A From-Scratch Gradient Descent Based Logistic Regression Model to Classify Cases of Malignant Breast Cancer

Brandon Salter

Link to Code: https://github.com/brandonsalter/From-Scratch-Logistic-Regression

Abstract

A logistic regression class was constructed in Python in order to further develop an understanding of the method. To learn model parameters, rather than likelihood maximization, a convex loss function was derived, which then allowed for gradient descent optimization to be utilized. Using the breast cancer Wisconsin dataset, five-fold cross validation using grid search was performed in order to find optimal hyperparameter values. Following this, the generalization performance of the model was assessed. Results are provided in the form of a convergence plot, accuracy score, misclassification table, and confusion matrix. The final model had a test accuracy of roughly 98%, having both a false positive and a false negative misclassification. The paper concludes with ideas for continuation.

Introduction

Linear regression methods are typically used to estimate the conditional expectations of continuous variables. However, when the response variable takes on discrete values, these methods are not very effective. In the specific case where the outcome of the response variable is binary, typical logistic regression can be an effective probabilistic classification method. The model is trained on a tabular dataset consisting of labeled observations, where each observation contains one or more feature values. After training the model, unseen observations are then input into the model and predictions are made as to which binary label they most likely belong. Unlike traditional linear regression, in logistic regression, a linear response function is not used directly but is instead input into a nonlinear activation function. To deepen my understanding of logistic regression, I built a model from the ground up to classify the Wisconsin breast cancer dataset.

We will express the net input z into our model as:

$$z = w_1 x_1 + \ldots + w_m x_m + \mathrm{b} = \mathbf{w}^{\mathrm{T}}\mathbf{x} + \mathrm{b},$$

a linear combination of weighted feature values and a bias unit, where m is the total number of observations. This directly corresponds with a multiple linear regression model (i.e. $y = \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x} + \beta_0$).

To introduce logistic regression, let's first define the logit function:

$$logit(p) = \ln\left(\frac{p}{1-p}\right)$$

The logit function takes as input a conditional probability, $p \coloneqq p(y = 1|x)$. We can think of the p as the probability of observing a positive event (a label of 1) given the set of features associated with that observation. The output of this function is the natural log of the odds, which spans values across the entire real-number range.

Assuming a linear relationship between the weighted inputs and the logit function, it follows that $logit(p) = z$.

However, what is useful for the purpose of classification is the probability p. So, we now invert the logit function and solve for p; which results in the sigmoid (activation) function:

$$\sigma(z) = \frac{1}{(1+e^{-z})}$$ (see appendix for derivation)

Notice that $\sigma(z)$ approaches 1 as z → ∞, and 0 as z → $-\infty$ with an intercept $\sigma(0)$ = 0.5. The output of the sigmoid function is then the desired probability of a positive class label. This can be interpreted as a class label prediction of 1 if $\sigma(z) \geq 0.5$, and 0 otherwise.

Since we are dealing with the binary classification of i.i.d. events, the likelihood function for our model is the product of Bernoulli pmfs, with p being our activation function $\sigma(z)$:

$$L(\boldsymbol{w}, b|\boldsymbol{x}) = \prod_{i=1}^{n}\left(\sigma\left(z^{(i)}\right)^{y^{(i)}}\left(1-\sigma\left(z^{(i)}\right)\right)^{1-y^{(i)}}\right)$$

To find the optimal weight coefficients and bias unit for our model, we will obtain a loss function first from the log-likelihood:

$$l(\boldsymbol{w}, b|\boldsymbol{x}) = \sum_{i=1}^{n}\left[y^{(i)}ln\left(\sigma\left(z^{(i)}\right)\right) + \left(1-y^{(i)}\right)ln\left(1-\sigma\left(z^{(i)}\right)\right)\right]$$

Taking the negative of this function and averaging across all examples will allow us to later perform gradient descent in order to parametrize the model, as opposed to likelihood maximization. This results in a convex loss function to minimize:

$$L(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\left[-y^{(i)} ln\left(\sigma\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right) ln\left(1 - \sigma\left(z^{(i)}\right)\right)\right]$$

It can be seen that for each individual training example, the loss function equals $-\ln\left(\sigma(z)\right)$ if y = 1 and $-\ln\left(1 - \sigma(z)\right)$ if y = 0. This results in the loss being zero with a correct label prediction and approaching infinity at an increasingly larger rate as the prediction moves further in the incorrect direction. It can be shown that $\frac{dL^2}{dz^2} \geq 0$ (see appendix for full calculation), which implies that the function is convex and suitable for gradient optimization.

