

Name: Hongda Li
Class: AMATH 584

Theory and Setup

We were given a set of data, Let's denote the matrix A to be the column data matrix storing all the vectorized images from the MNIST data set. Let B be the matrix storing all the labels for the images.

The Image is 28×28 , vectorized into 784 vector and packed into the matrix A .

The label axes are the Standard Basis vector for 10th dimension, with the digit 0 corresponds to e_{10} . Each basis vector represents a digit in the decimal counting system.

Our objective is to find a sparse linear model that maps from the image space to the Label space, let this matrix be X , assuming we were given N images and labels, then predicting the label using the model and input data will simply become:

$$XA = B$$

Where X is a 10×28^2 matrix. And each row will be an linear model for each digit. For this assignment, I set the size for the training set to be 6000, randomly chosen from the given 60000 images from the MNIST training data set, without replacement. which is enough to produce good model, and not enough to crash my computer.

When is the system over-determined and when is it Under-determined?

When the number of images used to train the model is over 28^2 , the system will be over-determined. Therefore, for the HW, we must always use more than 784 images for training the model.

Let's take a look at various type of regression solver for this HW assignment.

1. Back slash Solver (Black Box Solver)
2. Penrose's Psuedo Inverse (Least Square Subspace Project)
3. Lasso Regression with different values for Regularization.
4. Hybrid solution where we use a sparse filter produced by Lasso and then filter out data to get a sparse model via backslash.

Among all the solvers, the Lasso Regression Deserves the most attention because it prompts Sparsity.

The Lasso Solver takes in a Matrix and a vector, it solves the the $Ax = b$ problem by solving the the following optimization problem:

$$\operatorname{argmin}_x \left(\frac{1}{2N} \|Ax - b\|_2^2 + \lambda \|x\|_1 \right)$$

And with intercept for the model, we have:

$$\operatorname{argmin}_{x, \beta} \left(\frac{1}{2N} \|Ax + \beta - b\|_2^2 + \lambda \|x\|_1 \right)$$

Where λ are called the regularizers, and that is the parameter which will promote sparsity for the model. The 1-Norm will forces the optimization algorithm to reduces the size of the vector x by reducing some of the predictors that doesn't contribute a lot, and parameters (x_i) with different sizes are treated the same because of the 1-Norm. 2-Norm on the other hand, will penalize parameters that are relatively larger, so it will try to even out all the components of the x vector.

Why Sparsity? We want sparsity to get the most important predictors for the model, and it will reduce the model complexity in our case.

How do you use this regression algorithm for our model? The major problem is that, the regression model is for solving a $Ax = b$, but we are solving the $XA = B$.

We do it row by row on the X matrix.

$$XA = B \implies A^T X^T = B^T$$

Which is actually a system of simple regressions:

$$A^T (X^T)_{:,k} = (B^T)_{:,k} \quad \forall 1 \leq k \leq 10$$

And then we do it for each rows of the matrix X . What we are doing is, each row of the matrix X is a for identifying a single digit, the first row for 1, the second row for 2 .. and the last row for 0.

And each individual digit is a simple regression model with 784 predictors and one predictant. The k th row predicts whether the input is the k th digits. “1” being true, and “0” being false.

Performance Measure

Because the fact that the out put of the vector XA will be discrete, therefore by default, we take out the maximal elements in the output vector and mark it as a one on that row, and set all other entries to zero, and this vector will instead be the prediction made by the model. Name this process ‘Idxmax’

$$\left(\frac{\sum_{i,j} (\text{IdxMax}(XA) - B)_{i,j}}{2M} \right)$$

Where M is the number of columns on B . This score is the totally percentage of labels that our model predicted correctly.

It’s dividing by $2M$ because $\|\vec{e}_i - \vec{e}_j\|_1 = 2$ whenever $i \neq j$. Whenever an error is made, the 1-Norm difference will be 2.

Use Lasso For Sparse Model

One of the simple way to find the best λ is to turn up the “CV” for cross validation, the lasso command in matlab will return the best lambda that has the minimum amount of predicted variance on the model parameters.

This is slow, but it produces the best lambda to be around $\lambda = 0.01$ for all the digits. However, take note that, the regularizer needed for each digit can be very different.

After the sparse model is produced, I use the non-zero entries of the model the filter out the data matrix. Which is basically equivalent to ignoring all the pixels that didn’t play any role in the sparse model. And then after that, I use backslash to solve for a new model that is still sparse one the filtered data matrix.

The filtering process of the data matrix is:

$$F = ((X)_{K,:} \neq 0) \circ A$$

Where the operator \circ is the element wise multiplication in matlab. This is saying: Mark all the non zero entries in the k th row of the model matrix X , and then multiply it on all columns of A , sweeping it through. This will get rid of all the pixels that are viewed as not important predictor by the Lasso Regression for a certain digit.

Take notice that, **we don’t really need the lasso’s model to predict the labels**, we only need the model to predict the best predictors (The most effective pixels). And that is why I throw the Lasso Model away after getting the sparse filter.

In addition, the model requires the use of a “Intercept” vector, which means it needs a constant vector on the end, which will change the dimension of the matrix A , however I made the choice of keeping intercept for the Standardization of data, which is important for numerical stability.

However, there is another alternative approach that allows us to solve $XA = B$ as a matrix vector problem:

$$(I \otimes A)x = \text{Vec}(B^T)$$

Where I is a 10×10 matrix in this context

Unfortunately this way of solving the problem requires the use of Sparse matrix for larger data set, but the lasso command in matlab doesn't allow for a sparse matrix to be the inputs.

Visualizing the Models

The rows of the matrix X is vectorized into 28×28 and are packed into a 2×5 grid.

