Computing q-values conditioned on covariates using `swfdr`

29 October 2019

Contents

1 Introduction

Multiple hypothesis testing is a common concern in modern data science workflows: after performing several - in some cases, thousands - statistical tests on a dataset, many of the resultant p-values are expected to be small purely by chance. So it is imperative to interpret the results in a manner that limits spurious associations. At the same time, a parallel requirement is to identify significant hits despite having performed many tests.

In addition to the classic multiple hypothesis testing problem described above, analyses of complex datasets are also complicated by covariates. Given p-values calculated using a particular statistical test, background knowledge may suggest that the test may be more appropriate for some data instances than for others, or that some results may be more or less expected a-priori. A tempting analysis strategy in such situations might be to partition the dataset and to study each part separately. However, direct stratification is not trouble-free. It relies on splitting data into discrete groups, which can introduce challenges of its own, including in the choice of appropriate thresholds. It also leads to further statistical testing, which is redundant and magnifies the burden of multiple hypothesis testing. Thus, an alternative, more systematic, approach is needed to utilize background knowledge in statistical analyses.

There are, in principle, many possible ways to use covariate variables when tackling the multiple hypothesis testing problem. This vignette does not attempt an unbiased exposition of the available methods; some of them have been summarized and benchmarked by Korthauer et al. (2018). Instead, this vignette provides a practical guide to using the `swfdr` package, which implements algorithms for modeling the effect of covariates on a quantity called π_0, which can be interpreted as the probability that a null hypothesis is true. The theoretical aspects and modeling details are provided by Jager and Leek (2013), Boca and Leek (2017), and Boca and Leek (2018). The approach extends methods by Storey (2002) and Storey and Tibshirani (2003). In practice, these methods take in a set of p-values and output another set of quantities, q-values, that can be interpreted in the context of a multiple hypothesis analysis.

The vignette begins with notes on how to install the `swfdr` software. Next, it provides a review of q-values using the classic `qvalue` package. This section also introduces some conceptual datasets and motivates the use of covariates to augment standard q-value analysis. Subsequent sections explain how to condition q-values on covariates using `swfdr`. The main points are summarized in the discussion and some technical details are discussed in an appendix.

2 Installation

The `swfdr` package can be installed via the Bioconductor package manager, Morgan (2018). If you don't already have the package manager, you'll first have to install it.

```
install.packages("BiocManager")
```

This might show a prompt to select a package repository, and you can choose any one that seems convenient for you.

To perform the package installation, load the Bioconductor manager and request to install `swfdr`.

```
library("BiocManager")
BiocManager::install("swfdr")
```

After the above command completes, we can load the package.

```
library(swfdr)
```

For this vignette, we will also use Bioconductor package `qvalue`, Storey et al. (2018). This can be installed also via Bioconductor

```
BiocManager::install("qvalue")
library(qvalue)
```

3 q-values without covariates

Suppose we have measurements from two groups. To perform a statistical analysis, we would formulate a null hypothesis, compute a p-value against that hypothesis, and then use some criteria to call the result as significant or not. The practical details may vary from application to application. For example, a common use case is to assume that measuremets come from distributions with equal variances, formulate the hypothesis that the distributions have equal means, and compute p-values using a t-test. An interpretation strategy might consist of calling a result a 'hit' or 'significant' if the p-value is below a threshold, often 0.05.

In this section, we will look at how to interpret results from a large set of such statistical tests and control the rate of false discoveries using q-values. While the technical definitions for p-values and q-values are given in textbooks and original references, it is useful to highlight some of their properties here. Consider a statistical test and an associated numerical result v. The interpretation with respect to p-values and q-values would be as follows (Storey and Tibshirani (2003)).

- *p-value:* Assuming that the original data is consistent with the null hypothesis, the probability that the single result with $p = v$ is a false positive is v.
- *q-value:* Among the tests assigned q-values below v during an analysis, the proportion of false positives is v.

There are some important conceptual and practical differences between these definitions.

- The interpretation of the p-value is focused on a single test instance, so any adjustments motivated by multiple hypothesis testing must be performed *post-hoc*. In contrast, the q-value is defined relative to an ensemble of tests and thus intrinsically refers to a multiple-hypothesis context.
- The p-value is defined from the perspective of a hypothetical ensemble of data that is consistent with the null hypothesis. The q-value is instead formulated with respect to the ensemble of performed measurements. (Technically, the properties of the q-value are valid in the limit of a large ensemble of measurements, but it is possible to replace that hypothetical ensemble by an actual set of measurements and thus work with q-value *estimates*.)

