1a)

1)a) The covariance of a Matrix A is given as follows:

$$C = \frac{1}{N-1}\sum_{i=1}^{n}(A_i - \bar{A})(A_i - \bar{A})'$$

This can be simplified to:

$$C = \frac{1}{N-1}(A')^T A$$

where A' is result of each data value with its mean subtracted ~~out & transposed~~.

The original equation in the problem:

$$C = \frac{1}{N-1} A^T \left[I_N - \frac{1}{N} 1_N \right] A$$

mean-centering matrix, which is idempotent.

$$C = \frac{1}{N-1} \left[A^T I_N - A^T \left(\frac{-1}{N} 1_N \right) \right] A$$

$$C = \frac{1}{N-1} \left[A^T - \bar{A}^T \right] A$$

simply the $\bar{A}^T$ because we are adding $\frac{1}{N^{th}}$ of each value N times.

$$C = \frac{1}{N-1} (A')^T A$$

or

$$C = \frac{1}{N-1} (A')^T A$$

1b)

1b) $C = \frac{1}{N-1} A^T \left[I_N - \frac{1}{N} 1_N \right] A$

$(N-1)(C) = \frac{1}{N-1} A^T \left[I_N - \frac{1}{N} 1_N \right] A$

$(N-1)(X \Lambda X^T) = A^T \left[I_N - \frac{1}{N} 1_N \right] A$ → substitute $X \Lambda X^T$ for C

$(N-1)(X \Lambda X^T) = \left[A^T I_N - A^T \left(\frac{1}{N}\right) 1_N \right] A$

$(N-1)(X \Lambda X^T) = A^T A - \frac{1}{N} A^T 1_N A$

Substitute $A = U \Sigma V^T$

$(N-1)(X \Lambda X^T) = (V \Sigma^T U^T)(U \Sigma V^T) - \frac{1}{N} A^T 1_N A$

$\boxed{(N-1)(X \Lambda X^T) = V \Sigma^2 V^T - \frac{1}{N} A^T 1_N A}$

$U^T U$ is orthogonal so it becomes I. It is cancelled.

Σ is a diagonal matrix, so $\Sigma^T \Sigma = \Sigma^2$.

1c)

1C) $A = U\Sigma V^T$ $\quad C = X\Lambda V^T$

Let $A' = (A - \frac{1}{N}1_N)A$ as shown in Part a:

this is the colun centered matrix with $\bar{A}$ taken out.

$$C = \frac{1}{N-1}(A')^T A$$

Subsitute:

$$X\Lambda X^T = (V\Sigma U^T)(U\Sigma V^T)$$

Again, $U^T U = I$ because U is orthogonal

$\Sigma^T\Sigma = \Sigma^2$ because Σ is diagonal

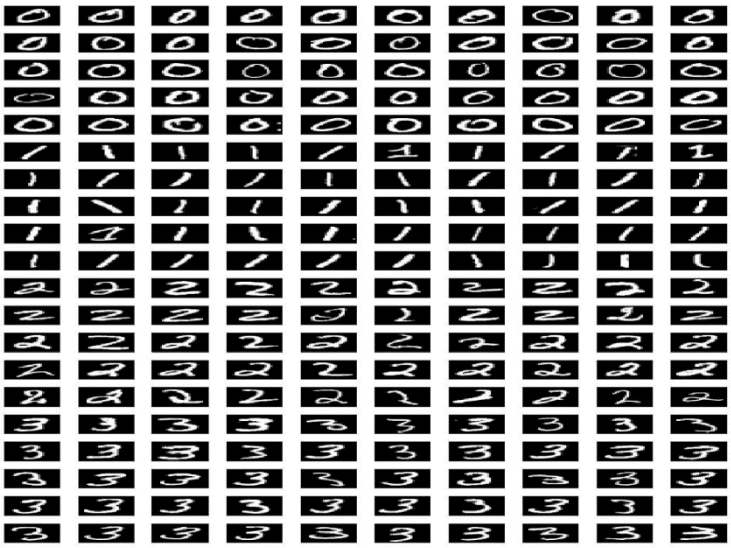
$$X\Lambda X^T = V\Sigma^2 V^T\left(\frac{1}{n-1}\right)$$

