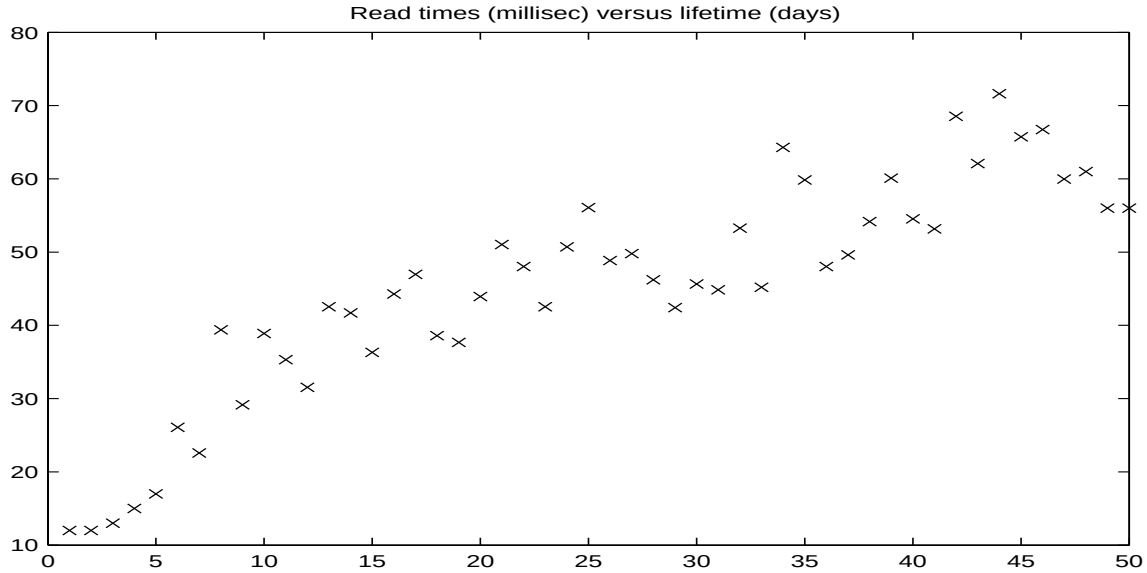
To introduce our second topic of data analysis we're going to switch to a different question. Suppose we wish to examine how disk fragmentation affects file read times. The more a disk is used by various programs, the more that small files tend to accumulate on the disk. The fact that small files are scattered all across the disk platters means that larger files have to be split up into smaller pieces in order to find enough room for storage. Splitting them up means it takes longer to read the contents of such files.

In this example we are interested in measuring the amount of time, in milliseconds, that it takes to read an entire file into memory as a function of the lifetime, in days, of the disk. We will assume that the disk is used by various programs, possibly by various users, during every day of its lifetime. The test program we will use to measure read time will create a file of 100MB, write it to disk, and then read it back. Due to increasing disk fragmentation through time, we expect that the read time will increase over the disk's lifetime.

Suppose the data look as follows (some of the data are omitted for brevity, but the graph that will follow contains them all):

Day	**Read time (millisec)**
1	12
2	12
3	13
4	15
5	17
...	...
49	56
50	56

We can graph these data, which form a time series, as shown below:



It doesn't take much effort to see that the data shown above follow a more or less linearly increasing path. In the next section, we'll talk about how to fit a line to data.

7 Regression

In the previous example, the data followed an approximately linear path. By plotting a line through the data points and extending it farther to the right, we can *extrapolate* to predict the y values for new x values. Similarly, we can *interpolate* between two existing x values and predict the corresponding y values. But how do we compute this line?

The topic of fitting a line to a set of given data points is known as *linear regression*.[1] Let's say you have collected some data $\{x_i\}$ and $\{y_i\}$ such that each y_i is associated with the corresponding x_i. We wish to come up with a linear function

$$y_i = wx_i + b$$

[1] If you wish to fit a different kind of function to your data, e.g., a polynomial function, then *non-linear* regression can also be employed.

that takes any x_i point as input and outputs the corresponding y_i value.[2] Recall from basic algebra that a linear function has the form $y = wx + b$. In the case of linear regresion, the w is called the *weight*, and the b is called the *bias* term.

