MCEN 5125: Optimal Design

Least Squares Classification

Jacob Haimes

1 Introduction

Even now, after the technological breakthroughs of telecommunications and mobile computers, the United States Postal Service alone delivers well over 100 billion units of mail each year [1]. Imagining how many man-hours it takes to deliver all of this mail quickly makes it apparent that any way to automate even a portion of this process would be invaluable. One way to improve the efficiency of this task would be to use a computer to sort the addresses on each piece of mail. This, however, requires machine learning. Luckily, we can actually utilize a tool that we have, up to this point, primarily used for curve fitting: least squares optimization. In this paper we will attempt to sort images of handwritten digits using a least squares classifier.

2 Pre-Processing

2.1 The Data

Before we can begin to construct our classifiers, we first must obtain some set of data. For this task, we will be using a slightly pre-processed version of the Modified National Institute of Standards and Technology Database, which we will designate as `mnist` [2]. We then manipulate the data from `mnist` into a desirable form. Each provided datum is a linear representation of a 28×28 matrix. Each element within these matrices contains the `uint8` grayscale value for its corresponding pixel. When formatted correctly, each row can reconstruct an image of a handwritten digit. A column vector containing the correct digit for each image (row) is also provided in the data set. I wrote a simple function to visualize these pictures easily along with their intended label, `vis_dig`, which can be found in Appendix B.

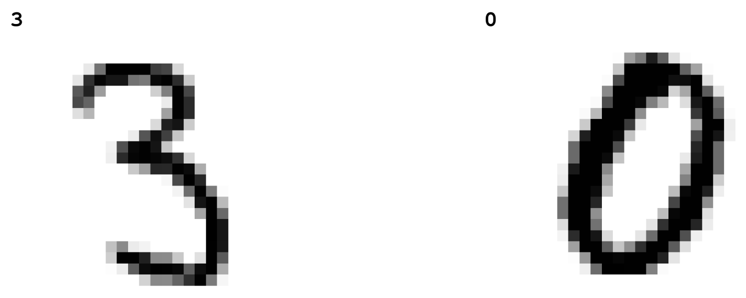


Figure 1: Plots of two images provided in the training data set, created by calling the `vis_dig` function. The label for each picture, as defined by the `mnist` data set, is northwest of both handwritten numbers.

Each image has a one pixel width border of padding in which all values are set to zero. These added pixels are not helpful to us, so we wish to remove them in order to reduce the number of extraneous features in our problem set. A visualization of the pixels that we will be removing can be seen in Fig. (2). Understanding the relation between the one dimensional array representation that we are provided and the two dimensional matrix representation that maps easily to the images will allow us to efficiently remove whatever pixels we desire.

2.2 Exploring Index Relations

When converting from a one dimensional array, P_{ℓ^2}, to a two dimensional array, $Q_{\ell\times\ell}$, Eq. (1) can be used to determine corresponding indices[1]. Similarly, Eq. (2) can be used to convert from two dimensional

[1]Eq. (1) and Eq. (2) are only true when converting between arrays and matrices whose indexing begins at 1.

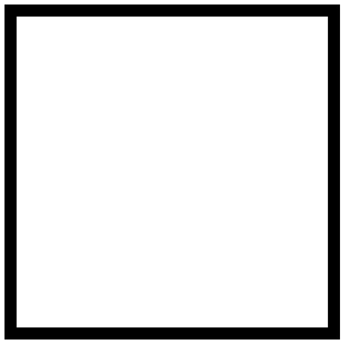


Figure 2: Visualization of the pixels that won't be considered in our solution to this problem. Pixels that will be removed are shaded. The removal of the shaded pixels reduces the number of pixels per image from 784 to 676.

indices to one dimensional indices. A visual representation of the pixel locations (v and w in an $\ell \times \ell$ matrix) their corresponding index in the provided form (u in a $1 \times \ell^2$ array) is given in Eq. (3).

$$g : P_u \to Q_{v,w}$$

$$\text{where } \ g(u) = (v, w) = \left((u - 1 \pmod{\ell}) + 1, \quad \left\lfloor \frac{u-1}{\ell} \right\rfloor + 1\right) \tag{1}$$

$$h : Q_{v,w} \to P_u$$

$$\text{where } \ h(v, w) = u = \ell(w - 1) + v \tag{2}$$

$$\begin{bmatrix} 1 & \ell+1 & 2\ell+1 & \cdots & \ell(\ell-1)+1 \\ 2 & \ell+2 & 2\ell+2 & \cdots & \ell(\ell-1)+2 \\ 3 & \ell+3 & 2\ell+3 & \cdots & \ell(\ell-1)+3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell & 2\ell & 3\ell & \cdots & \ell^2 \end{bmatrix} \tag{3}$$

With these tools, it is clear that our "border" indices in the two dimensional $\ell \times \ell$ matrix correspond to the following indices:

$$\begin{aligned} &\{1, 2, 3, \ldots, \ell\} \\ \cup\ &\{\ell+1,\ 2\ell+1,\ 3\ell+1, \ldots,\ \ell(\ell-1)+1\} \\ \cup\ &\{\ell,\ 2\ell,\ 3\ell, \ldots,\ \ell^2\} \\ \cup\ &\{\ell(\ell-1)+1,\ \ell(\ell-1)+2,\ \ell(\ell-1)+3 \ldots,\ \ell^2\}. \end{aligned}$$

Code to create an array whose value is 1 on the border and 0 otherwise (or vice versa) is trivial and can be seen in the `pre_proc` function in Appendix B. We then take advantage of logical indexing to remove all unwanted indices from our data.

3 Formulating the Least Squares Problem

3.1 Understanding Multi-Class Classifiers

Prior to using least squares to classify our ten digits, we must first understand how least squares can be used as a Boolean classifier. A Boolean classifier can be described as a function whose input is a vector s that contains n elements, and whose output is a single scalar value of ± 1. Constructing a Boolean classifier $\hat{f}$ using the least squares method begins with a training set, that is, a list of m true classifications $y^{(m)}$ corresponding to known vectors $s^{(m)}$. We now define a function $\tilde{f}(s^{(i)})$ to be some linear combination of p basis functions $f^{(j)}$, $j \in \{1, 2, \ldots, p\}$, that is

$$\tilde{f}(s^{(i)}) = \sum_{j=1}^{p} \theta^{(j)} f^{(j)}(s^{(i)}) + \nu, \tag{4}$$

where $s^{(i)}$ is a the i^{th} image in stack s, $\theta \in \mathbb{R}$, and $\nu \in \mathbb{R}$. Furthermore, we will choose all $\theta^{(j)}$, $f^{(j)}$, and ν such that $\tilde{f}(s^{(i)})$ minimizes the sum of squared error between $y^{(i)}$ and $\tilde{f}(s^{(i)})$ $\forall\, i \in m$. Mathematically, this can be written

$$\text{minimize} \quad \sum_{i=1}^{m} \left(y^{(i)} - \tilde{f}\left(s^{(i)}\right)\right)^2 . \tag{5}$$

The range of typical least squares optimization, however, is not binary, so we will define our Boolean classifier to be

$$\hat{f}\left(s^{(i)}\right) = \mathbf{sign}\left(\tilde{f}\left(s^{(i)}\right)\right), \tag{6}$$

where **sign** of some real valued constant z is

$$\mathbf{sign}(z) = \begin{cases} 1, & z \geq 0 \\ \text{-}1, & z < 0. \end{cases} \tag{7}$$

There are a few important aspects that we have not yet considered, the first being the choice of our basis functions, $f^{(j)}$. Note that now we are examining our specific problem, digit recognition; our choices for $f^{(j)}$, however, will generally be analogous to basis functions for other problems. Knowing that some linear combination of our basis functions must allow us to obtain every possible input image, the simplest set of basis functions would be

$$f^{(j)}\left(s^{(i)}\right) = \mathbf{val}\left(s^{(i,j)}\right), \qquad i \in \{1, 2, \ldots, m\}, \quad j \in \{1, 2, \ldots, n\}, \tag{8}$$

where $\mathbf{val}(s^{(i,j)})$ returns the normalized[2] `uint8` grayscale value of the pixel in $s^{(i)}$ at index j. In addition, we will also include a single basis function that is the sum of all pixel values in an image; the scaler value ν associated with this function can be thought of as a shift in the output dimension, much like the y-intercept b in the canonical equation for a line in two dimensions, $y = mx + b$. Although our classifier would still work with many more equations, which is leveraged for "feature engineering," we will stick to this simple option.

