COMP 790-124, HW2

Wile E. Coyote

September 26, 2013

Deadline: 10/10/2013 11:59PM EST
Submit `hw2.pdf` by e-mail to `vjojic+comp790+hw2@cs.unc.edu`

In the next few problems we will derive and implement an ADMM algorithm for logistic regression with a fused lasso penalty. You can refer to the ADMM write up available on the course website `http://www.cs.unc.edu/~vjojic/comp790/notes/ADMMFLSA.pdf`, but note that, while there are similarities with the problems below, there are differences as well.

Problem 1(1pt) The optimization problem for logistic regression with a fused lasso penalty is given as

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & \sum_i \log\left\{1+\exp\left\{-y_i(\mathbf{x}_i\mathbf{w})\right\}\right\} + \\ & \lambda\left\|\mathbf{w}\right\|_1 + \mu\left\|\mathbf{D}\mathbf{w}\right\|_1. \end{aligned}$$

We are going to introduce new variables $\mathbf{z}^0, \mathbf{z}^1, \mathbf{z}^2$ and reformulate the problem

$$\begin{aligned} \underset{\mathbf{w},\mathbf{z}_0,\mathbf{z}_1,\mathbf{z}^2}{\text{minimize}} \quad & \sum_i \log\left\{1+\exp\left\{-y_i\mathbf{z}^0\right\}\right\} + \lambda\left\|\mathbf{z}^1\right\|_1 + \mu\left\|\mathbf{z}^2\right\|_1 \\ \text{subject to} \quad & \mathbf{z}_i^0 = \mathbf{x}_i\mathbf{w}, i = 1,\dots n \\ & \mathbf{z}^1 = \mathbf{w} \\ & \mathbf{z}^2 = \mathbf{D}\mathbf{w} \end{aligned}$$

Write out the augmented lagrangian for the above problem

$$\begin{aligned} \mathrm{AL}(\mathbf{w},\mathbf{z}^0,\mathbf{z}^1,\mathbf{z}^2,\mathbf{u}^0,\mathbf{u}^1,\mathbf{u}^2) &= \boxed{\texttt{answer}} \\ &+ \boxed{\texttt{answer}} \\ &+ \boxed{\texttt{answer}} \\ &+ \boxed{\texttt{answer}} \\ &+ \boxed{\texttt{answer}} \end{aligned}$$

Hint: Make sure that you have all the terms from the objective, that is logistic regression and the two norms. For each constraint you should have a term that involves the dual variables, for example $u_i^0(z_i^0 - \mathbf{x}_i\mathbf{w})$, and a term that augments the lagrangian, for example $\frac{\rho}{2}\left\|z_i^0 - \mathbf{x}_i\mathbf{w}\right\|_2$.

Problem 2(1pt) Implement function that computes the value of augmented lagrangian

```
function val = computeAL(y,X,D,lambda,mu,rho,w,z0,z1,z2,u0,u1,u2)
val = ...
```

Problem 3(1pt) We will use superscript k to indicate the iteration, so for example $\mathbf{w}^k$ denotes the state of variable $\mathbf{w}$ in the k^{th} iteration. Collect the terms in the augmented lagrangian that involve $\mathbf{w}$, complete the square and plug in the resulting expression below

$$\mathbf{w}^k \quad = \quad \underset{\mathbf{w}}{\text{argmin}} \boxed{\texttt{answer}}$$

When completing squares here do it so that $\mathbf{w}, \mathbf{u}_0, \mathbf{z}_0$ occur under one, $\mathbf{w}, \mathbf{u}_1, \mathbf{z}_1$ occur under another, and $\mathbf{w}, \mathbf{u}_1, \mathbf{z}_2$ occur under yet another norm term.

