AMATH 563 HW 1
Classifying Digits of Handwritten Numbers

Katie Johnston

April 21, 2020

Abstract

In this project, we explore the MNIST data set, a set of handwritten numbers. We used a variety of regression models to classify the digits from pixel space. We then used Lasso with increasing alpha to promote sparsity and observed that the most important pixels are near the middle and the accuracy decreases exponentially as the number of nonzero terms decreases. These behaviors were also observed when analyzing each digit individually, and we also saw that the patterns of the most important pixels were different for the different digits.

1 Introduction and Overview

Regression is one of the most basic methods of machine learning. Regression can be applied to a large variety of problems that can be reduced to an $\mathbf{Ax} = \mathbf{b}$ problem on linear equations, and there are a variety of variation of regression solvers for solving this type of systems including Least Squares and Lasso. In this paper, we examine how we can use regression to classify handwritten numbers from pixel space to the digits 0-9. We will be examining the MNIST data set which includes images of 70,000 total handwritten digits and their labels.

While exploring this dataset, we also explore how promoting sparsity in a model affects the results. Sparsity is an important topic in regression to be able make as simple a model as possible in order to generalize models to unseen data sets. In this paper, we examine how sparsity affects the outcomes when classifying all digits together as well as classify each digit individually.

This paper will start with discussing the theoretical background of the regression methods used and important topics related to regression. Next, will be an overview of the algorithms used, and then a description of the computational results observed with many accompanying graphs.

2 Theoretical Background

We want to be able to classify handwritten digits from pixel space to the digits 0-9. Our problem comes down to solving an $\mathbf{Ax} = \mathbf{b}$ system of equations. If $\mathbf{A}$ were a square matrix, we would have one solution to this system. However, because there are many more data points that variables (the number of pixels), the system is overdetermined and does not have an exact solution. Thus, in order to find a solution to this system, we introduce a constraint to optimize our solution in order to find the "best" solution to our problem.

There are a large amounts of methods to regularize to solve a an overdetermined $\mathbf{Ax} = \mathbf{b}$ system. One of the most common is least squares regression, which minimizes the 2-norm, $||\mathbf{Ax} - \mathbf{b}||_2$. Another simpler method which is very similar to least squares is using the pseudo inverse, which is given by $\mathbf{A}^\dagger = (A*A)^{-1}A^*$, and acts as an inverse of $\mathbf{A}$, so we can use matrix multiplication to solve $\mathbf{x} = \mathbf{A}^\dagger\mathbf{b}$.

We want to find the "best" model for our data. "Best" is a broad word that is not well defined for this situation. The "best" model is the one which generalizes to unseen data with the lowest amount of error. For our problem, our error measure is the accuracy which the model predicts the classified digits. Thus, the model which can classify the highest percentage of unseen data points will be determined the "best" model. This is where the idea of overfitting comes into play. We can train our data with as many terms as we like, and eventually we will be able to perfectly predict our data with zero error. However, if we take this complicated model with many terms, our model will probably not predict new data very well. Thus, we

need to cross-validate with data that we did not train our model on. The ideal model will have the highest accuracy on this validating data set.

Another factor to consider is the number of terms in our model. Based on the work of Pareto, parsimony is an important component in modeling. The Pareto Frontier is defined by the models with the lowest error for a given by number of terms. The "best" models will be along this line. To choose the "best" model on this line we want to use cross validation, which we will do by comparing the error on a testing data set that was not used for training. We also want to choose a model with as minimal terms as possible, while still having low error, because models with fewer terms typically generalize better to new data.

There are also methods to help promote sparsity by adding a penalty term to the constraint that we are optimizing on. One of these methods is Lasso, which adds a ℓ_1 penalty term to the constraint. Lasso is given by

$$\mathbf{x} = \underset{\mathbf{x}'}{argmin} ||\mathbf{A}\mathbf{x}' - \mathbf{b}||_2 + \alpha||\mathbf{x}||_1$$

This ℓ_1 penalty promotes sparsity because each nonzero term will add to the term, and we want to find the lowest possible. Thus, a term will only be nonzero if the term lowers the ℓ_2 difference more than the term will raise it. The larger the value of α the more penalty is given to the terms, and the less nonzero terms the model will have. Thus, least squares has the same constraint as lasso with $\alpha = 0$.

