Testing and Evaluation

Thomas Breuel

UniKL

Statistical Testing

In many areas of science and engineering, we are trying to answer questions like:

- Does a medical treatment have an effect?
- Is one population more susceptible to a disease than another population?
- Has something changed that causes people to get sicker?
- Is my new algorithm/method better than the old one?

Examples:

- Do sugary soft drinks cause obesity?
- Does a low-carb diet reduce obesity?
- Does living near a nuclear power plant cause cancer?
- Is gun control effective in reducing gun violence?
- Do health care subsidies lead to better health outcomes?
- Do school districts that spend more on education do better than the rest?

Usually, there are two questions / steps:

Correlation Does a change / difference in a controlled variable correlate with a change / difference in an observable?

Causation Is the change in the controlled variable the *cause* of the change in the observable?

Causation:

What does causation mean? It's different from correlation.

Causality is related to intervention. Variable X causes variable Y if intervening to change/fix only variable X (keeping everything else constant) correlates with a change in variable Y .

Causal calculus:

conditional probabilities: $P(\text{cancer}|\text{smoke})$

interventional probabilities: $P(\text{cancer}|\text{do}(\text{smoke}))$

Under some circumstances, we can infer interventional probabilities from conditional probabilities.

Remember:

Causality is complicated. Just remember that establishing an effect is not sufficient for establishing causation.

Here, we are concerned with establishing the presence of an effect.

Binomial Distributions

Assume we want to find out who the next president is going to become.

There is a large population of voters, a fraction of p voting for Obama, a fraction of $1 - p$ voting for Romney.

From this population, we take a subsample of $N = 1000$ voters.

Let's say $p = 0.51$.

Sampling Error:

Different polls get different results simply due to the fact that we are taking a finite, random sample from a large population.

This variation is referred to as the *sampling error*.

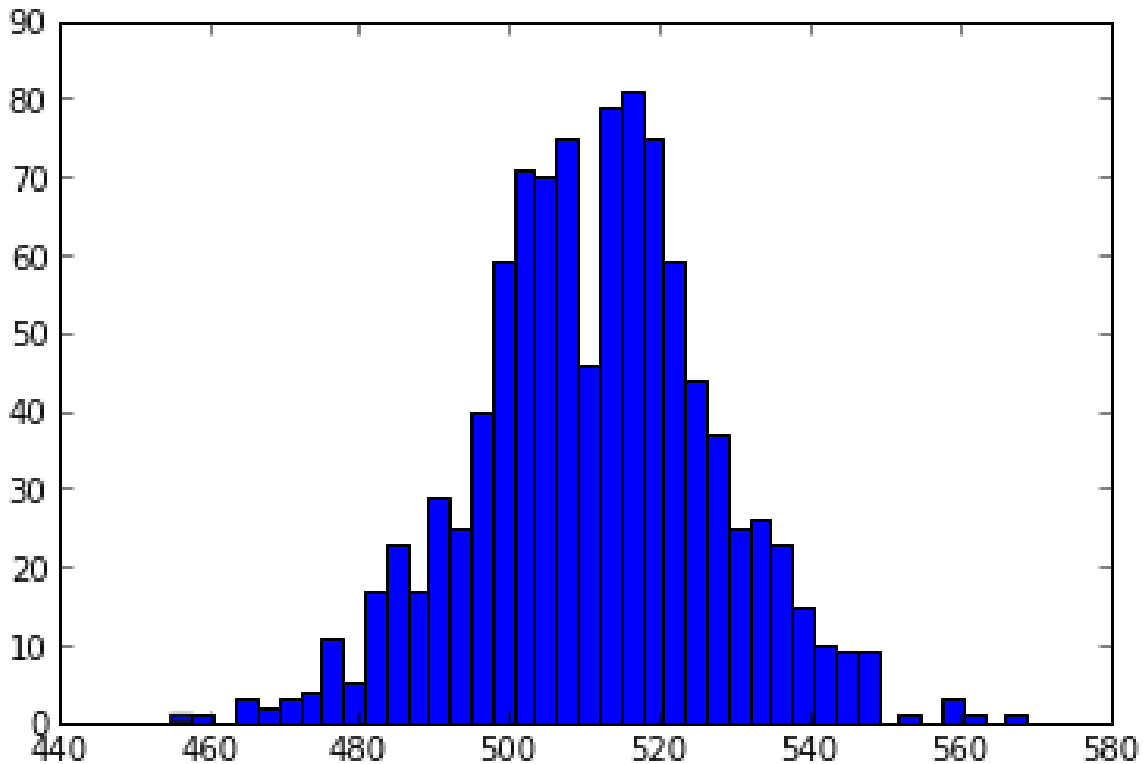
A thousand polls:

Let's perform a thousand polls with $p = 0.51$.

```
1 p = 0.51
2 polls = array([sum(rand(1000)<p) for i in range(1000)])
```

poll results

```
1 _=hist(polls,bins=40)
```



Each poll gives us an empirical estimate $\hat{p}$ of the probability p .

