Solutions for CPSC 340 Assignment 4

Submitted by:

Armaan Kaur Bajwa
Student ID: 87921193

Sugun Machipeddy
Student ID: 65753337

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}

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).

   Solution:

   $f'(w) = \frac{df(w)}{dw}$

   $= 2\alpha w - \beta$

   Now, $f''(w) = 2\alpha$

   Given, $\alpha \geq 0$

   So, $f''(w) \geq 0$.

   Hence the given function is convex.

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

   Solution:

   $f'(w) = log(w) + w\frac{1}{w}$

   $= log(w) + 1$

   Now, $f''(w) = \frac{1}{w}$

   Given, $w > 0$

   So, $\frac{1}{w} > 0$

   So the function is convex.

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

   Solution:

   Taking $\|Xw - y\|^2$,

   We know that a squared norm is a convex function. Also, if 'f' is convex, then $f(Xw - y)$ is also convex. Hence, the squared norm of $Xw - y$ is convex.

   i.e., $\|Xw - y\|^2$ is convex.

   Now, taking $\lambda\|w\|_1$,

   $\|w\|_1$ is convex, because norms are convex functions.

   Also, since λ is a constant which is greater than 0, so $\lambda\|w\|_1$ will also be convex.

   Now, $f(w) = \|Xw - y\|^2 + \lambda\|w\|_1$

   i.e. the sum of the above two convex functions. We know that the sum of convex functions is a convex function. So $f(w)$ is 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).

   Solution:

   Replacing $y_i w^T x_i$ by z , we can write:

   $f(z) = \sum_{i=1}^{n} \log(1 + \exp(-z))$

   $f'(z) = \sum_{i=1}^{n} \frac{-\exp(-z)}{1+\exp(-z)}$

   $f''(z) = \sum_{i=1}^{n} \frac{\exp(-z)}{(1+\exp(-z))^2}$

As we can see above, $f''(z) \geq 0$, so the given function 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).

   Solution:

   $\frac{\lambda}{2}\|w\|_2^2$ is convex, because it is a norm multiplied by a constant.

   Now, $w_0 - w^T x_i$ is linear, so it is convex. So $\max\{0, w_0 - w^T x_i\}$ is convex, because max of two convex functions is a convex.
   From the above statements, $\max\{0, w_0 - w^T x_i\} - w_0$ is also convex, because it is a sum of two convex functions.
   Hence, $\sum_{i=1}^{N}[\max\{0, w_0 - w^T x_i\} - w_0]$ is convex because it is summation of n convex functions.
   So, we can say that $f(w, w_0) = \sum_{i=1}^{N}[\max\{0, w_0 - w^T x_i\} - w_0] + \frac{\lambda}{2}\|w\|_2^2$ is convex, again because it is sum of convex functions.

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 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.

Solution:

The code is available in linear_model.py file
Logistic loss without regularization:

logReg Training error 0.000
logReg Validation error 0.084
Number of features: 101
Number of gradient descent iterations : 121

Logistic Loss with L2-Regularization:

logRegL2 Training error 0.002
logRegL2 Validation error 0.074
Number of features: 101
Number of gradient descent iterations : 36

With L2-Regularization the number of gradient descent iterations required to converge decreased from 121 to 36. The training error increased from 0 to 0.002 and there is a decrease in the validation error. There was no feature selection, both of them used all the 101 features.

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.

Solution:

The code is available in linear_model.py file
Logistic loss without L1-Regularization:

logRegL1 Training error 0.000
logRegL1 Validation error 0.048
Number of features selected: 72
Number of gradient descent iterations : 351

In Comparision with No Regression, the training error remains the same i.e 0.0, the validation error decreases and there is better feature selection. The nummber of features selected are 72 in comparision to 101. L1- Regualrization falls behind in terms of the numebr of gradient descent iterations required to converge, it took 351 itearations.

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.

Solution:

The code is available in linear_model.py file
Logistic loss without L0-Regularization:

Training error 0.000
Validation error 0.020
Number of fetures selected: 26
Number of gradient descent iterations : 179

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!

