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, vetorized into 7841 vector and packed into the matrix A.

The label ares 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-determine?

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 msut 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 (Lease 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:

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

And with intercept for the model, we have:

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

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:

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

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 kth 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 $||\vec{e}_i - \vec{e}_j||_1 = 2$ whenver $i \neq j$. 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:

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

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:

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

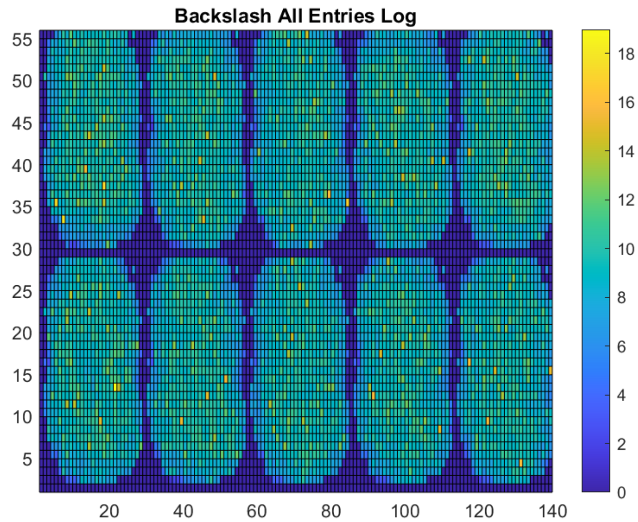
This extract out all absolute value of the powers on the entries and preserves all the zero terms.

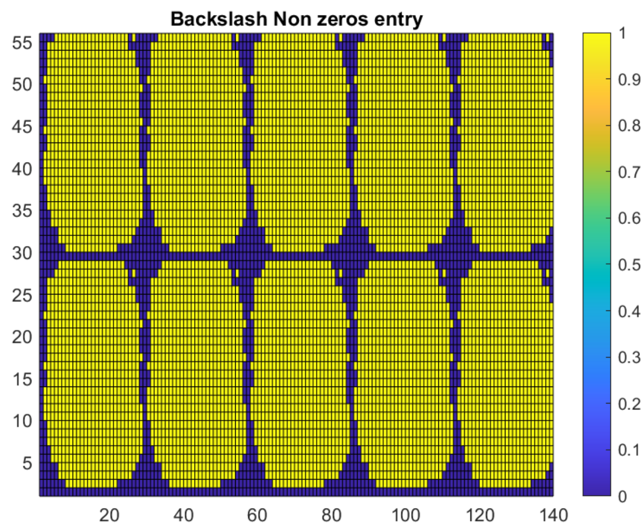
This is necessary because when I first plotted it, it didn't turns out great because most of the entries are having exponents that are in the range of 10^{-8}. Which makes sense because the image are having pixels that are in the int8 range, and there are 784 pixels in total, and just one of them will 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.

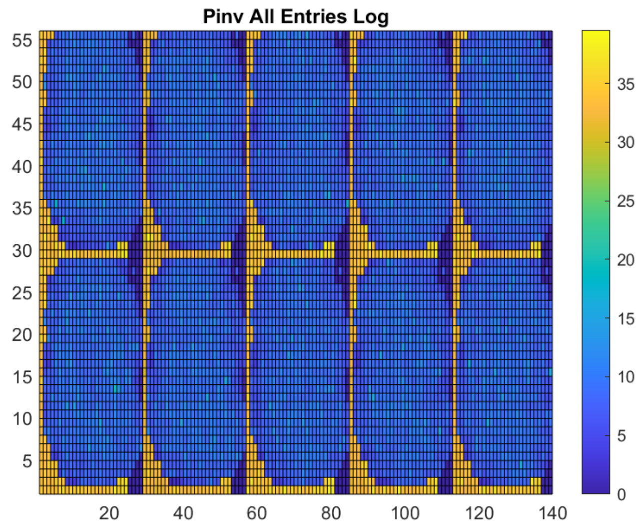
Let's take a look at the these linear model.