This is in a similar format

$$X = V$$
$$\Lambda = \frac{1}{n-1}\Sigma^2$$

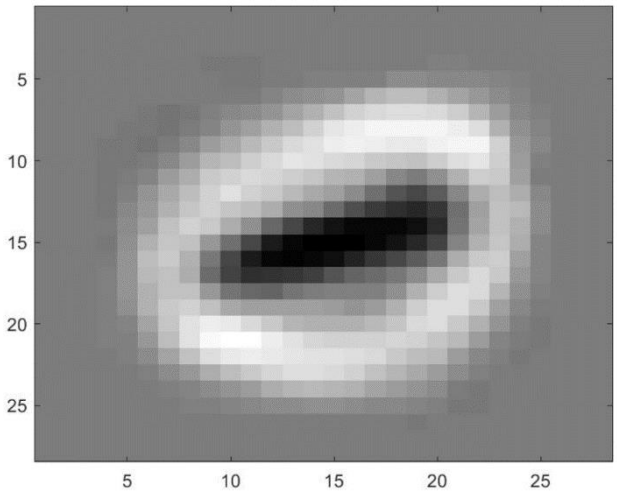
2)

Part b:

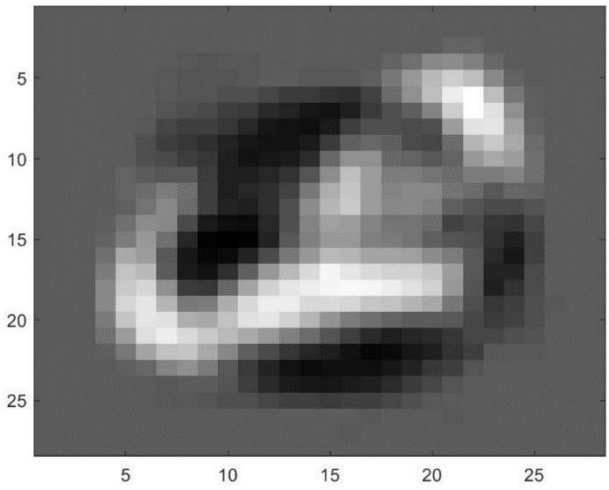


Part c:

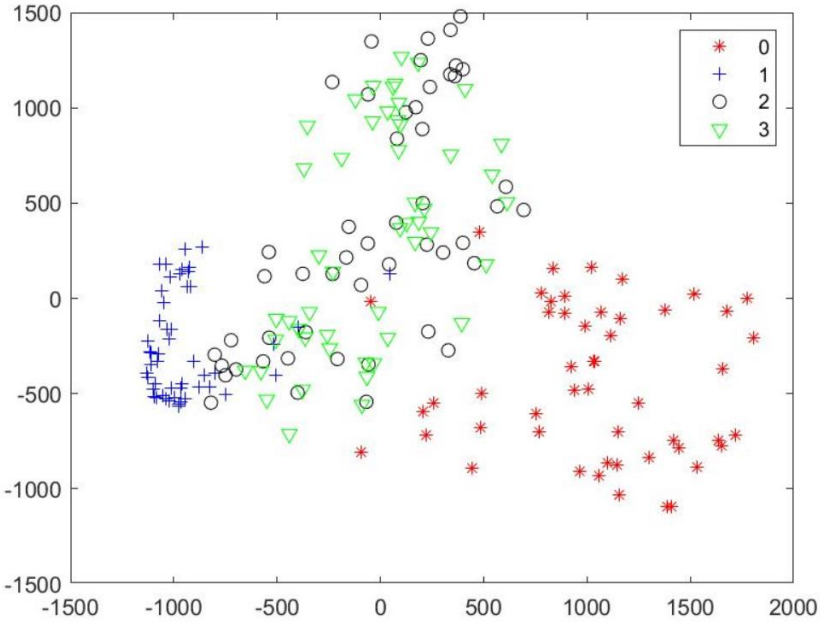
1st PC



2nd PC

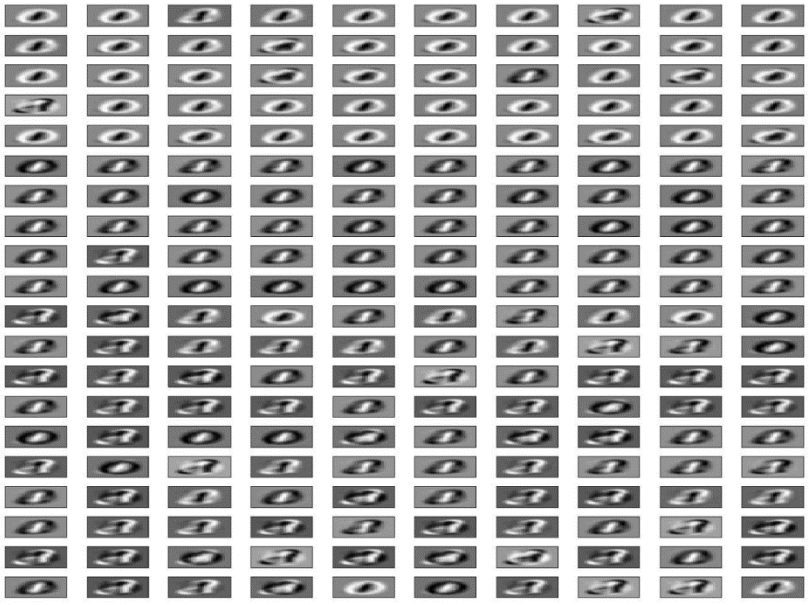


Part d:



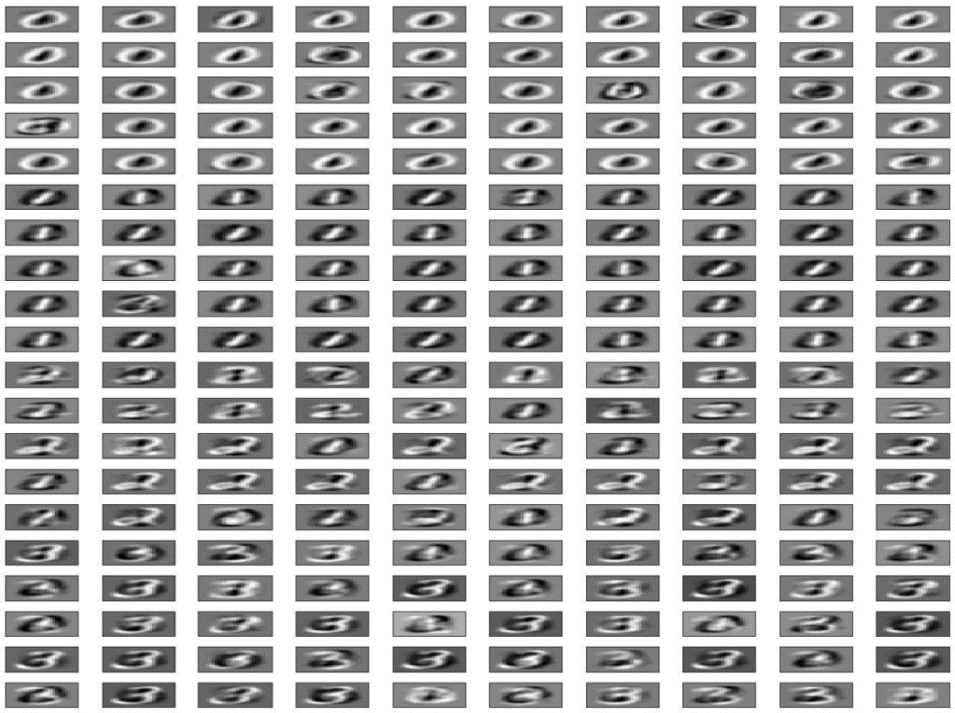
Digit 0 and digit 1 are easily discerned since they are farther apart. Digit 2 and digit 3 are harder to discern.