Now, keep in mind that the data you've supplied ($\{x_i\}$) may not be perfectly linear, so the function we create will never be exactly correct for all data points, i.e., some of the y_i values output by our function will not exactly match the $\{y_i\}$ that we supplied. To denote the fact that our output y values are not exact, we'll add a "hat" to the y_i so it becomes $\hat{y}_i$:

$$\hat{y}_i = wx_i + b$$

The objective in linear regression is to come up with values for w and b so that the *distance* between each $\hat{y}_i$ – the approximate y value as estimated by the linear function – and the *true* value y_i is minimized. This should be intuitive because a "good" linear approximation to the data should pass as closely to the $\{y_i\}$ as possible. We wish to set w and b so that the sum of squared errors between each y_i and $\hat{y}_i$ is minimized. Since each $\hat{y}_i$ depends on w and b, we will write the sum as a function:

$$f(w, b) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

This function f may be called variously the reconstruction error, or the sum of residual errors, among other things.

Plugging in $wx_i + b$ for $\hat{y}_i$, we get

$$f(w, b) = \sum_{i=1}^{N} (wx_i + b - y_i)^2$$

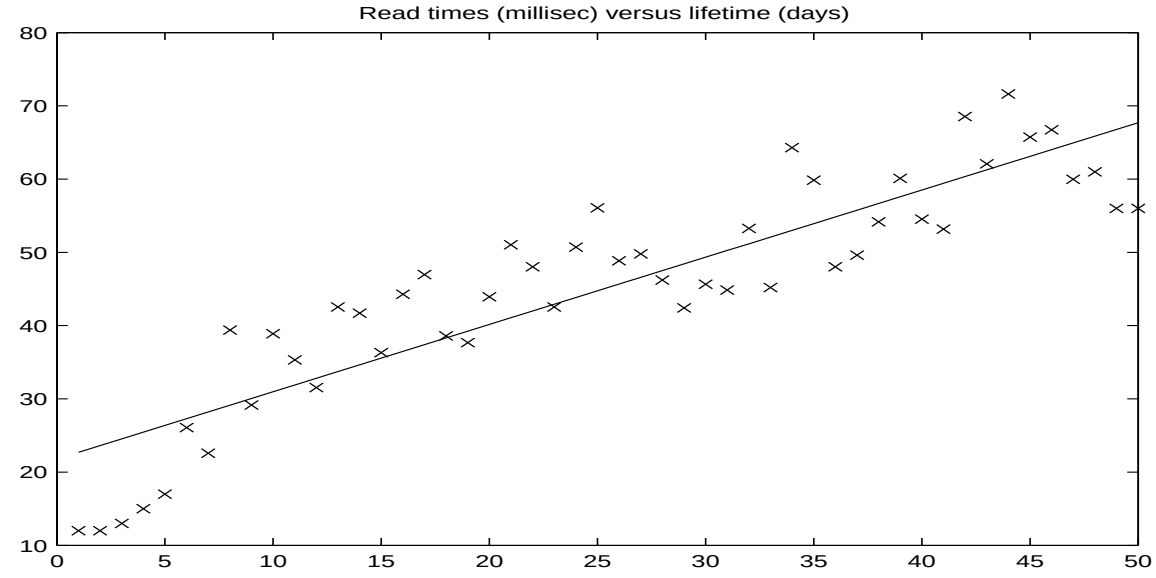
This function can be minimized using elementary differential calculus.

Given that we've computed w and b to give the best line that approximates the $\{y_i\}$ for each corresponding $\{x_i\}$, we can actually drop the subscript by i because this linear function can be used to estimate the y value for *any* corresponding x value:

$$\hat{y} = wx + b$$

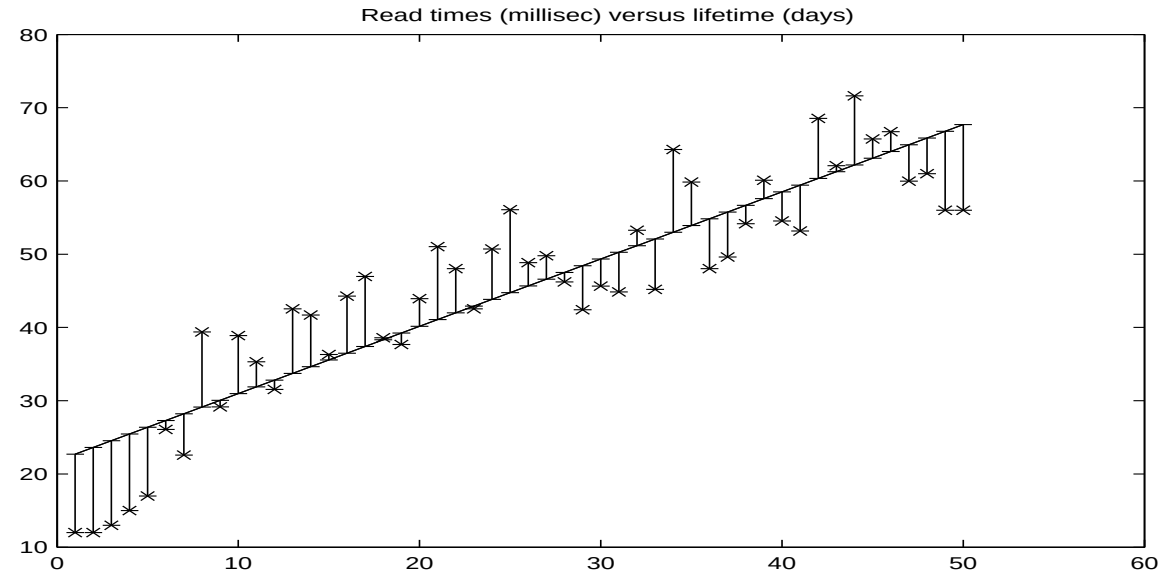
7.1 Example

Continuing with the disk fragmentation example started earlier, let's fit the best line possible to our data. It turns out that that values $w =$ and $b =$ minimize the sum of residual errors. The result is shown below:



[2] An example of a *non*-linear function would be $y_i = wx_i^2 + b$ because now the input is squared.

We can also show the individual deviations from the prediction for each data point, i.e., $y_i - \hat{y}_i$, for each i:



7.2 Code

In Octave, the easiest way (that I know of) of computing the best linear fit of a set of data is to use the `polyfit` function. It's called that because it fits a polynomial function of arbitrary degree (a linear function is a polynomial of degree 1) to a set of input data. It takes care of the calculus for you. All you give it are the $\{x_i\}$ and corresponding $\{y_i\}$, and it returns to you w and b. Actually, it will combine both w and b into a 2×1 vector, but this shouldn't be hard to work with. It works as follows:

```
x = ...  % fill in values
y = ...  % fill in values
v = polyfit(x, y);
w = v(1);
b = v(2);
```

You can now obtain the estimated $\hat{y}$ for a new x point with the line:

```
yhat = w * xNew + b;
```

7.3 Generalization to Vectors

Up to now, the $\{x_i\}$ and $\{y_i\}$ we've been dealing with have been scalar quantities, i.e., simple real numbers. However, they could instead be vectors of multiple numbers, and the mathematics largely remains the same. Linear regression over vectors is sometimes called multi-linear regression. We won't explore this topic further in this tutorial.