The other obstacle that we still must address is that there are ten categories in our digit classification problem, not two. Perhaps we will create ten different Boolean classifiers $\hat{f}_{(0)}, \hat{f}_{(1)}, \ldots, \hat{f}_{(9)}$. Each classifier $\hat{f}_{(k)}$ would then classify the input image $s^{(i)}$ as the digit k, or as *not* the digit k. This wouldn't work, as there would be a problem if two different classifiers, $\hat{f}_{(\alpha)}$ and $\hat{f}_{(\beta)}$, both determined that some input $s^{(i)}$ should be classified as digit α and digit β, respectively. We recall, however, that our function $\tilde{f}$ returns a real valued scalar, not a binary one, and that a larger positive value implies more certainty in a classification. Our multi-class classifier, $\hat{f}_M$, can then be defined

$$\hat{f}_M\left(s^{(i)}\right) = \mathbf{max}\left\{\tilde{f}_{(0)}\left(s^{(i)}\right), \tilde{f}_{(1)}\left(s^{(i)}\right), \ldots, \tilde{f}_{(9)}\left(s^{(i)}\right)\right\}, \tag{9}$$

where **max** returns the digit of the classifier, k, that produced the largest value for $\tilde{f}_k(s^{(i)})$.

[2] Because there are 256 possible values for a `uint8` number, normalizing is equivalent to dividing by 256.

3.2 Building the Binary Classifiers

Although we now understand where we are going with this approach, we are still missing one component: the actual values for the weighting of each pixel, $\theta_{(k)}$, and the constant shift $\nu_{(k)}$. Fortunately, finding these coefficients can be thought of as a least squares optimization problem of the form

$$Ax = b, \tag{10}$$

where

$$A = \begin{bmatrix} s^{(1)} & 1 \\ s^{(2)} & 1 \\ \vdots & \vdots \\ s^{(m)} & 1 \end{bmatrix} = \begin{bmatrix} s & \mathbb{1} \end{bmatrix}, \qquad x = \begin{bmatrix} \theta_{(k)}^{(1)} \\ \theta_{(k)}^{(2)} \\ \vdots \\ \theta_{(k)}^{(n)} \\ \nu_{(k)} \end{bmatrix} = \begin{bmatrix} \theta_{(k)} \\ \nu_{(k)} \end{bmatrix}, \qquad b = \begin{bmatrix} y_{(k)}^{(1)} \\ y_{(k)}^{(2)} \\ \vdots \\ y_{(k)}^{(m)} \end{bmatrix} = y_{(k)},$$

and

$$y_{(k)}^{(i)} = \begin{cases} 1, & \texttt{label}(i) = k \\ \text{-1}, & \text{otherwise.} \end{cases} \tag{11}$$

The well known solution to the least squares problem,

$$x^* = A^\dagger b, \tag{12}$$

can then be used to find the values of the constants that define a Boolean classifier for each digit. Visualizations of $\theta_{(k)}$ can be seen in Fig. (3).

Before moving, we examine these plots on a surface level for general trends. Interestingly, we can see that these heat maps seem to give an impression of their corresponding digit. Personally, I think that the $k = 0$ and the $k = 3$ heat maps create a shape most similar to their digit, while the $k = 9$ heat map is the least related. Another interesting occurence is that, in general, the more central pixels have a significantly smaller magnitude than those nearest the border padding.

4 Classifying the Data

Now that we understand our multi-class classifier $\hat{f}_M$ and our Boolean k classifiers $\tilde{f}_{(k)}$, we can use them to sort our data. We will designate our training data, the 60000 images that the classifiers are to be trained on, as `traim_im`, and our testing data, the 10000 images that will be used for testing, as `test_im`.

Recall that we must evaluate $\tilde{f}_{(k)}(s^{(i)})$ for each digit, and then compare their results to choose our final digit classification for image i. To efficiently calculate these values, Eq. (4) can be rewritten in the typical matrix form, $Ax = b$, as

$$\begin{bmatrix} \theta_{(k)}^{(1)} & \theta_{(k)}^{(2)} & \cdots & \theta_{(k)}^{(n)} & \nu_{(k)} \end{bmatrix} \begin{bmatrix} s^{(i,1)} \\ s^{(i,2)} \\ \vdots \\ s^{(i,n)} \\ 1 \end{bmatrix} = \tilde{f}_{(k)}\left(s^{(i)}\right). \tag{13}$$

Using terms that we have already defined, Eq. (13) can be simplified to

$$\tilde{f}_{(k)}\left(s^{(i)}\right) = \begin{bmatrix} \theta_{(k)} & \nu_{(k)} \end{bmatrix} \begin{bmatrix} s^{(i)} \\ 1 \end{bmatrix}. \tag{14}$$

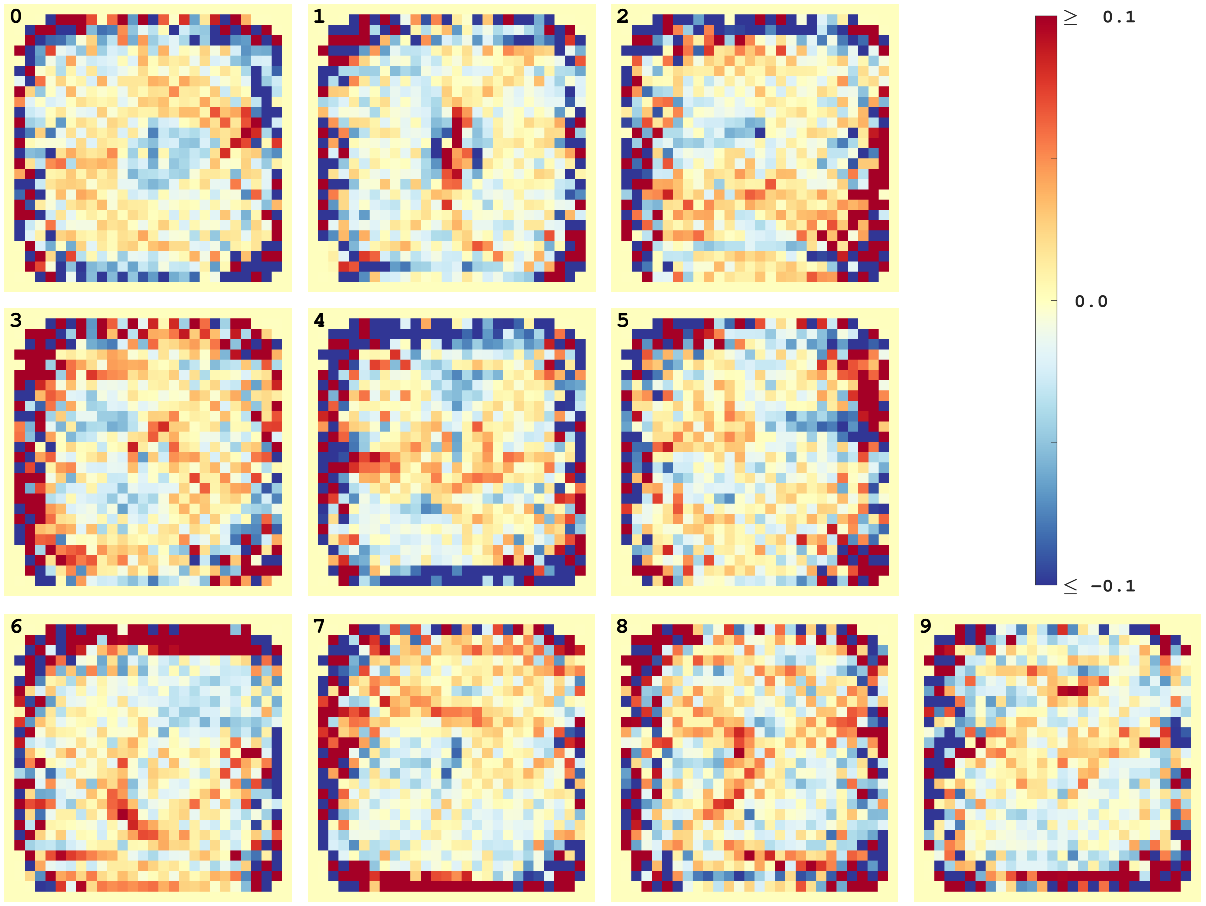


Figure 3: In this figure, the weighting of each pixel in the coefficient matrix $\theta_{(k)}$ for each digit $k \in \{0, 1, \ldots, 9\}$ is mapped to a color (northeast corner). The digit k can be seen in the northwest corner of its corresponding image. Note that we *have* included the one pixel width border that was removed during pre-processing, and that the value of each pixel in this set has been set to $\theta_{(k)}^{(j)} = 0$. The `colormap` used in this image originates from the ColorBrewer package.

Although this is an intuitive way to define $\tilde{f}_{(k)}(s^{(i)})$, it can be extended such that we can obtain the $\tilde{f}_{(k)}$ for all m images in a stack s with a single operation, which can be seen in Eq. (15).

$$\begin{aligned}\tilde{f}_{(k)}(s) &= \begin{bmatrix} s^{(1)} & 1 \\ s^{(2)} & 1 \\ \vdots & \vdots \\ s^{(m)} & 1 \end{bmatrix} \begin{bmatrix} \theta^{(1)}_{(k)} \\ \theta^{(2)}_{(k)} \\ \vdots \\ \theta^{(n)}_{(k)} \\ \nu_{(k)} \end{bmatrix} \\ &= Ax^*_{(k)} \end{aligned} \tag{15}$$

4.1 Training and Testing

We start by finding the values of each $\tilde{f}_{(k)}$(`train_im`) using Eq. (15). We also note that because $\hat{f}_M$ is a function of each Boolean classifier's results, the performance metrics of these classifiers can often be a useful tool when analyzing the performance of our multi-class classifier. Example results for $\hat{f}_{(0)}$(`train_im`), including a sorted histogram of the data, confusion matrix, and a table of performance metrics are seen in Fig. (4) and Tables I and II. Once we have obtained $\tilde{f}_{(k)}$(`train_im`) $\forall\, k$, we can determine our final classification for each image using Eq. (9). The confusion matrix for these results can be seen in Table III.

