Classification of the Mnist Dataset via Linear Programming

Riley Kenyon

04/25/2020

Abstract

In this report linear programming will be used to determine the coefficients of hyper planes for binary classifiers. The coefficients will be used in a directed acyclic graph to determine the most likely classification of the number. The classifier will be trained and tested on the mnist dataset to perform handwritten character recognition. The solution minimizes the 2-norm of the hyper plane coefficients whilst considering the margin from the hyper planes, weighed by a factor γ.

1 Introduction

1.1 Problem Statement

The proposed classification problem will introduce a number of hyperplanes with coefficients that separate a feature set with some margin determined by a weighting function. The problem utilizes the mnist handwritten digit data base to build the classifier. The digit database will be separated into training data that is used to determine the hyperplanes, and testing data that is used to determine the accuracy of the model. The split is 60,000 images used for training and 10,000 images used for testing. Each feature is an image of size 28x28 pixels which is 'flattened' to a row vector of size 784. The data set represents integer numbers 0 through 9 and will require $\binom{10}{2}$ or 45 sets of hyperplanes to distinguish each binary classification in the set.

2 Linear Programming

In linear programming, a method used to solve optimization problems, the combination of constraint and criteria are written in the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \end{array} \tag{1}$$

where the first equation represents the cost function, and the second states the problem is "subject to" the constraints. This form in particular is representative of equality constraints, however the constraints can also described by an inequality. To pose classification as a linear programming problem the cost function will be derived from the margin of a hyperplane.

2.1 Deriving the Cost Function

For the two dimensional case, a hyperplane can be thought of as a line that is strictly separable for a set of data based on it's properties as seen in figure 1. For the example, the feature set can be described by the coefficients of the hyperplane, in this case the slope of the line and the y-intercept. The form of a line, trivially being

$$y = mx + b \tag{2}$$

Alternatively, if this was put into the form of a vector consisting of the elements y and x, the form would take on the form of

$$\begin{bmatrix} 1 & -m \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} - b = 0 \tag{3}$$

In higher order systems, this becomes more interesting due to the additional coefficients. The equation of the hyperplane can thereby be written as

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n - b = 0 \tag{4}$$

or alternatively in the matrix form of

$$a^T x - b = 0 \tag{5}$$

where a is a vector of coefficients and b is an offset. Looking at the 2D problem as classification, if the features (coordinates) are greater than the

hyperplane by some margin, ϵ, then the features are identified as blue and if they are less than the hyperplane by some margin, then the features are identified as red.

Looking at the 2D data set, there can be an infinite number of hyperplanes as long as the data is strictly separable. In order to determine a cost function, the equation of two points $\hat{x}_1$ and $\hat{x}_2$ will be looked at. Assuming the hyperplane parameters a and b, and a margin of $\epsilon = 1$, the equations of one point $\hat{x_1}$ can be shown to be

$$a^T \hat{x}_1 - b = 1 \tag{6}$$

where $\hat{x}_1$ is greater than the hyperplane. The second equation of $\hat{x}_2$ being

$$a^T \hat{x}_2 - b = -1 \tag{7}$$

where $\hat{x}_2$ is less than the hyperplane. The distance between these points can be represented by a vector

$$\hat{x}_2 = \hat{x}_1 - t\frac{a}{||a||} \tag{8}$$

where t is a scalar multiplier and vector is the unit version of a. Substituting $\hat{x}_2$ in equation 7 results in the expression

$$a^T \left(\hat{x}_1 - t\frac{a}{||a||} \right) - b = -1 \tag{9}$$

simplifying the equation by using $a^T \hat{x}_1 - b = 1$ and solving for t results in

$$t = \frac{2}{||a||_2} \tag{10}$$

in essence, the margin of the hyperplane can be represented by the 2-norm of the coefficients a. In implementation, setting the margin to a fixed value may not result in the most effective classifier. To compensate for the variability in each binary classifier, a weighting function based on slack variables is used to loosen or tighten the requirements of how close a hyperplane can be to the data. Essentially the weighting function acts as a tuning knob on the margin which ultimately determines the values of hyperplane coefficients, the weight

