Final Project

Evan Ott

December 5, 2016

Stochastic Gradient Descent (Logistic Regression)

Let's load in the data and Dr. Scott's algorithm:

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```r
library(latex2exp)
library(Matrix)
library(Rcpp)
library(RcppEigen)
library(reshape2)

Rcpp::sourceCpp("scott_sgdlogit.cpp")

raw_data = read.csv("master_countedByKegg.csv", header = TRUE)

# Read specific columns from the data

gene = as.character(raw_data[ , 1])

data = raw_data[ , 2:ncol(raw_data)]

N = ncol(data)
P = nrow(data)
M = rep(1, N) # Used in logistic regression

# For C++ code, it's easiest to use a sparse matrix
# TODO: might want to consider scaling the counts, either in the way DESeq2 does
# it or something similar.
X = Matrix(as.matrix(unname(data)), sparse = TRUE)

# X[i, j] == data[i, j] is non-negative, so this trims out the genes with no counts
X_nozero = X[which(rowSums(data) > 0), ]

# Convert the column names to 0 = control, 1 = treatment
Ynames = names(data)
Y = stringi::stri_endswith(Ynames, fixed = "T") * 1
```

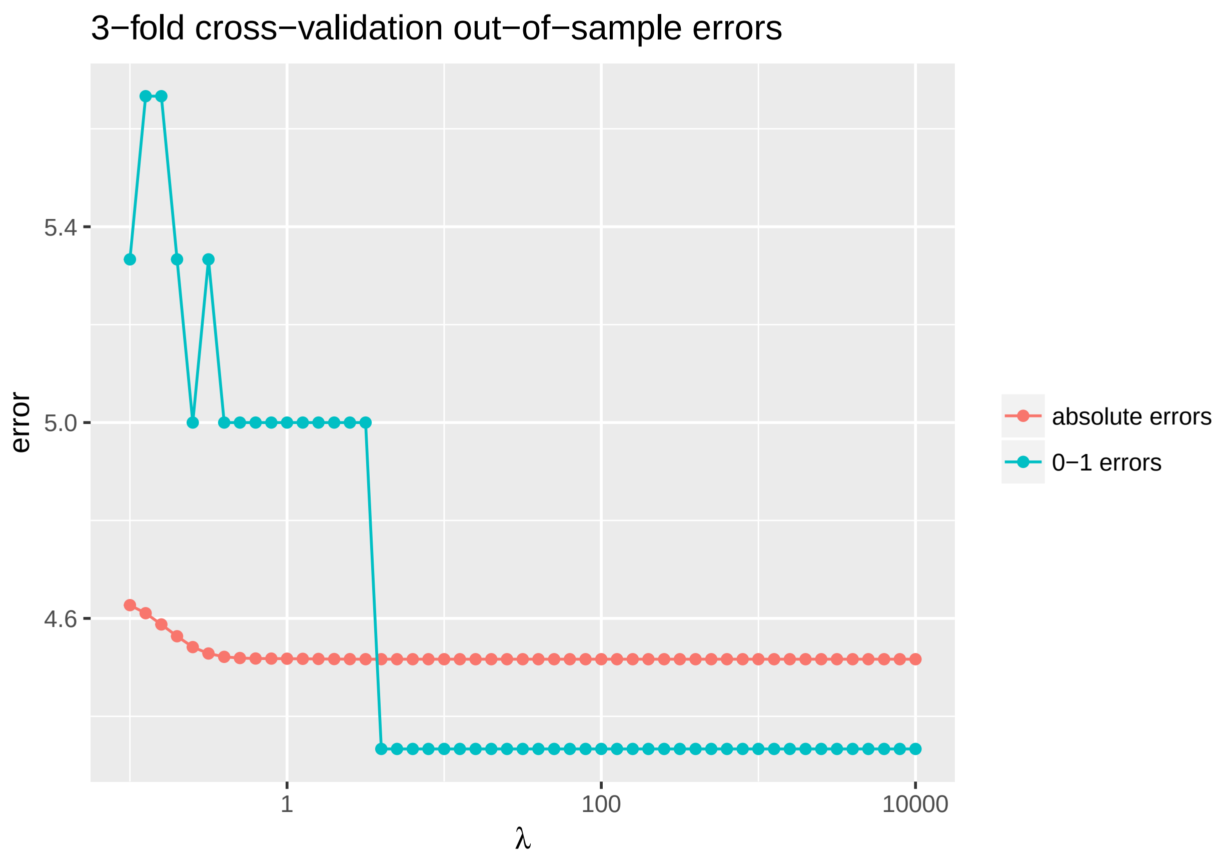
Now, let's create a transformed version of X where the values are on the log scale.

