CPSC 340 Assignment4
Student Name: Weining Hu
Student Number: 45606134

1 Regularized Logistic Regression

1.1 L2-Regularization

Code:

```
function [model] = logRegL2(X,y,lambda)
[n,d] = size(X);
maxFunEvals = 400; % Maximum number of evaluations of objective
verbose = 1; % Whether or not to display progress of algorithm
w0 = zeros(d,1);
model.w = findMin(@logisticLoss,w0,maxFunEvals,verbose,X,y,lambda);
model.predict = @(model,X)sign(X*model.w); % Predictions by taking sign
model.lambda = lambda;
end

function [f,g] = logisticLoss(w,X,y,lambda)
yXw = y.*(X*w);
f = sum(log(1 + exp(-yXw))) + (lambda/2)*(w'*w); % Function value
g= -X'*(y./(1+exp(yXw)))+lambda*w; % Gradient
end
```

Report number of nonzeros and validation error:

numberOfNonZero =101
trainingError =0.0020
validationError =0.0740

1.2 L1-Regularization

code:

```
function [model] = logRegL1(X,y,lambda)

[n,d] = size(X);

maxFunEvals = 400; % Maximum number of evaluations of objective
verbose = 1; % Whether or not to display progress of algorithm
w0 = zeros(d,1);

model.w = findMinL1(@logisticLoss,w0,lambda,maxFunEvals,verbose,X,y);
model.predict = @(model,X)sign(X*model.w); % Predictions by taking sign
model.lambda = lambda;
end

function [f,g] = logisticLoss(w,X,y)
yXw = y.*(X*w);
```

```
f = sum(log(1 + exp(-yXw))); % + (lambda/2)*norm(w,1); % Function value
g= -X'*(y./(1+exp(yXw))); % Gradient
end
```

Report number of nonzeros and validation error:

numberOfNonZero =71
trainingError =0
validationError =0.0520

1.3 L0-Regularization

code:

```
function [model] = logRegL0(X,y,lambda)

[n,d] = size(X);
maxFunEvals = 400; % Maximum number of evaluations of objective
verbose = 0; % Whether or not to display progress of algorithm
w0 = zeros(d,1);
oldScore = inf;

% Fit model with only 1 variable,
% and record 'score' which is the loss plus the regularizer
ind = 1;
w = findMin(@logisticLoss,w0(ind),maxFunEvals,verbose,X(:,ind),y);
score = logisticLoss(w,X(:,ind),y) + lambda*length(w);
minScore = score;
minInd = ind;

while minScore ~= oldScore
   oldScore = minScore;
   fprintf('\nCurrent set of selected variables (score = %f):',minScore);
   fprintf(' %d',ind);

   for i = 1:d
      if any(ind == i)
         % This variable has already been added
         continue;
      end

      % Fit the model with 'i' added to the features,
      % then compute the score and update the minScore/minInd
      ind_new = union(ind,i);
      % fit new model
      w_new = findMin(@logisticLoss,w0(ind_new),maxFunEvals,verbose,X(:,ind_new),y);
```

```
        score_new = logisticLoss(w_new,X(:,ind_new),y) + lambda*length(w_new);
        if (score_new < minScore)
            minScore = score_new;
            minInd = ind_new;
        end

    end
    ind = minInd;
end

model.w = zeros(d,1);
model.w(minInd) = findMin(@logisticLoss,w0(minInd),maxFunEvals,verbose,X(:,minInd),y);
model.predict = @(model,X)sign(X*model.w); % Predictions by taking sign
end

function [f,g] = logisticLoss(w,X,y)
yXw = y.*(X*w);
f = sum(log(1 + exp(-yXw))); % Function value
g = -X'*(y./(1+exp(yXw))); % Gradient
end
```

Report number of nonzeros and validation error:

