CPSC 340 Assignment 4 (due November 13)

Regularized Logistic Regression, PCA, Outlier Detection

- You can work in groups on the assignments. However, please hand in your own assignments and state the group members that you worked (as well as other sources of help like online material).
- Place your name and student number on the first page, and submit all answers as a single PDF file to handin.
- For questions that ask for code, you should include the relevant parts of the code in the appropriate place in the PDF file.
- Please organize your submission sequentially according to the sections used in this document.
- All Sections (1-5) are equally weighted.
- There may be updates/clarifications to the assignment after the first version is put online. Any modifications will be marked in red.

1 Regularized Logistic Regression

If you run the function *example_logistic*, it will:

1. Load a binary classification dataset containing a training and a validation set.
2. 'Standardize' the columns of X and add a bias variable.
3. Apply the same transformation to $Xvalidate$.
4. Fit a logistic regression model.
5. Report the number of features selected by the model (number of non-zero regression weights).
6. Report the error on the validation set.

Logistic regression does ok on this dataset, but it uses all the features (even though only the prime-numbered features are relevant) and the validation error is above the minimum achievable for this model (which is 1 percent). In this question, you will modify this demo to use different forms of regularization to improve on these aspects.

1.1 L2-Regularization

Make a new function, *logRegL2*, that takes an input parameter λ and fits a logistic regression model with L2-regularization. Specifically, while *logReg* computes w using

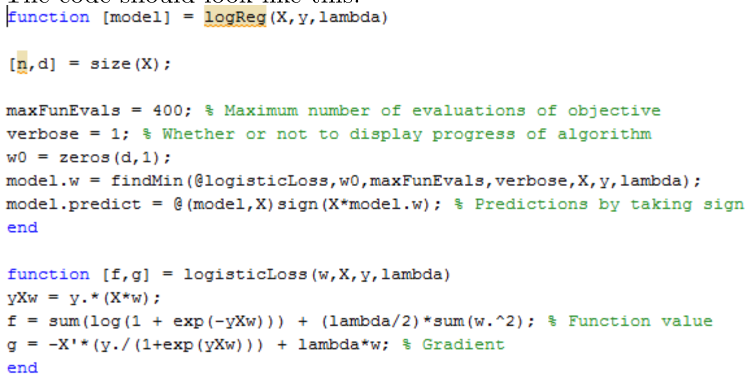
$$\operatorname*{argmin}_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)),$$

your new function *logRegL2* should compute w using

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \sum_{i=1}^{n} \left[\log(1+\exp(-y_i w^T x_i))\right] + \frac{\lambda}{2}\|w\|^2.$$

Hand in your updated code. Using this new code, report the number of non-zeroes and the validation error with $\lambda = 1$.

Answer:
The code should look like this:



```
function [model] = logReg(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
end

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

With L2-regularization, the validation error decreases to 0.074 but the number of non-zeroes stays at 101.

1.2 L1-Regularization

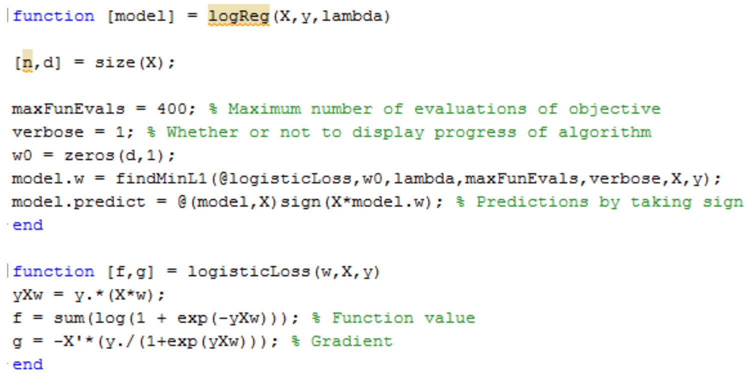
Make a new function, *logRegL1*, that takes an input parameter λ and fits a logistic regression model with L1-regularization,

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \sum_{i=1}^{n} \left[\log(1+\exp(-y_i w^T x_i))\right] + \lambda\|w\|_1.$$

Hand in your updated code. Using this new code, report the number of non-zeroes and the validation error with $\lambda = 1$.

Hint: you can use the function *findMinL1* to minimze the sum of a differentiable function and $\lambda\|w\|_1$. This function has a similar interface to *findMin*, except that you (a) only provide the code to compute the function/gradient of the differentiable part and (b) need to provide the value λ. (This function implements a generalization of the projected-gradient algorithm we discussed in class for solving non-negative problems. If you are curious, you can look at *findMinNN* which implements projected-gradient with non-negative constraints.)

Answer:
The code should look like this:



```
function [model] = logReg(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
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
```

With L1-regularization, the validation error decreases to 0.052 and the number of non-zeroes decreases to 71.

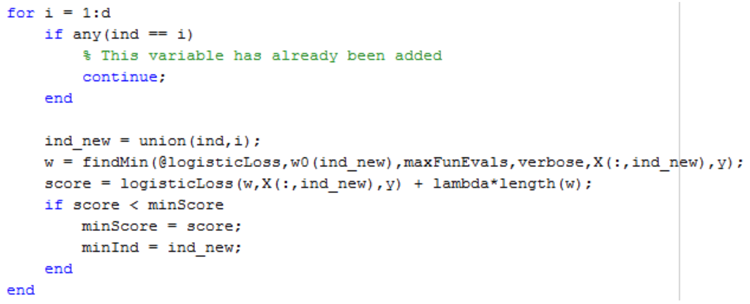
1.3 L0-Regularization

The function *logRegL0* contains part of the code needed to implement the *forward selection* algorithm, which approximates the solution with L0-regularization,

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda \|w\|_0.$$

The 'for' loop in this function is missing the part where we fit the model using the subset *ind_new*, then compute the score and updates the *minScore/minInd*. Modify the 'for' loop in this code so that it fits the model using only the features *ind_new*, computes the score above using these features, and updates the *minScore/minInd* variables. Hand in your updated code. Using this new code, report the number of non-zeroes and the validation error with $\lambda = 1$.

Answer:
The for loop should now contain something look like this:



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

    ind_new = union(ind,i);
    w = findMin(@logisticLoss,w0(ind_new),maxFunEvals,verbose,X(:,ind_new),y);
    score = logisticLoss(w,X(:,ind_new),y) + lambda*length(w);
    if score < minScore
        minScore = score;
        minInd = ind_new;
    end
end
```

With L0-regularization, the validation error decreases to 0.018 and the number of non-zeroes decreases to 24.

2 Principal Component Analysis

The function *example_PCA* will load the animals dataset from the previous assignment, standardize the features, and then give two unsatisfying visualizations of it. First it shows a plot of the matrix entries, which has too much information and thus gives little insight into the relationships between the animals. Next it shows a scatterplot based on the first two features, and you can click on the names of the points to reveal the corresponding animals. However, this reveals very little about the data. Further, because of the binary features, even a scatterplot matrix will show us almost nothing about the data.

2.1 Data Visualization

Modify this demo so that it applies PCA to this dataset, by taking the singular value decomposition (SVD) of the matrix X,