```r
# Copy it as to not damage the original
X_nozero_log = X_nozero
# Oooh I figured out a sweet hack. So the dgCMatrix class could be used in the
# following way:
# X@i are the row indices - 1 (that have ncol sections in non-decreasing order)
# X@p are the indices - 1 in X@i where we switch to a new column
```

```r
#
# OR
# X@x is the (non-zero) data itself, so why not just edit that? =D
# Here, add a small value because log(1) == 0, and we want to maintain a value there.
# #oneliner
X_nozero_log@x = log2(X_nozero_log@x + 0.1)
```

Now, let's try to use cross-validation (on the limited number of samples we have) to determine a good range for λ, the LASSO penalty factor in the stochastic gradient decent logistic regression code.

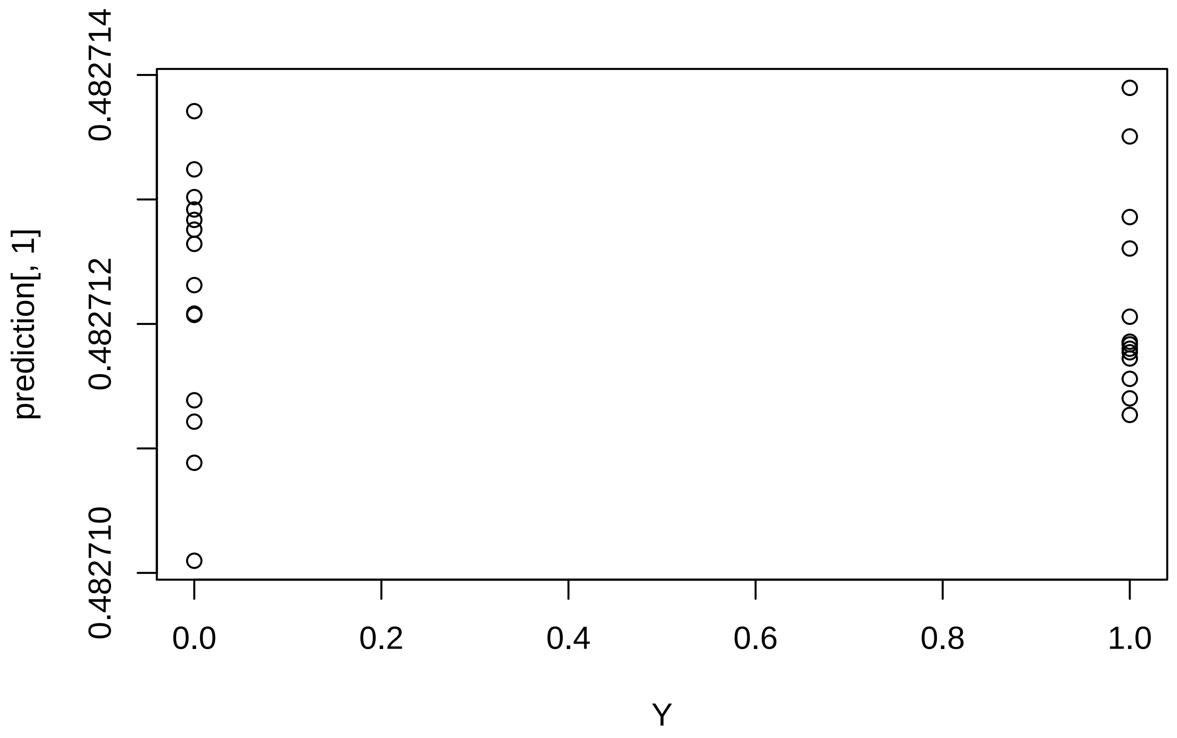
```r
# Initial guess for the betas.
beta_init = rep(0, P)
# For the weighting in the skip scenario for a feature
# TODO: algorithm is very sensitive to this value
eta = 0.0002
# passes through the data
nIter = 100
# exponentially-weighted moving average factor
# TODO: could explore other values.
discount = 0.01