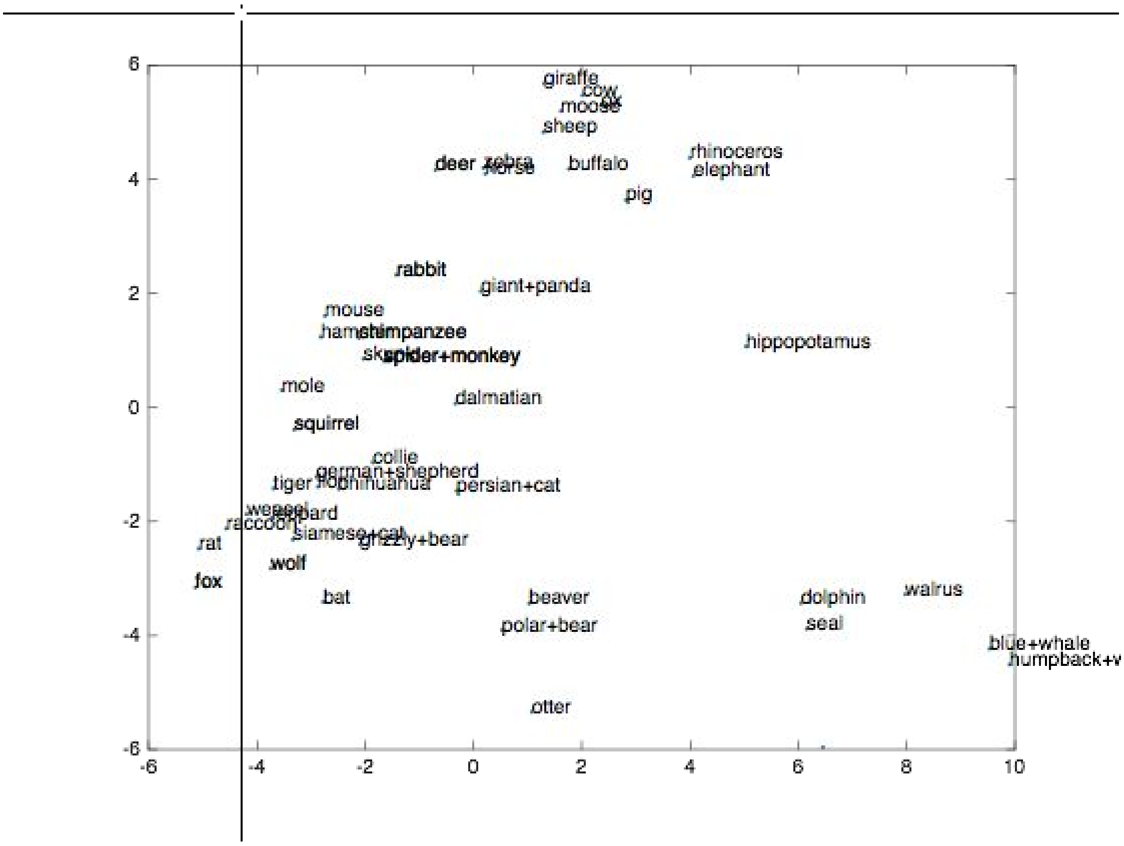
numberOfNonZero =24
trainingError =0
validationError =0.0180

2 Principal Component Analysis

2.1 Data Visualization

code:

```
load animals.mat
[n,d] = size(X);
X = standardizeCols(X);
[U,S,V]=svd(X);
W=V(:,1:2)';
Z=X*W';
plot(Z(:,1),Z(:,2),'.');
gname(animals);
```



2.2 Data Compression and Variance

code to calculate the ratio from k = 1 to 3:

```
load animals.mat
[n,d] = size(X);
X = standardizeCols(X);
[U,S,V]=svd(X);
for k = [1,2,3]
   W = V(:,1:k)';
   Z = X*W';
   ratio = norm((X-Z*W),'fro')^2/norm(X,'fro')^2
end
```

result:

k = 1, ratio =0.8279

k = 2, ratio =0.6981

k = 3, ratio =0.6122

code to find the k that lets ratio decrease to 20%:

```
for k = [1:d]
   [U,S,V] = svd(X);
   W = V(:,1:k)';
   Z = X*W';
   ratio = norm((X-Z*W),'fro')^2/norm(X,'fro')^2
   if (ratio <=0.2)
      k
      ratio
      break
   end
end
```

result: I stop when k =16,ratio =0.1967
From observations, and the results proved here: http://www.cs.yale.edu/homes/el327/datamining2012aFiles/06_singular_value_decomposition.pdf. We could see that the ratio could be calculated by the square root of the sum of the first K squared singular values divided by the sum of all the squared singular values.

3 Outlier Detection

3.1 Model-Based Outlier Detection

code:

```
load cities.mat
for c = 1:9
   my_z = zscore(ratings(:,c));
   index = find(abs(my_z) >= 4);
   [m,n] = size(index);
   if (m >0)
      fprintf('Category is %s\n', categories(c,:));
      city = names(index,:);
      disp(city);
   end
end
```

Category is housing
Norwalk, CT
Stamford, CT
Category is health
Boston, MA
Chicago, IL

New York, NY
Category is crime
Miami-Hialeah, FL
New York, NY
Category is arts
Chicago, IL
Los Angeles, Long Beach, CA
New York, NY
Category is economics
Midland, TX