Using Pcolor, I visualized the absolute value of the exponent on each of the entires, which is computed via:

$$|\log((X == 0) + X)|$$

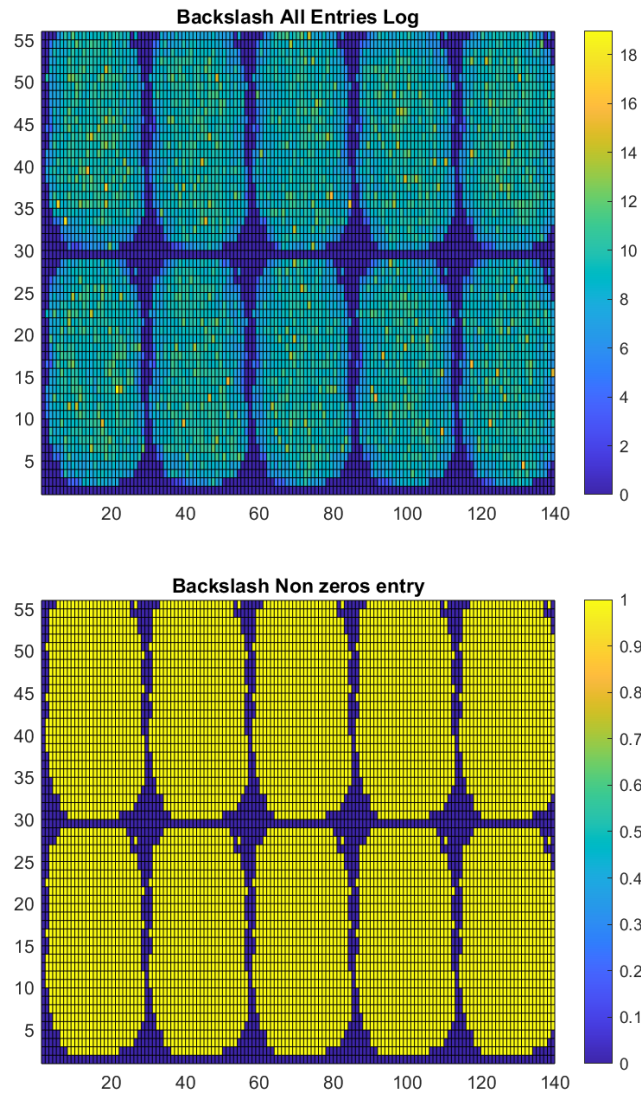
This extract out all absolute value of the powers on the entries and preserves all the zero terms.

This is necessary because when I first plotted it, it didn't turns out great because most of the entries are having exponents that are in the range of 10^{-8} . Which makes sense because the image are having pixels that are in the int8 range, and there are 784 pixels in total, and just one of them will sum up way beyond 1, and hence the matrix is going to reduce them by multiplying a small enough number, summing them up, before mapping them to the label space.

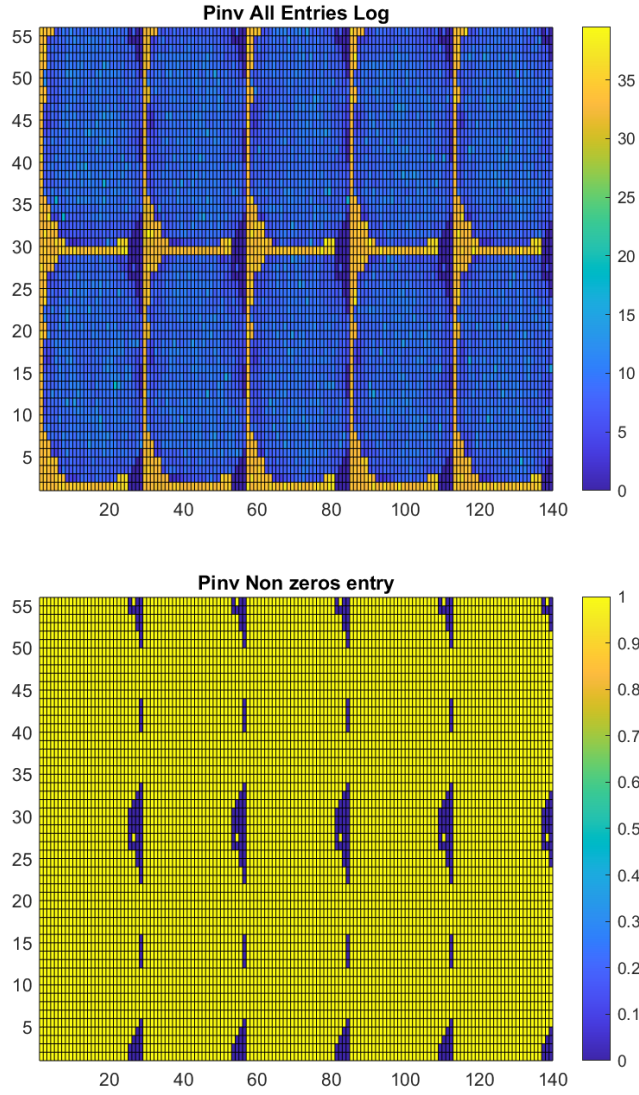
A second way of visualizing it is just marking all the nonzero entries as 1, and then plot them out.

Let's take a look at the these linear model.

