CSE 546: MACHINE LEARNING HOMEWORK 2

DAVID P. FLEMING

Contents

Introduction

Please note that a copy of all the code I wrote to answer the questions in this assignment are included in my submission but also located online at `https://github.com/dflemin3/CSE_546/tree/master/HW2`. Some scripts require data, such as MNIST data, to run to completion and were not included on my github or in my submission due to file size constraints. If requested, I will happily send a compressed directory containing my homework and detailed instructions on how to reproduce my work. My primary email is dflemin3 (at) uw (dot) edu. Also for reproducibility, I set the random number generator seed near the top of each script such that rerunning my script will yield the same answers as presented here.

I collaborated with Matt Wilde, Serena Liu, and Janet Matsen for various questions on this assignment.

Question 1: Multi-Class Classification using Least Squares

In this question, I solve 10 linear regression problems using regularized Ridge Regression for all 10 digits of the MNIST dataset to build a "one vs all" classifier. Each regression is a binary classifier for the corresponding digit. I classify a sample according to the largest predicted score among my 10 predictors. The code used

Date: October 28^{th}, 2016.

to solve this question is in the following attached files: `hw2_1.1.py, hw2_1.2.py, classifier_utils.py, regression_utils.py, validation.py, mnist_utils.py`.

1.1: One vs all classification.

1.1.1. For ordinary least squares (OLS) linear regression, $\hat{Y}_{OLS} = X(X^TX)^{-1}X^TY$ where X is an $N \times d$ matrix and Y is an $N \times 1$ vector for N samples and d features. The complexities of taking multiple matrix products sum since they happen sequentially. If we take the pairwise matrix products to compute $\hat{Y}_{OLS}$, we get $\mathcal{O}(Nd^2) + \mathcal{O}(Nd^2) + \mathcal{O}(dN^2) + \mathcal{O}(N^2)$. Combining those terms with the $\mathcal{O}(d^3)$ complexity from matrix inversion, I get a total complexity of $\mathcal{O}(d^3 + (1 + 2d)N^2 + Nd^2)$ which asymptotes to $\mathcal{O}(d^3 + dN^2 + Nd^2)$. If $N > d$, this term can be reduced further however I make no assumptions about whether d or N is larger. For a regularized regression, specifically the ridge regression I used for this question, The estimator is $\hat{Y}_{ridge} = X(\lambda I + X^TX)^{-1}X^TY$ has the same asymptotic complexity as OLS since the addition of the identity matrix adds $\mathcal{O}(d^2)$ which is dominated by the other terms. Therefore if I solve k linear regressions, the total complexity would be $\mathcal{O}(kd^3 + kdN^2 + kNd^2)$ as this entire procedure is repeated k times within a loop.

1.1.2. In this question I compute the complexity for the OLS solution and then state how it generalizes to Ridge Regression. I can improve upon the complexity found in the previous problem by nothing that I can fit for all weight coefficients for each class by having my regression yield a prediction vector instead of a prediction value. This can be accomplished by replacing the $n \times 1$ label vector Y by the $n \times k$ matrix Y for k classes. Each row of Y would be 0 except for the index equal to the class label, e.g., index 5 would be 1 for a label of 4 and all other indices would be 0 in that row. With this representation, the estimator remains $\hat{Y}_{OLS} = X(X^TX)^{-1}X^TY$ where $\hat{Y}_{OLS}$ is now a $n \times k$ matrix. To make a prediction for the ith sample, one need only return the `argmax(`$\hat{y}_i$`)`. Note that for my implementation, I use ridge regression where $\hat{Y}_{ridge} = X(\lambda I + X^TX)^{-1}X^TY$ but the same argument still holds.

As before, the complexities of taking multiple matrix products sum since they happen sequentially. If we take the pairwise matrix products to compute $\hat{Y}_{OLS}$, we get $\mathcal{O}(Nd^2) + \mathcal{O}(Nd^2) + \mathcal{O}(dN^2) + \mathcal{O}(kN^2)$ where the extra factor of k on the last term comes about from the fact that Y is now $n \times k$. There is again an additional factor of $\mathcal{O}(d^3)$ from the matrix inversion. This gives a total complexity of $\mathcal{O}(d^3 + Nd^2 + (d + k)N^2)$ which is less than complexity of solving k linear regressions: $\mathcal{O}(kd^3 + kdN^2 + kNd^2)$.

