Tufts University

CS 136 - 2022s
Final Project Write-up

Alexander Lobo and Nate Davis

May 13, 2022

CS 136 - 2022s - Final Project Write-up

Student Names: Alexander Lobo and Nate Davis

Collaboration Statement:

Turning in this assignment indicates you have abided by the course Collaboration Policy:

`www.cs.tufts.edu/comp/136/2022s/index.html#collaboration-policy`

Total hours spent: 20

We consulted the following resources:

- Course website
- Lecture Videos
- Bishop Textbook

Contents

Project Summary

Our dataset contains 6,598 confirmations (orientations/rotations) of 102 molecules, all of which have been classified in terms of smell as musk or non-musk by human experts. If a molecule has been deemed a musk, all of its confirmations are listed so, and the same is true for non-musks. The dataset is imbalanced, with 5,581 non-musks and 1,016 musks. The features of our dataset comprise 166 measures of intramolecular distance.

We use a logistic regression model with sigmoid link function and multivariate Gaussian prior on the weight vector to predict whether a given confirmation is musk or non-musk. Originally, our learning method was first-order stochastic gradient descent. However, after experiencing some trouble implementing a second-order stochastic gradient descent for our upgrade, we opted to implement batch versions of our methods to compare instead. In order to test for convergence, we calculate a adjusted log loss (which includes a term for the prior) per training example before each weight update and compare it to the log loss from the prior step. If the difference in the log loss between steps becomes less than the predefined threshold or if the maximum iterations is met, then the algorithm terminates.

Our upgrade is a change of learning method to second-order gradient descent. The main problem we seek to address with this upgrade is the time it takes to train our model. Incorporating information from the second derivative of the loss function, as occurs in second-order gradient descent, should allow the model to converge more quickly.

We have made some changes to our first-order gradient descent implementation since checkpoint 2, so we produce new baseline measurements below, but the metrics by which we compare our model pre- and post-upgrade are number of iterations until convergence, time until convergence, and accuracy.

Upgrade Implementation

Background

In Bayesian logistic regression, the Posterior distribution is a useful probability distribution that may be used to predict the probability of certain weight vectors given then occurrence of specific data. It is a powerful tool that allows us to utilize information from the Likelihood (probability of data given weight vector), the Prior (a priori knowledge about the weight vector), the Evidence (marginal probability of the data). One goal of Bayesian reasoning for logistic regression is to find the most probable weight vector that maximizes the probability density of the Posterior, which is known as the MAP estimate. By taking the negative of the Posterior, we can convert our optimization problem from a maximization to a minimization problem by convention. However, unlike with linear regression, the MAP estimate of logistic regression is not known to have an analytical solution. Therefore we depend on optimization methods like gradient descent to solve

$$\hat{\mathbf{w}}_{\text{MAP}} = \arg\min_{\mathbf{w}\in\mathbb{R}^M} -\ln p(\mathbf{w}|\mathbf{t}) \tag{1}$$

where M is the feature dimension and $-\ln p(\mathbf{w}|\mathbf{t})$ is the negative of the posterior distribution.

We then have two options to choose from: first-order and second-order gradient descent. First order gradient descent uses a first-order derivative of the Posterior to make a step change in the weight vector, while second-order gradient descent uses a second-order derivative. Changing the gradient descent method from first-order to second-order is the chosen upgrade to improve the prediction of molecule confirmations as either musk (positive output class) or non-musk (zero output class) smelling.

Implementation

In Checkpoint 3, we derived the following update equations that would be use for our pre- and post-upgrade models using stochastic gradient descent. Each method has two upgrade equations because we originally decided to keep the weight vector and bias term stored as separate variables.

First-Order Stochastic Gradient Descent

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta \underbrace{\left[\left(\sigma(\mathbf{w}^t \cdot x_{(i)}) - y_{(i)}\right) x_{(i)} + \alpha \mathbf{w}^t\right]}_{g_{(i)}(\mathbf{w}^t)} \tag{2}$$

$$c^{t+1} \leftarrow c^t - \eta(\sigma(\mathbf{w}^t \cdot x_{(i)}) - y_{(i)}) \tag{3}$$