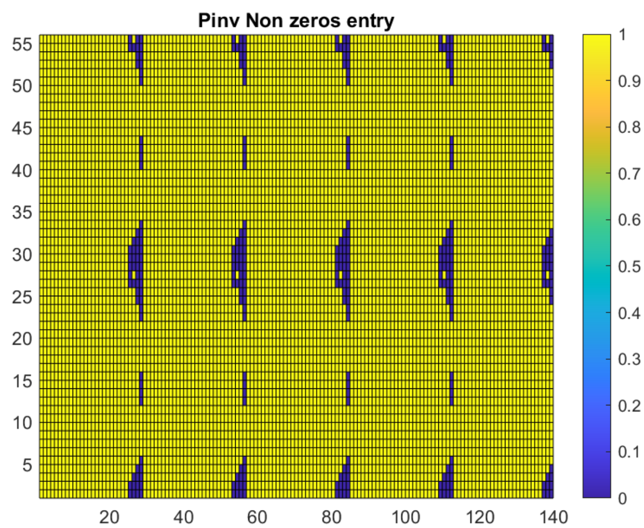
Plots and Interpretations





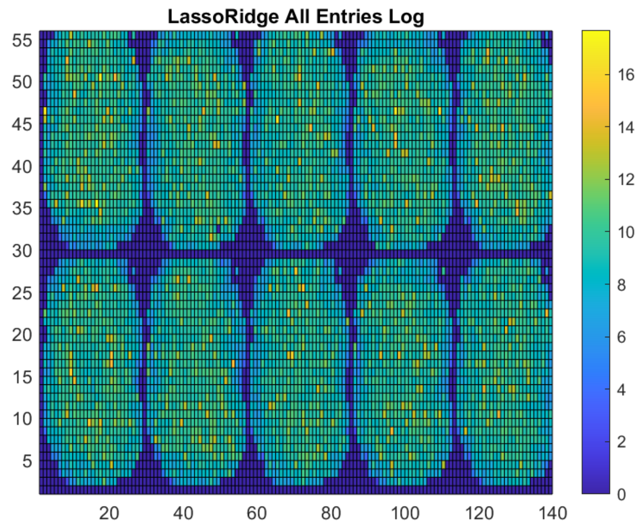
Observe that backslash produced a matrix that is dense. When it solves the matrix problem, it look for the least square projection on to the subspace, and then reduce the 2-Norm of the projection vector using a QR based algorithm.

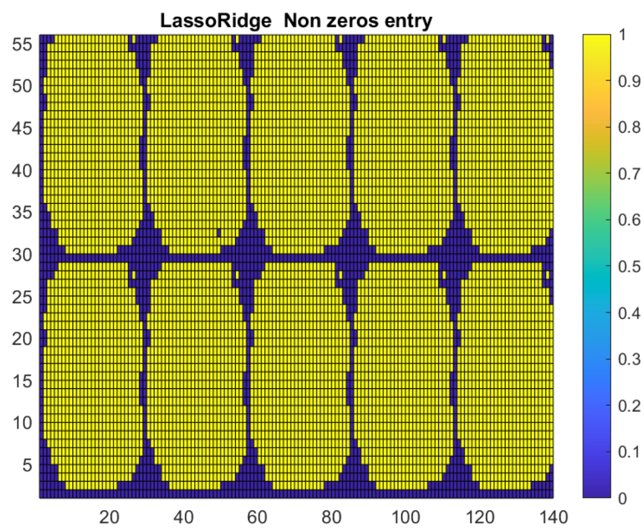




Psuedo inverse on the other hand, produces a model that contains entries with a larger variance compare to the backslash model, suggesting that it made no attempt to regularize the norm of the solution, unlike back slash.

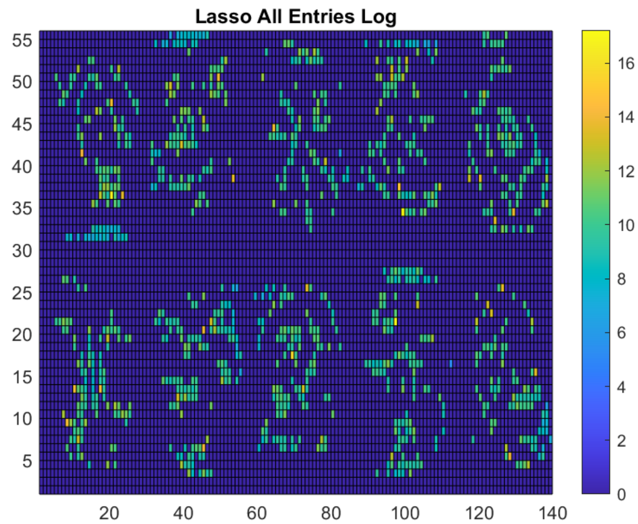
In addition observe that the exponent is much larger compare to the backslash model and it spans a larger range.

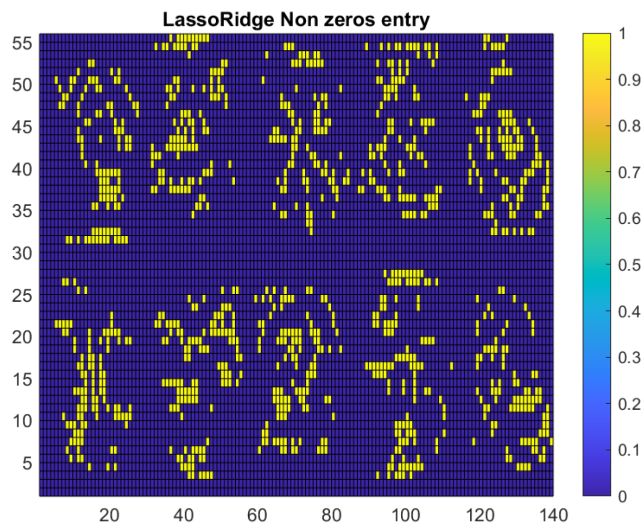




Observe that the lassoridge model, which is done by using the 'lasso" command in matlab by setting up $\alpha \approx 0$, something clsoe to zero, we obtain a model that looks very similar to the backslash.

