Using copa

James W. MacDonald

October 17, 2016

1 Overview

In certain cancers (lymphoma, sarcoma, leukemia), it is common to have recurrent chromosomal rearrangements that may be a causal factor in the progression of the disease **?**. However, until recently these rearrangements have not been commonly found in other carcinomas **?**. Tomlins et al. describe a method they call Cancer Outlier Profile Analysis (COPA) that can be used to detect recurrent chromosomal rearrangements using microarray data. Their method however is limited to use with the Oncomine website `www.oncomine.org`, and does not at this time either pre-filter the data for likely candidates, nor compute any sort of inferential statistic.

2 Introduction

The idea behind COPA is very simple; it is well known that genetic translocations occur in cancer cells, and that these translocations can result in the up-regulation of oncogenes that may affect the progression of the cancer. This happens when the 5' activation domain of a constitutively expressed (or up-regulated) gene is fused to the 3' portion of a given oncogene, thus increasing the expression of the oncogene. This translocation can happen between the activating gene and multiple oncogenes, as was shown by Tomlins *et al.*, as well as others **?**.

Since a given translocation is only likely to occur once per sample, if there were multiple partners for a given activating gene, we would expect to see certain cancer samples with a high expression of say, gene A, whereas other cancer samples might have high expression of gene B, but these samples would be mutually exclusive. In addition, we would expect that the normal samples would not have high expression for either gene A nor B. We can

use this idea to both pre-filter genes as well as finding interesting genes that may be involved in translocations.

Common methods for detecting differences between tumor and normal samples will not work for finding these genes (e.g., t-tests); we need to find those genes where only a subset of the samples have high expression. To do this, we center and scale the data (on a row-wise basis) using the median and median average difference (MAD). We can then select a common value (default is 5) as a cutoff for 'outlier' status and apply this to all genes. We then simply look for pairs of genes that have a large number of mutually exclusive outlier (cancer) samples, but few or no normal outliers. The candidate gene pairs will be ranked based on the sum of outlier samples for each pair, as this seems to be the most reasonable criterion for ranking.

Since there may be several gene pairs with the same number of outliers we need to add an additional criterion to rank the ties. We use a modification of the ranking scheme used by Tomlins *et al.*. They simply ranked the genes using the 75^{th}, 95^{th} and 99^{th} percentiles of the centered and scaled expression values. Since we are looking at pairs rather than individual genes, we take the difference between the 75^{th} percentile of the tumor and normal samples, and then compute the sum of these differences for each gene pair. This value quantifies how different the outlier pairs are from their corresponding normals. We chose the 75^{th} percentile for ranking rather than say, the 95^{th} because the values at the higher percentile are what caused the gene pairs to be selected in the first place, so we want to use a less extreme percentile to distinguish between the tied pairs.

3 Using copa

To search for gene pairs that may be involved in translocations is very simple. Just load the *copa* package, and run the `copa` function using your microarray expression values. For this vignette, we will be using the *colonCA* package, which contains an `ExpressionSet` with normal and tumor colon expression data.

```
> library(copa)
> library(colonCA)
> data(colonCA)
> head(pData(colonCA), 10)

   expNr samp class
1      1   -1     t
```

```
2      2    1     n
3      3   -2     t
4      4    2     n
5      5   -3     t
6      6    3     n
7      7   -4     t
8      8    4     n
9      9   -5     t
10    10    5     n
```

We will use the third column of the `phenoData` object as our classlabel, which tells `copa` which samples are tumor and which are normal. There is no need to pre-filter the expression data; `copa` has an internal pre-filtering step that selects the top *pct*(percentile) of the data, based on the number of outliers. The default is to use the 95^{th} percentile as a cutoff; if this results in more than 1000 genes, `copa` will give a warning and allow you to abort the run (and presumably re-run with a higher value for *pct*).

One thing to keep in mind is that `copa` is going to be computing all pairwise sums of outliers, which can get to be a large number of computations really quickly (hence the warning at n = 1000). Although this portion of the function is written in C, and is actually quite fast, a large number of comparisons will tend to slow things down.

```
> rslt <- copa(colonCA, as.numeric(pData(colonCA)[,3]))
```

We can now look at a plot showing the number of outliers for each gene.

```
> plotCopa(rslt, idx = 1, col = c("lightgreen","salmon"))
```

Figure **??** shows the outlier status of the 'top' gene pair (based on having the most outlier samples). If using an Affymetrix GeneChip for which there is an annotation package, one can label the plots with the corresponding gene symbol by specifying the *lib* argument. Unfortunately, the colonCA data is based on a Hum6000 Affy chip, for which there is no annotation package.