Second-Order Stochastic Gradient Descent

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta \underbrace{\left[r_{(i)}(\mathbf{w^t})(x_{(i)} \otimes x_{(i)}) + \alpha\right]^{-1}}_{H_{(i)}(\mathbf{w}^t)^{-1}} \underbrace{\left[\left(\sigma(\mathbf{w}^t \cdot x_{(i)}) - y_{(i)}\right) x_{(i)} + \alpha \mathbf{w}^t\right]}_{g_{(i)}(\mathbf{w}^t)} \tag{4}$$

$$c^{t+1} \leftarrow c^t - \eta \left[r_{(i)}(\mathbf{w^t})(x_{(i)} \otimes x_{(i)})\right]^{-1} \left[\left(\sigma(\mathbf{w}^t \cdot x_{(i)}) - y_{(i)}\right)\right] \tag{5}$$

We found that the first-order stochastic gradient descent update worked quite well, and we were able to show a noisy decrease in the loss function moving towards the global minimum. It seemed promising that a similar result would be achieved for the second-order stochastic gradient descent. Instead, the log loss would always shoot to infinity after a couple iterations regardless of how we tuned the hyperparameters. Our code would specifically error when trying to compute the inverse of the Hessian, throwing a "singular matrix error". We tried debugging the code for a while before concluding that the inverse Hessian calculation does not perform properly when using a single example. Hence, we abandoned the stochastic approach and decided to implement batch gradient descent methods instead. In order to simplify our weight update equations, we also decided to combine the weights and bias terms into a single weight vector so that only a single weight update equation is needed for each method.

To derive the necessary equations for first- and second-order gradient descent, we must revisit the log loss for Bayesian logistic regression that was derived in Checkpoint 3:

$$\ln \mathcal{L}(\mathbf{w}) = -\left(\sum_{n=1}^{N} t_n \ln(\sigma(\mathbf{w}^\top x_n)) + (1 - t_n) \ln(1 - \sigma(\mathbf{w}^\top x_n))\right) + \frac{1}{2}\alpha \mathbf{w}^\top \mathbf{w} \tag{6}$$

$$= -\left(\mathbf{t}^\top \ln(\sigma(\mathbf{xw})) + (1 - \mathbf{t})^\top \ln(1 - \sigma(\mathbf{xw}))\right) + \frac{1}{2}\alpha \mathbf{w}^\top \mathbf{w} \tag{7}$$

First-Order Batch Gradient Descent

Now, with using Bishop equation (4.91), we can take the gradient of the log loss with respect to $\mathbf{w}$ to obtain

$$\nabla_{\mathbf{w}} \ln \mathcal{L}(\mathbf{w}) = \sum_{n=1}^{N} \sigma(\mathbf{w}^\top x_n) - t_n) x_n + \alpha \mathbf{w} \tag{8}$$

Using matrix notation, the gradient of the log loss can be written as

$$g(\mathbf{w}) = \nabla_{\mathbf{w}} \ln \mathcal{L}(\mathbf{w}) = \mathbf{x}^\top (\sigma(\mathbf{xw}) - \mathbf{t}) + \alpha \mathbf{w} \tag{9}$$

We can now use the gradient to make a step change given some step-size η and iterate until we converge to a solution:

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta \underbrace{\left[\mathbf{x}^\top(\sigma(\mathbf{x}\mathbf{w}^t) - \mathbf{t}) + \alpha\mathbf{w}^t\right]}_{g(\mathbf{w}^t)} \tag{10}$$

Second-Order Batch Gradient Descent

In order to perform second-order gradient descent, we can consider the second-order gradient Hessian matrix $H(\mathbf{w}) \in \mathbb{R}^{M\times M}$ to be

$$H(\mathbf{w}) = \nabla_{\mathbf{w}}\nabla_{\mathbf{w}} \ln \mathcal{L}(\mathbf{w}) = \mathbf{x}^\top R(\mathbf{w})\mathbf{x} + \alpha I_M \tag{11}$$

where $R(\mathbf{w}) \in \mathbb{R}^{N\times N}$ is defined as

$$R(\mathbf{w}) = \begin{bmatrix} \sigma(\mathbf{w}^\top x_1)(1-\sigma(\mathbf{w}^\top x_1) & & \\ & \ddots & \\ & & \sigma(\mathbf{w}^\top x_n)(1-\sigma(\mathbf{w}^\top x_n) \end{bmatrix} \tag{12}$$

a diagonal matrix with all 0 off-diagonal entries. Now each update in the weight vector is