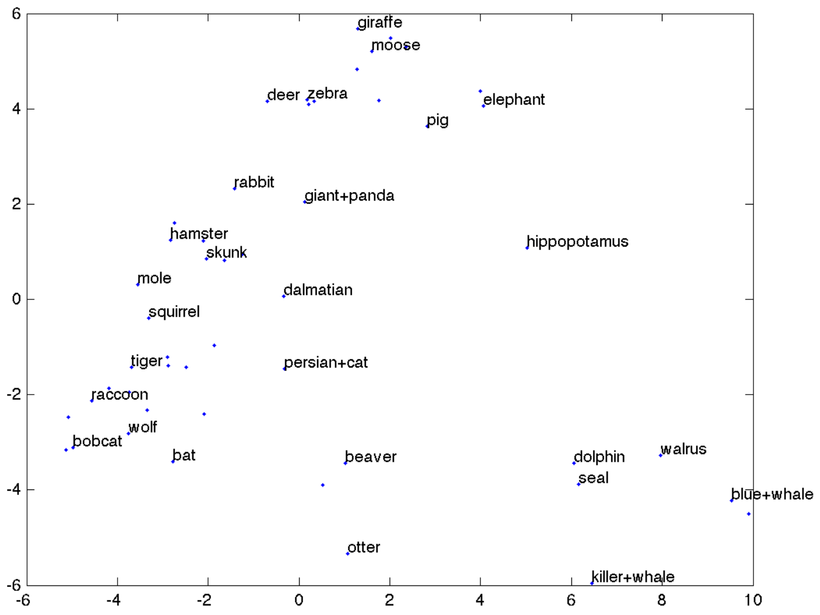
$$U\Sigma V^T = X,$$

as shown in class (use the function *svd*). This should give you a 2 by 85 matrix W, whose rows should have an L2-norm of 1 and where the inner product between the rows is zero. Use this matrix to construct a 50 by 2 matrix Z containing the low-dimensional representation of the animals dataset. Make a scatterplot of the two columns in Z, and use the *gname* function to label a bunch of the points in the scatterplot. Hand in your modified code and the scatterplot.

Hint: After standardizing the columns, your next two steps should be:
```
[U,S,V] = svd(X);
W = V(:,1:2)';
```

Answer:
The code should look like this:
```
[U,S,V] = svd(X);
W = V(:,1:2)';
Z = X*W';
plot(Z(:,1),Z(:,2),'.');
gname(animals);
```

The scatterplot should like this:



The points that are labeled could vary. (Roughly it looks like as we move from the upper-left to the lower-right we go from terrestrial to aquatic animals, and as we move from the lower-left to the upper-right the animals are getting bigger. But they are definitely exceptions to these rules that the higher-order principal components might capture, and the 'crowding' effect places some very disimilar animals next to each other.)

2.2 Data Compression and Variance

PCA uses an approximate matrix factorization,

$$X \approx ZW,$$

and the number k of principal components controls the accuracy of this approximation. The 'Frobenius' norm is a natural generalization of the Euclidean norm to matrices, and is defined by

$$\|X\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{d} x_{ij}^2}.$$

In Matlab, you can use *norm(X,'fro')* to compute this quantity. (Since X is assumed to have a mean of zero when we apply PCA, notice that the squared Frobenius norm is proportional to the variance of the entries of the matrix.) In class, we said that PCA finds the W and Z that minimize the squared Frobenius norm,

$$\operatorname*{argmin}_{W,Z} \|X - ZW\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{d}(x_{ij} - w_j^T z_i)^2.$$

If we view PCA as a data compression algorithm, one way to measure the compression error is with the ratio

$$\frac{\|X - ZW\|_F^2}{\|X\|_F^2}.$$

A value of zero means that ZW is a perfect approximation of X, while a value close to 1 indicates that ZW does nothing (and values bigger than one could mean that ZW is worse than doing nothing). Report the value of this ratio when $k = 1$, when $k = 2$, and when $k = 3$. By looking at the diagonal elements of Σ (the singular values), what seems to be the relationship between the compression ratio and the singular values? By looking at Σ, how many principal components would be needed to achieve a ratio of 20%?

Hint: You will find three transformations very useful to discover this relationship. First, square the singular values. Next, divide each squared singular value by the sum of the squared singular values. Third, you may want to look at the cumulative sum of these values.

Answer:
For $k = 1$ we get 0.8279. For $k = 2$ we get 0.6981. For $k = 3$ we get 0.6122. The ratio is equal to $1 - (\sum_{i=1}^{k}\sigma_i^2/(\sum_{i=1}^{d}\sigma_i^2)$, so the first k normalized singular values give us the approximation ratio. By using this, we know that we can get a ratio of 20% using 16 principal components.

3 Outlier Detection

This question uses *cities.mat*, a collection of 9 categories of ratings ($d = 9$) for 329 American cities ($n = 329$). We'll use this dataset to illustrate different approaches to outlier detection.

