joepy

Monday, March 25, 2013

Multiple Comparison and Tukey HSD or why statsmodels is awful

Introduction

Statistical tests are often grouped into one-sample, two-sample and k-sample tests, depending on how many samples are involved in the test. In k-sample tests the usual Null hypothesis is that a statistic, for example the mean as in a one-way ANOVA, or the distribution in goodness-of-fit tests, is the same in all groups or samples. The common test is the joint test that all samples have the same value, against the alternative that at least one sample or group has a different value.

However, often we are not just interested in the joint hypothesis if all samples are the same, but we would also like to know for which pairs of samples the hypothesis of equal values is rejected. In this case we conduct several tests at the same time, one test for each pair of samples.

This results, as a consequence, in a multiple testing problem and we should correct our test distribution or p-values to account for this.

I mentioned some of the one- and two sample test in statsmodels before. Today, I just want to look at pairwise comparison of means. We have k samples and we want to test for each pair whether the mean is the same, or not.

Instead of adding more explanations here, I just want to point to R tutorial and also the brief description on Wikipedia. A search for "Tukey HSD" or multiple comparison on the internet will find many tutorials and explanations.

The following are examples in statsmodels and R interspersed with a few explanatory comments.

The Data

To make it simple, I just define the data as a numpy rec.array, and add some imports.

```
import numpy as np
from scipy import stats

from statsmodels.stats.multicomp import (pairwise_tukeyhsd,
                                         MultiComparison)


dta2 = np.rec.array([
(  1,   'mental',  2 ),
(  2,   'mental',  2 ),
(  3,   'mental',  3 ),
(  4,   'mental',  4 ),
(  5,   'mental',  4 ),
(  6,   'mental',  5 ),
(  7,   'mental',  3 ),
(  8,   'mental',  4 ),
(  9,   'mental',  4 ),
( 10,   'mental',  4 ),
( 11, 'physical',  4 ),
( 12, 'physical',  4 ),
( 13, 'physical',  3 ),
( 14, 'physical',  5 ),
( 15, 'physical',  4 ),
( 16, 'physical',  1 ),
( 17, 'physical',  1 ),
( 18, 'physical',  2 ),
( 19, 'physical',  3 ),
( 20, 'physical',  3 ),
( 21,  'medical',  1 ),
( 22,  'medical',  2 ),
( 23,  'medical',  2 ),
( 24,  'medical',  2 ),
( 25,  'medical',  3 ),
( 26,  'medical',  2 ),
( 27,  'medical',  3 ),
( 28,  'medical',  1 ),
( 29,  'medical',  3 ),
( 30,  'medical',  1 )], dtype=[('idx', '<i4'),
                                ('Treatment', '|S8'),
                                ('StressReduction', '<i4')])
```

Part 1: Tukey's HSD or studentized range statistic

Jumping right in, Tukey's studentized range test is a popular test with good statistical properties for the comparison of all pairs of means with k samples. See Wikipedia for more details.

In the example we have three different treatments, and we want to test whether the differences in means for the three pairs of treatments are statistically different.

Running the test in statsmodels

```
res2 = pairwise_tukeyhsd(dta2['StressReduction'], dta2['Treatment'])
print res2[0]

mod = MultiComparison(dta2['StressReduction'], dta2['Treatment'])
print mod.tukeyhsd()[0]
```

both print the following

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================
group1 group2 meandiff  lower  upper  reject
---------------------------------------------
  0      1      1.5     0.3217 2.6783  True
  0      2      1.0    -0.1783 2.1783 False
  1      2     -0.5    -1.6783 0.6783 False
---------------------------------------------
```

The group labels, for 0, 1, 2 respectively, are

```
>>> mod.groupsunique
rec.array(['medical', 'mental', 'physical'],
      dtype='|S8')
```

From the result, we can see that we reject the hypothesis that the zero and first treatment, that is medical and mental, have the same mean, but we cannot reject either of the other two pairs.

The following is mostly a comparison with R, and also illustrates where the implementation in statsmodels is lacking. The multiple comparisons and tukeyhsd were initially written based on a SAS examples from the internet or SAS documentation. Some of the unit tests are written against R.

The output that corresponds to the above is in R

```
> options(digits=4)
> TukeyHSD(aov(StressReduction ~ Treatment, dta))
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = StressReduction ~ Treatment, data = dta)

$Treatment
                 diff     lwr    upr  p adj
mental-medical    1.5  0.3215 2.6785 0.0106
physical-medical  1.0 -0.1785 2.1785 0.1079
physical-mental  -0.5 -1.6785 0.6785 0.5514
```

Comparing my confidence interval with those of R, we can see that there is a small difference, that most likely comes from the precision of the distribution of the studentized range statistic, which is not a standard distribution and is not implemented to a very high precision.