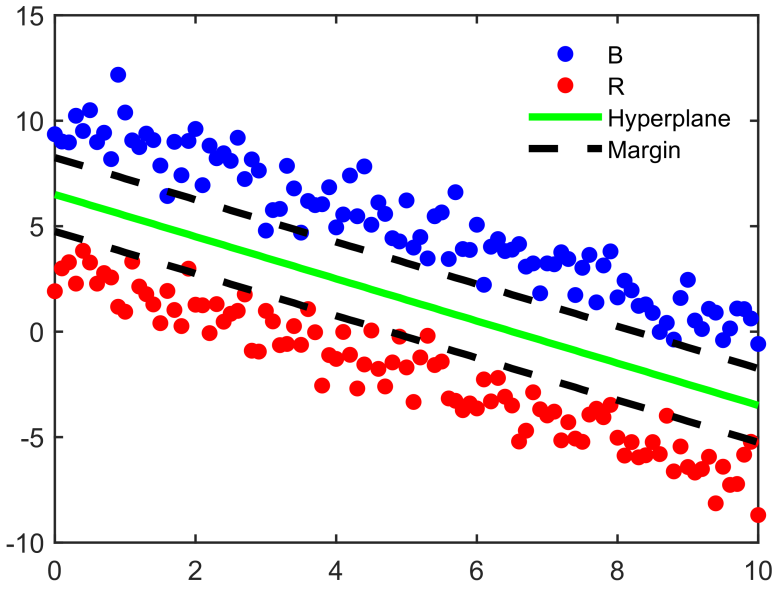

Figure 1: Example of a hyperplane in 2D that separates data based on characteristics. In this case, there is also a margin between the hyperplane (represented as a line) and the two feature sets.

is represented by the variable γ. Formally this can be shown by the cost function

$$\begin{array}{ll} \text{minimize} & ||a||_2 + \gamma(\mathbf{1}^T u + \mathbf{1}^T v) \\ \text{subject to} & x_1^T a - b \geq \quad 1 - u \\ & x_2^T a - b \leq -(1 - v) \\ & u > 0 \\ & v > 0 \end{array} \tag{11}$$

notice that the constraints are represented as $x^T a$, fundamentally this is the same as $a^T x$ but is necessary in the minimization for the vector a to be a column vector.

3 Image Set Formulation

The mnist data set is comprised of handwritten digits in the range 0 to 9. For each image in the set, there is an accompanying label that will be used to

determine if the classifier correctly identified the images as well as training the model. Each image is represented as a row vector of a matrix,

$$Images = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \\ x_4^T \\ x_5^T \end{bmatrix} \tag{12}$$

in the training set there are 60,000 images of size 28x28 which corresponds to a matrix size (60,000x784). In order to use a binary classifier (data separated by a hyperplane), the images are sorted according to their label to each set, totalling to 10 different matrices. To construct a binary classifier, the optimization problem is set up to solve for the coefficients of the hyperplane that separates two sets of digits. For this to be uniform, the smaller of the two digits is setup in the first constraint, being greater than the hyperplane, and the second digit is setup to be less than the hyperplane. For an example comparing the digits 0 and 1, this corresponds to the inequalities

$$\begin{aligned} X_0^T a - b \geq \quad & 1 - u \\ X_1^T a - b \leq & -(1 - v) \end{aligned} \tag{13}$$

where X_0 corresponds to the matrix of digits labelled as 0 and X_1 corresponds to the matrix of digits labelled as 1. Recall from the minimization in equation 11, u and v are slack variables that are weighted according to a factor γ to allow the hyperplane to deviate from a unit margin. Each row of the matrices correspond to an individual feature in the respective labeled set. This procedure is repeated until all permutations of the digits are accounted for, resulting in 45 total hyperplane coefficients vectors.