Plots and Interpretations

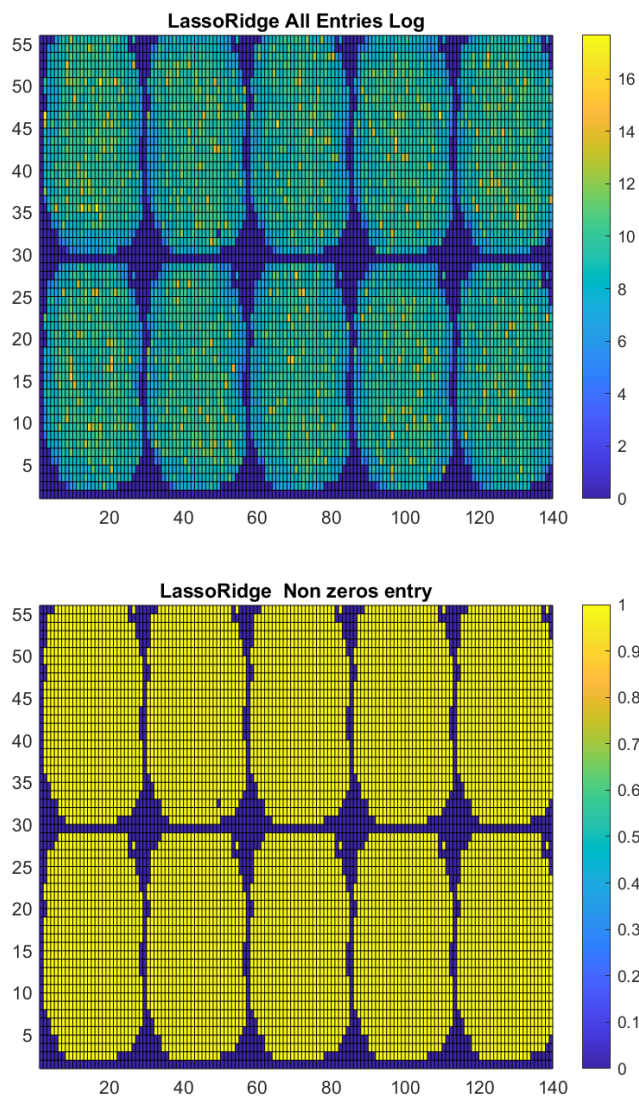


Observe that backslash produced a matrix that is dense. When it solves the matrix problem, it look for the least square projection on to the subspace, and then reduce the 2-Norm of the projection vector using a QR based algorithm.



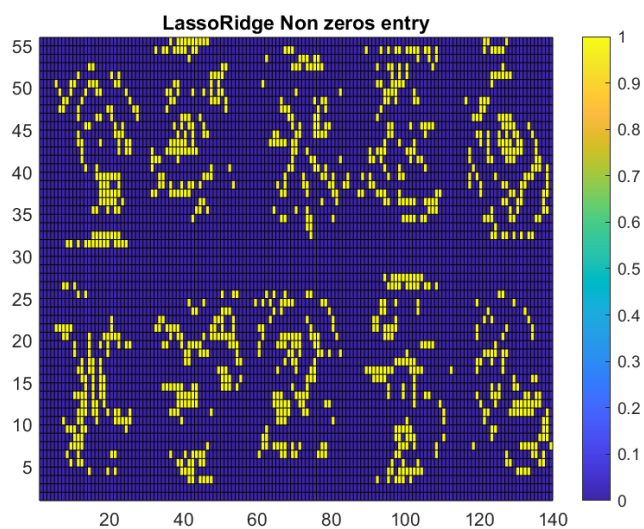
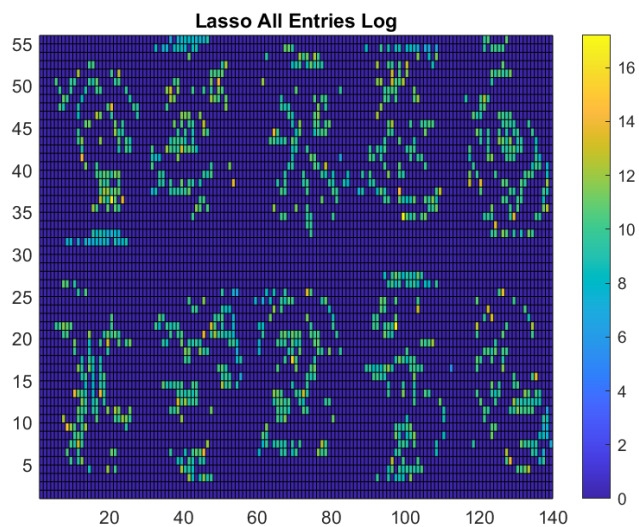
Pseudo inverse on the other hand, produces a model that contains entries with a larger variance compare to the backslash model, suggesting that it made no attempt to regularize the norm of the solution, unlike back slash.

In addition observe that the exponent is much larger compare to the backslash model and it spans a larger range.



Observe that the lassoridge model, which is done by using the ‘lasso’ command in matlab by setting up $\alpha \approx 0$, something close to zero, we obtain a model that looks very similar to the backslash.

As α approaches zero, the lasso regression will approach a Ridge regression.



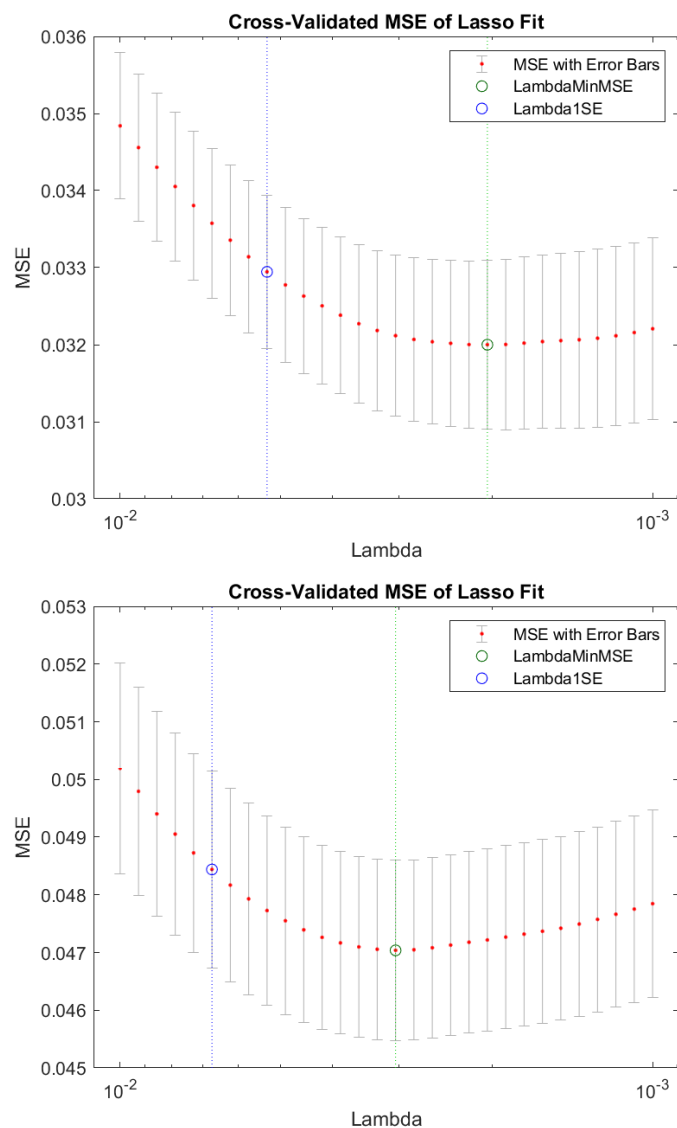
SPARSITY. The ridge regression with λ set to 0.008 produces this model, and observe that how most of the entries on the matrix X are zeroes. This is saying that the Lasso optimizer is able to throw out redundant predictors.