```
>>> res2[1][4] - treatment[:, 1:3]
array([[ 0.00022057, -0.00022057],
       [ 0.00022057, -0.00022057],
       [ 0.00022057, -0.00022057]])
```

The output of `tukeyhsd` in statsmodels is a bit hard to digest (formatted for line breaks):

```
>>> res2
(<class 'statsmodels.iolib.table.SimpleTable'>, ((array([0, 0, 1]),
       array([1, 2, 2])), array([ True, False, False], dtype=bool),
       array([ 1.5,  1. , -0.5]), array([ 0.33609963,  0.33609963,  0.33609963]),
       array([[ 0.32171204,  2.67828796],
       [-0.17828796,  2.17828796],
       [-1.67828796,  0.67828796]]), 3.5057698487864877, 27,
       array([ True, False, False], dtype=bool)))
```

Statsmodels doesn't provide p-values for tukeyhsd, so let's try to partially reverse engineer them. The returned results from tukeyhsd include the mean difference and the standard deviation for the studentized range statistic. Statsmodels also includes a function for the pvalue of the studentized range test written by Roger Lew .

```
>>> from statsmodels.stats.libqsturng import psturng
>>> studentized range statistic
>>> rs = res2[1][2] / res2[1][3]

>>> pvalues = psturng(np.abs(rs), 3, 27)
>>> pvalues
array([ 0.01054871,  0.10790278,  0.54980308])
```

difference to R

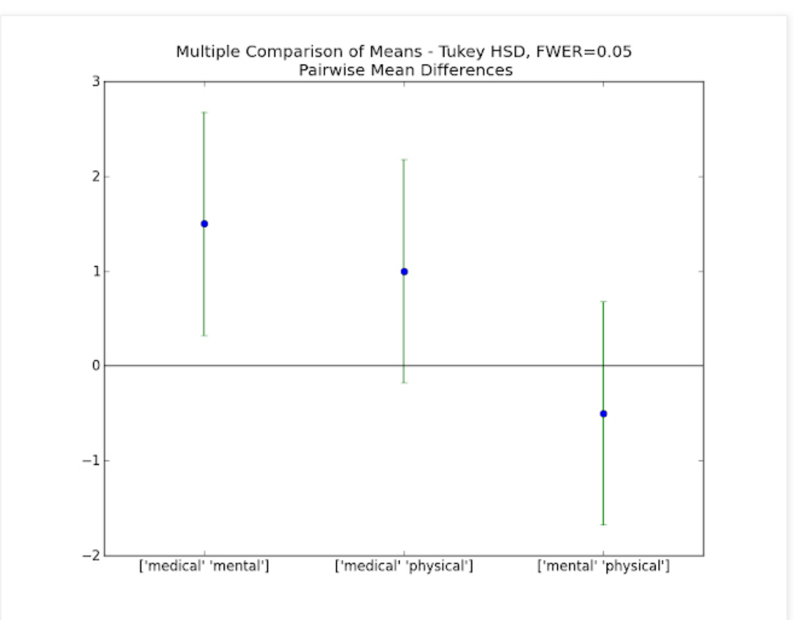
```
>>> pvalues - treatment[:, -1]
array([-0.00001409, -0.00000068, -0.00158736])
```

Now, where are the plots?

A quick plot for now

```
import matplotlib.pyplot as plt
plt.plot([0,1,2], res2[1][2], 'o')
plt.errorbar([0,1,2], res2[1][2], yerr=np.abs(res2[1][4].T-res2[1][2]), ls='o')
xlim = -0.5, 2.5
plt.hlines(0, *xlim)
plt.xlim(*xlim)
pair_labels = mod.groupsunique[np.column_stack(res2[1][0])]
plt.xticks([0,1,2], pair_labels)
plt.title('Multiple Comparison of Means - Tukey HSD, FWER=0.05' +
          '\n Pairwise Mean Differences')
```

which produces:



The first confidence interval of the difference in means does not include zero, so it is statistically different. For the two other pairs, we cannot reject the hypothesis that the difference is zero.

Part 2: Pairwise T-tests

Tukey's studentized range test (HSD) is a test specific to the comparison of all pairs of k independent samples. Instead we can run t-tests on all pairs, calculate the p-values and apply one of the p-value corrections for multiple testing problems.

Both statsmodels and R have several options to adjust the p-values. statsmodels has among others false discovery rate corrections *fdr_bh* (Benjamini/Hochberg) and *fdr_by* : Benjamini/Yekutieli. In the following I will mainly show Holm and Bonferroni adjustments.

Paired Samples

In this case the assumption is that samples are paired, for example when each individual goes through all three treatments.

Note: I'm using the data just as illustration. I did not check whether it would actually make sense to do this with this dataset.

The command and result in R for Holm p-value correction are