4 Setting up the Linear Program

The optimization solver used in this analysis is the auxiliary convex optimization solver CVX used in $Matlab$, although this procedure could be performed with the $Matlab$ routine *linprog*, the additional work necessary to set up the constraint in the matrix is extensive and is more readily suited for the text based setup of CVX. The solver can utilize both equality and inequality constraints as inputs but was configured according to the linear program formulation shown in equation 11. In addition to setting up the

optimization, the solver was changed from SDPT3 to sedumi, as it seemed to work better in this formulation. The coefficients a and b were saved in a 10x10 matrix of cells, with each individual cell containing the optimization parameters for the comparison of row m to column n, where m was chosen to represent the smaller of the two numbers. According to the methods used in determining the coefficients, in order to use the classifier properly, the smaller of the two digits was required by definition to reside in the first constraint. This method was reflected in the cell matrix as well, and subsequently the cell matrix is upper diagonal. For example, the comparison of the digits 0 and 1 corresponds to coefficients in the cell $A\{1, 2\}$ due to $Matlab$ indexing beginning at 1 rather than 0. The same format is used for the coefficients b, stored int the cell matrix B.

5 DAG

A directed acyclic graph, or DAG, is used to determine the outcome of the classifier. The process of elimination follows from 10 evaluations, with the final determining the most likely digit that the input image represents. A representation of the DAG is shown graphically in figure 2. Looking at this method from the perspective of the problem, each binary classification corresponds to evaluating the image with the coefficients contained the representative cell. The starting point for the classifier is the top right cell in the matrix that compares the digits 0 and 9. Incorporating the DAG in the sense of the cell of matrices has some useful intuition. If the matrix multiplication results in a scalar value greater than zero, it follows the logic that the input inmage is not the second element in the comparison. This relation manifests itself in decrementing the column of the cell matrix to continue evaluating the cells of binary classifiers representing the digit in row m. The converse is true for the matrix multiplication resulting in a scalar value less than zero, this corresponds to moving down the other path in the DAG, and in the sense of the cell matrix, incrementing the rows. By following this process, the classifiers continue evaluating the more likely option against the rest of the digits in the set. At the 10th iteration of the classifier, the evaluation corresponds to the diagonal cells of the matrix and the outcome represents the most likely digit that the input image represents.

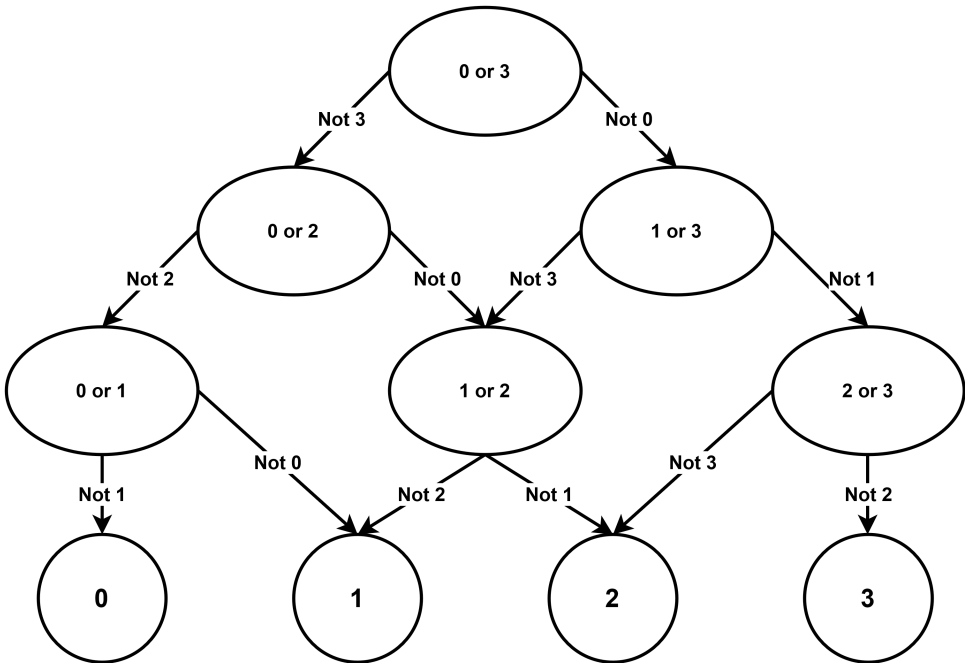

Figure 2: A DAG for an example of binary classifiers of digits ranging 0 to 3. The principle can be extrapolated to the full size of the mnist data set.