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





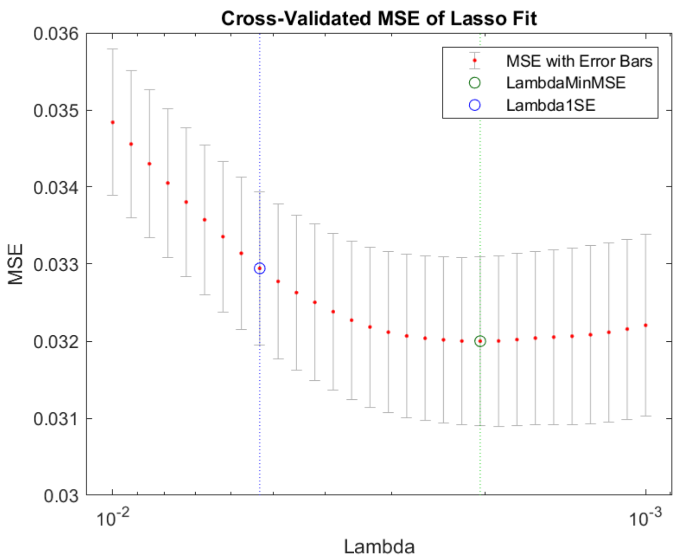
SPARSITY. The ridge regression with λ set to 0.008 produces this model, and observe that how most of the entries on the matrix X are zeroes. This is saying that the Lasso optimizer is able to throw out redundant predictors.

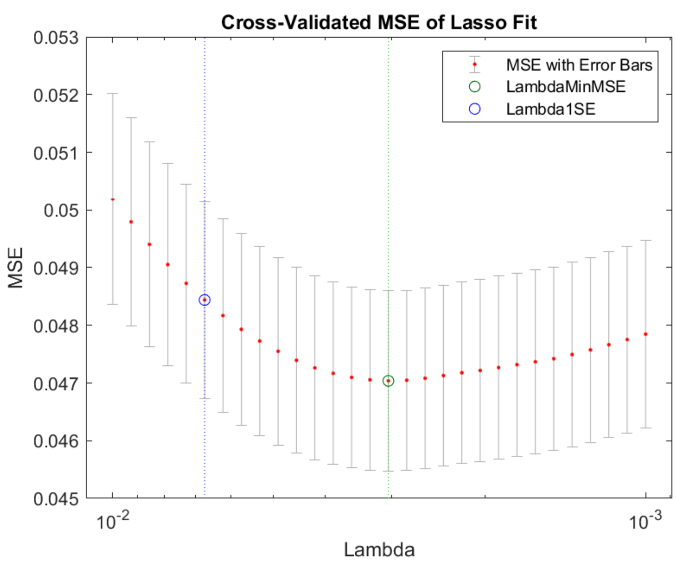
The idea here is that, if we wrote down a digit "1", then it's likely that a lot of pixels will appear together, and that is covariance, or colinearity, and this is what the Lasso optimizer is design to eliminate.

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 t he "CV" option for "lasso" to get an estimation for the MSE and the Variance of MSE.





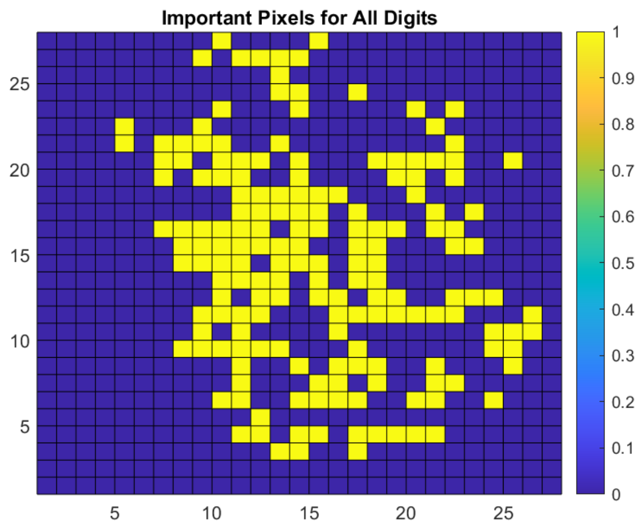
The first plot is for the digit 1 and the second one is for the digit 5. Observe that they have different optimal lambdas for smallest MSE and smallest Variance on MSE. .