# X_cv = X_nozero
X_cv = X_nozero_log

folds = 3
col_breaks = cut(1:length(Y), folds, labels = FALSE)
abs_errors = c()
zero_one_errors = c()
lambdas = 10 ^ seq(-1, 4, 0.1)
for (lambda in lambdas) {
  absFoldErrors = c()
  zeroOneFoldErrors = c()
  for (fold in 1:folds) {
    test_ind = which(col_breaks == fold)
    train_ind = which(col_breaks != fold)
    X_nozero_train = X_cv[ , train_ind]
    Y_train = Y[train_ind]
    X_nozero_test = X_cv[ , test_ind]
    Y_test = Y[test_ind]

    train_result = sparsesgd_logit(X_nozero_train, Y_train, M[train_ind], eta, nIter, beta_init, lambda

    intercept = train_result$alpha
    beta = train_result$beta
    prediction_test = 1/(1 + exp(-(intercept + t(X_nozero_test) %*% beta)))

    # metrics to choose from
    zero_one_error_test = sum(Y_test != round(prediction_test))
    abs_error_test = sum(abs(Y_test - prediction_test))

    absFoldErrors = c(absFoldErrors, abs_error_test)
    zeroOneFoldErrors = c(zeroOneFoldErrors, zero_one_error_test)
  }
  abs_errors = c(abs_errors, mean(absFoldErrors))
  zero_one_errors = c(zero_one_errors, mean(zeroOneFoldErrors))
}
```



So based on 3-fold cross-validation, the absolute error $\sum_i |y_i - \hat{y}_i|$ seems to reach its best (minimal) value when $\lambda = 10^{-0.4}$ while the 0-1 error $\sum_i \delta\left(y_i,\ \mathrm{round}(\hat{y}_i)\right)$ (number of incorrect predictions) exhibits the same behavior around $\lambda = 10^{0.6}$.

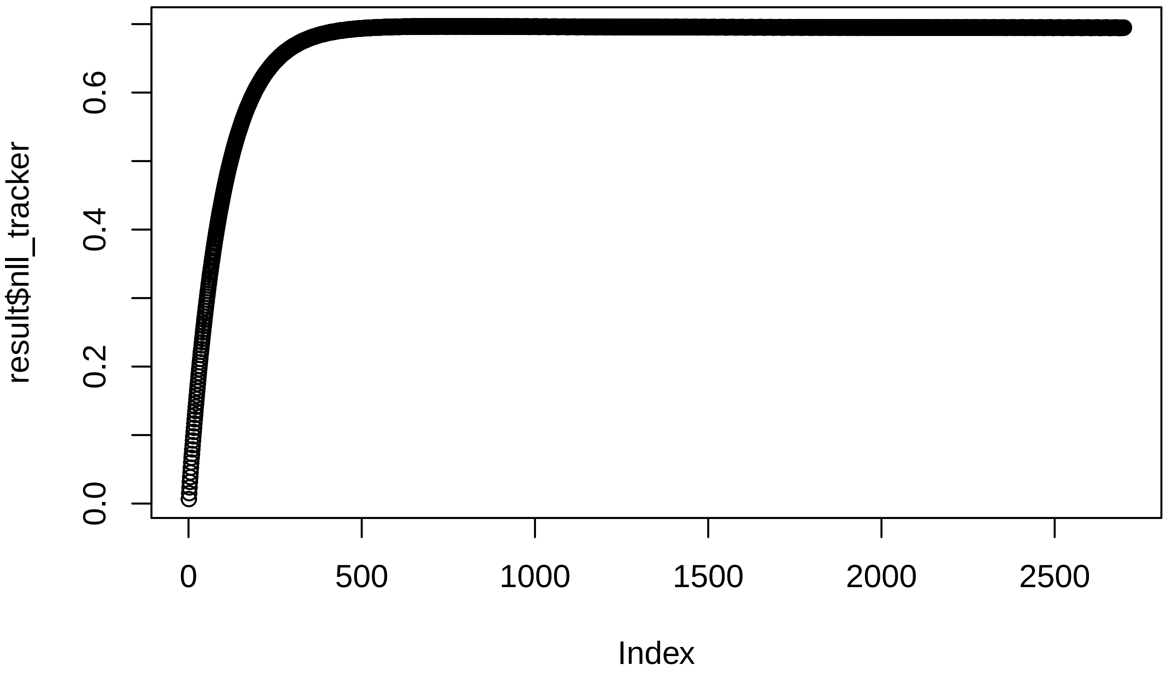
Let's take the latter.