6 Results

The problem is fully defined and can be solved using linear programming with CVX. The solution vector obtained contains the hyperplane coefficients a and the offset coefficients b, which are stored in a cell matrix of size 10x10. For the training data, this means performing calculation based on a feature of size 784 and iteratively solving for the optimal solution based on the cost function. The weighting coefficient for the margin γ is chosen ahead of solving but produces varying results depending on the value. In the instance of this project, the value of is chosen to vary over a logarithmic range until reasonable accuracy is obtained and further granularly iterated to the desired accuracy. In the end, a value of $\gamma = 10^{-4}$ was chosen and produced an accuracy of 94.58%. The chart below outlines the various iterations and percent accuracy until converging to an acceptable classifier.

γ	Accuracy (%)
100	91.63
10	91.63
1	91.73
0.1	91.75
0.01	91.87
1e-3	93.03
4e-4	93.69
1e-4	94.58
1e-5	94.06

As seen in the table above the value of $\gamma = 10^{-4}$ performed best out of the binary classifiers in the DAG, although all of the configurations were able to correctly classify the images with an accuracy over 90%. This performance was based on the test images from the data set which consisted of 10,000 images in the mnist data set. Although once trained, the classifier was able to classify an image under 10^{-5} seconds, the time required to train each set of hyperplane parameters took on the order of 4 hours.

7 Downsampling by Interpolation

To reduce the time required for training, adjustments could be made to the feature set in the form of filtering or reduced sampling. One method of

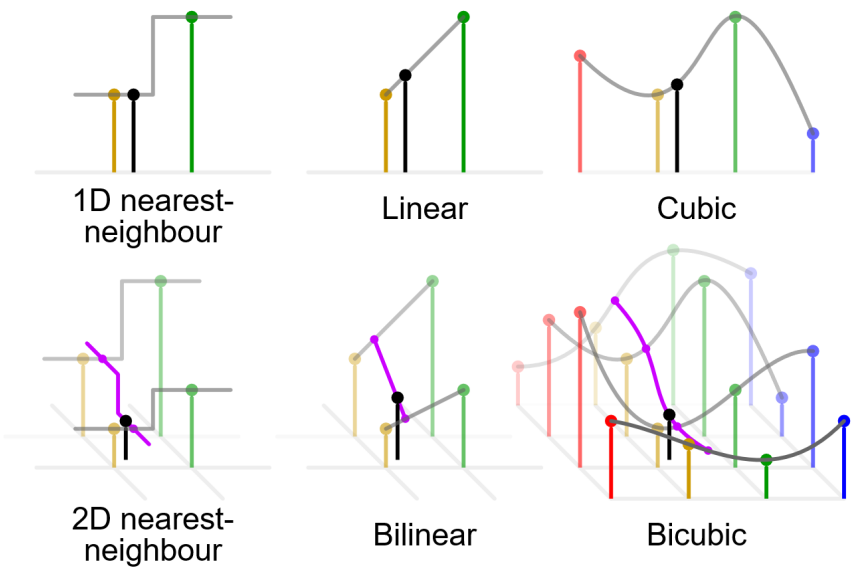

Figure 3: "Comparison of 1D and 2D interpolation" by cmglee from wikipedia under CC BY-SA 4.0, cropped to fit document format. Bicubic interpolation is the default for image resize function in *matlab*.

reducing the number of calculations is to resize the image by some factor. There are several methods to perform this operation, all of which involve some form of interpolation. The most simple being the nearest neighbor approach, where the position of a pixel is reduced by a factor and takes on a value of the nearest neighboring pixel. The most complex involves bicubic interpolation, where each row and column are interpolated by cubic functions. The bicubic method is utilized as the default for the *matlab* function *imresize*.

This method of interpolation was used to reduce the size of an image by a factor of 4 and by a factor of 16. The feature size is reduced from a row vector of 784 to vectors of size 196 and 47. Using the reduction significantly decreases the computation time required to determine the coefficients of the hyperplane. For the analysis, a value of $\gamma = 4e - 4$ was chosen. An example of the reduction can be seen in the following figure

The scaling factors produce an image similar to the original. With a scaling factor of 0.5 in each direction or a total reduction of 4, the content of the image is still visible. However reducing the original image size by 16 results in blurred image vaguely resembling the original. In the table below,