Methodology

I first loaded in the Wisconsin breast cancer dataset, which is available in the sklearn.datasets module. It includes 569 total instances of 30 features with binary class labels. The features include various numeric medical measurements of patients being tested for the disease.

Following this, I then created a logistic regression class, LR_GD, that takes as arguments: eta (η) – the learning rate, n_iter – number of iterations, and random_state – a random seed parameter for reproducibility. The two main methods of the class are fit and predict. The fit method is used to learn model weights. In it, we first randomly generate the weight vector and bias unit to small values, a list is then created to store loss values for plotting convergence later on. Next, a for loop iteratively updates our weights and bias unit once after each pass over the training data. Our gradient descent update rule is defined as follows:

$$\mathbf{w} := \mathbf{w} + \eta * [-\boldsymbol{\nabla}_{\boldsymbol{w}} L(\mathbf{w}, \mathrm{b})],$$

$$\mathrm{b} := \mathrm{b} + \eta * [-\boldsymbol{\nabla}_{b} L(\mathbf{w}, \mathrm{b})], \text{ where:}$$

$$\boldsymbol{\nabla}_{\boldsymbol{w}} L(\mathbf{w}, \mathrm{b}) = \frac{\partial L}{\partial w} = \frac{-(y - \sigma(z)) * x}{n}, \text{ and}$$

$$\boldsymbol{\nabla}_{b} L(\mathbf{w}, \mathrm{b}) = \frac{\partial L}{\partial b} = \frac{-(y - \sigma(z))}{n}$$

(see appendix for derivation).

The gradient descent algorithm in our fit method works as follows:

1. Compute the net input across the entire training dataset
2. Use the net input to compute the sigmoid activation
3. Compute the error (actual – sigmoid)
4. Update the weights and bias unit in accordance with the update rule
5. Compute the loss
6. Append the loss to losses list
7. Repeat process n_iter times

This process should effectively find a minimum value of the loss function. The optimal "step size", determined by our eta parameter, is chosen such that our algorithm will be able to escape shallow local minima and will be found via grid search cross validation.

The predict method in LR_GD, is a simple threshold function that returns a class label of 1 if the sigmoid output is ≥ 0.5 and 0 otherwise. This is used after the model is trained to make class label predictions on the test dataset.

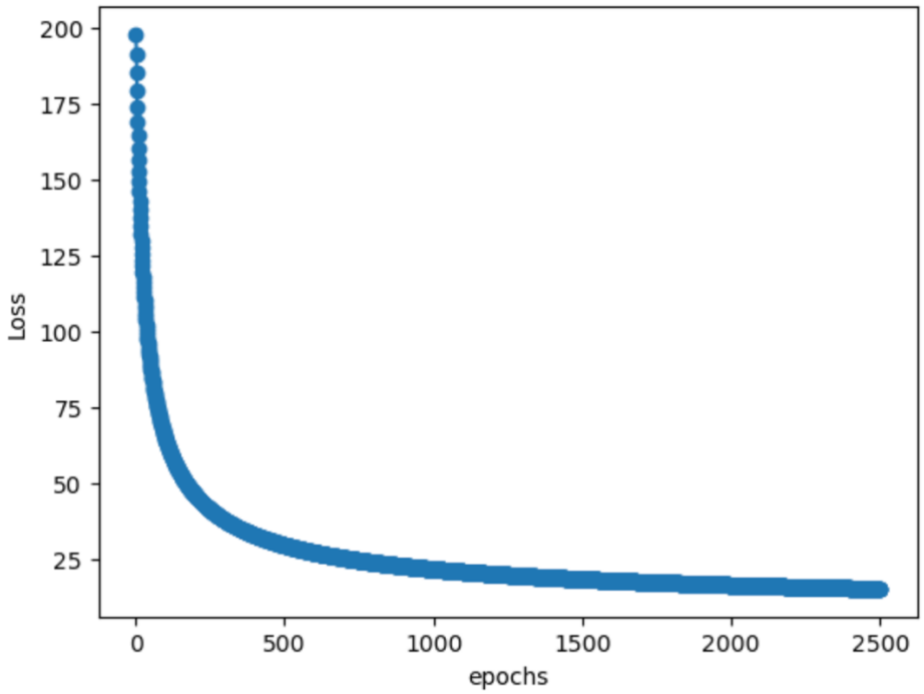
After splitting the data into training and test subsets and standardizing the feature values, grid search five-fold cross validation was performed on an exhaustive list of possible hyperparameters, eta and n_iter, to determine their appropriate values. The cross validation function, five_fold_cv, works as follows:

1. Shuffle the training data
2. Split the training data into 5 roughly equal folds
3. Train on 4 of 5 folds, validate on the remaining fold
4. Compute classification accuracy
5. Repeat process, using different combinations of 4 folds for training each time until all possible combinations have been used
6. Return the average accuracy across the five rounds

The grid_search function simply iterates over a list of ordered pairs corresponding to the eta and n_iter values wished to be evaluated. In each iteration, the five_fold_cv function is carried out with input values associated with the index of the list of parameter tuples. It returns the parameter settings corresponding to the highest cross validation accuracy. The optimal parameters were found to be 0.015 and 2500 for eta and n_iter, respectively. With these settings, a new LR_GD object was then instantiated and a model was fit to the training dataset.

Results

After training the model, its generalization performance on the test set was evaluated and was found have an accuracy of slightly above 98% in predicting malignant cases of breast cancer. This corresponds to correctly classifying 112 out of 114 observations. Convergence of loss was reached after around the 2400th iteration:



The model made both a false-positive and a false-negative misclassification. A table consisting of the misclassified observations, the target and predicted labels, and an accompanying confusion matrix are presented:

mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error
13.80	15.79	90.43	584.1	0.10070	0.1280	0.07789	0.05069	0.1662	0.06566	0.2787
14.22	27.85	92.55	623.9	0.08223	0.1039	0.11030	0.04408	0.1342	0.06129	0.3354

texture error	perimeter error	area error	smoothness error	compactness error	concavity error	concave points error	symmetry error	fractal dimension error	worst radius	worst texture
0.6205	1.957	23.35	0.004717	0.02065	0.01759	0.009206	0.01220	0.003130	16.57	20.86
2.3240	2.105	29.96	0.006307	0.02845	0.03850	0.010110	0.01185	0.003589	15.75	40.54

worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	Actual Label	Predicted Label
110.3	812.4	0.1411	0.3542	0.2779	0.13830	0.2589	0.10300	0	1
102.5	764.0	0.1081	0.2426	0.3064	0.08219	0.1890	0.07796	1	0

```
array([[42,  1],
       [ 1, 70]])
```

Discussion

K-fold cross validation with grid search proved useful in finding optimal model hyperparameter values. Further, it seems that on this particular dataset, our loss function can be effectively minimized via gradient descent and subsequently, the logistic regression model learns the given task well. Ultimately, the purpose of this project was deepen my understanding of the mechanics of logistic regression. To this end, the goal is felt to have been achieved.

Further

Some ideas to improve upon the current project include the following:

1. Perform statistical inference on model parameters (i.e. likelihood ratio tests, hypothesis testing on parameter subsets).
2. Add a regularization term to the loss function to reduce dimensionality.
3. Explore likelihood maximization techniques.
4. Verify the assumption of linearity between independent variables and log-odds is not violated with a Box-Tidwell Test.
5. Logistic regression with a linear net input produces a linear classification boundary and thus works best when classes are linearly separable. A function could be created to plot two (or three) features and their decision regions to illustrate this.

References

Addagatla, Arun. “Maximum Likelihood Estimation in Logistic Regression.” *Medium*, Medium, 26 Apr. 2021, https://arunaddagatla.medium.com/maximum-likelihood-estimation-in-logistic-regression-f86ff1627b67.

Holehouse, Alex. “06: Logistic Regression.” *Stanford Machine Learning Notes*, Stanford University, 2011, http://www.holehouse.org/mlclass/06_Logistic_Regression.html.

Leung, Kenneth. “Assumptions of Logistic Regression, Clearly Explained.” *Townrdsdatascience.com*, Medium, 4 Oct. 2021, https://towardsdatascience.com/assumptions-of-logistic-regression-clearly-explained-44d85a22b290. Accessed 2 Dec. 2022.

