Homework 2 - Analysis of High Dimensional Data

Tavis Abrahamsen, Ray Bai, Syed Rahman and Andrey Skripnikov

Logistic Regression with Lasso Penalty Consider a classification problem, where the response is binary $y \in \{0, 1\}$ and the predictor variables numerical. Generate a design matrix X of size 5,000 by 300 (i.e. 5,000 observations and 300 variables). Generate the corresponding response y by selecting an appropriate coefficient vector β and adding a noise term.

Consider a model where the true coefficient vector has only 20 non-zero elements and another with 75 non-zero elements. Devise a sub-gradient based algorithm (soft-thresholding) to solve the problem, where the Gram matrix $X'X$ is well-conditioned.

Compare the performance of your algorithm for a grid of the tuning parameter λ.

We are interested in solving the logistic regression from last time with a *lasso* penalty added on. Recall that last time our objective was to minimize the negative log likelihood

$$-l(\beta) = -\sum_{i=1}^{n} [y_i \log(\pi_i) + (1 - y_i) \log(1 - \pi_i)]$$

where

$$\log(\frac{\pi_i}{1 - \pi_i}) = x_i'\beta.$$

While $\nabla(-l(\beta)) = 0$ has no closed form solutions, it is a convex function as we showed last time and hence we can use the gradient descent algorithm

$$\beta^{k+1} = \beta^k - \alpha^k \nabla(-l(\beta^k))$$

where α^k is a suitably chosen stepsize and $-\nabla l(\beta^k) = -X'(y - \frac{e^{X\beta}}{1+e^{X\beta}})$. This time we have an added constraint. The primal problem we are interested in solving can be formulated as:

$$\textbf{PRIMAL:} \min_{\beta \in \mathbb{R}^p} -l(\beta)$$
$$\text{subject to } \|\beta\|_1 \leq t$$

which is a convex problem. Hence we can invoke Slater's condition as $\exists$ a strictly feasible β. Hence there is no duality gap and we can instead solve the dual problem:

$$\begin{aligned}\textbf{DUAL: } &\max g(\lambda) = \max \inf_{\beta \in \mathbb{R}^p} (-l(\beta) + \lambda(\|\beta\|_1 - t)) \\ &\text{subject to } \lambda \geq 0\end{aligned}$$

As $g(\lambda)$ is not differentiable, we have to use sub-differentials. For β to satisfy $\inf_{\beta \in \mathbb{R}^p}(-l(\beta) + \lambda(\|\beta\|_1 - t))$, we must have that $\nabla l(\beta) = \lambda \text{sign}(\beta)$, where

$$(\text{sign}(\beta))_i = \text{sign}(\beta_i) = \begin{cases} +1 \text{ if } \beta_i > 0 \\ 0 \text{ if } \beta_i = 0 \\ -1 \text{ if } \beta_i < 0 \end{cases}$$

This is because a sub-gradient of $-l(\beta) + \lambda(\|\beta\|_1 - t)$ is $-\nabla l(\beta) + \lambda \text{sign}(\beta)$. An interesting point in this case is that the sub-gradient is independent of t. Thus we use the following sub-gradient descent scheme:

$$\begin{aligned}\beta^{k+1} &= \beta^k - \alpha^k(-\nabla l(\beta^k) + \lambda \text{sign}(\beta^k)) \\ &= \beta^k - \alpha^k(-X'(y - \frac{e^{X\beta^k}}{1 + e^{X\beta^k}}) + \lambda \text{sign}(\beta^k))\end{aligned}$$

and β_{est}^{k+1} such that

$$\beta_{est}^{k+1} = \arg \min_{i=1,\ldots,k+1} (-l(\beta^i) + \lambda \|\beta^i\|_1).$$

In the last step, we ignore the value of t because for every t in the primal problem, there is a corresponding λ in the dual problem. However, we are not interested in using a particular λ corresponding to a pre-specified t. Thus we can ignore t from the objective function without affecting the results.