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta H(\mathbf{w}^t)^{-1} g(\mathbf{w}^t) \tag{13}$$

By substituting the formulas given by equation (9) and (11) into the weight update equation, we obtain

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta \underbrace{\left[\mathbf{x}^\top R(\mathbf{w})\mathbf{x} + \alpha I_M\right]^{-1}}_{H(\mathbf{w}^t)^{-1}} \underbrace{\left[\mathbf{x}^\top(\sigma(\mathbf{x}\mathbf{w}) - \mathbf{t}) + \alpha\mathbf{w}\right]}_{g(\mathbf{w}^t)} \tag{14}$$

Code Implementation

Now, the following steps are taking to implement the first- and second-order batch gradient descent learning algorithm:

1. Initialize the python class `MAPEstimator()` with parameters (e.g. prior weight vector, solver type, maximum iterations, convergence tolerance, step size, etc.)
2. Use `if` statement to determine if the solver type is first-order (`'fo'`) or second-order (`'so'`).

3. initialize loop counting parameter to keep track of number of iterations and initialize loss array to keep track of log loss changes between iteration steps.
4. Transform the training dataset to include a column of ones (this gets matrix multiplied by the bias term in the weight vector).
5. Initiate `while` loop.
6. Use equation (10) or equation (14) to perform the weight update, depending on the solver method.
7. Compute updated log loss using updated weight vector.
8. Iterate until maximum iterations are reached or the difference in log loss between steps becomes less than the pre-defined tolerance.

Bottlenecks

Stochastic vs. Batch Gradient Descent

As discussed before, one of the issues we ran into was getting our second-order stochastic gradient descent method to converge. Although we could not figure out the issue (or even determine if it is even feasible), we addressed this issue by pivoting our project to focus on batch first and second-order gradient descent. Although we had to spend a bit of extra time to re-implement our pre-upgrade code, we are now able to obtain and analyze the results for both methods.

Weight updates

TODO: Describe how keeping the bias and weight terms separate led to some issues.

Training Time

Considering that the computational complexity ($O(m^3)$) of second-order gradient descent, we expected the second-order method to take a bit longer to converge than the first-order method when the hyperparameters are not optimally set. However, we instead found that the second-order method took much longer than expected. The second-order method is also a lot more sensitive to the step-size than the first order method is. As a result, it was much more difficult to tune the hyperparameters for the second-order method. Considering how long each grid search would take (10 minutes on average), we ultimately decided to limit the maximum iterations to 100 to save time.

Performance Hypotheses for Upgrade

Hypothesis 1: We hypothesize that our upgrade from first- to second-order gradient descent will cause our model to converge in fewer iterations because second-order gradient descent's incorporation of the second-degree derivative of the loss function should make each step more efficient in minimizing the loss.

- Our implementation of gradient descent keeps track of the number of iterations until convergence, and we look at the distribution of these counts over k-fold cross validation with 10 folds.

Hypothesis 2: We hypothesize that our upgrade to second-order gradient descent will cause our model to converge in less time because, although time complexity of one step of second-order gradient descent is $O(m^3)$, versus $O(m)$ for first-order gradient descent (with m being the size of the weight vector), the aforementioned incorporation of second-degree derivative information should lower the total number of steps

- We calculate runtime until convergence using the python `time()` function for each of the 10 iterations in our k-fold cross validation.

Hypothesis 3: We hypothesize that our upgrade to second-order gradient descent will have no major effect on the accuracy of our model, as in theory it should converge to the same minimum as first-order gradient descent, just faster.

- We measure accuracy on held-out data via k-fold cross validation with 10 folds.

TODO: Expand on "Links to previously described model or inference property of upgrade"

TODO: Expand on "Links to previously described dataset property of upgrade"

Evaluating hypotheses for upgrade

For each of our hypotheses, we used the following hyperparameters for our learning methods:

Table 1: Optimized Hyperparameters

	Gradient Descent Method	
Hyperparameter	First-Order	Second-Order
step size (η)	0.0001	0.1
α	1	1
max. iterations	1000	100
tolerance	0.00001	0.01

TODO: DISCUSSION OF HYPERPARAMETER CHOICE

Hypothesis 1

We hypothesized that using second-order gradient descent, our model would converge in fewer iterations. To evaluate this hypothesis, we performed k-fold cross validation with 10 randomly selected folds on our model using both first- and second- order gradient descent. For each of the 10 folds, we tracked the number of iterations it took to converge.