Part e:



Digit 0 and digit 1 are still easier to discern as before. Digit 2 and 3 are hard to discern again. Same result.

Part f:



Increasing rank to 50 makes it easy to discern between all digits. They are all easy to read now.

3.

a) iii) Since the rank is 56 but there are 58 columns, there must be 2 non-linearly independent columns. The 2 non-linearly independent columns are 37 and 42. This can be done easily one we reduce the matrix to reduced row echelon form. We can see that this matrix has column 37 and 42 which are non-pivot columns with a 0 value. Thus, these are linearly dependent.

b) i) The top-10 attributes with the largest absolute parameter values after standardizing the data using MLR are the features 45,46,4,57,16,35,23,15,14, and 27 according to MATLAB. Their parameter values are (in order): 4.45326e+06, 4.1673e+06, -2.6382e+06, 7.4089, 0.1370, 0.1298, -0.1128, 0.1083, -0.0867, 0.0853. The top-10 attributes with the largest absolute parameter values after standardizing the data using Lasso are the features 35, 15,16, 27, 36, 26, 56, 39, 14, 13. Their parameter values are (in order): 0.084596034842525, 0.074122182676981, 0.044611062033937, 0.043221241478524, 0.039649656040052, 0.030475172146205, 0.021568998074490, 0.016445400118705, 0.015535626775817, 0.015505859172787

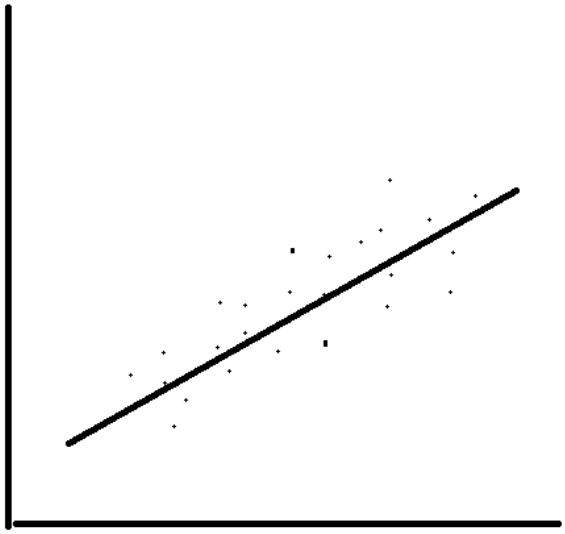
ii). The correlation for MLR is -0.0073. The correlation for Lasso is 0.0831. Since the correlation for Lasso is higher, it performs the best on the test-set.

3c) The kernel ridge file provided to us is not working with my data. I keep getting the error "Unable to perform assignment because the left and right sides have a different number of elements". It points to line 41 in the applyKernel.m file since the size of pred and (alpha'*K)' are different. I have tried changing size of my training and testing data and no matter what I seem to get this error. So, unfortunately I was not able to do Kernel Ridge regression. Also the applyKernel.m file was given to us 1 day before submission so I had no time to implement my own.

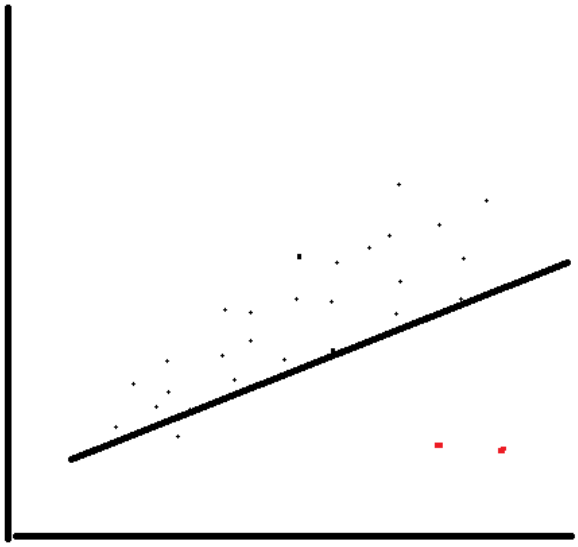
3d) After applying the natural logarithmic transformation for columns, the correlation for MLR is fairly unchanged at -0.0072. The correlation for Lasso has increased dramatically, to 0.1824. Again, I was unable to test Kernel ridge regression. Of the two (Lasso and MLR), Lasso performed better again so it is the best. It also improved from the data before the logarithmic transformation.

