Worksheet for ICTIR 2013 Tutorial on Significance Testing in Information Retrieval in Practice and in Theory

Ben Carterette
University of Delaware
carteret@udel.edu

This is a worksheet to accompany the slides for my ICTIR 2013 tutorial. It is meant to explain how to perform standard significance tests in R and how to use simulation to investigate tests more deeply. My hope is that the reader that follows along will reach the same conclusion that I did: that there are so many factors that can affect the computation of a p-value, and so many of those that do not accurately reflect the reality of IR experimentation, that no objective meaning should be assigned to any p-value, no matter how carefully it has been computed.

Using R for statistical hypothesis testing

1. Basic R functions:

   Student's t-test

   ```
   t.test(x, y, mu=0, alternative="two.sided", paired=TRUE, conf.level=0.95)
   ```

   Perform a t-test for the null hypothesis that the mean difference is zero. `x` and `y` are vectors of evaluation results. Be sure to specify `paired=TRUE` as the function defaults to `FALSE` (which is less powerful).

   `t.test(x-y)` is identical to the paired test above, without needing to specify `paired=TRUE`. A paired test is simply a one-sample test on differences; we will use this fact to expedite our simulations below.

   Wilcoxon signed rank test

   ```
   wilcox.test(x, y, mu=0, alternative="two.sided", paired=TRUE, conf.level=0.95)
   ```

   Perform a Wilcoxon signed rank test for the null hypothesis that the median difference is zero. `x` and `y` are vectors of evaluation results. Be sure to specify `paired=TRUE` as the function defaults to `FALSE` (which is less powerful).

   Sign test

   ```
   binom.test(s, n, p=0.5, alternative="two.sided", conf.level=0.95)
   ```

   Perform a sign test for the null hypothesis that the probability of success is 0.5. For this function, `s` is the number of "successes", i.e. the number of times S_1 outperforms S_2, which you have to count yourself. If `x` and `y` are vectors of results, you can count successes using `sum(x > y)`.

2. Tests based on sampling: while there are some packages that include functions for randomization tests and bootstrap tests, it is easy to do it yourself.

   Randomization test: This function randomly changes the signs of differences of `x-y` and computes the p-value of the observed difference. This is a paired, one-tailed test for $x > y$, but modifying it to do a two-tailed test is easy.

   ```
   rand.test <- function(x, y, B=10000) {
     dist <- replicate(B, mean((2*rbinom(length(x), 1, 1/2)-1)*(x-y)))
     return mean(mean(x-y) > dist)
   }
   ```

   Bootstrap test: This function samples with replacement from the differences of `x-y` and computes the p-value of the observed difference. This is a paired, one-tailed test for $x > y$, but modifying it to do a two-tailed test is easy.

```
bootstrap.test <- function(x, y, B=10000) {
  dist <- replicate(B, mean(sample(x-y, length(x-y), replace=T)))
  return mean(mean(x-y) > 0)
}
```

3. The linear model: the t-test and ANOVA are both special cases of the linear regression model. To fit a linear mode, the data needs to be in an R data frame with at least three columns: evaluation measures, system ID (as a factor variable), and topic ID (as a factor variable).

   ```
   dat <- data.frame(m=c(x, y), sys=c(rep("x", length(x)), rep("y", length(y))),
                     topic=as.factor(1:length(x)))
   ```

   To perform an ANOVA test,

   ```
   summary(aov(m ~ sys + Error(topic), data=dat))
   ```

   This will report an F statistic and p-value; note that when there are two systems, F is the square of the t-statistic reported by `t.test`.

   To fit a linear regression model,

   ```
   summary(lm(m ~ sys + topic, data=dat))
   ```

   Note that the coefficient on `y` (the `sysy` coefficient) is equal to the mean difference between `x` and `y`, and the p-value is equivalent to the one reported by the t-test.

Simulating experiments to investigate tests

While most tests have a deep theory that can relate accuracy and power to effect size and sample size, the best way to understand them is to simply generate random data, run the test, and see what happens. It is necessary to do many thousands of trials, as p-values themselves have variance depending on the sample. This is part of why investigating test performance based on only TREC systems is insufficient: TREC gives us only a few samples of topics and experiments to base conclusions on, and we know very little about the representativeness of those samples.

Simulating no effect

Simulating the null hypothesis being true is easy: just sample values from a normal distribution with mean 0.[1]

```
> sim.map <- rnorm(50, 0, 1)
```

We can then perform a t-test on this data and retrieve the p-value. Note that testing on `sim.map`, which can be expected to have values greater than zero as well as less than zero, is equivalent to a paired test.

```
> t.test(sim.map)$p.value
0.4305228
```

Repeating this with a new random sample will produce a different p-value (demonstrating the variability in the p-value even when the null hypothesis is true); repeat it 10,000 times to generate a distribution of t-test p-values and draw the histogram:

```
> hist(replicate(10000, t.test(rnorm(50, 0, 1))$p.value))
```

[1] This is another problem with using TREC systems to investigate tests: they very rarely actually have no effect compared to a baseline; the null hypothesis is nearly always false. Over the 20+ years of TREC, there are only a couple of cases of TREC systems with identical evaluation results.