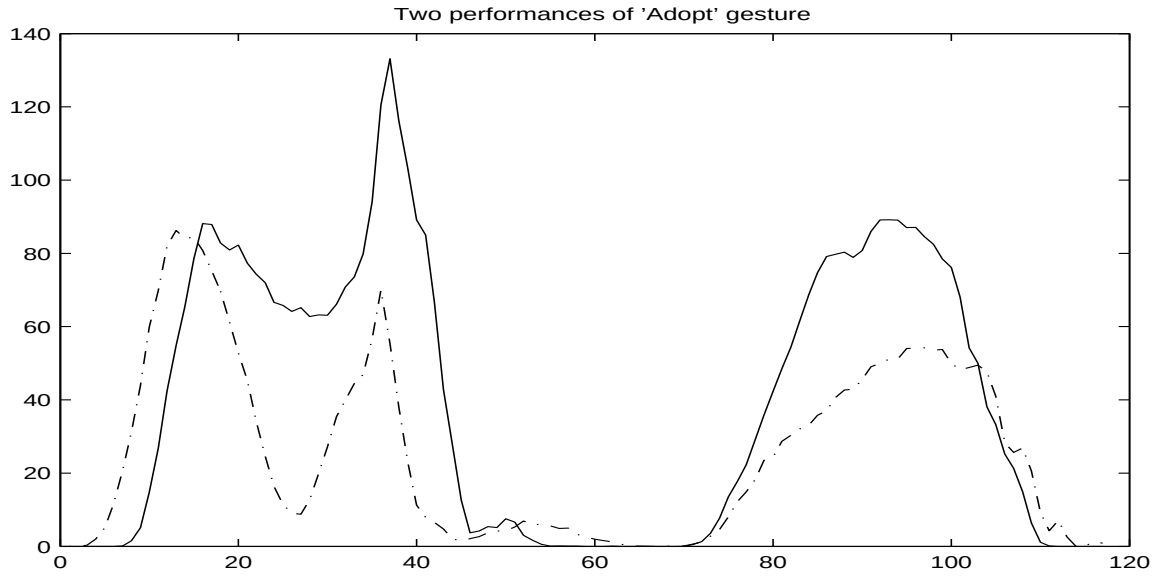
8 Third Example: Hand Gesture Recognition

This example will use real data obtained from Andrew Williams. Suppose you videorecord a person gesturing in front of a camera, and using the video input, wish to determine whether the person is making a particular hand gesture. This is an example of *pattern recognition*, a topic within the broader field of *machine learning*. In this example, we will use the following strategy to recognize a hand gesture: In each video frame, measure the total amount of motion that took place relative to the previous frame. This results in a single number (if you sum over all pixels in the image) for each frame. When we wish to recognize a person's gesture, we compare a *time series* (over all video frames we record) of that person's "motion numbers" to a database of previously recorded and analyzed gestures. Whichever gesture most

closely "matches up," in terms of those motion numbers, to the gesture in the newly recorded video, will be assigned to that new video.

The question is, how do we know how well each gesture in our database "matches up"?

Below is an example of two instances of the same hand gesture, performed (I believe) by two different people. Clearly, they are similar. But how do we measure this objectively and quantitatively? This leads us to the topic of *correlation*, discussed in the next section.



9 Correlation

There are different kinds of correlation. The most common is the *Pearson* correlation.

9.1 Pearson Correlation

The Pearson correlation r is a similarity measure between two data series $\{y_i\}$ and $\{z_i\}$ such that $-1 \leq r \leq 1$. **Informally**: If $r = 1$, then the two series are "completely similar." If $r = 0$, then they are not similar at all. If $r = -1$, then they are completely *inverse* to each other, i.e., when one series goes up, the other one goes down, and vice-versa.

The idea behind the Pearson correlation is that, given two data series $\{y_i\}$ and $\{z_i\}$, we want to get rid of "superficial differences," and *then* look at how close the data are to each other. By superficial differences, we mean any *scaling* or *shifting* of one series toward the other. Scaling and shifting are the result of a *linear transformation*, which we already discussed under the topic of regression, and indeed regression and correlation have much in common (they are highly correlated, harhar). The question when computing the "correlation coefficient" r is, after linearly transforming one data series to look as similar as possible to the other, how much *residual error* is there between the two series *relative to* the amount of variance in that target series?

9.2 Computing r^2

In this section we will discuss the meaning of the r^2 value, where r is the Pearson correlation coefficient. There's actually an explicit formula for r as well, but the one for r^2 offers a perhaps more intuitive interpretation. The formula for r^2 is as follows:

$$r^2 \quad = \quad 1 - \frac{\sum_{i=1}^{N}(\hat{z}_i - z_i)^2}{\sum_{i=1}^{N}(z_i - \overline{z})^2}$$

Let's analyze this equation starting with the numerator of the fraction: $\hat{z}_i$ is the *best estimate* of each z_i from the corresponding y_i value using linear regression, as discussed in the previous section. Hence,

the entire numerator is just the sum of residual errors. For two data series that are highly correlated, the sum of residual errors will be relatively small, and for data series that are not very correlated, the sum will be relatively large. Relative to what? To the denominator: the denominator measures the *variance* of the $\{z_i\}$. (In fact, the denominator is the variance multiplied by $N - 1$, but this ends up being unimportant). Hence, in the formula for r^2, the fraction, taken as a whole, measures how much residual error there is, when trying to predict the $\{z_i\}$ from the $\{y_i\}$, relative to how much *variance* there is in the $\{z_i\}$. If the variance in the $\{z_i\}$ is low, then they are clustered relatively close to their mean. Hence, if the correlation between $\{y_i\}$ and $\{z_i\}$ is low, then the sum of residual errors better must also be low.

