CPSC 340 Assignment 4 (due Friday, March 9 at 9:00pm)

Instructions

Rubric: {mechanics:3}

The above points are allocated for following the general homework instructions. In addition to the usual instructions, **we have a NEW REQUIREMENT for this assignment** (and future assignments, unless it's a disaster): if you're embedding your answers in a document that also contains the questions, your answers should be in blue text. This should hopefully make it much easier for the grader to find your answers. To make something blue, you can use the LaTeX macro `\blu{my text}`.

1 Convex Functions

Rubric: {reasoning:5}

Recall that convex loss functions are typically easier to minimize than non-convex functions, so it's important to be able to identify whether a function is convex.

Show that the following functions are convex:

1. $f(w) = \alpha w^2 - \beta w + \gamma$ with $w \in \mathbb{R}, \alpha \geq 0, \beta \in \mathbb{R}, \gamma \in \mathbb{R}$ (1D quadratic).
   $f'(w) = 2\alpha w - \beta$
   $f''(w) = 2\alpha \geq 0$

2. $f(w) = w \log(w)$ with $w > 0$ ("neg-entropy")
   $f'(w) = 1 + \log(w)$
   $f''(w) = \frac{1}{w} \geq 0$

3. $f(w) = \|Xw - y\|^2 + \lambda\|w\|_1$ with $w \in \mathbb{R}^d, \lambda \geq 0$ (L1-regularized least squares). $\|Xw - y\|^2$ is convex function with respect to w because it is a L2 norm and norms are convex.
   $\lambda\|w\|_1$ is also convex since L1 norm is convex and λ is positive. The sum of two convex function is also convex.

4. $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$ with $w \in \mathbb{R}^d$ (logistic regression).
   Sum of convex functions is convex. We just need to show $f(w) = \log(1 + \exp(-y_i w^T x_i))$ is convex. $-y_i w^T x_i$ is linear. It doesn't affect the convexity. Let $z = -y_i w^T x_i$. The problem goes to check the convexity of $g(z) = \log(1 + \exp(z))$.
   $g(z)' = \frac{\exp(z)}{1+\exp(z)} = \frac{1}{1+exp(-z)}$
   $g(z)'' = \frac{\exp(-z)}{(1+\exp(-z))^2} \geq 0$ Since exponential function and quadratic function are alway positive. Thus the original function is convex function.

5. $f(w, w_0) = \sum_{i=1}^{N}[\max\{0, w_0 - w^T x_i\} - w_0] + \frac{\lambda}{2}\|w\|_2^2$ with $w \in R^d, w_0 \in \mathbb{R}, \lambda \geq 0$ ("1-class" SVM).
   $w_0 - w^T x_i$ is convex function. 0 is constant, and it is convex. maximum of two convex function is convex. $w0$ is convex function. summation of convex functions is convex function so the first term is convex. L2 norm is convex. λ is positive, so the second term is convex too. By summing two convex function, the original function is convex function.

2 Logistic Regression with Sparse Regularization

If you run `python main.py -q 2`, 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.

2.1 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 with $\lambda = 1$, report how the following quantities change: the training error, the validation error, the number of features used, and the number of gradient descent iterations.

logRegL2 Training error 0.002
logRegL2 Validation error 0.074
nonZeros: 101
The training error increases, validation error decreases, numbers of features are the same. The number of iterations is 36.
`https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/k6y1b_a4/blob/master/code/linear_model.py`

2.2 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 with $\lambda = 1$, report how the following quantities change: the training error, the validation error, the number of features used, and the number of gradient descent iterations. `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/k6y1b_a4/blob/master/code/linear_model.py`

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 (a) you only pass in the the function/gradient of the differentiable part, g, rather than the whole function f; and (b) you need to provide the value λ. Thus, your `funObj` shouldn't actually contain the L1 regularization, since it's implied in the way you express your objective to the optimizer.

logRegL1 Training error 0.000 It decreses
logRegL1 Validation error 0.052 It increases
nonZeros: 71. It decreses!!
number of iteration is 78. It increses.

2.3 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 with $\lambda = 1$, report the training error, validation error, and number of features selected.
`https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/k6y1b_a4/blob/master/code/linear_model.py`

Training error: 0.000
Validation error: 0.032
nonZeros: 24 = number of features is 24

2.4 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!

L1 regularization can do variable selection. L0 also can do more job of variable selection. L2 regularization cannot do that. In terms of curing overfitting, L2 is not good. But L0 takes a long time for doing nested loops.

2.5 Comparison with scikit-learn

Rubric: {reasoning:1}

Compare your results (training error, validation error, number of nonzero weights) for L2 and L1 regularization with scikit-learn's LogisticRegression. Use the `penalty` parameter to specify the type of regularization. The parameter `C` corresponds to $\frac{1}{\lambda}$, so if you had $\lambda = 1$ then use `C=1` (which happens to be the default anyway). You should set `fit_intercept` to `False` since we've already added the column of ones to X and thus there's no need to explicitly fit an intercept parameter. After you've trained the model, you can access the weights with `model.coef_`.

L1: Training error 0.000
Validation error 0.052
number of nonZeros: 71
It is the same as my previous L1 regularization
L2: Training error 0.002
Validation error 0.074
number of nonZeros: 101
It is the same as my previous L1 regularization
But numbers of features are the same. `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/k6y1b_a4/blob/master/code/linear_model.py`

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 0 or 4.