An important difference from gradient descent is that the choice of step-size has to be pre-specified. Due to the superior asymptotic properties, we choose diminishing step-sizes $\alpha^k = \frac{1}{k}$ such that $\sum_{i=1}^k (\alpha^k)^2 < \infty$ and $\sum_{i=1}^k \alpha^k = \infty$.

In generating our data we essentially used the methods from last time to generate a design matrix $X \in \mathbb{R}^{5000 \times 300}$ such that $X'X$ has condition number,

$CN(X) = 1$. We can do this by generating a random matrix, $A \in \mathbb{R}^{5000\times300}$, and doing an SVD to get $A = U\Sigma V'$, where $U \in \mathbb{R}^{5000\times5000}$ and $V \in \mathbb{R}^{300\times300}$ are orthonormal matrices of the right dimensions. Then let

$$XU\begin{pmatrix} I \\ 0 \end{pmatrix} V' = U\begin{pmatrix} V' \\ 0 \end{pmatrix}.$$

Thus $CN(X) = CN(U)CN(V) = 1$ and generate $\beta_1 \sim \mathcal{N}_d(0, 2I_{d\times d})$ with $d = 20$ or 75 being the number of non-zero elements. Now let

$$\beta = \begin{pmatrix} \beta_1 \\ 0_{(p-d)\times1} \end{pmatrix}$$

and

$$\pi_i = \frac{e^{x_i'\beta}}{1 + e^{x_i'\beta}}, \quad i = 1, ..., n.$$

Then generate $y_i \overset{ind}{\sim} Bin(1, \pi_i)$. The value of λ's used where from 0.1 through to 2.0 incremented by 0.1 at each step. While there is no good stopping criteria, we tried to implement one based on Epsilon at the k-th iteration, or $\epsilon_k < 10^{-5}$ where

$$\epsilon_k \coloneqq \|\beta_{est} - \beta_k\|_2.$$

and the maximum number of iterations was set to 10,000. Unfortunately, the stopping criteria was almost never reached, and the program usually ran for the maximum number of iterations.

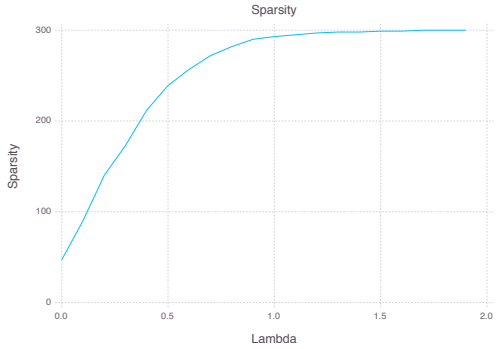
We defined the optimal λ,

$$\lambda_{opt} \coloneqq \min\{\lambda_1\} \text{ such that } \lambda_1 = \arg\min_{\lambda} \|\beta_{est}^{\lambda} - \beta_{true}\|_2,$$

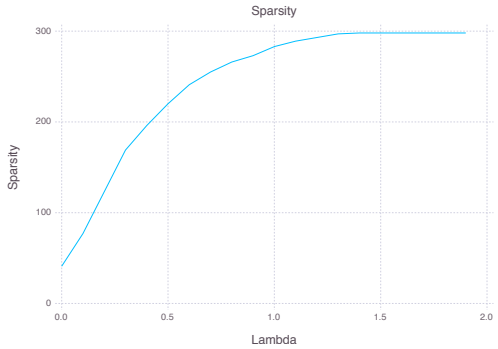
that is, the smallest value of λ at which the the norm of $\beta_{est}^{\lambda} - \beta_{true}$ is minimized. These results are presented below. When the number of non-zero elements in β_{true} was 20, $\lambda_{opt} = 1.0$. The diagnostic plots in this case are shown in Figure 4a,4b and 4c. However β_{est} gets closest to the desired level of sparsity at a slightly smaller value of $\lambda = 0.8$ as can be seen in Figure 1a. These diagnostic plots are shown in Figure 5a,5b and 5c. Additionally, as Figure 2a shows, the norm of the estimated beta from the true beta is slightly below 7.5. Finally, the negative log likelihood plus the penalty term reaches around 3,460 as shown in Figure 4c. While a higher value of λ seems to lead to marginally higher values ot this term, it hardly seems to matter

