Week 13 - Lecture 25

István Albert
BMB, Bioinformatics
T-test for independent samples

• We have two sample populations

• We assume identical variance in the two populations

**Question:** what is the likelihood that the mean values are the identical for both populations
Independent T-test with scipy

```python
import utils
from scipy import stats

x = utils.create(10, 1, 100)
y = utils.create(10.5, 1, 100)

score, pval = stats.ttest_ind(x, y)
print "Different?", pval < 0.05, pval
```

Command Output:
```
Different? True 6.86757060938e-06
```
T-test for related samples

• Two samples, but the subjects in each sample are related (for example measuring the change before/after for each subject)

• We assume that the variances are the same

**Question:** what is the likelihood that the mean values are the same for both populations
import random
from scipy import stats

# SAT scores before and after training
before = [1611, 1854, 2021, 1700, 2223]
after = [1721, 1622, 2103, 1732, 2210]

score, pval = stats.ttest_rel(before, after)

print 'Score %.2f, pval=%.2g' % (score, pval)

Score 0.07, pval=0.95
Reminder on T-tests

```python
from scipy import stats

a = [1, 2, 3, 3, 3, 2, 1]
b = [1, 2, 2, 3, 2, 2, 1]

# the tests below assume equal variance

# t-test to compare the sample mean to population mean
score, pval = stats.ttest_1samp(a, 2)

# t-test to compare two independent sample means
score, pval = stats.ttest_ind(a, b)

# t-test to compare two related sample means
score, pval = stats.ttest_rel(a, b)
```
Test for normality – Kolmogorov-Smirnoff

Note that we now want large p-values! We want the null-hypothesis to be true.

```python
# Kolmogorov-Smirnov test for normality

Null hypothesis: the random variable has a normal distribution.

import utils, numpy
from scipy import stats

data = utils.create(0, 1, 100)

score, pval = stats.kstest(data, 'norm')

print "Data normal?", pval > 0.05, pval
```

Data normal? True 0.814701514417
Need to rescale if data not mean=0, sigma=1

```python
6 import utils, numpy
7 from scipy import stats
8 9 data = utils.create(10, 20, 100)
10 # transform data to have a mean=0 and sigma=1
11 m = numpy.mean(data)
12 s = numpy.std(data)
13
def transform(x):
    return (x-m)/s
15
16 data = map(transform, data)
17 score, pval = stats.kstest(data, 'norm')
18 print "Data normal?", pval > 0.05, pval
```

Data normal? True 0.9945587871
Two sample KS test

```python
""
Kolmogorov-Smirnov 2 sample test

Both samples were drawn from the same continuous distribution.
""

import utils, numpy
from scipy import stats

x = utils.create(10, 2, 100)
y = utils.create(10, 3, 100)

score, pval = stats.ks_2samp(x, y)
print "Same distribution?", pval > 0.05, pval
```

```
Same distribution? True 0.0690924348894
```
### RxC tables (Row x Column)

<table>
<thead>
<tr>
<th>Category</th>
<th>Diagnosis Positive</th>
<th>Diagnosis Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoker</td>
<td>60</td>
<td>300</td>
</tr>
<tr>
<td>General Population</td>
<td>40</td>
<td>390</td>
</tr>
</tbody>
</table>

- **categories**: Smoker, General Population
- **counts**: counts
Chi-Square test

```python
import numpy
from scipy import stats

observe = [60, 300]
expect = [40, 390]

observe = numpy.array(observe)
expect = numpy.array(expect)

# this test operates on numpy arrays only
score, pval = stats.chisquare(observe, expect)
print "Different?", pval < 0.05, pval
```