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:

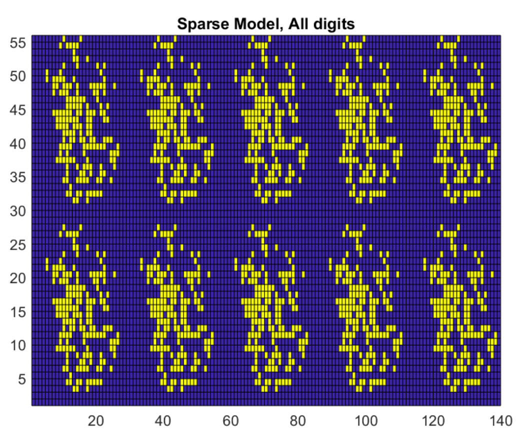


Intuitively, people will think that the important pixels should be all clustered at the center, however that is exactly not the right way, because every digits will have some strokes that cross the center, and in that case, it makes the pixels in the center most redundant.

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 in put 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.

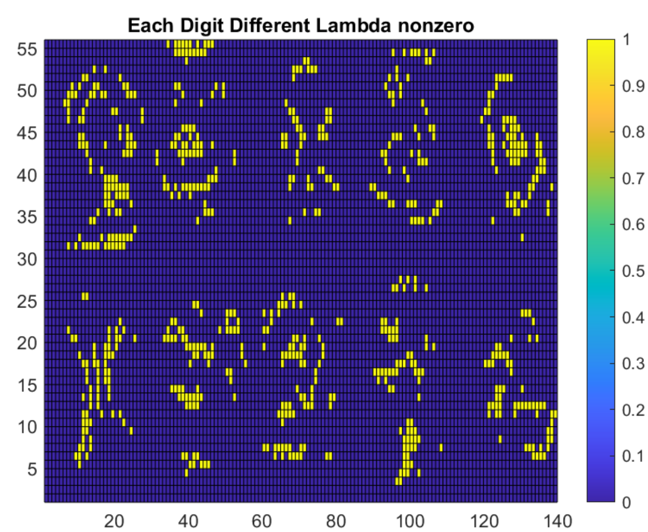
$$F = ((X)_{K,:} != 0) \circ A$$

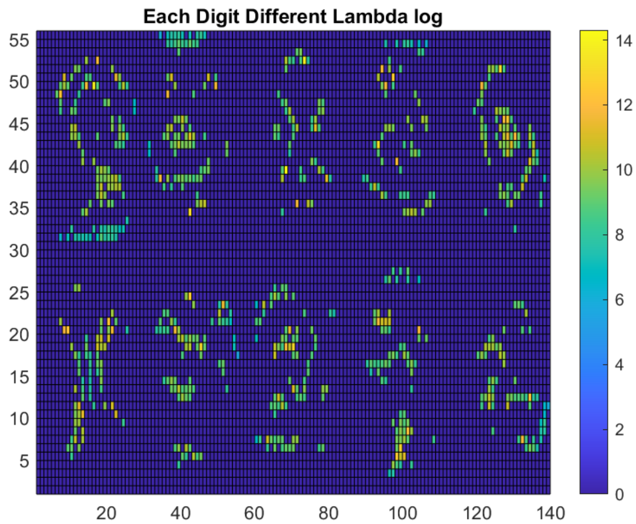
Then solves

$$x_k F = B$$

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 rading 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, visuzlize the most important pixels for all the digits.

```matlab
function [Model, Filters] = EachDigitSparseModel(datapool, Filters)
    % Theory:
    %       Use the sparse filter for each digit and then use backslash to
    %       get the model.
    % DataPool:
    %       A datapool object.
    % Filters: 28 x 28 x 10
    %       10 Filters, for each digit, the filter should come from one of
    %       the sparse model using Lasso L1 Norm.
    %

    if nargin == 1  % Compute the best filters.
        Filters = zeros(28, 28, 10);
        datapool.ApplySparseFilter(ones(28^2, 1));
        %datapool.Intercept = 0;
        for II = 1: 10
            disp(strcat("Computing Best Filter for digit: ", num2str(II)));
            Options             = optimset("display", "iter", "MaxFunEvals", 40);
            [LambdaOptimal, ~] = fminbnd(@(x) GetError(datapool, II, x), 0.00, 0.05, Options);
            [Filter, ~]         = datapool.LassoSingleDigitTrain(II, LambdaOptimal, 1);
            Filters(:, :, II)  = reshape(Filter, 28, 28);
        end
        %datapool.Intercept = 1;
    end

    Model = zeros(10, 28^2);
    for II = 1: 10
        datapool.ApplySparseFilter(Filters(:, :, II) ~= 0);
        A = datapool.DataMatrix;
        B = datapool.LabelMatrix;
        B = B(II, :)';
        Model(II, :) = (A'\B)';
    end

    function Error = GetError(datapool, digit, lambda)
        [M, stats]     = datapool.LassoSingleDigitTrain(digit, lambda, 1);
        Predicted      = M'*datapool.Afull + stats.Intercept;
        Density        = sum(M ~= 0)/784;
        CorrectAnswers = datapool.Bfull(digit, :);
        Error          = norm(Predicted - CorrectAnswers)/(1 - Density);
    end
end
```

```matlab
function [A, B, Images, Labels] = GetMatrices(toRead)
    % Read the A, B matrix from the data set as specified in the HW
    % assignment.
    % toRead:
    %   Binary variable, if it's 1, then it's the training set if it's 2
    %   then it's the validation set.

    mode           = ["train", "t10k"];
    DataFile       = strcat("data\", mode(toRead), "-images.idx3-ubyte");
    Labels         = strcat("data\", mode(toRead), "-labels.idx1-ubyte");
    [Data, Labels] = MNISTRead(DataFile, Labels);
    Images         = Data;

    [m, n, q] = size(Data);
    A         = zeros(m*n, q);
    for II = 1: size(Data, 3)
        A(:, II) = reshape(Data(:, :, II), m*n, 1);
    end

    B            = zeros(10, q);
    for II = 1: q
        if Labels(II) == 0
            B(10, II) = 1;
            continue;
        end
        B(Labels(II), II) = 1;
    end

end
```

```
