AMATH 563 Homework 1

Christopher Liu

April 17, 2020

Abstract

Regression is a powerful tool that allows us determine a relationship between dependent and independent variables. In this assignment we will be applying regression to classify handwritten digits from the MNIST data set. We will explore how different methods affect the accuracy of our classifier and cross-validation methods will be employed to select the best relationship or model. Despite, getting decent accuracy, our results suggest that our classifier is too simple for this problem. We will also promote sparsity in our model to determine which parts of a handwritten digit are the most important when attempting to classify all digits and individual digits.

1 Introduction and Overview

Regression in its simplest form can be thought of as fitting a curve to a set of points (or observations). A regression algorithm is able to produce the "best-fit" line by minimizing some error between the calculating curve and the observations and may include additional constraints. Once a regressional model has been constructed for a set of observations, we can use the model to both extrapolate and interpolate the behavior described by our observations.

In order to explore the effects of our choice of objective function and regularization on our regression model, we will employ the MNIST data set. The MNIST data set is a database of handwritten digits (0 to 9) which consists of a training set of 60000 images and a testing set with 10000 images, all with their corresponding correct labels. Each image is 28 by 28 pixels and has been normalized and centered.

Using regression, we will attempt to create a mapping from the image space to the label space and examine how mappings produced by different algorithms vary in accuracy. The regression will be done by using 4 solvers, the pseudo-inverse, lasso, ridge and elastic net. Once an acceptable mapping has been created using our training data, we can then apply it to our test data to classify the digits.

By promoting sparsity in our mapping, we can also attempt to determine which pixels are the most important in correctly identifying a handwritten digit. As datasets can be very large and computational power is a limited resource, we would like to determine what data is important when trying to classify our digits so that we can discard the rest. Finally, we will also explore the concept of k-fold cross-validation and how that affects our mapping on the withheld (or test) data.

2 Theoretical Background

2.1 Regression on an Over-Determined System

In its broadest sense, regression can be thought of as estimating the relationship between independent variables, X, and dependent variables, Y using a statistical method [1].

$$Y = f(X, \beta) \tag{1}$$

This relationship is given by the regression function, f, and its parameters, β. In this assignment, we will be performing regression by solving a linear system,

$$AX = B \tag{2}$$

where A are our independent variables, X is a mapping or relationship we wish to calculate and B are our dependent variables. Since A is usually not square, we instead look to solve the following optimization problem instead [1]

$$\underset{x}{\operatorname{argmin}}(\|AX - B\| + \lambda g(x)) = \tag{3}$$

for an over-determined system, where $\|AX - B\|$ is our objective function of interest and $g(x)$ is our regularization which is a constraint or penalty with a parameter λ. An over-determined system is one where there are significantly more observations than there are variables (a tall and skinny matrix A). It is important to note that there are no exact solutions to an over-determined system so an approximated solution is calculated instead. Our choice in regularization has a profound effect on our outputted model. For example, one can promote sparsity in X by letting $g(x) = \|x\|_1$ and increasing the value of the parameter λ [1]. This allows us to highlight the 'most important' entries in X as we will later see.

2.2 AX=B Solvers

Four different AX=B solvers will be employed. They are the Moore-Penrose pseudo-inverse, Lasso, Ridge regression and elastic net. All 4 solvers use the same mathematical framework as follows [1],

$$\hat{X} = \underset{X}{\operatorname{argmin}} \|AX - B\|_2 + \lambda_1 \|X\|_1 + \lambda_2 \|X\|_2 \tag{4}$$

and differ in their choice of the 2 parameters λ_1 and λ_2. For the pseudo-inverse, the algorithm solves for (4) $\lambda_1 = \lambda_2 = 0$. For Lasso, it solves with $\lambda_1 > 0$ and $\lambda_2 = 0$. For ridge, it solves with $\lambda_1 = 0$ and $\lambda_2 > 0$. Finally for elastic net, it solves with $\lambda_1 > 0$ and $\lambda_2 > 0$ [1].

