AMATH 563 – HW1

Nick Montana

4/19/2019

Abstract

This report is an exploration of model selection. It investigates the MNIST data set, a collection of 60,000 images of handwritten single digits and their corresponding labels which tell us what digit the image is of. Our goal is to find a good model which gathers the most important pixels from the image, so that, if given a new image, we would be able to determine which digit it is. This problem can be seen as a way of solving the equation Ax = b.

Sec. I: Introduction and Overview

The images of the digits are given to us as a 60000x784 matrix, where each row represents an image: a 784x1 vector which can be reshaped to be a 28x28 matrix in which each entry represent a pixel. The labels of the images (0,1,…,9) are given to us as a 60000x1 vector which we translate to a 10x60000 matrix to make calculations possible. The label 1 is translated to [1; 0; …; 0], etc.

Let's examine how this exercise examines the equation Ax = b. We call the 60000x784 transposed image matrix, A, and the 60000x10 transposed label matrix, B. Our model is trying to find a 784x10 matrix X which satisfies Ax = b.

Sec. II: Theoretical Background

Because A and b are long and skinny, i.e. have more rows than they have columns, the system we are trying to solve is an overdetermined system. An overdetermined system generally has no exact solution. We therefore do what we can to minimize the error in solving the system, which we measure with the following the equation:

$$\min E = \operatorname{argmin}_x \|\mathbf{Ax} - \mathbf{b}\|_2$$

Sec. III: Algorithm Implementation and Development

We start by downloading the data by following this link: http://yann.lecun.com/exdb/mnist/. Because the data is in an unusual file format, we use the MNIST helper code given here: http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset. These functions help us to translate the data into matrices as well as to display the images. Once we have the data downloaded into matrices, our program converts the labels from digits 0,…, 9 to a matrix of 1's and 0's described in the introduction. Transposing the image and label matrices gives us the proper dimensions to work with. Theses will now become our A and B matrices in the equation Ax = B.

Our program then implements 3 Ax = B solvers: lasso, ridge, and pinv. However, it should be noted that lasso and ridge must be used by computing B one column at a time as the input for the response variable must a vector. For lasso, we handpicked the value of lambda in such a way that promoted sparsity, but not so sparse that the matrix was all 0's. Lambda was chosen by trial error and looking at the resulting X matrix in image form. These 3 solvers each gives us a different matrix for X, which we then visualize and examine the results.

Sec. IV: Computational Results

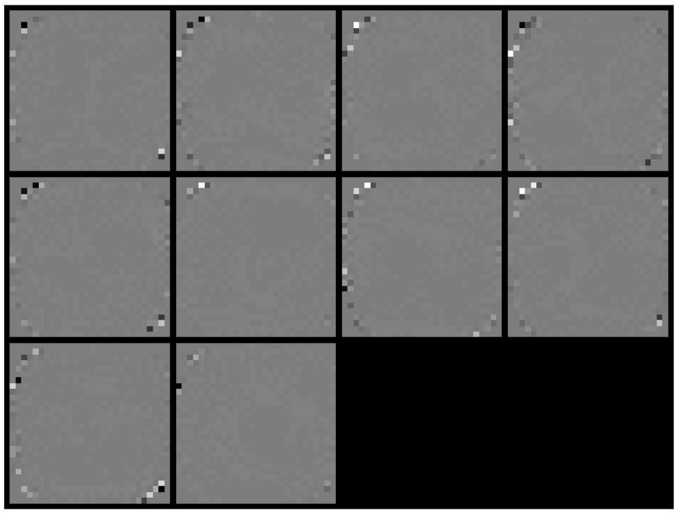

Figure 1: pinv results

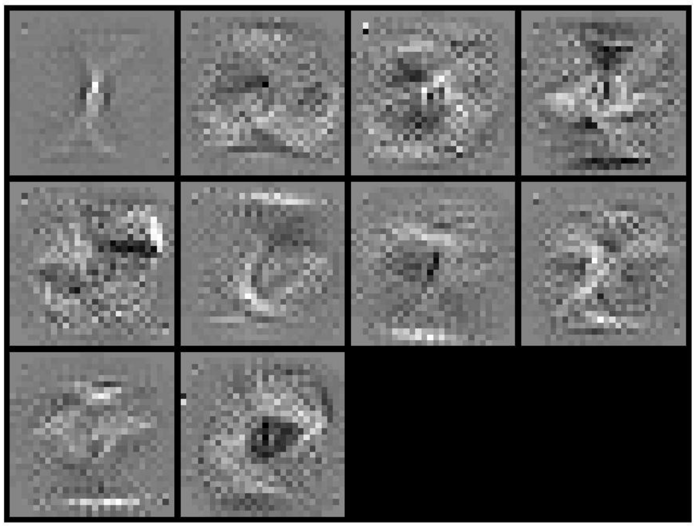

Figure 2: ridge results

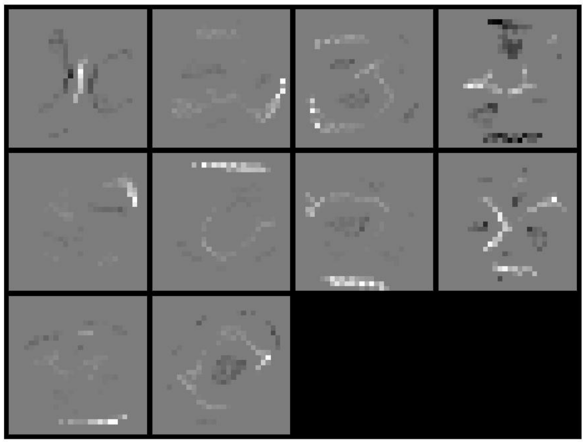

Figure 3: lasso results

Figures 1 – 3 show the results, in image form, of the 3 different methods used to solve for X. Going from left to right, each figure shows the results from numbers 1, …, 9, 0.