If $\hat{p} \leq 0.5$, we erroneously conclude that Obama loses, even though his actual $p \geq 0.5$.

The probability of making an incorrect decision is therefore:

```
1 sum(polls/1000.0<0.5)*1.0/len(polls)
```

```
0.218
```

This is a simulation-based estimate of the probability of error in our decision.

We can also make a frequentist and a Bayesian estimate.

Bayesian estimate:

Let n be the actual count of respondents. $\hat{p} = n/N$.

$$P(n|p) \sim B(N, p)$$

We are interested in the opposite question and can apply Bayes rule:

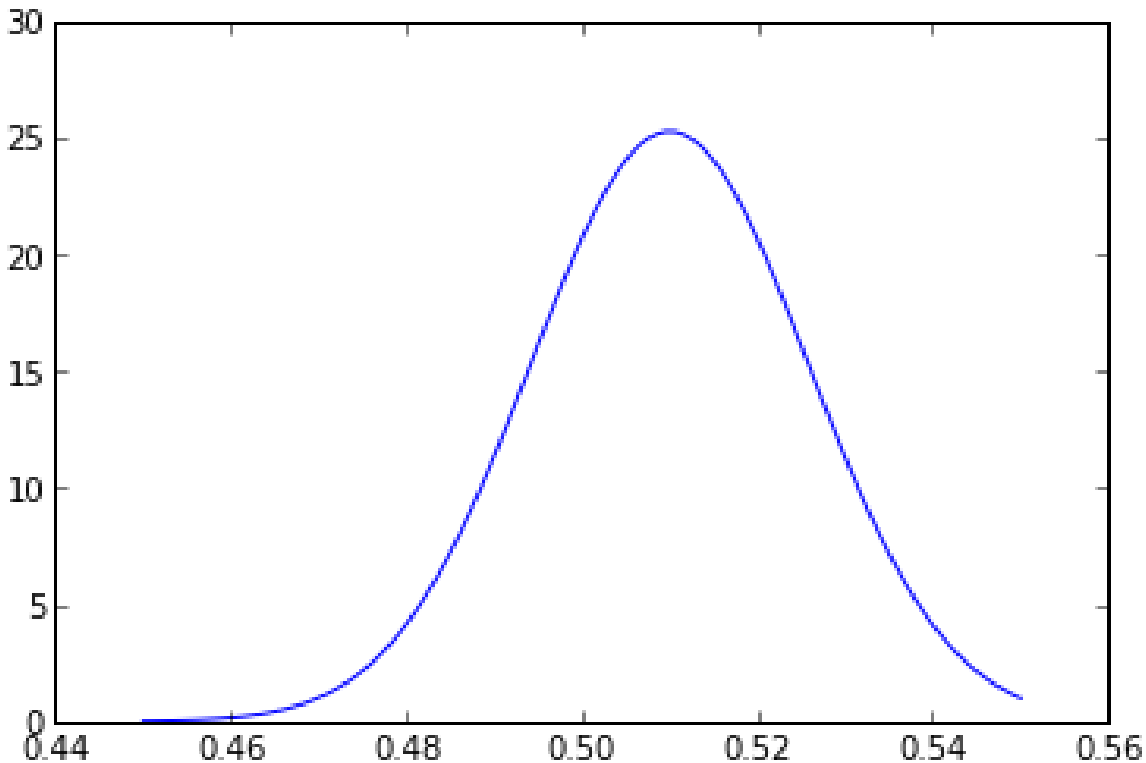
$$p(p|n) = \frac{P(n|p)p(p)}{P(n)}$$

This distribution is known as the Beta distribution:

$$p(p|n) = \text{Beta}(p; n, N - n)$$

distribution of p given n=510

```
1 from scipy.stats import beta
2 ps = linspace(0.45,0.55,100)
3 N = 1000; k = 510
4 bs = beta.pdf(ps,k+1,N-k+1)
5 plot(ps,bs)
```



Note that this is different from the original histogram:

- repeat experiment many times with $p = 0.51$
- given some unknown p , measure $n = 510$; what is the distribution of p ?

Note that we also assume a uniform prior.

Assuming we measure $n = 510$, what is the probability that $p \geq 0.5$, i.e. that Obama wins?

```
1 N = 1000; k = 510
2 1-beta.cdf(0.50,k+1,N-k+1)
```

```
0.736341802862815 03
```

Statistical Hypothesis Testing:

- formulate the research hypothesis
- state the null hypothesis (alternative)
- compute some function of the measured data, the test statistic
- derive the distribution of the test statistic under the null hypothesis
- select a significance level (e.g., 5%)
- reject the null hypothesis if the test statistic has a less than 5% probability of assuming its value under the test statistic

Statistical Hypothesis Testing:

- Obama wins, $p \geq 0.5$ (hypothesis)
- best case loss, $p = 0.5$ (null hypothesis)
- measure n (test statistic, same as for Bayesian in this case)
- $n \sim B(p, N)$ (distribution of the test statistic under the null hypothesis)
- significance level of 5%

Simple Statistical Hypothesis Test

If we do this test, note that we use the binomial distribution. We get a result similar to the Bayesian test, but it gives us a significance level (in this case, too low).

```
1 from scipy.stats import binom
2 print binom.cdf(510,1000,0.5)
3 print 1-binom.sf(510,1000,0.5)
```

```
0.746669978687
0.746669978687
```

Bayesian vs Frequentist Statements:

Bayesian Assuming a uniform prior on p , the probability that Obama wins is 73.6% given that 510 out of 1000 respondents answered they would vote for him.