Same thing for z_i^0, note that here we are considering just a single z_i^0 not the full vector. You can convince yourself that if you write out the objective for the full vector it is separable into objectives across z_i^0.

$$z_i^{0,k} \quad = \quad \underset{z_i^0}{\text{argmin}} \boxed{\texttt{answer}}$$

And once more for z_i^1, using the fact that $\left\|\mathbf{z}^1\right\|_1 = \sum_i |z_i^1|$

$$z_i^{1,k} \quad = \quad \underset{z_i^1}{\text{argmin}} \boxed{\texttt{answer}}$$

Finally, do the same for z_i^2

$$z_i^{2,k} \quad = \quad \underset{z_i^2}{\text{argmin}} \boxed{\texttt{answer}}$$

Problem 4(1pt) Copy over the update for $\mathbf{w}$ from above

$$\mathbf{w}^k \quad = \quad \underset{\mathbf{w}}{\text{argmin}} \boxed{\texttt{answer}}.$$

Note that $\mathbf{w} = \mathbf{I}\mathbf{w}$ and refer to the appendix of the Lecture 4 notes on how to stack these two linear systems, as well as code to do it. Of course, the ADMM writeup also has some of this. Implement a function that gives update for $\mathbf{w}$

```
function w = updatew(X,b,z0,z1,u0,u1,rho)
w = ...
```

Problem 5(1pt) To update z_i^0 we will be solving an optimization problem

$$\underset{z \in \mathbf{R}}{\text{minimize}} \log\{1 + \exp\{-yz\}\} + \gamma(z-a)^2.$$

Note that z is a scalar, so the objective is univariate. Here is a an implementation of Newton algorithm with backtracking, it takes a function handle and starting value as a input.

```
function z = newtonWolfeBacktrack(objective, initZ)
% newtonWolfeBacktrack solves an unconstrained optimization problem
%    using Newton's method with backtracking and the Weak Wolfe
%    condition for backtracking termination.
%
% z = newtonWolfeBacktrack(objective, initZ)
%   objective  a handle of function that take input z and returns
%              the values of the objective, the first, and the
%              second derivative at z
%   initZ      initial value of z
%
ALPHA = 0.05;
BETA = 0.5;
TOL = 1e-6;
MAX_ITER = 100;
z = initZ;
for it=1:MAX_ITER
    [val,d,dd] = objective(z);

    % Newton direction
    dz = -d/dd;

    % we are done if derivative is small enough
    if abs(dz) < TOL
        break;
    end

    % backtracking with the Wolfe condition check
    step = 1;
    while objective(z + step*dz) > val + ALPHA*step*dz*d
          step = step*BETA;
    end
    z = z + step*dz;
end
```

Save this file as `newtonWolfeBacktrack.m`. In the code above, the while loop is terminated when the Wolfe condition is satisfied. Read the code and explain what the Wolfe condition is trying to accomplish. The Wolfe condition is `answer` .

Now we will write a function that computes the objective and its derivatives.

```
function [val,d,dd] = logRegGaussPrior(z,y,a,gamma)
val = log(1 + exp(-y*z)) + gamma*(z-a)^2;
if nargout > 1 % caller asked for derivatives
    d = ... % first derivative
    dd = .... % second derivative
end
```

An example of how to use these two pieces together:

```
y = 1; a = 0.1; gamma = 0.5;
f = @(z) logRegGaussPrior(z,y,a,gamma);
z = newtonWolfeBacktrack(f,0);
```

Of course you will be plugging in different values of `y,a,gamma`. Note that `f` is really a function with some arguments to `logRegGaussPrior` hardcoded. In the above example `0.1` will be hardcoded in the definition of `f`. So even if you subsequently change variable `a`, the function `f` will still supply `0.1` as the third argument to `logRegGaussPrior`.

Use the `newtonWolfeBacktrack` and `logRegGaussianPrior` to implement

```
function z0i = updatez0i( ... )
...
```

Problem 6(1pt) Copy over update for z^1 from above

$$z_i^{1,k} = \underset{z_I^1}{\operatorname{argmin}} \boxed{\texttt{answer}}.$$

Refer to ADMM writeup on how to solve this in a closed form the problem of this shape and write it out here

$$z_i^{1,k} = \boxed{\texttt{answer}}.$$

write code to implement this update

```
function z1i = updatez1i( ... )
z1i = ...
```

Problem 7(1pt) Copy over update for z_i^2 from above

$$z_i^{2,k} = \underset{z_i^2}{\operatorname{argmin}} \boxed{\texttt{answer}}.$$

Refer to ADMM writeup on how to solve this in a closed form the problem of this shape and write it out here

$$z_i^{2,k} = \boxed{\texttt{answer}}.$$

write code to implement this update

```
function z1i = updatez2i( ... )
z2i = ...
```