1.1.3. I trained my classifier using Ridge Regression to leverage regularization to prevent my weight vector coefficients from becoming to large. The optimal regularization parameter λ was fit for along a regularization path. Since my training set was sufficiently large with $N = 60,000$ samples, I randomly partitioned it into a validation set with $N_{val} = 6,000$ and a training set with $N_{train} = 54,000$. I iterated over $\lambda \in [10^{-1}, 10^{4}]$ in 5 logarithmic bins for 5 of the digits. For each λ, I fit the ridge regression on the training set then tested it on the validation data. I used the validation predictions to compute the 0-1 loss and stored it for each point. I found the optimal $\lambda = 10^4$ from the minimum value of the 0-1 loss vector. I adopted this value for λ as it was approximately the same for all tested digits. With the optimal λ, I then fit the ridge regression model on the entire $N = 60,000$ sample training set for each digits to get a model with a w_0 with shape $k \times 1$ for $k = 10$ digits and a $d \times k$ $\hat{w}$.

To make a prediction, I compute a $k \times 1$ prediction vector $\hat{y}_i$ using all k regression models on the *ith* sample. The class was selected using the regression model which predicted the largest element of $\hat{y}_i$. On the training set, I found a 0/1 loss of 0.141 and a square loss of 265.5. On the testing set, I found a 0/1 loss of 0.138 and a square loss of 264.4. Note that when computing both losses, I normalized by the number of samples in each set for comparison purposes since otherwise the training set loss would be much larger than the testing set loss by virtue of the training set having 6 times as many samples. Additionally for the square loss, I summed over the differences between all elements on my prediction vector $\hat{y}_i$ and the label vector y_i instead of just comparing my predicted digit against the truth according to the following given formula for the square loss:

$$L(w) = \frac{1}{N}\sum_{i=1}^{N}\left(y^i - \sum_{j=1}^{k} w_j h_j(x^i)\right)^2 \tag{1}$$

. Overall, my model performs rather well as it classifies ∼86% of the samples correctly in both sets.

1.2: Neural Nets with a random first layer: Using more features. I mapped X from a $n \times d$ matrix to a $n \times k$ matrix for $k = 10000$ according to the procedure outlined in the question. Specifically, I generated a $d \times k$ matrix v where each column v_j was a vector of length d whose elements were independently sampled from the standard uniform distribution. I then mapped X by taking the dot product $X \cdot v$ subject to the constraint that $h_j(x) = \max(v_i \cdot x, 0)$ is the *jth* column out of k total of the updated X matrix. I applied this procedure to both the training and testing images from the MNIST dataset taking care to use the same v for both mappings. I cached v for future questions.

Following the same procedure outlined in the previous question, I ran my linear regression multi-class algorithm on the mapped MNIST data. Given the size of the data involved, namely a 60000×10000 input matrix, solving for the coefficients using Ridge Regression was rather computationally expensive. As such, I tested a few values of λ for regularization starting with the best fit of 10^4 found in the previous section. I tested other values according to the following heuristic presented by Prof. Kakade in lecture:

$$\lambda_{best} = cE[||X||^2] \tag{2}$$

where c is a scale constant. I found $c = 1.0 \times 10^{-2}$ gave the best performance for this data.

I fit the transformed data using a multi-class Ridge Regression algorithm as was done on the original MNIST data in the previous question. On the training set, I found a 0/1 loss of 0.141 and a square loss of 265.5. On the testing set, I found a 0/1 loss of 0.228 and a square loss of 260.4. Interestingly, I found comparable losses on the training data as I did for my Ridge Regression on the original MNIST data. Regression on the transformed testing set, however, yielded much worse losses that the regression on the original MNIST testing set.

Question 2: Multi-Class Classification using Logistic Regression and Softmax

The code used to answer this question and all subquestions are contained within the following python files: `classifier_utils.py, validation.py, gradient_descent.py, mnist_utils.py, hw2_2.1.py`.

2.1: Binary Logistic Regression. For this question, I filtered the MNIST data such that all Y digits equal to 2 were set to 1 and all other labels set to 0 to make this a binary classification problem. I trained a logistic regression model on the MNIST training data using regularized batch gradient ascent to minimize the log-loss of the data as defined in class notes as

$$NLL = -\sum_j [y^j(w_0 + \sum_{i=1}^d X_i^j \hat{w}_i) - \log(1 + \exp(w_0 + \sum_{i=1}^d X_i^j \hat{w}_i))]. \tag{3}$$

where NLL indicates that the log-loss = -log-likelihood. I derived the update rules for w and w_0 used for my algorithm by taking the derivative of the NLL, Eqn. 3, with an additional l_2 norm penalty of $\lambda \sum ||w||_2^2$ to get a regularized solution.