%% PRODUCING VARIOUS MODELS USING DIFFERENT SOLVERS
clear variables; clc;

data        = TrainingDataPool(6000);
DataMatrix  = data.DataMatrix;
LabelMatrix = data.LabelMatrix;
%%
X1          = (DataMatrix'\LabelMatrix')';
X2          = (pinv(DataMatrix')*LabelMatrix')';
[X3, Beta3] = data.SingleLambdaLasso(0.008, 0.0001);  % Ridge Regression.
[X4, Beta4] = data.SingleLambdaLasso(0.008, 1);       % Full L1
data.Intercept = 0;
XX3         = data.SingleLambdaLasso(0.008, 1);       % L1 Lasso no intercept.
data.Intercept = 1;


%% Model Assessment
X1Score      = data.GetModelScore(X1);
X2Score      = data.GetModelScore(X2);
X3Score      = data.GetModelScore(X3, Beta3);
X4Score      = data.GetModelScore(X4, Beta4);
XX3Score     = data.GetModelScore(XX3);

%% VISUALIZING VARIOUS MODELS PRODUCED

VisualizeAllLayers(log(1./abs((X1 == 0) + X1))); colorbar; title("Backslash All Entries Log");
saveas(gcf, "backslash-log", "png");

VisualizeAllLayers(sign(abs(X1))); colorbar; title("Backslash Non zeros entry");
saveas(gcf, "backslash-nonzero", "png");

VisualizeAllLayers(log(1./abs((X2 == 0) + X2))); colorbar; title("Pinv All Entries Log");
saveas(gcf, "pinv-log", "png");

VisualizeAllLayers(sign(abs(X2))); colorbar; title("Pinv Non zeros entry");
saveas(gcf, "pinv-nonzero", "png");

VisualizeAllLayers(log(1./abs((X3 == 0) + X3))); colorbar; title("LassoRidge All Entries Log");
saveas(gcf, "lassoridge-log.png", "png");

VisualizeAllLayers(sign(abs(X3))); colorbar; title("LassoRidge  Non zeros entry");
saveas(gcf, "lassoridge-nonzero.png", "png");

VisualizeAllLayers(log(1./abs((X4 == 0) + X4))); colorbar; title("Lasso All Entries Log");
saveas(gcf, "lasso-log.png", "png");

VisualizeAllLayers(sign(abs(X4))); colorbar; title("LassoRidge Non zeros entry");
saveas(gcf, "lasso-nonzero.png", "png");

%% VARIABILITY IN LAMBDA
[B, Stats] = data.LassoSingleDigitTrainCV(1, logspace(-3, -2, 30), 1);

%%
lassoPlot(B, Stats, "PlotType", "CV");
```

```
legend("Show");
saveas(gcf, "digit-1-lasso-cv", "png")

%%
[B, Stats] = data.LassoSingleDigitTrainCV(5, logspace(-3, -2, 30), 1);

%%
lassoPlot(B, Stats, "PlotType", "CV");
legend("Show");
saveas(gcf, "digit-5-lasso-cv", "png");


%% SPARSE FILTER ALL DIGITS

Filter = VisualizeOverlayed(X4, 3); title("Important Pixels for All Digits")
saveas(gcf, "filter-alldigits", "png");
Filter  = Filter + zeros(28, 28, 10);
[X5, ~] = EachDigitSparseModel(data, Filter);
disp(strcat("Sparse model all digits scored: ", num2str(data.GetModelScore(X5)), "%"));
disp(strcat("Model Density: ", num2str(sum(sum(X5~=0))/7840)));
VisualizeAllLayers(X5 ~= 0); title("Sparse Model, All digits");
saveas(gcf, "sparse-all-digits", "png");

%% SPARSE EACH DIGITS DIFFERENT LAMBDA FOR EACH DIGITS
[X6, ~] = EachDigitSparseModel(data);
disp(strcat("Model Density: ", num2str(sum(sum(X6~=0))/7840)));
VisualizeAllLayers(X6 ~= 0);
colorbar; title("Each Digit Different Lambda nonzero");
saveas(gcf, "each-digit-diff-lambda-log", "png");
VisualizeAllLayers(log(1./abs((X6 == 0) + X6)));
colorbar; title("Each Digit Different Lambda log");
saveas(gcf, "each-digit-diff-lambda-nonzero", "png");
X6Score = data.GetModelScore(X6);
disp(strcat("Model Score: ", num2str(X6Score)));
```

```
clc; clear variables;
% Load the Training set.

N    = 60000;  % When N > 784, over determined, when N < 784, underdetermined.
data = TrainingDataPool(N);
A    = data.DataMatrix;