```
> pairwise.t.test(dta$StressReduction, dta$Treatment, p.adj = "holm", paired=TRUE)

        Pairwise comparisons using paired t tests

data:  dta$StressReduction and dta$Treatment

         medical mental
mental   0.009   -
physical 0.170   0.427

P value adjustment method: holm
```

and in statsmodels

```
>>> from scipy import stats
>>> rtp = mod.allpairtest(stats.ttest_rel, method='Holm')
>>> print rtp[0]
Test Multiple Comparison ttest_rel
FWER=0.05 method=Holm
alphacSidak=0.02, alphacBonf=0.017
=============================================
group1 group2   stat   pval  pval_corr reject
---------------------------------------------
  0      1    -4.0249 0.003    0.009    True
  0      2    -1.9365 0.0848   0.1696  False
  1      2     0.8321 0.4269   0.4269  False
---------------------------------------------
```

Note, that I'm reusing the `Multicomparison` instance that I created for the `` `tukeyhsd `` test.

Using Bonferroni p-value correction the results in R are

```
> pairwise.t.test(dta$StressReduction, dta$Treatment, p.adj = "bo", paired=TRUE)

        Pairwise comparisons using paired t tests

data:  dta$StressReduction and dta$Treatment

         medical mental
mental   0.009   -
physical 0.254   1.000

P value adjustment method: bonferroni
```

and in statsmodels

```
>>> print mod.allpairtest(stats.ttest_rel, method='b')[0]
Test Multiple Comparison ttest_rel
FWER=0.05 method=b
alphacSidak=0.02, alphacBonf=0.017
=============================================
group1 group2   stat   pval  pval_corr reject
---------------------------------------------
  0      1    -4.0249 0.003    0.009    True
  0      2    -1.9365 0.0848   0.2544  False
  1      2     0.8321 0.4269    1.0    False
---------------------------------------------
```

The pvalues returned by R and statsmodels agree at several decimals. However, R doesn't return anything besides p-values

Independent Samples

I show the result of doing the same as before, i.e. use the two-sample test function and apply it to each pair of samples.

Using statsmodels, I get with Bonferroni correction

```
>>> print mod.allpairtest(stats.ttest_ind, method='b')[0]
Test Multiple Comparison ttest_ind
FWER=0.05 method=b
alphacSidak=0.02, alphacBonf=0.017
=============================================
group1 group2   stat   pval  pval_corr reject
---------------------------------------------
  0      1     -3.737 0.0015   0.0045   True
  0      2    -2.0226 0.0582   0.1747  False
  1      2     0.9583 0.3506    1.0    False
---------------------------------------------
```

and in R

```
> tr = pairwise.t.test(dta$StressReduction, dta$Treatment, p.adj = "b",
                       paired=FALSE)
> tr

        Pairwise comparisons using t tests with pooled SD

data:  dta$StressReduction and dta$Treatment

         medical mental
mental   0.012   -
physical 0.135   0.906

P value adjustment method: bonferroni
```

Now that looks pretty different. What's going on? A bug, different tests?

We can print higher precision results, so we can compare in more details. I'm also printing the pvalues without multiple testing correction to find the difference between R and statsmodels.

```
> options(digits=17)
> as.numeric(tr$p.value)
[1] 0.01172540911560978 0.13452902413031653  NA 0.90646656753379151
> tr = pairwise.t.test(dta$StressReduction, dta$Treatment, p.adj = "none", paired=FALSE)
> as.numeric(tr$p.value)
[1] 0.003908469705203262 0.044843008043438846  NA 0.302155522511263819
```

The answer is that the pairwise.ttest for independent samples in R, as well as the Tukey HSD test in both packages, use the joint variance across all samples, while the pairwise ttest calculates the joint variance estimate for each pair of sample separately. `stats.ttest_ind` just looks at one pair at a time.

Now, we could calculate a pairwise ttest that takes a joint variance as given and feed it to `mod.allpairtest`. However, since this is already getting long, I just take a shortcut.

`tukeyhsd` returned the mean and the variance for the studentized range statistic. The studentized range statistic is the same as the t-statistic except for a scaling factor (`np.sqrt(2)`).