```
# Initial guess for the betas.
beta_init = rep(0, P)
# For the weighting in the skip scenario for a feature
# TODO: algorithm is very sensitive to this value
eta = 0.0002
# passes through the data
nIter = 100
# L1 regularization
lambda = 10^0.6
# exponentially-weighted moving average factor
# TODO: could explore other values.
discount = 0.01

# Run the algorithm
result = sparsesgd_logit(X_nozero_log, Y, M, eta, nIter, beta_init, lambda, discount)
intercept = result$alpha
beta = result$beta
prediction = 1/(1 + exp(-(intercept + t(X_nozero_log) %*% beta)))
plot(prediction[ , 1] ~ Y)
```



So, these predictions are, to use a technical term, “totally garbage.” Every single prediction is just slightly below $\frac{1}{2}$ so the algorithm is effectively saying, “I dunno, but maybe this data is control not treatment” so while the results below are potentially a starting point, it’d be rather foolish to grant them much authority.

```
## How many genes are in the final model?  4
## Which genes?
## 3.4.11.9
## 3.4.24.-
## 2.7.13.3
## 1.4.99.-
```



DESeq2

Now, let's use a pre-fabricated solution instead.

```
library(DESeq2)
#library(BiocParallel)
# Allow for parallelization of DESeq2 code.
#register(MulticoreParam(4))

# Simple function for formatting DESeq2 results
get_results = function(d) {
  nms = rownames(d)
  log2change = d[ , 2]
  padj = d[ , 6]
  idx = order(nms)
  return(data.frame(names = nms[idx], log2change = log2change[idx], padj = padj[idx]))
}

# Get the counts
count_data_raw = raw_data[ , 2:ncol(raw_data)]

########################################
#  Format the data in the way DESeq2 expects
########################################
new_colnames = rep(NULL, N)
condition = rep(NULL, N)
control = 0
treatment = 0
for (i in 1:N) {
  base = c("control", "treatment")[Y[i] + 1]
  val = 0
  if (base == "control") {
    control = control + 1
    val = control
  } else {
    treatment = treatment + 1
    val = treatment
  }
  new_colnames[i] = paste0(base, val)
  condition[i] = base
}
count_data = count_data_raw
colnames(count_data) = new_colnames
rownames(count_data) = gene

col_data = data.frame(condition)
rownames(col_data) = new_colnames

# remove the unobserved genes from the get-go
count_data = count_data[which(rowSums(data) > 0), ]

# Run DESeq2
dds = DESeqDataSetFromMatrix(countData = count_data,
                             colData = col_data,
                              design = ~ condition)
dds = DESeq(dds)#, parallel = TRUE)
res = results(dds)#, parallel = TRUE)

# plot(sort(res$pvalue))
# points(sort(res$padj), col = "red")
# P - sum(is.na(res$pvalue))
# P - sum(is.na(res$padj))
```

```
## How many genes are in the final model?  20
```

```
## [1] "Which genes in default DESeq2?"
```

```
##          names log2change         padj
## 1     1.1.1.36 -1.7920085 0.0984365586
## 2     1.14.-.- -1.4462186 0.0994066615
## 3    1.14.14.- -1.3495409 0.0984365586
## 4   1.14.14.10 -2.0835574 0.0781124498
## 5     1.2.1.39 -1.7601778 0.0372464266
## 6     1.2.99.8 -1.9174369 0.0679078389
## 7     1.3.99.5 -2.5358347 0.0520005061
## 8    2.1.1.140 -2.4009629 0.0546184085
## 9     2.7.7.73 -1.7331908 0.0984365586
## 10    3.1.3.21 -2.6488779 0.0399822976
## 11    3.4.11.-  0.9773685 0.0040824877
## 12    3.5.1.54 -2.0246868 0.0372464266
## 13     3.6.3.9  2.4123300 0.0372464266
## 14   4.2.1.153 -2.0440501 0.0994066615
## 15     5.1.1.7 -2.8294091 0.0006806538
## 16     5.3.3.- -2.1665531 0.0546184085
## 17   5.4.99.26 -1.6481080 0.0396149550
## 18     6.-.-.- -1.9153744 0.0984365586
## 19     6.3.3.3 -2.5694759 0.0658531485
## 20     6.4.1.6 -1.6692874 0.0784474253
```

