Draft version April 19, 2021
Typeset using LaTeX default style in AASTeX631

Optimizing Softmax Regression with Markov Chain Monte Carlo

PHYS 321

Zhengwen Fu,[1] Juan Felipe Duran[1] And Alexandre Amédée Stuart[1]

[1] *McGill university, 845 Sherbrooke St W, Montreal, Quebec H3A 0G4*

ABSTRACT

In this project, we implemented a parameter optimizing method based on the Bayesian inference with Markov chain Monte Carlo (MCMC) to optimize the parameters of a Softmax Regression model. We compared our method with the popular Gradient Descent method to investigate the validity and the performance of our method. We tested the Softmax Regression model with both optimization methods on two different multi-class classification tasks: Iris species classification and astronomical object types classification. We then studied the accuracy of the classification results and the training times taken by both optimizing methods. We determined that the Gradient Descent method is consistently much faster than the MCMC method and that the two optimizing methods produced similarly accurate results for the Iris species classification task. However the results of the MCMC method are more accurate for the astronomical object types classification task. This is likely due to the high noisiness of the SDSS dataset, and it may be a good idea to test this hypothesis on more datasets. Moreover, in the case of the MCMC method, we can compute the error bars (credible intervals) on the raw class probability distribution outputs, which is not possible for the Gradient Descent method. The results demonstrated that both methods have different optimal uses: Gradient Descent is to be preferred to design a faster algorithm, while MCMC is to be preferred for quantifying the uncertainties on the parameters and probably for more robust performances on noisy datasets.

Keywords: MCMC — Multi-Class Classification — Gradient Descent

1. INTRODUCTION

Imagine in 1940, you are British and you contemplate with dread, as France surrenders to the Germans. You feel bad and you feel even worse knowing that the British have intercepted German radio messages, which could potentially save millions of lives, but the messages are encoded by a German machine called Enigma, and thus none of the retrieved messages are of any use. All hope is not lost. The British were clever, as they had devised strong statistical models to decipher the codes. However, to their surprise, the Germans prepared for this: they hardened the Enigma machine by changing the parameters of their cipher everyday. This meant that whatever work a mathematician had done by hand towards deciphering a message would be lost at the end of each day. No human could solve equations by hand that fast, including Alan Turing, the father of theoretical computer science and AI. This conundrum led him to come up with a superhuman solution. Inspired by the Bomba Boslaugh (2003), Turing created Bombe, an electro-mechanical device which combined mathematical rigor and fast calculations to decipher the Enigma and eventually help win the war. This is how things were portrayed in the movie "The imitation game"Vogt et al. (2014) which I highly recommend.

We are blessed to be born in a period of relative peace, in which cutting-edge technologies are not just for military purposes. Computers are now extremely common, but optimizing parameters of statistical models through the superhuman computation powered by them is still one of the hottest topics. In this project, we implemented a parameter optimizing method based on the Bayesian inference with Markov chain Monte Carlo (MCMC) Liu (2021) to optimize the parameters of a Softmax Regression model Zhang et al. (2020), which is a statistical or machine learning model for multi-class classification. To investigate the validity and the performance of our method, we compared our method with the Gradient Descent method, which is the overwhelmingly most popular parameter-optimizing method for machine learning models. We adopted an existing implementation Fu et al. (2020) of the Softmax Regression model and the Gradient Descent optimizer. We tested the Softmax Regression model with both optimizing methods on two

different multi-class classification tasks: Iris species classification on the classical Iris dataset Pedregosa et al. (2011) and astronomical object types classification on the Sloan Digital Sky Survey (SDSS) dataset Fernandes (2021).

2. METHODS

2.1. *Software*

Our analysis code (Github page) is written in Python 3 with the IDE Jupyter Notebook.

2.2. *Logistic Regression*

Since Softmax Regression is the generalization of Logistic Regression, we introduce Logistic Regression first. Logistic Regression is for binary classsification of data, i.e. the label y can only be True (1) or False (0).

To classify binary labels, we want to know the probability that a data point with the feature vector x belongs to each label class y, $p(y|x)$. One naive way to tackle would be to parametrize $p(y|x)$ with a linear model,

$$p(y|x) = w^T x \tag{1}$$

where x is the feature vector and w is a weight vector (i.e. parameter vector).

The issue here is that the range of a linear function is all the real numbers, so in order to get proper probabilities out of a linear function, we rely on what is called the logistic function (also known as the sigmoid). The logistic function is defined as follows:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

The key property of the logistic function is that it maps the real line to probabilities; the range goes from all real numbers to the interval [0,1]. Then, we end up with our Logistic Regression model Hamilton (2021):

$$p(y|x) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}} \tag{3}$$

Then, the data point with the feature vector x belongs to the label class $y \in \{1, 0\}$ that gives higher $p(y|x)$.

2.3. *Softmax Regression*

One limitation of Logistic regression is that it is inherently a binary classification approach. To generalize for multi classes, we replace the logistic function with the softmax function and replace the binary label encoding (a single binary value) with the one-hot encoding (binary representation for each label in a vector). The softmax is defined as

$$softmax(z)[i] = \frac{e^{z[i]}}{\sum_{j=i}^{k} e^{z[j]}} \tag{4}$$

So our Softmax Regression Zhang et al. (2020), a linear multi-class classification model, is defined as:

$$p(y = k|x) = softmax(w^T x)[k] \tag{5}$$

Given a training dataset, the log likelihood based on the cross-entropy loss Zhang et al. (2020) is defined as:

$$l(y, \hat{y}) = \sum_{j=1}^{q} y_j \cdot log\hat{y}_j \tag{6}$$

2.4. *Gradient Descent*

To optimize the weights w (i.e. parameters) of the model with a training dataset, the Gradient Descent method repeatedly shifts the weights w in the direction of the gradient of the loss function to minimize the loss function. The loss function is essentially the negative log-likelihood. Therefore, minimizing the loss function is equivalent to maximize the log-likelihood or likelihood. We use the cross-entropy loss Zhang et al. (2020), which is the most common loss function for classification models.

2.5. *Markov Chain Monte Carlo*

Another solution to optimize our model is to use the Metropolis Hastings algorithm, which is an implementation of the Markov Chain Monte Carlo Liu (2021), to infer the distribution of the weights w based on the log-likelihood and pick a sample in the distribution that maximizes the log-likelihood.

We start at some point $W^{(n)}$ in parameter space. Then, we propose a step to a new location by drawing randomly from the proposal distribution $q(W^{'}|W)$. In our case, it is a Gaussian,

$$q(W^{'}|W) \propto exp(\frac{|W^{'} - W^{(n)}|^2}{2\eta^2}) \tag{7}$$

Then, we draw a random number r from a uniform distribution over $0 < r < 1$. If:

$$\frac{p(W^{'}|data)}{p(W^{(n)}|data} > r \tag{8}$$

then we accept the step and call $W^{(n+1)} = W^{'}$. If not, then we repeat the last location in the chain and set $W^{(n+1)} = W^{(n)}$. We keep iterating these steps, which will give a chain of values that are drawn from the probability distribution. These values then give a solid approximation of the posterior. We use the libraries emcee and corner in Python in order to perform MCMC's and output corner plots.

2.6. *Further details*

We divide each of our two datasets into two sets: the training set and the test set. We train the models using the training sets. We then run our models on the features of the test sets and compare our label predictions yo the actual labels of the test sets in order to calculate the accuracy of our model. The time taken to train the training set is also recorded for further analysis. Both of these variables are shown in tables in the results section below for each model.

Let's explore some differences between Gradient Descent and MCMC before we dive into the results. The main difference is that MCMC is not necessarily trying to optimize anything. Instead, it is estimating the probability density distribution of the parameters by generating a lot of random steps. This provides the user with a more descriptive output of the posterior, since we can use the distribution to calculate error bars (credible intervals). Such error bars are given for the first data point in each dataset discussed in the results section below. The limitation of this technique is that it will take longer to calculate than Gradient Descent since it takes a fixed large amount of steps (4000 steps in our algorithm). We expect the Gradient Descent to provide us with just a value of the posterior and be faster, while we expect the MCMC to give a distribution of the posterior and be slower.

2.7. *Datasets*

The classical iris dataset is from the datasets available in the Python library Scikit-Learn Pedregosa et al. (2011). It contains 3 classes of iris with 50 data points for each class. Each label class refers to one of three types of iris plant: Iris-setosa, Iris-versicolour, or Iris-virginica. There are four features in this dataset: sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm.

The SDSS dataset is from Kaggle Fernandes (2021). It contains 3 classes of astronomical objects with totally 10000 data points. Each label class refers to one of three types of astronomical object: GALAXY, STAR, or QSO. Out of the initial 17 features, we decided to discard 4 that were irrelevant (objid,rerun,specobjid,plate) and 4 that were correlated (g,r,i,mjd), leaving us with 9 features. Then, we scaled the dataset, converted the class column to 0,1,2 for easy one-hot encoding, and we converted the panda dataframe into a numpy array.

3. RESULTS AND DISCUSSION

3.1. *Results*

First, as shown in Table 1, the accuracy of each optimizing methods on each dataset was recorded.

Second, as shown in Table 2, the training time taken by each optimizing methods on each dataset was recorded with the library Time from Python.

Moreover, in the case of the MCMC method, the error bars (credible intervals) for the raw label class probability predictions can be computed. As shown in Figure 1 and Figure 2, on both datasets, we calculated the error bars of the parameters from the distribution produced by the MCMC method and then translate them into the error bars of

	MCMC	Gradient Descent
Iris Dataset	97.37%	97.37%
SDSS Dataset	91.28%	83.12%

Table 1. Accuracy of each method for each dataset

	MCMC	Gradient Descent
Iris Dataset	19.945s	0.234s
SDSS Dataset	256.194s	98.847s

Table 2. Time taken by each method for each dataset

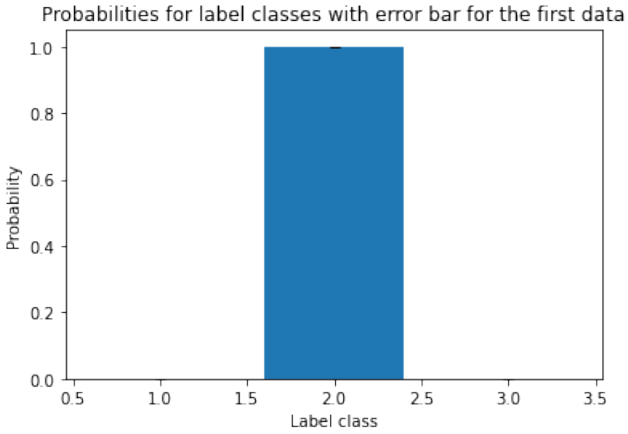

Figure 1. Probabilities for label classes 1,2 ,and 3, with error bars for the first data point in the Iris Dataset

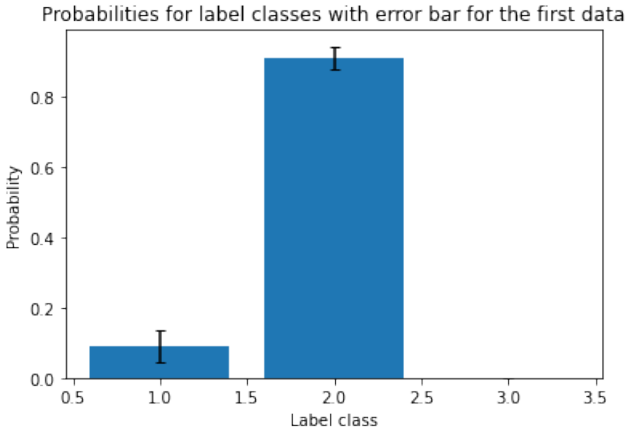

Figure 2. Probabilities for label classes 1,2 ,and 3, with error bars for the first data point in the SDSS Dataset

the probabilities that the points belong to each label class. In contrast, this is not possible for the Gradient Descent method.

Further, the optimizing process of the MCMC method can be visualized with a log-likelihood evolving plot and a cornor plot. As shown in Figure 3 and Figure 4 in the appendix, we made these plots for the Iris dataset.

3.2. *Discussion*

Observing the tables in the results section above we notice that the two methods produced the same high accuracy for the Iris dataset. We also notice that the MCMC was able to provide a better accuracy than Gradient Descent for the SDSS dataset. But neither method's accuracy is very high for the optimization of this dataset as compared to that for the Iris dataset. It is therefore likely that the SDSS dataset is more noisy than the Iris dataset. Considering that the accuracy of the MCMC optimization method is greater than that of the Gradient Descent method in such case, it is likely that the MCMC method is more robust for noisy datasets. In other words, we believe MCMC is stronger in terms of determining the optimal values of the parameters on noisy datasets since it gives a broader, more integral image of the function we are trying to optimize, with its ability to realize density distributions.

On the other hand, Gradient Descent outperformed MCMC in terms of time. The Gradient Descent algorithm was approximately 161x faster for the Iris dataset and approximately 2.6x faster for the SDSS dataset. This behaviour was expected, since MCMC goes through a fixed large amount of steps, and thus even when dealing with simple datasets such as the Iris one, it requires a considerable amount of time, compared to Gradient Descent.

Further, we were able to calculate error bars for the probabilities that a point belongs to a class, which was only possible with the MCMC algorithm. Computing these error bars added supplemental time to the predicting process. The longer the stored MCMC chain, the more time the error computation will take but the more accurate it will be. For example, in our experiment on the SDSS dataset, we kept the whole 4000 steps MCMC chain to compute the error bars and it took $\approx$ 10min to compute error bars during predicting. Computing these error bars is extremely beneficial in the context of science, where uncertainty sometimes provides us with more information than the data itself!

Regardless, both methods have advantages and disadvantages. But while Gradient Descent is the most favoured optimization method in machine learning, there is not yet a golden standard for optimization where uncertainty on the predicted parameters is required. And our implementation of Bayesian inference and MCMC is definitely not it, given its extreme training and prediction times. As of today, many scientists around the world are searching for new solutions to this problem, to allow our superhuman tools (in this case, computers) to better model, and thus analyse and understand the world around us.

4. CONCLUSION

To conclude, while in the well known Iris dataset, both optimization methods gave the same accuracy (97.37%) and the Gradient Descent method was approximately 161 times faster than MCMC method, we cannot omit the importance of computing error bars on predicted values, which can only be done with MCMC and not with Gradient Descent. Further, we were able to classify the SDSS data set into stars, quasars and galaxies with an accuracy of 83.12% with the Gradient Descent and it took 98.85 seconds. Similarly, we were able to classify the same dataset with the MCMC with accuracy 91.28% and it took 256.20 seconds and an additional prediction time for computing the error bars that can be large or small depending on the wanted accuracy of the error bars. Both methods worked well and we were able to inspect their behaviour and appreciate their corresponding pros and cons. While Gradient Descent was faster to compute, in the case of the SDSS, MCMC yielded results that were significantly more accurate. It is likely that this is because the SDSS data set is more noisy, but it might be of value to run both optimizers on more data sets and test this hypothesis. The Bayesian inference with MCMC was also able to provide us with error bars. These are of course extremely valuable in a scientific context, where we have a culture of completely rejecting any number that does not have error bars. One conclusion that we can all agree on is that the marriage of statistical models and computers have allowed us to go very far in terms of exploring the universe. It took a great mind to come up with such excellent ideas that revolutionized and changed humanity forever. We wonder what type of mind will it take to come up with the next idea that will revolutionize humanity. A superhuman one?

5. STATEMENT OF CONTRIBUTIONS

Zhengwen Fu contributed to the model implementing and report writing.

Juan Felipe Duran contributed to the code reviewing and report writing.

Alexandre Amédée Stuart contributed to the dataset preparing and report writing.

REFERENCES

Boslaugh, D. 2003, When Computers Went to Sea: The Digitization of the United States Navy, Perspectives (Wiley). https://books.google.ca/books?id=Mi8MhzheOokC

Fernandes, M. 2021, Logistic Regression + SGD in Python from scratch. https://www.kaggle.com/marissafernandes/logistic-regression-sgd-in-python-from-scratch/execution

Fu, Z., Cai, L., & Xiao, S. 2020, Multi-Class Logistic Regression and Gradient Descent. https://github.com/Foo2000/Multi-Class-Logistic-Regression-and-Gradient-Descent

Hamilton, W. 2021, 845 Sherbrooke St W, Montreal, Quebec H3A 0G4. https://cs.mcgill.ca/~wlh/comp451/files/comp451_chap6.pdf

Liu, A. 2021, CodingLab07_MCMC

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825

Vogt, F. P. A., Dopita, M. A., Kewley, L. J., et al. 2014, ApJ, 793, 127, doi: 10.1088/0004-637X/793/2/127

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. 2020, Dive into Deep Learning

194

APPENDIX

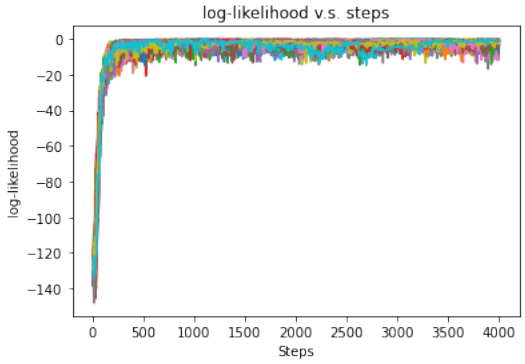

Figure 3. Log-likelihood as a function of steps. The first roughly 200 steps represent the burn-in.

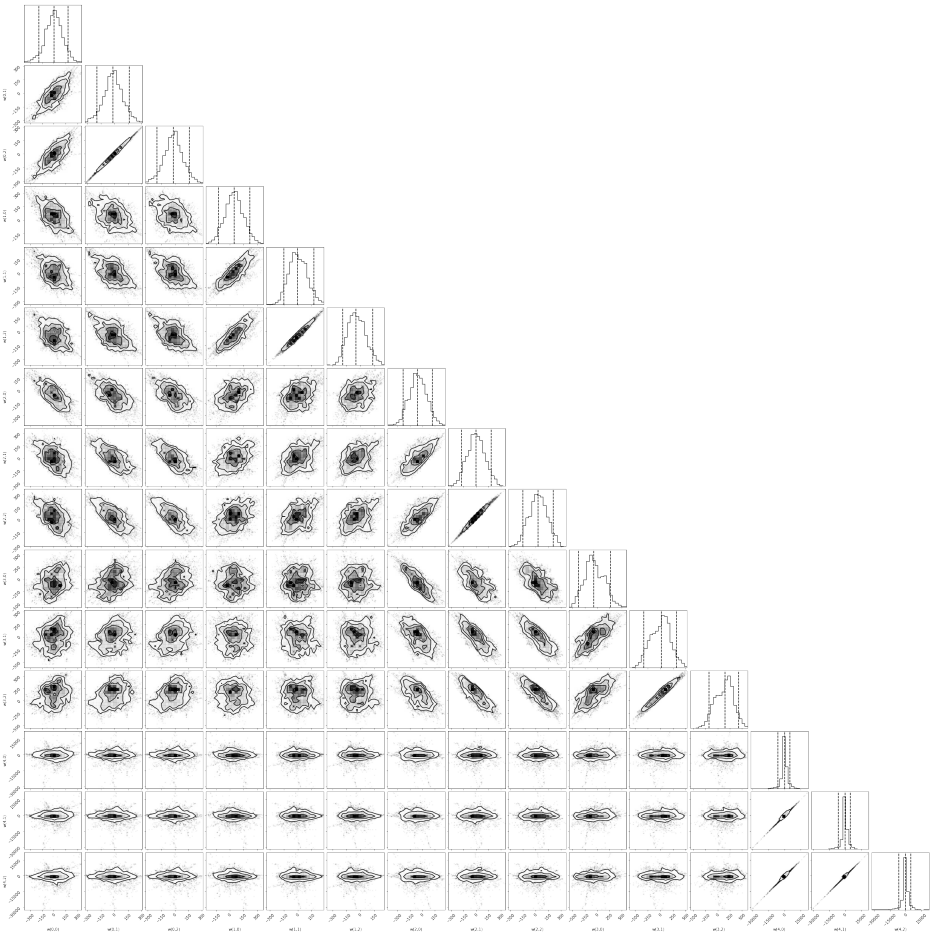

Figure 4. Corner plots from MCMC