The idea here is that, if we wrote down a digit “1”, then it’s likely that a lot of pixels will appear together, and that is covariance, or colinearity, and this is what the Lasso optimizer is design to eliminate.

Finding The Best Regularizer: λ

The best regularizer produces models with the least amount of variance on the parameters (through cross validations).

Unfortunately, doing this will result in too much computations on personal PC, and that is where we use the “CV” option for “lasso” to get an estimation for the MSE and the Variance of MSE.



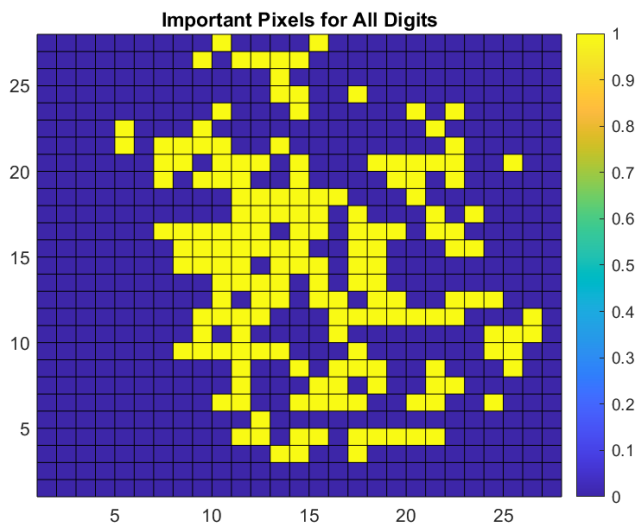
The first plot is for the digit 1 and the second one is for the digit 5. Observe that they have different optimal lambdas for smallest MSE and smallest Variance on MSE. .

Most Important Pixels for All Digits

For each column of the matrix produced from the L1 regularized Lasso Regression, if it has more than a certain number of non-zero digit, then that column will be important for all the digits. The reasonable threshold is 3. This value is found empirically.

To justify this approach, each of the columns of matrix X corresponds to a pixel on the image, and each of the row corresponds to a digit. If there is a column that has all ten entries as non-zero, then it means that pixel is viewed as an important predictor for all 10 digits.

By setting the threshold to be 3, the following plot is produced:

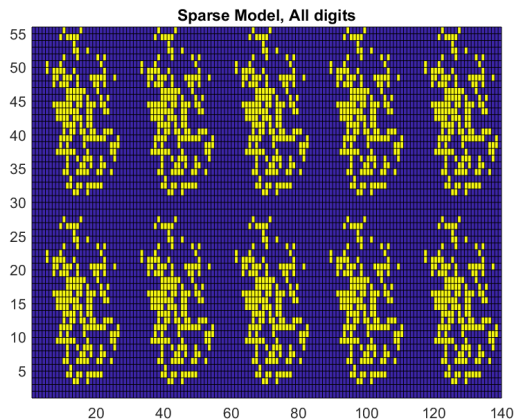


Intuitively, people will think that the important pixels should be all clustered at the center, however that is exactly not the right way, because every digits will have some strokes that cross the center, and in that case, it makes the pixels in the center most redundant.

However we want filters that are as spread out as possible (A big blob of pixels are likely to contain co-linearity among the pixels in it.), so that we can get more information from different places on the image, encapsulating features from all digits.

So I used this model to create a filter, and then I filter out the pixels that are not important. After that, I used the backslash to attain another model that is sparse and predicts with 83.42% of labels in the test set correctly for a particular instance of random sampling.

Visualizing the matrix X for this model, we have:



The density of the model is given by summing up all the non-zero terms and then divided by the total number of elements in the matrix, which gives: 20%.

Each Digit Analysis

For each digit, we have a regression model that takes in 784 predictors and predict whether the input is a certain digits or not.

One of the simple way of doing it is to turn off the "Intercept" options in the Lasso command in matlab and get a solution matrix X , training it on each digits with the same λ for the regularizer. To obtain the model, which is shown earlier.

Another approach is to look for the best regularizer for the regression model on each digit.

However, this is too slow using CV, and hence I cheated and use "fminbnd" and a creative loss function. Here is the routine.

Let $L(d, \lambda)$ be a function that performs lasso regression on the digit d and returns a linear model given λ , then we want to find the best lambda with the expression:

$$\lambda_d = \underset{\lambda}{\operatorname{argmin}} \left(\frac{\|L(d, \lambda)A + \beta - (B)_{d,:}\|_2}{1 - \rho} \right)$$

Where ρ denotes the density of the model, the ratio between all non-zero entries in the matrix and the total number of entries in the matrix. β denotes the intercept value for the lasso model that got trained.

This loss function promotes the accuracy of the model and penalize dense models. And it's then called by the "fminbnd" function on the interval $[0.001, 0.05]$ for λ .

This model directly produced by the lasso regression is not used to do the prediction, it's used as a filter to filter out pixels in the data matrix.