B    = data.LabelMatrix;

%% DIRECT SOLVE FOR A LINEAR MODEL
% Linear Mapping: R^784 --> R^10, then 10 by 784, then XA = B is the model
% description..
X1   = (A'\B')';
X2   = (pinv(A')*B')';


%% MULTI LASSO TRAIN
figure;
N = 1000;
for II = 1: 10
    data.Scramble(N);  % Batch size
    Lambdas     = linspace(0.001, 0.01, 10);
    ModelSeries = data.MultiLambdaLasso(Lambdas);
    Errors      = data.ModelsErrorsTypeI(ModelSeries);
    Errors      = Errors./(10000*2);  % Across 1k test data set.
    plot(Lambdas, Errors); hold on;
    disp(strcat("Training Lasso: ", num2str(II)));
end
% Lambda Optimal: 0.004;


%% LASSO MODEL TRAIN ALL
data.Scramble(6000);
data.ApplySparseFilter(ones(28, 28));  % CLEAR THE FILTERS!
%%
[X3, StatsX3] = data.SingleLambdaLasso(0.004);
%%
[X4, StatsX4] = data.SingleLambdaLasso(0.004, 0.8);
%%
[X5, StatsX5] = data.SingleLambdaLasso(0.004, 0.5);
%%
% [X6, StatsX6] = data.RobustFit();
[X6, StatsX6] = data.SingleLambdaLasso(0.004, 0.01);
%%
[X7, StatsX7] = data.SingleLambdaLasso(0.01);

%% SCORING BOARD
close all;
Score = data.GetModelScore(X1);
VisualizeOverlayed(X1); title("X1");
disp(strcat("BackSlack Model score: ", num2str(Score)));

Score = data.GetModelScore(X2);
VisualizeOverlayed(X2); title("X2");
disp(strcat("Puesdo Inverse Model Score: ", num2str(Score)));
```

```
Score = data.GetModelScore(X3);
VisualizeOverlayed(X3); title("X3");
disp(strcat("Lasso Single score: ", num2str(Score)));

Score = data.GetModelScore(X4);
VisualizeOverlayed(X4); title("X4");
disp(strcat("Lasso With Alpha 0.8: ", num2str(Score)));

Score = data.GetModelScore(X5);
VisualizeOverlayed(X5); title("X5");
disp(strcat("Lasso With Alpha 0.5: ", num2str(Score)));

Score = data.GetModelScore(X6);
VisualizeOverlayed(X6); title("X6");
disp(strcat("Ridge Regression: ", num2str(Score)));

Score = data.GetModelScore(X7);
VisualizeOverlayed(X7); title("X7");
disp(strcat("Extreme Lasso Regression: ", num2str(Score)));
%% SPARSE MODEL VISUALIZATION!!!

VisualizeAllLayers(X7); title("X7")
%%
VisualizeAllLayers(X4); title("X4")
%%
VisualizeAllLayers(X3); title("X3")


%% SPARSE MODEL TRAINING!!!
clc;
data.Scramble(6000);
Filter = VisualizeOverlayed(X3, 5);
disp(strcat("Filter Density: ", num2str(sum(sum(Filter))./(28^2))));
data.ApplySparseFilter(Filter);
A        = data.DataMatrix;
B        = data.LabelMatrix;
X1Sparse = (A'\B')';

VisualizeAllLayers(X1Sparse);
Score = GetModelScore(data, X1Sparse);
disp(strcat("BackSlash Sparse Model score: ", num2str(Score)));

%% FULL SPARSE MODEL FOR EACH DIGITS!
clc;
data.Scramble(6000);
[X8, X8Filters] = EachDigitSparseModel(data);
disp(strcat("Filter Density: ", num2str(sum(sum(X8~=0))./(10*28^2))));
VisualizeAllLayers(X8);

Score = data.GetModelScore(X8);
disp(strcat("BackSlash Sparse Model each digit: ", num2str(Score)));

%% HELPER FUNCTIONS
```

```
classdef TrainingDataPool < handle

    properties
        Images;     % Image Tensor Full
        Labels;     % Image Label vector full
        Afull;      % Reshaped Data Matrix full
        Bfull;      % Image lable matrix full

        % For training.
        DataMatrix;
        LabelMatrix;
        ChosenSubset;

        A = nan;    % Flatten Tensor
        b = nan;    % Flatten Label vector

        TestMatrixB; % 1k label matrix.
        TestMatrixA; % 1k Data matrix.

        Intercept = 1;  % An extra option for Lasso Model, if this is one,
