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

Linear Models Part 2

Instructions

Rubric: {mechanics:3}

The above points are allocated for following the general homework instructions on GitHub.

As usual, if you're using Python 2:

- Add `from __future__ import division` to the top of each Python file.
- Grab the Python 2 compatible data files from the "home" repo on GitHub.

1 Logistic Regression with Sparse Regularization

If you run `python main.py -q 1`, 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 (in *utils.load_dataset*).
3. Apply the same transformation to $Xvalidate$ (in *utils.load_dataset*).
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 know which features are relevant). In this question, you will modify this demo to use different forms of regularization to improve on these aspects.

Note: your results may vary a bit depending on versions of Python and its libraries.

1.1 L2-Regularization

Rubric: {code:2}

Make a new class, *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 $\lambda = 1$.

Note: as you may have noticed, `lambda` is a special keyword in Python and therefore we can't use it as a variable name. As an alternative I humbly suggest `lammy`, which is what my neice calls her stuffed animal toy lamb. However, you are free to deviate from this suggestion.

logRegL2 Validation error 0.074 nonZeros: 101

1.2 L1-Regularization

Rubric: {code:3}

Make a new class, *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 $\lambda = 1$.

You should use the function *minimizers.findMinL1*, which 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.

logRegL1 Validation error 0.052 nonZeros: 71

1.3 L0-Regularization

Rubric: {code:4}

The class *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 *selected_new*, then compute the score and updates the *minLoss/bestFeature*. Modify the `for` loop in this code so that it fits the model using only the features *selected_new*, computes the score above using these features, and updates the *minLoss/bestFeature* 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.

Validation error 0.026 nonZeros: 24

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!

According to my results, L0 performs better than L1, which performs better than L2, as L0 regularization seems to cover only the more important features which gives a lower error than fitting on a higher number of less important features, as seen with L1 and L2 regularization.

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. $f(w) = aw^2 + bw$
   $\nabla f(w) = 2aw + b$
   $\nabla^2 f(w) = 2a$, since $a > 0$, $\nabla^2 f(w) > 0$ hence, the function is convex.
2. $f(w) = -\log(aw)$
   $\nabla f(w) = -1/w$
   $\nabla^2 f(w) = 1/w^2$, since $w > 0$, $\nabla^2 f(w) > 0$ hence, the function is convex.
3. $f(w) = \|Xw - y\|_p + \lambda\|w\|_q$
   We know norms (p, q) are convex, and a non-negative constant (λ) multiplied by a convex function is also a convex function
   Hence, by adding two convex functions again we get a convex function.

4. $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$
We know sum of convex functions is convex, so we need to show $\log(1 + \exp(-y_i w^T x_i))$ is convex.
Let $g(w) = \log(1 + \exp(-y_i w^T x_i))$,

$$g'(w) = \frac{-y_i x_i (\exp(-y_i w^T x_i))}{1 + \exp(-y_i w^T x_i)}$$

Then,

$$g''(w) = \frac{((y_i x_i)^T (y_i x_i)(\exp(-y_i w^T x_i)))(1 + \exp(-y_i w^T x_i)) - (y_i x_i)^T (y_i x_i)(\exp(-y_i w^T x_i))}{(1 + \exp(-y_i w^T x_i))^2}$$

$$g''(w) = \frac{\exp(-2y_i w^T x_i)}{(1 + exp(-y_i w^T x_i))^2} > 0$$

Thus g is convex, and so is f as it is a sum of convex functions.

5. We know norms multiplied by positive constants are convex, and so are the sum of convex functions. Therefore we just need to show that -max$\{0, |w^T x_i - y_i| - \epsilon$ is convex.
We know $|w^T x_i - y_i|$ is always positive. If $\epsilon > |w^T x_i - y_i|$, 0 will be chosen by the max function, and the initial function will be convex, and if $\epsilon < |w^T x_i - y_i|$, then the max function will be positive and hence convex again.
Since sum of convex functions is convex, f(w) is 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

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

We then have to minimize the negative log-likelihood(NLL):

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

$$-\sum \log(p(w_j)) = -\sum_{i=1}^{n} \frac{\lambda}{2} + \lambda \sum_{i=1}^{n} |w_j| = -\sum \frac{\lambda}{2} + \lambda \|w\|_1$$

The error function then changes to:

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \lambda \|w\|_1$$

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

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

Minimizing the NLL:

$$-\sum_{i=1}^{n} \log(\exp(-|w^T x_i - y_i|)) = \sum_{i=1}^{n} |w^T x_i - y_i| = \|Xw - y\|_1$$

The error function then changes to:

$$f(w) = \|Xw - y\|_1 + \frac{\lambda}{2}\|w\|^2$$

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

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

Minimizing the NLL:

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

The error function then changes to:

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

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

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

Minimizing the NLL:

$$-\sum_{i=1}^{n} \log\exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma_i^2}\right) = \frac{1}{2}\sum_{i=1}^{n}\frac{(w^T x_i - y_i)^2}{\sigma_i^2} = \frac{\|(Xw - y)\sigma^{-1}\|^2}{2}$$

The error function then changes to:

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

Where σ is a n x 1 vector containing all variance values corresponding to each term.

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).
Minimizing the NLL:

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

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

Therefore, the error function then has to minimize - $\sum_{i=1}^{n} \log\left(1 + \frac{(w^T x_i - y_i)^2}{\nu}\right)$

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

The loss from the student t distribution is very robust as the error function is divided by the degree of freedom v, which lets us freely choose how important outliers may or may not be.

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

Rubric: {code:2}

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 class, *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

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.)

Here $x = \hat{x}^T$

$$w_1^T x = \begin{bmatrix} +2 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2 - 1 = 1$$

$$w_2^T x = \begin{bmatrix} +2 & +2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2 + 2 = 4$$

$$w_3^T x = \begin{bmatrix} +3 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 3 - 1 = 2$$

$$p(y_i = 1) = \exp(w_1^T x) = e$$
$$p(y_i = 2) = \exp(w_2^T x) = e^4$$
$$p(y_i = 3) = \exp(w_3^T x) = e^2$$

Since $p(y_i = 2) > p(y_i = 3) > p(y_i = 1)$, under this model 2 would be assigned to the test example.

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.

$$f'(W) = \sum_{j=1}^{d}(-\sum^{i=c} x_{ij} + \sum_{i=1}^{n} \frac{\exp(w_c^T x_i)x_{ij}}{\sum_{j=1}^{k} \exp(w_j^T x_i)})$$

Or, in simpler terms, the negative sum of all X values in the same column with same classifier, plus softmax probability.

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.

Validation error 0.008