Statistics and Hacking: A Stout Little Distribution

Originalartikel

Backup

<html> Previously, we discussed how to apply the most basic hypothesis test: the z-test. It requires a relatively large sample size, and might be appreciated less by hackers searching for truth on a tight budget of time and money. As an alternative, we briefly mentioned the t-test. The basic procedure still applies: form hypotheses, sample data, check your assumptions, and perform the test. This time though, we’ll run the test with real data from IoT sensors, and programmatically rather than by hand. The most important difference between the z-test and the t-test is that the t-test uses a different probability distribution. It is called the ‘t-distribution’, and is similar in principle to the normal distribution used by the z-test, but was developed by studying the properties of small sample sizes. The precise shape of the distribution depends on your sample size. <figure id=„attachment_283571“ class=„wp-caption alignright c2“><figcaption class=„wp-caption-text“>The t distribution with different sample sizes, compared to the normal distribution (Hackaday yellow). Source: Wikipedia</figcaption></figure> In our previous example, we only dealt with the situation where we want to compare a sample with a constant value – whether a batch of resistors were the value they were supposed to be. In fact there are three common situations: You want to compare a sample to a fixed value: One sample t-test You want to compare two independent samples: Two sample t-test You have two measurements taken from each sample (e.g. treatment and control) and are interested in the difference: Paired t-test The difference mainly affects how you might set up your experiment, although if you have two independent samples, there is some extra work involved if you

have different sample sizes or one sample varies more than the other. In those cases you’re probably better off using a slight variation on the t-test called Welsh’s t-test. In our case, we are comparing the temperature and humidity readings of two different sensors over time, so we can pair our data as long as the sensors are read at more or less the same time. Our null and alternate hypotheses are straightforward here: the sensors either don’t produce significantly different results, or they do. <figure id=„attachment_283574“ class=„wp-caption aligncenter c3“><figcaption class=„wp-caption-text“>The two DHT11 sensors were taped down to my desk. They were read with a NodeMCU and the data pushed to a ThingsBoard server.</figcaption></figure> Next, we can sample. The readings from both sensors were taken at essentially the same time every 10 seconds, and sent via MQTT to a Thingsboard server. After a couple of days, the average temperature recorded by each sensor over 10 minute periods was retrieved. The sensor doesn’t have great resolution (1 °C), so averaging the data out like this made it less granular. The way to do this is sort of neat in ThingsBoard. First you set up an access token: <pre class=„brush: bash; title: ; notranslate“ title=„“> $curl -X POST –header 'Content-Type: application/json' –header 'Accept: application/json' -d '{„username“:„yourusername“, „password“:„yourpassword“}' 'http://host.com:port/api/auth/login' </pre> Then you request all data for a particular variable, averaged out every 10 minutes in JSON format (timestamps will be included): <pre class=„brush: bash; title: ; notranslate“ title=„“> $curl -v -X GET „http://host.com:port/api/plugins/telemetry/DEVICE/devicekey/values/timeseries?keys=variablename&startTs=1510917862000&endTs=1510983920000&interval=600000&limit=10000&agg=AVG“ \ –header „Content-Type:application/json“ \ –header „X-Authorization:Bearer (token goes here)“ > result.txt </pre> What’s cool about using an API like this is that you can easily automate data management and testing as parts of a decision engine. If you’re using less accurate sensors, or are just measuring something that varies a lot, using statistical significance as the basis to make a decision instead of a single sensor value can really improve reliability. But I digress, back to our data! Next, I did a little data management: the JSON was converted to a CSV format, and the column titles removed (timestamp and temperature). That made it easier for me