We then conduct the same categorization process using our least squares classifier, but this time it is operating on the testing image set, `test_im`. Example results for $\hat{f}_{(0)}$(`test_im`), including a sorted histogram of the data, confusion matrix, and performance metrics, are seen in Fig. (4) and Tables I and II. Similarly formatted histograms, confusion matrices, and performance metrics for all other Boolean k classifiers $\hat{f}_{(k)}$, operating on both the training and testing image sets, are located in Appendices C, D, and E, respectively.

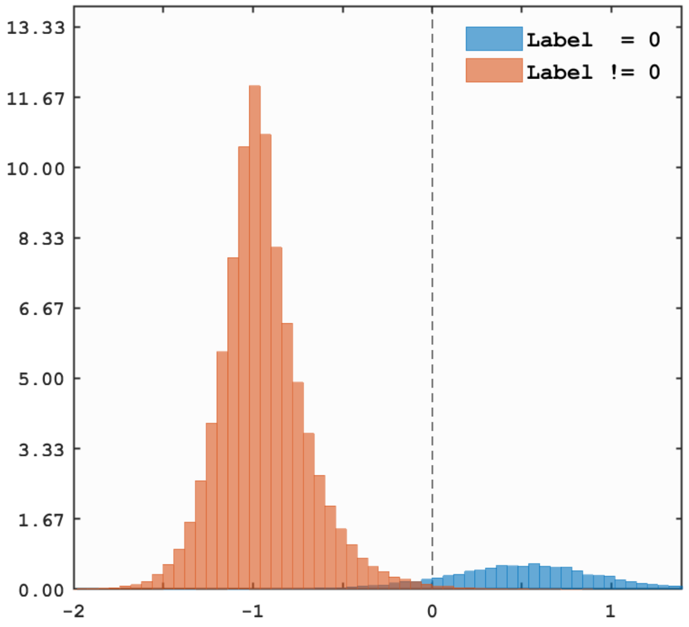
	Prediction		
Outcome	$\hat{f}_{(0)} = 1$	$\hat{f}_{(0)} =$ -1	**Total**
$y_{(0)} =$ 1	5344	316	5660
$y_{(0)} =$ -1	579	53761	54340
All	5923	54077	60000

(a) $\hat{f}_{(0)}$(`train_im`)

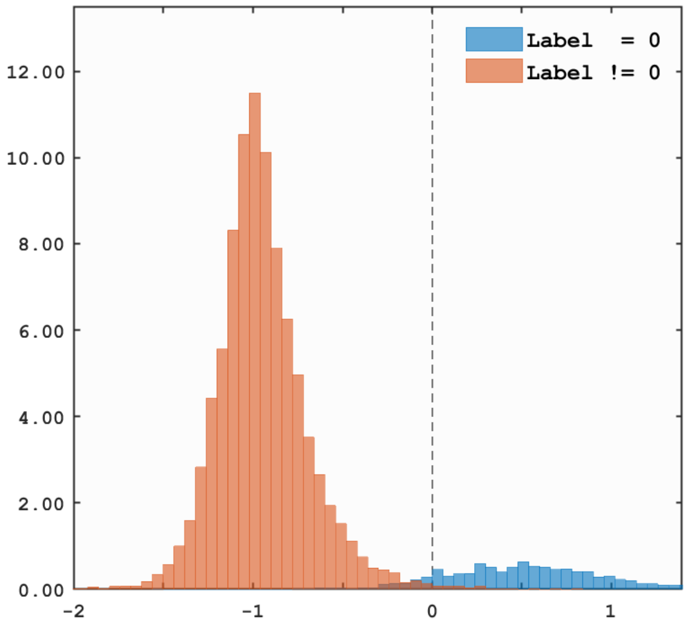
	Prediction		
Outcome	$\hat{f}_{(0)} = 1$	$\hat{f}_{(0)} =$ -1	**Total**
$y_{(0)} =$ 1	864	41	905
$y_{(0)} =$ -1	116	8979	9095
All	980	9020	10000

(b) $\hat{f}_{(0)}$(`test_im`)

Table I: Confusion matrices for the Boolean 0 classifier operating on the training (a) and testing (b) data. See Appendix A for more information regarding the contents of confusion matrices. Confusion matrices for other digits can be found in Appendix D.



(a) $\hat{f}_{(0)}(\texttt{train\_im})$



(b) $\hat{f}_{(0)}(\texttt{test\_im})$

Figure 4: A categorized histogram of the results of the Boolean 0 classifier operating on the training (a) and testing (b) images. For these plots, the count of each *blue* bin is equivalent to the percent of images in the input data set that are labeled 0 and have an $\tilde{f}_{(0)}$ value within its bounds. Similarly, *orange* bin counts are the percent of images in the input data set that are *not* labeled as 0 and have an $\tilde{f}_{(0)}$ value within its bounds. All images that fall on the positive side of the $\tilde{f}_{(0)} = 0$ dotted line are categorized as 0 by our Boolean 0 classifier $\hat{f}_{(0)}$, while all images on the negative side are categorized as *not* 0.

	Value (%)	
Metric	$\hat{f}_{(0)}(\texttt{train\_im})$	$\hat{f}_{(0)}(\texttt{test\_im})$
Error Rate	1.56	1.57
Sensitivity	96.74	95.47
False Alarm Rate	14.30	12.28
Specificity	98.60	98.72
Precision	87.12	88.16

Table II: Performance metrics for the Boolean 0 classifier operating on the training data, $\hat{f}_{(0)}(\texttt{train\_im})$. Equations for each metric can be found in a similar table in Appendix A. Performance metrics for other digits can be found in Appendix E.

4.2 Analyzing Results

By simply looking at the confusion matrices in Table III, we can see that our classifier performed relatively well. Specifically, we know this because the diagonal elements (representative of the images that were classified correctly) of both matrices are significantly larger than the off diagonal elements (representative of the images that were classified incorrectly). Still, we want to have a quantitative way to characterize our classifier's performance.

Although there are many different ways to measure accuracy for a multi-class classifier, we will use two of the more general types of metric. The first, *overall error rate*, denoted E_{oa}, represents the likelihood that our classifier makes a mistake. In other words this is the total number of errors, divided by the sample size. The other measurements that we will use to evaluate our classifier's accuracy are the *true label k rates*, denoted $L_{(k)}$. These metrics can be thought of as the fraction of images labeled as digit k that were correctly classified as digit k. The overall error rate and true label k rates our classifier operating on both image sets can be found in Table IV.

Our classifier had an overall error rate of 13.99% and a mean true label k rate of 85.79% when operating on the testing image set. Another aspect worth noting is that the success metrics of our classifier are relatively similar across the two data sets. The difference in overall error rate was 0.26%, while the maximum difference between corresponding true label k rates was 2.83%. Because our success metrics are relatively similar across the two image sets, we conclude that our classifier is not overfit[3]. If the data were overfit, we would expect the metrics associated with our testing images to be much worse than those for the training data (larger E_{oa} and smaller $L_{(k)}$).

[3]When a model has been trained on too much data, resulting in that model being highly sensitive. Akin to using a tenth degree polynomial to fit linear data.

	Prediction $\hat{f}_M(\texttt{train\_im})$										
Digit k	0	1	2	3	4	5	6	7	8	9	**Total**
0	5678	2	97	40	10	160	109	55	75	67	6293
1	7	6547	267	167	99	94	73	189	492	61	7996
2	20	40	4789	176	43	28	61	37	63	19	5276
3	15	15	149	5156	6	434	0	46	226	115	6162
4	25	19	107	32	5203	105	70	159	103	374	6197
5	43	31	11	128	50	3987	91	9	221	12	4583
6	64	14	235	56	39	191	5475	2	56	4	6136
7	5	14	92	116	23	40	0	5452	21	512	6275
8	60	54	192	134	60	239	36	11	4417	38	5241
9	6	6	19	126	309	143	3	305	177	4747	5841
All	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949	60000

(a) $\hat{f}_M(\texttt{train\_im})$

	Prediction $\hat{f}_M(\texttt{test\_im})$										
Digit k	0	1	2	3	4	5	6	7	8	9	**Total**
0	944	0	18	4	0	23	18	5	14	16	1042
1	0	1107	54	18	22	18	10	40	46	11	1326
2	1	2	812	23	6	3	10	16	11	2	886
3	2	2	25	878	1	72	0	6	30	17	1033
4	2	3	14	5	881	24	22	26	27	80	1084
5	7	1	0	17	5	659	17	0	39	1	746
6	14	5	41	9	10	24	874	1	15	1	994
7	2	1	23	25	2	14	0	885	13	75	1040
8	7	14	39	21	11	38	7	0	759	4	900
9	1	0	6	10	44	17	0	49	20	802	949
All	980	1135	1032	1010	982	892	958	1028	974	1009	10000

(b) $\hat{f}_M(\texttt{train\_im})$

Table III: Confusion matrices for the classification of training images (a) and testing images (b). All diagonal entries, which represent the number of images correctly classified for each digit, are blue, while the off diagonal entries are orange.