3.1 Conceptual datasets

To demonstrate calculations of q-values, let's work with some practical datasets. To cover various scenarios, let's define three synthetic datasets and some helper functions. Each dataset will consist of a moderate number of examples, i.e. instances requiring a statistical test.

```
N <- 1000
```

In the first dataset, `dataset0`, let's consider two groups with three measurements each and formulate a null hypothesis that there is no difference in means between the two groups. Moreover, let's construct the data so that all measurements come from the same generative process. In other words, despite fluctuations in individual measurements, all instances are concordant with the null hypothesis.

```
set.seed(12345)
dataset0 <- matrix(rnorm(6*N), ncol=6)
head(dataset0, 3)
##            [,1]        [,2]       [,3]       [,4]       [,5]       [,6]
## [1,]  0.5855288  1.67751179 -0.6078411  0.5098921 -1.0426910 -0.6924722
## [2,]  0.7094660  0.07947405  1.0762231 -0.7153643 -0.1839484  1.1246956
## [3,] -0.1093033 -0.85642750 -0.5764258 -0.4055215  0.1792667  0.1745351
```

From this data, we can compute p-values for each set of measurements (rows) using a t-test. Because a similar calculation will be required again later, it is best to define a helper function.

```
# compute t-tests on each row of a matrix
dataset_p <- function(D, group1=c(1,2,3), group2=c(4,5,6)) {
  t.test.groups <- function(x) {
    t.test(x[group1], x[group2])$p.value
  }
  round(apply(D, 1, t.test.groups), 5) # rounding for legibility
}
D0 <- data.frame(p=dataset_p(dataset0), truth=0)
summary(D0$p)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00527 0.26459 0.48864 0.50197 0.75112 0.99992
```

After the helper function, the above block defines a data frame with p-values and an integer code `truth`; we will use this code later. The p-values range from 0.00527 to 0.99992. Thus, although by construction all instances in `dataset0` are consistent with the null hypothesis, some p-values are nominally quite small.

For the second dataset, `dataset1`, let's again consider two groups of three measurements each, but give one the groups a shifted mean.

```
dataset1 <- matrix(c(rnorm(3*N, 0, 1), rnorm(3*N, 2, 1)), ncol=6)
head(dataset1, 3)
##            [,1]       [,2]      [,3]     [,4]      [,5]       [,6]
## [1,] -0.3487804 -0.6790374 -1.486991 1.180370 2.1612775  2.7892407
## [2,] -1.0257007 -0.5723894 -2.205152 3.375843 2.5319214 -0.4168186
## [3,] -0.9074263  0.8232486  1.405955 1.701138 0.7020966  1.5649241
D1 <- data.frame(p=dataset_p(dataset1), truth=1)
summary(D1$p)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00009 0.03287 0.07704 0.13351 0.17774 0.97222
```

The resulting p-values span a similar range as before, but are skewed toward lower values. It is important to note that although all instances are actually inconsistent with the null hypothesis, some p-values are nominally large.

To conclude this section, let's create a final dataset that is a concatenation of the previous two. We will only need the p-values, so we can omit creating the raw data and only define the summary data frame.

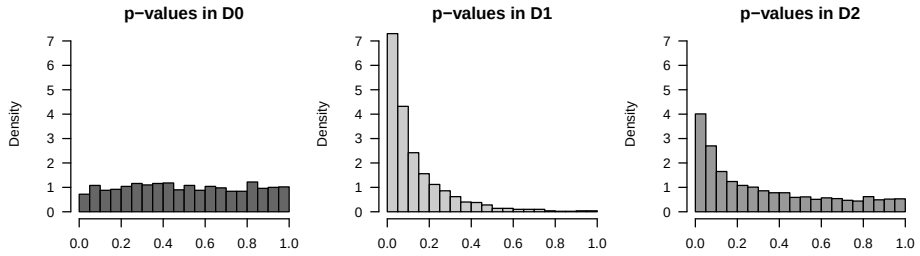
```
D2 <- rbind(D0, D1)
dim(D2)
## [1] 2000    2
table(D2$truth)
##
##    0    1
## 1000 1000
```

This combined dataset has 2000 instances. The first half is concordant with the null hypothesis (`truth` equal to zero) and the second half is not (`truth` equal to 1).