to process in Python. The t-test assumes normally distributed data just like the z-test does, so I loaded the data from the CSV file into a list and ran the test: <pre class=„brush: python; title: ; notranslate“ title=„“> import scipy.stats as stats import csv import math as math import numpy as numpy #Set up lists tempsensor1=[] tempsensor2=[] #Import data from a file in the same folder with open('temperature1.csv', 'rb') as csvfile: datareader = csv.reader(csvfile, delimiter=',', quotechar='|') for row in datareader: tempsensor1.append(float(row[1])) with open('temperature2.csv', 'rb') as csvfile: datareader = csv.reader(csvfile, delimiter=',', quotechar='|') for row in datareader: tempsensor2.append(float(row[1])) #Subtract one list from the other difference=[(i -j) for i, j in zip(tempsensor1, tempsensor2)] #Test for normality and output result normality = stats.normaltest(difference) print „Temperature difference normality test“ print normality </pre> In this case the normality test came back p>0.05, so we’ll consider the data normal for the purposes of our t-test. We then run our t-test on the data with the below. Note that the test is labeled ‘ttest_1samp’ in the statistics package – this is because running a 1-sample t-test on the difference between two datasets is equivalent to running a paired t-test on two datasets. We had already subtracted one list of data from the other for the normality test above, and now we’re checking if the result is significantly different from zero. <pre class=„brush: python; title: ; notranslate“ title=„“> ttest = stats.ttest_1samp(difference, 0, axis=0) mean=numpy.mean(difference) print „Temperature difference t-test“ print ttest print mean </pre> The test returns a t-test statistic of -8.42, and a p-value of 1.53×10^{-13}, which is much less than our threshold of p=0.05. The average difference was -0.364 °C. What that means is that the two sensors are producing significantly different results, and we have a ballpark figure for what the difference should be at a temperature of around 30 °C. Extrapolating that result to very different temperatures is not valid, since our data only covered a small range (29-32 °C). I also ran the above test on humidity data, but the results aren’t interesting because according to the datasheet (PDF warning), the relative humidity calculation depends on the temperature, and we already know the two devices are measuring significantly different temperatures. One interesting point was that the data was not normally distributed – so what to do? A commonly used technique is just to logarithmically transform the data without further consideration and see if that makes it normally distributed. A logarithmic transformation has the effect of bringing outlying values towards the average: <pre class=„brush: python; title: ; notranslate“ title=„“> difference=[(math.log1p(i) - math.log1p(j)) for i, j in zip(humidity1, humidity2)] normality = stats.normaltest(difference) print „Humidity difference (log-transformed) normality test“ print normality </pre> In our case, this did in fact make the data sufficiently normally distributed to run a test. However, it’s not a very rigorous approach for two reasons. First, it complicates exactly what you are comparing (what is the meaningful result if I compare the logarithm of temperature values?). Secondly, it’s easy to just throw various transformations at data to cover up the fundamental fact that your data is simply not appropriate for the test you’re trying to run. For more details, this paper points out some of the problems that can arise. A more rigorous approach that is increasing in popularity (just my opinion on both counts), is the use of non-parametric tests. These tests don’t assume a particular data distribution. A non-parametric equivalent to the paired t-test is the Wilcoxon signed-rank test (for unpaired data use the Wilcoxon rank-sum test). It has less statistical power than a paired t-test, and it discards any datum where the difference between pairs is zero, so there can be significant data loss when dealing with very granular data. You also need more samples to run it: twenty is a reasonable minimum. In any case, our data was sufficient, and running the test in Python was simple: <pre class=„brush: python; title: ;

notranslate“ title=„“> import scipy.stats as stats list1=[datalist_goes_here] list2=[datalist_goes_here] difference=[(i -j) for i, j in zip(list1, list2)] result=stats.wilcoxon(difference, y=None, zero_method='wilcox', correction=False) print result </pre> When we ran it, the measured humidity difference was significant, with an average difference of 4.19%. You might ask what the practical value of all this work is. This may just have been test data, but imagine I had two of these sensors, one outside my house and one inside. To save on air conditioning, a window fan turns on every time the temperature outside is cooler than the temperature inside. If I assumed the two devices were exactly the same, then my system would sometimes measure a temperature difference when there is none. By characterizing the difference between my two sensors, I can reduce the number of times the system makes the wrong decision, in short making my smart devices smarter without using more expensive parts. As a side note, it has been overstated that it’s easy to lie with statistics. To borrow an idea from Andrejs Dunkels, the real issue is that it’s hard to tell the truth without them. </html>