Problem 8(2pt) Putting all of this together

```
function w = solveLogRegFusedLasso(y,X,D)
rho = 1;
N = length(y); assert(N == size(X,1));
p = size(X,2); assert(p == size(D,2));
e = size(D,1);
w = zeros(p,1);
z0 = zeros(N,1); u0 = zeros(N,1);
z1 = zeros(p,1); u1 = zeros(p,1);
z2 = zeros(e,1); u2 = zeros(e,1);
MAXIT = 100;

for it=1:MAXIT
    before = computeAL(y,X,D,lambda,mu,rho,w,z0,z1,z2,u0,u1,u2);
    w = updatew( ... );
    after = computeAL(y,X,D,lambda,mu,rho,w,z0,z1,z2,u0,u1,u2);
    assert(after < before + 1e-6);

    before = after;
    for i=1:length(y)
        z0(i) = updatez0i( ... );
    end
    after = computeAL(y,X,D,lambda,mu,rho,w,z0,z1,z2,u0,u1,u2);
    assert(after < before + 1e-6);

    before = after;
    for i=1:length(y)
        z1(i) = updatez1i( ... );
    end
    after = computeAL(y,X,D,lambda,mu,rho,w,z0,z1,z2,u0,u1,u2);
    assert(after < before + 1e-6);

    before = after;
    for i=1:size(D,1)
        z2(i) = updatez2i( ... );
    end
    after = computeAL(y,X,D,lambda,mu,rho,w,z0,z1,z2,u0,u1,u2);
    assert(after < before + 1e-6);

    u0 = ...
    u1 = ...
    u2 = ...
end
```

If any of the asserts fail, it is most likely that you have a bug in your update. Remember the updates are derived so as to minimize the augmented lagrangian.

If your update does not reduce the augmented lagrangian, this is what the asserts check, then there is a bug.

Load `hw2.mat` and run your code on `y,X,lambda,mu` in that environment. Report `w`

```
w = ...
```

Problem 9(1pt) Poisson distribution is used to model count data.

$$p(x = k|\lambda) = \frac{\lambda^k \exp -\lambda}{k!}.$$

Note that poisson models counts, including count of 0. Recall that $0! = 1$. Write it as an exponential family member

$$p(x = k|\eta) = \boxed{\texttt{answer}}.$$

Write out base measure, sufficient statistic, natural parameter η in terms of λ, log-normalizing function

$$\begin{aligned} h(x) &= \boxed{\texttt{answer}} \\ T(x) &= \boxed{\texttt{answer}} \\ \eta &= \boxed{\texttt{answer}} \\ A(\eta) &= \boxed{\texttt{answer}} \end{aligned}$$

Compute derivative of the log-normalizing function

$$\frac{\partial}{\partial \eta} A(\eta) = \boxed{\texttt{answer}}$$

Compute the inverse mapping of the derivative of the log-normalizing function

$$\left(\frac{\partial}{\partial \eta} A\right)^{-1} = \boxed{\texttt{answer}}$$

Let the observed counts be

$$\mathbf{x} = \begin{bmatrix} 1 \\ 4 \\ 5 \\ 10 \\ 11 \\ 3 \end{bmatrix}$$

The sample empirical mean of sufficient statistic is, plug-in value,

$$\sum_i \frac{1}{N} T(x_i) = \boxed{\texttt{answer}}.$$

Then the maximum likelihood estimate for η of the Poisson distribution in exponential family form is, plug-in value,

$$\left(\frac{\partial}{\partial \eta} A\right)^{-1} \left(\sum_i \frac{1}{N} T(x_i)\right) = \boxed{\texttt{answer}}.$$

Problem 10(1pt) Now to confirm that we did this right we are going to explicitly derive a maximum likelihood solution of Poisson distribution. The log-likelihood is

$$\mathrm{LL}(\eta; \mathbf{x}) = \sum_{i}^{n} \log h(x_i) + \eta T(x_i) - A(\eta)$$

Plug-in $h(x_i), T(x_i), A(\eta)$ for Poisson distribution.

$$\mathrm{LL}(\eta; \mathbf{x}) = \boxed{\texttt{answer}}.$$

Take a derivative of log-likelihood with respect to η and equate it to zero. Solve the equation for η

$$\eta^{\mathrm{ML}} = \boxed{\texttt{answer}}.$$

Problem 11(1pt) In Matlab, run following code

```
example = poissrnd(15,300,1);
hist(example,1:50)
xlabel('count');ylabel('number of times count was observed');
hwplotprep
print -dpdf aPoissonExample.pdf
```

Replace `emptiness.pdf` with `aPoissonExample.pdf` below.