4)

a) Below is an image with no outliers present:



Below is an image with outliers present



A simple linear regression can be represented by y = Bx + a for a 1-dimensional problem. As seen above, the presence of outliers can shift the true regression fit towards the outliers. In the case above, the regression shifted downwards, towards the outliers, lowering the slope. This can mispresent the relationship of the data.

b)

4b) $f(w) = \sum \alpha (y_i - w^T x_i)^2 + \lambda \|w\|^2$

$= \alpha (y_i - w^T x_i)^T (y - w^T x_i) + \lambda w^T w$

$= \alpha (y_i^T - x_i^T w)(y_i - w^T x_i) + \lambda w^T w$

$= \alpha (y_i^T y_i - x_i^T w y_i - y_i^T w^T x_i + x_i^T w w^T x_i) + \lambda w^T w$

$= \alpha y_i^T y_i - \alpha x_i^T w y_i - \alpha y_i^T w^T x_i - \alpha x_i^T w w^T x_i + \lambda w^T w$

$= \alpha y_i^T y_i - 2\alpha y_i^T w^T x + \alpha x_i^T w w^T x_i + \lambda w^T w$

$\frac{d}{dw} = -2\alpha x_i^T y_i + 2\alpha x_i^T w x + 2\lambda w = 0$

$w(\alpha x_i^T x_i + \lambda I) = \alpha x_i^T y$

$w = (\alpha x_i^T x_i + \lambda I)^{-1} \alpha x_i^T y_i$

5)

Entropy for attribute A = A1 + A2

A1 = $\frac{30}{60}\left(-\frac{20}{30}log_2\frac{20}{30}-\frac{10}{30}log_2\frac{10}{30}\right)=\frac{30}{60}(0.91829)=0.45914$

A2 = $\frac{30}{60}\left(-\frac{10}{30}log_2\frac{10}{30}-\frac{20}{30}log_2\frac{20}{30}\right)=\frac{30}{60}(0.91829)=0.45914$

Entropy of A = A1 + A2 = $0.45914+0.45914=\mathbf{0.91829}$

Entropy for attribute B = B1 + B2 + B3

B1 = $\frac{20}{60}\left(-\frac{10}{20}log_2\frac{10}{20}-\frac{10}{20}log_2\frac{10}{20}\right)=\frac{20}{60}(1)=\frac{20}{60}=0.33333$

B2 = $\frac{5}{60}\left(-\frac{5}{5}log_2\frac{5}{5}-0\right)=\frac{5}{60}(0)=0$

B3 = $\frac{35}{60}\left(-\frac{15}{35}log_2\frac{15}{35}-\frac{20}{35}log_2\frac{20}{35}\right)=\frac{35}{60}(0.98522)=0.57471$

Entropy of B = B1 + B2 + B3 = $0.33333+0+0.57471=\mathbf{0.90803}$

The average weighted entropy for attribute B is smaller than A, **therefore attribute B should be chosen.**

6)

a)

The problem states that the decision classifier assigns majority class of training samples as the class label for each leaf node. It does not state what is assigned during ties. I assume in case of ties, the classifier assigns (+) label. My confusion matrices are calculated with this assumption.