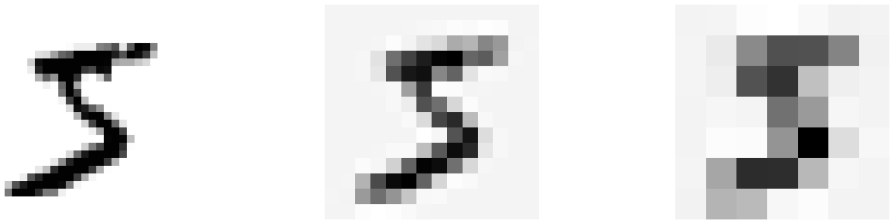

Figure 4: The bicubic interpolation to reduce the size of the image. (left) Original, (center) scaled by 0.25, (right) scaled by 0.0625. The image is the first element of the mnist dataset.

the time required to obtain the hyperplane coefficients from the training data is compared with the accuracy of the model inference for a $\gamma = 4e - 4$.

Scale	Time (minutes)	Accuracy (%)
1	180+	93.69
0.5	60	94.64
0.25	9.2	93.20

Notice the time elapsed for the solver is significantly decreased particularly for the reduction of 16. Interesting enough, the performance is similar to the full size images, likely due to the accuracy of the bicubic interpolations. In the case of the 0.5 scaling, the performance improved on the mnist data set compared to the original images.

8 Conclusion

Utilizing the mnist digit data set, binary classifiers were created by linear programming to solve for the coefficient of the hyperplanes. The margin was tuned according to a weighting parameter γ, the optimal solution of those tested occurred with a $\gamma = 10^{-4}$ to obtain a DAG that possessed an accuracy of 94.58%. The proportion of training data to testing data was 6:1, for a total set of 70,000 images. The image size (28x28) pixels was reduced by downsampling and bicubic interpolation through the *matlab* routine *imresize*. The scaling was reduced by 2 and 4, for an overall image reduction of 4 and 16 times respectively. The images, shown in figure 3

are representative of the quality obtained with the reduction. Although the image quality was degraded, the filtering significantly reduced the processing time of determining the hyperplane coefficients from the CVX optimization. The reduction factor of 2 actually increased the accuracy of the DAG to 94.64% for a weighting factor $\gamma = 4e - 4$. The processing time also scaled according to the image reduction, producing hyper plane coefficients at rates that were 4 to 16 times faster.

A Code

A.1 Training Function

```
%% Classification Project - Training
% Riley Kenyon
% MEID:272-513
% 04/24/2020
%--------------------------------------------------------------------------
close all; clear all; clc;

%% Import data
save_file = 'DAG_KENYON_BAK.mat';    % File to save coefficient cells A,B
load mnist.mat                       % Included is 60,000 sample images, and 10,00
load DAG_Kenyon.mat                  % Filtered dataset

if exist('images_play')             % Use filtered data if possible
   clear images
   images = images_play;
end

% Pre-allocate cells
idx = cell(10);
data = cell(10);
A = cell(10,10);
B = cell(10,10);

% Iteration parameters beginning from 0 idx
N = 9;
```

```
% Set weighting function gamma
g = 0.0001;

%% Sort images into categories - each row correspond to an image
for i = 0:9
    idx{i+1} = find(labels==i);                 % Find all zeros
    data{i+1} = double(images(idx{i+1},:));     % Re-assign all images for idx (ne
end

% Check
plotImg(idx{1}(1),images,labels);               % Visualize

%% Construct matrices for determining a and b
tic
for k = 1:length(g)
gamma = g(k);                       % For gamma iteration

for ii = 0:N
    for jj = 0:N
        if jj <= ii
            continue
        else
            A0 = data{ii+1};
            m0 = size(A0,1);
            A1 = data{jj+1};
            m1 = size(A1,1);
            n = size(A0,2);
            b = ones(m0,1);
            cvx_solver sedumi
            cvx_begin
            variables a(n) b u(m0) v(m1)
            minimize( norm(a,2) + gamma*(ones(1,m0)*u + ones(1,m1)*v) )
            subject to
                A0 * a - b >=  1 - u;       % Lower digit
                A1 * a - b <= -1*(1-v);     % Higher digit
                v > 0;
                u > 0;
```

```
            cvx_end
            A{ii+1,jj+1} = a;
            B{ii+1,jj+1} = b;
        end
    end
end
toc

% Save coefficients
save(save_file,'A','B');

end

function [] = plotImg(nHold,images,labels)
% Function for plotting a Handwritten digit
N = sqrt(length(images(1,:)));
disp(N)
IMAGE = reshape(images(nHold,:),N,N)';
disp(labels(nHold))
figure,imagesc(IMAGE)
colormap(flipud(gray(256)))
axis equal
set(gca, 'YTick', []);
set(gca, 'XTick', []);
axis off
end
```