3.2 pi0

Given p-values from individual statistical tests, we can begin looking at their properties as a set of multiple hypotheses. Let's visualize the distributions of p-values.

```
par(mfrow=c(1,3), las=1, mar=c(3,4,2,1))
p_hist <- function(p, breaks=20, freq=F, ylim=c(0, 7.2), ...) {
  hist(p, breaks=breaks, freq=freq, ylim=ylim, xlab="", ...)
}
p_hist(D0$p, col="#666666", main="p-values in D0")
p_hist(D1$p, col="#cccccc", main="p-values in D1")
p_hist(D2$p, col="#999999", main="p-values in D2")
```



In the first histogram based on `D0`, the distribution of p-values is flat. This is the expected pattern for data consistent with the null hypothesis. In the second panel based on dataset `D1`, the distribution is skewed and indicates deviation from the null hypothesis. The last histogram, which is based on the composition of `D0` and `D1`, shows an intermediate skew.

Thus, the shape of the histogram is an indicator for the extent to which a dataset is consistent with the null hypothesis. This intuition can be codified into a probability that the null hypothesis is true, denoted as π_0 or `pi0`. It is computed from the histogram by considering

the number of test instances that give p-values below various thresholds, $p < \lambda$, where λ might be 0.05, 0.1, etc. See e.g. Storey (2002) and Storey and Tibshirani (2003) for details. We can compute π_0 for each dataset using the `qvalue` package.

```
library(qvalue)
D0q <- qvalue(D0$p)
D1q <- qvalue(D1$p)
D2q <- qvalue(D2$p)
c(D0=D0q$pi0, D1=D1q$pi0, D2=D2q$pi0)
##         D0         D1         D2
## 1.00000000 0.01388784 0.51395921
```

The π_0 estimate for `D0` is unity, which is consistent with how we generated the dataset. For `D1`, π_0 is much lower. At 0.014, it is greater than the true value (zero; none of the instances in `D1` were generated in a way consistent with the null hypothesis). However, this estimate is nonetheless closer to the truth. Unsurprisingly, the last π_0 estimate - for the combined dataset `D2` - lies between the previous values.

3.3 q-value

While π_0 is an aggregate summary of an entire dataset, we are often interested in assigning significance calls to individual test instances. The q-value is a measure that controls for false positive calls and internally uses the π_0 to allow for true positive calls.

We already computed q-values when we ran the function `qvalue` in the previous section. We can display a summary for the `D0` dataset as follows.

```
summary(D0q)
##
## Call:
## qvalue(p = D0$p)
##
## pi0: 1
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1   <1
## p-value        0      0     3     15    36   90 1000
## q-value        0      0     0      0     0    0 1000
## local FDR      0      0     0      0     0    0    0
```

The table displays hit counts at various thresholds. To simplify the discussion, let's just look at the rows labeled 'p-value' and 'q-value' and the threshold of 0.05. For the following, a helper function will be useful to extract a subset of the relevant results.

```
# extract number of hits with values below a threshold
hit_counts <- function(x, threshold=0.05) {
  counts <- c(sum(x$pvalues<threshold), sum(x$qvalues<threshold))
  names(counts) <- paste0(c("p<", "q<"), threshold)
  counts
}
hit_counts(D0q)
```

```
## p<0.05 q<0.05
##     36      0
```

If we were to call significance using unadjusted p-values at a threshold of 0.05, we would have 36 hits. Because we explicitly constructed the dataset to be consistent with the null hypothesis, all of these hits would be false positives. In contrast, the number of hits using the q-value would drop to 0, which in this case is the appropriate answer.

As an aside, note that if we were to adjust p-values for multiple-hypothesis testing say with a Bonferroni correction, the number of hits would also drop to 0. To check that, evaluate `sum(D0$p < 0.05/N)`.

Let's move on to the second dataset, in which all instances are inconsistent with the null hypothesis.

```
hit_counts(D1q)
## p<0.05 q<0.05
##    365   1000
```

Here, the number of hits obtained by thresholding p-values is higher. However, at 365, it leaves a large number of instances of the dataset as false negatives. In contrast, the procedure based on q-values captures the entire dataset. Again, this is a satisfactory result for this conceptual dataset.

Continuing the previous side note, how many hits would we have using a Bonferroni correction? The answer is 0 hits, thus producing an even larger number of false negatives. This shows that procedures that use adjustments of p-values are quite distinct from q-values.