Ridge regression is similar to lasso because it also has a penalty term, but ridge has an ℓ_2 penalty term. Thus, $\mathbf{x} = \underset{\mathbf{x}'}{argmin} ||\mathbf{A}\mathbf{x}' - \mathbf{b}||_2 + \alpha||\mathbf{x}||_2$. The penalty term causes many terms to be small. However, they do not cause them to be zero, just small. Thus, Ridge will not be a good method when we want to support sparsity.

3 Algorithm Implementation and Development

First, we set up an $\mathbf{A}\mathbf{x} = \mathbf{b}$ system. Our $\mathbf{A}$ matrix is our data matrix of images and our $\mathbf{b}$ matrix is our label data. Our $\mathbf{A}$ matrix will be size $(60000, 784)$ and our $\mathbf{b}$ matrix will be size $(60000, 10)$. In our $\mathbf{A}$ matrix, each column represents one image that is reshaped into a column. Because we have categorical data, we will one-hot-encode our label data. This means that is the label is

$$\text{"1"} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \text{"2"} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad ,\cdots,\text{"9"} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \quad \text{"0"} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

After setting up our system, we solve our $\mathbf{A}\mathbf{x} = \mathbf{b}$ system using a variety of methods from different python libraries. I used least squares regression from numpy, the pseudo inverse, lasso with two different alpha values (0.1 and 1), ridge, and ElasticNet. Lasso, Ridge, and ElasticNet are all from sklearn.

To compare these models to each other, I found the accuracy on the test data. The resulting $\mathbf{x}$ matrix is size (784, 10), and each row gave a vector of numbers, where each element is less than 1, $|x_i| < 1$, for each pixel. In order to pick the "best" guess we pick the one which is the largest, and that was the choosen answer. To find the percent accurate, I compared the percentage of these that were correct from the given labels. The model with the highest accuracy on the testing data set was determined to be the "best" model because it predicted the most digits correctly on unseen data. We compared models on the testing set in order to validate the models and avoid overfitting.

Next, we wanted promote sparsity to answer a number of questions. To promote sparsity in the data set, I used Lasso with an increasing α value to limit the number of terms that were nonzero. I chose Lasso to promote sparsity because most of these simple models I tested could not easily be modified to promote sparsity and Lasso is very easy to change alpha to promote sparsity. I then compared how the number of terms affected the accuracy and which pixels were the most important. The accuracy was calculated the same as before when comparing different models. The importance of the pixels were determined by increasing alpha until only a small number of pixels were nonzero, and the pixels that were still nonzero

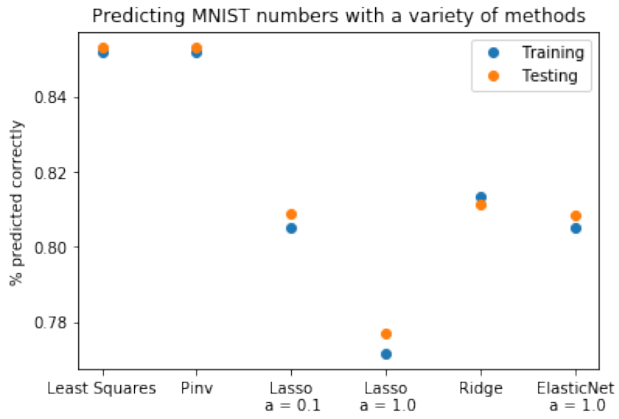


Figure 1: Comparing the error on a variety of methods. Observe that Least Squares and pinv have the highest accuracy while Lasso with a = 1 has the lowest accuracy. Also observe that the training and testing error is always very similar to each other. This implied that little to no overfitting is produced with these methods

were determined to be the most important. We mostly looked at the pattern which these important pixels formed as the number of nonzero terms decreased. In order to observe the patterns formed by the important samples, we used pcolor. Also because there were 10 elements in the vector which represented each pixel, the pcolor graph displayed the 2-norm of each vector.

We do not want to over fit our model to our data. In order to evaluate overfitting, We trained our model on the training set of 60,000 numbers, and then test the model on the 10,000 new data points given in the testing data set. We compared the percentage of correctly characterized numbers. If the testing error did not increase with the number of terms, then we have not yet reached overfitting. Overfitting was not a large concern for this problem, however, because we want to promote sparsity, and when we limit the number of terms, we typically increase our error (for both training and testing), and thus will not be overfitting.