3.1 Model-Based Outlier Detection

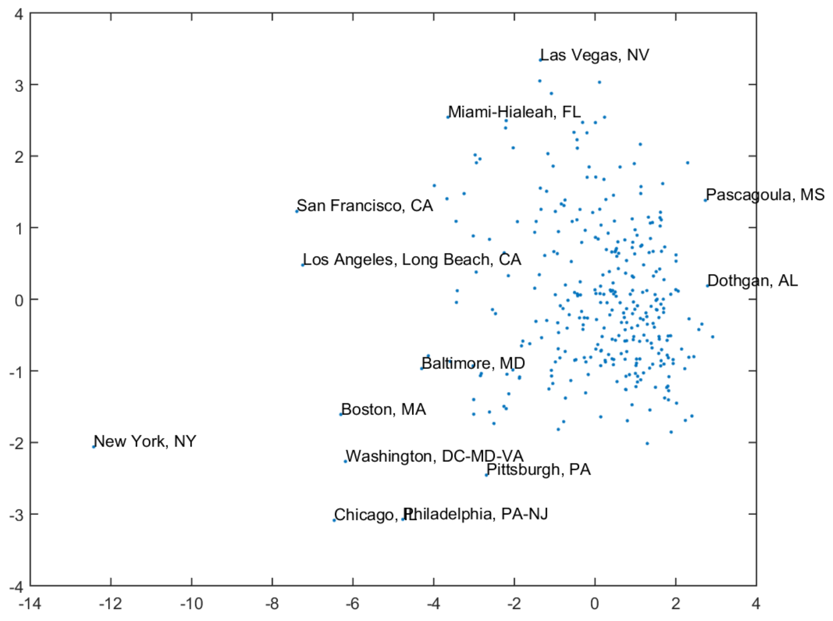
For each of the 9 categories, compute the z-score and report the cities that have $|z| \geq 4$ in any category.

Answer:
This is the list of cities that are outliers in at least one category (I've also included the categories, but they don't need to include the categories)):
Norwalk (housing)
Stamford (housing)
Boston (health)
Chicago (health, arts)
New York (health, crime, arts)
Miami-Hialeah (crime)
Los Angeles (arts)
Midland (economics)
Note that these are all positive outliers.

3.2 Graphical Outlier Detection

As in Question 2.1, make a 2D visualization of the data using PCA (remember that PCA takes standardizes features as input), and label the cities that appear to be outliers. Hand in the scatterplot.

Answer:
Here is the plot I got:



We'll accept any reasonable labelling of cities that appear at least a bit isolated from the other cities.

3.3 Distance-Based Outlier Detection

Write a script to compute the 'outlierness' of each city based on these ratings. Hand in your code to do this. For $k = 3$, report the cities with the top 10 values of 'outlierness' along with their 'outlierness' score.

Hint: you do not have to implement the 'influenced outlierness' score for this question. You may find it easier to first find the k-nearest neighbours of each city, then compute the average distance to the k-nearest neighbours of each city, and then finally compute the outlierness score.

Answer:
Here is the code that I used:

```
% Outlierness
k = 3;

D = X.^2*ones(d,n) + ones(n,d)*(X').^2 - 2*X*X';
D = sqrt(D); % Distances are based on the square root

% Find neighbours
for i = 1:n
    [minDist,sorted] = sort(D(:,i));
    neighbours(i,:) = setdiff(sorted(1:k+1),i);
end

% Compute average distances to neighbours
for i = 1:n
    A(i,1) = mean(D(i,neighbours(i,:)));
end

% Compute outlierness
for i = 1:n
    O(i,1) = A(i,1)/mean(A(neighbours(i,:)));
end

[values,sorted] = sort(O,'descend');
names(sorted(1:10),:)
```

If you do not standardize the features, you get these values:
New York, NY (7.30)
Newark, NJ (2.31)
Burlington, VT (1.95)
East St. Louis-Belleville, IL (1.93)
San Francisco, CA (1.89)
Stamford, CT (1.89)
Houma-Thibodaux, LA (1.86)
Philadelphia, PA-NJ (1.81)
Rochester, MN (1.80)
Iowa City, IA (1.66)
If you standardize the features, then you get the following values:
New York, NY (2.87)
Provo-Orem, UT (2.27)
Duluth, MN-WI (2.18)
Rochester, MN (1.96)
Pascagoula, MS (1.91)
Odessa, TX (1.87)
Reno, NV (1.77)
Bremerton, WA (1.71)
Miami-Hialeah, FL (1.69)
Richland-Kinnewick-Pasco, WA (1.68)
Both answers are acceptable. Notice that while New York is a clear outlier, the different methods found outliers in very different ways.