                        % Then all lasso model are going to have intercept.

    end

    methods

        function this = TrainingDataPool(N)
            % Constructor.
            [this.Afull, this.Bfull, this.Images, this.Labels] = ...
            GetMatrices(1);
            if N > length(this.Labels)/2
               error("No! Reserve 50% of them for cross validations.");
            end
            this.Scramble(N);
            [this.TestMatrixA, this.TestMatrixB, ~, ~] = ...
            GetMatrices(2);

        end

        function void = Scramble(this, N)
            % Scramble all the data and labels for training, N of them.
            RandomIdx         = randperm(length(this.Labels));  % No repreatition
            this.ChosenSubset = RandomIdx(1: N);
            this.DataMatrix   = this.Afull(:, this.ChosenSubset);
            this.LabelMatrix  = this.Bfull(:, this.ChosenSubset);
            void              = nan;
        end

        % Swap to Binary Label with +1, -1 for the label matrix.
        function void = SwapLabelBinaryMode(this)
            B                = this.Bfull(:, this.ChosenSubset);
            B                = -(B == 0) + B;
            this.LabelMatrix = B;
            void             = 0;
```

```matlab
        end

        % Swap back to 0, 1 label.
        function void = SwapLabelUnaryMode(this)
            this.LabelMatrix  = this.Bfull(:, this.ChosenSubset);
           void               = 0;
        end

        function [A, b] = FlattenTensor(this)
            % Get a matrix A, such that, solving Ax = b gives x that is a
            % flattened linear model of size 784 by 10. This is for using
            % lasso that does regression all digits at the same time.
            if ~isnan(this.A)
               A = this.A;
               b = this.b;
               return
            end
            B = this.LabelMatrix;
            A = kron(eye(10), this.DataMatrix');
            b = reshape(this.LabelMatrix', size(B, 1)*size(B, 2), 1);

        end

        % Recover the 7840 by 1 matrix to a 10 by 784 model.
        function X = VectorRecover(this, Vector)
            % Recover vectorized linear model from the flattened vector.
            X = reshape(Vector, size(this.Images, 1)*size(this.Images, 2), 10);
            X = X';
        end


        % Given a model 10 by 764 by, and get the prediction made by this
        % model when we multiply it on the right handside by the
        % DataMatrix.
        function [Predicted, Error] = Type1ModelPredictDiscrete(this, model)
            Intercept = zeros(10, 1);
            [Predicted, Error] = Type1ModelPredictDiscreteIntercept(this, model, Intercept);
        end

        % Predict the 10 by 784 model with an intercept vector.
        function [Predicted, Error] = Type1ModelPredictDiscreteIntercept(this, model, intercept)

            % Predict on ALL DATA with a given Type I model.
            Predicted = model*this.TestMatrixA + intercept;
            NewMatrix = zeros(size(Predicted));

            for ColIndex = 1: size(NewMatrix, 2)
               [~, Idx]                 = max(Predicted(:, ColIndex));
               NewMatrix(Idx, ColIndex) = 1;
            end
            Predicted = NewMatrix;
            Error     = sum(sum(abs(Predicted - this.TestMatrixB)));
        end