Left tree:

		Predicted	
		+	-
Actual	+	20	10
	-	10	20

Right tree:

		Predicted	
		+	-
Actual	+	20	10
	-	20	10

b)

Training error for left tree: It correctly classified 40 examples (top left + bottom right of confusion matrix), and misclassified 20 examples (top right + bottom left of confusion matrix). So, error is: 20/60 = 0.3333 = **33.333%**

Training error for right tree: It correctly classified 30 samples, and misclassified 30 samples. So, error is 30/60 = 0.5000 = **50%**

c) MDL is given by the following formula:

Cost = $x\lceil log_2 m\rceil + y\lceil log_2 c\rceil + z\lceil log_2 N\rceil$

Where m = # of unique attributes, c = # of classes, and N = # of training instances, x = # of total attribute (nodes), y = # of leaf nodes, and z = # of misclassified example.

We are given that there are 4 binary attributes, with two classes (+) and (-), for both trees.

Left tree:

Cost = $3 * \lceil log_2 4\rceil + 4 * \lceil log_2 2\rceil + 20 * \lceil log_2 60\rceil = (3 * 2) + (4 * 1) + (20 * 6) = \mathbf{130}$ **bits**

Right tree:

Cost = $2 * \lceil log_2 4\rceil + 3 * \lceil log_2 2\rceil + 30 * \lceil log_2 60\rceil = (2 * 2) + (3 * 1) + (30 * 6) = \mathbf{187}$ **bits**

Since the cost of the left tree is cheaper/lower, **left tree should be preferred according to MDL.**

7)a)

$x_1x_2 - x_3x_4 \geq 0$

$\phi_1(x) = x_1x_2$

$\phi_2(x) = x_3x_4$

$f(x) = w_1\phi(x) + w_2\phi_2(x)$

$w_1 = 1$ and $w_2 = -1$

$f(x) = x_1x_2 - x_3x_4$

b)

This data set cannot be perfectly classified by a linear classifier

c)

Start my taking the natural logarithm of both sides to reduce to a polynomial expansion:

$e^{(x_1-x_2)} > 100$

$x_1 - x_2 > \ln(100)$

$x_1 - x_2 - \ln(100) > 0$

To convert from $>$ to $\geq$ we need to add 1 to the left hand side of the equation:

$x_1 - x_2 - \ln(100) + 1 \geq 0$

$\phi_1(x) = x_1$

$\phi_2(x) = x_2$

$\phi_3(x) = {x_1}^0 = 1$

$f(x) = w_1\phi(x) + w_2\phi_2(x) + w_3\phi_3(x)$

$w_1 = 1, w_2 = -1, w_3 = 1 - \ln(100)$

$f(x) = x_1 - x_2 - \ln(100) + 1$

d)

$-(x_1x_2) \geq 0$

$\phi_1(x) = x_1x_2$

$f(x) = w_1\phi(x)$

$w_1 = -1$

$f(x) = -(x_1x_2)$

8)

Linear SVM:

Classification error according to MATLAB = 0.1206

Confusion matrix:

		Predicted	
		+	-
Actual	+	2556	323
	-	232	1490

Non-linear SVM:

Classification error according to MATLAB = 0.0646

Confusion matrix:

		Predicted	
		+	-
Actual	+	2661	170
	-	127	1643

Classification error is much smaller in non-linear SVM. Therefore non-linear SVM is more effective.

Training this using SVM took almost 2.5 hours. Just a heads up if you wanted to test my code, it is going to take a while!

My code:

2)

```
load('mnist_all.mat');
X = train0(1:50, :);
X(51:100,:) = train1(1:50,:);
X(101:150,:) = train2(1:50,:);
X(151:200,:) = train3(1:50,:);

N = 50;
numCols = 10;
numRows = ceil(4*N/numCols);
d = sqrt(size(X,2));

figure;
set(gcf, 'color', 'white');
set(gcf, 'Position', [520 85 1020 720]);
for i=1:size(X,1)
    subplot(numRows,numCols,i);
    img = reshape(X(i,:),d,d)';
    imagesc(img);
    set(gca, 'xtick', []);
    set(gca, 'ytick', []);
end
colormap(gray);

saveas(gcf, 'digit_image.jpg', 'jpeg');

X = double(X);
[U,Z,S] = pca(X);
pc1 = U(:,1);
pc2 = U(:,2);
figure;
set(gcf,'color', 'white');
img1 = reshape(pc1,d,d);
imagesc(img1);
colormap(gray);
saveas(gcf, 'pc1_plot.jpg', 'jpeg');

figure;
set(gcf,'color', 'white');
img2 = reshape(pc2,[d d]);
imagesc(img2);
colormap(gray);
saveas(gcf, 'pc2_plot.jpg', 'jpeg');

figure;
set(gcf, 'color', 'white');
plot(Z(1:50,1),Z(1:50,2),'r*', 'DisplayName', '0');
legend
hold on
plot(Z(51:100,1), Z(51:100,2), 'b+', 'DisplayName', '1');
plot(Z(101:150,1), Z(101:150,2), 'ko', 'DisplayName', '2');
plot(Z(151:200,1), Z(151:200,2), 'gv', 'DisplayName', '3');
hold off
saveas(gcf, 'cse881hw1_part_2d.jpg', 'jpeg');
%class 0 and 1 are easily discernable. class 2 and 3 are hard to distinguish

rank = 2;
W = Z(:,1:rank)*diag(S(1:rank))*U(:,1:rank)';
```

```
figure;
set(gcf, 'color', 'white');
set(gcf, 'Position', [520 85 1020 720]);
for i=1:size(W,1)
    subplot(numRows,numCols,i);
    img = reshape(W(i,:),d,d)';
    imagesc(img);
    set(gca, 'xtick', []);
    set(gca, 'ytick', []);
end
colormap(gray);

saveas(gcf, 'cse881hw_part_2e', 'jpeg');
%same result

rank = 50;
W = Z(:,1:rank)*diag(S(1:rank))*U(:,1:rank)';

figure;
set(gcf, 'color', 'white');
set(gcf, 'Position', [520 85 1020 720]);
for i=1:size(W,1)
    subplot(numRows,numCols,i);
    img = reshape(W(i,:),d,d)';
    imagesc(img);
    set(gca, 'xtick', []);
    set(gca, 'ytick', []);
end
colormap(gray);

saveas(gcf, 'cse881hw_part_2f', 'jpeg');
%all the digits are easily dicernable with a high rank of 50
```

3)

```
%part a)
D = importdata('OnlineNewsPopularity.csv');
predictor_variables = D.data(:, 1:58); %predictor
response_variable = log(D.data(:, end)); %log of response variable, last column
r = rank(predictor_variables); %56
piv_tol = max(size(predictor_variables)*eps*norm(predictor_variables)); %adujusting
pivot tolerance
[R, p] = rref(predictor_variables, piv_tol); %p shows that columns 37 and 42 are
dependent
discarded = predictor_variables(:, p); %discards the depended columns

%part b)
training_dataX = discarded(1:2000,:);
testing_dataX = discarded(2001:end, :);

%standardize each perdictor variable (trainingX)
Standdev = std(training_dataX);
Aver = mean(training_dataX);
[R,C] = size(training_dataX);
for c = 1:C
    for r = 1:R
        training_dataX(r,c) = (training_dataX(r, c)-Aver(c))/Standdev(c);
    end
end

training_dataY = response_variable(1:2000,:);
testing_dataY = response_variable(2001:end, :);
training_dataX = [training_dataX ones(size(training_dataX,1), 1)]; %add 1s to
predictor variables
testing_dataX = [testing_dataX ones(size(testing_dataX,1), 1)];

%MLR
disp('MLR');
mlr_reg = regress(training_dataY, training_dataX);
[B, I] = maxk(abs(mlr_reg), 10);    %top 10 attributes
B_mlr = mlr_reg(I);
disp(B_mlr);
disp(I);
mlr_pred = testing_dataX*mlr_reg;
mlr_corr = corr(testing_dataY, mlr_pred);
disp(mlr_corr); %correlation

%Lasso
disp('Lasso');
[w, stats] = lasso(training_dataX, training_dataY, 'Alpha', 1, 'CV', 10);
lassoPlot(w, stats, 'PlotType', 'CV');
wbest = w(:, stats.Index1SE);
lasso_coef = stats.Intercept(stats.Index1SE);
[Bl,Il] = maxk(abs(wbest), 10); %top 10 atributes
B_lasso = wbest(Il);
disp(B_lasso);
disp(Il);
lasso_pred = testing_dataX*wbest + lasso_coef;
lasso_corr = corr(testing_dataY, lasso_pred);
disp(lasso_corr);  %correlation
```

```
%part d) - natural log transform columns 18-20. Remove this part of code
%for other parts of the question
training_dataX(:, 18:29) = log(training_dataX(:, 18:29));
testing_dataX(:, 18:29) = log(testing_dataX(:, 18:29));
training_dataX(isnan(training_dataX)|isinf(training_dataX)|imag(training_dataX)~=0) = 0;
testing_dataX(isnan(testing_dataX)|isinf(testing_dataX)|imag(testing_dataX)~=0) = 0;
```