These images can be better understood by thinking of what it is that X *does.* X is trying to figure out which pixels of the training images are most important in determining whether the image is a particular digit or not. Thus, we find that the white pixels shown in the image are positive and are therefore an indication that if these pixels are used, the image is indeed the corresponding digit. Black pixels, however, are negative and are therefore, if used, an indication that the image is not in fact that digit. In general, the pixels with the largest absolute value are most relevant in determining which digit is being shown.

The results from using pinv are poor. The X matrix from this method is picking up pixels which are irrelevant as being important indicators. The results from using ridge are much better, but there is too much noise. That is, there are too many pixels which the method deems as important and a lot of varying degrees of importance. The results from lasso are the best, because, as shown in class, lasso promotes sparsity. We see only a few pixels which are most important.

Since lasso produced the best results, we will use this as our X. Now, given the test images, we can apply our model to see how accurate it is. Looking at the result of AX where A is the matrix of the test images, we find that we can accurately predict the label 78.35% of the time.

Sec. V: Summary and Conclusion

In sum, we created a model which we trained using images and labels from the MNIST data set. This model can then be used in the future to, given a new image that we have not seen before, predict which digit the image is representing. To do this, we applied 3 different models and examined the results. Some digits produced better results than others. 1, 2, 3, 8, and 0 can be sketched out of our resulting X matrix. While others, like 5, can barely be made out at all.

It would be an interesting phenomenon to investigate. Is this because there is much more variability in the way a digit like 5 is written? Most likely, to get better than 78% accuracy, a larger data set could be used.

Appendix A

The following functions were used in this implementation. Their description can be found on the Mathworks Documentation page:

B = lasso(X,y,Name,Value) fits regularized regressions with additional options specified by one or more name-value pair arguments. For example, 'Alpha',0.5 sets elastic net as the regularization method, with the parameter Alpha equal to 0.5.

B = ridge(y,X,k) returns coefficient estimates for ridge regression models of the predictor data X and the response y. Each column of B corresponds to a particular ridge parameter k. By default, the function computes B after centering and scaling the predictors to have mean 0 and standard deviation 1. Because the model does not include a constant term, do not add a column of 1s to X.

B = pinv(A) returns the Moore-Penrose Pseudoinverse of matrix A.

M = max(A,[],dim) returns the maximum element along dimension dim. For example, if A is a matrix, then max(A,[],2) is a column vector containing the maximum value of each row.

[M,I] = max(___) also returns the index into the operating dimension that corresponds to the maximum value of A for any of the previous syntaxes.

Appendix B

```
% Change the filenames if you've saved the files under different names

% On some platforms, the files might be saved as

% train-images.idx3-ubyte / train-labels.idx1-ubyte

images = loadMNISTImages('train-images-idx3-ubyte');

labels = loadMNISTLabels('train-labels-idx1-ubyte');


% We are using display_network from the autoencoder code

display_network(images(:,1:100)); % Show the first 100 images

disp(labels(1:10));


%We need to write something which changes the label value "5", etc. to the

%correct vector. Then, we will make a B matrix out of this.

B = zeros (10, 0);

{

for i = 1:60000
```

```
    x = labels (i, 1);
    y = zeros(10,1);
    if x == 0
        y (10, 1) = 1;
        B = [B y];
    else
        y (x, 1) = 1;
        B = [B y];
    end
end



%We have to transpose A and B to make the dimensions of Ax = B make sense.
B_better = B.';
A_better = images.';

%Implementing lasso
X = zeros (784, 10);
for i = 1:10
    X(:, i) = lasso (A_better, B_better(:, i), 'Lambda', 0.01);
end

%Implementing ridge
X = zeros (784, 10);
for i = 1:10
    X(:, i) = ridge (B_better(:,i), A_better, 0.01);
end

%Implementing pinv
X = zeros (784, 10);
```

```
A_inverse = pinv(A_better);
X = A_inverse*B_better;



j = 10;
H = reshape (X(:, j), 28, 28);
pcolor(H);
title (0);

display_network(X(:, 1:10));

output_image (1, images);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This section uses the test data.

test_images = loadMNISTImages('t10k-images-idx3-ubyte');
test_labels = loadMNISTLabels('t10k-labels-idx1-ubyte');

%This takes the test images and multiplies the matrix X that we found with
%the training data to see if the result matches the labels
Y = test_images.';
Result = Y*X;

%This function returns the largest value of Result from each row and puts
%the value into M, while putting which column this largest value was in
%into .
[M,label_results] = max(Result,[],2);
%The 10th column is actually meant for 0.
```

```
for i = 1:10000
   if label_results(i, 1) == 10
      label_results (i, 1) = 0;
   end
end

%Put these two into the same matrix.
Match = zeros(10000, 2);
Match (:,1) = label_results;
Match (:,2) = test_labels;

%Let's see how many equal each other.
h = 0;
for i =1:10000
   if Match (i, 1) == Match (i, 2)
      h = h + 1;
   end
end

%It looks like we predicted the images accurately 78.35% of the time.

%This takes the images matrix and looks at just the first coumn,
%makes it a square matrix, and then displays it as a colored image.
%It actually displays it upside down for some reason?
function f = output_image (i, images)
   M = images (:,i);
   M = M.';
   M_square = reshape (M, 28, 28);
   pcolor (M_square);
end
```