CPSC 340 Assignment 4 (due Sunday March 19th at 11:59pm)

Linear Models Part 2

1 Logistic Regression with Sparse Regularization

1.1 L2-Regularization

The updated code can be found at `https://github.ubc.ca/cpsc340/jeanlam_nafis1_hw4/blob/master/code/linear_model.py`. The new training error is 0.002, with validation error 0.074, and 101 nonZeros.

1.2 L1-Regularization

The updated code can be found at `https://github.ubc.ca/cpsc340/jeanlam_nafis1_hw4/blob/master/code/linear_model.py`. The new training error is 0.000, with validation error 0.052, and 71 nonZeros.

1.3 L0-Regularization

The updated code can be found at `https://github.ubc.ca/cpsc340/jeanlam_nafis1_hw4/blob/master/code/linear_model.py`. The new training error is 0.000, with validation error 0.022, and 29 nonZeros.

1.4 Discussion

Rubric: {reasoning:2}

In a short paragraph, briefly discuss your results from the above. How do the different forms of regularization compare with each other? Can you provide some intuition for your results? No need to write a long essay, please!

L2-Regularization produces the most non-zero values as L2 will never produce $w_j = 0$, but rather will be close to 0, and never eliminating the feature from the set if it were to be irrelevant.

L1 regularization yields less non-zero values due to the minimum often found at the discountinuity at 0. The value of lambda, 1, is large enough such that the minimum of the parabola is past the origin.

L0 regularization, using the greedy forward selection approach, produces the least number of non-zero values.

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

Rubric: {reasoning:5}

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).

1.$f(w) = aw^2 + bw => \frac{df}{dw} = 2aw + b => \frac{d^2f}{dw^2} = a$. Since it is given that a is greater than 0 therefore its is convex .

2.$f(w) = -log(aw) => \frac{df}{dx} = -1/x => \frac{d^2f}{dx^2} = 1/x^2$. Since it x is greater than 0 therefore it is convex.

3.The summation of two convex function is a convex function.Here we have $\|Xw - y\|_p$, we are composing a convex of $_p$ and a linear function $Xw - y$.Therefore, $\|Xw - y\|_p$ is a convex function and $\|w\|_q$ is a norm therefore it is a convex. Also, lambda is greater than 0 therefore $\lambda\|w\|_q$ is convex.All of the elements of the function is a convex and summation of convex is a convex.

4.$f(w) = \sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i))$ The sum of convex function is a convex.Here,we have to show that the log sigmoid is convex.We know that $(-y_i w^T x_i)$ is linear.1 $+(-y_i w^T x_i)$ is convex since it is linear. Therefore we have to show log(1+e$((-y_i w^T x_i)$ is convex.

Using differentiation:Treating $(-y_i w^T x_i)$ as a constant x we have: 1/1+e(-x). Differentiating again we have: $\frac{e(-x)}{(1+e(-x))^2} = \frac{1}{1+e(-x)}\frac{e(-x)}{1+e(-x)} = sigmoid(-x)sigmoid(x)$, This is the sigmoid function and it cannot be negative.Therefore, f(w) is convex.

5.$|w^T x_i - y_i| - \epsilon\}$ this is convex , since $|w^T x_i - y_i|$ is a convex and is a linear function, $\sum_{i=1}^{N} \max\{0, |w^T x_i - y_i|$ is convex because maximum of a convex is convex.Lambda is greater than 0(given) therefore f(w) is convex as sum of convex is a convex.

2.2 MAP Estimation

Rubric: {reasoning:5}

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"?

Changes in the loss function:

1.Regularizer is changed to $-\lambda\|w\|_j$

2.Model fitting changes to $\|Xw - y\|_1$

3.Model fitting changes to $\frac{1}{2\sigma^2}\|Xw - y\|^2$

4.Model fitting changes to $\frac{1}{2}(Xw - y)^T Z(Xw - y)$ and Z is a diagonal matrix with variance squared in the diagonals.

3.Model fitting changes to $\frac{1}{2\sigma^2}\|Xw - y\|^2$

5.Model fitting changes to $\frac{v+1}{2} sum_{i=1}^{N} \log(1 + \frac{(w^T x_i - y_i)^2}{v})$

The loss is "very robust" because it is dependant on log of $(1 + \frac{(w^T x_i - y_i)^2}{v})$ instead of squared or absolute error.

3 Multi-Class Logistic

If you run `python main.py -q 3` the code loads a multi-class classification dataset with $y_i \in \{0, 1, 2, 3, 4\}$ 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

The updated code can be found at `https://github.ubc.ca/cpsc340/jeanlam_nafis1_hw4/blob/master/code/linear_model.py`.

logLinearClassifier Training error 0.084
logLinearClassifier Validation error 0.070

3.2 Softmax Classification

Rubric: {reasoning:2}

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.)

The class label to be chosen is the index of the maximum of $w_{11}x_1 + w_{12}x_2, w_{12}x_1 + w_{22}x_2, w_{13}x_1 + w_{23}x_2$.
$w_{11}x_1 + w_{12}x_2 = (2)(1) + (-1)(1) = 1$
$w_{12}x_1 + w_{22}x_2 = (2)(1) + (2)(1) = 4$
$w_{13}x_1 + w_{23}x_2 = (3)(1) + (-1)(1) = 2$
Thus, the largest one features column 2, reflecting that the test sample [1 1] is of class 2.

3.3 Softmax Loss

Rubric: {reasoning:3}

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.

Negative log of probability = loss

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

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

We can further factor out the $(x_i)_j$.

3.4 Softmax Classifier

Rubric: {code:3}

Make a new class, *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 `utils.check_gradient` to check that your gradient code is correct.

The updated code can be found at `https://github.ubc.ca/cpsc340/jeanlam_nafis1_hw4/blob/master/code/linear_model.py`