My update rules for gradient descent are as follows:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta(\lambda w_i^{(t)} - \sum_j x_i^j [y^j - \hat{P}(Y^j = 1|x^j, w^{(t)})]) \tag{4}$$

and for the unregularized constant offset term

$$w_0^{(t+1)} \leftarrow w_0^{(t)} - \eta(-\sum_j [y^j - \hat{P}(Y^j = 1|x^j, w^{(t)})]). \tag{5}$$

Minimizing Eqn. 3 over the training set by applying the update rules given in Eqn. 4 and Eqn. 5 in a gradient descent algorithm then yields the predicted parameters $\hat{w}, w_0$. I called my solution converged once the log-loss changed by less than 0.25%. I varied my convergence threshold and found comparable performance.

For my regularized batch gradient ascent implementation, I used the l_2 norm penalty as was derived in class. When fitting this algorithm, I need to optimize of the following hyperparameters: the learning rate η and the regularization constant λ. The hyperparameters of my model were simultaneously fit for along a regularization grid. Since my training set was sufficiently large with $N = 60,000$ samples, I randomly partitioned it into a validation set with $N_{val} = 6,000$ and a training set with $N_{train} = 54,000$. I iterated over a two-dimensional grid of $\eta \in [10^{-6}, 1]$ in 6 logarithmic bins and $\lambda \in [10^{-3}, 10^{3}]$ in 6 logarithmic bins. For each (η, λ) pair, I ran batch gradient ascent on the training set then tested it on the validation data. I used the validation predictions to compute the 0/1 loss and stored it for each grid point. I found the optimal $\eta = 2.5 \times 10^{-4}$ and $\lambda = 10^{3}$ from the minimum value of the 0/1 loss grid.

2.1.1. As discussed above in my description of how I optimized my hyperparameters, I found that $\eta = 2.5 \times 10^{-4}$ gave the best fit. In practice in my algorithm at the start of each iteration, I reset $\eta = k\eta_0/N$ where N is the number of samples fit over in a given batch, $k = 1/\sqrt{t}$ is a scaling constant where t is the iteration number and $\eta_0 = 2.5 \times 10^{-4}$. I found that the inclusion of k which tends to suppress later step sizes prevented the solution from bouncing around optima and instead allowed for more runs to converge. In practice, k could also slow down convergence by reducing the step size even if the solution was tending towards the correct global optimum but I found that this was not an issue.

2.1.2. With my optimal hyperparameters, I refit the batch gradient ascent on the entire $N = 60,000$ sample training set to get the final fit parameters w_0 and $\hat{w}$. At each iteration of this fit, I computed the log-loss and 0/1 loss for both the training and testing set and plotted them in Fig. 1 and 2 below. As expected, the testing error is worse than the training error for most of the batch gradient ascent path. Interestingly, the testing loss actually dips below the training loss at the end of the fit near convergence.