We also considered each number individually to see which pixels were the most important in categorizing each individual number. The **A** matrix was the same as with the full model, but the **b** vector was a little different. **b** was a vector of length 60000, which contained a 1 if that data point is the number of interest, or will be 0 if it is any other number. The fitting was very similar to when we fit to all digits. I increased alpha until there was only one digit left, and this digit was determined to be the most important pixel for that digit. Similar to how we examined with all digits, w also examined the patterns that the important terms formed while the number of nonzero terms decreased.

4 Computational Results

First, we compared the error of our predicted as calculated in a variety of methods. The errors on both the training and testing sets can be seen in Fig (1). From this graph, you can see that the most accuracy model was least squares and pinv. You can also see that lasso with alpha = 1.0 is the least accurate. We also want to consider the number of nonzero terms from each method and how the error is affected by the number of terms. Fig (2) shows the accuracy of all six methods but sorted by number of terms. In this figure, it's a litte hard to see, but Least Squares and pinv have the same number of terms and accuracy as well as Lasso with alpha = 0.1 and ElasticNet with alpha = 1.0. From this graph, we can see that although Least Sqaures and pinv have the highest accuracy, they also have the largest number of terms, which is not ideal if we want a model with a small number of terms. We can also observe that while Lasso with a = 0.1, ElasticNet with a = 1, and Ridge have similar accuracy (within 1% of each other), Lasso with a = 0.1 and ElasticNet with a = 1 have much less number of nonzero terms.

Next, we want to promote sparsity and find the most important pixels. When we promote sparsity with lasso, you can see in Fig (3) how the weights of the pixels change as the number of nonzero terms decreases.

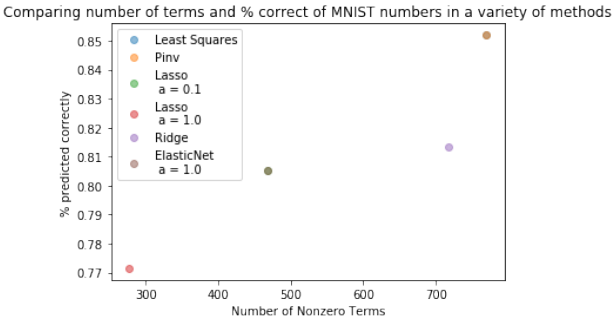


Figure 2: Comparing the error on a variety of methods while sorted by numver of nonzero terms. Observe that while Least Sqaures and pinv have the highest accuracy, they also have the largest number of terms, which is not ideal if we want a model with a small number of terms. Also observe that while Lasso with a = 0.1, ElasticNet with a = 1, and Ridge have similar accuracy (within 1% of each other), Lasso with a = 0.1 and ElasticNet with a = 1 have much less number of nonzero terms.

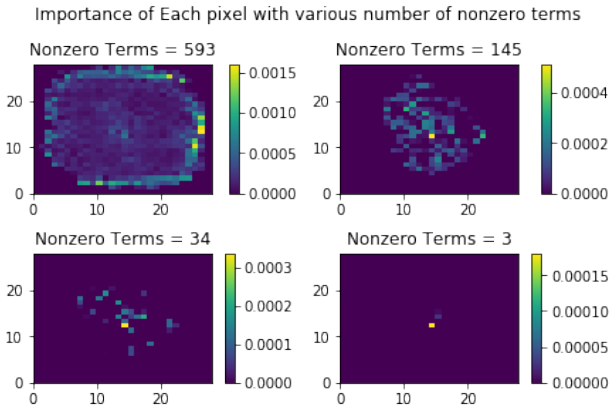


Figure 3: Pcolor map of different number of important samples. Each pixel represents a normed vector from the x matrix. Observe that as the number of nonzero terms decreases, the more center pixels are more important.

When the number of nonzero terms is high, the importance of the pixels form kind of a donut shape, with important pixels in the making a circle around the outside and some important pixels in the middle. As we decrease the number of pixels, the middle points become more and more important, but there is still a little bit of spread around. When there are very few nonzero terms, we see that the most pixel is the one in the middle (which is index number 350).