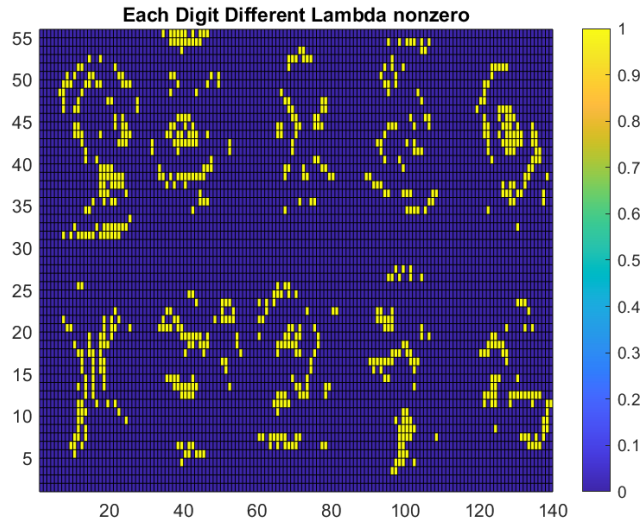
$$F = ((X)_{K,:} \neq 0) \circ A$$

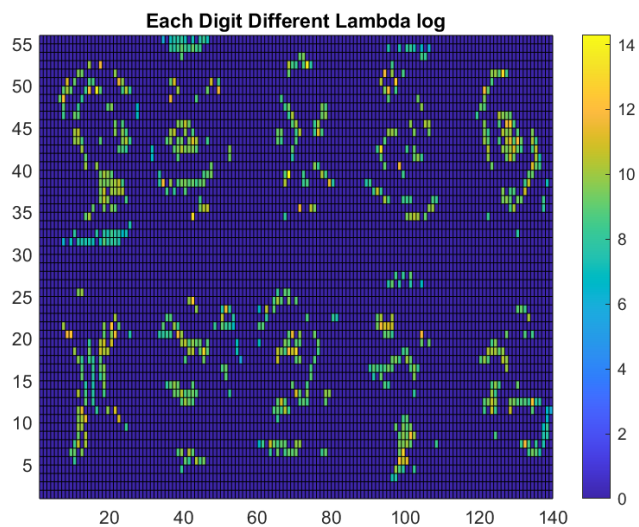
Then solves

$$x_k F = B$$

Where x_k is the k th row of the matrix X , Using Backslash.

Using this method, we were able to get the following model:





Observe that this model is even sparser than the model we got from lasso regression with $\lambda = 0.008$ for all digits.

The model predicts 83% labels correctly over the MNIST test dataset. On average, the model uses only 9% of all pixels. Half as many pixels comparing using the same filter for all digits.

Code

1. `EachDigitSparseModel.m`
This is a function that can produce a sparse model given a Filter, if a filter is not provided, then it will optimize the best lasso regression model for each digit, and then use the sparse model to filter out pixels. And then in the end, it will solve for the rows of X for each digit with backslash.
2. `GetMatrices.m`, `MNISTRead.m`
Code for reading the files and getting the matrices.
3. **`HWDemonstration.m`**
Demonstrating what ideas in this paper in code.
4. `ScrachPaper.m`
Scratch paper works most of the code doesn't work.
5. `TrainingDataPool.m`
A class that contains important codes we need to interact with the training, testing on data set.
6. `VisualizeAllLayers.m`
A file that will visualize all the entries in a 10×784 model.
7. `VisualizeOverlayed.m`
Given a model, visualize the most important pixels for all the digits.

Like to my [Repo](#), containing all the code used for this assignment.

```

1  function [Model, Filters] = EachDigitSparseModel(datapool, Filters)
2      % Theory:
3      %     Use the sparse filter for each digit and then use backslash to
4      %     get the model.
5      % DataPool:
6      %     A datapool object.
7      % Filters: 28 x 28 x 10
8      %     10 Filters, for each digit, the filter should come from one of
9      %     the sparse model using Lasso L1 Norm.
10     %
11
12     if nargin == 1 % Compute the best filters.
13         Filters = zeros(28, 28, 10);
14         datapool.ApplySparseFilter(ones(28^2, 1));
15         %datapool.Intercept = 0;
16         for II = 1: 10
17             disp(strcat("Computing Best Filter for digit: ", num2str(II)));
18             Options = optimset("display", "iter", "MaxFunEvals", 40);
19             [LambdaOptimal, ~] = fminbnd(@(x) GetError(datapool, II, x), 0.00, 0.05, Options);
20             [Filter, ~] = datapool.LassoSingleDigitTrain(II, LambdaOptimal, 1);
21             Filters(:, :, II) = reshape(Filter, 28, 28);
22         end
23         %datapool.Intercept = 1;
24     end
25
26     Model = zeros(10, 28^2);
27     for II = 1: 10
28         datapool.ApplySparseFilter(Filters(:, :, II) ~= 0);
29         A = datapool.DataMatrix;
30         B = datapool.LabelMatrix;
31         B = B(II, :)';
32         Model(II, :) = (A'\B)';
33     end
34
35     function Error = GetError(datapool, digit, lambda)
36         [M, stats] = datapool.LassoSingleDigitTrain(digit, lambda, 1);
37         Predicted = M'*datapool.Afull + stats.Intercept;
38         Density = sum(M ~= 0)/784;
39         CorrectAnswers = datapool.Bfull(digit, :);
40         Error = norm(Predicted - CorrectAnswers)/(1 - Density);
41     end
42 end

```

```

1  function [A, B, Images, Labels] = GetMatrices(toRead)
2      % Read the A, B matrix from the data set as specified in the HW
3      % assignment.
4      % toRead:
5      %   Binary variable, if it's 1, then it's the training set if it's 2
6      %   then it's the validation set.
7
8      mode          = ["train", "t10k"];
9      DataFile       = strcat("data\", mode(toRead), "-images.idx3-ubyte");
10     Labels          = strcat("data\", mode(toRead), "-labels.idx1-ubyte");
11     [Data, Labels] = MNISTRead(DataFile, Labels);
12     Images          = Data;
13
14     [m, n, q] = size(Data);
15     A          = zeros(m*n, q);
16     for II = 1: size(Data, 3)
17         A(:, II) = reshape(Data(:, :, II), m*n, 1);
18     end
19
20     B          = zeros(10, q);
21     for II = 1: q
22         if Labels(II) == 0
23             B(10, II) = 1;
24             continue;
25         end
26         B(Labels(II), II) = 1;
27     end
28
29 end