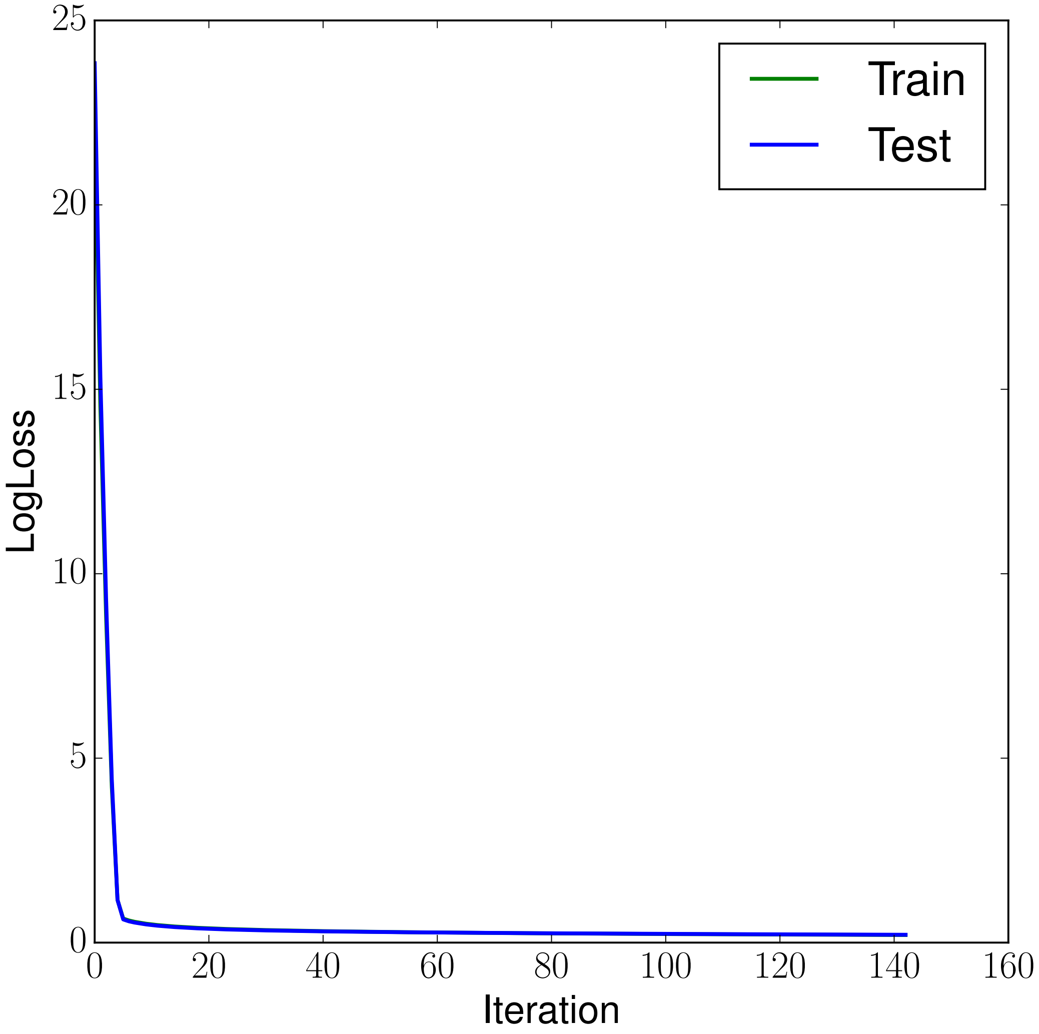

Figure 1. Log loss as a function of iteration for both the MNIST training and testing datasets for binary logistic regression. The log loss on the testing set was evaluated using the model parameters computed by fitting on the training set for each iteration. The log-losses for both the training and testing sets are very close and hence their evolution appear to overlap in the figure. Note how the loss decreases for both sets with each iteration as the fit improves. In general, the testing loss is slightly larger than the training loss as expected.

