Optimal Design Project 4-
Image Classification with Quadratic Programming

Cameron Sprenger

May 2024

1 Introduction

This project is a continuation of the first project from this class which was the classification of handwritten digits from the *EMNIST* dataset using a least squares classifier and a "1 vs. the rest" classification structure. The same training and testing sets will be used in order to directly compare the performance of the two classification methods. The pixel data was preprocessed in the same way as the previous project. The values were converted from values from 0-255 to decimals between 0 and 1. And, in order to reduce the dimensionality of the problem, the pixels on the border of the image were removed as well since they don't contain any information and therefore shouldn't affect the classification accuracy. The accuracy of the models and the training time will be compared to determine which method is best for certain applications. The training set is made of 60,000 sample images and labels, and the testing set contains 10,000 sample images and labels. For this project, the problem will formulated as a quadratic program and solved using Matlab's *quadprog* function which takes in inputs of (H,f,A,b). The form of *quadprog* is:

$$\begin{aligned} min_x \quad & 1/2\, x'Hx + f'x \\ & S.T.: Ax \leq b \end{aligned} \tag{1}$$

The images will be classified using a directed acyclic graph (DAG) which is a method of using n choose 2 binary classifiers to sort the images where $n = 10$ which of the number of unique digits. Each of the classifiers are only trained on two digits because it makes a decision on whether an image is not a digit or not another. For example, the classifier [3,5] would check if an image is not a 3, or not a 5 based on the sign of the output. Not-a-3 would be signified with a negative output, and not-a-5 would be a positive output from the classifier. Depending on the decision of the classifier, the image then moves down the classification structure and is tested by a different classifier determined by the previous decision. The entire of structure of the DAG classification structure for the number 0 through 9 is shown below in figure **??**.

One of the advantages to this classification structure over the 1-vs-rest is that each of the binary classifiers in the DAG is more accurate than the 1-vs-rest classifiers. This is because in the DAG, the classifiers are only separating two clusters of data from each other instead of separating a single cluster from all of the other data which can lead to false classifications. The margin is a measure of the separation between the data and the support vector machine (SVM) or separating line, and the larger the margin, the better the accuracy of the classifier is. The images contain 676 features which can be plotted in R^{676}, so visualizing the data and the separating hyperplanes is impossible. A 2-D representation of the different separating methods is shown in figure 1.

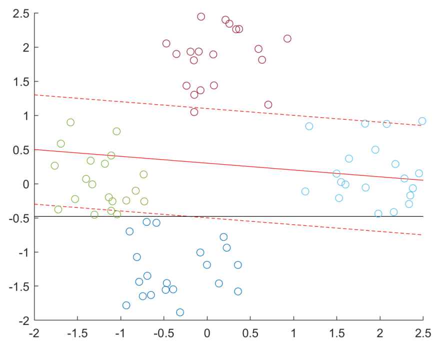


Figure 1: The difference in separation of data between the two methods, DAG and 1-vs-rest. The DAG separating line between blue and maroon is shown in red, and the margins are represented by the dotted lines. The 1-vs-rest separating line between blue-vs-rest is shown by the black line. The change that an image is classified incorrectly with the 1-vs-rest classifier is much higher than with the DAG.