after $\lambda = 1.0$. When λ was very small (≤ 0.2), the program did terminate after a few thousand iterations. However, as mentioned before, in most cases, the program ran for $10,000$ iterations.

When the number of non-zero elements in β_{true} was 75, the program never seemed to reach the stopping criteria and reached the maximum number of iterations allowed at 10,000 for all λ. Increasing the maximum number of iterations to 100,000 didn't seem to make a difference in this regard either. The best results were at $\lambda_{opt} = 0.7$ in this case and these diagnostic plots are shown in Figure 6a, 6b and 6c. However, the desired level of sparsity is reached at $\lambda = 0.5$, whose plots are shown in Figure 7a, 7b and 7c. In the case of λ_{opt}, the normed difference between the estimated beta from the true beta dips slightly below 15, as can be seen in Figure 2b. As expected and similar to the case of 20 non-zero elements in β_{true}, Figure 1b shows that our measure of sparsity and shrinkage keeps on increasing towards 300 as λ increases. In this case, the negative log likelihood plus the penalty term reaches a level that is slightly below 3,460 and from thereon out keeps increasing at a very slow rate as λ increases as shown in Figure 6c.

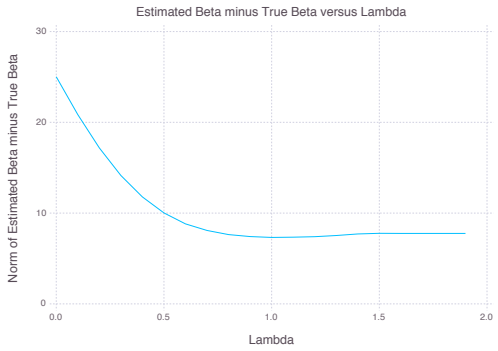


(a) A plot of the sparsity/shrinkage versus λ. The criteria for sparsity/shrinkage was the number of elements of β_{est} smaller than 10^{-3} when β_{true} had 20 nonzero elements. The desired level of sparsity is reached at slightly below $\lambda = 1$.

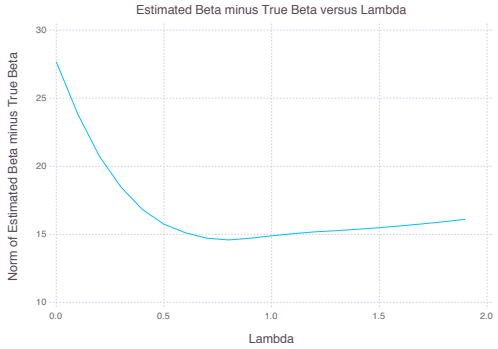


(b) A plot of the sparsity/shrinkage versus λ. The criteria for sparsity/shrinkage was the number of elements of β_{est} smaller than 10^{-3} when β_{true} had 75 nonzero elements. The desired level of sparsity is reached at $\lambda = 0.5$.

Figure 1: A plot of the sparsity/shrinkage versus λ.

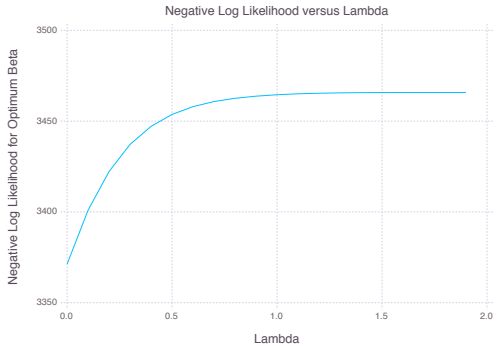


(a) A plot of the norm of $\beta_{true} - \beta_{est}$ versus λ when β_{true} had 20 non-zero elements. $\lambda = 1.0$ minimizes the norm.