```
>>> t_stat = res2[1][2] / res2[1][3] / np.sqrt(2)
>>> print t_stat
[ 3.15579093  2.10386062 -1.05193031]

>>> my_pvalues = stats.t.sf(np.abs(t_stat), 27) * 2   #two-sided
>>> my_pvalues
array([ 0.00390847,  0.04484301,  0.30215552])
```

and now the difference for the uncorrected p-values between the two packages is

```
>>> r_pvalues = np.array([0.003908469705203262, 0.044843008043438846,
...                       0.302155522511263819])
>>> list(my_pvalues - r_pvalues)
[6.0715321659188248e-18, 1.3877787807814457e-17, -2.7755575615628914e-16]
```

Note: I'm just using the conversion to list as a cheap way to increase print precision temporarily.

We can also infer the t-statistic from the R pvalues (since R doesn't give us the t-statistic directly)

```
>>> print stats.t.isf(r_pvalues/2, 27)
[ 3.15579093  2.10386062  1.05193031]
```

Again, this is very close between my result and those of R.

Once we have the uncorrected p-values, we can just do a multiple testing p-value correction, for example for Bonferroni correction, we get

```
>>> from statsmodels.stats.multitest import multipletests
>>> res_b = multipletests(my_pvalues, method='b')

>>> r_pvalues_b = np.array([0.01172540911560978, 0.13452902413031653,
...                         0.90646656753379151])
>>> list(res_b[1] - r_pvalues_b)
[2.2551405187698492e-17, 5.5511151231257827e-17, -8.8817841970012523e-16]
```

Just to verify another case, we can also get false discovery rate correction

```
>>> res_fdr = multipletests(my_pvalues, method='fdr_bh')

>>> r_pvalues_fdr = np.array([0.01172540911560978, 0.06726451206515827, 0.30215552251126382])
>>> list(res_fdr[1] - r_pvalues_fdr)
[2.4286128663675299e-17, 2.7755575615628914e-17, -2.7755575615628914e-16]
```

Finally

By the time I wrote this blog article, I could have written a pull request for improving this. However, I was in a bit of a grumpy mood, and I thought I join the "statsmodels is awful" theme.

What got me started on this story is that I was browsing the htmlhelp for current master and saw this . As it turned out, there is the wrong function included in the documentation, `tukeyhsd` instead of `pairwise_tukeyhsd`.

Josef Perktold at 8:47 PM

Share

4 comments:

Tyler April 16, 2013 at 11:14 AM

This is super interesting. I have the requisite grad school minimum of a stats background, but don't have the understanding to code up my own hsd. I desperately need a tukey hsd in python that I can trust. Matlab has an outstanding implementation (http://www.mathworks.com/help/stats/multcompare.html) that I had gotten accustomed to, but now that my entire workflow lives in python I'd very much like to ween myself off it (but want something similar enough).

From reading your article, it sounds like you're saying that statsmodels is accurate (as compared to R), but just not very user-friendly. Would you agree? Any pending/necessary changes? If statsmodels is mathematically sound, it would be great to either modify the source to be a little prettier, and even have some native plotting functionality ala your plot and Matlab. At least we could write a nice python wrapper to do the dirty work. I might try the second option.

I was astounded to not find tukey hsd in scipy or other major modules, and to only find this wonky-to-use function in statsmodels, given its popularity in my field. Thanks for the down and dirty explanation.

Reply

Josef Perktold April 16, 2013 at 1:15 PM

The main improvement that pairwise_tukeyhsd would need is to return a results instance, where we can attach the results, the string output and the plots, instead of returning the list of heterogenous results.

I don't have any changes pending, I plan to do some cleanup, but in the meantime I still have multipletesting to finish, and I would like to get some parts done for equivalence testing (before I forget what I read in my latest set of readings).

as a related aside: In the meantime I found https://pypi.python.org/pypi/pyvttbl/0.5.2.2 by Roger Lew which has some of the same functions and more on repeated measures.

Reply

Tyler April 24, 2013 at 2:31 PM

So i've taken a hack at implementing the nifty multcompare figure that Matlab generates (like the ones shown here: http://www.mathworks.com/help/stats/multcompare.html) and I think the best place to put this functionality would be in your statsmodels.stats.multicomp.MultiComparison class.

The workflow would be something like:

mc = MultiComparison(data, labels)
res = mc.tukeyhsd() # res contains string output, everything else is attributes of mc instance
mc.plot_tukeyhsd_intervals()

The tukeyhsd intervals are based on Hochberg's generalized Tukey-Kramer confidence interval calculations. (Hochberg, Y., and A. C. Tamhane. Multiple Comparison Procedures. Hoboken, NJ: John Wiley & Sons, 1987.) Those CI's are unique in that they allow simultaneous comparison between all means with only a single interval per group.

So I have the plot_tukeyhsd_intervals currently working on top of the MultiComparison class - is this something you'd be comfortable adding to the module if I made a pull request? This is really a lot of the functionality I was looking for in the class. Otherwise, the tukeyhsd() works splendidly once you figure out the results output.

Reply

Josef Perktold April 24, 2013 at 3:09 PM

Thanks Tyler, Yes I'm very interested in this.

I opened https://github.com/statsmodels/statsmodels/issues/789 if you want to discuss it there.
I also linked there to what I started with pairwise comparisons of proportions.
The main question for the design is whether or what to add to the MultiComparison class and what to a results class for reuse by other multiple comparisons (plots ?).

Reply