Frequentist Given that 510 out of 1000 respondents answered that they would vote for him, we cannot conclude at the 5% significance level that he will win.

Statistical Hypothesis Testing:

- the test statistic need not be a complete description of the data
- the significance level is not a probability
- a significant result strongly suggest an effect
- lack of significance tells you nothing

A statistical significance test is not a Bayesian probability estimate.

Confidence Intervals:

Often, frequentists talk about *confidence intervals*.

These are ranges of a measurement such that the actual value is with high probability (e.g., 95%) within that range given the actual measurement.

For the Binomial distribution, a confidence interval based on the normal distribution is given by:

$$p \pm Z_c \sqrt{\frac{p(1-p)}{n}}$$

Here, Z_c is the confidence interval for the normal density; $Z_c \approx 2$ for a 95% confidence interval.

If we calculate the confidence interval, we see that $p = 0.5$ is contained in the confidence interval, therefore we cannot conclude with sufficient confidence which candidate is going to win:

$$\hat{p} = 0.51 \pm 2\sqrt{\frac{0.51(1-0.51)}{1000}} = 0.51 \pm 0.16$$

$$\hat{p} \in [0.494, 0.526] \text{ with probability } 95\%$$

Importance for Simulations, Machine Learning, Computer Science:

Many experiments in computer science are of the form:

Measure success/failure for a method on a set of inputs.

- What's the actual rate of success/failure?
- What's the probability that the actual failure rate is higher than some p ?

More binomial testing:

Many more experiments are of the form:

Measure success/failure for Method A vs Method B on a set of inputs.

How is this different?

- The hypothesis is $p_A p_B$ vs $p_A = p_B$.
- Input samples are the same, so the performance of the methods may be correlated.
- NB: the latter is usually not true in medical tests or biology; why?

Tests:

- unpaired tests (random samples, different methods)
- paired tests (similar samples, different methods)
- repeated measures tests (same sample, different methods)

Normal Distributions

Frequently, we measure some continuous variable depending on conditions:

- height vs vegetarianism
- body fat percentage vs low carb diet
- IQ vs eyewear

Assumptions:

- each dependent variable is normally distributed
- the variances of both the treated and untreated conditions are approximately the same
- we don't know either means or variances

Then, we are asking the question:

Given samples from $\mathcal{N}(\mu_1, \sigma)$ and $\mathcal{N}(\mu_2, \sigma)$:

- hypothesis: $\mu_1 \neq \mu_2$
- null hypothesis: $\mu_1 = \mu_2$

one sample t-test:

We don't know σ , so we need to estimate it as well.

This gives rise to the t -statistic. In the simplest case, we test whether the empirical mean $\hat{\mu}$ is the same as some given number μ_0 :

$$t = \frac{\hat{\mu} - \mu_0}{\hat{\sigma}/\sqrt{n}}$$

Here, $\hat{\sigma}$ is the sample standard distribution.

Note that if the samples are normally distributed, $\hat{\sigma}^2$ is χ -squared distributed.

two-sample t-tests:

There are many different two-sample t -test, depending on whether the sample sizes and variances are the same or different, and whether measurements are dependent or not.

Generally:

$$t = \frac{\hat{\mu}_1 - \hat{\mu}_2}{s(n_1^{-1} + n_2^{-1})^{1/2}}$$

$$s = \sqrt{\frac{(n_1 - 1)\hat{\sigma}_1^2 + (n_2 - 1)\hat{\sigma}_2^2}{n_1 + n_2 - 2}}$$

```
1 n = 100
2 s1 = randn(n)
3 s2 = randn(n)+0.1
4 print mean(s1),mean(s2)
```

```
-0.171941806685 0.219090745372
```

We can carry out a t-test using a library function in scipy:

```
1 from scipy.stats.mstats import ttest_ind
2 ttest_ind(s1,s2)
```

```
(array(-2.815419054455246),
 masked_array(data = 0.00536383874336,
              mask = False,
        fill_value = 1e+20)
)
```

Let's calculate everything by hand:

```
s = (((n-1)*var(s1)+(n-1)*var(s2))/(n+n-2))**.5
t = (abs(mean(s1)-mean(s2)))/(s*(n**-1+n**-1)**.5)
print "s␣=",s
print "t␣=",t
```

```
s = 0.977175267323
t = 2.82960261553
```