The value of the fraction just described can be at most 1 because the sum of residual errors can never exceed the variance. (Just accept this as fact for now.) Hence, when we subtract it from 1, we arrive at the following interpretation for r^2: it is the proportion of variance of $\{z_i\}$ that does not arise from the residual error, i.e., it is the proportion of variance of $\{z_i\}$ that is accounted for by $\{y_i\}$. For two sets of data $\{y_i\}$ and $\{z_i\}$ that are highly correlated, the proportion of variance of $\{z_i\}$ that is accounted for by the $\{y_i\}$ (after linearly transforming them) must be high.

9.3 Procedure

To summarize the above mathematics in a procedural form, here is how the r^2 value is computed between data $\{y_i\}$ and $\{z_i\}$:

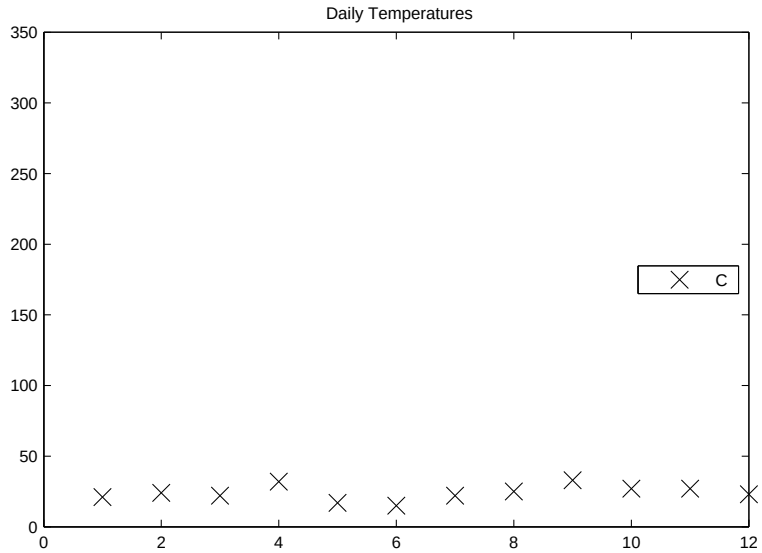
1. Perform linear regression from $\{y_i\}$ to $\{z_i\}$. As discussed in Section 7, linear regression minimizes the sum of residual errors, which is the numerator in Equation 1. This gives you a w and b with which you can estimate the $\{z_i\}$ as $\hat{z}$.
2. Calculate the sum of residual squared errors $g(w, b)$ for the $\{z_i\}$.
3. Calculate the variance of $\{z_i\}$.
4. Compute r^2 using the equation listed above.

One usually relies on a statistical software package to take care of this math. In Octave, this works as follows:

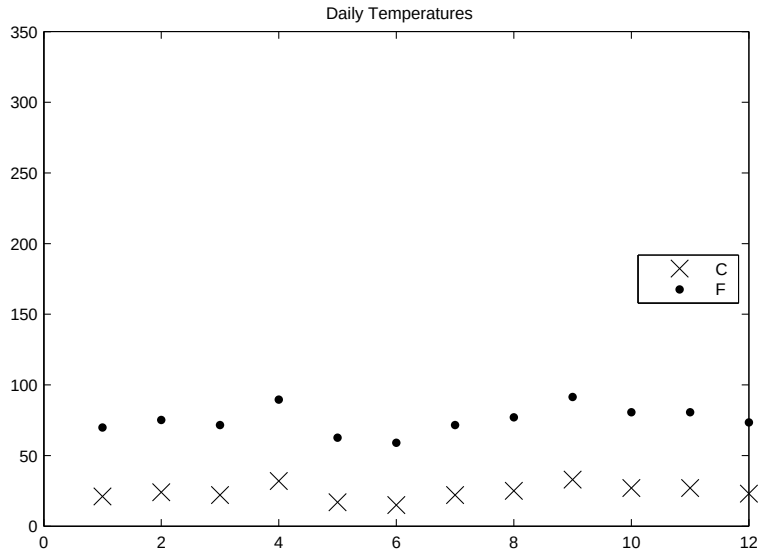
```
y = ...  % fill in values
z = ...  % fill in values
r = corrcoeff(y, z);
```