Finally, let's consider the combined dataset.

```
hit_counts(D2q)
## p<0.05 q<0.05
##    401      0
```

The number of hits based on p-values is simply the sum of the previous two results, $401 = 36 + 365$. To look at this result from a different perspective and in preparation for subsequent calculations, we can display this information as a confusion matrix comparing expected versus computed hit counts.

```
# compare expected and observed p-values or q-values
confusion_matrix <- function(truth, x,
                             criterion=c("qvalues", "pvalues"),
                             threshold=0.05) {
  criterion <- match.arg(criterion)
  data = data.frame(truth, x[[criterion]]<threshold)
  colnames(data)[2] = paste0(substring(criterion,1,1), "<", threshold)
  table(data)
}
```

We can now get a summary of performance based on thresholding on p-values.

```
confusion_matrix(D2$truth, D2q, "p")
##      p<0.05
## truth FALSE TRUE
```

```
##     0   964   36
##     1   635  365
```

The 2000 instances are split into true positives (bottom-right), false-positives (top-right), true negatives (top-left), and false negatives (bottom-left). Overall, there are many false negatives and quite a few false positives.

The confusion matrix based on q-values is as follows.

```
confusion_matrix(D2$truth, D2q, "q")
##      q<0.05
## truth FALSE
##     0  1000
##     1  1000
```

There are no positive hits - we have lost all signal. However, we know that calling hits based on q-values produced correct result based on data from `D0` and `D1` separately. Thus, it might be possible to improve the calculation for the joint dataset. This leads to using background knowledge to condition calculations of π_0 and q-values in the next section.

4 Conditioning on covariates

In addition to a core set of data, many datasets come with additional descriptive variables - metadata. While these descriptive variables might not enter into a primary statistical analysis, it may nonetheless be fruitful to incorporate them in a larger analysis pipeline. Although many numerical recipes are possible in this direction. Package `swfdr` provides two functions, `lm_pi0` and `lm_qvalue`, that use linear models to condition estimates for π_0 and q-values.

4.1 Conditioned pi0

Following from the datasets and examples of the previous section, let's compute π_0 estimates for `D2` conditioned on the dataset of origin.

```
D2_lm_pi0 <- lm_pi0(D2$p, X=D2$truth)
```

The output is a composite object (list) with several components.

```
names(D2_lm_pi0)
## [1] "call"       "lambda"     "X.names"    "pi0.lambda" "pi0"
```

For some follow-up calculations, we will retrieve data from these components. However, it is instructive to first examine a summary of the object, via `summary(D2_lm_pi0)` or `print(D2_lm_pi0)` or by simply executing it in the console.

```
D2_lm_pi0
```

This should produce output with multiple sections. The first few summarize technical details.

```
##
## Call:
```

```
## lm_pi0(p = D2$p, X = D2$truth)
##
## lambda:
## (Length)      Min     Mean   Median      Max
##       19     0.05      0.5      0.5     0.95
```

The top part reiterates the function call and the section on `lambda` provides details on how the results were estimated, see Storey (2002) and Storey and Tibshirani (2003) for details. The next section summarizes the covariates.

```
##
## covariates:
## (Length)
##        1
```

In this case, there is only one covariate. The last section summarizes π_0 estimates.

```
##
## pi0:
## (Length)      Min     Mean   Median      Max
##     2000   0.0177    0.508    0.508   0.9983
```

In contrast to the π_0 estimate from function `qvalue`, we now have one estimate per instance in the dataset, ranging from 0.018 to 0.998. Internally, the function used logistic regression to estimate how the distribution of input p-values depends on the covariate, see Boca and Leek (2017) for details.

4.2 Conditioned q-values

Let's now compute q-values conditioned on the same covariate.

```
D2_lm_q <- lm_qvalue(D2$p, X=D2$truth)
```

Again, the output is a composite object. A summary is available by evaluating it in the console.

```
D2_lm_q
```

The majority of the output is analogous to what was displayed for `D2_lm_pi0`. The new section summarizes hit counts at various thresholds.

```
##
## Cumulative number of significant calls:
##             <1e-4   <1e-3   <0.01   <0.05    <0.1      <1
##   p-value       1       7      96     401     671    2000
##   q-value       0       0     913    1000    1000    2000
```

In this case, we have 1000 hits based on q-values at a threshold of 0.05. We can further compute a confusion matrix.

```
confusion_matrix(D2$truth, D2_lm_q)
##      q<0.05
```

```
## truth FALSE TRUE
##     0  1000    0
##     1     0 1000
```

This reveals a strong presence of true positives with 0 false positives and 0 false negatives. The analysis of the combined dataset with covariate thus reproduces the performance we obtained when we analyzed `D0` and `D1` separately.