Command Output

```
Different? True 2.90609482007e-08
```
### Scipy reference

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cov(m, y, rowvar, bias)</code></td>
<td>Estimate the covariance matrix.</td>
</tr>
<tr>
<td><code>corrcosl(x, y, rowvar, bias)</code></td>
<td>The correlation coefficients formed from 2-d array x, where the rows are the observations, and the columns are variables.</td>
</tr>
<tr>
<td><code>f_oneway(*args)</code></td>
<td>Performs a 1-way ANOVA.</td>
</tr>
<tr>
<td><code>pearsonr(x, y)</code></td>
<td>Calculates a Pearson correlation coefficient and the p-value for testing.</td>
</tr>
<tr>
<td><code>spearmanr(x, y)</code></td>
<td>Calculates a Spearman rank-order correlation coefficient and the p-value.</td>
</tr>
<tr>
<td><code>pointbiserialr(x, y)</code></td>
<td>Calculates a point biserial correlation coefficient and the associated p-value.</td>
</tr>
<tr>
<td><code>kendalltau(x, y)</code></td>
<td>Calculates Kendall’s tau, a correlation measure for ordinal data.</td>
</tr>
<tr>
<td><code>linregress(*args)</code></td>
<td>Calculate a regression line.</td>
</tr>
<tr>
<td><code>ttest_1samp(a, popmean[, axis])</code></td>
<td>Calculates the T-test for the mean of ONE group of scores a.</td>
</tr>
<tr>
<td><code>ttest_ind(a, b[, axis])</code></td>
<td>Calculates the T-test for the means of TWO INDEPENDENT samples of scores.</td>
</tr>
<tr>
<td><code>ttest_rel(a, b[, axis])</code></td>
<td>Calculates the T-test on TWO RELATED samples of scores, a and b.</td>
</tr>
<tr>
<td><code>ks_2samp(data1, data2)</code></td>
<td>Performs the Kolmogorov-Smirnov test for goodness of fit.</td>
</tr>
<tr>
<td><code>mannwhitneyu(x, y, use_continuity)</code></td>
<td>Computes the Mann-Whitney rank test on samples x and y.</td>
</tr>
<tr>
<td><code>tiecorrect(rankvals)</code></td>
<td>Tie-corrector for ties in Mann Whitney U and Kruskal Wallis H tests.</td>
</tr>
<tr>
<td><code>ranksums(x, y)</code></td>
<td>Compute the Wilcoxon rank-sum statistic for two samples.</td>
</tr>
<tr>
<td><code>wilcoxon(x[, y])</code></td>
<td>Calculate the Wilcoxon signed-rank test.</td>
</tr>
<tr>
<td><code>kruskal(*args)</code></td>
<td>Compute the Kruskal-Wallis H-test for independent samples.</td>
</tr>
<tr>
<td><code>friedmanchisquare(*args)</code></td>
<td>Computes the Friedman test for repeated measurements.</td>
</tr>
<tr>
<td><code>ansari(x, y)</code></td>
<td>Perform the Ansari-Bradley test for equal scale parameters</td>
</tr>
<tr>
<td><code>bartlett(*args)</code></td>
<td>Perform Bartlett’s test for equal variances.</td>
</tr>
<tr>
<td><code>levene(*args, **kwds)</code></td>
<td>Perform Levene test for equal variances.</td>
</tr>
<tr>
<td><code>shapiro(x[, a, reta])</code></td>
<td>Perform the Shapiro-Wilk test for normality.</td>
</tr>
<tr>
<td><code>anderson(x[, dist])</code></td>
<td>Anderson-Darling test for data coming from a particular distribution</td>
</tr>
<tr>
<td><code>binom_test(x[, n, p])</code></td>
<td>Perform a test that the probability of success is p.</td>
</tr>
<tr>
<td><code>fligner(*args, **kwds)</code></td>
<td>Perform Fligner’s test for equal variances.</td>
</tr>
<tr>
<td><code>mood(x, y)</code></td>
<td>Perform Mood’s test for equal scale parameters.</td>
</tr>
<tr>
<td><code>oneway(*args, **kwds)</code></td>
<td>Test for equal means in two or more samples from the normal distribution.</td>
</tr>
<tr>
<td><code>glm(data, para)</code></td>
<td>Calculates a linear model fit.</td>
</tr>
</tbody>
</table>
Homework 25 (part 1)

Generate two lists with random numbers one with a normal distribution (mean=0, sigma=1) the other with a uniform random distribution in the range=(0, 1)

• What is the minimal sample size that can reliably distinguish between the two distributions?
Homework 25 (part 2)

Fitness levels for football players and ballet dancers are:

football = [ 89.2, 78.2, 89.3, 88.3, 87.3, 90.1 ]
ballet = [ 79.3, 78.3, 85.3, 91.2, 93.3, 79.9 ]

Is there a difference between the fitness levels?
Performance after motivational speaking

before = [ 66, 67, 65, 68, 69, 70 ]
after  = [ 65, 75, 80, 77, 74, 69 ]

Did the motivational speaking improve the scores?
We assume that a disease is inherited as an autosomal dominant trait (present 50% of the time) in immediate descendants.

Out of 40 patients with parents having the disease 14 have the disease.

