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).
2. $f(w) = w\log(w)$ with $w > 0$ ("neg-entropy")
3. $f(w) = \|Xw - y\|^2 + \lambda\|w\|_1$ with $w \in \mathbb{R}^d, \lambda \geq 0$ (L1-regularized least squares).
4. $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$ with $w \in \mathbb{R}^d$ (logistic regression).
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).

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 $\frac{\exp(z)}{1+\exp(z)} = \frac{1}{1+\exp(-z)}$.

1. $f''(w) = 2\alpha \geq 0$, thus $f(w)$ is convex.
2. $f''(w) = \frac{1}{w}$. For $w > 0$, $f''(w)$ is positive, thus $f(w)$ is convex.
3. Norm equations are convex and the sum of convex equations are convex, thus $f(w)$ is convex.
4. Set $z = -y_i w^T x_i$. z is linear function (also convex) of w.

   So we have $\log(1 + \exp(-y_i w^T x_i)) = \log(1 + exp(z)) = g(z)$.

   $g''(z) = \frac{\exp(-z)}{(1+exp(-z))^2} > 0$, thus $g(z)$ is convex and the sum of $g(z)$, $f(w)$, is convex of w.

5. As $w_0 - w^T x_i$ are linear (convex) to w, their maximum is also convex and the sum of the maximums is convex.
   Also the norm equation is convex. Thus $f(w)$ is convex.

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.
Answer: The result from the q2 code is that the training error is 0.000, the validation error is 0.084 and the non-zero parameter number is 101.

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 neice 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: The code is in /code/linear_model.py.

The corresponding result is as follows:

logRegL2 Training error 0.002
logRegL2 Validation error 0.074
nonZeros: 101

Compared to the q2 result, when applying the L2-regularization, the training error increases to 0.002, the validation error decreases to 0.074.

Non-zero number stays the same as 101.

the iteration of gradient descent decreases from 121 to 36.

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: The code is in /code/linear_model.py.

The corresponding result is as follows:

logRegL1 Training error 0.000
logRegL1 Validation error 0.052
nonZeros: 71

Compared to the Q2 result, when applying the L1-regularization, the training error is the same as 0.000, the validation error decreases to 0.052.

The non-zero number decreases to 71.
The iteration drops from 121 to 78.

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: The code is in /code/linear_model.py.

The corresponding result is as follows:

Training error 0.000
Validation error 0.018
nonZeros: 24

Compared to the q2 result, when applying the L0-regularization, the training error is the same as 0.000.
The validation error further decreases to 0.018.
The non-zero number further decreases to 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!
Answer:
It is clear from the the above results that L2-regularization helps decrease the validation error although leads to a slight increase of training error.
L1- and L0- regularizations maintain the training error as 0 and decreases validation error at the same time.

The validation error results show that $L0 - reg < L1 - reg < L2 - reg$ and the number of non-zero parameters used is also $L0 - reg < L1 - reg < L2 - reg$.
Both L2 and L1 use smaller iterations to find the minimum.

As L0 and L1 are for feature selection and interpretability, so few parameters are used and the model can learn with exponential number of irrelevant features. L2 result is unique and the irrelevant features to learn are of linear number. It can help address over-fitting problem.

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_`.

The code is in /code/main.py.
The answer is as follow:

scikit-L2
Training error 0.002
Validation error 0.074
nonZeros: 101

scikit-L1
Training error 0.000
Validation error 0.052
nonZeros: 71

All the results from the scikit-learn are the same as those I got above.

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.)

For the multi-class linear classifier with softmax loss, we that the maximum values of $w_c^T x_i$ to predict class for x_i.

$w_1^T \hat{x}^T = 2 \times 1 - 1 \times 1 = 1$

$w_2^T \hat{x}^T = 2 \times 1 + 2 \times 1 = 4$

$w_3^T \hat{x}^T = 3 \times 1 - 1 \times 1 = 2$

The $w_2^T \hat{x}^T$ is maximal, hence we assign the test example to class label 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.

See code in /code/linear_model.py.

The answer is as follows:

logLinearClassifier Training error 0.084
logLinearClassifier Validation error 0.070

In my code, the dimensionality of matrix W ($d \times k$) is different from the slides ($k \times d$). This is because if we want to use the 'funObj' function, we need to match the dimensionality.

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.

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

See code in /code/linear_model.py.

The answer is as follows:

Training error 0.000
Validation error 0.008

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.

Set the C = 10000.

scikit-learn-one vs all
Training error 0.084
Validation error 0.070

The errors are just the same in our implementation of One-vs-All.

scikit-learn-Softmax
Training error 0.000
Validation error 0.008

The errors are just the same in our implementation of softmax.

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

In the following, we set k is the number of class labels.

1. Assume in the gradient descent, we have T iterations.
   In each iteration, we have n training examples. The cost is split into two parts: the computation of softmax loss function and the gradient.

   The main cost is the computation of $\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)$, which is $O(kd)$.
   So the total cost is $O(nkdT)$.
2. Suppose the size of test example is t.
   The cost is the computation of $X_{test} \times W$, which is $O(tdk)$.
   (The size of X_{test} is $t \times d$; the size of W is $d \times k$)

4 Very-Short Answer Questions

Rubric: {reasoning:9}

1. Why would you use a score BIC instead of a validation error for feature selection?
2. Why do we use forward selection instead of exhaustively search all subsets in search and score methods?
3. In L2-regularization, how does λ relate to the two parts of the fundamental trade-off?

4. Give one reason why one might chose to use L1 regularization over L2 and give one reason for the reverse case.
5. What is the main problem with using least squares to fit a linear model for binary classification?
6. For a linearly separable binary classification problem, how does a linear SVM differ from a classifier found using the perceptron algorithm?
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.
8. What is the difference between multi-label and multi-class classification?
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 ??.

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

1. Because the feature selection with lowest validation error will have optimization bias. So put a penalty on the model complexity, like BIC, is a more reasonable "score".
2. Since in the "Search and Score", it is hard to search for the best "S". The total number of possible sets is 2^d, which is expensive.

   Although "forward selection" does not guarantee the best set, but reduces many problems. It is cheaper and has fewer false positive.
3. In L2-regularization, the E_{train} goes up and E_{approx} goes down as λ increases.
4. L1-regularization: It can do the regularization and feature selection.
   L2-regularization: It has a closed-form unique solution.
5. The squared error would be very huge when $|w^T x_i| >> 1$. Hence, least squares penalized for being "too right".
6. Perceptron algorithm could have many possible classifiers with zero error, while the SVM outputs the maximum-margin classifier which has the largest distance to closet examples.
7. All of them output linear classifiers.
8. Multi-class classification has many different class labels, while multi-label classification may have more than one "correct" class label in one class.
9. For one-vs-all multi-class logistic regression, we are solving k optimization problems of dimension $n \times d$, since we are finding k binary classifiers.

   For softmax logistic regression, we are solving 1 optimization problem of dimension $n \times d \times k$, since we are minimizing the softmax loss function.