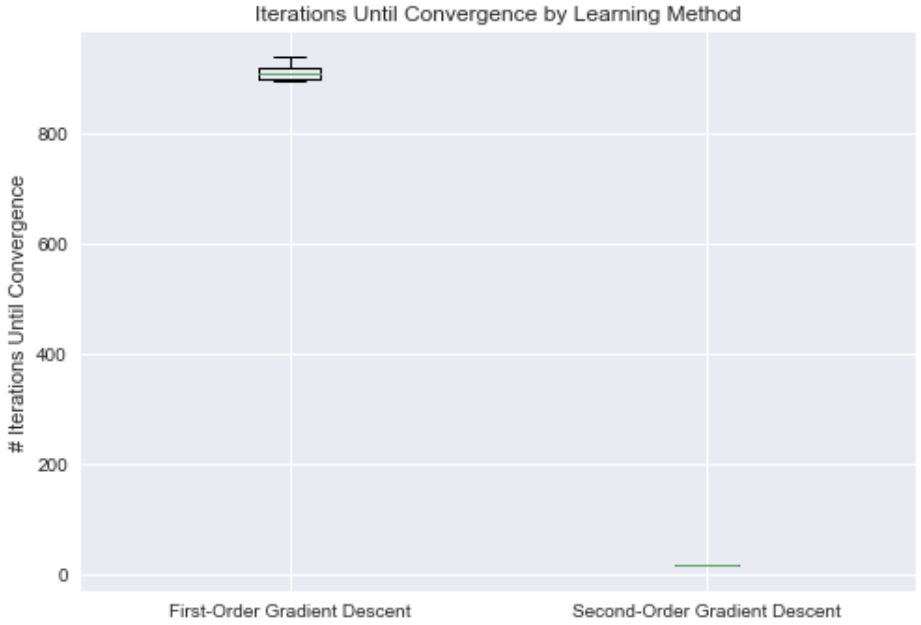

Figure 1. *Held-out accuracy rate by learning method*

Every run of first-order gradient took almost two orders of magnitude more iterations than second-order gradient descent to converge, with each first-order run taking about 900 iterations and each second-order run taking 20.

TODO: Expand on results for each upgrade

- Analyze the implications of the result in approximately 1 paragraph. This should link back to the specific dataset and model/learning method properties you included in your original hypothesis.
- Was your hypothesis correct? Spend 2-3 sentences reflecting on why that might be the case.

Hypothesis 2

We hypothesized that using second-order gradient descent, our model would converge in less time. To evaluate this hypothesis, we performed k-fold cross validation with 10 randomly selected folds on our model using both first- and second- order gradient descent. For each of the 10 folds, we tracked the time it took for the model to converge using the time() function in Python.

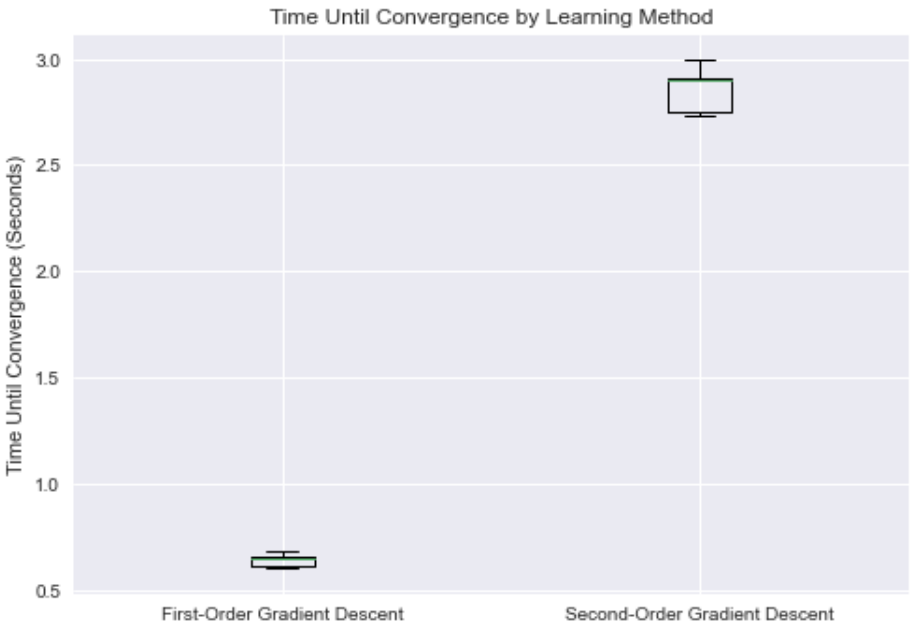

Figure 2. *Time until convergence by learning method*

In contrast to number of iterations, every run of second-order gradient descent took approximately 4 times longer than each run of first-order gradient descent.