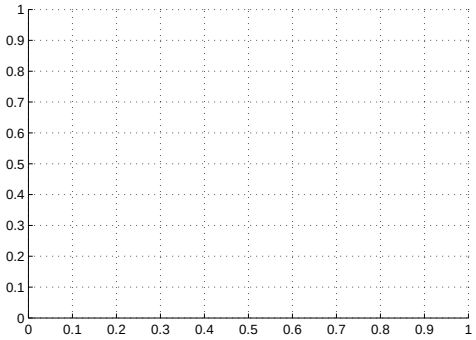


Figure 1: This is emptiness, it earns no points.

Load `hw2.mat` and run following code

```
hist(counts,1:50)
xlabel('count');ylabel('number of times count was observed');
hwplotprep
print -dpdf HW2Counts.pdf
```

Replace `emptiness.pdf` with `HW2Counts.pdf` below.

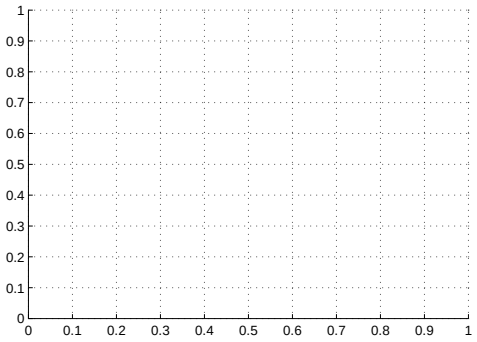
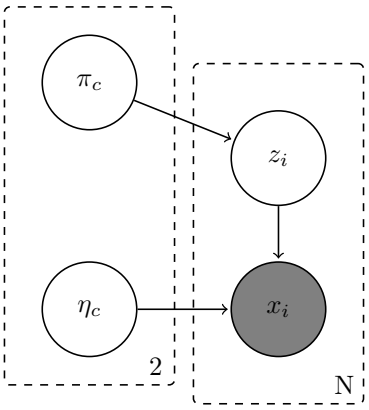


Figure 2: This is emptiness, it earns no points.

Comment on the differences between the two histograms and a possible cause `answer` .

Problem 12(1pt) We will model a vector of counts $\mathbf{x}$ as samples from a mixture of two Poisson distributions. The graphical model is simple



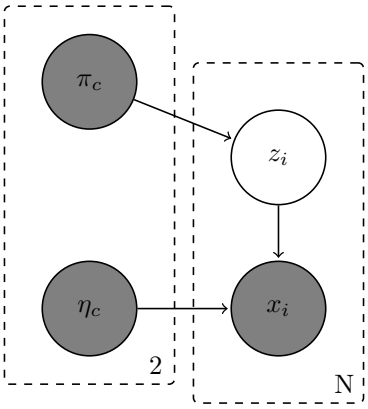
We will now derive and implement an EM algorithm for a mixture of two Poisson distributions.

$$\begin{aligned} p(z) &= \begin{cases} \pi_1, & z = 1 \\ 1 - \pi_2, & z = 2 \end{cases} \\ p(x|z) &= h(x) \exp \eta_z T(x) - A(\eta_z) \end{aligned}$$

Recall the structure of the an algorithm

$$\begin{aligned} E: q^{\text{new}} &= \operatorname*{argmax}_q \sum_z q(\mathbf{z}) \log p(\mathbf{x}, \mathbf{z}|\eta) - \sum_{\mathbf{z}} q(\mathbf{z}) \log q(\mathbf{z}) \\ M: \eta^{new} &= \operatorname*{argmax}_\eta \sum_{\mathbf{z}} q^{\text{new}}(\mathbf{z}) \log p(\mathbf{x}, \mathbf{z}|\eta) - \sum_{\mathbf{z}} q^{\text{new}}(\mathbf{z}) \log q^{\text{new}}(\mathbf{z}) \end{aligned}$$

In the case of exact EM $q(\mathbf{z}) = p(\mathbf{z}|\mathbf{x}, \eta)$, further we note that computing $p(\mathbf{z}|\mathbf{x}, \eta)$ in the above model corresponds to marginalization in the graphical model



Are z_i conditionally independent given π and $\mathbf{x}$? `answer` . Describe reasoning behind your observation using Bayes ball paths. `answer` .