2.3 K-Fold Cross-Validation

Cross-validation is an important step of regression in over-determined systems as it is possible for the model to over-fit the data. Over-fitting is when additional complexity is added to the model that reduces the error on the training data but increases error on the withheld testing set [1]. This commonly occurs when trying to fit high-dimensional data.

We can overcome overfitting by implementing a cross-validation strategy. One such strategy is k-fold cross validation. Rather than using the entire training to calculate the model, we instead randomly separate our training set into k sets. We then create a model from each kth set and take the average of the k models. This is then applied to our test data for further cross-validation.

3 Algorithm Implementation and Development

3.1 Loading Data and Setup

We first begin by loading the data and re-structuring it into the form $AX = B$. Using the MNIST library in Python, we can load the image data into a matrix A where each row of A contains the pixel intensity of each image (0 is white, 255 is black). Our labels, B, are column vectors which contain values from 0-9 which correspond to the handwritten digit. In order to improve accuracy in our regression model, we convert our labels to a matrix where each row is all 0s except for a 1 in the position corresponding to the correct digit. This results in A being $n \times 784$ and B being $n \times 10$ where n is the number of images in the set.

3.2 Fitting the Model and Testing

In order to fit our model, we will be using the pseudo-inverse function from the Numpy library and the Lasso, Ridge and Elastic Net functions from the Scikit Learn library. To carry out k-fold cross validation, we create a random permutation of numbers from 0 to 59999 and use that to randomly separate our training data into k-folds. The produced models from each fold are then averaged. We the model is fit for individual digits, our B is instead a column vector with a value of 1 when it is the digit of interest and 0 otherwise. For all the digits, we test our model by applying it to our training or testing data and choosing the value

closest to 1 in each row of our output, B, which is then compared with the provided correct labels. For the individual digits, since it is a column vector, we simply round our resulting B noting that 0.5 is rounded down to 0.

3.3 Sparsity and Ranking Pixels

To promote sparsity, we use Lasso and increase the parameter α (which corresponds to λ_1 in Equation 4) until we reach the desired sparsity which we define as the number of non-zero entries in X. We can determine which pixels are most important by taking the 2-norm of each row of X, which are the weights for a given pixel, and comparing the values. The larger the value, the more important the pixel. When this is done for individual digits, the absolute value of each element is taken instead. To test our model accuracy when only the N most important pixels are used, we reconstruct A such that the columns corresponding to not important pixels are omitted and solve $Ax = B$ for our truncated A.