A.2 Testing Function

```
    %% Classification Project - Testing
% Riley Kenyon
% MEID:272-513
% 04/14/2020
%-------------------------------------------------------------------------
clear all; close all; clc;
useFilteredData = true;     % Change to true to use Ap and Bp

% Load test data and hyperplane coefficients
```

```
load mnist.mat              % interested in the 10,000 test images
load DAG_Kenyon.mat         % Load coefficients

idx = cell(10); data = cell(10);        % Pre-allocate cells

if useFilteredData == true
   clear images_test A B
   images_test = images_test_play;
   A = Ap;
   B = Bp;
end

correct = 0;    % counter

% Determine accuracy of classifier
tic
for k = 1:length(images_test)
    im = double(images_test(k,:)); % import image

    % Create DAG
    ii = 0;             % First Digit
    jj = 9;             % Last Digit
    N = 9;              % Total allowable comparisons

    count=0;            % Initialize comparison count

    while count < N
        % +1 to account for matlab index starting at 1 rather than 0
        val =  im*A{ii+1,jj+1} - B{ii+1,jj+1};

        if val > 0
            % More likely to be ii than jj -> index -jj
            jj = jj-1;
        else
            % More Likely to be jj than ii --> index ii
            ii = ii+1;
        end
```

```
        % Log comparison
        count = count +1;
    end

    % More likely to be ii
    if val > 0
        prediction = ii;

        % More likely to be jj
    else
        prediction = jj;
    end

    % Increment counter if prediction is correct
    if prediction == labels_test(k)
        correct = correct+1;
    end
end
toc

% Output stats
fprintf("Accuracy %0.2f\n",correct/k);
fprintf("Number Correct %0.0f\n",correct);
```

A.3 Custom Scaling Function

```
    %% Classification Project - Image Scaling
% Riley Kenyon
% MEID:272-513
% 04/24/2020
%--------------------------------------------------------------------------
close all; clear all; clc;

%% Load dataset and configure
load mnist.mat

% Set scaling for resizing and interpolation
scale = 4;
```

```
factor = 1/scale;

% Filter dataset
[m,n] = size(images);
images_play = zeros(m,n*factor^2);
images_test_play = zeros(length(images_test),n*factor^2);
for ii = 1:length(images)
   images_play(ii,:) =  filterImg(images(ii,:),factor);
end
for jj = 1:length(images_test)
   images_test_play(jj,:) = filterImg(images_test(jj,:),factor);
end

function[IMAGE] = filterImg(img,factor)
% Function to resize image using bicubic interpolation
N = sqrt(length(img));
IMAGE = reshape(img,N,N)';
IMAGE = imresize(IMAGE,factor)';
IMAGE = reshape(IMAGE,[1,(N*factor)^2]);
end
```

A.4 Plotting

```
    %% Making figures for report
clear all; close all; clc;

% Hyperplane analogy
f = @(a,x,b) a.*x+b + randn([length(x),1]);
x = [0:0.1:10]';
a = -1;
b1 = 10;
b2 = 3;
b_mid = (b1+b2)/2;
m1 = (b2+b_mid)/2;
m2 = (b_mid+b1)/2;

data1 = f(a,x,b1);
data2 = f(a,x,b2);
```

```
figure, hold on
plot(x,data1,'o','MarkerFaceColor','b','MarkerEdgeColor','b')
plot(x,data2,'o','MarkerFaceColor','r','MarkerEdgeColor','r')
plot(x,a*x+(b1+b2)/2,'g','LineWidth',3)
plot(x,a*x+m1,'k--','LineWidth',3)
plot(x,a*x+m2,'k--','LineWidth',3)
legend('B','R','Hyperplane','Margin','box','off')
box('on')
```