**New Perspective on MNIST Dataset: Regression and Sparsity**

Elise Bishoff

University of Washington

Department of Applied Mathematics

1410 NE Campus Parkway, Seattle WA 98105

**Abstract**

In this paper we dive into the effects of using different solvers on the MNIST dataset. We dive into the effects of using ridge and lasso regularization on the mapping from the image space to the label space. Then, we focus specifically on the lasso linear regression model to determine and rank which pixels in the MNIST set are most informative in correctly labeling the digits by looking at the model on a whole and by looking at the mappings for specific digits. Overall, we find that if we include 144 of the top pixels from the lasso model with we get an 81.72% accuracy on the test dataset.

**Sec. I. Introduction and Overview**

The MNIST dataset is a classical introduction to the classification problem within machine learning. It is a dataset of 70,000 handwritten digits sized at pixels with labels of what the handwritten digits represent. For this particular problem we use different linear regression to create a mapping from the image space to our label space in the framework of where is our mapping.

In this paper we first discuss the necessary theoretical background needed to properly understand the analysis we did on this problem. Then, we discuss the algorithm development and implementation while referencing our Python code. Then we dive into our discussion on the analysis we performed on our MNIST dataset. We conclude with a short summary and conclusion.

**Sec. II. Theoretical Background**

The most basic regression technique would be curve fitting by finding a mapping from the input space to our output space. Finding this mapping can be found by solving the well-known linear system

In our particular problem with the MNIST dataset our is our feature space or our pixel space of where is the number of samples. Thus, we have the pixel value normalized between 0 and 1 for each sample for each pixel value. Then is our label space that we have one hot encoded. Thus, since we have 10 digits where every value is nonzero expect the number it represents is 1. Then is our mapping space that maps our feature space to our label space. It is where each column represents the coefficient values for each pixel in the feature space. This is why when we solve Eq 2.1 we are finding the mapping between the feature space and our label space.

Different optimization techniques are used to help us find the best model by deterring from overfitting and promoting unique solutions when dealing with overdetermined and underdetermined systems. Usually, our problem with overdetermined systems is that we have no solutions to our problem in Eq. 2.1. For the MNIST problem we are dealing with an overdetermined system. Also, our other problem with underdetermined systems is that we have infinite solutions to our problem in Eq 2.1. To overcome this issue is to apply constraints or regularization to our Eq 2.1. The constraints for overdetermined and underdetermined optimization problems for linear systems are given by Eq. 2.2 and Eq. 2.3 respectively.

In particular we can make any constraint we want. In practice we usually have minimizing the and norms in Eq 2.2 for our particular problem at hand. Each has a distinct impact on the optimal solutions achieved. The norm promotes sparsity and increases the number of terms in the matrix. The norm promotes giving every entry a nonzero entry, but where some entries are relatively very small. In the problem we also use elastic net which tries to balance between the effects of the and norms.

**Sec. III. Algorithm Implementation and Development**

Overall process included loading the dataset, testing different models and analyzing the different effects of choosing only a few features for our linear regression model with lasso regularization.

In **Section 1** we loaded the MNIST dataset. We created a training set of 55,000 examples, a validation set of 5,000 examples, and a test set of 10,000 examples. I made sure to include a validation set for parameter tuning instead of tuning on the test set.

In **Section 2** we use various solvers to determine a mapping from the image space to the label space. We use the different functions in the sklearn package in Python. In **Section 2.1** we created a function that calculates the accuracy score from the predicted values to the actual labels. I converted our prediction to one hot encoding to make it possible to compare labels since our predicted output estimates a value for each digit. We interpreted that the largest value in the array is the predicted value. Then, afterwards we find the accuracy on the validation dataset for the four different methods used. In **Section 2.2** we have the visual representations for the mappings from the image space to the label space for each digit and each method. These can be seen in Figure 4.1.

In **Section 3** we have two uniquely defined functions. The function plot\_coef, plots the different summed coefficient values on a bar graph. The function max\_coef\_x returns the original mapping with only the highest summed absolute value of coefficients.