4 Computational Results

4.1 Solving AX=B

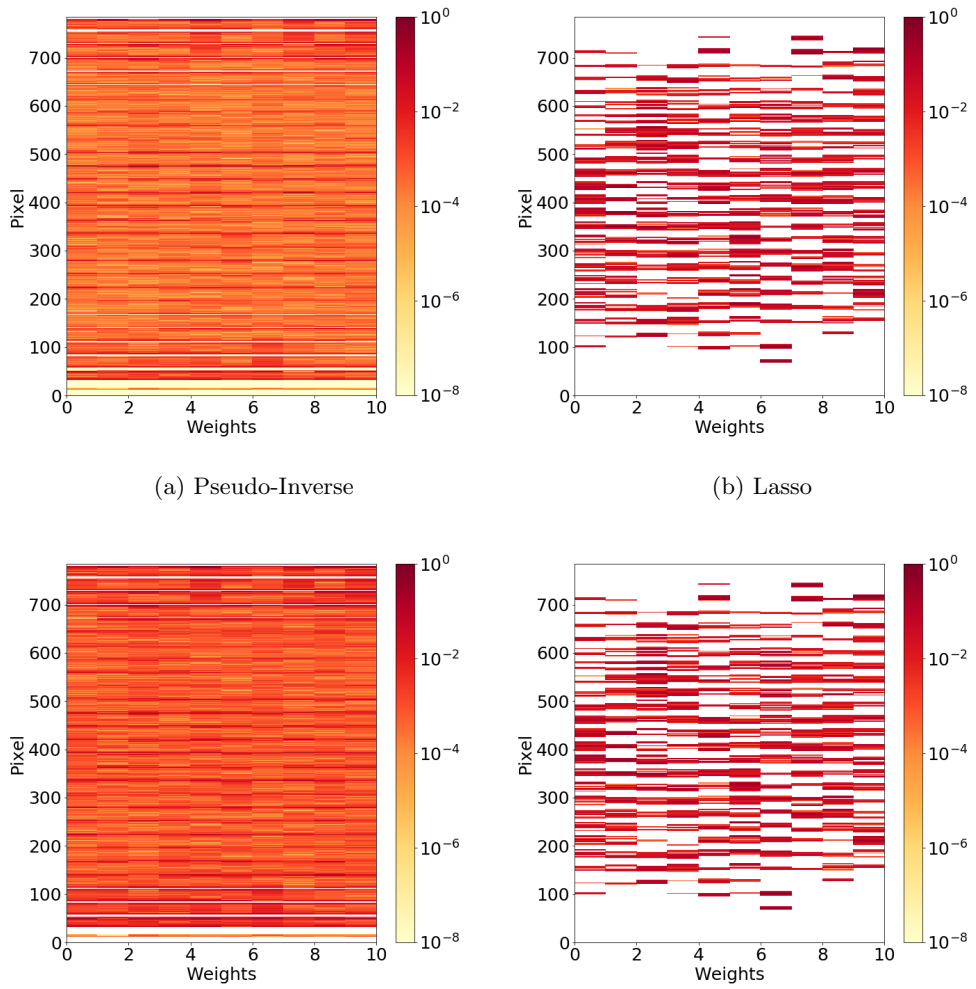
We begin by exploring 4 different Ax=B solvers and their ability to correctly classify the handwritten digits. The results from the testing is shown in Table 1. Training fit refers to how well the 6-fold model fits the training data while Test Accuracy refers to how well the model fits the test data. For Lasso and Ridge, a parameter of $\alpha = 0.5$ was used. For Elastic Net, $\alpha = 1$ and l1_ratio $= 0.5$ were used where α and l1_ratio control the values of λ_1 and λ_2. Note that the values of α were not chosen to produce the best accuracy and lowering α for Lasso and Elastic net led to better results but when $\alpha = 0$, then there is no regularization and they become similar to the pseudo-inverse. Rather, the values of α were chosen to promote sparsity and to highlight overfitting vs underfitting of the data.

The pseudo-inverse (linear regression) performed the best followed by Ridge. Elastic Net and Lasso, both of which have an L1 component in their regularization, were slightly worse and had identical performance. This suggests that our classifier is underfitting the data and is not complex enough. In the case of underfitting, removing model complexity, such as by promoting sparsity, will increase the error in both the training and testing data. Furthermore, we note that there is little difference between the 1-fold and 6-fold cross-validated models. Since we k-fold cross-validation is done to overcome potential overfitting, this also suggests that the model is not overfitting.

Solver	Training Fit	6-Fold Test Accuracy	1-Fold Test Accuracy
Pseudo-inverse	84.9%	85.1%	85.3%
Lasso	78.8%	79.3%	79.2%
Ridge	81.0%	80.9%	81.2%
Elastic Net	78.8%	79.3%	79.2%

Table 1: Performance of each solver

Normalized pseudo-color plots of the resulting X from each solver are shown in Figure 1. We notice that even for a low value of λ_1, we get a significantly sparser model. If we look at the plot for the pseudo-inverse, nearly all entries are non-zero despite having the best accuracy on the test data which suggests we are not overfitting.



(a) Pseudo-Inverse

(b) Lasso

(c) Ridge

(d) Elastic Net

Figure 1: Normalized pseudo-color plots of each solver's mapping

4.2 Promoting Sparsity and Pixel Ranking

To promote sparsity in our model, $\lambda_1 = 5$ was chosen to give 191 non-zero elements in the resulting matrix X which is 784×10. This is shown in Figure 2 which is a pseudo-color plot of X that has been normalized and plotted with a log color scale. Comparing it to the models in Figure 1, we see that this is significantly sparser.The resulting vector of the 2-norm of each row has 137 non-zero entries.