9.4 Example: Temperature Conversion

Suppose we measure the temperature, in degrees Celsius, of 12 consecutive days in Cape Town. The $\{x_i\}$ in this case will just be $1, 2, \ldots, 12$, and the $\{y_i\}$ will be the temperature measurements. We can plot these data in the following manner:



Now, suppose we convert the measurements in Celsius into measurements in Fahrenheit. We'll call the new "data" $\{z_i\}$. We can plot the two data series in the same graph:



Clearly, the two data series are related ("correlated", in fact) – each z_i is a linear transformation of the corresponding y_i (Recall that, to convert from Celsius to Fahrenheit, the formula is $T_f = \frac{9}{c}T_c + 32$.) Given y_i, we can perfectly compute z_i, and vice-versa.

Given either y_i or z_i we can perfectly estimate each of the other variable by performing some linear transformation – i.e., by multiplying the variable by some number, and then adding something to it. This stems from the fact that the $\{y_i\}$ and $\{z_i\}$ are *perfectly correlated*. In the language of statistics, we say that the Pearson r correlation between variables $\{y_i\}$ and $\{z_i\}$ is 1. What does this mean mathematically?

9.5 Computing the Pearson r correlation coefficient

The direct method of computing the correlation between two sets of data $\{y_i\}$ and $\{z_i\}$ is via the *covariance* of these two sets of data. First off, reality check: when computing the correlation between two sets of data points, each set must contain an equal number of elements. Second, we must first understand the concept of *covariance*.

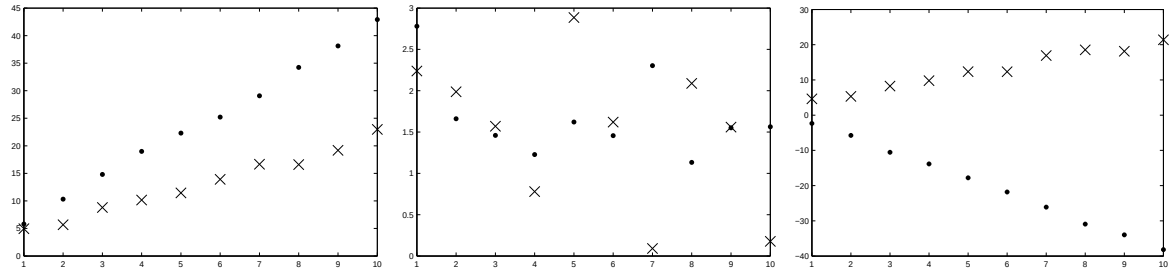
9.5.1 Covariance

The quantity

$$\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \overline{y})(z_i - \overline{z}$$

is known as the covariance of $\{y_i\}$ and $\{z_i\}$. Intuitively, if $\{y_i\}$ and $\{z_i\}$ have *positive* covariance, then each y_i tends to be bigger, on average, than $\overline{y}$ whenever z_i is bigger than $\overline{z}$. If they have *negative* covariance, then each y_i tends to be smaller, on average, than $\overline{y}$ whenever z_i is bigger than $\overline{z}$. Finally, if two variables have covariance of 0, then the fact that $y_i > \overline{y}$ for some i has no bearing whatsoever on whether $z_i > \overline{z}$.

