```
% Read in Data,credit to Daniel.
% http://daniel-e.github.io/2017-10-20-loading-mnist-handwritten-digits-with-
octave-or-matlab/
data = load('mnist.mat');
trainIM = double(transpose(data.trainX));
trainLB = double(data.trainY);
testIM = double(transpose(data.testX));
testLB = double(data.testY);
trainLBM = zeros(10,60000);
for i = 1:length(trainLB)
    a = trainLB(i);
    if a == 0
        a = 10;
    end
    vector = zeros(10,1);
    vector(a,1) = 1;
    trainLBM(:,i) = vector;
end

testLBM = zeros (10,10000);
for i = 1:length(testLB)
    a = testLB(i);
    if a == 0
        a = 10;
    end
    vector = zeros(10,1);
    vector(a,1) = 1;
    testLBM(:,i) = vector;
end

%%
% Part a, b, c
% Pseudoinverse
X1 = trainLBM*pinv(trainIM);
test1 = X1*testIM;
test1LB = zeros(1,10000);
for i = 1:10000
    max = 0;
    position = 0;
    for j = 1:10
        if(test1(j,i)>max)
            max = test1(j,i);
            position = j;
            if (position == 10)
                position = 0;
            end
        end
    end
    test1LB(1,i) = position;
end
Error1 = length(find(abs(test1LB-testLB)));
Accuracy1 = (10000-Error1)/10000;
% Pseudoinverse provides very good precision to the testset. I am able to
% achieve around 85% accuracy to classify test set (error rate 15%).
% Although this method yields great accuracy, it fails to promote sparsity.

%%
% Lasso 0.1
X2 = zeros(10,784);
```

```
for i = 1:10
    X2(i,:) = lasso(trainIM.',trainLBM(i,:).','Lambda',0.1).';
end
test2 =X2*testIM;
test2LB = zeros(1,10000);
for i = 1:10000
    max = 0;
    position = 0;
    for j = 1:10
        if(test2(j,i)>max)
            max = test2(j,i);
            position = j;
            if (position == 10)
                position = 0;
            end
        end
    end
    test2LB(1,i) = position;
end
Error2 = length(find(abs(test2LB-testLB)));
Accuracy2 = (10000-Error2)/10000;
% Lasso lambda being 0.1 is over-penalizing, as the accuracy has dropped
% significantly. We need to relax the L1 norm in order to increase
% precision.


%%
% Lasso 0.05
X3 = zeros(10,784);
for i = 1:10
    X3(i,:) = lasso(trainIM.',trainLBM(i,:).','Lambda',0.05).';
end

P3 = all(X3==0);

test3 = X3*testIM;
test3LB = zeros(1,10000);
for i = 1:10000
    max = 0;
    position = 0;
    for j = 1:10
        if(test3(j,i)>max)
            max = test3(j,i);
            position = j;
            if (position == 10)
                position = 0;
            end
        end
    end
    test3LB(1,i) = position;
end
Error3 = length(find(abs(test3LB-testLB)));
Accuracy3 = (10000-Error3)/10000;
% Lasso with lambda being 0.05 is an improvement. 65% Accuracy rate is
% acceptable, with significantly less parameter used to make predictions.
% This is the algorithm that promotes sparsity while keeping the accuracy
% in an acceptable range. There are 784-624 = 160 pixels that are most
% informative in labeling images to their digits.The corresponding pixels
% are the 0 entries of variable P3.
```

```
%%
% Backslash
X4 = (trainIM.'\trainLBM.').';

test4 = X4*testIM;
test4LB = zeros(1,10000);
for i = 1:10000
    max = 0;
    position = 0;
    for j = 1:10
        if(test4(j,i)>max)
            max = test4(j,i);
            position = j;
            if (position == 10)
                position = 0;
            end
        end
    end
    test4LB(1,i) = position;
end
Error4 = length(find(abs(test4LB-testLB)));
Accuracy4 = (10000-Error4)/10000;
% The backslash also yields accuracy as high as around 85%, but the problem
% is the same with pseudoinverse: failing to promote sparsity. So we pick
% Lasso with lambda = 0.05 to promote sparsity, while keeping accuracy high
% enough.

% In all, the X's we get from different solvers is a mapping that maps our
% images to their respective label spaces. A*X does not always give integral
solutions.
% We pick the largest element out of the 10 decimals from the same column,
% make the largest in value 1 and all other numbers 0, and claim that the
% image belongs to this set. We pick the largest in value because it has
% the greatest probability to belong to that set.This matrix X that we get
% from different solvers bridges images with label spaces.
%%
% Part d
% X3 is the best solution that promotes sparsity while
% keeping accuracy rate at 65%. Now, we want to find the most important
% pixels for each labeling digit. The non-zero entries of each row of
% the mapping matrix will be the most important Pixels for that digit.
% It will be very straight forward with a graph(which I am providing right now).
i1 = reshape(X3(1,:), 28, 28)';
% pcolor(i1)
% title('Important Pixels for Label 1')
i2 = reshape(X3(2,:), 28, 28)';
% pcolor(i2)
% title('Important Pixels for Label 2')
i3 = reshape(X3(3,:), 28, 28)';
% pcolor(i3)
% title('Important Pixels for Label 3')
i4 = reshape(X3(4,:), 28, 28)';
% pcolor(i4)
% title('Important Pixels for Label 4')
i5 = reshape(X3(5,:), 28, 28)';
% pcolor(i5)
% title('Important Pixels for Label 5')
i6 = reshape(X3(6,:), 28, 28)';
```

```
% pcolor(i6)
% title('Important Pixels for Label 6')
i7 = reshape(X3(7,:), 28, 28)';
% pcolor(i7)
% title('Important Pixels for Label 7')
i8 = reshape(X3(8,:), 28, 28)';
% pcolor(i8)
% title('Important Pixels for Label 8')
i9 = reshape(X3(9,:), 28, 28)';
% pcolor(i9)
% title('Important Pixels for Label 9')
i10 = reshape(X3(10,:), 28, 28)';
% pcolor(i10)
% title('Important Pixels for Label 10')


% These pictures could give us a decent idea of which pixels are most
% important for each digit. The more stand-out the color is, the more
% important it is.
Ahat = sort(abs(X3),2,'descend');

%Ahat Matrix shows that there are 26 pixels important for label 1, 21
%pixels important for label 2, 21 important for label 3, 16 important for
%label 4, 9 important for label 5, 21 important for label 6, 26 important
%for label 7, 10 important for label 8, 16 important for label 9, and 24
%important for label 10.

% Surprisingly, when we test the set that concerns only one digit, the
% accuracy goes up even higher than the backslask command. The reason is that if
% we only concern the accuracy of a single digit, then we can only take
% pixels that are important for that digit only. This will make it do
% better because there are less interference from other pixels that might
% hinder recognition.
```