```

```
        % Get the percentable of correctly predictor lable on the MNIST
        % test set. the model given 10 by 784 matrix.
        function Score = GetModelScore(this, model, intercept)
                if nargin == 2
                   intercept = zeros(10, 1);
                end
                Predicted  = this.Type1ModelPredictDiscreteIntercept(model, intercept);
                ModelScore = sum(sum(abs(Predicted - this.TestMatrixB)));
                Score      = 1 - ModelScore./(2*size(this.TestMatrixB, 2));
        end

        function Errors = ModelsErrorsTypeI(this, models)
            % Given a series of linear model, return all the errors on prediction.
            % Models:
            %       10 by 784 by N

            Errors = zeros(1, size(models, 3));
            for II = 1: size(models, 3)
                [~, Error] = this.Type1ModelPredictDiscrete(models(:, :, II));
                Errors(II) = Error;
            end
        end

        function ModelSeries = MultiLambdaLasso(this, Lambdas)
            % Given multiple lambdas, returns a series of 10 by 764
            % matrices for each of the given lambdas.
            [m, n] = size(this.Images(:, :, 1));
            p      = length(Lambdas);
            A      = this.DataMatrix;
            B      = this.LabelMatrix;
            Models = zeros(10, m*n, p);

            for II = 1: 10  % 10 rows.
                % [Outputs, ~] = lasso(A', B(II, :)', "lambda", Lambdas);
                [Outputs, ~] = this.LassoSingleDigitTrain(II, Lambdas, 1);

                for JJ = 1: length(Lambdas)
                   Models(II, :, JJ) = Outputs(:, JJ);
                end
            end
            ModelSeries = Models;
        end

        function [Model, Intercept] = SingleLambdaLasso(this, lambda, alpha)
            % Given a value for lambda, get a 10 by 784 model for all the
            % all the digits with the same lambda.
            if nargin == 2
               alpha = 1;
            end
            [m, n] = size(this.Images(:, :, 1));
            A      = this.DataMatrix;
            B      = this.LabelMatrix;
            Model  = zeros(10, m*n);
            Intercept  = zeros(10, 1);
```

```matlab
            for II = 1: 10                  % 10 rows.
                [Outputs, Info]  = this.LassoSingleDigitTrain(II, lambda, alpha);
                Model(II, :) = Outputs;
                if this.Intercept == 1
                    Intercept(II) = Info.Intercept;
                end
            end
        end

        % Train and get model for a single digit.
        % This is a subroutine that got repeatedly use hence it's been
        % factored out here.
        function [SingleDigitModel, Stats] = LassoSingleDigitTrain(this, idx, lambdas, alpha)
            A                = this.DataMatrix;
            B                = this.LabelMatrix;
            [Outputs, Info]  = lasso(A', B(idx, :)', ...
                "lambda", lambdas, "alpha", alpha, "Intercept", boolean(this.Intercept));
            SingleDigitModel = Outputs;
            Stats            = Info;
        end

        % Train a single digit model with given regularizer.
        function [SingleDigitModel, Stats] = LassoSingleDigitTrainCV(this, idx, lambdas, alpha)
            A                = this.DataMatrix;
            B                = this.LabelMatrix;
            [Outputs, Info]  = lasso(A', B(idx, :)', "lambda", lambdas, "alpha", alpha, "CV", 20);
            SingleDigitModel = Outputs;
            Stats            = Info;
        end

        % Perform Robust fit on all digits.
        function [Model, Stats] = RobustFit(this)
            [m, n] = size(this.Images(:, :, 1));
            A      = this.DataMatrix;
            B      = this.LabelMatrix;
            Model  = zeros(10, m*n);
            Stats  = cell(1, 10);
            for II = 1: 10  % 10 rows.
                [Outputs, Info] = robustfit(A', B(II, :)', [], [], "off");
                Model(II, :) = Outputs;
                Stats{II}    = Info;
                disp("Robust fit...")
            end
        end

        % Filter out things with the sparse model
        % Filter is a binary matrix.
        % Return a sparse data matrix and lable matrix, replace the field
        % with the sparse data matrix, without changing the size of it.
        % (So it's full of zeroes in it. )
        % Call it without any argument and it will restore dense data
        % matrix.
        function [SparseDataMatrix] = ApplySparseFilter(this, Filter)
            if isequal(size(Filter), [28, 28])
```

```
                Filter = reshape(Filter, 28^2, 1);  % Shape to column vector.
            end
            AChosen           = this.Afull(:, this.ChosenSubset);
            this.DataMatrix   = Filter.*AChosen;
            SparseDataMatrix  = this.DataMatrix;
        end
    end
end
```

```
function Layers = VisualizeAllLayers(model)


    Layers = zeros(28, 28, 10);
    for II = 1: 10
       Layers(:, :, II) = reshape(model(II, :), size(Layers, [1, 2]));
    end

    DrawingBoard = zeros(28*2, 28*5);
    for II = 1: 10
        III = floor((II - 1)/5);
        JJJ = mod((II - 1), 5);
        DrawingBoard(III*28 + 1: (III + 1)*28, JJJ*28 + 1: (JJJ + 1)*28)...
            = Layers(:, :, II);
    end
    figure; pcolor(DrawingBoard);

end
```

```matlab
function Filter = VisualizeOverlayed(model, threshold)  % Visualize the model.
    if nargin == 1
        threshold = 5;
    end

    model    = model ~= 0;  % Important!
    SummedUp = zeros(1, size(model, 2));
    for II = 1: 10
       SummedUp = SummedUp + model(II, :);
    end
    SummedUp = reshape(SummedUp, 28, 28);
    Filter   = SummedUp >= threshold;   % Threshold
    figure;
    pcolor(Filter); colorbar;
end
```