Below are examples of $\{y_i\}$ and $\{z_i\}$ that have positive (Left), zero (Middle), and negative (Right) covariance:



Given a basic understanding of covariance, we can now compute the Pearson r correlation coefficient r (note that $-1 \leq r \leq 1$). The formula is:

$$r = \frac{\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \overline{y})(z_i - \overline{z})}{s_y s_z}$$

where s_y and s_z are the standard deviations of $\{y_i\}$ and $\{z_i\}$, respectively. You will recognize immediately the numerator, which is the covariance. When you divide the covariance by the product of the standard deviations $s_y s_z$, the fraction becomes *normalized*, i.e., it is bounded below by -1 and above by 1.

9.5.2 Interpreting r

If the Pearson correlation r between two sets of data $\{y_i\}$ and $\{z_i\}$ is 1, then the two sets of data are perfectly positively correlated. This means that, if $y_i > y_j$, then $z_i > z_j$. Moreover, the *amount* $y_i - y_j$ is always in fixed proportion to $z_i - z_j$.

If r is -1, then the two sets of data are perfectly negatively correlated, i.e., if $y_i > y_j$, then $z_i < z_j$

Finally, if $r = 0$, then the data are uncorrelated.

9.6 Interpreting r^2

Why the h*ll did I go through all this math? Because it is important to understand it when interpreting correlation coefficients. Suppose you use some statistical software package like Octave, Stata, Matlab, or whatever else, and determine that for two sets of data, their Pearson correlation coefficient $r = 0.7$. Since the maximum r is 1, this sounds reasonably high. The r^2 value is then 0.49. We can then interpret this number as follows: just under half of the variance in the $\{z_i\}$ is accounted for when predicting $\{z_i\}$ from the $\{y_i\}$ using linear regression. Where does the remaining variance come from? Noise. That is, *most* of the variation in the $\{z_i\}$ has nothing to do with the corresponding $\{y_i\}$.

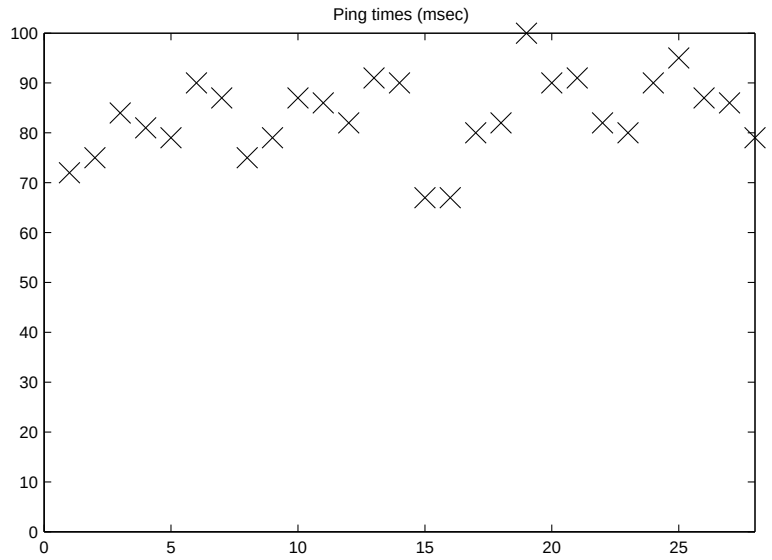
9.7 Example

Let's look at another example of correlations, this time using a topic from computer science. Suppose we're interested in how the average "ping" time from our computer to a particular server fluctuates as a

function of time. Let's say we measure the average ping time (in milliseconds) on 28 consecutive days, starting on a Saturday, and that we get the following data:

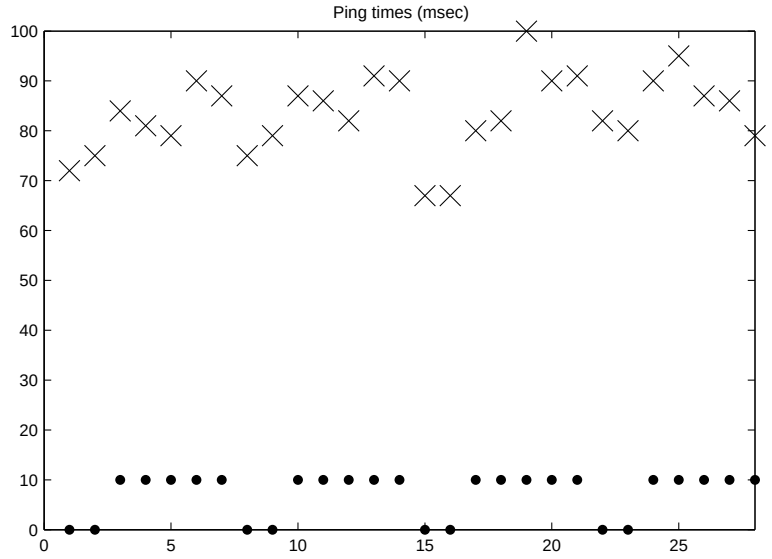
$$y = [72, 75, 84, 81, 79, 90, 87, 75, 79, 87, 86, 82, 91, 90, 67, 67, 80, 82, 100, 90, 91, 82, 80, 90, 95, 87, 86, 79]$$