For a two-sided t -test, we need to add the two tails of the t distribution together to get the final p -value:

```
from scipy.stats import t as tdist
tdist.cdf(-t,2*n)+tdist.sf(t,2*n)
```

```
0.00513576747257955 13
```

paired samples:

The test for paired samples is actually particularly simple, since you just take a one-sample test of the differences of the paired results:

$$t = \frac{\hat{\mu}_\Delta}{\hat{\sigma} \cdot n^{-1/2}}$$

performing a t-test:

- calculate the sample means and variances
- compute the t statistic
- determine the significance level (p -value) from a table or calculator

Also:

- determine whether the variances are likely the same
- verify that the distributions are approximately normal
- normal rank plot
- Kolmogorov-Smirnov test

Bayesian view:

- compute $p(\mu_1|\text{data})$ and $p(\mu_2|\text{data})$
- compute $p(|\mu_1 - \mu_2| > \epsilon)$

Note:

- in a Bayesian framework, we must assume a prior
- (frequentist tests actually do the same thing, implicitly)
- it doesn't make sense to test for "equality", but we need to specify an effect size
- (frequentist test implicitly pick different ϵ for different N)

Non-Parametric Tests

All the tests discussed so far assume a particular kind of distribution.

What happens if we don't know the distribution?

We can use a non-parametric test.

Mann-Whitney U Test:

Determines whether two arbitrary distributions have the same median.

Procedure:

- for each sample in population 1, count the number of samples in population 2 that are smaller than it
- sum up the total number of ranks and call it U

For large sample sizes, U is approximately normally distributed:

$$m_U = \frac{n_1 n_2}{2}$$

$$\sigma_U = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}$$

Why don't we always use the U test?

If it shows a statistically significant result, you're fine.

If not, you don't know.

The U -test is weaker than a parametric test.

There is a library function performing the Mann-Whitney U test in scipy:

Note that this is a one-sided test, so we need to compare it to the one-sided t test:

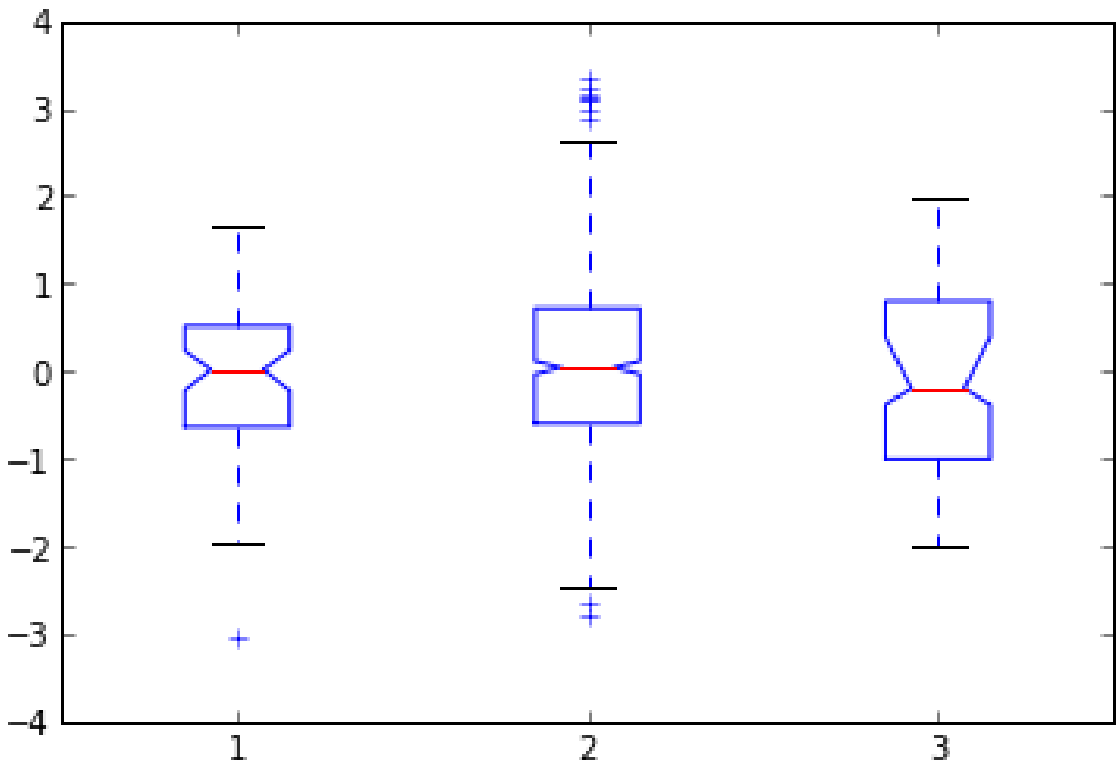
We see that the t test gives us a higher p value.

```
1 from scipy.stats import mannwhitneyu
2 print "one sided U-test",mannwhitneyu(s1,s2)
3 print "one sided t-test",tdist.sf(t,2*n)
```

```
one sided U-test (3990.0, 0.00682011117729674 96)
one sided t-test 0.00256788373629
```