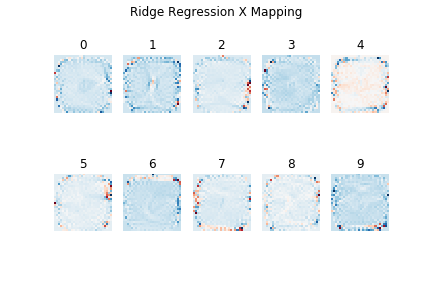
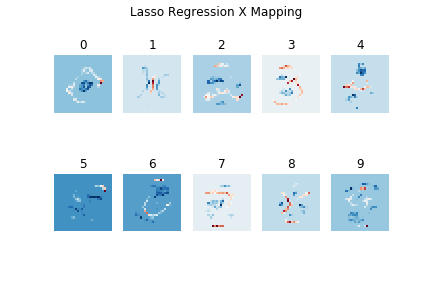
In **Section 4**, we take a deep look into linear regression with lasso regularization since lasso promotes sparsity. First, we tried to tune the parameter and found the best value was with . This seemed strange at first since a lower meant our model was converging more towards a linear regression model which we wanted to avoid. But, even with an we still promoted sparsity and only had 292 out of 784 nonzero coefficient values. Thus, we still had a sparse solution.

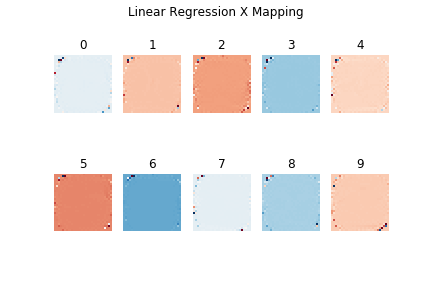
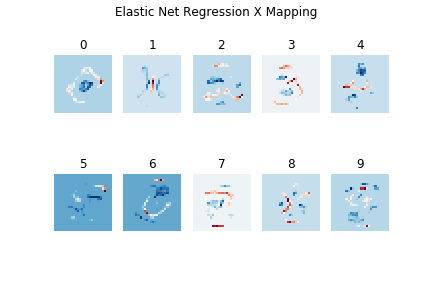
In **Section 4.2** we created visuals to see the impact of choosing only a certain amount of the features instead of looking at all 784 pixels in the model. We look at the top 144 features in the original mapping space and also in a condensed space of just the 144 features. I did this to sanity check that we would get better accuracy if we made a new model with our new features instead of including the original coefficient values from our original mapping. It made quite a difference from 62% accuracy on the validation dataset with the 144 features on the original mapping versus 81% accuracy on the validation dataset with the 144 features on a new mapping space.

In **Section 5** we conducted much of the same analysis in **Section 4**, but by taking to top pixels form individual digits instead of the whole model. In **Section 6** we calculated the test accuracy with our final model.

**Sec. IV. Computational Results**

First, I determined various mappings from the image space to the label space with our MNIST dataset. For this I focused on four different models, ordinary least squares linear regression, constrained linear regression (Lasso regression), constrained linear regression (Ridge regression), and a combined and constrained linear regression (Elastic Net regression). These were all ways to solve our problem. The main takeaway I got from focusing on the mapping space for each method was that each method promotes different reactions to the mapping space.





*Figure 4.1 Here we have the four different mapping spaces for each digit for the different regression methods.*

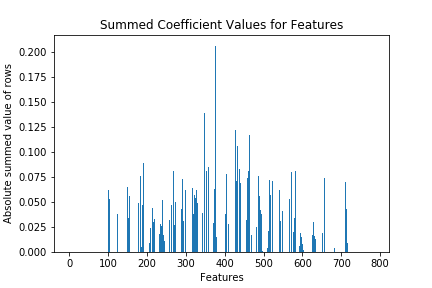
In **Figure 4.1** we see the different mappings for each digit for the different methods. As we can see from the figure, Lasso and Elastic Net mapping seems to have distinct pixels that either look like the digit or important blank spaces for the digit. For example, the digit 0 has positive values for the zero like shape and negative values for the pixels in the middle. While I was expecting something like this to happen, I was still surprised that for the most part the mappings look so similar to the digits. The Ridge mapping seems to take into account more of the edge pixels. For this mapping you can still see somewhat of a number outline, but it isn’t as distinct as the Lasso and Elastic Net. Then, the linear regression seems to give each pixel a part in determining the digit value and thus we don’t see any real distinct patterns to the feature space.