Next, we want to compare how the different number of nonzero terms affect the error of the output. Fig (4) shows the accuracy of the training and testing data for varying numbers of nonzero terms. From this graph, you can see that the accuracy increases exponentially as the number of nonzero terms increases. You can also see that the testing accuracy is always very close to the training accuracy. This shows that we are not overfitting because the training accuracy does not start decreasing as we increase the number of terms. It also shows that our model is generalizing in an expected manner. You can see on the low end, with only 3 nonzero terms that the accuracy is only 11%, which is because with a lower number of term, it is predicting all to be 1s.

Next, we found the most important pixel for each digit individually. Fig (5) shows a plot of all of the most important pixels. Each choose point was the most important pixel in at least one digit (two pixels were chosen twice). We can see that this plot looks similar pixel plot when comparing all the digits with a small number nonzero pixels.

We can also analyze what pixels are important as we decrease the number of nonzero pixels. In Fig (6a)

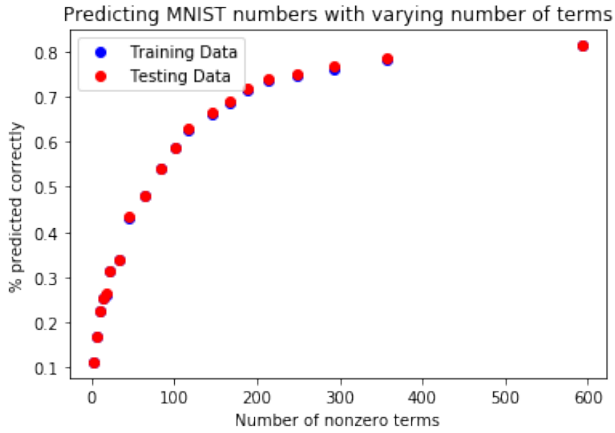


Figure 4: The percent accuracy based on the number of nonzero terms. The number of nonzero terms was made smaller by increasing alpha on Lasso. From this, you can see that the accuracy is increasing exponentially as the number of nonzero terms increases.

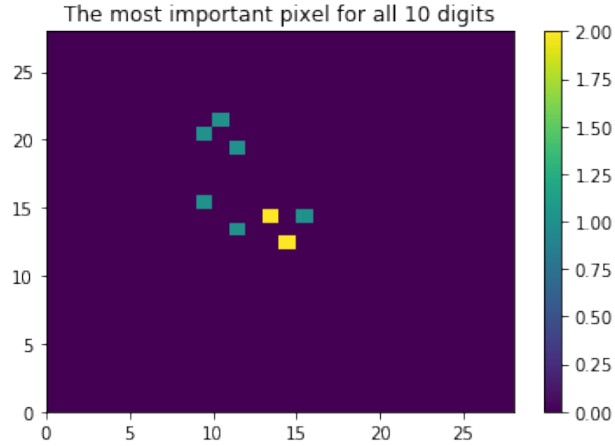
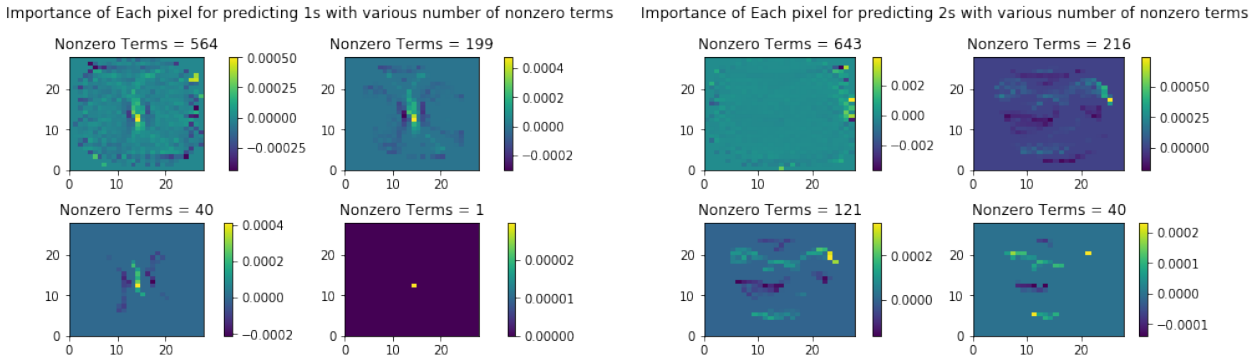


Figure 5: The most important digit of each digit individually. The most important digit was calculated by increasing alpha on Lasso until only 1 digit was nonzero. Observe how this pattern in similar to the pattern of a small number of nonzero terms when all digits are considered. Two of the pixels were chosen to be the most important by two different digits.