Box Plots

```
1 from scipy.stats import cauchy,uniform
2 _=boxplot([randn(100),randn(1000),4*rand(100)-2],notch=1,bootstrap
      =10000)
```



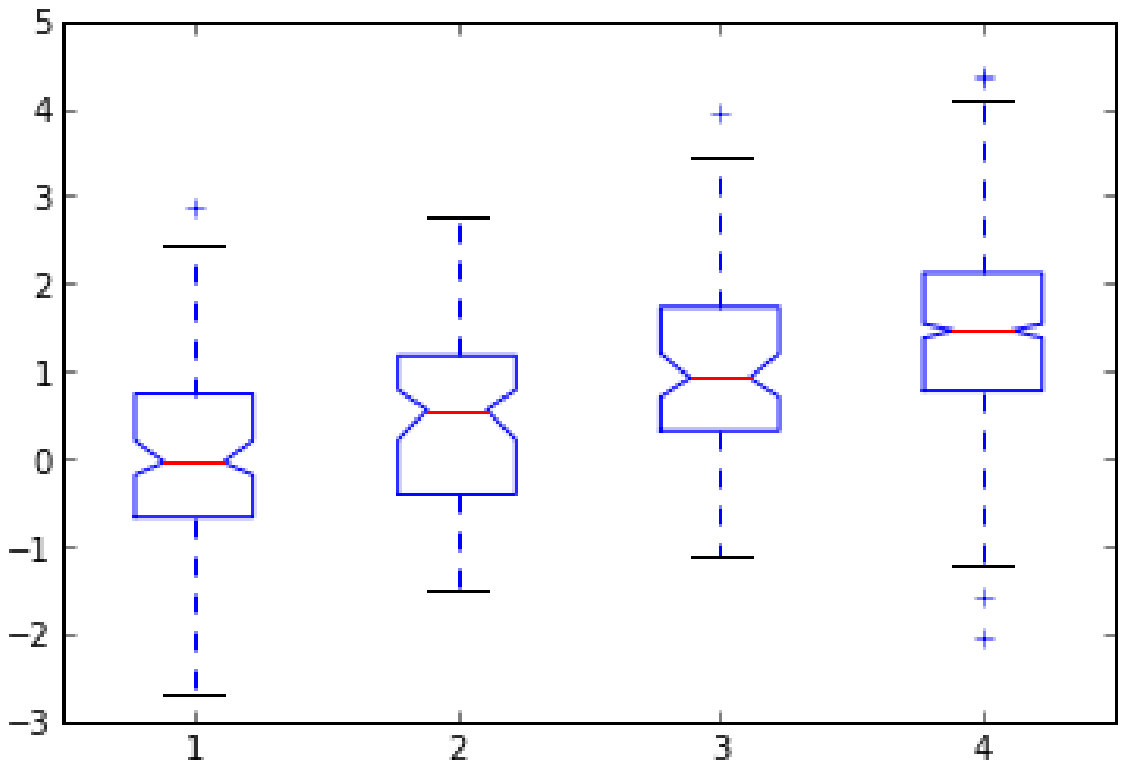
Components of the boxplot:

- red line shows median
- boxes show quartiles
- notches indicate 95
- whiskers go to most extreme point within 1.5 times outside the box
- fliers are all outliers beyond the whiskers

Notches allow quick nonparametric significance tests.