TODO: Expand on results for each upgrade

- Analyze the implications of the result in approximately 1 paragraph. This should link back to the specific dataset and model/learning method properties you included in your original hypothesis.
- Was your hypothesis correct? Spend 2-3 sentences reflecting on why that might be the case.

Hypothesis 3

We hypothesized that using second-order gradient descent, our model would exhibit similar accuracy. To evaluate this hypothesis, we performed k-fold cross validation with 10 randomly selected folds on our model using both first- and second- order gradient descent. For each of the 10 folds, we tracked score the accuracy of our model on the held-out data.

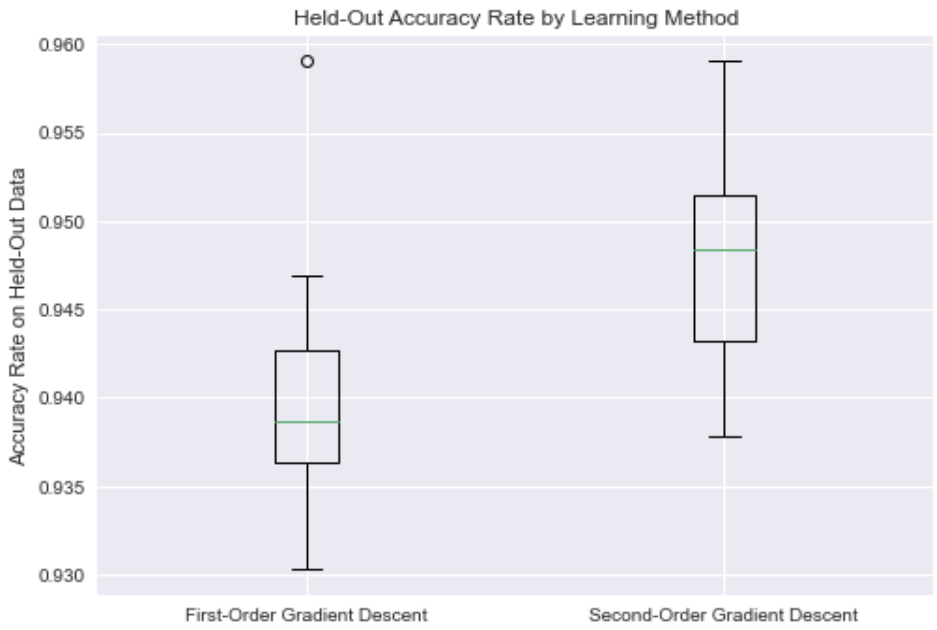

Figure 3. *Held-out accuracy rate by learning method*

While each method exhibited a range of accuracy, second-order gradient descent on average converged to a higher accuracy rate.

TODO: Expand on results for each upgrade

- Analyze the implications of the result in approximately 1 paragraph. This should link back to the specific dataset and model/learning method properties you included in your original hypothesis.
- Was your hypothesis correct? Spend 2-3 sentences reflecting on why that might be the case.

Subsection grading rubric (for each hypothesis):

- Describe implementation details of how you evaluated your hypothesis (1 point)
- Include a specific result linked to the evaluation of your hypothesis (2 points)
- Is your result coherently presented (axis labels, titles, legends etc) (1 point)

- Description of the behavior of result (2 points)
- Analysis of implication of result (3 points)
- Link back to hypothesis: why was or wasn't it right? (2 points)

Reflection

TODO: Complete Reflection

In this section, you should reflect on your project in 2-3 paragraphs, being sure to answer the following questions:

- Did anything about how your original model worked on your data surprise you? (3 points)
- What about your upgrade? (3 points)
- If you were to continue working with this data, what would you like to try next? Why? (3 points)

This section should 1/2-2/3 of a page long.