Ans: L2-Regialrization prevents over fittng and that is why it has a non zero training error. Its training error is lower than standard logistic regression and also uses all the features.
L1-Regularization has a training error of zero as a feature is not selected if the error is too large. It also has the lowest validation error among all regularizations. The classifier selected 72 features among the 101 features. It is also slow as we can see from the number of gradient descent iterations
L0-Regularization uses forward selection to select a subset of features. it is faster than L1, but slower than L2-Regularization. the classifier selected 26 features in my case.

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_`.

Solution:

The code is available in linear_model.py file
Scikit learns L2-Regularization

Training error 0.002
Validation error 0.074
Number of fetures selected: 101

our implemetnation of L2-Regularization

logRegL1 Training error 0.002
logRegL1 Validation error 0.074
Number of fetures selected: 101

Scikit learns L1-Regularization

Training error 0.000
Validation error 0.052
Number of fetures selected: 71

our implemetnation of L1-Regularization

logRegL1 Training error 0.000
logRegL1 Validation error 0.048
Number of fetures selected: 72

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.)

Solution:

$\hat{x}.(w1.T) = 1 * 2 - 1 * 1 = 1$
$\hat{x}.(w2.T) = 2 * 1 + 2 * 1 = 4$
$\hat{x}.(w3.T) = 3 * 1 - 1 * 1 = 2$
since the score of $\hat{x}.(w2.T)$ is the highest . the class label to be assigned is '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.

Solution:

One-vs-all Logistic Regularization

The code is available in linear_model.py file
logLinearClassifier Training error 0.084
logLinearClassifier Validation error 0.070

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.

Solution:

Softmax classifier implementation

The code is available in linear_model.py file
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.

Solution:

The code is available in linear_model.py file
Scikit learns one vs all implementation
Training error 0.084
Validation error 0.070

our one vs all implementation
logLinearClassifier Training error 0.084
logLinearClassifier Validation error 0.070

Scikit learns softmax implementation
Training error 0.000
Validation error 0.012

our softmax implementation
Training error 0.000
Validation error 0.008

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 Big-O notation, what is the cost of training the softmax classifier?
   Answer: $O(ndkT)$
2. What is the cost of classifying the test examples?
   Answer: $O(tdk)$

4 Very-Short Answer Questions

Rubric: {reasoning:9}

1. Why would you use a score BIC instead of a validation error for feature selection?
   Answer: Because as 'n' goes to infinity, BIC recovers the true model as it penalizes more for a large number of objects, whereas optimizing too many times for validation error might include optimization bias.

2. Why do we use forward selection instead of exhaustively search all subsets in search and score methods?
   Answer: Because using forward selection is comparatively cheaper, overfits less and has fewer false positives.

3. In L2-regularization, how does λ relate to the two parts of the fundamental trade-off?
   Answer: In L2-regularization, as λ increases, the training error increases and the approximation error decreases.

4. Give one reason why one might chose to use L1 regularization over L2 and give one reason for the reverse case.
   Answer: We should choose L1 regularization over L2 when we need to absolutely know which features are selected during feature selection, because in L1-regularization makes some 'w' equal to zero for the features which are not required.
   We choose L2-regularization over L1 in case of non-unique solution.

5. What is the main problem with using least squares to fit a linear model for binary classification?
   Answer: With binary classification, if you use squared error, you may get penalized for being 'too right'.

6. For a linearly separable binary classification problem, how does a linear SVM differ from a classifier found using the perceptron algorithm?
   Answer: For a classifier found via Perceptron algorithm, all the training examples contribute to the loss and hence to the value of w, whereas, in case of SVM, if something is not a support vector, i.e. $y_i w^T x_i \geq 0$ then its loss is zero and it "doesn't contribute".

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.
   Answer: All of them.

8. What is the difference between multi-label and multi-class classification?
   Answer: In multi-label problems it is possible to have more than one 'correct' class label, whereas in multi-class, it is one vs all, i.e. only one correct class label is allowed.

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: for one-vs-all multi-class logistic regression, we are solving k, the number of class labels optimization problem(s) of dimension $d * 1$. On the other hand, for softmax logistic regression, we are solving $'1'$ optimization problem(s) of dimension $d * k$.