		Value (%)	
Metric		$\hat{f}_M(\texttt{train\_im})$	$\hat{f}_M(\texttt{test\_im})$
Overall Error Rate (E_{oa})		14.25	13.99
True Label Rates $(L_{(k)})$	0	95.86	96.33
	1	97.11	97.53
	2	80.38	78.68
	3	84.10	86.93
	4	89.06	89.71
	5	73.55	73.88
	6	92.51	91.23
	7	87.02	86.09
	8	75.49	77.93
	9	79.79	79.48

Table IV: Comparison of the accuracy metrics of our least squares classifier, $\hat{f}_M$, operating on both the training and testing data.

References

[1] United States Postal Service, "A decade of facts and figures," Postal Facts - U.S. Postal Service, 01-Apr-2021. [Online]. Available: https://facts.usps.com/table-facts/.

[2] Y. LeCun, C. Cortes, and C. Burges, "The MNIST Database," *MNIST handwritten digit database*. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[3] S. Boyd and L. Vandenberghe, "Chapters 13 and 14," in *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*, Cambridge: Cambridge University Press, 2018, pp. 245-278 and 285–304.

A Table Formulations

	Prediction		
Outcome	$\hat{f}_{(k)} = 1$	$\hat{f}_{(k)} =$ -1	**Total**
$y_{(k)} = \ 1$	N_{tp}	N_{fn}	N_{p^*}
$y_{(k)} =$ -1	N_{fp}	N_{tn}	N_{n^*}
All	$N_{\text{tp}} + N_{\text{fp}}$	$N_{\text{fn}} + N_{\text{tn}}$	N

Table V: General format of a confusion matrix for a Boolean k classifier, where $k \in \{0, 1, \ldots, 9\}$. In this example table, N_{ab} is the total number of images in a set characterized by subscripts a and b. The subscript a = t implies that the set contains images that our k classifier has correctly categorized (true classification), while a = f means that the set contains images which have been incorrectly classified (false classification). Similarly, b = p corresponds to a set containing images that were designated to be digit k ($\hat{f}_{j(k)}$ is positive), and b = n means that the set contains images that are marked as not the digit k ($\hat{f}_{j(k)}$ is negative). Additionally, N_{p^*} equals the number of images that were labeled as k, N_{n^*} equals the number of images that were *not* labeled as k, and N is the total number of images in the relevant `mnist` data set.

Metric	**Equation**
Error Rate	$\dfrac{N_{\text{fp}} + N_{\text{fn}}}{N}$
Sensitivity	$\dfrac{N_{\text{tp}}}{N_{\text{p}^*}}$
False Alarm Rate	$\dfrac{N_{\text{fp}}}{N_{\text{n}^*}}$
Specificity	$\dfrac{N_{\text{tp}}}{N_{\text{n}^*}}$
Precision	$\dfrac{N_{\text{tp}}}{N_{\text{tp}} + N_{\text{fp}}}$

Table VI: Equations that define each of the performance metrics used for Boolean classifiers. See Table V for the definition of each variable.

B Matlab Code

B.1 Training Script

```
%% Load training images
load('mnist.mat','images','labels');

%% Calculate x_star values for each digit

% Pre-process train images
train_im = pre_proc(images);
num_train = size(train_im,1);

% Create train_s_stack
%   train_s_stack is a column vector containing the pixel values of each image in the
%   training data set concatenated with a [1], which will correspond to the constant
%   shift of our hyperplane.
train_s_stack = [ train_im, ones(num_train,1) ];

% Initialize a bunch of variables for saving stuff
%   The corresponding digit k for an element in these variables is L-1, where L is the
%   value of the highest dimension index.
train_p_inds = zeros(num_train,10);
x_star = zeros(677,10);
train_f_tilde = zeros(num_train,10);
train_Bconf = zeros(3,3,10);

% Take the pseudo-inverse of the pre-processed s stack matrix
train_ps_stack = pinv(train_s_stack);

% Calculate theta_star and f_tilde values for each digit
for i = 0:9
    % Create and save logical array containing relevant label data
    %   True implies image at the corresponding index is labeled as the current digit.
    train_p_inds(:,i+1) = labels == i;

    % Create y
    %   Set all indices whose corresponding value in labels is not equal to our current
    %   digit to -2. All others (label = our current digit) will be set to 0. We then
    %   shift y so that our y matches the convention that 1 corresponds to indices
    %   labeled as our current digit, and -1 corresponds to indices that are not.
    y = -2*~train_p_inds(:,i+1) + 1;

    % Calculate x_star
    %   By multiplying our pseudo-inversed s stack by the known outputs we obtain the
    %   x_star for our problem. Note that this can be thought of as the A matrix when
    %   calculating f_tilde.
    x_star(:,i+1) = train_ps_stack * y;

    % Visualize the coefficients in theta_star for this digit
    if plotflag(2)
        coef_fig = vis_coef(x_star(1:end-1,i+1)',i);
        if saveflag(2)
            exportgraphics(coef_fig,sprintf('plots/train/coeffs_%i.eps',i));
        end
    end

    % Calculate f_tilde for each image
    train_f_tilde(:,i+1) = train_s_stack * x_star(:,i+1);

    % Create actual positive and actual negative subsets
    %   These will be used for plotting and the confusion matrix.
    train_p_vals = train_f_tilde(labels == i,i+1);
    train_n_vals = train_f_tilde(labels ~= i,i+1);
    %  Plot histogram for this digit
    if plotflag(3)
        hist_fig = bin_hist(train_p_vals,train_n_vals,i,num_train,8300);
        if saveflag(3)
            exportgraphics(hist_fig,sprintf('plots/train/hist_%i.eps',i));
        end
    end

    % Calculate confusion matrix for this sub-problem
    train_Bconf(:,:,i+1) = confmat(train_p_vals,train_n_vals);

    if train_Bconf(3,3,i+1) ~= num_train
        warning('Sum of elements in confusion matrix is not number of inputs.');
    end
end

%% Save properly formatted A and b that define each hyperplane for Shalom
A = x_star(1:end-1,:)';
b = x_star(end,:)';
save('HP_Haimes.mat','A','b');

%% Calculate performance metrics for each binary classification problem
[train_Ber,train_Btp,train_Bfp,train_Btn,train_Bpr] = Bconf2perf(train_Bconf);

%% Determine digit classification of training set for multi-class problem

% Find the maximum value of f_tilde for each image
%   Save its value and its index, the digit classification is one less than the index.
[train_val,train_dig] = max(train_f_tilde(:,1:10),[],2);

% Calculate confusion matrix
%   For each page (digit), we only look at images labeled as that digit. Then, for each
%   digit, we sum the number of times that our least squares classifier labeled it that
%   digit. The result is our confusion matrix, excluding the last row and column. We
%   then calculate these for our final confusion matrix.
train_temp = reshape(train_p_inds,[],1,10);
train_Mconf = reshape(sum(train_temp & train_dig(:,:) == 1:10),10,10);
train_Mconf = [ train_Mconf  sum(train_Mconf,2) ];
train_Mconf = [ train_Mconf; sum(train_Mconf)   ];

% Calculate performance metrics
train_tli = ( 1 ./ train_Mconf(11,1:10) ) .* sum(eye(10).*train_Mconf(1:10,1:10));
train_oer = 1 - ( 1 / train_Mconf(11,11) ) * sum(eye(10).*train_Mconf(1:10,1:10),'all');
```