Of course, the above calculation is idealized in that the covariate perfectly stratifies the instances. To finish this section, we can repeat the calculation in a more difficult setting where the covariate carries imperfect signal. To this end, let's define an object similar to `D2`.

```
D2_noisy <- data.frame(p=D2[, "p"],
                       truth.noisy=D2[, "truth"]+rnorm(2*N, 0, 0.4))
head(D2_noisy, 3)
##         p truth.noisy
## 1 0.30806   0.3076642
## 2 0.44155  -0.4250800
## 3 0.16487  -0.4386329
```

The new variable `truth.noisy` is now a real number based on the previous `truth`, but carries considerable noise. We can repeat the `lm_qvalue` calculations using this as a covariate.

```
D2_noisy_lm_q <- lm_qvalue(D2_noisy$p, X=D2_noisy$truth.noisy)
```

The hit summary for this object is as follows.

```
##
## Cumulative number of significant calls:
##             <1e-4   <1e-3   <0.01   <0.05    <0.1      <1
##   p-value       1       7      96     401     671    2000
##   q-value       0       0       5     442     837    2000
```

Let's inspect the confusion matrix at various thresholds.

```
confusion_matrix(D2$truth, D2_noisy_lm_q)
##      q<0.05
## truth FALSE TRUE
##     0   999    1
##     1   559  441
confusion_matrix(D2$truth, D2_noisy_lm_q, threshold=0.1)
##      q<0.1
## truth FALSE TRUE
##     0   976   24
##     1   187  813
```

In each case, the confusion matrix reveals a moderate or large number of true positives with a limited number of false positives. Thus, even in this noisy setting, we have a high hit rate while restraining false discovery.

Although the last example introduced some noise, all calculations in this section are still based on synthetic data and are thus quite artificial. We have also considered analyses based on just one covariate. Nonetheless, the examples are instructive because the analytical workflow and syntax also apply in more complex scenarios, as demonstrated in the next section.

5 Conditioning on multiple covariates

The package comes with a prepared dataset, `BMI_GIANT_GWAS_sample`, that summarizes a portion of a genome-wide association study by Locke et al. (2015). The dataset holds associations between simple-nucleotide polymorphisms (SNPs) and body-mass-index (BMI).

```
GWAS <- BMI_GIANT_GWAS_sample
dim(GWAS)
## [1] 50000     9
head(GWAS)
## # A tibble: 6 x 9
##   SNP   A1    A2    Freq_MAF_Hapmap        b      se     p      N
##   <chr> <chr> <chr>           <dbl>    <dbl>   <dbl> <dbl>  <dbl>
## 1 rs10~ T     C              0.025    1.47e-2 0.0152  0.334 212965
## 2 rs91~ A     G              0.342   -3.40e-3 0.0037  0.358 236084
## 3 rs48~ A     C              0.00830  1.63e-2 0.0131  0.213 221771
## 4 rs17~ A     G              0.167    4.00e-4 0.00480 0.934 236177
## 5 rs46~ C     G              0.25     1.10e-3 0.0042  0.793 236028
## 6 rs11~ G     A              0.233   -6.00e-4 0.0042  0.886 235634
## # ... with 1 more variable: Freq_MAF_Int_Hapmap <fct>
```

This is a table with 50,000 rows. Column `SNP` contains an identifier for a genetic polymorphism. Column `p` contains the p-value from a statistical test linking that polymorphism and BMI. The other columns contain metadata about the polymorphism - see the original article Locke et al. (2015) or the data set annotation (`help(BMI_GIANT_GWAS_sample)`).

In this vignette, we will use two of the metadata columns, `N` and `Freq_MAF_Hapmap`. The first is an integer field conveying the number of patients in the study for which the SNP was measured. This impacts on the power of the individual statistical tests. The second feature is the prevalence of the polymorphism in a background population. This impacts the likelihood that the SNP is associated with an unhealthy phenotype.

5.1 Conditioned pi0s with multiple covariates

The calculation for π_0 proceeds just like in the previous section. In this case, however, argument `X` should be a matrix or data frame containing both covariates of interest.

```
GWAS_lm_pi0 <- lm_pi0(GWAS$p, X=GWAS[, c("N", "Freq_MAF_Hapmap")])
```

We can view a summary of the output object by executing it in the console.

```
GWAS_lm_pi0
##
## Call:
## lm_pi0(p = GWAS$p, X = GWAS[, c("N", "Freq_MAF_Hapmap")])
##
## lambda:
## (Length)      Min     Mean   Median      Max
##       19     0.05      0.5      0.5     0.95
##
## covariates:
```

```
## (Length)
##        2
##
## pi0:
## (Length)      Min     Mean   Median      Max
##    50000   0.7439   0.9424   0.9524        1
```

The output structure is by now familiar. One interesting section here summarizes the covariates and confirms that the function used two variables in the calculation.