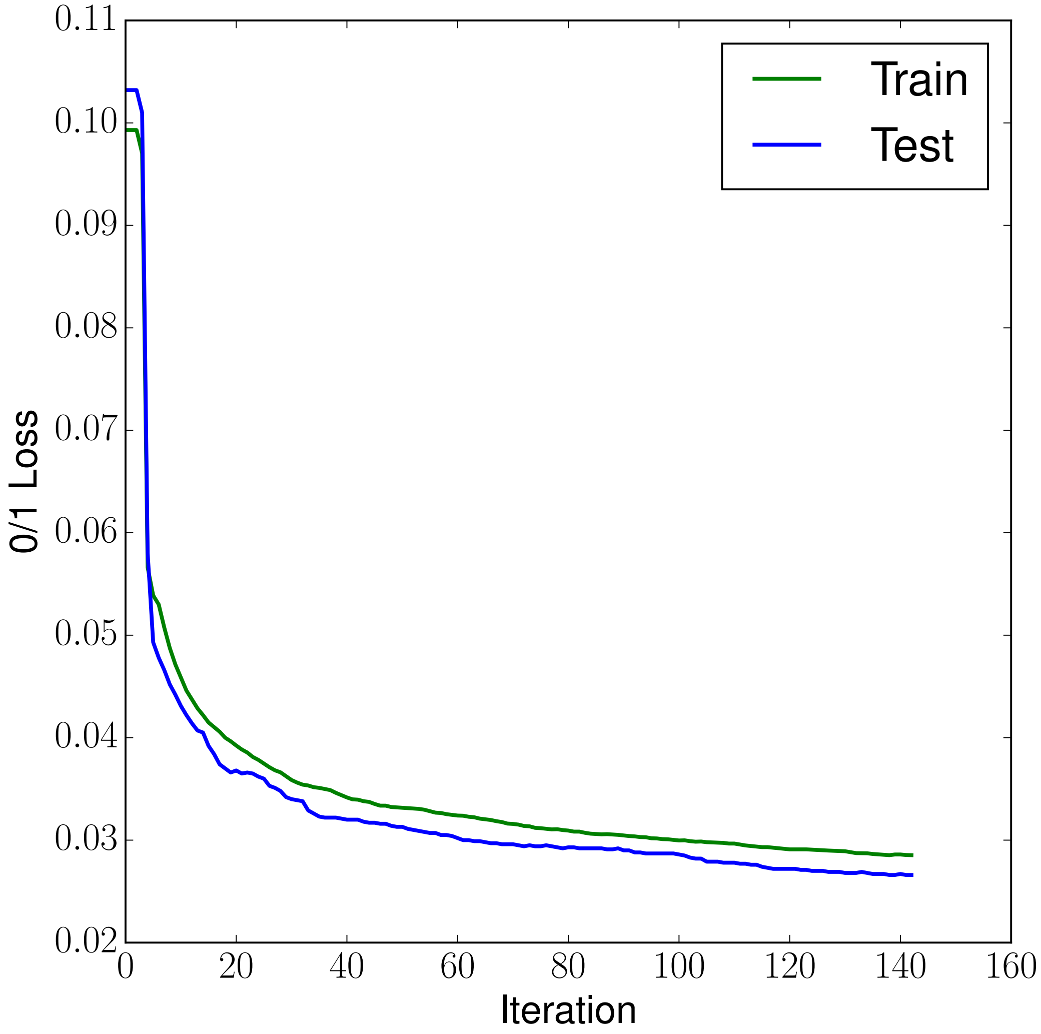

FIGURE 2. 0/1 loss as a function of iteration for both the MNIST training and testing datasets for binary logistic regression. The 0/1 loss on the testing set was evaluated using the model parameters computed by fitting on the training set for each iteration. Note how the loss decreases for both sets with each iteration as the fit improves. In general, the testing loss is slightly larger than the training loss as expected.

The log-loss for both sets decreases dramatically during early iterations but then slowly approaches convergence. The 0/1 loss, on the other hand, does not decrease as dramatically but instead gradually improves until the gradient descent algorithm has convergence. Once converged, I find my model performs question well and appreciably better than my linear binary classifier from Homework 1 as anticipated.

2.1.3. Given how the sigmoid binary logistic regression link function maps a prediction to a probability-like range $[0, 1]$, a natural threshold for classification is to say if $\hat{y} > 0.5$, then $Y = 1$ and 0 otherwise. With this threshold, I found a training log loss and 0/1 loss of 0.206 and 0.027, respectively. I found a testing log loss and 0/1 loss of 0.213 and 0.027, respectively. Note that I normalized both losses by the size of each respective set for comparison's sake. Overall, my model performs very well at is can successfully classify $\sim$97% of 2s in the MNIST dataset

2.2: Softmax classification: gradient descent. In this section I used multi-class softmax classification optimized using batch gradient descent to predict the label of a sample from the MNIST dataset. For k classes, the labels range over $[0, k-1]$ where for $Y > 0$

$$P(Y = l|x, w) = \frac{\exp(w_{0,l} + w^l \cdot x)}{1 + \sum_{i=1}^{k-1} \exp(w_{0,i} + w^i \cdot x)} \tag{6}$$

where I have added a constant offset w_0 term for each class for my calculations. Empirically, I find the model performs slightly better with the inclusion of w_0. For $Y = 0$,

$$P(Y = l|x, w) = \frac{1}{1 + \sum_{i=1}^{k-1} \exp(w_{0,i} + w^i \cdot x)} \tag{7}$$

where the probabilities sum to 1. For classification, I take the largest probability among the coefficients of the prediction vector. The log-loss for softmax classification is given by

$$L(w) = \frac{-1}{N} \sum_{i=1}^{N} \log(P(Y = y^i|x^i, w)) \tag{8}$$

for N samples. Note that this sum is over the entire prediction vector and not just the predicted probability corresponding to the correct label. When optimizing my model to find w_0 and $\hat{w}$, I seek to minimize Eqn. 8.

2.2.1. Here I take the derivative of the softmax function with respect to w_p for the lth class. Note that in this case, there are two separate regimes: where $l = p$ and $l \neq p$ where l is the true class for the jth sample. Also, I define the indicator function as

$$I(y^j, l) \tag{9}$$

which returns 1 if $y^j = l$ and 0 otherwise.