```

```

1  %% PRODUCING VARIOUS MODELS USING DIFFERENT SOLVERS
2  clear variables; clc;
3
4  data      = TrainingDataPool(6000);
5  DataMatrix = data.DataMatrix;
6  LabelMatrix = data.LabelMatrix;
7  %%
8  X1        = (DataMatrix'\LabelMatrix)';
9  X2        = (pinv(DataMatrix')*LabelMatrix)';
10 [X3, Beta3] = data.SingleLambdaLasso(0.008, 0.0001); % Ridge Regression.
11 [X4, Beta4] = data.SingleLambdaLasso(0.008, 1);      % Full L1
12 data.Intercept = 0;
13 XX3           = data.SingleLambdaLasso(0.008, 1);    % L1 Lasso no intercept.
14 data.Intercept = 1;
15
16
17 %% Model Assessment
18 X1Score       = data.GetModelScore(X1);
19 X2Score       = data.GetModelScore(X2);
20 X3Score       = data.GetModelScore(X3, Beta3);
21 X4Score       = data.GetModelScore(X4, Beta4);
22 XX3Score      = data.GetModelScore(XX3);
23
24 %% VISUALIZING VARIOUS MODELS PRODUCED
25
26 VisualizeAllLayers(log(1./abs((X1 == 0) + X1))); colorbar; title("Backslash All Entries Log");
27 saveas(gcf, "backslash-log", "png");
28
29 VisualizeAllLayers(sign(abs(X1))); colorbar; title("Backslash Non zeros entry");
30 saveas(gcf, "backslash-nonzero", "png");
31
32 VisualizeAllLayers(log(1./abs((X2 == 0) + X2))); colorbar; title("Pinv All Entries Log");
33 saveas(gcf, "pinv-log", "png");
34
35 VisualizeAllLayers(sign(abs(X2))); colorbar; title("Pinv Non zeros entry");
36 saveas(gcf, "pinv-nonzero", "png");
37
38 VisualizeAllLayers(log(1./abs((X3 == 0) + X3))); colorbar; title("LassoRidge All Entries Log");
39 saveas(gcf, "lassoridge-log.png", "png");
40
41 VisualizeAllLayers(sign(abs(X3))); colorbar; title("LassoRidge Non zeros entry");
42 saveas(gcf, "lassoridge-nonzero.png", "png");
43
44 VisualizeAllLayers(log(1./abs((X4 == 0) + X4))); colorbar; title("Lasso All Entries Log");
45 saveas(gcf, "lasso-log.png", "png");
46
47 VisualizeAllLayers(sign(abs(X4))); colorbar; title("LassoRidge Non zeros entry");
48 saveas(gcf, "lasso-nonzero.png", "png");
49
50 %% VARIABILITY IN LAMBDA
51 [B, Stats] = data.LassoSingleDigitTrainCV(1, logspace(-3, -2, 30), 1);
52
53 %%
54 lassoPlot(B, Stats, "PlotType", "CV");

```

```

55 legend("Show");
56 saveas(gcf, "digit-1-lasso-cv", "png")
57
58 %%
59 [B, Stats] = data.LassoSingleDigitTrainCV(5, logspace(-3, -2, 30), 1);
60
61 %%
62 lassoPlot(B, Stats, "PlotType", "CV");
63 legend("Show");
64 saveas(gcf, "digit-5-lasso-cv", "png");
65
66
67 %% SPARSE FILTER ALL DIGITS
68
69 Filter = VisualizeOverlaid(X4, 3); title("Important Pixels for All Digits")
70 saveas(gcf, "filter-alldigits", "png");
71 Filter = Filter + zeros(28, 28, 10);
72 [X5, ~] = EachDigitSparseModel(data, Filter);
73 disp(strcat("Sparse model all digits scored: ", num2str(data.GetModelScore(X5)), "%"));
74 disp(strcat("Model Density: ", num2str(sum(sum(X5~=0))/7840)));
75 VisualizeAllLayers(X5 ~= 0); title("Sparse Model, All digits");
76 saveas(gcf, "sparse-all-digits", "png");
77
78 %% SPARSE EACH DIGITS DIFFERENT LAMBDA FOR EACH DIGITS
79 [X6, ~] = EachDigitSparseModel(data);
80 disp(strcat("Model Density: ", num2str(sum(sum(X6~=0))/7840)));
81 VisualizeAllLayers(X6 ~= 0);
82 colorbar; title("Each Digit Different Lambda nonzero");
83 saveas(gcf, "each-digit-diff-lambda-log", "png");
84 VisualizeAllLayers(log(1./abs((X6 == 0) + X6)));
85 colorbar; title("Each Digit Different Lambda log");
86 saveas(gcf, "each-digit-diff-lambda-nonzero", "png");
87 X6Score = data.GetModelScore(X6);
88 disp(strcat("Model Score: ", num2str(X6Score)));

```



```

1  clc; clear variables;
2  % Load the Training set.
3
4  N    = 60000; % When N > 784, over determined, when N < 784, underdetermined.
5  data = TrainingDataPool(N);
6  A    = data.DataMatrix;
7  B    = data.LabelMatrix;
8
9  %% DIRECT SOLVE FOR A LINEAR MODEL
10 % Linear Mapping:  $R^{784} \rightarrow R^{10}$ , then 10 by 784, then  $XA = B$  is the model
11 % description..
12 X1   = (A'\B')';
13 X2   = (pinv(A')*B')';
14
15
16 %% MULTI LASSO TRAIN
17 figure;
18 N = 1000;
19 for II = 1: 10
20     data.Scramble(N); % Batch size
21     Lambdas    = linspace(0.001, 0.01, 10);
22     ModelSeries = data.MultiLambdaLasso(Lambdas);
23     Errors      = data.ModelsErrorsTypeI(ModelSeries);
24     Errors      = Errors./(10000*2); % Across 1k test data set.
25     plot(Lambdas, Errors); hold on;
26     disp(strcat("Training Lasso: ", num2str(II)));
27 end
28 % Lambda Optimal: 0.004;
29
30
31 %% LASSO MODEL TRAIN ALL
32 data.Scramble(6000);
33 data.ApplySparseFilter(ones(28, 28)); % CLEAR THE FILTERS!
34 %%
35 [X3, StatsX3] = data.SingleLambdaLasso(0.004);
36 %%
37 [X4, StatsX4] = data.SingleLambdaLasso(0.004, 0.8);
38 %%
39 [X5, StatsX5] = data.SingleLambdaLasso(0.004, 0.5);
40 %%
41 % [X6, StatsX6] = data.RobustFit();
42 [X6, StatsX6] = data.SingleLambdaLasso(0.004, 0.01);
43 %%
44 [X7, StatsX7] = data.SingleLambdaLasso(0.01);
45
46 %% SCORING BOARD
47 close all;
48 Score = data.GetModelScore(X1);
49 VisualizeOverlayered(X1); title("X1");
50 disp(strcat("BackSlack Model score: ", num2str(Score)));
51
52 Score = data.GetModelScore(X2);
53 VisualizeOverlayered(X2); title("X2");
54 disp(strcat("Puesdo Inverse Model Score: ", num2str(Score)));