The final table summarizes the distribution of π_0 values. Recall that π_0 estimate the probability that a null hypothesis is true. Here, the estimates range from 0.7439 to 1. Because the values are all fairly close to unity, they suggest that the test instances are largely concordant with no association between SNPs and BMI. This is an expected property for a GWAS study.

5.2 Conditioned q-values with multiple covariates

Next, we can evaluate whether or not individual instances in the dataset are significant.

```
GWAS_lm_qvalue <- lm_qvalue(GWAS$p, X=GWAS[, c("N", "Freq_MAF_Hapmap")])
```

We can preview the contents of the output object by executing it on the console.

```
GWAS_lm_qvalue
```

Most of the output should be the analogous to that of `GWAS_lm_pi0`. The new section is a summary of hits at various thresholds.

```
##
## Cumulative number of significant calls:
##             <1e-4   <1e-3   <0.01   <0.05    <0.1      <1
##   p-value     186     405    1388    3771    6468   49619
##   q-value      49      70     126     254     374   49912
```

We can use our helper function to focus just on the 0.05 threshold.

```
hit_counts(GWAS_lm_qvalue)
## p<0.05 q<0.05
##   3771    254
```

There are 3771 SNPs that have a nominal p-value lower than 0.05. (That hit count is unadjusted for multiple-hypothesis testing and is likely to include many false positives. If we were to correct for multiple testing via a Bonferroni correction, we would instead impose a threshold of 0.05/50000. The result would be 64 hits.) In contrast, using a q-value threshold, we have 254 hits.

5.3 Comparison with `qvalue`

In the above calculations, we computed q-values conditioned on two covariates. Let's now see how those results compare with q-values without conditioning. We can compute those by repeating the same steps as before, but without specifying argument `X`.

Computing q-values conditioned on covariates using `swfdr`

```
GWAS_lm_qvalue_0 <- lm_qvalue(GWAS$p)
## Warning: X is missing or NULL - without covariates, modeling will have no
## effect
```

In this calculation, `lm_qvalue` displays a warning that raises attention to the fact that a function meant to model the effect of covariates is actually used without any covariates at all. This message is useful to detect coding errors, e.g. unintentionally providing an empty object for modeling. Here, omitting the covariates was the intended action, so we can look past the warning and inspect the output.

```
##
## pi0:
## (Length)      Min     Mean   Median      Max
##    50000   0.9564   0.9564   0.9564   0.9564
##
## Cumulative number of significant calls:
##             <1e-4   <1e-3   <0.01   <0.05    <0.1      <1
##   p-value     186     405    1388    3771    6468   49619
##   q-value      49      69     124     247     354   50000
```

This reveals that although the function computes 50,000 estimates for pi0, all the estimates are equal. The hit table is slightly different than before.

When there are no covariates, the calculations performed by `lm_qvalue` are conceptually equivalent to those in function `qvalue` from the `qvalue` package. We can check this explicitly.

```
GWAS_qvalue = qvalue(GWAS$p)
summary(GWAS_qvalue)
##
## Call:
## qvalue(p = GWAS$p)
##
## pi0: 0.9559508
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1    <1
## p-value      186    405  1388   2404  3771 6468 49619
## q-value       49     69   124    185   247  354 50000
## local FDR     42     50    76    107   148  203 27449
```

Apart from the stylistic discrepancies, the final q-value results in the hits table are the same as those output by `lm_qvalue`. There is, however, a small discrepancy in the estimate for π_0. This is due to a slight difference in an internal algorithm, which is elaborated upon in the appendix. Here, we note that the discrepancy is $<1\%$ so it is likely to be inconsequential in most workflows. However, it is possible to obtain exact parity by calling `lm_qvalue` with an additional argument `smoothing` set to `smooth.spline`. We can verify this by repeating the calculation with this modified setting.

```
GWAS_lm_qvalue_smooth <- lm_qvalue(GWAS$p, smoothing="smooth.spline")
## Warning: X is missing or NULL - without covariates, modeling will have no
## effect
```

The same warning appears, but we can again skip it and inspect the output.

```
table(GWAS_lm_qvalue_smooth$pi0)
##
## 0.955950835227995
##             50000
```

This reveals all the π_0 estimates are equal to the output from `qvalue` - we have exact parity with the `value` package. This is a useful technical control and it builds confidence in the correctness of the results. However, running this code in the console, there will be a noticeable difference in execution time. Thus, for large datasets, it is more convenient to use the default settings.