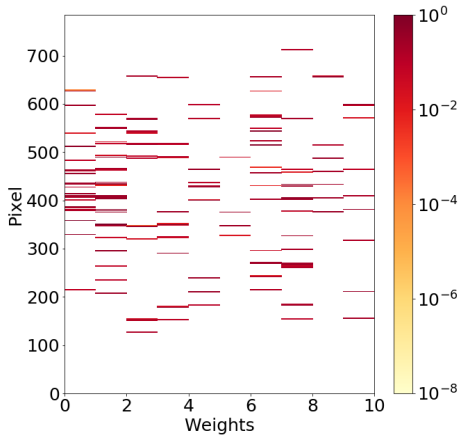
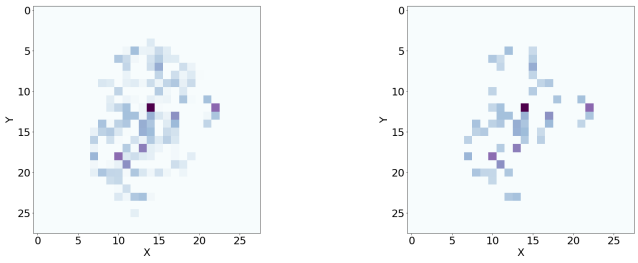


Figure 2: Lasso Sparse $\lambda_1 = 5$

We can plot the pixel locations and rank as shown in Figure 3 where the darker the color, the higher the rank. The important pixels are towards the center of the image which is expected as our images have all be centered.



(a) 137 Most Important Pixels (b) 50 Most Important Pixels

Figure 3: Important Pixel Locations

Once we have ranked our 137 pixels, we can plot the accuracy versus the N most important pixels used in the model. The pseudo-inverse was used to test our models after non-important pixels were discarded from the data. This is shown on Figure 4. Using all 137 pixels gives us an accuracy of 82.1% which is close to our result with the pseudo-inverse when all 784 pixels are used. Using the 20 most important pixels gives us an accuracy of 67.2% which is not bad considering we are using less than 3% of the total pixels.

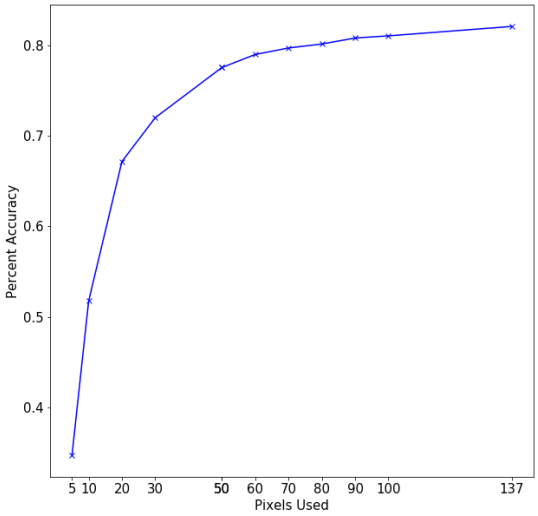
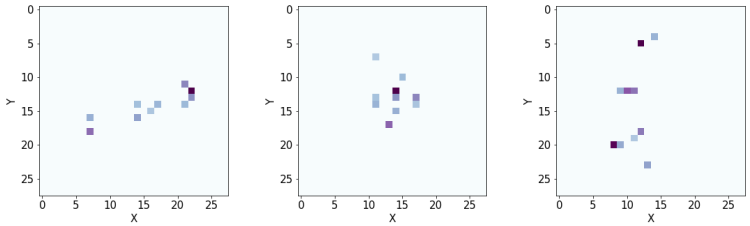


Figure 4: Accuracy versus Pixels Used

4.3 Individual Digits

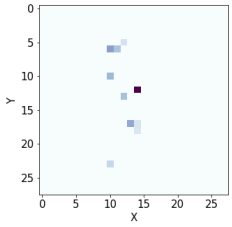
We can apply the same analysis from the previous part to individual digits. The 10 most important pixels are shown for each digit in Figure 5. Note that for '5' there were only 5 non-zero entries after doing a Lasso fit. The 10 most important pixels do not seem to look like any of the digits as an important pixel is not only where the pixel in the image should be black for a given digit, but also where it should be blank. For '0', it is possible that some of the important pixels are towards the center of the image because the classifier recognizes that a '0' has a hole.