3.1 Softmax Classification, toy example

Rubric: {reasoning:2}

Linear classifiers make their decisions by finding the class label c maximizing the quantity $w_c^T x_i$, so we want to train the model to make $w_{y_i}^T x_i$ larger than $w_{c'}^T x_i$ for all the classes c' that are not y_i. Here c' is a possible label and $w_{c'}$ is row c' of W. Similarly, y_i is the training label, w_{y_i} is row y_i of W, and in this setting we are assuming a discrete label $y_i \in \{1, 2, \dots, k\}$. Before we move on to implementing the softmax classifier to fix the issues raised in the introduction, let's work through a toy example:

Consider the dataset below, which has $n = 10$ training examples, $d = 2$ features, and $k = 3$ classes:

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

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:

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

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

$W\hat{x}^T = \begin{bmatrix} +2 & -1 \\ +2 & +2 \\ +3 & -1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix}$ We assign the test example to the second label.

3.2 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.

https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/k6y1b_a4/blob/master/code/linear_model.py
logLinearClassifier Training error 0.084
logLinearClassifier Validation error 0.070
https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/k6y1b_a4/blob/master/code/linear_model.py

3.3 Softmax Classifier Implementation

Rubric: {code:5}

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 loss, which is given by

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

The partial derivatives of this function, which make up its gradient, are given by

$$\frac{\partial f}{\partial W_{cj}} = \sum_{i=1}^{n} x_{ij}[p(y_i = c|W, x_i) - I(y_i = c)]\,,$$

where...

- $I(y_i = c)$ is the indicator function (it is 1 when $y_i = c$ and 0 otherwise)
- $p(y_i = c|W, x_i)$ is the predicted probability of example i being class c, defined as

$$p(y_i|W, x_i) = \frac{\exp(w_{y_i}^T x_i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)}$$

(Good news: in previous offerings of CPSC 340, you had to derive this! I think you've probably taken enough derivatives by now though.)

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 `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/k6y1b_a4/blob/master/code/linear_model.py`.

Validation error 0.008

Hint: you may want to use `utils.check_gradient` to check that your implementation of the gradient is correct.

3.4 Comparison with scikit-learn, again

Rubric: {reasoning:1}

Compare your results (training error and validation error for both one-vs-all and softmax) with scikit-learn's `LogisticRegression`, which can also handle multi-class problems. One-vs-all is the default; for softmax, set `multi_class='multinomial'`. For the softmax case, you'll also need to change the solver. You can use `solver='lbfgs'`. Since your comparison code above isn't using regularization, set `C` very large to effectively disable regularization. Again, set `fit_intercept` to `False` for the same reason as above.

Training error 0.084
Validation error 0.070
number of nonZeros: 15
Training error 0.000
Validation error 0.012
number of nonZeros: 15

3.5 Cost of Multinomial Logistic Regression

Rubric: {reasoning:2}

Assuming that we have

- n training examples.
- d features.
- k classes.
- t testing examples.

- T iterations of gradient descent for training.

1. In $O()$ notation, what is the cost of training the softmax classifier?
   O(ndkT).
2. What is the cost of classifying the test examples?
   O(tdk).

4 Very-Short Answer Questions

Rubric: {reasoning:9}

1. Why would you use a score BIC instead of a validation error for feature selection? Using validation error is suspectible to choosing a model due to optimization bias when a large number of models are compared. BIC compares model based penalized likelihood, which does not have this problem.
2. Why do we use forward selection instead of exhaustively search all subsets in search and score methods? Exhaustive search cost is too expensive. Forward selection is less likely to make false positive.
3. In L2-regularization, how does λ relate to the two parts of the fundamental trade-off? λ increase, then it penalize more, so larger training error but lower approximation error.
4. Give one reason why one might chose to use L1 regularization over L2 and give one reason for the reverse case. L1 regularization is likely to give sparse solutions, can help variable selection. L2 regularization has a unique solution and can be analytically computed, Insensitive to changes in data.
5. What is the main problem with using least squares to fit a linear model for binary classification? Even the sign is the same, the model will penalize a lot if the absolute value of a evaluated value is much larger then the true label value. That is, it will penalize a lot for very right values if the value($w^T x_i$) is much greater than 1 or even less then -1.
6. For a linearly separable binary classification problem, how does a linear SVM differ from a classifier found using the perceptron algorithm?
   The SVM classifier returns the maximum-margin classifier. That is, it maximize distance to the training examples. On the other hand, the perceptron method could return any classifier.
7. Which of the following methods produce linear classifiers? (a) binary least squares as in Question 3, (b) the perceptron algorithm, (c) SVMs, and (d) logistic regression. They are all linear classifiers. The performance can be different from each model.
8. What is the difference between multi-label and multi-class classification?
   multi class classification is it has more than two classes for its response value. Multi label classification is the case that the situation is multi class classificaiton and at the same time, each object can have more than 1 label for its label.
9. Fill in the question marks: for one-vs-all multi-class logistic regression, we are solving k optimization problem(s) of dimension d. On the other hand, for softmax logistic regression, we are solving 1 optimization problem(s) of dimension $d * k$.
   k,d,1,d*k

Hints: we're looking for short and concise 1-sentence answers, not long and complicated answers. Also, there is roughly 1 question per lecture.