The Q-Value for Multiple Comparison Correction

Nick Strayer

April 24, 2015

We will be using the algorithm outlined in Storey and Tibshiran, 2003 for multiple comparison corrections. The algorithm uses a metric known as the "q-value" for finding significance. Its input is an array of p-values.

Testing Data:

For this process we will be using data from the UVM Complex Systems Center to test the algorithm. Specifically a list of words most associated with obesity in a given population.

The Hedonometer is a tool gathering data from twitter to help guage happiness in a population on a macro scale.

Througout this writeup I will refer to the input data in two forms: `word` which is an array of the feature names and `pval` which is an array of the associated p-values. The data we are dealing with comes in the following form:

```
##     word        pval
## 1    pic 1.33318e-16
## 2   cafe 6.06703e-14
## 3  photo 4.86716e-13
## 4  sushi 9.93030e-13
## 5 posted 2.68595e-12
## 6   thai 3.69366e-12
```

The algorithm:

1) Order the p-values.

```
tests = arrange(tests, pval)
head(tests)
```

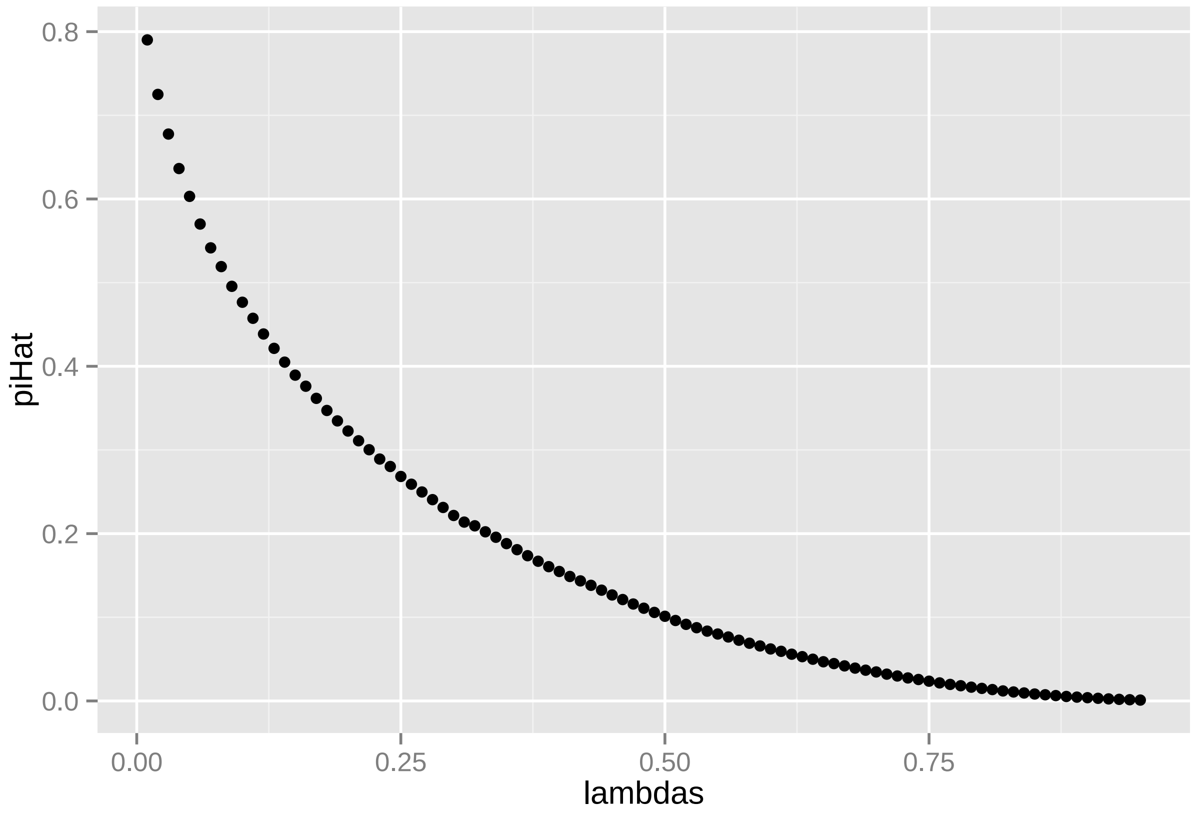
```
##     word        pval
## 1    pic 1.33318e-16
## 2   cafe 6.06703e-14
## 3  photo 4.86716e-13
## 4  sushi 9.93030e-13
## 5      i 1.00012e-12
## 6 posted 2.68595e-12
```