8)

```
A = load('spambase.data');
N = size(A,1);
seed = 10;
rng(seed);
A = A(randperm(N),:);

X = A(:,1:end-1); %split predictors into X
y = A(:,end);   %split class label into y

NumfoldsOuter = 10;      %outerloop
NumfoldsInner = 5;  %innerloop
Lambda = logspace(-4,3,11);     % hyperparameters for linear SVM
bc = logspace(-3,3,11);         % hyperparameters for nonlinear SVM
ks = logspace(-3,3,11);         % hyperparameters for nonlinear SVM

indexes = crossvalind('KFold',N,NumfoldsOuter);  % indices for cross validation

Linear_Predictions = zeros(N,1);
Nonlinear_Predictions = zeros(N,1);

for fold=1:NumfoldsOuter
    guide_msg = sprintf('Executing fold %d', fold);
    disp(guide_msg);
    testIdx = find(indexes == fold);
    trainIdx = find(indexes ~= fold);
    disp(' ---- Executing linear SVM ---');
    linearSVMm =
fitclinear(X(trainIdx,:),y(trainIdx),'Kfold',NumfoldsInner,'Learner','svm','Lambda',Lambda);
    ce = kfoldLoss(linearSVMm);
    [~,bestIdx] = min(ce);
    bestLambda = Lambda(bestIdx(1));
    linearSVMm = fitclinear(X(trainIdx,:),y(trainIdx),'Learner','svm','Lambda',bestLambda);
    Linear_Predictions(testIdx) = predict(linearSVMm, X(testIdx,:));
    cvloss = zeros(length(bc),length(ks));

    disp('  --- Executing nonlinear SVM ---');
    for i=1:11
        for j=1:11
            nonlinearSVMm =
fitcsvm(X(trainIdx,:),y(trainIdx),'Standardize',true,'KernelFunction','RBF','KernelScale',ks(j),'
BoxConstraint',bc(i),'Kfold',NumfoldsInner);
            cvloss(i,j) = kfoldLoss(nonlinearSVMm);
        end
    end

    [bcIdx,ksIdx] = find(cvloss == min(cvloss(:)));
    bestbc = bc(bcIdx(1));
    bestks = ks(ksIdx(1));
    nonlinearSVMm =
fitcsvm(X(trainIdx,:),y(trainIdx),'Standardize',true,'KernelFunction','RBF','KernelScale',bestks,
'BoxConstraint',bestbc);
    Nonlinear_Predictions(testIdx) = predict(nonlinearSVMm, X(testIdx,:));

end

cp1 = classperf(y);
classperf(cp1,Linear_Predictions);
cp1.DiagnosticTable
cp1.ErrorRate

cp2 = classperf(y);
classperf(cp2,Nonlinear_Predictions);
cp2.DiagnosticTable
cp2.ErrorRate

save q8answers.mat
```