Note that the proportion of p-values less than 0.05—i.e. those for which we would incorrectly reject the null hypothesis—is pretty close to 0.05:

```
> mean(replicate(10000, t.test(rnorm(50, 0, 1))$p.value) < 0.05)
0.0531
```

This is a feature of the test: it falsely rejects the null hypothesis $100 \cdot \alpha$ percent of the time. (Of course you will get different results each time you do this test; the distribution of *those* results will be centered at 0.05.)

If we want our simulation to be more "IR-like" (that is, based on evaluation numbers between 0 and 1), we could reduce the variance of the normal distribution we're sampling from, or sample from Beta distributions like so:

```
> hist(replicate(10000, t.test(rbeta(50,1,1) - rbeta(50,1,1))$p.value))
```

Note that the proportion of p-values less than 0.05 is still pretty close to 0.05.

```
> mean(replicate(10000, t.test(rbeta(50, 1, 1) - rbeta(50, 1, 1))$p.value) < 0.05)
0.0483
```

That suggests that the oft-stated assumption that "the data is normal" is not really a big concern for the t-test.

If the null hypothesis is really true, it does not matter how large the sample is; the proportion of p-values less than 0.05 will still be around 0.05:

```
> mean(replicate(10000, t.test(rnorm(5000, 0, 1))$p.value) < 0.05)
0.0497
```

But, as I discussed in the tutorial, the null hypothesis is never true.

Simulating some effect

Simulating effect sizes greater than zero is somewhat trickier, as (for the t-test) effect size can be greater if the mean of the distribution is greater *or* if the variance is lower. For the sake of simplicity we will just look at the case where the mean is greater while variance stays fixed at 1 (the reader is invited to try other cases). [2]

When the mean difference is 0.05, the probability of correctly rejecting the null hypothesis based on a sample of 50 topics goes up only slightly from 0.05:

```
> mean(replicate(10000, t.test(rnorm(50, 0.05, 1))$p.value) < 0.05)
0.0645
```

What we should glean from this is that a small p-value should not always carry a lot of weight. If we measure a mean difference of 0.05 over a sample of 50 topics, and the t-test tells us $p = 0.05$, it is almost equally likely that the "real" effect in the population of topics is 0 as that the "real" effect is 0.05.

In particular, what is the posterior probability of H_0 being true against the posterior probability of $H_1 : \mu = 0.05$ being true?

$$P(H_0|p < 0.05) = \frac{P(p < 0.05|H_0)P(H_0)}{P(p < 0.05|H_1)P(H_1) + P(p < 0.05|H_0)P(H_0)} = \frac{0.05 \times 0.5}{0.0645 \times 0.5 + 0.05 \times 0.5} \approx 0.44$$

We must conclude that even though we have a p-value at the cutoff and we could reject H_0 based on that, we don't really have enough evidence to state that H_0 is false.

When the mean difference increases to 0.1, the probability of correctly rejecting the null hypothesis based on a sample of 50 topics goes up again:

```
> mean(replicate(10000, t.test(rnorm(50, 0.1, 1))$p.value) < 0.05)
0.1062
```

[2] Note that the t-test says nothing about *relative* differences! In fact, that is true of all the tests covered above. Yet relative differences are what we really more often care about in information retrieval!

But with a p-value of 0.05, the posterior probability of H_0 is still 0.33.

This trend continues, with the probability of correctly rejecting the null reaching about 95% when the mean difference gets to a little over 0.5 (again, with variance 1, which is unrealistic for IR—the reader is encouraged to try more realistic values of variance). It is only when the mean difference reaches 0.5 that the posterior probability of H_0 being true (given a p-value of 0.05 exactly) reaches 0.05.

It is also the case that increasing sample size increases the probability of obtaining a p-value less than 0.05. For example, doubling the sample size to 100 increases our ability to detect a mean difference of 0.05:

```
> mean(replicate(10000, t.test(rnorm(100, 0.05, 1))$p.value) < 0.05)
0.0773
```

Compare to the 0.0645 reported above. When sample size increases to 1000, the proportion of p-values less than 0.05 rises to about 0.35:

```
> mean(replicate(10000, t.test(rnorm(1000, 0.05, 1))$p.value) < 0.05)
0.3569
```

and with 10,000 topics, we have a nearly 100% chance of rejecting the null.

```
> mean(replicate(10000, t.test(rnorm(10000, 0.05, 1))$p.value) < 0.05)
0.9987
```

Based on this, you can see that small differences that are significant over 10,000 or more queries should *always* be analyzed deeper. Even if we can reject H_0 based on a p-value less than 0.05, the probability that H_0 is true may still be high.