```
# Not exactly sure what this plot does either.
# plotMA(res, main="DESeq2", ylim=c(-3,3))

# Benjamini-Hochberg by hand (DESeq2 does something similar...)
alpha = 0.1
sorted_pval = sort(res$pvalue, na.last = TRUE)
numNA = sum(is.na(res$pvalue))
bh = sapply(1:P, function(i){ sorted_pval[i] <= alpha * i / (P - numNA)})
threshold = sorted_pval[max(which(bh))]
```

```
## How many genes in customized Benjamini-Hochberg procedure
## 13
```

```
## [1] "Which genes in customized Benjamini-Hochberg procedure?"
```

```
##          names log2change         padj
## 1   1.14.14.10 -2.0835574 0.0781124498
## 2     1.2.1.39 -1.7601778 0.0372464266
## 3     1.2.99.8 -1.9174369 0.0679078389
## 4     1.3.99.5 -2.5358347 0.0520005061
## 5    2.1.1.140 -2.4009629 0.0546184085
## 6    3.1.3.21 -2.6488779 0.0399822976
## 7    3.4.11.-   0.9773685 0.0040824877
## 8    3.5.1.54 -2.0246868 0.0372464266
## 9     3.6.3.9  2.4123300 0.0372464266
## 10    5.1.1.7 -2.8294091 0.0006806538
## 11    5.3.3.- -2.1665531 0.0546184085
## 12  5.4.99.26 -1.6481080 0.0396149550
## 13    6.3.3.3 -2.5694759 0.0658531485
```

```
# These two are equal -- throws out genes that were never observed section 1.5.3 of DESeq2 paper
print(paste(sum(rowSums(count_data) == 0),
            sum(is.na(res$pvalue))))
```

```
## [1] "0 0"
```

```
# 118687 genes are thrown out by the "independent filtering" for having a low
# mean normalized count
sum(is.na(res$padj)) - sum(is.na(res$pvalue))
```

```
## [1] 347
```

```
# Not entirely sure what this plot means.
# plot(metadata(res)$filterNumRej,
#      type = "b", ylab = "number of rejections",
#      xlab = "quantiles of filter")
# lines(metadata(res)$lo.fit, col = "red")
# abline(v = metadata(res)$filterTheta)

#nofilter
# If you turn off the filtering, only two of the adjusted p-values are less than 0.1
# where the adjustment basically just accounts for Benjamini-Hochberg, I think
resNoFilt <- results(dds, independentFiltering = FALSE)
addmargins(table(filtering = (res$padj < .1),
                 noFiltering = (resNoFilt$padj < .1)))
```

```
##          noFiltering
## filtering FALSE TRUE  Sum
##     FALSE  1860    0 1860
##     TRUE      6   14   20
##     Sum    1866   14 1880
```

```
## How many genes without independent filtering
## 14
```

```
## [1] "Which genes without independent filtering?"
```

```
##         names log2change         padj
## 1  1.14.14.10 -2.0835574 0.0925300137
## 2    1.2.1.39 -1.7601778 0.0441211660
## 3    1.2.99.8 -1.9174369 0.0804418921
## 4    1.3.99.5 -2.5358347 0.0615984718
## 5   2.1.1.140 -2.4009629 0.0646995722
## 6    3.1.3.21 -2.6488779 0.0473620089
## 7    3.4.11.-   0.9773685 0.0048360107
## 8    3.5.1.54 -2.0246868 0.0441211660
## 9     3.6.3.9  2.4123300 0.0441211660
## 10    5.1.1.7 -2.8294091 0.0008062851
## 11    5.3.3.- -2.1665531 0.0646995722
## 12  5.4.99.26 -1.6481080 0.0469268643
## 13    6.3.3.3 -2.5694759 0.0780079584
## 14    6.4.1.6 -1.6692874 0.0929268171
```