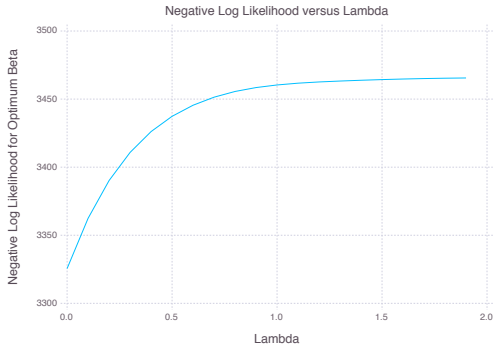


(b) A plot of the norm of $\beta_{true} - \beta_{est}$ versus λ when β_{true} had 75 non-zero elements. In this case it is clear that $\lambda = 0.7$ minimizes the norm.

Figure 2: A plot of the norm of $\beta_{true} - \beta_{est}$ versus λ.

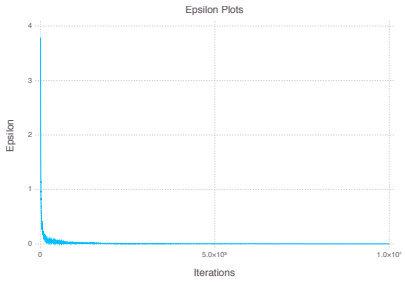


(a) A plot of the negative log-likelihood versus λ when β_{true} had 20 non-zero elements. It seems to reach a maximum at $\lambda = 1.0$ and then stay at that level.



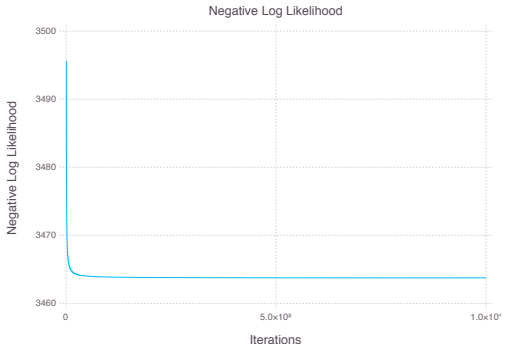
(b) A plot of the negative log-likelihood versus λ when β_{true} had 75 non-zero elements. As in the case with 25 non-zero elements, it seems to reach a maximum at $\lambda = 1.0$ and then increase at a very slow rate.

Figure 3: Negative log-likelihood versus λ.



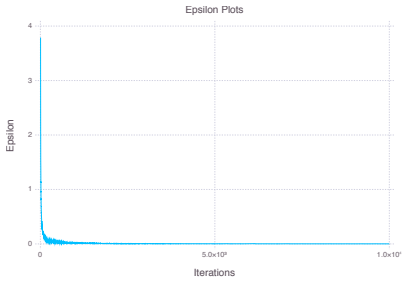
(a) Epsilon for $\lambda = 1$

(b) Norm of $\beta_{true} - \beta_{est}$ when $\lambda = 1$



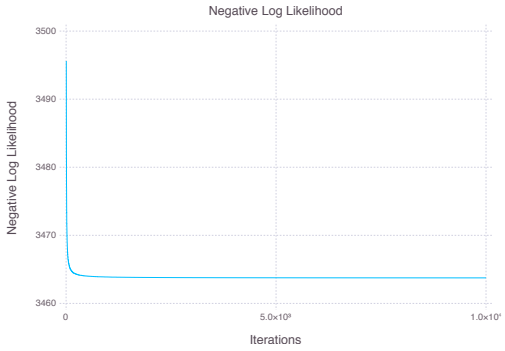
(c) Negative Log-likelihood when $\lambda = 1$

Figure 4: Plots for Subgradient Descent Algorithm with β_{true} having 20 non-zero elements and $\lambda = 1$.