B.2 Testing Script

```
%% Load external data
load('mnist.mat','images_test','labels_test');      load('HP_Haimes.mat','A','b');

%% Pre-process data and set up other variables

% Reformat digit hyperplanes
x_star = [ A b ]';

% Pre-process test images and create s matrix
test_im = pre_proc(images_test);                    test_num = size(test_im,1);
test_s_stack = [ test_im, ones(test_num,1) ];

% Initialize some storage variables
test_f_tilde = zeros(test_num,10);                  test_Bconf = zeros(3,3,10);
test_p_inds = false(test_num,10);

% Save logical arrays
test_p_inds(:,1:10) = labels_test == 0:9;
test_n_inds = ~test_p_inds;

%% Use least squares classification model on test data
for i = 1:10
    % Calculate f_tilde
    test_f_tilde(:,i) = test_s_stack * x_star(:,i);

    % Create actual positive and actual negative subsets
    %   These will be used for plotting and the confusion matrix.
    test_p_vals = test_f_tilde(test_p_inds(:,i),i);
    test_n_vals = test_f_tilde(test_n_inds(:,i),i);

    % Calculate confusion matrix for this sub-problem
    test_Bconf(:,:,i) = confmat(test_p_vals,test_n_vals);

    % Plot histogram for this digit
    %   This histogram is of the Boolean digit classifier operating on the testing data.
    if plotflag(4)
        hist_fig = bin_hist(test_p_vals,test_n_vals,i-1,test_num,1350);
        if saveflag(4)
            exportgraphics(hist_fig,sprintf('plots/test/testhist_%i.eps',i-1));
        end
    end
end

%% Calculate performance metrics for each binary classification problem
[test_Ber,test_Btp,test_Bfp,test_Btn,test_Bpr] = Bconf2perf(test_Bconf);

% Find the maximum value of f_tilde for each image
%   Save its value and its index, the digit classification is one less than the index.
[test_val,test_dig] = max(test_f_tilde(:,1:10),[],2);

% Calculate confusion matrix
%   For each page (digit), we only look at images labeled as that digit. Then, for each
%   digit, we sum the number of times that our least squares classifier labeled it that
%   digit. The result is our confusion matrix, excluding the last row and column. We
%   then calculate these for our final confusion matrix.
test_temp = reshape(test_p_inds,[],1,10);
test_Mconf = reshape(sum(test_temp(:,:,1:10) & test_dig(:,:) == 1:10),10,10);
test_Mconf = [ test_Mconf  sum(test_Mconf,2) ];
test_Mconf = [ test_Mconf; sum(test_Mconf)   ];

% Calculate performance metrics
test_tli = ( 1 ./ test_Mconf(11,1:10) ) .* sum(eye(10) .* test_Mconf(1:10,1:10));
test_oer = 1 - ( 1 / test_Mconf(11,11) ) * sum(eye(10) .* test_Mconf(1:10,1:10),'all');
```

B.3 Pre-processing function

```matlab
function imgs = pre_proc(imgs)
%PRE_PROC pre-processing images
%   Function to get input images into the desired form for the least squares machine
%   learning problem. This involves changing the data type to double, dividing by 255,
%   and removing the "border" pixels of each image.

% Check and save size of input
%   Input should have each image as a single row in a matrix. That matrix should be X by
%   Y, where X is the number of input images, and Y is the total number of pixels in an
%   image. Because the images are square, the square root of Y will be a natural number.
n = sqrt(size(imgs,2));
if n ~= floor(n)
    error('Input images should be a row representation of a square matrix.');
end
X = size(imgs,1);

% Create mask for input matrix
%   We want to remove all "border" pixels from this data. We have already determined the
%   indices of the elements to remove, so we create a mask in which ones will be
%   removed, and zeros will remain.
mask = zeros(1,n^2);                        % initialize
mask( 1:n ) = 1;                            % left
mask( (n+1):n:(n*(n-1)+1) ) = 1;            % top
mask( n:n:(n*n) ) = 1;                      % bottom
mask( (n*(n-1)+1):(n^2) ) = 1;              % right
mask = repmat(mask,size(imgs,1),1);         % stack rows X times

% Convert image values to double and normalize
%   Because the input has values from [0,255], we divide the whole thing by 255 in order
%   to obtain only values between [0,1]. We also transpose the input, so that each
%   column is a different image, because of the way linear indices are defined.
imgs = double(imgs)' / 255;

% Use the mask and reshape output
%   Logical indexing allows us to remove all undesired indices from the input matrix in
%   one operation, but the output is a one dimensional array. Using the reshape function
%   and knowledge of the output geometry we reconstruct our masked images. Finally, we
%   take the transpose again, so that each row is a different image.
imgs = reshape(imgs(~mask'),(n-2)^2,X)';

end
```

B.4 Coefficient visualization function

```
function fig = vis_coef(imgs,labs)
%VIS_COEF visualize coefficients of each pixel
%   Function to visualize images for the least squares machine learning problem.

% Check and save size of input
%   Input should have each image as a single row in a matrix. That matrix should be X by
%   Y, where X is the number of input images, and Y is the total number of pixels in an
%   image. Because the images are square, the square root of Y will be a natural number.
n = sqrt(size(imgs,2));
if n ~= floor(n)
    error('Input images should be a row representation of a square matrix.');
end
X = size(imgs,1);

% Rearrange input images into desired format for plotting
%   First transpose the input so that each column represents a unique image. After that,
%   reshape each image into a square matrix. Images then stack in the 3rd dimension.
imgs = reshape(imgs',n,n,X);

% Save digits in string form if input
%   This will be used to easily name/label the figures/plots
if exist('labs','var')
    if numel(labs) == X
        labs = string(labs);
    else
        warning('Number of labels does not match number of images, labels ignored.');
        labs = repmat("No Label",X,1);
    end
else
    labs = repmat("No Label",X,1);
end

% Plot each coefficient matrix as an image
for i = 1:X
    % Pad each image with zeros
    %   These are the zeros that were removed prior to our least squares classification.
    %   This will make all images displayed a uniform 28 by 28 pixels.
    temp = [zeros(28,1) [ zeros(1,26); imgs(:,:,i); zeros(1,26)] zeros(28,1)];

    % Set upper and lower bounds of colorbar
    %   This makes the output visually understandable to a human. Without this step,
    %   most of the coefficients are indistinguishable from each other. The values of .1
    %   and -.1 were chosen after a process of trial and error, and were informed by
    %   figure 14.3 of Introduction to Applied Linear Algebra by Boyd and Vandenberghe.
    temp(temp < -.1) = -.1;                    temp(temp > .1) = .1;

    % Create the figure
    fig = figure('Name',sprintf('Visualized Coefficients - %s',labs(i)));
        set(fig,'Position',[2,200,500,475])
        ax = axes('Parent',fig,'Color','none','FontName','Monospaced','FontWeight',...
            'Bold','FontSize',11,'Visible','off','Position',[.09,.09,.82,.82]);
        hold(ax,'on');                         coef = imagesc(temp);

        % Set colormap
        colormap(ax,brewermap([],'*RdYlBu'));
        view(ax,[90 90]);                      axis(ax,'equal');
        annotation(fig,'textbox',[.1285 .8315 .155 .055],'String',labs(i),...
            'EdgeColor','none','FontWeight','Bold','FontSize',22,'FontName',...
            'Monospaced','VerticalAlignment','middle','HorizontalAlignment','left');
        hold(ax,'off');
end

end
```

B.5 Histogram plotting function

```matlab
function curr_fig = bin_hist(p_vals,n_vals,i,n,m)
%BIN_HIST create categorized histogram
%   Function to plot a categorized histogram for the results of a binary classifier

curr_fig = figure('Name',sprintf('Binary Classification Results - %i',i));
set(curr_fig,'Position',[2,200,560,510])
curr_ax = axes('Parent',curr_fig,'Color',[.99 .99 .99],'FontName','Monospaced',...
    'FontWeight','Bold','FontSize',10.5,'LineWidth',1);
box(curr_ax,'on');                    hold(curr_ax,'on');

p_hist = histogram(curr_ax,p_vals,'EdgeColor','#0072BD','EdgeAlpha',.75,'BinWidth',.06);
n_hist = histogram(curr_ax,n_vals,'EdgeColor','#D95319','EdgeAlpha',.75,'BinWidth',.06);
zeroline = plot(curr_ax,[0 0],[0,m],'--','Color',[.1 .1 .1],'HandleVisibility','off');

uistack(zeroline,'bottom');
xlim([-2,1.4]);                       ylim([0,m]);
curr_ax.YTickLabel = arrayfun(@(S) sprintf('%.2f',S),...
    100*round(curr_ax.YTick / n,4),'UniformOutput',0);
N_xtl = numel(curr_ax.XTickLabels);

for j=1:N_xtl
    if contains(curr_ax.XTickLabels{j},'.')
        curr_ax.XTickLabels{j} = '';
    end
end

hold(curr_ax,'off');
curr_leg = legend(curr_ax,{sprintf('Label  = %i',i),sprintf('Label != %i',i)},...
    'Location','north east','FontSize',12,'interpreter','tex','box','off');

end
```

B.6 Binary confusion matrix function

```matlab
function conf = confmat(p_vals,n_vals)
%CONFMAT Create a confusion matrix for a Binary classifier

% Initialize
conf = zeros(3,3);

% Populate matrix
conf(1,1) = sum(p_vals >= 0);             % True positive
conf(2,1) = sum(p_vals  < 0);             % False positive
conf(1,2) = sum(n_vals >= 0);             % False negative
conf(2,2) = sum(n_vals  < 0);             % True negative

% Add along both dimensions to get sums in third row/column
conf(1:2,3) = sum(conf(1:2,:),2);
conf(3,:) = sum(conf(:,:));

end
```