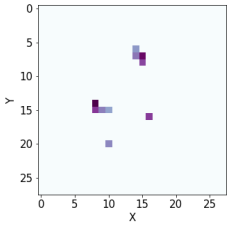
(a) 0

(b) 1

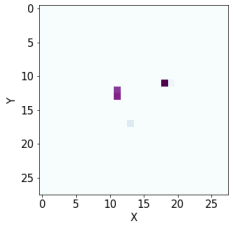
(c) 2



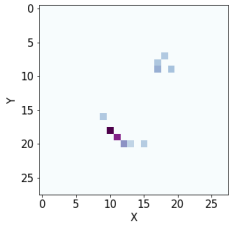
(d) 3



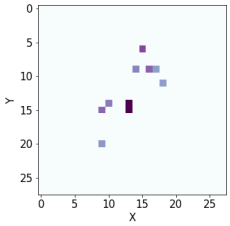
(e) 4



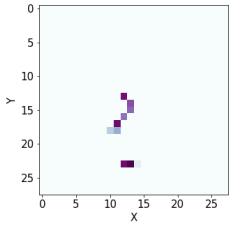
(f) 5



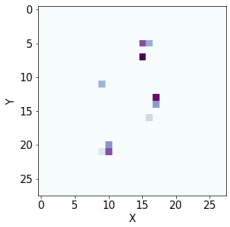
(g) 6



(h) 7



(i) 8



(j) 9

Figure 5: 10 Most Important Pixels for Each Digit

Table 2 compares the model for each digit when all pixels are used versus the top 10. Unsurprisingly, when only looking at 1 digit of interest, our model has a high accuracy as it only needs to determine if an image is the digit of interest or not rather than having to determine which out of 10 digits the image is. Using 10 pixels also seems to maintain a high degree of accuracy and seems to be better than using all pixels in some cases.

Digit	Test Fit	10 Pixel Fit
0	96.0%	81.3%
1	93.8%	96.6%
2	94.5%	92.2%
3	94.8%	92.5%
4	91.6%	91.8%
5	91.9%	91.1%
6	96.2%	95.2%
7	93.6%	93.4%
8	95.4%	90.3%
9	91.7%	89.9%

Table 2: Model Performance for each digit

5 Summary and Conclusions

We have explored the concept of regression and model selection using the MNIST data set. Using 4 different solvers, each with different regularizations, and k-fold cross-verification we were able to highlight how our classification scheme is underfitting the data. This suggests that looking at the pixel intensity and location are not enough to properly classify our data with regression. Next, by promoting sparsity, we were able to determine the most important pixels in our classification and showed that we can still retain relatively high accuracy using less than 15% of our pixels. We then applied the same analysis to each individual digit and were able to determine the important pixels for each digit.

References

[1] Jose Nathan Kutz and Steven Brunton. *Data-driven science & engineering machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

Appendix A MATLAB Functions

Below are the key MATLAB functions implemented.

- `mndata = MNIST('filepath')` specifies the location of the MNIST data
- `images_tr, labels_tr = mndata.load_training()` loads the training data and labels
- `images_tt, labels_tt = mndata.load_testing()` loads the testing data and labels
- `np.asarray(A)` converts the input to a numpy array
- `np.random.permutation(x)` creates a random permutation from 0 to x
- `plt.pcolor(A)` creates a pseudocolor plot with a non-rectangular grid using where A is the weights.
- `np.linalg.pinv(A)` returns the Moore-Penrose psuedo-inverse of A
- `np.count_nonzero(A)` returns the number of non-zero entries in A
- `clf.fit(A,B)` Fits the linear model $AX = B$ and returns X
- `np.argmin(A)` Returns the indices of the minimum values
- `np.linalg.norm(x)` Returns the two-norm of vector x
- `x.argsort()[-n:][::-1]` returns the indices of the n largest values of x from largest to smallest
- `np.sort(x)` sorts the vector from smallest to largest

Appendix B Python Code

Github Repo: https://github.com/chrismhl/AMATH563/tree/master/Homework1