First, I expand Eqn. 8 and neglect the regularization term for now as it adds a λw_l to the gradient. I find

$$\begin{aligned} NLL &= \sum_j \left[\log \left[\frac{\exp\left(w_{0,l} + w^l \cdot x^j\right)}{1 + \sum_{l=1}^{k-1} \exp\left(w_{0,l} + w^i \cdot x^j\right)} \right] \right] \\ &= \sum_j \left[(w_{0,l} + w^l \cdot x^j) - \log[1 + \sum_{l=1}^{k-1} \exp\left(w_{0,l} + w^l \cdot x^j\right)] \right] \end{aligned} \tag{10}$$

via log identities where this quantities is summed over all j samples.

Now, I take the gradient of Eqn. 10 for $Y \neq 0$:

$$\begin{aligned} &\frac{\partial}{\partial w_p} \left(\sum_j \left[(w_{0,l} + w^l \cdot x^j) - \log[1 + \sum_{l=1}^{k-1} \exp\left(w_{0,l} + w^l \cdot x^j\right)] \right] \right) \\ &= \sum_j \sum_{l \neq p} I(y^j = l) \left(\frac{-x^j \exp(w_{0,p} + w_p \cdot x^j)}{1 + \sum_l \exp(w_{0,l} + w_l \cdot x^j)} \right) \\ &+ I(y^j = p) \left(x^j - \frac{x^j \exp(w_{0,p} + w_p \cdot x^j)}{1 + \sum_l \exp(w_{0,l} + w_l \cdot x^j)} \right) \end{aligned} \tag{11}$$

where the two terms account for cases when the true label $l \neq p$ and when $l = p$ where p is the current class whose derivative I evaluate. Replacing terms with conditional probabilities yield

$$\sum_j \sum_{l \neq p} -I(y^j = l) x^j P(y^j = p | x^j, w) + I(y^j = p) x^j \left[1 - P(y^j = p | x^j, w)\right] \tag{12}$$

which is similar to the binary logistic regression case but with additional terms for all incorrectly labeled terms in the prediction vector.

Now the gradient for the $Y = 0$ term proceeds similarly but without the $w_{0,l} + w^l \cdot x^j$ term as goes as follows:

$$\begin{aligned} &\frac{\partial}{\partial w_p} \left(\sum_j \left[-\log[1 + \sum_{l=1}^{k-1} \exp\left(w_{0,l} + w^l \cdot x^j\right)] \right] \right) \\ &= \sum_j \sum_{l \neq p} I(y^j = l) \left(\frac{-x^j \exp(w_{0,p} + w_p \cdot x^j)}{1 + \sum_l \exp(w_{0,l} + w_l \cdot x^j)} \right) \\ &+ I(y^j = p) \left(-\frac{x^j \exp(w_{0,p} + w_p \cdot x^j)}{1 + \sum_l \exp(w_{0,l} + w_l \cdot x^j)} \right). \end{aligned} \tag{13}$$

Replacing with conditional probabilities and simplifying gives

$$\sum_j \sum_{l \neq p} -I(y^j = l) x^j P(y^j = p | x^j, w) + I(y^j = p) x^j \left[-P(y^j = p | x^j, w)\right] \tag{14}$$

The gradient derived above becomes the gradient for my regularized gradient descent update rule

$$w_l^{(t+1)} \leftarrow w_l^{(t)} - \eta(\lambda w_l^{(t)} + \nabla_l NLL(w_l)). \tag{15}$$

In practice, I convert my $n \times 1$ label vector y into $n \times k$ label matrix for k classes. For the jth sample y_j, the element corresponding to the correct label is 1 while all other elements are 0. Naturally, my w becomes a $d \times k$ matrix and w_0 becomes a $k \times 1$ vector. Inserting my derived gradients into the gradient descent update rule, I update w and w_0 as follows

$$w_l^{(t+1)} \leftarrow w_l^{(t)} - \eta(\lambda w_l^{(t)} - \sum_j x_i^j [y^j - \hat{P}(Y^j = l | x^j, w_l^{(t)})]) \tag{16}$$

and for the unregularized constant offset term

$$w_{0,l}^{(t+1)} \leftarrow w_{0,l}^{(t)} - \eta(-\sum_j [y^j - \hat{P}(Y^j = l | x^j, w^{(t)})]). \tag{17}$$