This plot doesn't look that impressive, as there are only a few samples that fulfill the criterion for outlier status. We can look at how many gene pairs there are with a given number of outliers using the `tableCopa` function.

```
> plotCopa(rslt, idx = 1, col = c("lightgreen","salmon"))
```

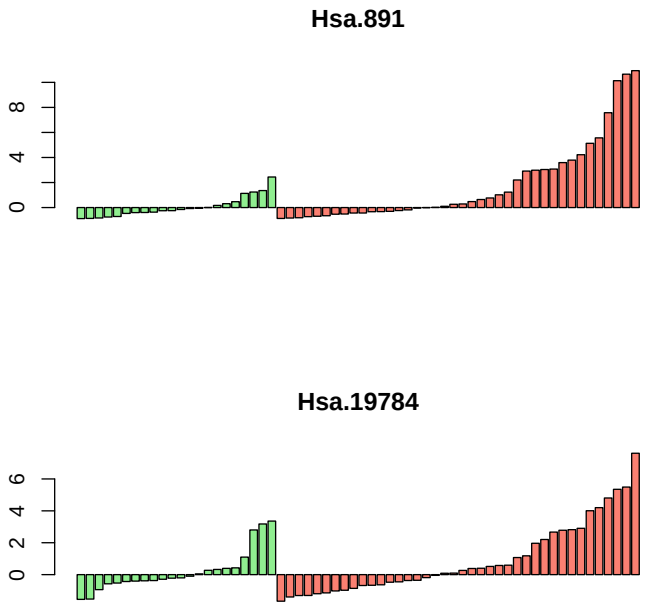

Figure 1: Plot of 'Top' Gene Pair

```
> tableCopa(rslt)

  9   8   7   6
 24 130 254 894
```

We might then want to know which genes have 9. We can list them out using the `summaryCopa` function.

```
> summaryCopa(rslt, 9)

   Number.of.pairs Probe.ID.1 Probe.ID.2
1                9    Hsa.891  Hsa.19784
2                9  Hsa.21562    Hsa.891
3                9    Hsa.140    Hsa.891
4                9  Hsa.22762    Hsa.891
5                9   Hsa.1765    Hsa.891
6                9   Hsa.1862    Hsa.891
7                9  Hsa.17564    Hsa.891
8                9    Hsa.891   Hsa.8831
9                9  Hsa.41247    Hsa.891
10               9  Hsa.23249    Hsa.891
11               9  Hsa.25536    Hsa.891
12               9  Hsa.21847    Hsa.891
13               9    Hsa.627   Hsa.8214
14               9  Hsa.20324    Hsa.891
15               9  Hsa.27085    Hsa.891
16               9    Hsa.891  Hsa.22614
17               9    Hsa.627    Hsa.594
18               9   Hsa.1331    Hsa.891
19               9   Hsa.2739    Hsa.891
20               9   Hsa.1799    Hsa.891
21               9   Hsa.3969    Hsa.891
22               9   Hsa.1574    Hsa.891
23               9    Hsa.627   Hsa.2772
24               9    Hsa.845    Hsa.891
```

We might now want to know how significant this result is. We can test this hypothesis by permuting the class labels of the samples many times and then checking to see how often we see pairs of genes with a certain number of outliers. By permuting the class labels we are mixing up the tumor and normals, so any pair with a large number of outliers by definition

has arisen by chance. If we get many gene pairs with say, 9 outliers then it is fairly likely that our observed results could have arisen by chance as well. However, if the opposite is true, then we have some reassurance that the observed results are not a chance event, and these gene pairs may well be undergoing some sort of recombination.

```
> prm <- copaPerm(colonCA, rslt, 9, 24)

Counting permutations...
100

> sum(prm >= 9)

[1] 3
```

In this instance, there are 3 times that the permuted data resulted in a number of outliers as large or larger than what we observed. This indicates that there may well be some recombination going on here, and it might be worthwhile to explore further.

A few notes about this function. First, it repeatedly re-runs the `copa` function after permuting the classlabels, so any caveats that I gave above about the number of genes to use above applies a hundred fold here. Note that the percentile cutoff used to create the `copa` object will be re-used for the permutations, so if the cutoff is too lenient, you may repeatedly be queried because of too many genes.

Second, the default for this function is 100 permutations. This is enough to get a basic idea, but is far too few to calculate a p-value or false discovery rate (FDR). For that, one should use at least 500 - 1000 permutations. Even at 1000 permutations, the smallest p-value will be 0.001 (actually the smallest will be 0, but the second smallest will be 0.001). Running `copaPerm` here on approximately 91 genes takes about 90 seconds. Increasing either the number of genes or the permutations may necessitate an overnight run.