Problem 13(1pt) We will now compute $q(z_i) = p(z_i|x_i, \eta)$ using Bayes rule

$$q(z_i) = p(z_i|x_i, \eta) = \boxed{\texttt{answer}}$$

Note that you can use π_{z_i} and η_{z_i} notation to denote parameters of class z_i. Write a Matlab function that computer $q(z_i)$

```
function qzi = computeQ(xi,etas,pis)
for c=1:2
    qzi(c) = log(...) %
end
qzi = exp(qzi - logsum(qzi));
```

Problem 14(1pt) Recall M-step, where we eliminated portions of the objective that do not depend on η

$$M: \eta^{new} = \operatorname*{argmax}_\eta \sum_{\mathbf{z}} q^{\text{new}}(\mathbf{z}) \log p(\mathbf{x}, \mathbf{z}|\eta)$$

Show that the above expression is equal to

$$M : \eta^{new} = \underset{\eta}{\text{argmax}} \sum_i q^{\text{new}}(z_i) \log p(x_i, \mathbf{z}|\eta)$$

We can say that

$$p(\mathbf{x}, \mathbf{z}|\eta) = \prod_i p(x_i, z_i|\eta)$$

because `answer` . Using the above equality and

$$\sum_{\mathbf{z}} q(\mathbf{z}) f(z_i) = \sum_{z_i} q(z_i) f(z_i)$$

we can rewrite the M-step update

$$M : \eta^{new} = \underset{\eta}{\text{argmax}} \boxed{\texttt{answer}}$$

Take the derivative of the objective under the argmax and set it to 0 to obtain updates

$$\begin{aligned} \eta_1 &= \boxed{\texttt{answer}} \\ \eta_2 &= \boxed{\texttt{answer}} \end{aligned}$$

write matlab code that implements M-step

```
function [etas,pis] = updateParameters(q,x)
pis = zeros(2,1);
for i=1:length(x)
    for c=1:2
        pis(c) = pis(c) + q(i,c);
    end
end
pis = pi/sum(pis);

for c=1:2
    for i=1:length(x)
        eta(c) = ...
    end
end
```

Problem 15(2pt) Putting it all together,

```
function [eta,pi] = EMMoP(x)
etas = log(mean(x) + 0.0001*rand(2,1));
pis = 0.5*ones(2,1);

for it=1:MAXIT
```

```
    before = computeBound(q,x,etas,pis)
    for i=1:length(x)
        q(:,i) = computeQ(x(i),etas,pis);
    end
    after = computeBound(q,x,etas,pis);
    assert(after > before);
    before = after;
    [etas,pis] = updateParameters(q,x);
    after = computeBound(q,x,etas,pis);
    assert(after > before);
    bounds(it) = bound;
    plot(bounds);
    drawnow;
end

function bound = computeBound(q,x,etas,pis)
for i=1:length(x)
    for c=1:2
        bound = bound + q(c,i)*(etas(c)*x(i)-exp(etas(c))+log(pis(c)));
    end
end
for i=1:length(x)
    for c=1:2
        bound = bound - q(c,i)*log(q(c,i));
    end
end
```

Load `hw2.mat` and run this code on `counts`. Report here resulting `eta,pi`

```
eta = ...
pi = ...
```

Problem 16(1pt) Now we convert the above EM algorithm to K-means analog for Poisson distribution. Change function `computeQ` you implemented earlier to assign all probability to the most likely class, so each `qzi` should be binary.

```
function qzi = computeQ(xi,etas,pis)
for c=1:2
    qzi(c) = ... %
end
```

Load `hw2.mat` and run `EMMoP` code on `x`. Report here resulting `etas,pis`

```
etas = ...
pis = ...
```