2.2.2. TODO... learning rate and λ optimization

2.2.3. In Fig. 3 and Fig. 4, I plot the log-loss and 0/1 loss as a function of iteration for my softmax logistic regression batch gradient descent solution. Initially in the first two or so iterations, my solution moves away from the local minimum, likely because the initial step size η was a bit too large. As the iteration number increased and eta decayed as $1/\sqrt{t}$ for iteration number t, the decreased step size allowed the algorithm to advance towards to local minimum. Eventually around iteration 15, the solution reached close to the local minimum and slowly converged from there. Note that for this question, I said my solution was converged once the log-loss changed by less than 0.5% between iterations. A stricter converged criterion did not appreciably improve final losses and only slowed down the code. Interestingly, I found lower losses on my testing set indicated that my model performed marginally better on my testing set than the training set.

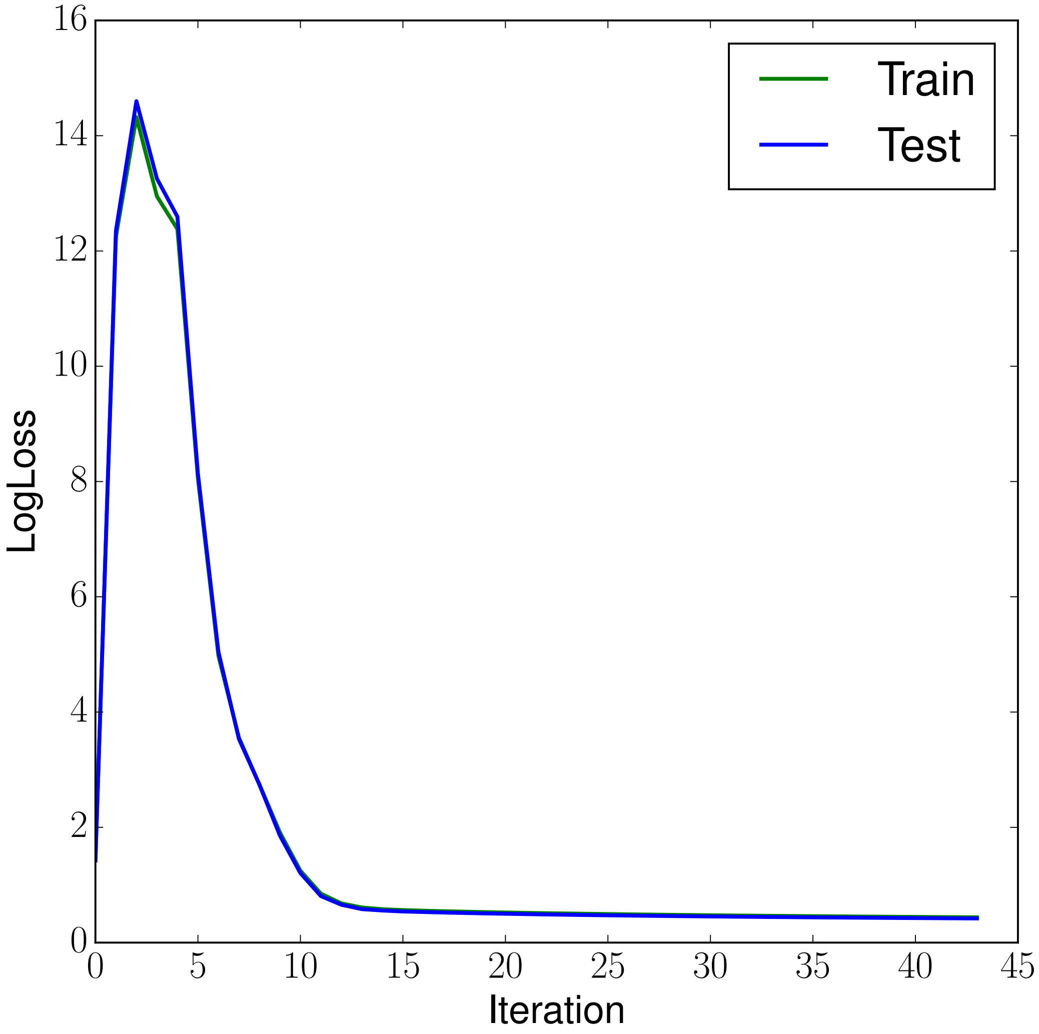

Figure 3. Log loss as a function of iteration for both the MNIST training and testing datasets for softmax logistic regression using batch gradient descent. The log loss on the testing set was evaluated using the model parameters computed by fitting on the training set for each iteration. The log-losses for both the training and testing sets are very close and hence their evolution appear to overlap in the figure. Note how the loss decreases for both sets with each iteration as the fit improves.