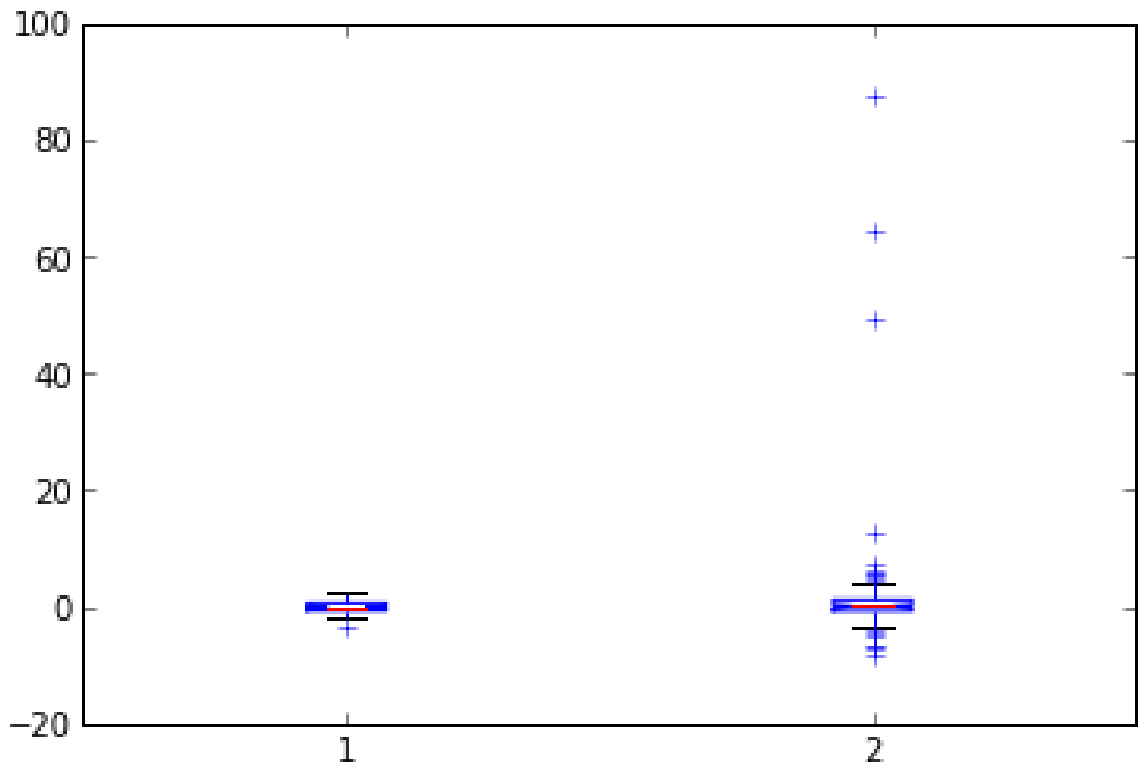
nonparametric signifiance test with boxplots

```
1 _=boxplot([randn(100),randn(100)+0.5,randn(100)+1.0,randn(1000)
      +1.5],notch=1,bootstrap=10000)
```



Note the large number of outliers with the Cauchy distribution.

```
cs = cauchy.rvs(size=100)
_=boxplot([randn(100),cs],notch=1)
```



Publication Bias

Experiment:

Does compound X improve performance on IQ tests?

Experiment: Give compound X to half of test takers.

Null hypothesis: It does not, the two distributions are the same.

- You do the experiment and don't get a significant result at the 5% level.
- Lack of a significant result proves nothing by itself. (Why?)
- You probably don't publish.

Research groups:

Assume compound X is a really important new drug, so 20 research groups all do this experiment.

What happens?

- Significance at the 5% level means 1:20 chance of getting a significant result by chance.
- One of the 20 research groups gets a significant result.
- The significant result gets published.

Press:

"Scientists have proven that compound X improves performance on IQ tests."

Machine learning:

Lots of groups try out some new learning algorithm on lots of datasets.

For each group and problem, numerous parameters are tried out to optimize algorithm performance.

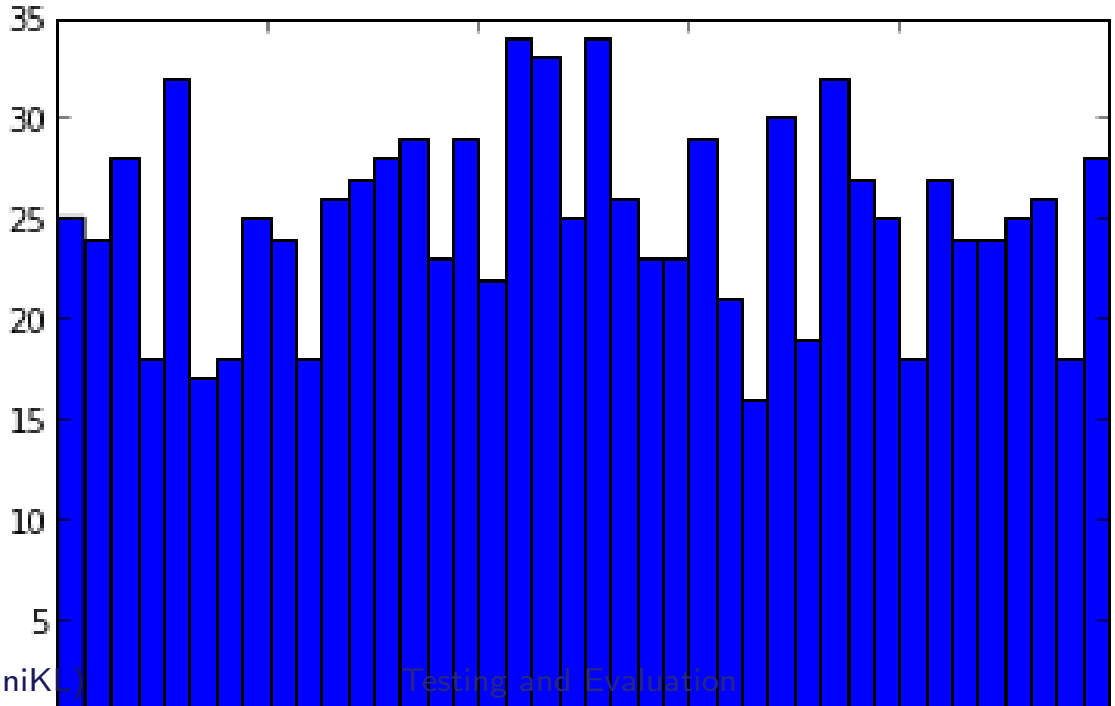
The old algorithm is simply run with standard parameters from the library.