B.7 Binary accuracy metrics function

```
function [er,tp,fp,tn,pr] = Bconf2perf(conf)
%BCONF2PERF - Get performance metrics from the confusion matrix
%   Output an array of the performance metrics for any number of binary classifiers
%   operating on some data. Input should be a 3x3xN matrix, where N is the number of
%   confusion matrices input.

% Save the number of confusion matrices input
N = size(conf,3);

% Initialize variables as N by 1 vectors
er = zeros(N,1);
tp = zeros(N,1);
fp = zeros(N,1);
tn = zeros(N,1);
pr = zeros(N,1);

% Calculate performance metrics for each binary classification problem
er(:) = (conf(2,1,:) + conf(1,2,:)) ./ conf(3,3,:);    % Error rate
tp(:) = conf(1,1,:) ./ conf(1,3,:);                    % Sensitivity
fp(:) = conf(2,1,:) ./ conf(1,3,:);                    % False alarm rate
tn(:) = conf(2,2,:) ./ conf(2,3,:);                    % Specificity
pr(:) = conf(1,1,:) ./ conf(3,1,:);                    % Precision

end
```

B.8 Handwritten digit visualization function

```
function vis_dig(imgs,labs)
%VIS_DIG visualize digit images
%   Function to visualize images for the least squares machine learning problem.

% Check and save size of input
%   Input should have each image as a single row in a matrix. That matrix should be X by
%   Y, where X is the number of input images, and Y is the total number of pixels in an
%   image. Because the images are square, the sqaure root of Y will be a natural number.
n = sqrt(size(imgs,2));
if n ~= floor(n)
    error('Input images should be a row representation of a square matrix.');
end
X = size(imgs,1);

% Rearrange input images into desired format for plotting
%   First transpose the input so that each column represents a unique image. After that,
%   reshape each image into a square matrix. Images then stack in the 3rd dimension.
imgs = reshape(imgs',n,n,X);

% Save digits in string form if input
%   This will be used to easily name/label the figures/plots
if exist('labs','var')
    if numel(labs) == X
        labs = string(labs);
    else
        warning('Number of labels does not match number of images, labels ignored.');
        labs = repmat("No Label",X,1);
    end
else
    labs = repmat("No Label",X,1);
end

% Plot each input digit
for i = 1:X
    % Pad each image with zeros
    %   These are the zeros that were removed prior to our least squares classification.
    %   This will make all images displayed a uniform 28 by 28 pixels.
    temp = [zeros(28,1) [ zeros(1,26); imgs(:,:,i); zeros(1,26)] zeros(28,1)];

    % Create the figure
    fig = figure('Name',sprintf('Handwritten Digit - %s',labs(i)));
        set(fig,'Position',[2,200,500,475])
        ax = axes('Parent',fig,'Color','none','FontName','Monospaced','FontWeight',...
            'Bold','FontSize',11,'Visible','off','Position',[.09,.09,.82,.82]);
        hold(ax,'on');
        dig = imagesc(temp);                  colormap(flipud(gray(256)));
        view(ax,[90 90]);                     axis(ax,'equal');
        annotation(fig,'textbox',[.1275 .8325 .155 .055],...
            'String',labs(i),'EdgeColor','none','FontWeight','Bold','FontSize',20,...
            'FontName','Monospaced','VerticalAlignment','middle',...
            'HorizontalAlignment','left');
        hold(ax,'off');
end

end
```

C Histograms

C.1 Training

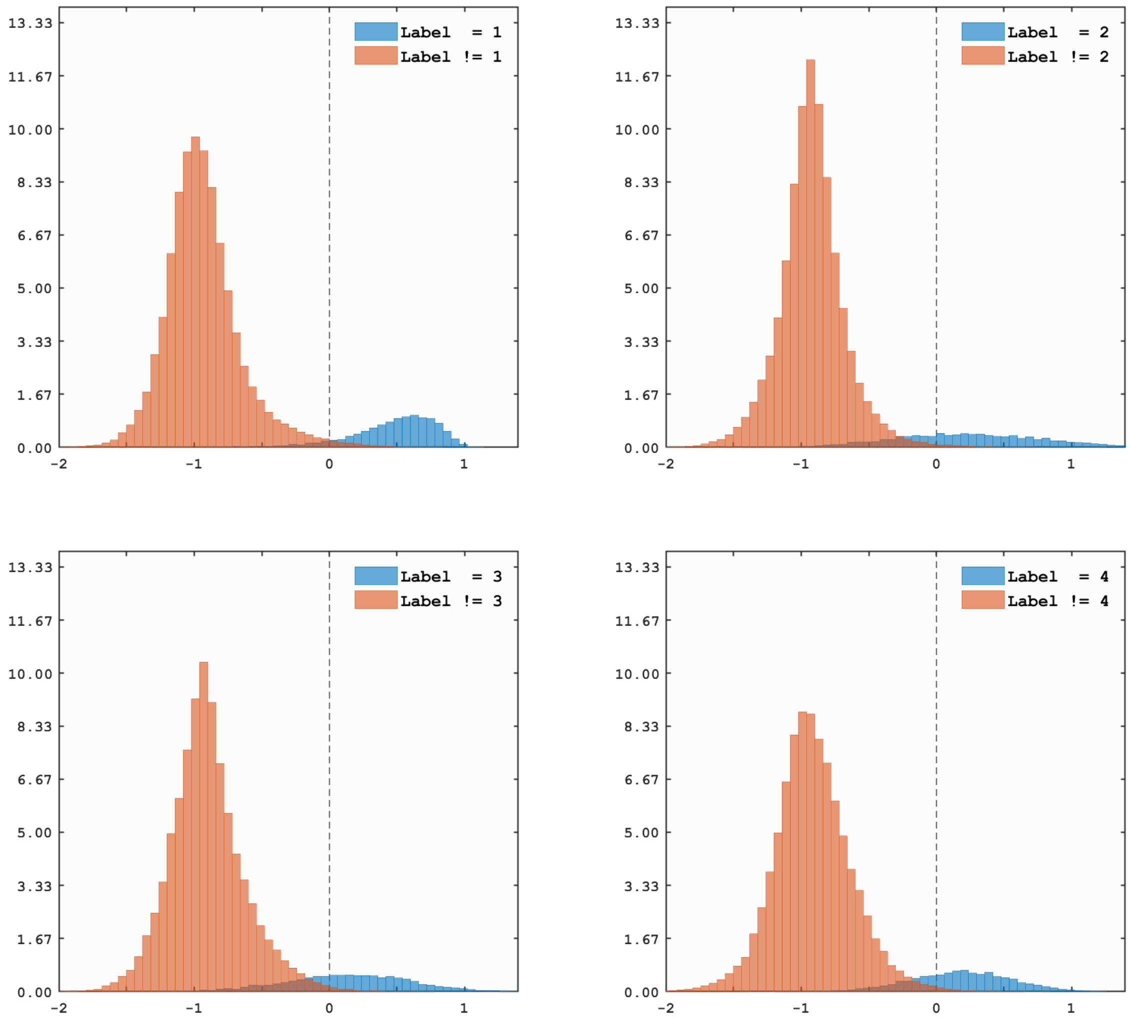


Figure 5: Categorized histogram of the results of the Boolean k classifiers ($k \in \{1, 2, \ldots, 9\}\}$) operating on the training images, $\hat{f}_{(k)}(\texttt{train\_im})$. The value of k for each graph can be seen in the legend in the northeast of each plot. The count of each *blue* bin is equivalent to the percent of images in the training data set that are labeled k and have an $\tilde{f}_{j(k)}$ value within its bounds. Similarly, the *orange* bin count is the percent of images in the training data set that are *not* labeled k and have an $\tilde{f}_{j(k)}$ value within its bounds. All images that fall on the positive side of the $\tilde{f}_{(k)} = 0$ dotted line are categorized as k, while all images on the negative side are categorized as *not* k. All histograms in this figure use the same bin width of 0.06.