6 Discussion

Interpreting statistical results can be challenging, especially when an analysis involves a large number of tests and p-values. When the aim is to call significant hits while controlling the rate of false discoveries, it can be useful to re-frame the interpretation in terms of q-values. The influencial package `qvalue` provides functions for this purpose, but it is only applicable in contexts when all the statistical tests in an analysis are *a-priori* similar and can be weighted equally. These constraints are at odds with the analysis of complex datasets for which metadata may hold signals about the tested relationships. The `swfdr` package provides functions for using such metadata as covariates while controlling false discovery rates.

In this vignette, we saw how to install the software and walked through the analysis of some small datasets. These datasets, being synthetically generated, do not capture the nuances and complexities of a real-word analysis. Instead, the hope is that their they capture the value of using q-values for controlling false discovery and the potential benefits of using informative covariates. In particular, they demonstrate the utility of q-values for interpret datasets with a very high proportion of instances that are significant, as well as for modeling with covariates to identify hits when they are surrounded by data that is concordant with the null hypothesis. These are illustrative examples; further evaluations are described in the original articles, Boca and Leek (2017) and Boca and Leek (2018), as well as in an independent review, Korthauer et al. (2018), which concluded that `swfdr` consistently performs well in a range of scenarios.

All the analyses included a discussion of π_0, the probabilities that a null hypothesis is true, before going on to compute q-values. Given the outputs of functions `lm_pi0` and `lm_qvalue`, it should be apparent that the second calculation is an extension of the first. This duplication of effort is included in the vignette to emphasize the central role π_0 plays in computing q-values and in conditioning on covariates. Practical pipelines, however, may rely on the single function `lm_qvalue`.

If you use package `swfdr`, please cite:

- the article describing the method for conditioning false-discovery rates on covariates via π_0, Boca and Leek (2018)
- the R `swfdr` software, Leek, Jager, and Boca (2018)

To view the package source, report a bug, or contribute bug fixes, please see the github development page https://github.com/leekgroup/swfdr.

A Technical notes

The main vignette described a complete, coherent workflow for controlling false discovery rates using functions `lm_pi0` and `lm_qvalue` from the `swfdr` package. The focus was on demonstrating the core features of those functions and on interpreting their outputs. But those same functions also support other arguments which can fine-tune aspects of the calculations.

These additional arguments have default values that should be suitable for common use cases. However, (rare) situations may occur when it may be desirable to adjust the default settings. This section quickly summarizes those options and their implications for the interpretation of the results and computational performance. The arguments are applicable to both `lm_pi0` and `lm_qvalue`, unless specified otherwise.

A.1 Modeling resolution (argument `lambda`)

As hinted in one of the early sections, estimates for π_0 are computed from the distribution of the input p-values. The details for the algorithm are described e.g. in Storey (2002) and Storey and Tibshirani (2003). Briefly, the calculations take a threshold, say $\lambda = 0.05$, and evaluate the proportion of p-values below that threshold, and then repeat for several thresholds, say $\lambda = 0.1, 0.15, \ldots$. By using such estimates together with a spline fit, the calculations arrive at a single estimate for π_0. Argument `lambda` provides the series of thresholds used in this process.

A large number of thresholds, i.e. a long vector `lambda`, is needed to allow the spline to fit the trend in proportions in a reliable manner. However, excessive length can be counter-productive if the input set of p-values is short. Long `lambda` vectors can also increase running time.

In many common applications, the default `lambda` vector will be appropriate and it is recommended to leave it unchanged. For small datasets, say below 100 items, it may be advisable to provide a shorter vector, e.g. `seq(0.05, 0.95, 0.1)`.

A.2 Covariate matrix (argument `X`)

One of the key inputs to `lm_pi0` and `lm_qvalue` is an object, `X`, capturing covariates. When there is only one covariate, `X` can be specified as a vector. Otherwise, it should be either a matrix or a data frame.

Covariate data is used internally in relatively simple linear models and thus object `X` cannot contain missing (`NA`) or non-finite values (`NULL`, `Inf`, `NaN`). Datasets with such components should either omit incompatible parts before processing with `swfdr`, or impute values.

Covariate data should contain only numeric data. Factors and other categorical data are not supported.

A.3 Model type (argument `type`)

Modeling the effect of covariates on π_0 is controlled via argument `type`. The default value for this argument is `logistic`. In this configuration, functions `lm_pi0` and `lm_qvalue` use logistic regression with one term for each given covariate. The other supported value for the `type` argument is `linear`, which uses a standard linear regression model instead.

The run time of linear regression is typically faster than logistic regression. However, linear regression is uninformed about the fact that the output variable is meant to be a probability in a finite range. Thus, logistic is usually the more appropriate approach.