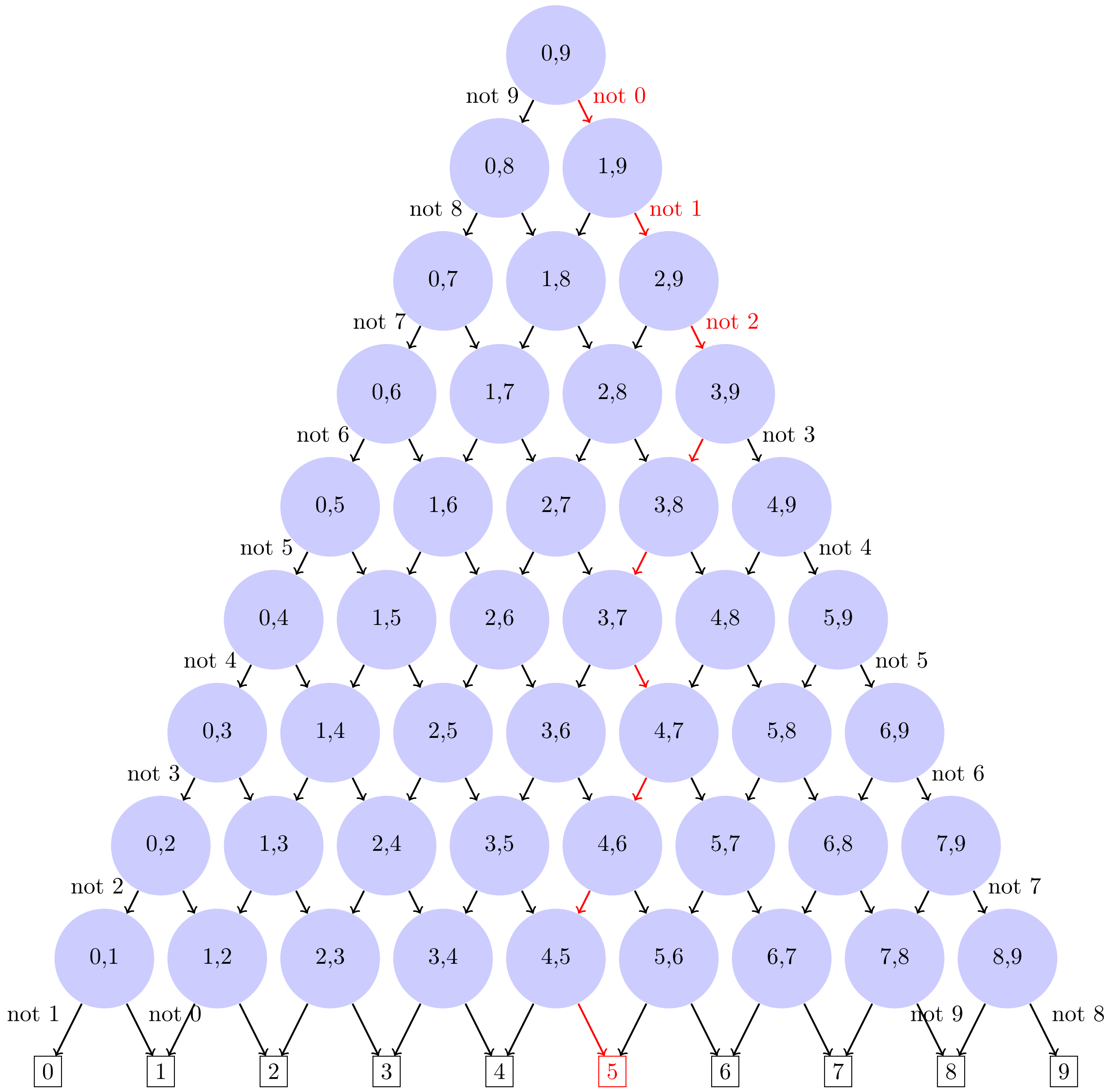


Figure 2: Directed Acyclic Graph (DAG) comprised of the 45 binary classifiers. The arrows indicate a movement in a direction based on the sign of the score of an image. Movement **right** is a positive score, and movement **left** is a negative score. The red arrows indicate a possible path an image can take to be classified as a 5.

2 Formulation

As mentioned in the introduction, in order to create a more accurate classifier, the margin between the separating plane and the data needs to be maximized. The definition of what the margin is needs to be better defined in order to be turned into an equation. The margin is twice the perpendicular distance between a line of best fit and the closest data points of the different clusters. In figure **??**, the margin is the distance between the two dotted lines. In the figure there is a single maroon circle in the margin which should technically not be there, but that is simply a limitation of creating random plots and forcing a margin on it. The actual margin is calculated to fit perfectly between 2 or 3 data points from the two datasets. The equation for the margin is given in the problem statement as:

$$max \text{ Margin} = \ 2/\|a\|_2 \tag{2}$$

As seen in equation 1, *quadprog* solves the minimization of the cost function, so the maximization of the margin turns into:

$$\begin{aligned} min \ \|a\|_2 \\ \text{or: } min \ a^T a \end{aligned} \tag{3}$$

This formulation would work well assuming there are no outliers in the data, as outlier can greatly reduce the margin if they are in the region between the two clusters. To allow the line of best fit and margins to have some flexibility, additional terms can be added. Often referred to as slack variables, these terms can be added to the cost function to control how strict the cost function is. Slack variables can be added as representing the distance each sample is from the margin lines. This loosens the constraint that the margins have to fully sit in between the two data sets by allowing points to sit in that region, it only comes at the expense of adding to the cost function. A separating hyperplane of n-dimensions can be represented as $A^T x = b$, and the margins in either side being $A^T x = b \pm 1$. Excluding erroneous outliers, all of the samples from a cluster will fall outside of the margin lines on either the positive of negative sides. In equation form this will look like:

$$\begin{aligned} A_1^T x \geq b + 1 \\ A_2^T x \leq b - 1 \end{aligned} \tag{4}$$

Samples from one cluster will be on the positive side of the margin, and samples from the other cluster will be in the negative side of the other margin.