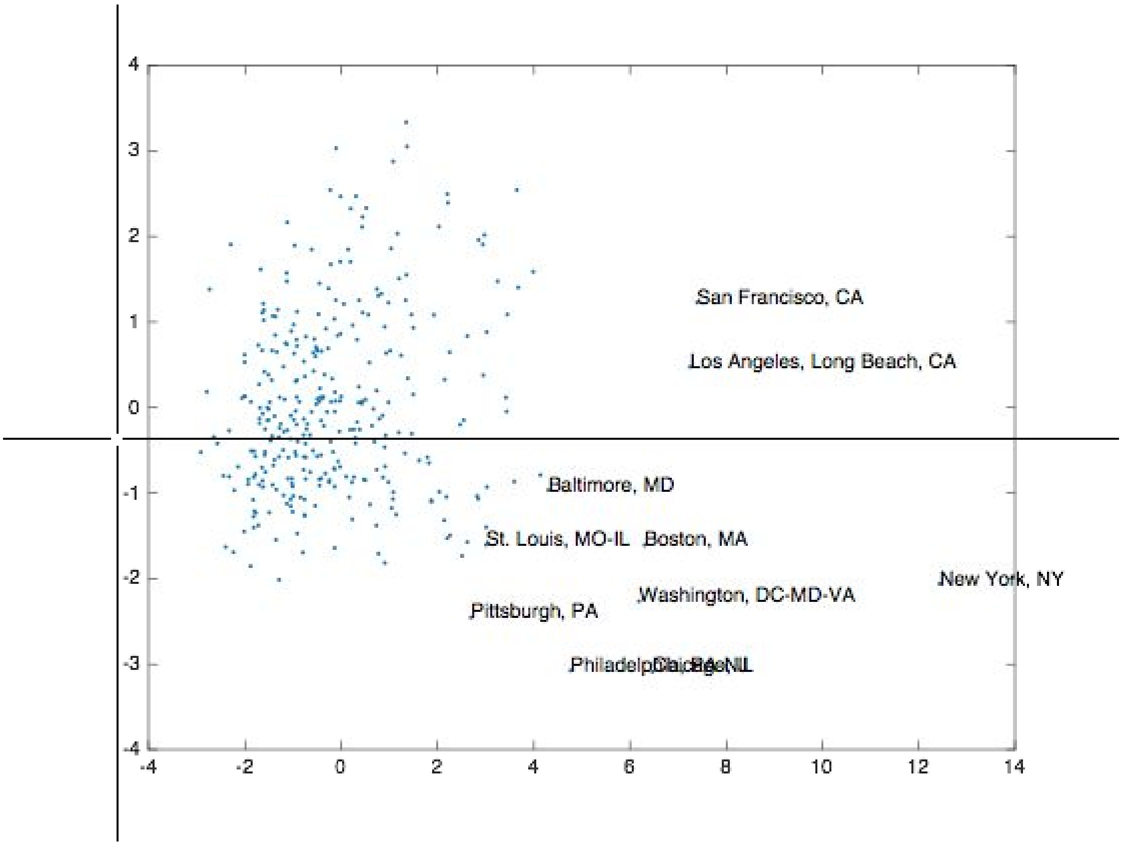
3.2 Graphical Outlier Detection

code:

```
load cities.mat

[n,d] = size(ratings);
X=ratings;
X = standardizeCols(X);
[U,S,V]=svd(X);
W=V(:,1:2)';
Z=X*W';

plot(Z(:,1),Z(:,2),'.');
gname(names);
```



3.3 Distance based outlier

code:

```
load cities.mat
[N,D] = size(X);

Dist = sqrt(X.^2*ones(D,N) + ones(N,D)*(X').^2 - 2*X*X');

K = 3;
avg = zeros(N,1);
for t = 1:N
    test = Dist(:,t);
    [sortDist,sortIndex] = sort(test,'ascend');
    % because the minimum would be the point itself, so we start from 2
    minIndex = sortIndex(2:K+1);
    avg(t) = sum(Dist(minIndex,t))/K;
end

% Then calculate the outlierness
outlierness = zeros(N,1);
for t = 1:N
    test = Dist(:,t);
    [sortDist,sortIndex] = sort(test,'ascend');
    minIndex = sortIndex(2:K+1);
```

```
    outlierness(t) = avg(t)/(sum(avg(minIndex))/K);
end

[sortOutlierness,sortOutindex] = sort(outlierness,'descend');
    maxoutindex = sortOutindex(1:10);
    names(maxoutindex,:)
    sortOutlierness(1:10)
```

result:
New York, NY
Newark, NJ
Burlington, VT
East St. Louis-Belleville, IL
San Francisco, CA
Stamford, CT
Houma-Thibodaux, LA
Philadelphia, PA-NJ
Rochester, MN
Iowa City, IA

```
ans =

    7.3044
    2.3150
    1.9535
    1.9305
    1.8919
    1.8878
    1.8583
    1.8102
    1.8025
    1.6645
```