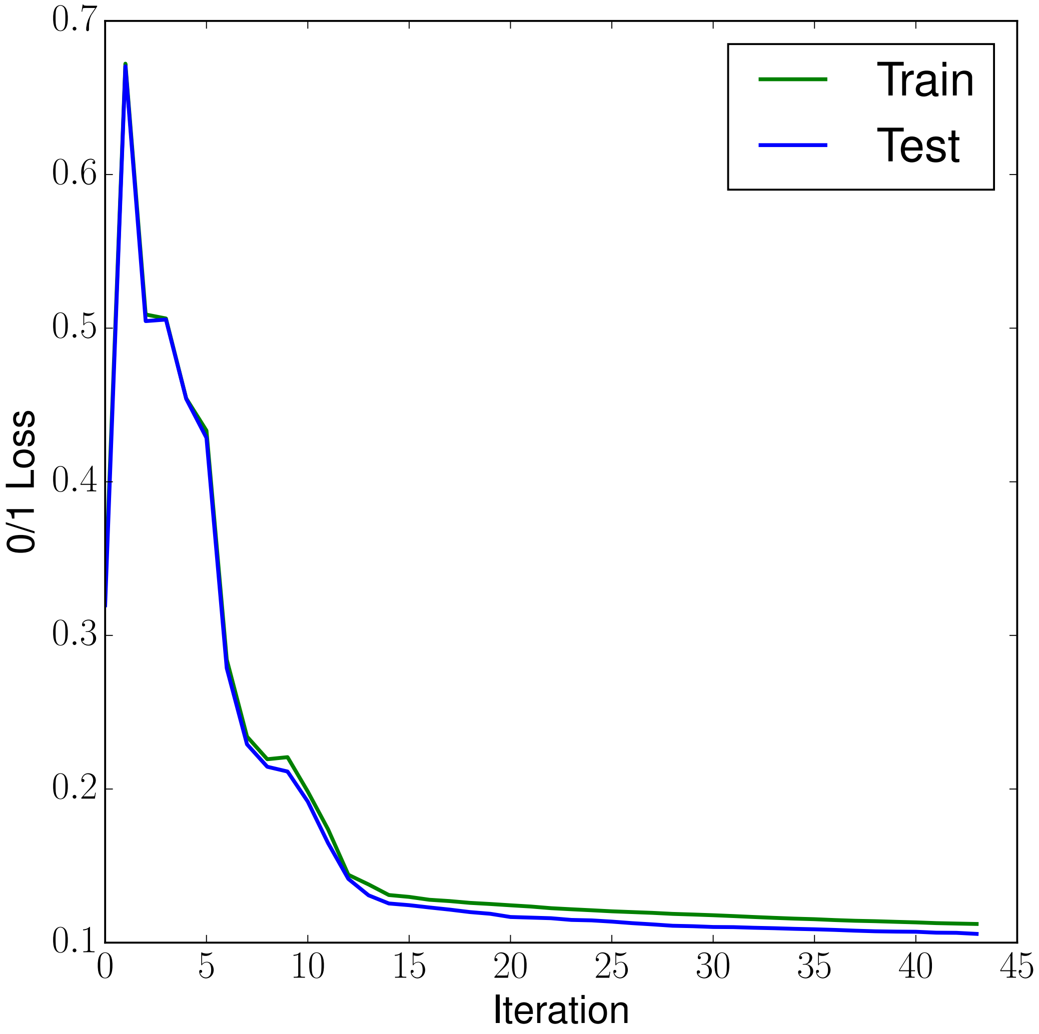

Figure 4. 0/1 loss as a function of iteration for both the MNIST training and testing datasets for softmax logistic regression using batch gradient descent. The 0/1 loss on the testing set was evaluated using the model parameters computed by fitting on the training set for each iteration. Note how the loss decreases for both sets with each iteration as the fit improves.

2.2.4. On the training set, I achieved a log-loss and a 0/1 loss of 0.439 and 0.112, respectively. On the testing set, I achieved a final log-loss and a 0/1 loss of 0.419, 0.106, respectively. My simple batch gradient descent solution for softmax logistic regression performs rather well as it can correctly classify ∼90% of digits in the MNIST dataset.

2.3: Softmax classification: stochastic gradient descent.