CPSC 340 Assignment 4

Tristan Rice, q7w9a, 25886145

1 Logistic Regression with Sparse Regularization

1.1 L2-Regularization

```
function [model] = logRegL2(X,y,lambda)

[n,d] = size(X);

maxFunEvals = 400; % Maximum number of evaluations of objective
verbose = 1; % Whether or not to display progress of algorithm
w0 = zeros(d,1);
model.w = findMin(@logisticLossL2,w0,maxFunEvals,verbose,X,y,lambda);
model.predict = @(model,X)sign(X*model.w); % Predictions by taking sign
end


function [f,g] = logisticLossL2(w,X,y,lambda)
yXw = y.*(X*w);
% Add an L2 regularizer.
f = sum(log(1 + exp(-yXw))) + w'*w*lambda/2; % Function value
g = -X'*(y./(1+exp(yXw))); % Gradient
end
```

```
At maximum number of function evaluations
numberOfNonZero =  101
trainingError = 0
validationError =  0.086000
```

1.2 L1-Regularization

```
function [model] = logRegL1(X,y,lambda)

[n,d] = size(X);

maxFunEvals = 400; % Maximum number of evaluations of objective
verbose = 1; % Whether or not to display progress of algorithm
w0 = zeros(d,1);
model.w = findMinL1(@logisticLoss,w0,lambda, maxFunEvals,verbose,X,y);
model.predict = @(model,X)sign(X*model.w); % Predictions by taking sign
end

function [f,g] = logisticLoss(w,X,y)
yXw = y.*(X*w);
f = sum(log(1 + exp(-yXw))); % Function value
g = -X'*(y./(1+exp(yXw))); % Gradient
end
```

```
Problem solved up to optimality tolerance
numberOfNonZero =  71
trainingError = 0
validationError =  0.052000
```

1.3 L0-Regularization

```
function [model] = logRegL0(X,y,lambda)
```

```
[n,d] = size(X);
maxFunEvals = 400; % Maximum number of evaluations of objective
verbose = 0; % Whether or not to display progress of algorithm
w0 = zeros(d,1);
oldScore = inf;

% Fit model with only 1 variable,
% and record 'score' which is the loss plus the regularizer
ind = 1;
w = findMin(@logisticLoss,w0(ind),maxFunEvals,verbose,X(:,ind),y);
score = logisticLoss(w,X(:,ind),y) + lambda*length(w);
minScore = score;
minInd = ind;

while minScore ~= oldScore
    oldScore = minScore;
    fprintf('\nCurrent set of selected variables (score = %f):',minScore);
    fprintf(' %d',ind);

    for i = 1:d
        if any(ind == i)
            % This variable has already been added
            continue;
        end

        % Fit the model with 'i' added to the features,
        % then compute the score and update the minScore/minInd
        ind_new = union(ind,i);

        w = findMin(@logisticLoss,w0(ind_new),maxFunEvals,verbose,X(:,ind_new),y);
        score = logisticLoss(w,X(:,ind_new),y) + lambda*length(w);
        if score < minScore
            minInd = ind_new;
            minScore = score;
        end
    end
    ind = minInd;
end

model.w = zeros(d,1);
model.w(minInd) = findMin(@logisticLoss,w0(minInd),maxFunEvals,verbose,X(:,minInd),y);
model.predict = @(model,X)sign(X*model.w); % Predictions by taking sign
end

function [f,g] = logisticLoss(w,X,y)
yXw = y.*(X*w);
f = sum(log(1 + exp(-yXw))); % Function value
g = -X'*(y./(1+exp(yXw))); % Gradient
end
```

```
nZero =  24
trainingError = 0
validationError =  0.018000
```

2 Convex Functions and MLE/MAP Loss Functions

2.1 Showing Convexity from Definitions

2.1.1 Quadratic

$$f(w) = aw^2 + bw$$
$$f'(w) = 2aw + b$$
$$f''(w) = 2a > 0$$

Since the second derivative is positive, $f(w)$ is convex.

2.1.2 Negative logarithm

$$f(w) = -\log(aw)$$
$$f'(w) = -\frac{a}{aw} = -\frac{1}{w}$$
$$f''(w) = \frac{1}{w^2}$$

Since $w > 0$, $f''(w) > 0$ and thus $f(w)$ is convex.

2.1.3 Regularized regression (arbitrary norms)

L1 norms are convex and the summation of two convex functions is also convex. $Xw - y$ is a linear function, and the composition of a convex function and a linear function is convex. Thus, the whole function is convex.

2.1.4 Logistic regression

The sum of convex functions is convex.

The composition of a complex function and a linear function is convex.