Preferentially results get published that show that the new algorithm is better than old algorithms.

false high p-values in simulations

```
1 n = 10000
2 p_values = array([ttest_ind(randn(n),randn(n))[1] for i in range
      (1000)])
3 print sum(p_values<0.05)*1.0/len(p_values)
4 _=hist(p_values,bins=40)
```

0.047



Keep in mind:

Just because someone publishes a statistically significant result doesn't mean that the result is likely to be true.

Resampling Methods

As you can see, traditional statistical tests can be pretty tricky to apply, and Bayesian methods require complicated modeling.

Is there a better way?

Yes: resampling methods.

Bootstrap

Bootstrapping:

- estimate the distribution of an estimator
- construct hypothesis tests

How does it work?

- take the original sample of size N
- take from it another N samples, using sampling *with replacement*
- compute the statistic of interest on this new sample

Generate a poll result in which exactly 510 respondents answered YES

```
1 sample = array(linspace(0.0,1.0,1000)<0.51)
2 shuffle(sample)
3 sum(sample)*1.0/len(sample)
```

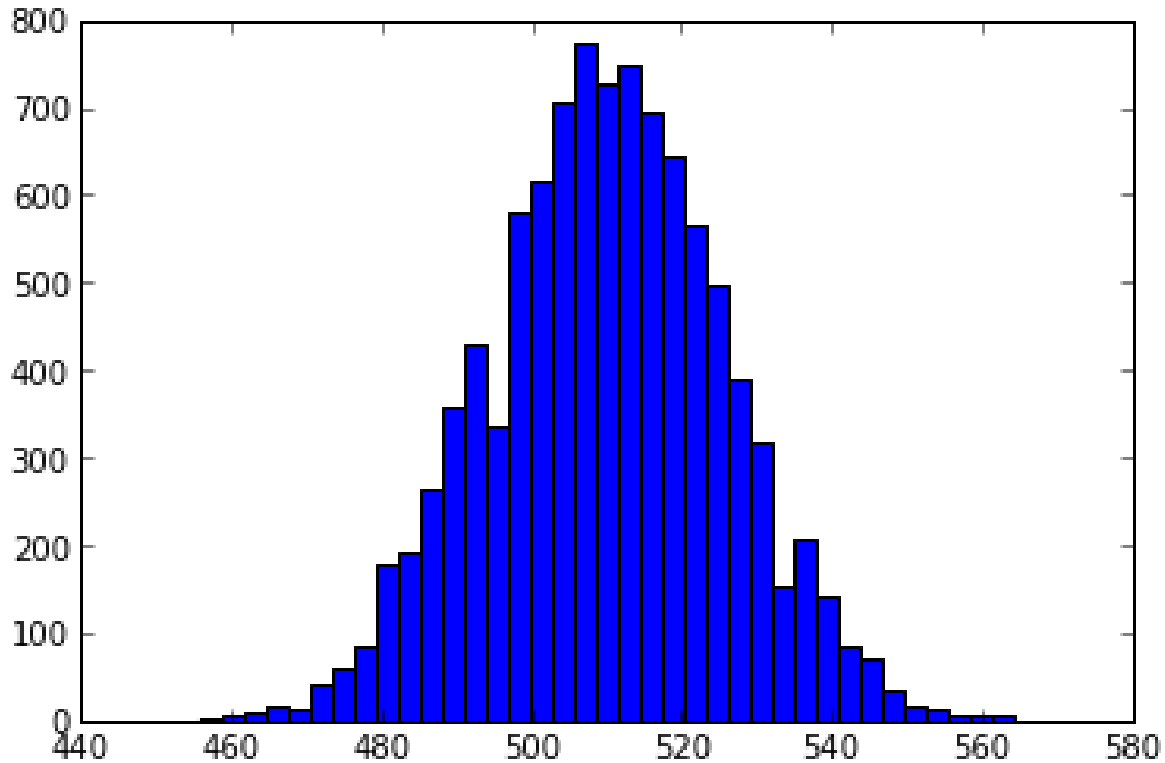
```
0.51000000000000001
```

Now perform bootstrap resampling and recompute the test statistic (just the number of votes).

```
from numpy.random import choice
counts = array([sum(choice(sample,size=1000,replace=True)) for i in
    range(10000)])
```

Let's look at the histogram again.

```
1 _=hist(counts,bins=40)
```



Finally, from this distribution, we can estimate the probability that Obama will lose given that 510 out of 1000 respondents actually said they would vote for him.

```
1 sum(counts/1000.0<0.5)*1.0/len(counts)
```

```
0.25779999999999997
```

Now that you know the principle, how would you apply this to the cases where we used parametric tests before?