I found these results interesting and somewhat surprising. I expected all the mappings to look somewhat like the digits, but some mappings do more than others. Then, I thought this would have a significant impact on the accuracy on the validation set, but it seems to be more complicated than that. In **Figure 4.2** we have a table of the different accuracy scores for the different methods. For the most part the accuracies are comparable, but initially I assumed that the mappings that looked more distinctly like numbers would have a higher overall accuracy, but in fact this wasn’t the case. Linear Regression performed higher than Lasso and Elastic Net and very similarly to Ridge, but its features space didn’t seem to have any mapping space that looked like digits. Same with Ridge Regression.

|  |  |
| --- | --- |
| Method | Accuracy Score on Validation Set |
| Ridge | 85.7% |
| Lasso | 81.78% |
| Elastic Net | 83.64% |
| Linear Regression | 85.68% |

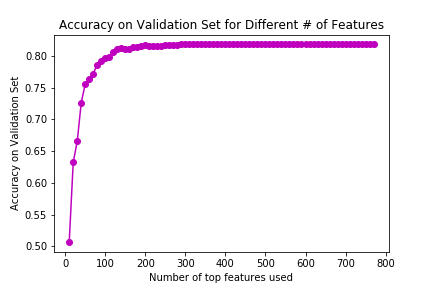
*Figure 4.2 Accuracy Scores on Validation Set for the different methods used.*

The next task I focused on was just focusing on a single model that promoted sparsity and discovering which pixels were most informative for correctly labeling digits. I chose the Lasso Regression model with since it promotes sparsity by setting feature values to zero if they do not play an important role in determining the digit value. In **Figure 4.3** we take a look at the different feature values. The heuristic I used for determining the top features was quite simple. Our mapping is a matrix of the size , but we only want to look at 784 features for the whole dataset instead of . So, I summed the absolute values of all the rows to get 784 features. This is what is shown in **Figure 4.3**. Then I picked the top values by picking the largest positive summed coefficient values.



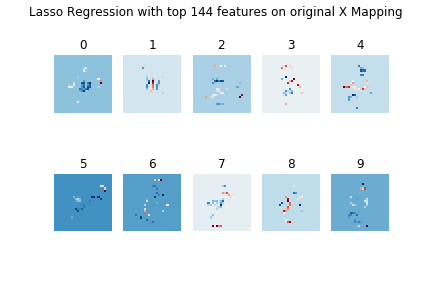
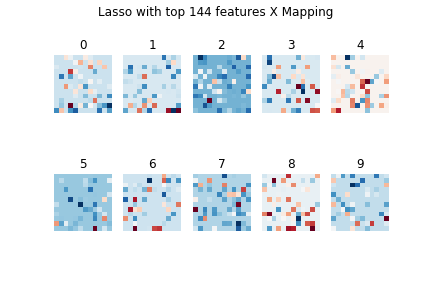
*Figure 4.3 Coefficient Values for Features in Lasso Model.*

From **Figure 4.3** we can see there are quite a bit of features that have a zero value. This is the main takeaway from this graph that Lasso does indeed promote sparsity. In fact, there are features that are zero. That means only 292 features have some value real value. This ties into **Figure 4.4** below. In this figure we plotted the number of features used in the model and the accuracy produced on the validation set if we used just these features. We see that once we hit 292 features the accuracy flattens. Also, this figure gives us some insight into what happens if we just use the top 10 features versus the top 100 features. We see that with only 10 features we get 50% accuracy whereas top 100 jumps to getting 79.68% accuracy. Also, there seems to be some threshold value where we stop gaining huge benefits from adding anymore features.