Since $-y_i w^T x_i$ is linear with respect to w, we just have to show that $\log(1 + \exp(g(x)))$ is convex.

$$\frac{d}{dx}\log(1 + \exp(g(x))) = \frac{\exp(g(x))g'(x)}{1 + \exp(g(x))}$$
$$\frac{d^2}{dx^2}\log(1 + \exp(g(x))) = \frac{\exp(g(x))(\exp(g'(x))g''(x) + g''(x) + g'(x)^2)}{(1 + \exp(g(x)))^2}$$

In this case, $g(x)$ is linear and thus $g''(x) = 0$. This simplifies the above equation to

$$\frac{d^2}{dx^2}\log(1 + \exp(g(x))) = \frac{\exp(g(x))g'(x)^2}{(1 + \exp(g(x)))^2}$$

This is clearly positive since both exponentiation and squaring result in positive number. Thus, when $g(x)$ is linear, $log(1+\exp(g(x)))$ is convex. Therefore, $f(w)$ is convex.

2.1.5 Support vector regression

L2-squared norms are convex. Summation of convex functions are convex. Max of convex functions are convex. 0 is linear and thus convex. $w^T x_i - y_i$ is linear since it varies linearly with respect to w. Norm of a linear function is also convex, thus, $|w^T x_i - y_i|$ is convex. Therefore, $f(w)$ is convex.

2.2 MAP Estimation

2.2.1 Laplace prior

$$p(w_j) = \frac{\lambda}{2}\exp(-\lambda|w_j|)$$

Log-prior

$$\log(p(w_j)) = -\lambda|w_j| + \log(\frac{\lambda}{2})$$

The negative log-prior is

$$-\log(p(w)) = \lambda\sum_{j=1}^{d}(|w_j| - \log(\frac{\lambda}{2})) = \lambda\|w\|_1 + \text{const}$$

Thus, using this prior instead of the normal one would change it to using a L1-regularizer.

2.2.2 Laplace likelihood

$$p(y_i|x_i, w) = \frac{1}{2}\exp(-|w^T x_i - y_i|)$$

Applying negative log-likelihood.

$$f(x) = -\sum_{i=1}^{n} \log(\frac{1}{2}\exp(-|w^T x_i - y_i|))$$

$$f(x) = -\sum_{i=1}^{n} \left[\log(\exp(-|w^T x_i - y_i|)) - \log(2)\right]$$

$$f(x) = \sum_{i=1}^{n} \left[|w^T x_i - y_i| + \log(2)\right]$$

$$f(x) = \sum_{i=1}^{n} \left[|w^T x_i - y_i|\right] + \log(2)n$$

$$f(x) = \|Xw - y\|_1 + \log(2)n$$

This assumption would make the equation use the L1 norm for loss instead of least squares.

2.2.3 Gaussian likelihood where variance is σ^2

$$p(y_i|x_i, w) = \frac{1}{\sqrt{2\sigma^2\pi}}\exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma^2}\right)$$

Apply NLL.

$$f(w) = -\sum_{i=1}^{n} \log(\frac{1}{\sqrt{2\sigma^2\pi}}\exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma^2}\right))$$

$$f(w) = -\sum_{i=1}^{n} \left(-\frac{(w^T x_i - y_i)^2}{2\sigma^2} - \log\sqrt{2\sigma^2\pi}\right)$$

$$f(w) = \frac{1}{2\sigma^2}\sum_{i=1}^{n}(w^T x_i - y_i)^2 + n\log\sqrt{2\sigma^2\pi}$$

$$f(w) = \frac{1}{2\sigma^2}\|Xw - y\|^2 + n\log\sqrt{2\sigma^2\pi}$$

This is the same as least squares, but there is a different variance.

2.2.4 Gaussian likelihood where variance is σ_i^2

$$p(y_i|x_i, w) = \frac{1}{\sqrt{2\sigma_i^2\pi}}\exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma_i^2}\right)$$

Reusing some work from 2.2.3.

$$f(w) = \sum_{i=1}^{n}\left(\frac{(w^T x_i - y_i)^2}{2\sigma_i^2} + \log\sqrt{2\sigma_i^2\pi}\right)$$

Since variance is always positive, we can merge the two squared terms.

$$f(w) = \sum_{i=1}^{n}\left(\frac{1}{2}(\frac{w^T x_i - y_i}{\sigma_i})^2 + \log(\sqrt{2\pi}\sigma_i)\right)$$

We can simplify further by defining V as a diagonal matrix with $\frac{1}{\sigma_i}$ along the diagonal.

$$f(w) = \frac{1}{2}\|V(Xw - y)\|^2 + \sum_{i=1}^{n} \log(\sqrt{2\pi}\sigma_i)$$

Since there is a specific variance on each training example, this gives us weighted least squares.

2.2.5 student t likelihood

$$p(y_i|x_i, w) = \frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi}\Gamma(\frac{v}{2})}\left(1 + \frac{(w^T x_i - y_i)^2}{v}\right)^{-\frac{v+1}{2}}$$

$$f(w) = -\sum_{i=1}^{n}\left[\log\left(\frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi}\Gamma(\frac{v}{2})}\right) - \frac{v+1}{2}\log\left(1 + \frac{(w^T x_i - y_i)^2}{v}\right)\right]$$

This gives us the sum of the log of least squares. This is likely very robust as it takes the log of the square of the loss. Since this loss increases more slowly as the numbers get larger, very large numbers (typically outliers) will have less effect on the overall model leading to a more robust model with low test error.