What is the likelihood that our assumption is correct.
Week 13 - Lecture 26

István Albert
BMB, Bioinformatics
Numerical Python

• Developed to make number crunching faster and more seamless

• Provides a Python interface to very fast numerical libraries

• Use minimal memory but can only represent homogenous data structures → everything is an integers, floats etc

• NumPy's main object is the homogeneous multidimensional array
Full control over the data sizes!

```python
import numpy, time

# in your code start with
# a smaller number 1 million
N = 10 ** 8
# x = range(N)
# x = numpy.ones(N)
x = numpy.ones(N, numpy.int8)
time.sleep(10)
print 'Done'
```

Limit the size to $2^{**}8 = 256$ (integers from -128 to 127)
Basic constructs

```python
start, stop = 0, 1

# initialize an array to zeroes or ones
print numpy.zeros(5)
print numpy.ones(5)

# a range with floating point steps
print numpy.arange(start, stop, 0.25)

# divide range into N segments
print numpy.linspace(start, stop, 6)
```

```
[ 0.  0.  0.  0.  0.]
[ 1.  1.  1.  1.  1.]
[ 0.  0.25 0.5  0.75]
[ 0.  0.2  0.4  0.6  0.8  1.]
```
Multi dimensional arrays

```python
import numpy

d = [1, 2, 3, 4, 5, 6, 7, 8]
e = numpy.array(d)
print e
print e[3]

# multidimensional arrays --> rows x columns
f = e.reshape(2, 4)
print f
print f[1,2]
```

```
[[1 2 3 4]
 [5 6 7 8]]
```
Array attributes/methods

```python
import numpy

e = numpy.zeros(10)

print(dir(e))
```

Output:

Operations for each axis (row, column)

```python
import numpy

x = numpy.array( range(10) )
x = x.reshape(2, 5)

print x

# operations may be
# performed on rows or columns

print numpy.mean(x)
print numpy.mean(x, axis=0)
print numpy.mean(x, axis=1)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
4.5
[ 2.5 3.5 4.5 5.5 6.5]
[ 2.  7.]
```
Arithmetic operations happen element-wise

```python
import numpy

N = 10 ** 6

data = numpy.zeros(N)
print(data)

data = data + 100
print(data)
```

```
[ 0.  0.  0. ...,  0.  0.  0.]
[ 100. 100. 100. ..., 100. 100. 100.]
```
Cumulative sum

```python
import numpy

x = numpy.array( range(9) )
x = x.reshape(3, 3)

print x

# cumulative sums
print x.cumsum()
print x.cumsum(axis=0)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]]

[[ 0  1  3  6 10 15 21 28 36]]
```

Universal functions

```python
import numpy

# use the mathematical functions
# provided by numpy to broadcast over all elements
a = numpy.array([1, 4, 9])
b = numpy.sqrt(a)
c = a + b

print a
print b
print c
```

```
[1 4 9]
[1. 2. 3.]
[2. 6. 12.]
```
import timeit
setup = ""
import numpy, math
N = 10 ** 6
data1 = xrange(N)
data2 = numpy.array(data1)
""

code1 = "z = map(math.sqrt, data1)"
code2 = "z = numpy.sqrt(data2)"

print timeit.timeit(code1, setup=setup, number=10)
print timeit.timeit(code2, setup=setup, number=10)
import numpy

a = numpy.array([10, 20, 0, 20, 10])

# prints the indices where the values are not zero
print a.nonzero()

# boolean operators also return numpy arrays
print a > 15

# finds the indices where the condition is true
print (a > 15).nonzero()
A typical workflow for chromosome scale computations

• Read in the genomic coordinates in a file

• Create a numpy array that corresponds to the genomic regions of interest

• Populate the array with values from the file

• Use simply numpy commands to answer various questions
You will need to filter it to keep only chromosome 4
Create an array representation of the chromosome

```python
values = numpy.zeros(5*10**6, numpy.int16)

# we want to add 1 to every index that spans a feature
for chrom, start, end in all:
    # the file is one based, the array is zero based
    start, end = int(start) - 1, int(end)
    addone = numpy.ones(end-start, numpy.int16)
    values[start:end] += addone

print values[:10]
```

```
[3 3 3 3 3 3 3 3 3 3]
```
Homework 26

The answer to each of these questions is a one or two lines of code!

• Based on this file what is the maximum number of features that cover a base in chromosome 4?

• How many locations (bases) have a coverage that is more than half of the maximum?

• Print the first 10 genomic indices that have more than half of this coverage