```

```

55
56 Score = data.GetModelScore(X3);
57 VisualizeOverlaid(X3); title("X3");
58 disp(strcat("Lasso Single score: ", num2str(Score)));
59
60 Score = data.GetModelScore(X4);
61 VisualizeOverlaid(X4); title("X4");
62 disp(strcat("Lasso With Alpha 0.8: ", num2str(Score)));
63
64 Score = data.GetModelScore(X5);
65 VisualizeOverlaid(X5); title("X5");
66 disp(strcat("Lasso With Alpha 0.5: ", num2str(Score)));
67
68 Score = data.GetModelScore(X6);
69 VisualizeOverlaid(X6); title("X6");
70 disp(strcat("Ridge Regression: ", num2str(Score)));
71
72 Score = data.GetModelScore(X7);
73 VisualizeOverlaid(X7); title("X7");
74 disp(strcat("Extreme Lasso Regression: ", num2str(Score)));
75 %% SPARSE MODEL VISUALIZATION!!!
76
77 VisualizeAllLayers(X7); title("X7")
78 %%
79 VisualizeAllLayers(X4); title("X4")
80 %%
81 VisualizeAllLayers(X3); title("X3")
82
83
84 %% SPARSE MODEL TRAINING!!!
85 clc;
86 data.Scramble(6000);
87 Filter = VisualizeOverlaid(X3, 5);
88 disp(strcat("Filter Density: ", num2str(sum(sum(Filter))./(28^2))));
89 data.ApplySparseFilter(Filter);
90 A = data.DataMatrix;
91 B = data.LabelMatrix;
92 X1Sparse = (A'\B')';
93
94 VisualizeAllLayers(X1Sparse);
95 Score = GetModelScore(data, X1Sparse);
96 disp(strcat("BackSlash Sparse Model score: ", num2str(Score)));
97
98 %% FULL SPARSE MODEL FOR EACH DIGITS!
99 clc;
100 data.Scramble(6000);
101 [X8, X8Filters] = EachDigitSparseModel(data);
102 disp(strcat("Filter Density: ", num2str(sum(sum(X8~=0))./(10*28^2))));
103 VisualizeAllLayers(X8);
104
105 Score = data.GetModelScore(X8);
106 disp(strcat("BackSlash Sparse Model each digit: ", num2str(Score)));
107
108 %% HELPER FUNCTIONS

```

```

1  classdef TrainingDataPool < handle
2
3      properties
4          Images;      % Image Tensor Full
5          Labels;      % Image Label vector full
6          Afull;      % Reshaped Data Matrix full
7          Bfull;      % Image label matrix full
8
9          % For training.
10         DataMatrix;
11         LabelMatrix;
12         ChosenSubset;
13
14         A = nan;      % Flatten Tensor
15         b = nan;      % Flatten Label vector
16
17         TestMatrixB; % 1k label matrix.
18         TestMatrixA; % 1k Data matrix.
19
20         Intercept = 1; % An extra option for Lasso Model, if this is one,
21                        % Then all lasso model are going to have intercept.
22
23     end
24
25     methods
26
27         function this = TrainingDataPool(N)
28             % Constructor.
29             [this.Afull, this.Bfull, this.Images, this.Labels] = ...
30             GetMatrices(1);
31             if N > length(this.Labels)/2
32                 error("No! Reserve 50% of them for cross validations.");
33             end
34             this.Scramble(N);
35             [this.TestMatrixA, this.TestMatrixB, ~, ~] = ...
36             GetMatrices(2);
37
38         end
39
40         function void = Scramble(this, N)
41             % Scramble all the data and labels for training, N of them.
42             RandomIdx = randperm(length(this.Labels)); % No repetition
43             this.ChosenSubset = RandomIdx(1: N);
44             this.DataMatrix = this.Afull(:, this.ChosenSubset);
45             this.LabelMatrix = this.Bfull(:, this.ChosenSubset);
46             void = nan;
47         end
48
49         % Swap to Binary Label with +1, -1 for the label matrix.
50         function void = SwapLabelBinaryMode(this)
51             B = this.Bfull(:, this.ChosenSubset);
52             B = -(B == 0) + B;
53             this.LabelMatrix = B;
54             void = 0;

```

```

55     end
56
57     % Swap back to 0, 1 label.
58     function void = SwapLabelUnaryMode(this)
59         this.LabelMatrix = this.Bfull(:, this.ChosenSubset);
60         void = 0;
61     end
62
63     function [A, b] = FlattenTensor(this)
64         % Get a matrix A, such that, solving  $Ax = b$  gives  $x$  that is a
65         % flattened linear model of size 784 by 10. This is for using
66         % lasso that does regression all digits at the same time.
67         if ~isnan(this.A)
68             A = this.A;
69             b = this.b;
70             return
71         end
72         B = this.LabelMatrix;
73         A = kron(eye(10), this.DataMatrix');
74         b = reshape(this.LabelMatrix', size(B, 1)*size(B, 2), 1);
75
76     end
77
78     % Recover the 7840 by 1 matrix to a 10 by 784 model.
79     function X = VectorRecover(this, Vector)
80         % Recover vectorized linear model from the flattened vector.
81         X = reshape(Vector, size(this.Images, 1)*size(this.Images, 2), 10);
82         X = X';
83     end
84
85
86     % Given a model 10 by 764 by, and get the prediction made by this
87     % model when we multiply it on the right handside by the
88     % DataMatrix.
89     function [Predicted, Error] = Type1ModelPredictDiscrete(this, model)
90         Intercept = zeros(10, 1);
91         [Predicted, Error] = Type1ModelPredictDiscreteIntercept(this, model, Intercept);
92     end
93
94     % Predict the 10 by 784 model with an intercept vector.
95     function [Predicted, Error] = Type1ModelPredictDiscreteIntercept(this, model, intercept)
96
97         % Predict on ALL DATA with a given Type I model.
98         Predicted = model*this.TestMatrixA + intercept;
99         NewMatrix = zeros(size(Predicted));
100
101         for ColIndex = 1: size(NewMatrix, 2)
102             [~, Idx] = max(Predicted(:, ColIndex));
103             NewMatrix(Idx, ColIndex) = 1;
104         end
105         Predicted = NewMatrix;
106         Error = sum(sum(abs(Predicted - this.TestMatrixB)));
107     end
108

