CPSC 340 Assignment 4 (due November 14)

Linear Models Part 2

1 Logistic Regression with Sparse Regularization

If you run the function *example_logistic*, it will:

1. Load a binary classification dataset containing a training and a validation set.
2. 'Standardize' the columns of X and add a bias variable.
3. Apply the same transformation to $Xvalidate$.
4. Fit a logistic regression model.
5. Report the number of features selected by the model (number of non-zero regression weights).
6. Report the error on the validation set.

Logistic regression does ok on this dataset, but it uses all the features (even though only the prime-numbered features are relevant) and the validation error is above the minimum achievable for this model (which is 1 percent, if you have enough data and now which features are relevant). In this question, you will modify this demo to use different forms of regularization to improve on these aspects.

1.1 L2-Regularization

Make a new function, *logRegL2*, that takes an input parameter λ and fits a logistic regression model with L2-regularization. Specifically, while *logReg* computes w by minimizing

$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)),$$

your new function *logRegL2* should compute w by minimizing

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

Hand in your updated code. Using this new code, report the number of non-zeroes and the validation error with $\lambda = 1$.

Answer:
The code should look like this:

```
function [model] = logReg(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(@logisticLoss,w0,maxFunEvals,verbose,X,y,lambda);
model.predict = @(model,X)sign(X*model.w); % Predictions by taking sign
end

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

With L2-regularization, the validation error decreases to 0.074 but the number of non-zeroes stays at 101.

1.2 L1-Regularization

Make a new function, *logRegL1*, that takes an input parameter λ and fits a logistic regression model with L1-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda\|w\|_1.$$

Hand in your updated code. Using this new code, report the number of non-zeroes and the validation error with $\lambda = 1$.

Hint: the function *findMinL1* implements a proximal-gradient method to minimize the sum of a differentiable function g and $\lambda\|w\|_1$,

$$f(w) = g(w) + \lambda\|w\|_1.$$

This function has a similar interface to *findMin*, except that you (a) only provide the code to compute the function/gradient of the differentiable part g and (b) need to provide the value λ.

Answer:
The code should look like this:

```
function [model] = logReg(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
```

With L1-regularization, the validation error decreases to 0.052 and the number of non-zeroes decreases to 71.

1.3 L0-Regularization

The function *logRegL0* contains part of the code needed to implement the *forward selection* algorithm, which approximates the solution with L0-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda \|w\|_0.$$

The 'for' loop in this function is missing the part where we fit the model using the subset *ind_new*, then compute the score and updates the *minScore/minInd*. Modify the 'for' loop in this code so that it fits the model using only the features *ind_new*, computes the score above using these features, and updates the *minScore/minInd* variables. Hand in your updated code. Using this new code, report the number of non-zeroes and the validation error with $\lambda = 1$.

Note that the code differs a bit from what we discussed in class, since we assume that the first feature is the bias variable and assume that the bias variable is always included. Also, note that for this particular case using the L0-norm with λ is equivalent to what is known as the Bayesian information criterion (BIC) for variable selection.

Answer:
The for loop should now contain something look like this:

```
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
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