It is recommended to leave `type` at its default value. The argument is provided primarily for backward compatibility and as a inerface for exploring other modeling approaches.

A.4 Thresholding (argument `threshold`)

Internally, `lm_pi0` and `lm_qvalue` train models on the input data and then predict individual `pi0` estimates. Because these predictions represent probabilities, they are expected to be in the range $[0, 1]$. However, some predictions may fall outside this range. By default, the functions threshold spurious values and impose the expected lower and upper bounds. Setting argument `threshold` to `FALSE` allows the raw prediction values to propagate to the output.

The value of the `threshold` argument has almost no impact on performance. Changing it from its default value can make interpretation of the output less intuitive, although unusual values can in principle provide insight on the quality of the covariate modeling.

It is recommended to leave `threshold` at its default value of `TRUE`. The argument is provided primarily for testing and research purposes, as well as for backward compatibility.

A.5 Smoothing (argument `smoothing`)

As described above, the calculations internally use splines to model the effect of λ on π_0. There exist several algorithms for fitting splines, and argument `smoothing` toggles between two approaches. Both procedures are motivated by the same conceptual reasoning, but differ in the implementation, computational performance, and provided output.

Setting `smoothing` to `smooth.spline` instructs the algorithm to use base R's `smooth.spline` to fit the spline. This is the spline-fitting approach that is used within the `qvalue` package and in early versions of the `swfdr` package. In the context of covariate modeling, this approach is, however, computationally inefficient. Setting `smoothing` to `unit.spline` shifts the spline calculations to a bespoke implementation that exploits the fact that multiple splines are fitted for the same series of λ and the fact that the objective of the calculation is to obtain a single value π_0 and not the actual smoothing spline. The bespoke approach can reduce the running time many-fold with only small numerical discrepancies with the original algorithm.

It is important to stress that these implementations are provided for performance reasons only; you are welcome to pick the smoothing method of your choice.

A.6 Smoothing degrees of freedom (argument `smooth.df`)

Spline-fitting can be further tuned by setting the spline degrees of freedom via argument `smooth.df`. The value of `smooth.df` must be ≥ 3 to fit p-value distributions such as in the examples in the vignette. Higher values would allow the internal model to accommodate more complex distributions. However, if the p-value distribution deviates substantially from the examples, the premise of the π_0 calculation comes into question. Thus, there is rarely a good reason to increase the value of `smooth.df`.

It is recommended to leave this argument at its default value. The argument is provided primarily for testing, diagnostics, and backward compatibility.

References

Boca, Simina M, and Jeffrey T Leek. 2017. "A Regression Framework for the Proportion of True Null Hypotheses." *bioRxiv*. Cold Spring Harbor Labs Journals, 035675.

———. 2018. "A Direct Approach to Estimating False Discovery Rates Conditional on Covariates." *PeerJ* 6. PeerJ Inc.:e6035.

Jager, Leah R, and Jeffrey T Leek. 2013. "Empirical Estimates Suggest Most Published Medical Research Is True." *Biostatistics*.

Korthauer, Keegan, Patrick K Kimes, Claire Duvallet, Alejandro Reyes, Ayshwarya Subramanian, Mingxiang Teng, Chinmay Shukla, Eric J Alm, and Stephanie C Hicks. 2018. "A Practical Guide to Methods Controlling False Discoveries in Computational Biology." *bioRxiv*. Cold Spring Harbor Laboratory, 458786.

Leek, Jeffrey T., Leah Jager, and Simina M. Boca. 2018. *Swfdr: Science-Wise False Discovery Rate and Proportion of True Null Hypotheses Estimation*.

Locke, Adam E, Bratati Kahali, Sonja I Berndt, Anne E Justice, Tune H Pers, Felix R Day, Corey Powell, et al. 2015. "Genetic Studies of Body Mass Index Yield New Insights for Obesity Biology." *Nature* 518 (7538). Nature Publishing Group:197–206.

Morgan, Martin. 2018. *BiocManager: Access the Bioconductor Project Package Repository*. https://CRAN.R-project.org/package=BiocManager.

Storey, John D. 2002. "A Direct Approach to False Discovery Rates." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 64 (3). Wiley Online Library:479–98.

Storey, John D, Andrew J Bass, Alan Dabney, and David Robinson. 2018. *Qvalue: Q-Value Estimation for False Discovery Rate Control*. http://github.com/jdstorey/qvalue.

Storey, John D, and Robert Tibshirani. 2003. "Statistical Significance for Genomewide Studies." *Proceedings of the National Academy of Sciences* 100 (16). National Acad Sciences:9440–5.