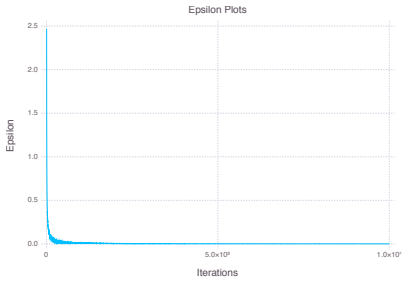
(a) Epsilon for $\lambda = 0.8$

(b) Norm of $\beta_{true} - \beta_{est}$ when $\lambda = 0.8$

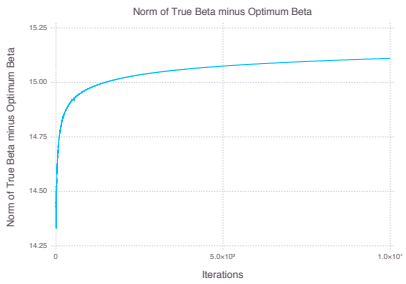


(c) Negative Log-likelihood when $\lambda = 0.8$

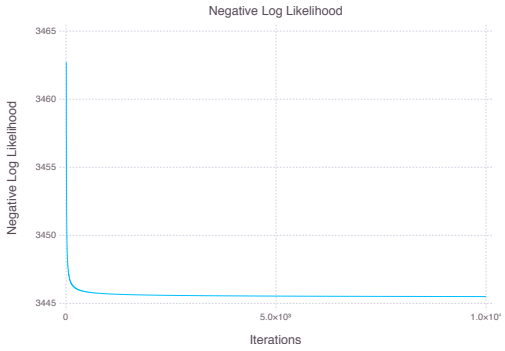
Figure 5: Plots for Subgradient Descent Algorithm with β_{true} having 20 non-zero elements and $\lambda = 0.8$.



(a) Epsilon for $\lambda = 0.7$

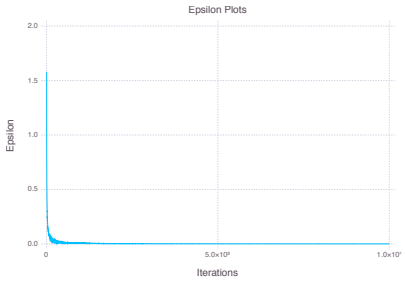


(b) Norm of $\beta_{true} - \beta_{est}$ when $\lambda = 0.7$

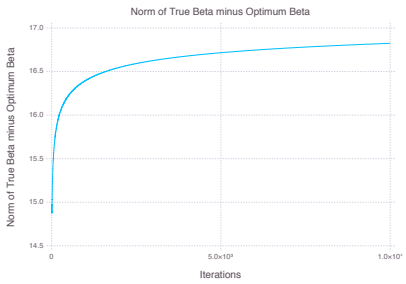


(c) Negative Log-likelihood when $\lambda = 0.7$

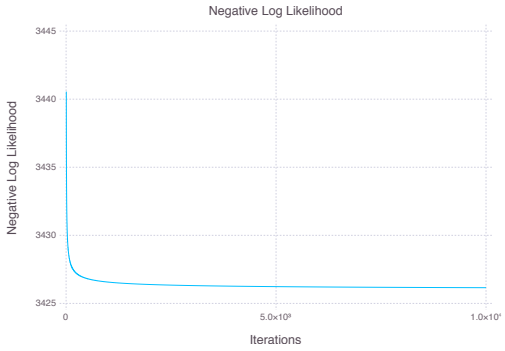
Figure 6: Plots for Subgradient Descent Algorithm with β_{true} having 75 non-zero elements and $\lambda = 0.7$.