Evaluating tests

The Neyman-Pearson school of thought says that significance tests have accuracy, power, Type I error rates, and Type II error rates, illustrated by the following contingency table:

H_0	true	false
not rejected	accuracy	Type II error
rejected	Type I error	power

Table 1: Contingency table for evaluating significance tests. They're basically binary classifiers!

First, let us state that there is no practical way to evaluate significance tests. This is partly because we cannot know the truth of the null hypothesis, and partly because researchers do not entirely agree on what they are telling us: Ronald Fisher has one school of thought; Jerzey Neyman and Egon Pearson another; what we actually use is a mishmash of the two that is not internally consistent.

But more importantly, it is because the null hypothesis is *never* true. In TREC data, it is only true when a participating group accidentally submits the exact same system twice, or when a system has such a severe bug that it retrieves no relevant documents (and thus scores 0 on every query). In the most literal sense, every time we conclude a difference is not significant we have committed a Type II error, unless the systems being compared are actually identical.[3] Therefore the best test is one that just rejects every null hypothesis without even looking at the data. Is that a useful test? Of course not, but that's because we are using tests not to determine the truth of H_0 but as a heuristic to help guide research.

It can still be useful to compare tests to see how often they disagree. R provides an easy way to create a contingency table between two tests, but first we will write a few helper functions:

```
t.test.sig <- function(z) t.test(z)$p.value < 0.05
wilcox.test.sig <- function(z) wilcox.test(z)$p.value < 0.05
rand.data <- function(n=50, mu=0, var=1, B=10000) replicate(B, rnorm(n, mu, var))
```

[3]Or in the very very low-probability case that they actually place different relevant documents at the same ranks such that their evaluation results are identical.

Then we will create some random data, apply both t-test and Wilcoxon signed rank test to it, and use the `table` function to generate the contingency table:

```
> dat <- rand.data()
> table(ttest=apply(dat, 2, t.test.sig), wilcox=apply(dat, 2, wilcox.test.sig))
       wilcox
ttest   FALSE TRUE
  FALSE  9407   91
  TRUE    100  402
```

When there is no effect, the two tests both have Type I error rates of about 5%, but even then they disagree pretty significantly on *which* experiments should be rejected.

Even with a small effect, the relative power of the t-test becomes evident:

```
> dat <- rand.data(mu=0.1)
> table(ttest=apply(dat, 2, t.test.sig), wilcox=apply(dat, 2, wilcox.test.sig))
       wilcox
ttest   FALSE TRUE
  FALSE  8777  139
  TRUE    211  873
```

The t-test is now correctly rejecting H_0 about 50% more often than the Wilcoxon signed rank test. Though of course every single case where either of them failed to reject is an error.

These experiments demonstrate the following:

- that two tests will not agree on what is significant, even when every null hypothesis is false;
- that the t-test is better at detecting differences that actually exist.

The reader is encouraged to try them with different tests, with different values of n, μ, σ, and with sampling from different (non-normal) distributions.

Modeling effects

Suppose we have obtained two complete sets of relevance judgments from different groups of assessors. We compute evaluation results with both sets and find that one system is better with one set while the other system is better with the other set. We might even find that the difference is significant with one set, but not with the other.

This demonstrates the importance of choosing which effects to model in a test. By default, using tests "out of the box", we *only* model topic and system effects—and "system effects" conflate the effects of a whole host of system modules, from the parser to the tokenizer to the stemmer to the ranking function to ... all of which can be developed independently but that interact with each other in a complete system.

Let us see what happens if we simulate random effects due to assessor. We will do that as follows: first we will generate random evaluation data using our `rand.data()` function, then we will randomly perturb it to simulate a second assessor that produces similar but not identical evaluation results.

```
> assess1 <- rand.data(mu=0.1)
> assess2 <- assess1 + rand.data(mu=0, var=0.1)
```

Plot `assess1` versus `assess2` to see how close they are. Then we can compare t-test results using the two assessors:

```
> table(assess1=apply(assess1, 2, t.test.sig), assess2=apply(assess2, 2, t.test.sig))
       assess2
assess1 FALSE TRUE
  FALSE  8825   78
  TRUE     83 1014
```

We find a small amount of disagreement, and the first assessor resulting in rejected H_0 slightly more often than the second. Increasing the amount of disagreement results in a more pronounced difference:

```
> assess2 <- assess1 + rand.data(mu=0, var=0.2)
> table(assess1=apply(assess1, 2, t.test.sig), assess2=apply(assess2, 2, t.test.sig))
       assess2
assess1 FALSE TRUE
  FALSE  8781  138
  TRUE    169  912
```

Thus assessor disagreement is probably something we should take into account when drawing conclusions about differences between systems.