```

```

109 % Get the percentable of correctly predictor lable on the MNIST
110 % test set. the model given 10 by 784 matrix.
111 function Score = GetModelScore(this, model, intercept)
112     if nargin == 2
113         intercept = zeros(10, 1);
114     end
115     Predicted = this.Type1ModelPredictDiscreteIntercept(model, intercept);
116     ModelScore = sum(sum(abs(Predicted - this.TestMatrixB)));
117     Score = 1 - ModelScore./(2*size(this.TestMatrixB, 2));
118 end
119
120 function Errors = ModelsErrorsTypeI(this, models)
121     % Given a series of linear model, return all the errors on prediction.
122     % Models:
123     %     10 by 784 by N
124
125     Errors = zeros(1, size(models, 3));
126     for II = 1: size(models, 3)
127         [~, Error] = this.Type1ModelPredictDiscrete(models(:, :, II));
128         Errors(II) = Error;
129     end
130 end
131
132 function ModelSeries = MultiLambdaLasso(this, Lambdas)
133     % Given multiple lambdas, returns a series of 10 by 764
134     % matrices for each of the given lambdas.
135     [m, n] = size(this.Images(:, :, 1));
136     p = length(Lambdas);
137     A = this.DataMatrix;
138     B = this.LabelMatrix;
139     Models = zeros(10, m*n, p);
140
141     for II = 1: 10 % 10 rows.
142         % [Outputs, ~] = lasso(A', B(II, :)', "lambda", Lambdas);
143         [Outputs, ~] = this.LassoSingleDigitTrain(II, Lambdas, 1);
144
145         for JJ = 1: length(Lambdas)
146             Models(II, :, JJ) = Outputs(:, JJ);
147         end
148     end
149     ModelSeries = Models;
150 end
151
152 function [Model, Intercept] = SingleLambdaLasso(this, lambda, alpha)
153     % Given a value for lambda, get a 10 by 784 model for all the
154     % all the digits with the same lambda.
155     if nargin == 2
156         alpha = 1;
157     end
158     [m, n] = size(this.Images(:, :, 1));
159     A = this.DataMatrix;
160     B = this.LabelMatrix;
161     Model = zeros(10, m*n);
162     Intercept = zeros(10, 1);

```

```

163         for II = 1: 10 % 10 rows.
164             [Outputs, Info] = this.LassoSingleDigitTrain(II, lambda, alpha);
165             Model(II, :) = Outputs;
166             if this.Intercept == 1
167                 Intercept(II) = Info.Intercept;
168             end
169         end
170     end
171
172     % Train and get model for a single digit.
173     % This is a subroutine that got repeatedly use hence it's been
174     % factored out here.
175     function [SingleDigitModel, Stats] = LassoSingleDigitTrain(this, idx, lambdas, alpha)
176         A = this.DataMatrix;
177         B = this.LabelMatrix;
178         [Outputs, Info] = lasso(A', B(idx, :)', ...
179             "lambda", lambdas, "alpha", alpha, "Intercept", boolean(this.Intercept));
180         SingleDigitModel = Outputs;
181         Stats = Info;
182     end
183
184     % Train a single digit model with given regularizer.
185     function [SingleDigitModel, Stats] = LassoSingleDigitTrainCV(this, idx, lambdas, alpha)
186         A = this.DataMatrix;
187         B = this.LabelMatrix;
188         [Outputs, Info] = lasso(A', B(idx, :)', "lambda", lambdas, "alpha", alpha, "CV", 20);
189         SingleDigitModel = Outputs;
190         Stats = Info;
191     end
192
193     % Perform Robust fit on all digits.
194     function [Model, Stats] = RobustFit(this)
195         [m, n] = size(this.Images(:, :, 1));
196         A = this.DataMatrix;
197         B = this.LabelMatrix;
198         Model = zeros(10, m*n);
199         Stats = cell(1, 10);
200         for II = 1: 10 % 10 rows.
201             [Outputs, Info] = robustfit(A', B(II, :)', [], [], "off");
202             Model(II, :) = Outputs;
203             Stats{II} = Info;
204             disp("Robust fit...")
205         end
206     end
207
208     % Filter out things with the sparse model
209     % Filter is a binary matrix.
210     % Return a sparse data matrix and lable matrix, replace the field
211     % with the sparse data matrix, without changing the size of it.
212     % (So it's full of zeroes in it. )
213     % Call it without any argument and it will restore dense data
214     % matrix.
215     function [SparseDataMatrix] = ApplySparseFilter(this, Filter)
216         if isequal(size(Filter), [28, 28])

```

```

217         Filter = reshape(Filter, 28^2, 1); % Shape to column vector.
218     end
219     AChosen      = this.Afull(:, this.ChosenSubset);
220     this.DataMatrix = Filter.*AChosen;
221     SparseDataMatrix = this.DataMatrix;
222 end
223 end
224 end

```

```

1  function Layers = VisualizeAllLayers(model)
2
3
4      Layers = zeros(28, 28, 10);
5      for II = 1: 10
6          Layers(:, :, II) = reshape(model(II, :), size(Layers, [1, 2]));
7      end
8
9      DrawingBoard = zeros(28*2, 28*5);
10     for II = 1: 10
11         III = floor((II - 1)/5);
12         JJJ = mod((II - 1), 5);
13         DrawingBoard(III*28 + 1: (III + 1)*28, JJJ*28 + 1: (JJJ + 1)*28)...
14             = Layers(:, :, II);
15     end
16     figure; pcolor(DrawingBoard);
17
18 end

```



```

1  function Filter = VisualizeOverlaid(model, threshold) % Visualize the model.
2      if nargin == 1
3          threshold = 5;
4      end
5
6      model      = model ~= 0; % Important!
7      SummedUp = zeros(1, size(model, 2));
8      for II = 1: 10
9          SummedUp = SummedUp + model(II, :);
10     end
11     SummedUp = reshape(SummedUp, 28, 28);
12     Filter   = SummedUp >= threshold; % Threshold
13     figure;
14     pcolor(Filter); colorbar;
15 end

```