3 Multi-Class Logistic

3.1 One-vs-all Logistic Regression

```
function [model] = logLinearClassifier(X,y)
% Classification using one-vs-all with logistic loss.

% Compute sizes
[n,d] = size(X);
k = max(y);

W = zeros(d,k); % Each column is a classifier
for c = 1:k
    yc = ones(n,1); % Treat class 'c' as (+1)
    yc(y ~= c) = -1; % Treat other classes as (-1)
    % W(:,c) = (X'*X)\(X'*yc);

    maxFunEvals = 400; % Maximum number of evaluations of objective
    verbose = 1; % Whether or not to display progress of algorithm
    w0 = zeros(d,1);
    W(:,c) = findMin(@logisticLoss,w0,maxFunEvals,verbose,X,yc);
end

model.W = W;
model.predict = @predict;
end

function [yhat] = predict(model,X)
    W = model.W;
    [~,yhat] = max(X*W,[],2);
end

function [f,g] = logisticLoss(w,X,y)
yXw = y.*(X*w);
f = sum(log(1 + exp(-yXw))); % Function value
g = -X'*(y./(1+exp(yXw))); % Gradient
end
```

```
errors =  0.070000
```

3.2 Softmax Classification

$$max_{y \in 1,2,3} p(y|W, \hat{x})$$

$$\sum_{c=1}^{k} \exp(w_c^T x_i)$$

is constant for all y_i, thus don't need to compute it. Just take the max of all the results.

$$p(1|W, \hat{w}) = \exp(\begin{bmatrix} +2 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}) = \exp(2-1) = \exp(1)$$

$$p(2|W, \hat{w}) = \exp(\begin{bmatrix} +2 & +2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}) = \exp(2+2) = \exp(4)$$

$$p(3|W, \hat{w}) = \exp(\begin{bmatrix} +3 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}) = \exp(3-1) = \exp(2)$$

Since $p(2|W, \hat{w})$ is the highest, the class label would be 2.

3.3 Softmax Loss

$$f(W) = \sum_{i=1}^{n} \left[-w_{y_i}^T x_i + \log \left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \right) \right]$$

$$\frac{d}{dW_{jc}} f(W) = \sum_{i=1}^{n} \left[\frac{d}{dW_{jc}} (-w_{y_i}^T x_i) + \frac{d}{dW_{jc}} (\log \left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \right)) \right]$$

$$\frac{d}{dW_{jc}} (\log \left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \right)) = \frac{1}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)} * \frac{d}{dW_{jc}} \sum_{c'=1}^{k} \exp(w_{c'}^T x_i)$$

$$\frac{d}{dW_{jc}} (\log \left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \right)) = \frac{1}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)} * \exp(w_c^T x_i) x_{ij}$$

$$\frac{d}{dW_{jc}} (-w_{y_i}^T x_i) = I(y_i = c) * (-x_{ij})$$

$$\frac{d}{dW_{jc}} f(W) = \sum_{i=1}^{n} \left[I(y_i = c) * (-x_{ij}) + \frac{1}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)} * \exp(w_c^T x_i) x_{ij} \right]$$

3.4 Softmax Classifier

```
function [model] = softmaxClassifier(X,y)

% Compute sizes
[n,d] = size(X);
k = max(y);

W = zeros(d*k,1); % Each column is a classifier

[f,g] = softmaxLoss(W,X,y,d,k);
[f2,g2] = autoGrad(W,@softmaxLoss,X,y,d,k);

if max(abs(g-g2) > 1e-4)
    fprintf('User and numerical derivatives differ:\n');
```

```
    [g g2]
else
    fprintf('User and numerical derivatives agree.\n');
end


maxFunEvals = 1000; % Maximum number of evaluations of objective
verbose = 1; % Whether or not to display progress of algorithm


W = findMin(@softmaxLoss,W,maxFunEvals,verbose,X,y,d,k)
W = autoGrad(w,@softmaxLoss,X,y,d,k);

model.W = reshape(W, d, k);
model.predict = @predict;
end

function [yhat] = predict(model,X)
    W = model.W;
    [~,yhat] = max(X*W,[],2);
end

function [f,g] = softmaxLoss(w,X,y,d,k)
[n,d] = size(X);

w = reshape(w, d, k);

f = 0; % value
for i = 1:n
  smt = sum(exp(w' * X(i,:)'));
  f += -w(:,y(i))'*X(i,:)' + log(smt);
end
f;

g = zeros(d, k); % gradient
for j = 1:d
    for c = 1:k
        tempg = 0;
        for i = 1:n
            if y(i) == c
                tempg += -X(i, j)';
            end
            smt = sum(exp(w'*X(i,:)'));
            tempg += exp(w(:,c)'*X(i,:)')*X(i,j)/smt;
        end
        g(j,c) = tempg;
    end
end
g = reshape(g, d*k, 1);
end
```