Permutation Tests

Basic idea:

- We want to test whether two sample populations are "the same" with respect to some measurement.
- If they are the same, then it shouldn't matter (on average) whether we assign elements at random to one or the other population.

The original permutation test computes this over all permutations (i.e., all ways of assigning data to the two sets).

We can do something similar by random sampling, giving us a procedure similar to the bootstrap.

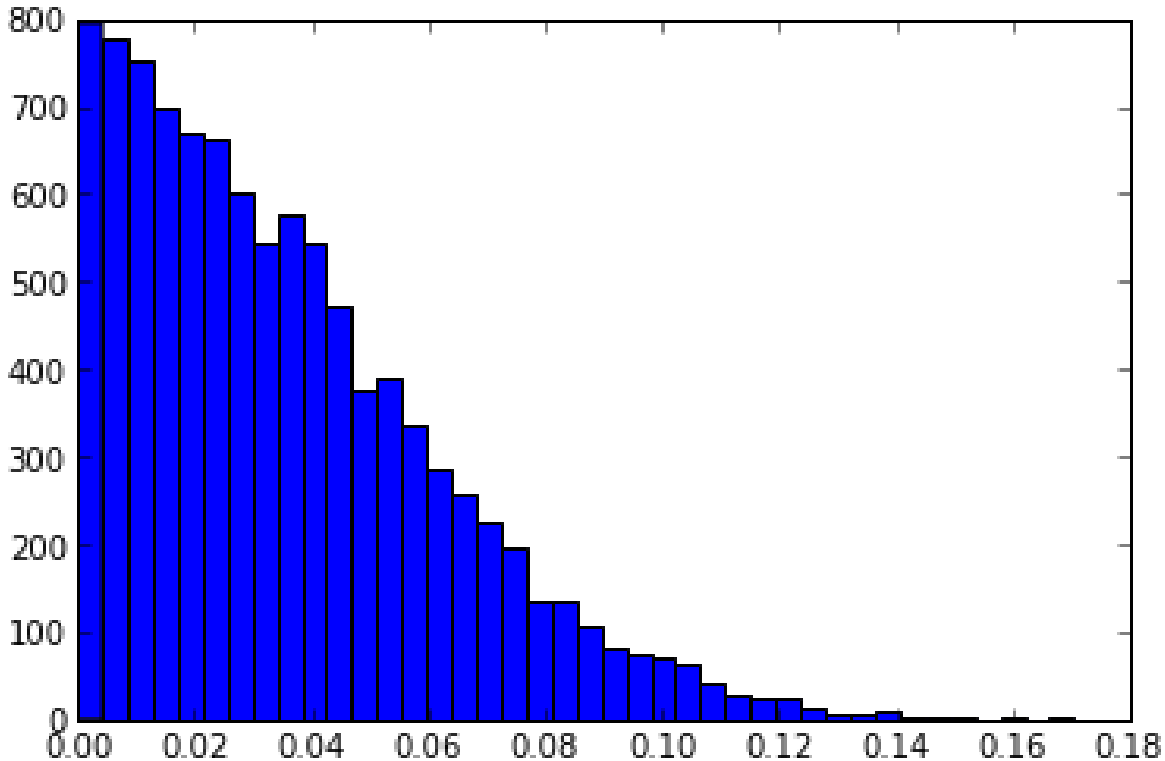
```
1 pop1 = randn(1000)
2 pop2 = randn(1000)+0.05
3 delta = abs(mean(pop1)-mean(pop2))
4 print mean(pop1),mean(pop2),delta
```

```
-0.0246806112386 0.0574570467328 0.0821376579714
```

Here, we use a simple resampling scheme. We get different permutations by shuffling the selector array.

```
pop = concatenate([pop1,pop2])
select = (arange(len(pop))>=len(pop1))
result = []
for i in range(10000):
    shuffle(select)
    samp1 = pop[select]
    samp2 = pop[select==0]
    result.append((abs(mean(samp1)-mean(samp2))))
result = array(result)
```

```
1 _=hist(result,bins=40)
```



```
1 sum(result>delta)*1.0/len(result)
```

```
0.06600000000000003
```

Cross-Validation

Cross-Validation:

We have already covered cross-validation in pattern recognition.

Cross-validation applies when we are building a *predictive model* (e.g., a classifier):

- divide the data set into a training and a test set
- train the predictive model on the training portion and measure its performance on the test set

The division is done as follows:

- $S = \biguplus_{i=1}^{n} S_i$ where $|S_i| \approx |S|/n$
- on run i , use $D = S - S_i$ for training and $T = S_i$ for testing

Caveats for cross-validation:

You could use resampling methods as well, but on each individual run, training and test sets must be disjoint.

Often, training samples are *correlated*. E.g., there are multiple inputs in a single font, or samples may even be repeated. In that case, you must make sure to account for those correlations in the split.