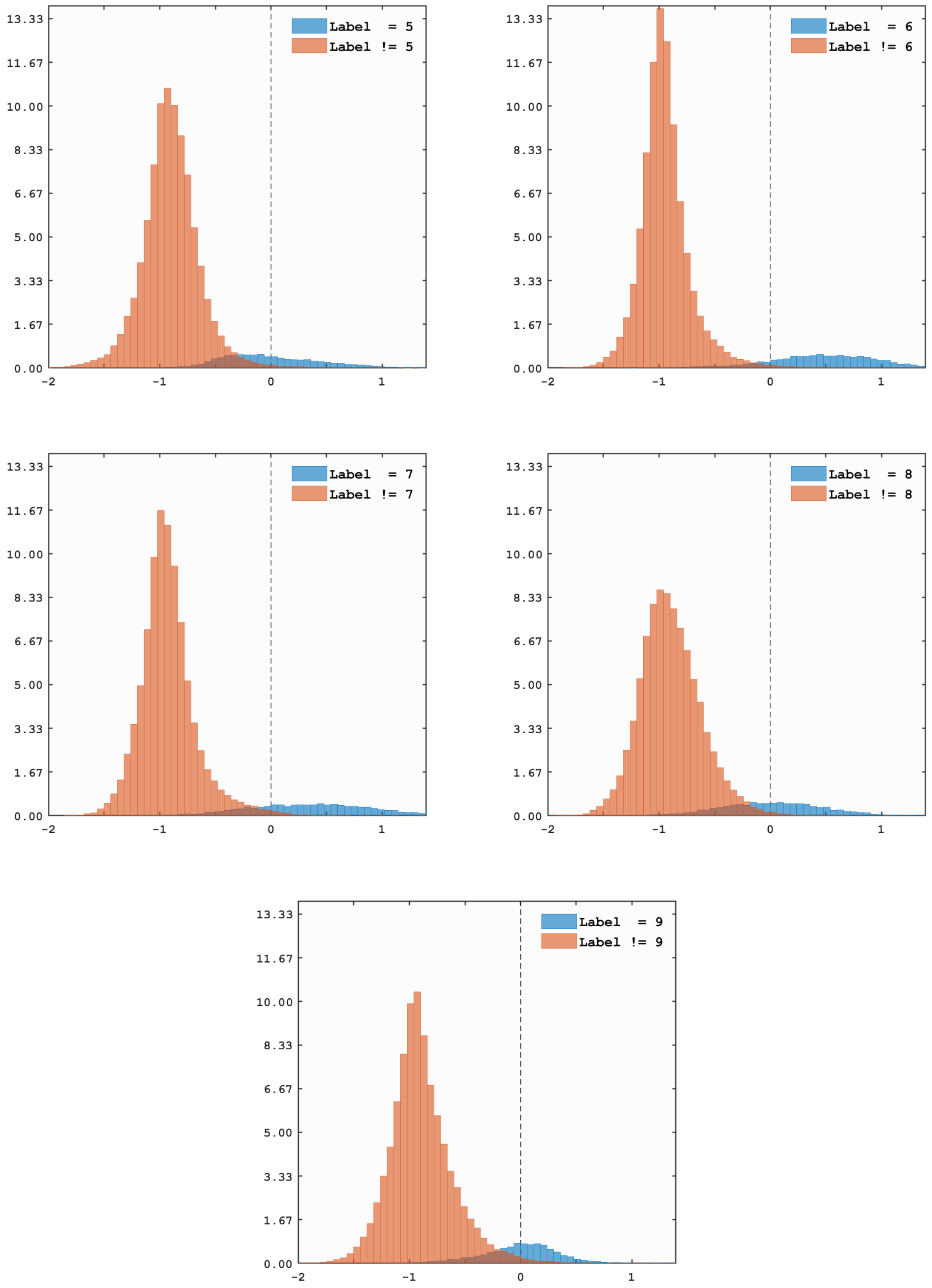


Figure 5 (cont.)

C.2 Testing

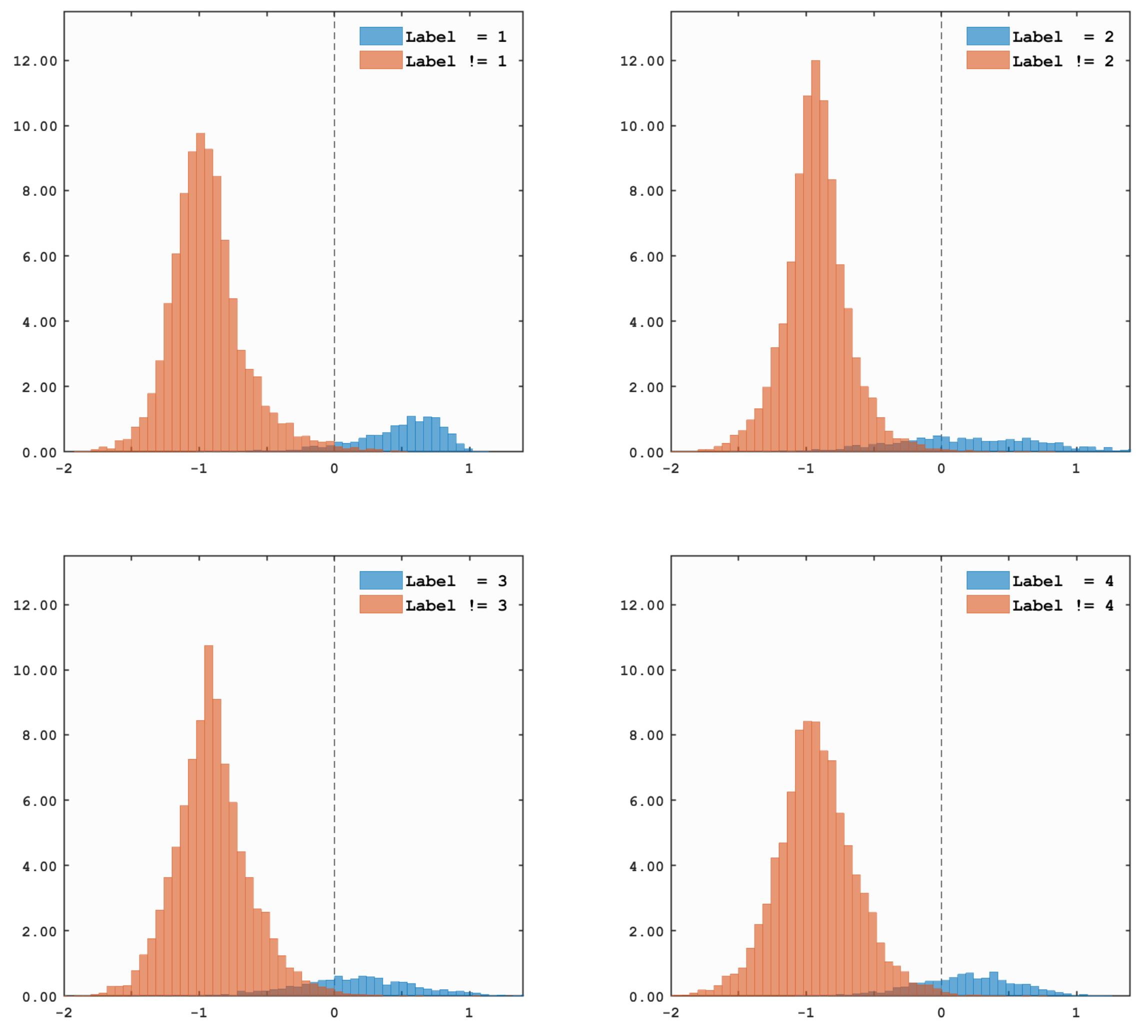


Figure 6: Categorized histogram of the results of the Boolean k classifiers ($k \in \{1,\ 2, \ldots,\ 9\}\}$) operating on the testing images, $\hat{f}_{(k)}(\texttt{test\_im})$. The format of each histogram in this figure is almost identical to the format of the histograms in Fig. (5), the one difference being that the maximum percent bound has been reduced by a small amount.