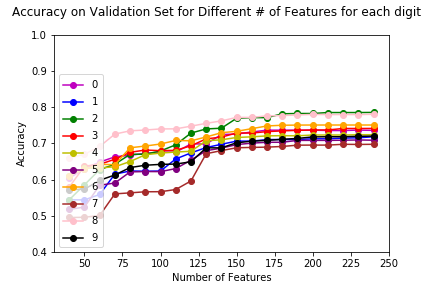
*Figure 4.4 Accuracy for different number of features used in Lasso Model.*

With **Figure 4.4**, I determined the most information features of the system existing in the top 144 features from Lasso. I used the heuristic that I wanted to get over 80% accuracy and have the number have a perfect square root so I could analyze the data in a more realistic image space. I still get an 81.14% accuracy on the validation set with only 144 features which is only 0.64% off from the full model. We could have chosen less features depending on our purpose of our feature reduction.



*Figure 4.5 Mapping of each digit with only the top 144 features. (Left) Lasso with new mapping with only our 144 features. (Right) Lasso with our new features projected onto our original mapping space.*

In **Figure 4.5** get a look into the different mappings based off our reduced feature space to 144 features. As we can see our feature space no longer look like digits on the left. This makes sense since we picked the top 144 features and then created a dataset from just these features. The right image depicts are new top 144 features projected onto the original mapping to determine which pixels were actually picked from the original mapping. This can be compared to the Lasso Regression model in **Figure 4.1**. When these two models are compared, we can see how the mapping in **Figure 4.4** right has most of the important features with a few distinct missing features from the original mapping space.



*Figure 4.6 The accuracy on the validation set for different number of features.*

Then, the next task I conducted was analyzing the accuracy on the validation set given that we only extracted the most important pixels from an individual digit. In **Figure 4.6** we have the results of this for each digit for different number of features included. Surprisingly, we got the accuracy up to 70% on the total validation set even though we were only using useful information from one digit. Overall, this surprised me that the most useful digits from one digit still was able to beneficial in predicting other digits.

Lastly, after the analysis we found the accuracy on the test dataset. With our final model of Lasso Regression with with only the top 144 features we get a test accuracy of 81.72%. This was great considering it is comparable to our accuracy on the validation dataset and the training dataset.

**Sec. V. Summary and Conclusion**

Overall, we were able to see some interesting insights about different regularizations on the linear mapping form image space to label space. Also, as we focused on the lasso model, we saw that the important features seemed to be where the points were representative of the digit or where the digit consistently did not appear in the image space. If we just focused on the important pixels digit wise, then we saw lower accuracy levels, but still surprisingly high accuracy given that we are only using the features that were important to a single digit. Lastly, we came to better understand the problem for our overdetermined system and the impact of using different regularization terms.

**Appendix A Python functions used and brief implementation explanation**

Tensorflow: This package is for neural networks, but I used it only to download the MNIST dataset.

Matplotlib.pyplot This package is to make plots look like MATLAB plots.

*Numpy:* This packages stands for Numerical Python. It is a scientific computing library.

*Pandas:* This package allows us to have structed datasets.

*Sklearn:* This package is a machine learning package. It has lots of tools available for data analysis.

* *Accracy\_score:* Gives us the accuracy score of our model with the given labels.
* *Linear\_model:* This holds many linear models and their methods. Used especially to make a linear regression model, linear regression with lasso, linear regression with elastic net, and linear regression with ridge.
* *Lasso():* The objective for Lasso is linear regression, but with regularization to promote sparsity.
* *Ridge():*The objective for Ridge is linear regression, but with regularization to give each feature a say in the mapping, but where most terms tend to 0.
* *Elastic\_Net():*The optimization objective for Elastic Net is linear regression, but with a balance between the effects of and regularization.
* *Linear\_regression():* ordinary least squares linear regression.
* *OneHotEncoder:* This turned our labels into one hot encoding.

*Accuracy(y\_predicted, y\_true):* Function that I created. It takes in predicted y values and actual y values to calculate the classification accuracy.

*Plot\_coef(model, x\_axis):* Function that I created. It takes in the model and the x-axis values to plot a bar graph of the summed absolute values of the coefficients.

*Max\_coef\_x(model, top):* Function that I created. It takes the top number of coefficients values and only includes the top pixel values in the original mapping.

**Appendix B Python code**