Moving b to the other side, and adding the slack variables results in:

$$\begin{aligned} A_1^T x - b \geq 1 - u \\ A_2^T x - b \leq 1 + v \end{aligned} \tag{5}$$

With the addition of the slack variables, the cost function needs to be updated to incorporate the penalty of the margins not being fully between the data sets. In order to do this, the values of u and v can be added to equation 3 with a coefficient to control the weighting of the slack terms. Doing so results in the updated cost function of:

$$min \ a^T a + \gamma(u_i + v_j) \tag{6}$$

where i and j are the number of samples in each of the two clusters.

Gamma allows the cost function to be tuned to yield the best accuracy because when gamma equals 1, the relationship between $A^T x$ and $u_i + v_j$ is somewhat random and giving less control over the cost function.

The constraints that u and v are positive need to be added to the subject to matrices, as well as the conditions for training. Equation 5 can be rearranged to have all of the unknowns on one side, and the knowns on the other. In this form, the matrices can be solved by *quadprog* and the classifier can be trained. Rearranging equation 5 into $A^T x \leq b$ results in:

$$\underbrace{\begin{bmatrix} -A_1 & 1 & -\vec{1} & \vec{0} \\ A_2 & -1 & \vec{0} & -\vec{1} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \vec{a} \\ b \\ \vec{u} \\ \vec{v} \end{bmatrix}}_{x} \leq \underbrace{\begin{bmatrix} -\vec{1} \\ -\vec{1} \end{bmatrix}}_{b} \tag{7}$$

A_1 and A_2 are matrices that contain the pixel values of the images that the classifier is being trained on. Using the example from earlier, classifier [3,5] would take in A_1 as all of the pixels labeled as an image of 5, and A_2 would be all of the pixels labeled as an image for a 3. The data sets for each image aren't the same size, so the height of the rows in equation 7 are different. The dimensionality of the rows are shown below. Each row of the matrix is a single equation representing the training of a single image based on the pixels.

$$\begin{bmatrix} M_{pos}\text{x}676 & m_{pos}\text{x}1 & M_{pos}\text{x}M_{pos} & M_{pos}\text{x}M_{neg} \\ M_{neg}\text{x}676 & m_{neg}\text{x}1 & M_{neg}\text{x}M_{pos} & M_{neg}\text{x}M_{neg} \end{bmatrix} \tag{8}$$

Where M_{pos} and M_{neg} are the number of images in the dataset for 5 and 3 respectively from the example above.

Adding the non-negativity constraint is simply:

$$\underbrace{\begin{bmatrix} \vec{0} & 0 & -I(M_{pos}) & \vec{0} \\ \vec{0} & 0 & \vec{0} & -I(M_{neg}) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \vec{a} \\ b \\ \vec{u} \\ \vec{v} \end{bmatrix}}_{x} \leq \underbrace{\begin{bmatrix} \vec{0} \\ \vec{0} \end{bmatrix}}_{b} \tag{9}$$

Combining the two constraints together for a final A_{hat} and b_{hat} is just a matter of stacking the matrices in equation 7 and 9.

$$\underbrace{\begin{bmatrix} -A_1 & 1 & -\vec{1} & \vec{0} \\ A_2 & -1 & \vec{0} & -\vec{1} \\ \vec{0} & 0 & -I(M_{pos}) & \vec{0} \\ \vec{0} & 0 & \vec{0} & -I(M_{neg}) \end{bmatrix}}_{A_{hat}} \underbrace{\begin{bmatrix} \vec{a} \\ b \\ \vec{u} \\ \vec{v} \end{bmatrix}}_{x} \leq \underbrace{\begin{bmatrix} -\vec{1} \\ -\vec{1} \\ \vec{0} \\ \vec{0} \end{bmatrix}}_{b_{hat}} \tag{10}$$

In order to formulate the cost function from equation 6 into the matrix form in equation 1, an H matrix and f vector need to be formulated to convert between the two. Fortunately, the form of the cost function used by *quadprog* is already very similar to the cost function in equation 6.

$$\min\ a^T a + \gamma(u_i + v_j) \ \Rightarrow \quad \min\ 1/2 x^T H x + f^T x \tag{11}$$

H just needs to be a variation of an identity matrix with ones in the elements that align with the elements of a_i. The identity is multiplied by 2 to account for the 1/2 in front of the cost function in *quadprog*. This takes the norm of just a and not the other variables in x. f just distributes across x and since we are only interested in u and v, f is a vector of zeros in the elements that align with a_i and b and gammas in the elements that align with u and v.