2) Look at the potential $\hat{\pi}_0(\lambda)$ values:

This will eventually help us arrive at an estimate for the proportion of features which are truly null (π_0). This is roughly equivalent to drawing a horizontal line at the point on the histogram of p-values where the distribution flattens and recording the y-axis value.

$$\hat{\pi}_0(\lambda) = \frac{\#\{p_i > \lambda\}}{m(1-\lambda)}$$

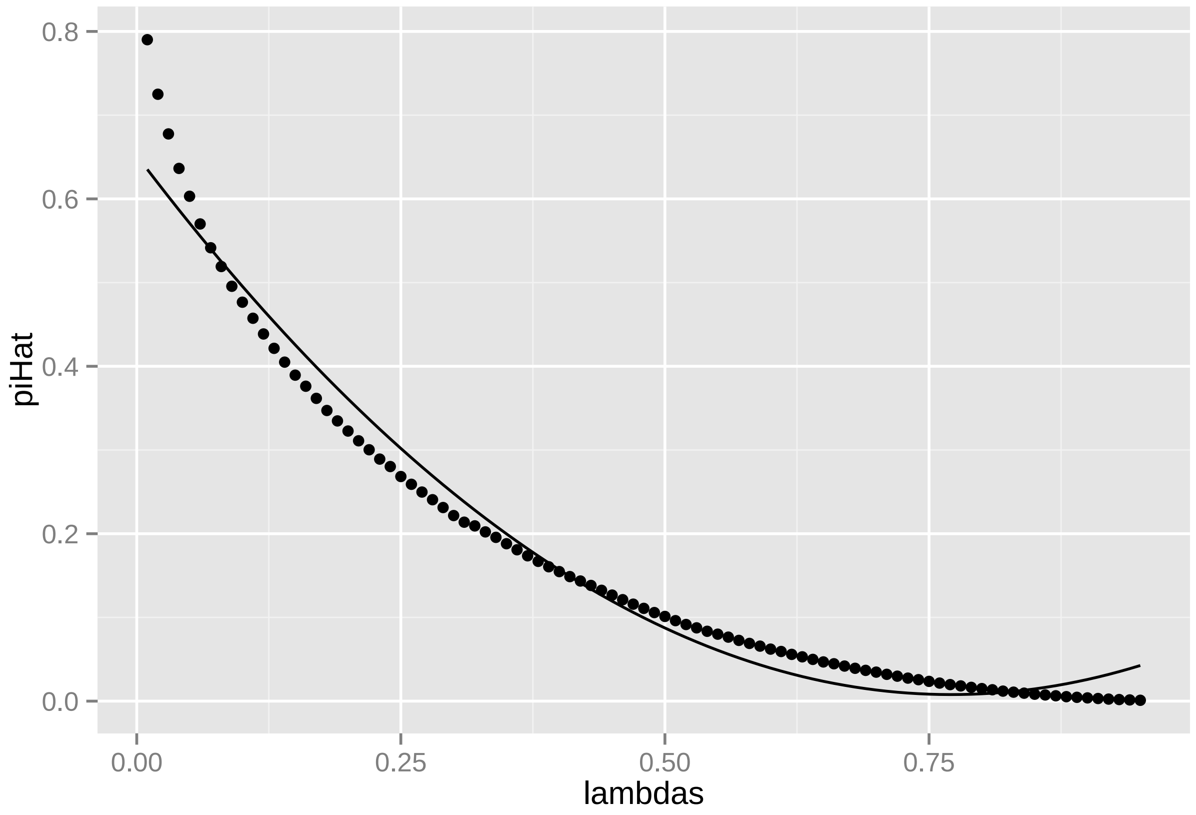
```
#Get length of data/ number of tests performed
m = length(tests$pval)
#Generate lambdas
lambdas = seq(0.01,0.95,0.01)
#Generate pi hats
piHat   = sapply(lambdas, function(l) sum(tests$pval > l)/m*(1 - l) )
#Build dataframe
piHat_df = data.frame(lambdas, piHat)
#Plot to see what we have
ggplot(piHat_df,aes(x=lambdas, y=piHat)) + geom_point()
```