Is there any meaningful trend in this data to be found? First, let's plot the data:



It's a bit hard to make out, but if you look carefully, you see that there are pairs of consecutive days, that occur every 7 days, on which the average ping time is lower. Why might this be? Where could this pattern of 5-days-higher followed by 2-days-lower come from? Recall that the first measurement was on a Saturday, which is a *weekend* day. It is conceivable that Internet traffic to the particular server we're querying might be lower on the weekend than on a weekday, and that the ping times would be correspondingly lower. Is there any quantitative evidence to support this hypothesis?

It turns out that correlation can help us here as well. Let's begin by adding some more "data" $\{z_i\}$ whose value is 0 if the corresponding x_i falls on a weekend, and 1 if x_i falls on a weekday. Let's plot the $\{z_i\}$ along with the $\{y_i\}$ ping data. To make the graph easier to read, we'll actually plot $\{10 * z_i\}$.



By examining the ping times above those days marked by $\{z_i\}$ as weekends, it seems, graphically, as though there might indeed be a trend. Now we can make use of correlation: Knowing whether each particular day x_i is a weekend, how well can we estimate each corresponding ping time y_i? In

particular, how much of the *variance* of the $\{y_i\}$ can we "explain" by making use of the $\{z_i\}$ under a linear transformation? This is simply the r^2 value, as we discussed before.

Using a statistical software package, we can calculate the Pearson r correlation coefficient. It turns out to be 0.72, which is quite high. This means that about $0.72^2 \approx 52\%$ of the variance in the ping times can be attributed to whether or not the day happens to be on a weekend. This is a fairly substantial fraction.

9.8 Spearman Rank Correlation

The Pearson correlation is the most commonly used form of regression. Another useful kind of correlation is the Spearman Rank Correlation. It is based on the following intuitive idea: Rank each data point in each set of data by how big it is relative to the other ones. That is, for each $\{x_i\}$, assign it rank 1 if it is the biggest, 2 if it is the second-biggest, and so on. Break ties arbitrarily. Then, compare the ranks of the $\{x_i\}$ with the ranks of the $\{y_i\}$, and assess how much they agree. Note that, in contrast to Pearson correlation, whether the relationship between the $\{x_i\}$ and the $\{y_i\}$ is linear or non-linear is irrelevant – all that matters is that, whenever x_i is bigger than x_j, the corresponding y_i should also be bigger than y_j.

9.9 Example

Consider the following data:

$$y = [1, 10, 0, 5, 3]$$

and

$$z = [2, 11, -4, 12, 10]$$

The Pearson correlation r is about 0.77, but the *Rank* correlation is 1. Notice how, if you sort the $\{y_i\}$, you'll get the same "ordering" as if you sort the $\{z_i\}$.

In Octave:

```
r = spearman(y, z)
```

10 Review and Concluding Remarks

This tutorial discussed the basic summary statistics of mean, variance, and standard deviation. It also introduced linear regression, which is a method of estimating one variable from another using a linear transformation, and correlation, which measures the proportion of variance of one set of data that can be explained by estimating them from another set of data using linear regression.

Regression is a very general technique and should be considered whenever you want to estimate one quantity, which presumably you do not know, from another, which is presumably easier to come by. Correlation is used to measure how related two sets of data are, i.e., whether knowing one can help you estimate the other. Both regression and correlation are bread-and-butter techniques of many, many empirical experiments that you may encounter in computer science and other science disciplines.