$$\min\ a^T a \quad \Rightarrow \quad \min\ 1/2 \begin{bmatrix} \vec{a} \\ b \\ \vec{u} \\ \vec{v} \end{bmatrix}^T \begin{bmatrix} 2 * I(676) & \vec{0} \\ \vec{0} & \vec{0} \end{bmatrix} \begin{bmatrix} \vec{a} \\ b \\ \vec{u} \\ \vec{v} \end{bmatrix} \tag{12}$$

$$\min\ \gamma(u_i + v_j) \Rightarrow \quad \min\ \gamma \begin{bmatrix} \vec{0} \\ 0 \\ \vec{1} \\ \vec{1} \end{bmatrix}^T \begin{bmatrix} \vec{a} \\ b \\ \vec{u} \\ \vec{v} \end{bmatrix} \tag{13}$$

Now that all of the inputs to *quadprog* have been formulated, it is just a matter of plugging them into the function and solving. The function returns x^* which is a vector with elements $[a, b, u_i, v_j]$, so each classifier is defined as $A = x^*(1:676)$, and $b = x^*(677)$.

At this stage, gamma can be iterated to generate trained models for each value. These models can then be tested against each other and the one with the highest accuracy will be used. Since gamma could be any number, the models first train based on order of magnitude. A *for* loop is used to train the model 6 times starting with $\gamma = .001$ to $\gamma = 100$ or $\gamma = 10^{-4+i}$ where $i = 1:6$. Hopefully this will cover a wide enough range so more fine tuning can be done once a general order of magnitude is found. Training all 45 classifiers takes more than an hour, so by efficiently iterating gamma, a lot of time can be saved compared to guessing and checking.

3 Testing and Results

Once all of the classifiers are trained, then the testing dataset of images can be passed through the DAG and classified. All of the classification can be done in a single nested *for* loop. The outer loop iterates from 1 to 10,000 for each image in the testing dataset, and the inner loop iterating from 1:9 for the 9 different levels of the DAG classifier in figure 2. As mentioned earlier, if a classifier scores an image positively, then the image moves down and to the left, and if an image is scored negatively, then the image moves down and to the right. A way to simplify the motion of the DAG is to slightly rotate the matrix so instead of moving down rows, an image trickles down the DAG from the top right to the diagonal. This simplifies the motion because instead of moving down, or down and over, an image just moves over or down. The matrices with the different DAG structures are below.

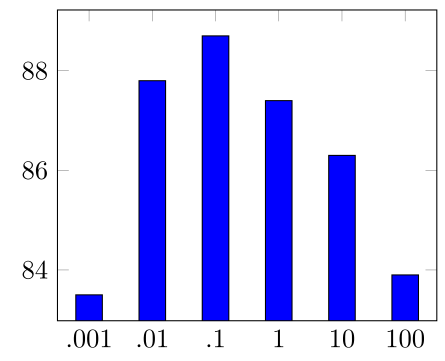
$$\left[\begin{array}{cccccccccc}
0,9 & & & & & & & & & \\
0,8 & 1,9 & & & & & & & & \\
0,7 & 1,8 & 2,9 & & & & & & & \\
0,6 & 1,7 & 2,8 & 3,9 & & & & & & \\
0,5 & 1,6 & 2,7 & 3,8 & 4,9 & & & & & \\
0,4 & 1,5 & 2,6 & 3,7 & 4,8 & 5,9 & & & & \\
0,3 & 1,4 & 2,5 & 3,6 & 4,7 & 5,8 & 6,9 & & & \\
0,2 & 1,3 & 2,4 & 3,5 & 4,6 & 5,7 & 6,8 & 7,9 & & \\
0,1 & 1,2 & 2,3 & 3,4 & 4,5 & 5,6 & 6,7 & 7,8 & 8,9 & \\
\hline
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
\end{array}\right] \tag{14}$$

Rotates and turns into:

$$\begin{bmatrix}
0 & 0,1 & 0,2 & 0,3 & 0,4 & 0,5 & 0,6 & 0,7 & 0,8 & 0,9 \\
 & 1 & 1,2 & 1,3 & 1,4 & 1,5 & 1,6 & 1,7 & 1,8 & 1,9 \\
 & & 2 & 2,3 & 2,4 & 2,5 & 2,6 & 2,7 & 2,8 & 2,9 \\
 & & & 3 & 3,4 & 3,5 & 3,6 & 3,7 & 3,8 & 3,9 \\
 & & & & 4 & 4,5 & 4,6 & 4,7 & 4,8 & 4,9 \\
 & & & & & 5 & 5,6 & 5,7 & 5,8 & 5,9 \\
 & & & & & & 6 & 6,7 & 6,8 & 6,9 \\
 & & & & & & & 7 & 7,8 & 7,9 \\
 & & & & & & & & 8 & 8,9 \\
 & & & & & & & & & 9
\end{bmatrix} \tag{15}$$

The DAG in equation 15 allows for an image to move through the structure with a single move. An image starts in the top right of the matrix and moves either left one, or down one based on the sign of the score from the classifier. After 9 classifications, the image ends up on the diagonal of the matrix, so the guess can be read as either the column-1 or row-1 of the image. The minus 1 is due to the indexing in the matrices as 0 is in position 1x1, 1 is in 2x2, and 9 is in 10x10.