3) Fit a curve to the $\hat{\pi}_0$ values

Note: These data don't respond as well to a natural cubic spline as is called for in the paper. See the end of this report on explorations of different methods of finding π_0.

```
spline = smooth.Pspline(lambdas,piHat, norder = 3, df = 3, method = 2)

piHat_df$spline = spline$ysmth

ggplot(piHat_df,aes(x=lambdas)) + geom_point(aes(y = piHat)) + geom_line(aes(y = spline))
```



4) Set $\hat{\pi_0}$ to the spline at 1.

```
pi0 = predict(spline, 1)
print(pi0[1,1])
```

```
## [1] 0.0646744
```

5) Calculate $\hat{q_m}$

To find the q-values we start from the bottom and work our way up to the most significant test. This is because the algorithm calls for us to assign a given test's q-value based upon the false discovery rate of all less significant tests.

$$\hat{q_m} = \hat{\pi_o} \cdot p_m$$

```
#Start the array of values by putting in the last one
qvals = pi0 * tail(tests$pval, 1)
```

6) Find all of the rest of the q vals:

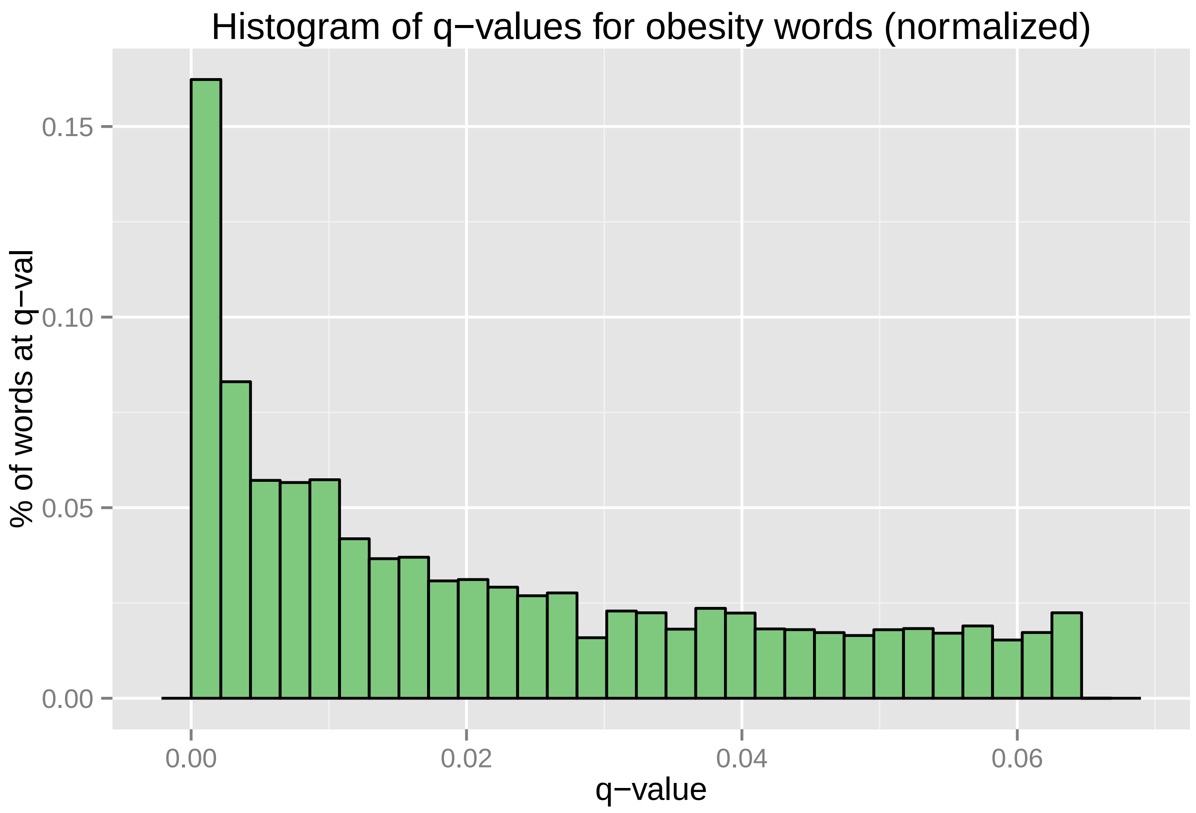
$$min((\hat{\pi_o} \cdot m \cdot p_i)/i, q_{i+1})$$

```
for(i in (m-1):1 ){

  latest = pi0 * m * tests$pval[i]/ i
  last   = qvals[1] #because we are filling qvals reverse, 1 will always be the last computed val

  q = min(latest, last)
  qvals = c(q, qvals)
}

tests$qval = qvals

head(tests)
```

```
##     word        pval         qval
## 1    pic 1.33318e-16 3.413812e-13
## 2   cafe 6.06703e-14 7.767781e-11
## 3  photo 4.86716e-13 4.154370e-10
## 4  sushi 9.93030e-13 5.121922e-10
## 5      i 1.00012e-12 5.121922e-10
## 6 posted 2.68595e-12 1.146298e-09
```

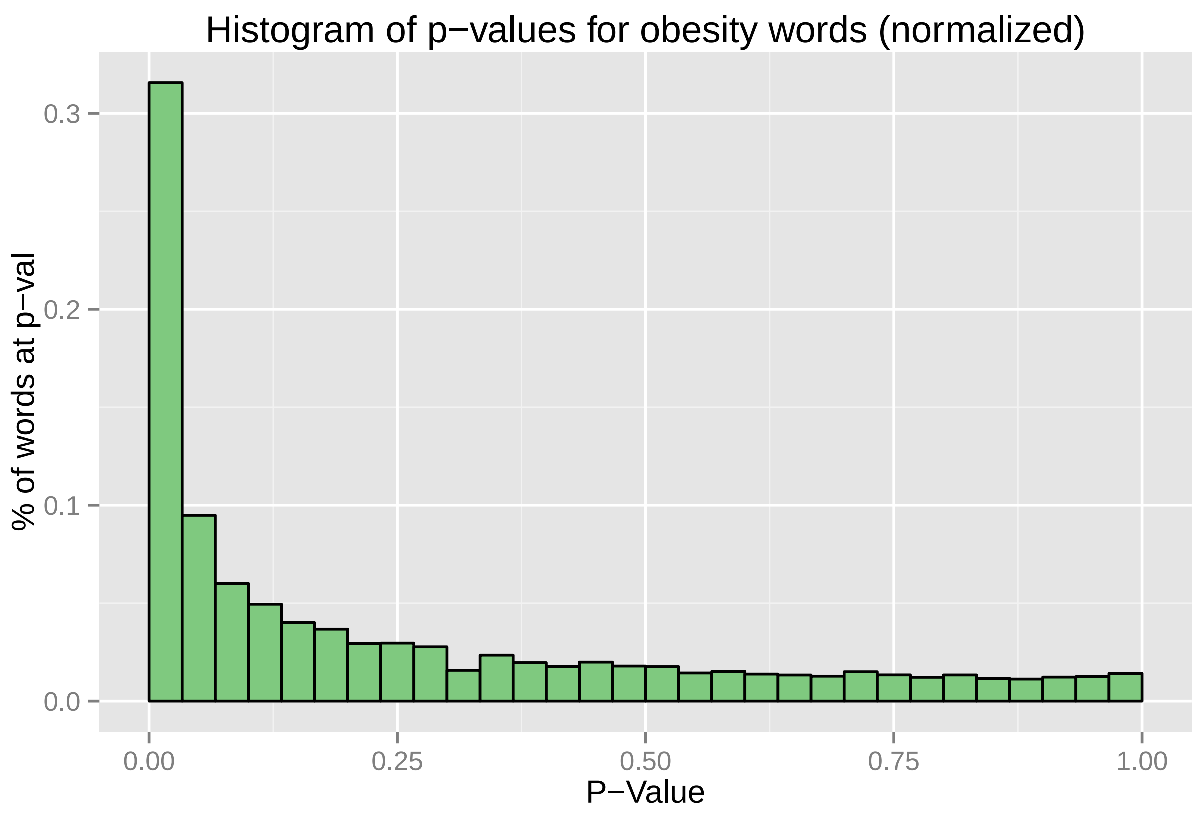
Histogram of the resultant q-values:

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



Compare that with the histogram of the p-values (note the x-scale):

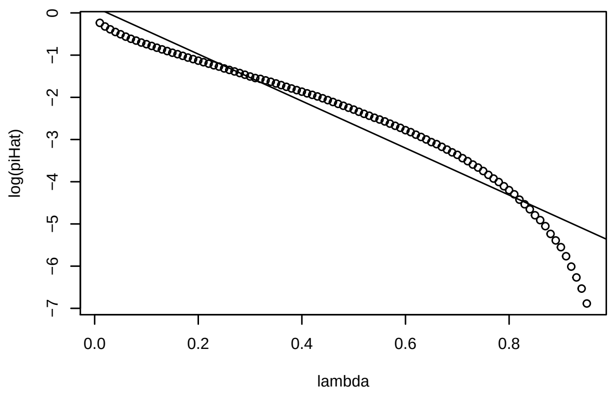
```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```

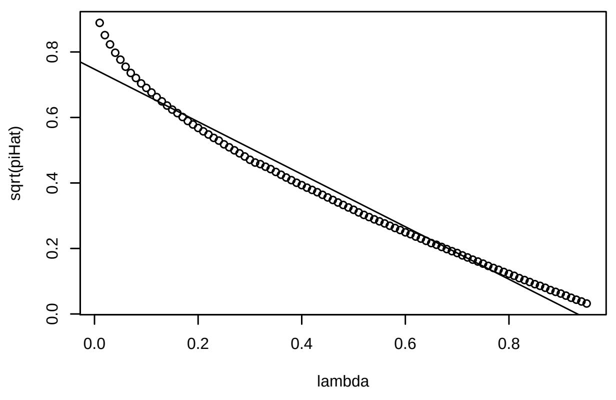


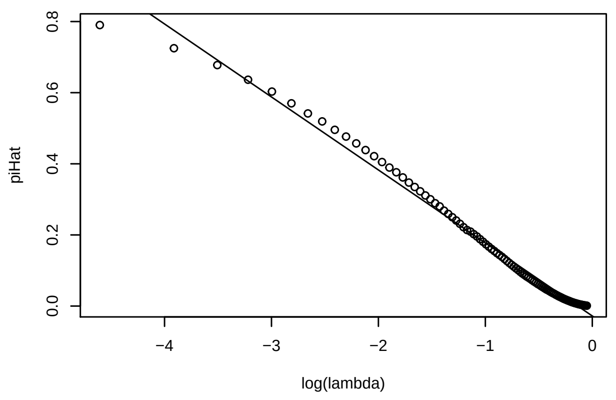
Point of note:

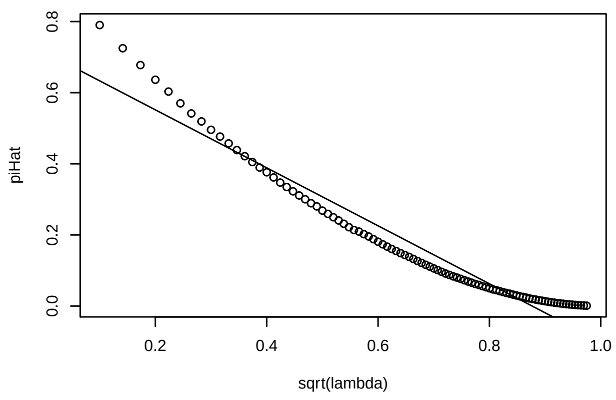
- I think it would be interesting to look more into the distribution of $\hat{pi}$ and also the type of curve fitted to it.

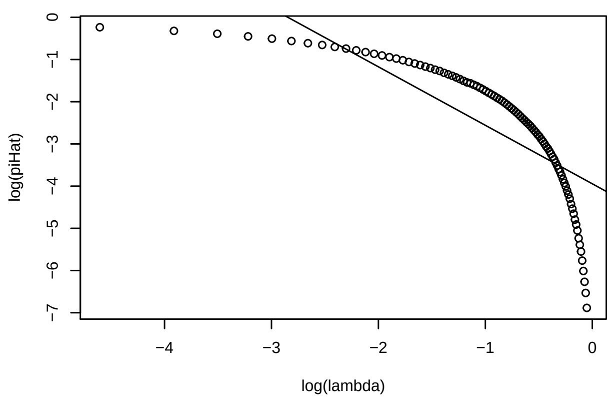
Here are a couple of extremely simple transformations applied to the data (log and square root respectively.) I have fitted a simple linear regression model to each transformation as well.

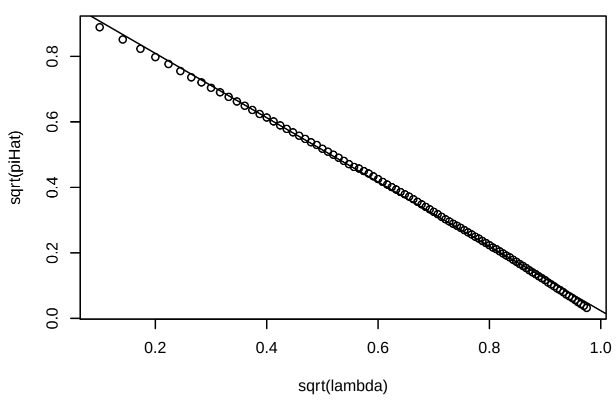












Something looks right with that last plot.

Let's see what type of π_0 it gives us.

```
model = lm(sqrt(piHat) ~ sqrt(lambda))
pi_zero = predict.lm(model, data.frame(lambda=1, piHat=0))[1]
print(pi_zero)
```

```
##           1
## 0.02340597
```

Discussion:

In an ideal world the algorithm for choosing a π_0 automatically would pick a value around .03 for this dataset (where the p-value histogram appears to level out on the y axis). The natural cubic spline method described in the paper picks .06 which is clearly way too high. By taking a double square root translation of the data and fitting a simple linear model we are able to obtain a π_0 estimate of 0.023 which is much closer to what we are looking for.

It must be noted however, that if we 'un-square' the estimate provided by the double squared prediction we get a much much lower value (.005). More data are neccesary to test if this is a good alternative method to a natural cubic spline.