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).

   Answer: The first derivative is $f'(w) = 2\alpha w - \beta$ and the second derivative is $f''(w) = 2\alpha$, which implies convexity since $\alpha > 0$.

2. $f(w) = w\log(w)$ with $w > 0$ ("neg-entropy")

   Answer: The first derivative is $f'(w) = 1 + \log(w)$ and the second derivative is $f''(w) = 1/w$, which implies convexity since $w > 0$.

3. $f(w) = \|Xw - y\|^2 + \lambda\|w\|_1$ with $w \in \mathbb{R}^d, \lambda \geq 0$ (L1-regularized least squares).

   Answer: We have $\|w\|_1$ is convex because it's a norm, and $\lambda \geq 0$ so $\lambda\|w\|_1$ is convex. The function $\|Xw - y\|^2$ is convex because we're composing the (convex) squared-norm $\|\cdot\|^2$ with an affine function $Xw - y$. Finally $f(w)$ is the sum of these two convex functions so it's 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).

   Answer: 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. $f(w, w_0) = \sum_{i=1}^{N}[\max\{0, w_0 - w^T x_i\} - w_0] + \frac{\lambda}{2}\|w\|_2^2$ with $w \in R^d, w_0 \in \mathbb{R}, \lambda \geq 0$ ("1-class" SVM).

   Answer: First we note that $w_0 - w^T x_i$ is a linear function so it's convex. Since 0 is also convex, and the max of convex functions is convex, the max inside the sum is convex. Since $\lambda > 0$ and squared-norm are convex, $(\lambda/2)\|w\|^2$ is convex. We also have that $-w_0$ is convex because it's linear. For here, f is a sum of functions we've already shown are convex, so f is convex.

General hint: for the first two you can check that the second derivative is non-negative since they are one-dimensional. For the last 3 you'll have to use some of the results regarding how combining convex functions can yield convex functions; these "notes on convexity" are posted on the course homepage as readings for Lecture 10.

Hint for part 4 (logistic regression): this function may seem non-convex since it contains $\log(z)$ and log is concave, but there is a flaw in that reasoning: for example $\log(\exp(z)) = z$ is convex despite containing a log. To show convexity, you can reduce the problem to showing that $\log(1 + \exp(z))$ is convex, which can be done by computing the second derivative. It may simplify matters to note that $\frac{\exp(z)}{1+\exp(z)} = \frac{1}{1+\exp(-z)}$.

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.

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 niece calls her stuffed animal toy lamb. However, you are free to deviate from this suggestion. In fact, as of Python 3 one can now use actual greek letters as variable names, like the λ symbol. But, depending on your text editor, it may be annoying to input this symbol.

Answer: With L2-regularization, the validation error decreases to 0.074 but the number of non-zeroes stays at 101. Gradient descent iterations go down from 121 to 36 (decreased by 85). Training error went from 0 to 0.002.

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.

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.

Answer: With L1-regularization, the validation error decreases to 0.052 and the number of non-zeroes decreases to 71. Solving requires 78 iterations. Training error is 0.

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.

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 $\lambda = 1$ is equivalent to what is known as the Akaike Information Criterion (AIC) for variable selection.

Answer: With L0-regularization, the validation error decreases to 0.038 and the number of non-zeroes decreases to 24. You might have gotten a slightly different answer depending on versions of Python and libraries. We've seen validation errors of 0.018. Training error is 0.

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!

Answer: L2 does not introduce any sparsity (zeros), while L1 introduces some and L0 even more. The sparsity seems to help with overfitting as the validation error goes down for each subsequent method. However, L0-regularization requires nested loops resulting in a much slower algorithm.

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_`.

Answer: We end up with exactly the same results (at least, when printed out to 3 decimal places), which is satisfying.

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 $y_i \in \{1, 2, \ldots, k\}$. 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.)

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'.

3.2 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.

Answer: This decreases the error from around 0.13 down to around 0.07.

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

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

(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.

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

Answer: 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.

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.

Answer: I get exactly the same results for one-vs-all, but a validation error of 0.016 for softmax (instead of 0.008). There's no sense reading too much into these numbers, I think. For example if you change the number of iterations of optimization then you get slightly different results. But, they are quite similar overall which is encouraging.

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?

   Answer: Training the model involves T iterations of gradient descent. Each iteration involves computing the function value and gradient over all n examples. To evaluate the function for one example, the dominant cost is computing $w_{c'}^T x_i$ for all k values of c', each of which costs $O(d)$. Thus evaluating the function for one example costs $O(dk)$, and the gradient has the same cost. Putting everything together gives $O(ndkT)$.

2. What is the cost of classifying the test examples?

   Answer: At test time the largest cost is computing $\hat{X}W$. Taking into account the dimensions of these matrices gives $O(tdk)$. (Technically, we could do some of these operations slightly faster using fast matrix multiplication methods.)

4 Very-Short Answer Questions

Rubric: {reasoning:9}

1. Why would you use BIC as a score instead of a validation error for feature selection?

   Answer: Validation error tends to give many false positives due to optimization bias.

2. Why do we use forward selection instead of exhaustively search all subsets in search and score methods?

   Answer: It's too expensive to try all subsets. Alternately, forward selection is less prone to false positives.

3. In L2-regularization, how does λ relate to the two parts of the fundamental trade-off?

   Answer: As λ increases, the training error goes up but the approximation error usually goes down.

4. Give one reason why one might chose to use L1 regularization over L2 and give one reason for the reverse case.

   Answer: L1 regularization tends to give sparse solutions. L2 regularization has a unique solution and for linear regression we can get the solution in closed form via the normal equations.

5. What is the main problem with using least squares to fit a linear model for binary classification?

   Answer: If $y_i w^T x_i \gg 1$ the squared error will be huge and we would get penalized for being "very correct".

6. For a linearly separable binary classification problem, how does a linear SVM differ from a classifier found using the perceptron algorithm?

   Answer: The SVM classifier returns the maximum-margin classifier (maximize distance to the closest training examples), whereas the perceptron method could return any classifier that makes no errors.

7. Which of the following methods produce linear classifiers? (a) least squares on binary targets, (b) the perceptron algorithm, (c) SVMs, and (d) logistic regression.

   Answer: These are all linear classifiers.

8. What is the difference between multi-label and multi-class classification?

   Answer: In multi-class classification there is one "true" label, whereas in multi-label several of the labels (or none) can be applicable.

9. Fill in the question marks: for one-vs-all multi-class logistic regression, we are solving ?? optimization problem(s) of dimension ??. On the other hand, for softmax logistic regression, we are solving ?? optimization problem(s) of dimension ??.

   Answer: One-vs-all: k problems, each d dimensions; Softmax: 1 problem, kd dimensions. Or, if you want to be careful about it, replace d with $d+1$ in both cases to account for biases.

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