The 6 values of gamma can be compared by plotting the gamma against the accuracy of the classifiers. The gamma with the highest accuracy will be chosen as the "optimal" value gamma. This value of gamma isn't the global optimal value, but out of the values that the models were trained, it yielded the highest accuracy. Below is a graph showing the relation between γ and the predictive accuracy. From the plot below, $\gamma = .1$ has the highest classification accuracy of 88.7%. So that value will be used for the final classification of the data and the rest of the results section.



	0	1	2	3	4	5	6	7	8	9	Total
0	972	0	10	3	1	19	13	4	7	7	1036
1	0	1026	1	0	0	1	1	1	1	3	1034
2	1	1	899	1	4	6	0	16	2	0	930
3	2	6	30	939	1	104	2	16	6	7	1113
4	0	1	8	1	911	15	3	8	3	16	966
5	0	1	2	6	1	446	3	0	2	2	463
6	1	5	23	5	15	24	928	2	6	0	1009
7	0	1	3	3	0	1	0	891	0	4	903
8	4	94	56	51	14	268	8	52	947	61	1555
9	0	0	0	1	35	8	0	38	0	909	991
	980	1135	1032	1010	982	892	958	1028	974	1009	10000

Confusion Matrix for classifiers trained on $\gamma = .1$. Rows indicate guessed classification, and columns indicate expected classification. Correct identifications are indicated by the number on the diagonal. The trace is the total number of correct classifications.

Analyzing the confusion matrix above shows that some of the digits were classified with a much higher accuracy than some other digits. For some reason, the classification of the digit 4 was much poorer than digits like 5.

4 Feature Engineering

Feature engineering is a method used to reduce the number of features of a sample to reduce computational time, or to add features to add to increase the accuracy of the model. For this project, the number of pixels that represent each image can be reduced with only a slight decrease in the overall accuracy of the classifier. In the initial project, the borders of each image were removed because they didn't contain any information for the image. This served to reduce the dimensionality of the problem from 784 down to 676 dimensions with no decrease in the accuracy. This can be taken further and more of the border can be removed because all of the images are centered so only a few pixels might be lost. This should hopefully greatly reduce the computational time it takes to train the models but retain nearly the same accuracy as the higher dimensional problem. A 4 pixel thick border was removed off of each image to bring the image sizes from 28x28 down to 20x20 which almost halves the number of pixels the represent each image. This form of feature engineering is relatively basic as it just cuts off the data on the sides of the images, but it could be refined further by iterating through what pixel locations contain important information and removing the ones that don't. In the textbook example from project 1, the borders and corners are removed because most of the digits are circular and don't extend into the corners of the images. This type of feature engineering would be more effective than simply removing the borders, but finding which pixels can be removed iteratively is computationally expensive and takes a long time.

Figure 3: Below is an example of the reduced image that has been reduced from 28x28 pixels down to 20x20 be removing a border of 4 pixels around the entire image. As you can see, the image is still clearly a 1 since there is enough information in the remaining data to determing the digit it represents.

The accuracy of the feature engineered classifiers is 77.2% which is quite a bit lower than than the original 88.7%. The thickness of the border that is removed can be reduced to increase the accuracy again.

5 Continuation

One of the problems with the DAG is if there is an ambiguous classification somewhere in the

In figure 2, diagonal lines can be drawn up from the bottom row so show the region for possible correct identification. As soon as an image is outside of that area, no matter what route it takes, it will never be classified correctly. This means the classification on the diagonal lines are important because they need to be correct, whereas the outcome of a classifier in the middle of the region doesn't effect the final classification as much. If you look at figure 2, and look at the paths 0 and 9 need to take to be correctly identified, you can see that all 9 classifications need to be made correctly, where as for a digit like 5, it may take a path that only has one critical decision right at the end. Thinking about this, the order of the numbers on the bottom can be rearranged such that numbers with high classification accuracy are on the outsides, and numbers with lower classification accuracy can sit in the middle.