(a) Epsilon for $\lambda = 0.7$



(b) Norm of $\beta_{true} - \beta_{est}$ when $\lambda = 0.7$



(c) Negative Log-likelihood when $\lambda = 0.7$

Figure 7: Plots for Subgradient Descent Algorithm with β_{true} having 75 non-zero elements and $\lambda = 0.5$.

Appendix:Julia code

```
using PyPlot
using Distributions
using Gadfly


srand(12345)
print("enter p "); p = int(readline(STDIN))
print("enter n  "); n = int(readline(STDIN))
print("enter notsparse  "); notsparse = int(readline(STDIN))

function gradf(X::Matrix{Float64},y::Vector{Int},
p::Vector{Float64}, lambda::Float64,beta::Vector{Float64})
    -X'*(y-p)+lambda*sign(beta)
end

function neglogLkhd(probOld::Vector{Float64},
y::Vector{Int}, lambda::Float64, betaOld::Vector{Float64})
    n = length(y)
    z = zeros(n)
    p = length(betaOld)
    penaltyterm = zeros(p)
    for i = 1:n
            term1 = y[i]*log(probOld[i])
            term2 = (1-y[i])*log(1-probOld[i])
            z[i] = -term1-term2
        end
     for i = 1:p
            penaltyterm[i] = lambda*abs(betaOld[i])
        end
(sum(z) + sum(penaltyterm))
end

expit(x::Float64) = exp(x)/(1+exp(x))
expit(V::Vector{Float64}) = [expit(v) for v in V]

rbern(p::Float64) = 0<p<1?rand(Bernoulli(p))
```

```
:error("p not in (0,1)")
rbern(V::Vector{Float64}) = [rbern(v) for v in V]

dataMat = rand(n,p)
(U,S,V) = svd(dataMat); @elapsed svd(dataMat)
#d = rand(Uniform(0,sqrt(30)),p)
#X = U*diagm(d)*V'
X = U*V'
betaTrue = rand(Normal(0,2),notsparse)
betaTrue =  [betaTrue;zeros((p-notsparse))]
Xbeta = X*betaTrue
probTrue = expit(Xbeta)
y = rbern(probTrue)

maxcount = 20
sparsityvec = zeros(maxcount)
for count = 1:maxcount
lambda = 0.1*count
maxIter = 100000
betaOld = zeros(p)+1
Xbeta = X*betaOld
probOld = expit(Xbeta)
alpha = 1
tol = 10.0^-5
iter = 0
eps = 1
betamat = betaOld
betaopt = betaOld
Xbeta = X*betaOld
probopt = expit(Xbeta)

neglogLkhdvec = neglogLkhd(probopt,y,lambda,betaopt)
epsvec = eps
trueerror = sqrt(dot(betaopt-betaTrue,betaopt-betaTrue))
trueerrorvec = trueerror

while iter< maxIter && eps > tol
betaNew = betaOld - (alpha/(iter+1))*gradf(X,y,
```

```
probOld,lambda, betaOld)
Xbeta = X*betaNew
     probNew = expit(Xbeta)
     eps = sqrt(dot(betaNew-betaopt,betaNew-betaopt))
trueerror = sqrt(dot(betaopt-betaTrue,betaopt-betaTrue))

if neglogLkhd(probNew,y,lambda,betaNew)
<neglogLkhd(probopt,y,lambda,betaopt)
           betaopt = betaNew
           Xbeta = X*betaopt
           probopt = expit(Xbeta)
        end

betaOld = betaNew
probOld = probNew

if (iter%100) == 0
   println("Iteration:",iter," " ,
neglogLkhd(probOld,y,lambda,betaOld)," ",eps,"\n")
end

neglogLkhdvec = [neglogLkhdvec
neglogLkhd(probopt,y,lambda,betaopt)]
epsvec = [epsvec eps]
trueerrorvec = [trueerrorvec trueerror]
iter = iter+1
end
end
```