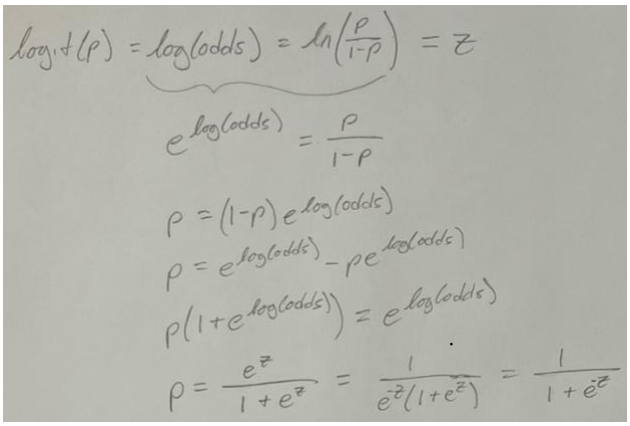
Montgomery, Douglas C., et al. “13. Generalized Linear Models.” *Introduction to Linear Regression Analysis*, 6th ed., Wiley, Hoboken, NJ, 2021, pp. 440–461. Wiley Series in Probability and Statistics.

Raschka, Sebastian, et al. “Chapter 3: A Tour of Machine Learning Classifiers Using Scikit-Learn.” *Machine Learning with PyTorch and Scikit-Learn: Develop Machine Learning and Deep Learning Models with Python*, Packt Publishing Ltd., Birmingham, UK, 2022, pp. 53–102.

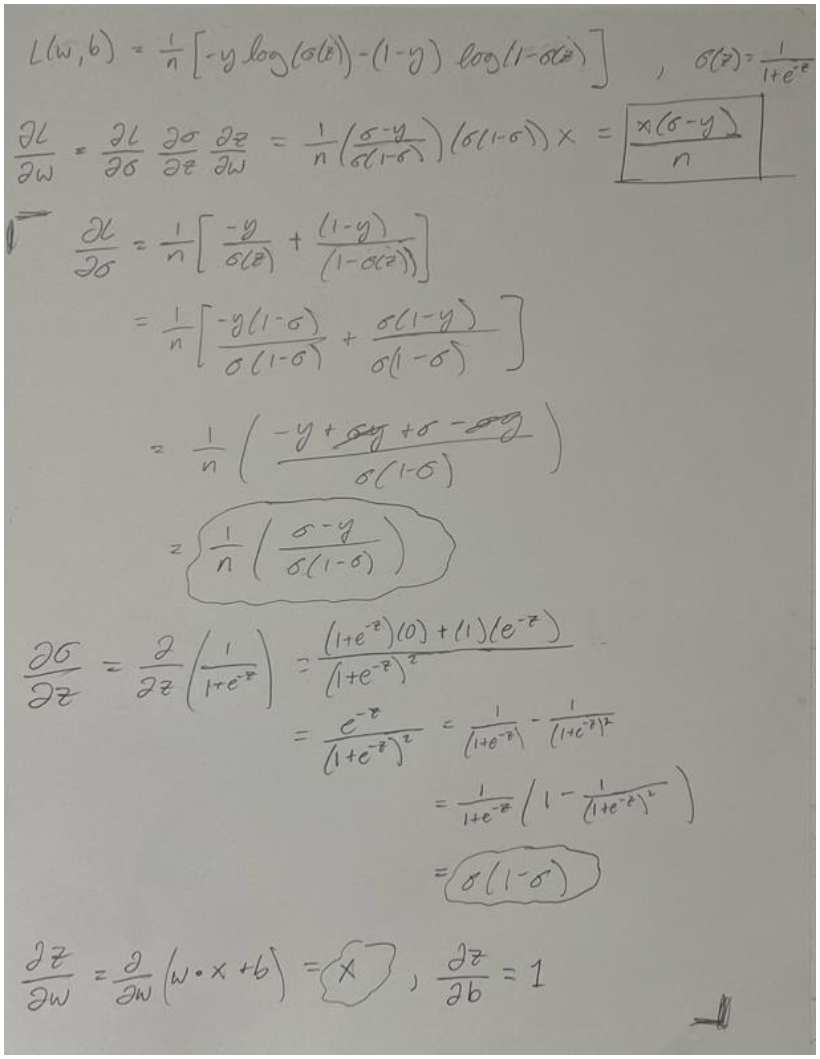
Yun, Sunghee. “Logistic Regression - Prove That the Cost Function Is Convex.” *Stack Exchange*, 23 Apr. 2019, https://math.stackexchange.com/questions/1582452/logistic-regression-prove-that-the-cost-function-is-convex. Accessed 27 Nov. 2022.

Appendix

Derivation of Logistic Sigmoid:



Chain Rule Gradient Derivation (dropping notation for indices):



Second Derivative of the Loss Function: