Amath563: Linear Regression for MNIST Data

Helena Liu (1922076), April 15th, 2020

Abstract: Linear regression is applied to the MNIST dataset. This framework is trained and tested on different sets of data. The error rate is significantly better than random guess but is higher than the error rate obtained using nonlinear models reported in the literature. Linear regression offers encouraging results and its limitations are discussed.

I. Introduction and Overview

Constructing a model with predictive power is at the core of machine learning. In the time of big data, high-dimensional datasets often make the relationship between independent and dependent variables elusive. Models can be applied to high dimensional variables to describe their relationships. Linear regression is often the starting point due to its simplicity. In this work, linear regression is applied to the high dimensional MNIST dataset and results are discussed.

II. Theoretical Background

A) Linear Regression

Suppose there are m datapoints $\in \mathbb{R}^d$ and outcomes $\in \mathbb{R}^n$, and one would like to model how the datapoints and their outcomes are related, one of the simplest approach is to assume they are related linearly. Each of the d dimensions in the datapoints can be viewed as an independent variable and each of the n dimensions in the outcome can be viewed as a dependent variable. One key assumption of a linear model is that the amount of change in a dependent variable due to one independent variable does not depend on the other variables.

To formulate a linear model, one can construct a matrix A $\in \mathbb{R}^{mxd}$, where each row corresponds to a single datapoint, a matrix b $\in \mathbb{R}^{mxn}$, where each row corresponds to a single outcome, and a matrix x $\in \mathbb{R}^{dxn}$ consisting of the linear coefficients. Hence, a linear relationship between datapoints A and outcomes b can be formulated by solving the system $Ax = b$.

It often happens that a linear system of equations has infinitely many solutions. This can happen when $m>d$, i.e. we have more equations than unknowns, and such system is called overdetermined. There may not exist an x that satisfies all m equations. This can be addressed using a least square criterion, where instead of requiring that $Ax = b$, x is picked to minimize $J = \|Ax - b\|_2^2$. These methods will be discussed in section III.

B) Promoting Sparsity

Using as few terms as possible in a model is often desirable due to better interpretability and generalization. Sparsity Promoting algorithms will be discussed in Section III. In general, these methods involve an additional L1-term in the cost function,

$$J = \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1 , \quad (1)$$

which punishes dense solution. λ is a parameter that allows one to control the importance of sparsity promotion relative to small least squares error.

A) Cross-Validation

Another strategy for promoting sparsity is to use cross-validation. Cross-validation involves testing the trained model on data that was not used during training. It gives one a general sense of how well the model performs on unseen data. It also allows one to select the number of model terms (or any other hyperparameters) for adequate description of new data by seeing how that affects the testing error.

III. Algorithm Implementation and Development

This section outlines the methods for constructing $Ax = b$ from the MNIST dataset and then using various least square and regularization methods to solve the system. All MATLAB modules are provided in **Appendix A**.

A) Data description

MNIST dataset consists of 28-by-28 images of handwritten digits and labels denoting the correct digit for each image. The dataset is split into training and testing sets. This split conveniently allows one to solve for x in $Ax = b$ using the training set, as discussed in Section III.B, and then test the model on the testing set, as discussed in Section III.C.

The $Ax = b$ system is constructed by letting A be the reshaped MNIST images, where each row pertains to one image and each column corresponds to one of the 784 pixels. Each entry in A stores the intensity of the corresponding image and pixel. The correct label for each image is stored in each row of b using one-hot encoding. In matrix b, the j^{th} column of a row is set to 1 if j is the label of the corresponding image. Matrix x therefore consists of coefficients that relates the pixel intensity to the correct label for each image.

B) Least Squares and Regularization

The simplest ways to solve for $Ax = b$ is to use pseudoinverse, SVD or the QR-based backslash, which applies QR decomposition. However, these methods do not involve regularization, which promotes sparsity and small coefficient values. The benefit of sparsity has been discussed in Section II.B. Small coefficient values can be advantageous as it reduces the sensitivity of the outcome to small perturbations in data.

To promote sparsity, lasso regularization is used, which promotes sparsity by minimizing the cost function in (1). To punish large coefficient values, ridge regression (or Tikhonov regularization) is used by adding a L2 penalty term and minimize the following cost function

$$J = \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1 , \quad (2)$$

To simultaneously promote both sparsity and small coefficient L2-norm, elastic net regularization is used, which minimizes the following the following objective function,

$$J = \frac{1}{2}\|Ax - b\|_2^2 + \lambda\left(\frac{1-\alpha}{2}\|x\|_2^2 + \alpha\|x\|_1\right). \quad (3)$$

The additional parameter α allows one to control the relative importance of minimizing L1 vs L2 norm, where α=1 coincides with lasso [1]. The lower the alpha, the less relative importance on sparsity, as illustrated in **Figure 2**. Elastic net is implemented in MATLAB using the lasso function but with α set to less than 1.

All these methods are applied to training data only, meaning A and b are constructed from training images and labels.

C) Cross-Validation

The linear model obtained from training data is applied to test data, which allows the validation of the trained data from unseen data. This is done by multiplying coefficient x, obtained from Section III.B, with A_{test}, which is the matrix constructed from test set images. The predicted label is then extracted from this product by finding the index of the maximum value in each row. The predicted label is then compared with the true label to obtain the error rate:

$$Error\ Rate = \frac{Number\ of\ (Predicted\ Test\ Label \neq True\ Test\ Label)}{Number\ of\ Test\ Images}. \quad (4)$$

Usually k-fold cross validation, which divides the training data into k sets of equal size and averages the parameters obtained across the training sets, is used to avoid overfitting. However, k-fold cross validation was tried but not included in the results because it was found to worsen the performance. This could be because the number of parameters (7840) relative to the number of datapoints (60000) is large, so dividing the training data would further raise this ratio.

IV. Computational Results

A) Train and Test for All Digits

In this subsection, coefficient matrix x is obtained by solving $Ax = b$ for images and labels in the training set using the procedure described in Section III.B. The model is then tested for all images and labels in the testing set using the procedure described in Section III.C.

Only the most important pixels, i.e. the rows in matrix x with the highest infinity norm, is used to predict the label for test data and obtain the error rate in (1) using the procedure in Section III.C. Other ranking measures, such as the Euclidean norm, have also been tried but resulted in higher error rate than infinity norm. This is probably because infinity norm is better at capturing pixels with high weight for a few digits only. The error rate is computed for various numbers of most important pixels used (from 100 pixels to 775 pixels in increment of 25 pixels) and plotted for different solvers in **Figure 1**:

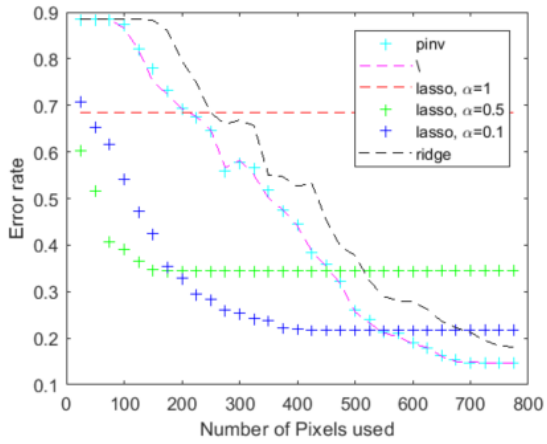

Figure 1: Error rate versus the number of pixels used for various solvers.

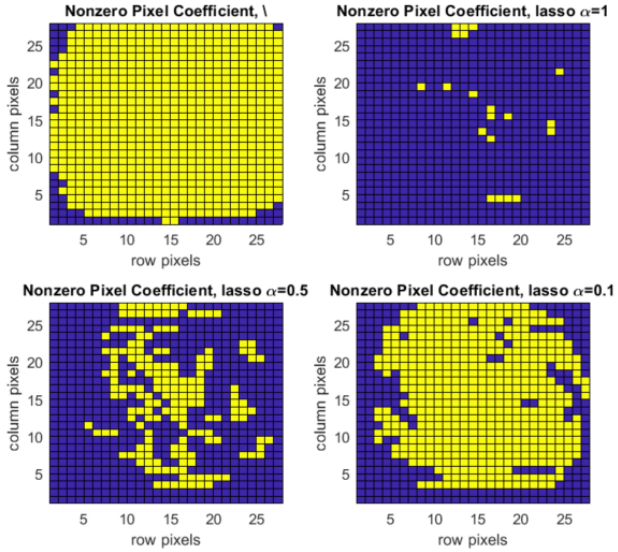

Figure 2: Illustration of the pixel coefficients x that are zero (blue) and nonzero (yellow) obtained using the QR-based backslash and lasso. Lasso results in more zero terms and increasing α reduces sparsity.

When 775 pixels (almost all pixels) are used, pseudoinverse and the QR-based backslash achieve the lowest error rate (14%), which is just slightly higher than the 12% error rate reported in [2] when an additional bias term was used. However, if fewer pixels are enforced, then lasso (or elastic net) regularization with α=0.1 achieves better error rate than other methods and its near optimal error rate at about 300 pixels.

All test errors do not increase with more terms are added, so there is no clear sign of overfitting, which suggests that the system $Ax = b$ is complex and requires high dimension to achieve an optimal error rate. This is probably why when lasso is used with a high α value (high importance in sparsity) performs much worse than other algorithms and the error stops decline even when additional terms are added, since not enough terms are employed to adequately explain the complexity (i.e. underfitting). The sparsity of x from lasso is illustrated in **Figure 2**. However, lasso with α=0.1 achieves far better results than other methods when only 300 terms are used. This is probably because simultaneously punishing density and large coefficient values can both promote efficiency and avoid large coefficients that may result in sensitivity to noise.

B) Train and Test for Individual Digits

Training and testing on each of the 10 digits are done separately in this section, meaning that for each digit j, b is the j^{th} column of the b in Section IV.A. Thus, an entry of 1 denotes the corresponding image belongs to the digit of interest and 0 otherwise. Matrix A remains the same as in the previous section. This effectively becomes a binary classification problem. The predicted label is set to 1 if the output is greater than 0.5 and set to 0 otherwise. Results are summarized in **Figure 3**. Two of the 10 digits are selected as examples for illustration. Results for the other digits follow similar trend.

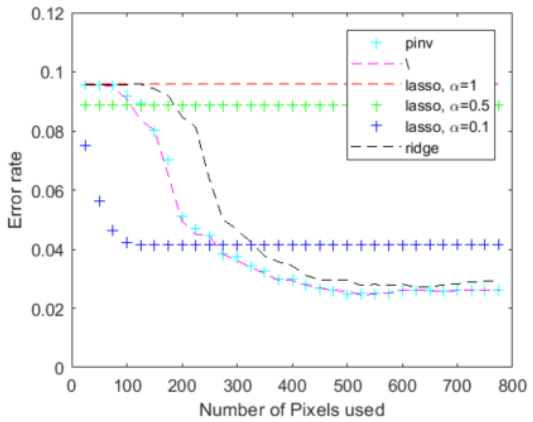

*Figure 3a: Error rate versus number of pixels used for **digit 6***

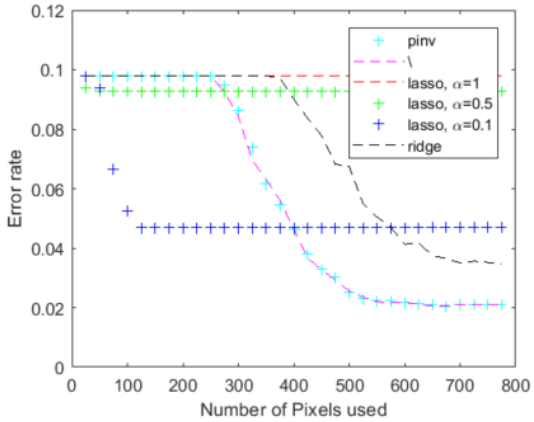

*Figure 3b: Error rate versus number of pixels used for **digit 0***

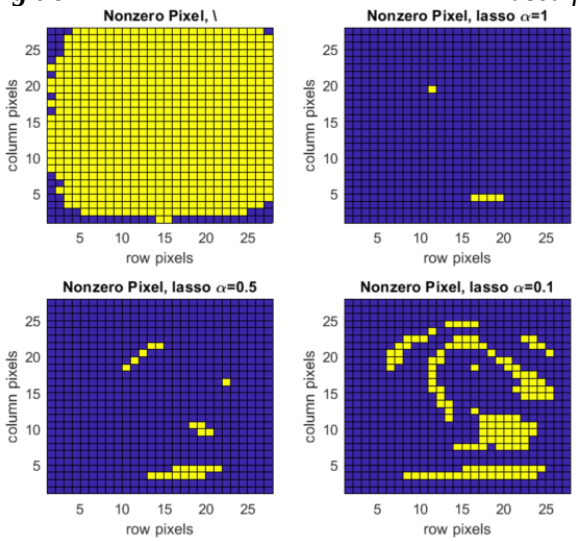

*Figure 4: Illustration of the pixel coefficients that are zero (blue) and nonzero (yellow) using the QR-based backslash and lasso for **digit 6**. Lasso promotes sparsity and this sparsity reduces with α.*

When almost all terms are used, the error rate is much better than that obtained in Section IV.A, possibly because the classification is between two classes instead of ten. Similar to the previous section, there is no sign of overfitting, as the test error rates do not increase as more terms are added. However, there is a clear sign of underfitting for the lasso method, as the error stops decreasing at far fewer terms compare to the previous section. This could probably be due to the increased sparsity in x when digits are trained individually, which can be seen by comparing **Figure 4** and **Figure 2**. In addition, the comparing **Figure 1** and **Figure 3** suggests that the methods achieve their optimal error with fewer terms when digits are trained individually. This may be because pixels are ranked based on their specific contribution to a digit.

C) Discussion of $Ax = b$

As mentioned in Section II, $Ax = b$ captures the linear relationship between independent and dependent variables. In the context of MNIST dataset, each pixel would be an independent variable that makes a linear contribution to the identity of the label. The error rate achieved is

lower than that of a random guess (1 / number of classes), meaning the model is capturing some sort of relationship between the independent and dependent variables.

One major caveat with using a linear model for the MNIST dataset is that the model is not scale and translational invariant. In other words, where in the image a digit will be written and how big it is written can have significant effect on the outcome, since they determine which pixels will be occupied. This issue would require the prediction to not look at each pixel independently, but in relation to their neighbour pixels. However, this cannot be accomplished with a linear model, because as mentioned in Section II, one of the key assumptions is that the contribution of an independent variable is independent of other variables. This is possibly why nonlinear models used in [2] performed better than the linear model.

Since nonlinear models are generally used for this dataset [2] for adequate performance, to achieve linear separability, a much higher dimension is required according to Cover's theorem [3]. This demand for high dimension when a linear model is used is probably why there is not sign of overfitting, and there is a striking sign of underfitting when lasso, which reduces the dimension of the model even more, is used.

V. Summary and Conclusions

Through this project, we become aware of some major limitations with linear models, such as its assumption of independent contributions between variables. Despite its major caveats, linear regression offered promising prediction accuracy for the MNIST dataset. Due to the complexity of the relationship between the pixels and the label, test error is at its lowest when almost all terms are involved and there is no sign of underfitting. Moreover, sparsity promotion can allow better prediction accuracy when fewer terms are used, but the relative importance of sparsity promotion (parameter α in lasso) needs to be controlled carefully in order to avoid underfitting.

VI. References:

[1] MathWorks. *Lasso and Elastic Net – MATLAB & Simulink*. [Online]. Available: https://www.mathworks.com/help/stats/lasso-and-elastic-net.html. [Accessed:15-Apr-2020].

[2] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE, 86(11):2278-2324*, November 1998.

[3] Cover, T. M. Geometrical and Statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE Transactions on Electronic Computers, EC-14 (3):326-334, March 1965.

Appendix A: MATLAB Functions Used

The following modules and documentations are provided in MATLAB R2017b by MathWorks.

```
function [B,stats] = lasso(X,Y,varargin)
%LASSO Perform lasso or elastic net regularization for linear regression.
%   [B,STATS] = lasso(X,Y,...) Performs L1-constrained linear least
%   squares fits (lasso) or L1- and L2-constrained fits (elastic net)
%   relating the predictors in X to the responses in Y. The default is a
%   lasso fit, or constraint on the L1-norm of the coefficients B.
%
%   Positional parameters:
%
%     X               A numeric matrix (dimension, say, NxP)
%     Y               A numeric vector of length N
%
%   Optional input parameters:
%
%     'Alpha'         Elastic net mixing value, or the relative balance
%                     between L2 and L1 penalty (default 1, range (0,1]).
%                     Alpha=1 ==> lasso, otherwise elastic net.
%                     Alpha near zero ==> nearly ridge regression.
%     'Lambda'        Lambda values. Will be returned in return argument
%                     STATS in ascending order. The default is to have LASSO
%                     generate a sequence of lambda values, based on 'NumLambda'
%                     and 'LambdaRatio'. LASSO will generate a sequence, based
%                     on the values in X and Y, such that the largest LAMBDA
%                     value is just sufficient to produce all zero coefficients B.
%                     You may supply a vector of real, non-negative values of
%                     lambda for LASSO to use, in place of its default sequence.
%                     If you supply a value for 'Lambda', 'NumLambda' and
%                     'LambdaRatio' are ignored.
```

```
function b = ridge(y,X,k,flag)
%RIDGE Ridge regression.
%   B1 = RIDGE(Y,X,K) returns the vector B1 of regression coefficients
%   obtained by performing ridge regression of the response vector Y
%   on the predictors X using ridge parameter K.  The matrix X should
%   not contain a column of ones.  The results are computed after
%   centering and scaling the X columns so they have mean 0 and
%   standard deviation 1.  If Y has n observations, X is an n-by-p
%   matrix, and K is a scalar, the result B1 is a column vector with p
%   elements.  If K has m elements, B1 is p-by-m.
%
%   B0 = RIDGE(Y,X,K,0) performs the regression without centering and
%   scaling.  The result B0 has p+1 coefficients, with the first being
%   the constant term.   RIDGE(Y,X,K,1) is the same as RIDGE(Y,X,K).
```

```
function X = pinv(A,tol)
%PINV   Pseudoinverse.
%   X = PINV(A) produces a matrix X of the same dimensions
%   as A' so that A*X*A = A, X*A*X = X and A*X and X*A
%   are Hermitian. The computation is based on SVD(A) and any
%   singular values less than a tolerance are treated as zero.
```

Appendix B – MATLAB CODE

The MATLAB code used in this assignment have been uploaded to GitHub: https://github.com/Helena-Yuhan-Liu/Inferring-Structure-of-Complex-Systems.

```
%% Setup data

% this code loads data 'digit*.mat' generated by converter.m wrtten by
% Dr. ruslan Salakhutdinov and downloaded from
%   http://www.utstat.toronto.edu/~rsalakhu/code_DBM/

% Then, construct the following for train and test sets:
% A: n by 784
% x: 784 by 10
% y: n by 10

format compact;
clc; clear;

% Initialize Ax=y
np=784;
ntrain=60000; ntest=10000;
ytrain=NaN(ntrain,10);
ytest=NaN(ntest,10);
Atrain=NaN(ntrain,np); % rank(Atrain) = 712
Atest=NaN(ntest,np); % rank(Atest) = 661

% iterate through the 9 digits and extract training and test data
itrain=0; itest=0;
for ii=0:9
    yi=zeros(1,10);     % each column in label matrix y
    if ii==0
        yi(10)=1; idx=10;
    else
        yi(ii)=1; idx=ii;
    end

    % load data
    load(['./digitData/digit' num2str(ii) '.mat']); % training data
    itrain = itrain+size(D,1);
    Atrain(itrain-size(D,1)+1:itrain,:) = D; % add data to A
    ytrain(itrain-size(D,1)+1:itrain,:) = repmat(yi,[size(D,1) 1]);

    load(['./digitData/test' num2str(ii) '.mat']); % repeat for test
    itest = itest+size(D,1);
    Atest(itest-size(D,1)+1:itest,:) = D;
    ytest(itest-size(D,1)+1:itest,:) = repmat(yi,[size(D,1) 1]);
end

disp('Done loading data');

% % visualize an example image
% figure; image(reshape(D(123,:),[28, 28])');

%% Different Regression
% Judging from the dimension this is an overdetermined system, since
% the number of equation n far exceeds the unknowns available

% several different lasso() are tried
% Alpha controls how much sparsity versus small 2norm should be promoted
% Alpha=1 corresponds to lasso, alpha=0 is similar to ridge,
```

```
% anything in between is elastic net
% The objective function is can be found at
%   https://www.mathworks.com/help/stats/lasso-and-elastic-net.html

% Ridge (Tikhonov regularization) punishes L2 norm

disp('begin regression');

x1 = pinv(Atrain)*ytrain;       % pseudoinverse

x2 = Atrain\ytrain;     % QR-based backslash

x3=NaN(np,10);      % Lasso with alpha=1
for kk=1:10
    x3(:, kk) = lasso(Atrain,ytrain(:,kk),'Lambda',0.1); % increasing lambda was worse
off
end

x4=NaN(np,10);      % Lasso with alpha=0.5
for kk=1:10
    x4(:, kk) = lasso(Atrain,ytrain(:,kk),'Lambda',0.1,'Alpha', 0.5);
end

x5=NaN(np,10);      % Lasso with alpha=0.1
for kk=1:10
    x5(:, kk) = lasso(Atrain,ytrain(:,kk),'Lambda',0.1,'Alpha', 0.1);
end

x6=NaN(np,10);      % Ridge (L2) regularization
for kk=1:10
    x6(:, kk) = ridge(ytrain(:,kk), Atrain(:,2:end), 0.5, 0);
end


%% Save data

fname1='x_all.mat';
save(fname1, 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', ...
    'Atrain', 'Atest', 'ytrain', 'ytest','ntest','ntrain');

%% Rank pixel contribution and test

npix_list = 25:25:780;
alg_list = 1:6;
alg_name = {'pinv', '\', 'lasso, \alpha=1',...
     'lasso, \alpha=0.5', 'lasso, \alpha=0.1', 'ridge'};
c_list = {'c', 'm', 'r', 'g', 'b', 'k', 'y'};

% Get error rate vs number of top pixels used
ErrRate = zeros(length(alg_list),length(npix_list));
Err2 = ErrRate;
for ii = 1:length(alg_list)
    for jj = 1:length(npix_list)
        npix = npix_list(jj); algn = alg_list(ii);
        eval(['x = x' num2str(algn) ';']);
        xE = max(abs(x), [], 2); % Tried 1-norm, 2-norm, but turns out inf-norm works
better in general
        [~, ixE] = sort( xE, 'descend' );
        x_red=zeros(size(x)); x_red(ixE(1:npix),:)=x(ixE(1:npix),:);

        % Look at test error only, no one cares about training error
        [~, labPred] = max(Atest*x_red,[],2); [~, labTrue] = max(ytest,[],2);
        ErrRate(ii,jj) = sum(labPred~=labTrue)/ntest;
```

```
        Err2(ii,jj) = norm(ytest-Atest*x_red)/norm(ytest);
    end
end

% Error rate plot
% seems like pinv and \ achieves better results with large n, but lasso has
% lower pareto optimal
figure;
for ii=alg_list
    if ii==4 || ii==5 || ii==1
            mstyle = '+';
        else
            mstyle = '--';
        end
    plot(npix_list, ErrRate(ii,:), [c_list{ii} mstyle]);
    if ii==1; hold on; end
end
hold off; xlabel('Number of Pixels used'); ylabel('Error rate');
legend(alg_name);
set(gcf, 'position', [100 100 450 350]); set(gcf,'color','w');

% visualize the most important pixel contribution
figure;
subplot(2,2,1); pcolor(reshape(max(abs(x2), [], 2)~=0,[28, 28])');
title('Nonzero Pixel, \');
xlabel('row pixels'); ylabel('column pixels');
subplot(2,2,2); pcolor(reshape(max(abs(x3)~=0, [], 2),[28, 28])');
title('Nonzero Pixel, lasso \alpha=1');
xlabel('row pixels'); ylabel('column pixels');
subplot(2,2,3); pcolor(reshape(max(abs(x4)~=0, [], 2),[28, 28])');
title('Nonzero Pixel, lasso \alpha=0.5');
xlabel('row pixels'); ylabel('column pixels');
subplot(2,2,4); pcolor(reshape(max(abs(x5)~=0, [], 2),[28, 28])');
title('Nonzero Pixel, lasso \alpha=0.1');
xlabel('row pixels'); ylabel('column pixels');
set(gcf, 'position', [100 100 650 550]); set(gcf,'color','w');

%% Test for each individual digit

% Don't have to redo the training, the b now becomes ith column of b
% 1 meaning it is the digit of interest, 0 is not
% This becomes a binary classification problem
% Just need to analyze each column individually

npix_list = 25:25:780;
alg_list = 1:6;
alg_name = {'pinv', '\', 'lasso, \alpha=1',...
     'lasso, \alpha=0.5', 'lasso, \alpha=0.1', 'ridge'};

% Loop through all 10 digits and compute the error rate
for dig=1:10

    % Get error rate vs number of top pixels used
    ErrRate = zeros(length(alg_list),length(npix_list));
    for ii = 1:length(alg_list)
        for jj = 1:length(npix_list)
            npix = npix_list(jj); algn = alg_list(ii);
            eval(['x = x' num2str(algn) '(:,dig);']);
%             eval(['x = x' num2str(algn) '{dig};']);
            xE = abs(x); [~, ixE] = sort( xE, 'descend' );
            x_red=zeros(size(x)); x_red(ixE(1:npix))=x(ixE(1:npix));

            % Look at test error only, no one cares about training error
```

```
            % y greater than 0.5 will be set to yes
            labPred = (Atest*x_red > 0.5);
            labTrue = ytest(:,dig);
            ErrRate(ii,jj) = sum(labPred~=labTrue)/ntest;
        end
    end

    % Error rate plot for each digit
    figure;
    for ii=alg_list
        if ii==4 || ii==5 || ii==1
            mstyle = '+';
        else
            mstyle = '--';
        end
        plot(npix_list, ErrRate(ii,:), [c_list{ii} mstyle]);
        if ii==1; hold on; end
    end
    hold off; xlabel('Number of Pixels used'); ylabel('Error rate');
    ylim([0 0.12]); legend(alg_name);
    set(gcf, 'position', [100 100 450 350]); set(gcf,'color','w');
end

% Sparsity plot for digit 6
figure;
subplot(2,2,1); pcolor(reshape(max(abs(x2(:,6)), [], 2)~=0,[28, 28])');
title('Nonzero Pixel, \');
xlabel('row pixels'); ylabel('column pixels');
subplot(2,2,2); pcolor(reshape(max(abs(x3(:,6))~=0, [], 2),[28, 28])');
title('Nonzero Pixel, lasso \alpha=1');
xlabel('row pixels'); ylabel('column pixels');
subplot(2,2,3); pcolor(reshape(max(abs(x4(:,6))~=0, [], 2),[28, 28])');
title('Nonzero Pixel, lasso \alpha=0.5');
xlabel('row pixels'); ylabel('column pixels');
subplot(2,2,4); pcolor(reshape(max(abs(x5(:,6))~=0, [], 2),[28, 28])');
title('Nonzero Pixel, lasso \alpha=0.1');
xlabel('row pixels'); ylabel('column pixels');
set(gcf, 'position', [100 100 650 550]); set(gcf,'color','w');
```