        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
                minScore = score;
                minInd = ind_new;
            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
```

With L0-regularization, the validation error decreases to 0.018 and the number of non-zeroes decreases to 24.

2 Convex Functions and MLE/MAP Loss Functions

This question gets you to explore two important concepts related to loss functions: the *convexity* of loss functions (since convex loss functions can be minimized with gradient descent) and the *probabilistic interpretation* of loss functions (since this allows us to define new loss functions when we encounter weird new situations.

2.1 Showing Convexity from Definitions

Show that the following functions are convex:

1. Quadratic	$f(w) = aw^2 + bw$	$w \in \mathbb{R}, a > 0$
2. Negative logarithm	$f(w) = -\log(aw)$	$w > 0$
3. Regularized regression (arbitrary norms)	$f(w) = \|Xw - y\|_p + \lambda\|w\|_q$	$p \geq 1, q \geq 1, \lambda \geq 0$
4. Logistic regression	$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$	$w \in \mathbb{R}^d$
5. Support vector regression	$f(w) = \sum_{i=1}^{N} \max\{0, \|w^T x_i - y_i\| - \epsilon\} + \frac{\lambda}{2}\|w\|_2^2$	$\lambda \geq 0$

Hint: for the first two you can use the second-derivative test. For the last 3 you'll have to use some of the results in class regarding how combining convex functions can yield convex functions (see Lecture 17).

Answer:

1.

$$f''(x) = a,$$

which implies convexity because $a > 0$.

2.

$$f'(x) = -\frac{1}{x},$$
$$f''(x) = \frac{1}{x^2},$$

which implies convexity because $x > 0$.

3. We have that $\|w\|_q$ is a convex function because it is a norm, and $\lambda \geq 0$ so $\lambda\|w\|_q$ is convex. The function $\|Xw - y\|_p$ is convex because we are composing the convex $\|\cdot\|_p$ with an affine function $Xw - y$. Finally, $f(w)$ is the sum of two convex functions so it is convex.

4. The function $-y_i w^T x_i$ is linear, so we just just need to show that the log-sigmoid function $g(z) = \log(1 + \exp(z))$ is convex to show that each term is convex. It will then follow because the sum of convex functions is convex. To show that the log-sigmoid functions is convex, note that

$$g'(z) = \frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)},$$

and

$$g''(z) = -\frac{1}{(1 + \exp(-z))^2}\frac{d}{dz}\exp(-z) = \frac{\exp(-z)}{(1 + \exp(-z))^2)} = \frac{1}{1 + \exp(-z)}\frac{exp(-z)}{1 + \exp(-z)} = h(-z)h(z),$$

where h is the sigmoid function. Since the sigmoid function is always positive, we've shown that $g(z)$ is convex.

5. First, we note that $|w^T x_i - y|$ is a norm composed with an affine function so it is convex. We then have that $\max\{0, |w^T x_i - y_i| - \epsilon\}$ is a max of convex functions so it is convex. We have that $\|w\|^2$ is convex since squared norms are convex. Finally, since $\lambda > 0$ we have that f is a sum of convex terms so it is convex.

2.2 MAP Estimation

In class, we considered MAP estimation in a regression model where we assumed that:

- The likelihood $p(y_i|x_i, w)$ is a normal distribution with a mean of $w^T x_i$ and a variance of 1.
- The prior for each variable j, $p(w_j)$, is a normal distribution with a mean of zero and a variance of λ^{-1}.

Under these assumptions, we showed that this leads to the standard L2-regularized least squares objective function:

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

For each of the alternate assumptions below, show how the loss function would change (simplifying as much as possible):

1. We use a zero-mean Laplace prior for each variable with a scale parameter of λ^{-1}, so that

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

2. We use a Laplace likelihood with a mean of $w^T x_i$ and a scale of 1, so that

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

3. We use a Gaussian likelihood where each datapoint where the variance is σ^2 instead of 1,

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

4. We use a Gaussian likelihood where each datapoint has its own variance σ_i^2,

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

5. We use a (very robust) student t likelihood with a mean of $w^T x_i$ and a degree of freedom of ν,

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

   where Γ is the "gamma" function (which is always non-negative).

Why is loss coming from the student t distribution "very robust"?

Answer:

1. This changes the regularizer to $\lambda\|w\|_1$.
2. This changes the data-fitting term to $\|Xw - y\|_1$.
3. This changes the data-fitting term to $\frac{1}{2\sigma^2}\|Xw - y\|^2$.
4. This changes the data-fitting term to $\frac{1}{2}(Xw - y)^T Z(Xw - y)$ where the diagonal matrix Z has $1/\sigma_i^2$ along its diagonals.
5. This changes the data-fitting term of $\frac{\nu+1}{2}\sum_{i=1}^n \log\left(1 + \frac{(w^T x_i - y_i)^2}{\nu}\right)$.

The student t loss is "very robust" because we only depend on the logarithm of the (squared) residual, as opposed to the squared error where we depend on the square and absolute error where we depend on the absolute value.

3 Multi-Class Logistic

The function *example_multiClass* loads a multi-class classification dataset and fits a 'one-vs-all' classification model using least squares, then reports the validation error and shows a plot of the data/classifier. The performance on the validation set is ok, but could be much better. For example, this classifier never even predicts that examples will be in classes 1 or 5.

3.1 One-vs-all Logistic Regression

Using the squared error on this problem hurts performance because it has 'bad errors' (the model gets penalized if it classifies examples 'too correctly'). Write a new function, *logLinearClassifier*, that replaces the squared loss in the one-vs-all model with the logistic loss. Hand in the code and report the validation error.

Answer:
The loop should should changed to something like this:

```
function [model] = logLinearClassifier(X,y)
% Regression using one-vs-all log-linear model

% 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) = findMin(@logisticLoss,W(:,c),500,1,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
```

The above uses *findMin* and *logisticLoss* as in previous assignments. This decreases the error from around 0.13 down to around 0.07.

3.2 Softmax Classification

Using a one-vs-all classifier hurts performance because the classifiers are fit independently, so there is no attempt to calibrate the columns of the matrix W. An alternative to this independent model is to use the softmax probability,

$$p(y_i|W, x_i) = \frac{\exp(w_{y_i}^T x_i)}{\sum_{c=1}^{k} \exp(w_c^T x_i)}.$$

Here c is a possible label and $w_{c'}$ is column c' of W. Similarly, y_i is the training label, w_{y_i} is column y_i of W, and in this setting we are assuming a discrete label $y_i \in \{1, 2, 3\}$. Before we move on to implementing the softmax classifier, let's do a simple example:

Consider the dataset below, which has 10 training examples and 2 features:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 3 \\ 3 \\ 3 \\ 3 \end{bmatrix}.$$

Suppose that you want to classify the following test example:

$$\hat{x} = \begin{bmatrix} 1 & 1 \end{bmatrix}.$$

Suppose we fit a multi-class linear classifier using the softmax loss, and we obtain the following weight matrix:

$$W = \begin{bmatrix} +2 & +2 & +3 \\ -1 & +2 & -1 \end{bmatrix}$$

Under this model, what class label would we assign to the test example? (Show your work.)

Answer:
This model bases its decision of maximizing the inner-product, $w_c^T \hat{x}$. For class 1 we have

$$w_1^T \hat{x} = (+2)1 + (-1)1 = 1.$$

For class 2 we have

$$w_2^T \hat{x} = (+2)1 + (+2)1 = 4.$$

For class 3 we have

$$w_3^T \hat{x} = (+3)1 + (-1)1 = 2.$$

So this model would also predict '2'.

3.3 Softmax Loss

The loss function corresponding to the negative logarithm of the softmax probability is given by

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

Derive the derivative of this loss function with respect to a particular element W_{jc}. Try to simplify the derivative as much as possible (but you can express the result in summation notation).

Hint: for the gradient you can use x_{ij} to refer to element j of example i. You can use an 'indicator' function, $I(y_i = c)$, which is 1 when $y_i = c$ and is 0 otherwise. Note that you can use the definition of the softmax probability to simplify the derivative.

Answer:
The loss is the negative logarithm of the probability,

$$-\log(p(y_i|W, x_i) = -w_{y_i}^T x_i + \log \left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \right).$$

The derivative with respect to a particular W_{ij} is given by

$$\frac{d}{dW_{jc}}[-\log(p(y_i|W,x_i)] = -I(y_i = c)(x_i)_j + \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k}\exp(w_{c'}^T x_i)}(x_i)_j.$$

This can be simplified even further by factoring out $(x_i)_j$ and noticing that the softmax probability appears in the second term,

$$\frac{d}{dW_{jc}}[-\log(p(y_i|W,x_i)] = (x_i)_j[p(y_i|W,x_i) - I(y_i = c)].$$

3.4 Softmax Classifier

Make a new function, *softmaxClassifier*, which fits W using the softmax loss from the previous section instead of fitting k independent classifiers. Hand in the code and report the validation error.

HInt: you may want to use the *autoGrad* function from A3 to check that your gradient code is correct.

Answer:
The code should look something like this:

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

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

W = zeros(d,k); % Each column is a classifier
W(:) = findMin(@softmaxLoss,W(:),500,1,X,y,k);

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

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

function [nll,g,H] = softmaxLoss(w,X,y,k)

[n,p] = size(X);
W = reshape(w,[p k]);

XW = X*W;
Z = sum(exp(XW),2);

ind = sub2ind([n k],[1:n]',y);
nll = -sum(XW(ind)-log(Z));

g = zeros(p,k);
for c = 1:k
    g(:,c) = X'*(exp(XW(:,c))./Z-(y == c));
end
```

```
g = reshape(g,[p*k 1]);
end
```

(Longer versions that use a 'for' loop instead of *sub2ind* are ok.) The validation error depends on how precisely you solve the optimization problem, but it decrease to something very small like 0.01 or 0.02.