We can incorporate assessor differences into a linear model. Suppose we have two systems, each evaluated over 50 topics, and for each topic we have two sets of judgments. This time, instead of generating differences from a normal distribution, we will generate values of an evaluation measure from an approximately uniform Beta distribution, with the first one skewed slightly to produce a mean of about 0.524 compare to a mean of 0.5 in the second (so population mean difference = 0.024). Then we will randomly perturb those values by sampling from a normal distribution.

```
> x.assess1 <- rbeta(50, 1.1, 1)
> y.assess1 <- rbeta(50, 1, 1)
> mean(x.assess1 - y.assess1)
0.08020204
> t.test(x.assess1 - y.assess1)$p.value
0.1726065
> x.assess2 <- x.assess1 + rnorm(50, 0, 0.1)
> y.assess2 <- y.assess1 + rnorm(50, 0, 0.1)
> mean(x.assess2 - y.assess2)
0.06951759
> t.test(x.assess2 - y.assess2)$p.value
0.2795553
```

These two systems are not "significantly different" under either assessor. The two assessors produce similar evaluation results, however.

Now create a data frame with four columns:

```
dat <- data.frame(m=c(x.assess1, x.assess2, y.assess1, y.assess2),
                  sys=c(rep("x", 100), rep("y", 100)),
                  topic=as.factor(1:50),
                  assess=c(rep("assess1",50),rep("assess2",50)))
```

Then fit a linear model to test the effect of system, assessor, and system/assessor interaction, with random errors due to topics and assessors[4]:

```
> summary(aov(m ~ sys + assess + sys*assess + Error(topic/assess), data=dat))
...
Error: Within
          Df Sum Sq  Mean Sq F value  Pr(>F)
sys        1 0.2802 0.280200  3.0292 0.08492 .
sys:ass    1 0.0014 0.001427  0.0154 0.90141
Residuals 98 9.0649 0.092499
```

Note the p-value for system effect is now 0.08. The fact that the two assessors roughly agreed on the mean difference gives more weight to the hypothesis that the systems really are different, as it should. When we repeat this 10,000 times with different simulated evaluation data each time, we find a p-value less than 0.05

[4]In fact there are several ways we could model assessor effect; this is just one possibility.

about 19% of the time, compared to $p < 0.05$ about 8% of the time when we just pool all experiments that are significant with assessor 1 or assessor 2 (or both).

This shows that assessor disagreement is another source of variance that can have a big effect on p-values; without having modeled it we should not take any p-value to have any deep meaning. Even when we do model it, there may be many other sources of variance we have not considered (or that we cannot model); if that is the case, why should we take *any* p-value to have meaning?

Multiple comparisons

We have shown that there are many factors that influence the calculation of p-values: choice of test, sample size, population effect size, data distribution, which effects we choose to model. Another factor that is extremely important and extremely simple, yet counterintuitive and difficult to model, is the *number* of experiments being done to test a single hypothesis. Essentially, as the number of experiments to test a true null hypothesis increases, the probability that at least one of them falsely reports significance increases.

At this point I will direct the reader to my ACM TOIS paper "Multiple Testing in Statistical Analysis of Systems-Based IR Experiments", particularly Section 2.2 on fitting models, Section 4 on the effect of multiple testing in TREC experiments (and the effect when adjusting for different conceptions of "single hypothesis"), Section 5 on the implications, and Appendix A for the R code.

Conclusions

The one message I would like readers to take away from this tutorial is:

> Never trust a p-value, even one you compute yourself.

There are simply too many decisions and factors that can affect the computation of a p-value. Furthermore, researchers are often completely unaware of many of the factors that can change results, and there are many decisions we do not think about because they have already been made when we start the test: sample size (in TREC scenarios), significance level ($\alpha = 0.05$), which sources of variance to model (only topic effects and a broadly-defined "system effect"), and how to model them (linearly and as additive effects). Furthermore, there are factors that we cannot ever hope to model, such as what experiments other researchers are currently doing. The only choice we make for ourselves, typically, is which test we use (i.e. the error distribution in the model), which in the grand scheme of things has a very small effect.

In the talk and above I mentioned that a significance test is essentially a binary classifier for the truth of H_0. In research, the truth of H_0 is rarely what we care about; rather, what we care about is how *interesting* it is if H_0 is false. In development too we do not care about the truth of H_0 as much as we care about what effect it has on users (and on our bottom line) if it is false. Significance tests cannot tell us much about that in either case. Like most AI technology, they are best used as an aide to human judgment rather than as a replacement for it.