and (6b), we can observe the nonzero pixels as we decrease the number. We can see that in with a large number of nonzero pixels, there's a ring around the outside and some inner pixels that seems important. This behavior is similar to what we observed when we looked at all the digits. We can also see that as we decrease the number of nonzero terms, the pixels that are nonzero between 1 compared to 2 are different. For distinguishing 1s, you can see that the middle pixels are the most important, and a vertical line of important pixels are observed. This behavior makes sense because a 1 is tall and skinny. When we look at the behavior when just distinguishing 2s, we see that we get horizontal lines that get skinnier as we decrease the number of terms. This also makes sense because a 2s distinguishing factors from other numbers are the horizontal parts on the top and bottom.

We can also look at how well we are predicting each digit individually. Fig (7) shows how the percent predicted correctly changes as the number of nonzero terms change for the 1s digit. The other digits also have similar results, with a mostly flat region until it starts decaying exponentially as the number of nozero terms decrease. With a very lower number of nonzero terms, we get an accuracy about 90%. This is because we are predicting all zeros (not predicting the chosen digit) and approximately 9/10 of the data are other



(a) (b)

Figure 6: Important pixels of 1 (a) and 2 (b) with decreasing number of pixels. Obverse that these two digits produce different patterns as the number of nonzero digits decreases.

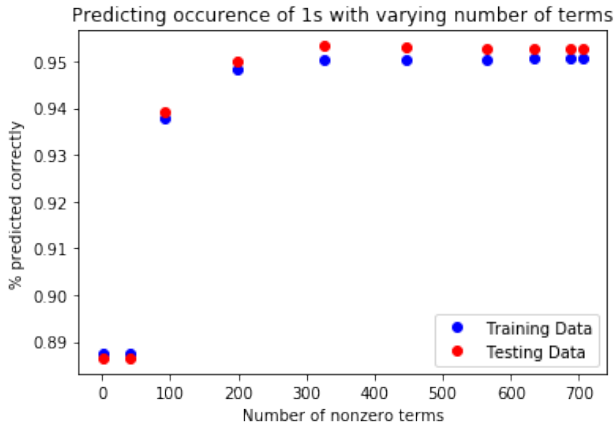


Figure 7: Percent accuracy of the 1s digit with varying amounts of nonzero terms. A very small number of nonzero terms will have about a 90% accuracy because about 90% of the data is nonzero. All digits had very similar behavior.

digits. We can also see that the accuracy on our testing data is very similar to the accuracy on our training data, usually even slightly higher for this example. This validates that we are not overfitting to our training data.

5 Summary and Conclusions

In this project we explored the MNIST dataset and discovered many interesting behaviors. We observed that Least Squares and using the Pseudo inverse produced the most accuracy predictions on our test set of our tested algorithms. These two algorithm had the same accuracy. We also observed that while they produced the best accuracy (even on the test set), they used the most number of terms.

Next, we used Lasso with increasing alpha to promote sparsity and observe the most important pixels. We discovered that the most important pixels are near the middle. We also observed that as we decrease the number of nonzero terms, our accuracy decreases exponentially. We also analyzed the individual digits. Similar to considering all the digits, we observed that the more important pixels are towards the center and the accuracy decreases exponentially as the number of nonzero terms decreases. We also observed how the patterns of the most important pixels vary with the different digits particularly between 1 and 2.

A Python functions used and brief implementation explanation

For the variety of regression models I used:

- `np.linalg.lstsq()` numpy's least squares solver
- `np.linalg.pinv()` numpy's puedo inverse and then numpy's matrix multiplication
- `sklearn.linear_model.Lasso()` sklearn's version of Lasso
- `sklearn.linear_model.Ridge()` sklearn's version of Ridge
- `sklearn.linear_model.ElasticNet()` sklearn's version of Elastic Net

Other important functions used:

- `pcolor` Used to make the pixel heat maps

B Python Code

Please see Github for code: `https://github.com/KatieJ16/AMATH563/tree/master/HW1`

Notebooks included:

- `HW1-Variety-methods` Contains code relevant to comparing different regression models to each other.
- `HW1-Lasso-all` Contains code relevant to promoting sparsity on all digits
- `HW1-just1` Contains code relevant to promoting sparsity on just one digit at a time