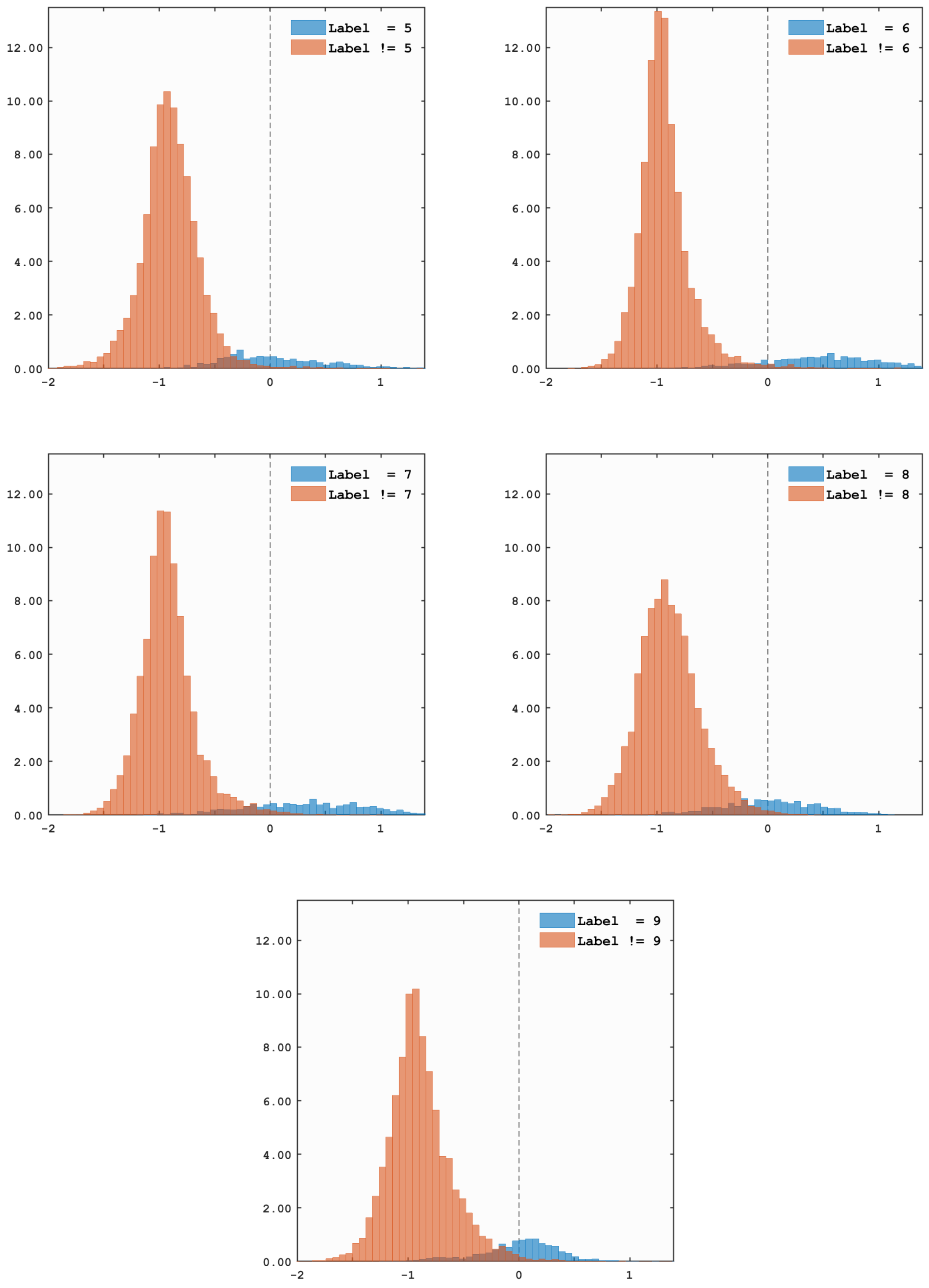


Figure 6 (cont.)

D Confusion Matrices

	Prediction		
Outcome	$\hat{f}_{(1)} = 1$	$\hat{f}_{(1)} = \text{-}1$	**Total**
$y_{(1)} = 1$	6133	532	6665
$y_{(1)} = \text{-}1$	609	52726	53335
All	6742	53258	60000

(a) $\hat{f}_{(1)}$(`train_im`)

	Prediction		
Outcome	$\hat{f}_{(1)} = 1$	$\hat{f}_{(1)} = \text{-}1$	**Total**
$y_{(1)} = 1$	1035	67	1102
$y_{(1)} = \text{-}1$	100	8798	8898
All	1135	8865	10000

(b) $\hat{f}_{(1)}$(`test_im`)

Table VII: Confusion matrices for the Boolean 1 classifier operating on the training (a) and testing (b) data.

	Prediction		
Outcome	$\hat{f}_{(2)} = 1$	$\hat{f}_{(2)} = \text{-}1$	**Total**
$y_{(2)} = 1$	3927	236	4163
$y_{(2)} = \text{-}1$	2031	53806	55837
All	5958	54042	60000

(a) $\hat{f}_{(2)}$(`train_im`)

	Prediction		
Outcome	$\hat{f}_{(2)} = 1$	$\hat{f}_{(2)} = \text{-}1$	**Total**
$y_{(2)} = 1$	644	28	672
$y_{(2)} = \text{-}1$	388	8940	9328
All	1032	8968	10000

(b) $\hat{f}_{(2)}$(`test_im`)

Table VIII: Confusion matrices for the Boolean 2 classifier operating on the training (a) and testing (b) data.

	Prediction		
Outcome	$\hat{f}_{(3)} = 1$	$\hat{f}_{(3)} = \text{-}1$	**Total**
$y_{(3)} = 1$	3837	242	4079
$y_{(3)} = \text{-}1$	2294	53627	55921
All	6131	53869	60000

(a) $\hat{f}_{(3)}$(`train_im`)

	Prediction		
Outcome	$\hat{f}_{(3)} = 1$	$\hat{f}_{(3)} = \text{-}1$	**Total**
$y_{(3)} = 1$	655	43	698
$y_{(3)} = \text{-}1$	355	8947	9302
All	1010	8990	10000

(b) $\hat{f}_{(3)}$(`test_im`)

Table IX: Confusion matrices for the Boolean 3 classifier operating on the training (a) and testing (b) data.

	Prediction		
Outcome	$\hat{f}_{(4)} = 1$	$\hat{f}_{(4)} = \text{-}1$	**Total**
$y_{(4)} = \ \ 1$	4033	219	4252
$y_{(4)} = \text{-}1$	1809	53939	55748
All	5842	54158	60000

(a) $\hat{f}_{(4)}(\texttt{train\_im})$

	Prediction		
Outcome	$\hat{f}_{(4)} = 1$	$\hat{f}_{(4)} = \text{-}1$	**Total**
$y_{(4)} = \ \ 1$	685	36	721
$y_{(4)} = \text{-}1$	297	8982	9279
All	982	9018	10000

(b) $\hat{f}_{(4)}(\texttt{test\_im})$

Table X: Confusion matrices for the Boolean 4 classifier operating on the training (a) and testing (b) data.

	Prediction		
Outcome	$\hat{f}_{(5)} = 1$	$\hat{f}_{(5)} = \text{-}1$	**Total**
$y_{(5)} = \ \ 1$	2387	303	2690
$y_{(5)} = \text{-}1$	3034	54276	57310
All	5421	54579	60000

(a) $\hat{f}_{(5)}(\texttt{train\_im})$

	Prediction		
Outcome	$\hat{f}_{(5)} = 1$	$\hat{f}_{(5)} = \text{-}1$	**Total**
$y_{(5)} = \ \ 1$	416	55	471
$y_{(5)} = \text{-}1$	476	9053	9529
All	892	9108	10000

(b) $\hat{f}_{(5)}(\texttt{test\_im})$

Table XI: Confusion matrices for the Boolean 5 classifier operating on the training (a) and testing (b) data.

	Prediction		
Outcome	$\hat{f}_{(6)} = 1$	$\hat{f}_{(6)} = \text{-}1$	**Total**
$y_{(6)} = \ \ 1$	4959	345	5304
$y_{(6)} = \text{-}1$	959	53737	54696
All	5918	54082	60000

(a) $\hat{f}_{(6)}(\texttt{train\_im})$

	Prediction		
Outcome	$\hat{f}_{(6)} = 1$	$\hat{f}_{(6)} = \text{-}1$	**Total**
$y_{(6)} = \ \ 1$	772	74	846
$y_{(6)} = \text{-}1$	186	8968	9154
All	958	9042	10000

(b) $\hat{f}_{(6)}(\texttt{test\_im})$

Table XII: Confusion matrices for the Boolean 6 classifier operating on the training (a) and testing (b) data.

	Prediction		
Outcome	$\hat{f}_{(7)} = 1$	$\hat{f}_{(7)} = \text{-}1$	**Total**
$y_{(7)} = \ \ 1$	4558	323	4881
$y_{(7)} = \text{-}1$	1707	53412	55119
All	6265	53735	60000

(a) $\hat{f}_{(7)}(\texttt{train\_im})$

	Prediction		
Outcome	$\hat{f}_{(7)} = 1$	$\hat{f}_{(7)} = \text{-}1$	**Total**
$y_{(7)} = \ \ 1$	734	62	796
$y_{(7)} = \text{-}1$	294	8910	9204
All	1028	8972	10000

(b) $\hat{f}_{(7)}(\texttt{test\_im})$

Table XIII: Confusion matrices for the Boolean 7 classifier operating on the training (a) and testing (b) data.

	Prediction		
Outcome	$\hat{f}_{(8)} = 1$	$\hat{f}_{(8)} =$ -1	**Total**
$y_{(8)} =$ 1	2994	220	3214
$y_{(8)} =$ -1	2857	53929	56786
All	5851	54149	60000

(a) $\hat{f}_{(8)}$(`train_im`)

	Prediction		
Outcome	$\hat{f}_{(8)} = 1$	$\hat{f}_{(8)} =$ -1	**Total**
$y_{(8)} =$ 1	502	42	544
$y_{(8)} =$ -1	472	8984	9456
All	974	9026	10000

(b) $\hat{f}_{(8)}$(`test_im`)

Table XIV: Confusion matrices for the Boolean 8 classifier operating on the training (a) and testing (b) data.

	Prediction		
Outcome	$\hat{f}_{(9)} = 1$	$\hat{f}_{(9)} =$ -1	**Total**
$y_{(9)} =$ 1	3088	484	3572
$y_{(9)} =$ -1	2861	53567	56428
All	5949	54051	60000

(a) $\hat{f}_{(9)}$(`train_im`)

	Prediction		
Outcome	$\hat{f}_{(9)} = 1$	$\hat{f}_{(9)} =$ -1	**Total**
$y_{(9)} =$ 1	562	67	629
$y_{(9)} =$ -1	447	8924	9371
All	1009	8991	10000

(b) $\hat{f}_{(9)}$(`test_im`)

Table XV: Confusion matrices for the Boolean 9 classifier operating on the training (a) and testing (b) data.

E Performance Metrics

	Value (%)									
Metric	0	1	2	3	4	5	6	7	8	9
Error Rate	1.56	1.90	3.78	4.23	3.38	5.56	2.17	3.38	5.13	5.58
Sensitivity	96.74	92.02	94.33	94.07	94.85	88.74	93.50	93.38	93.15	86.45
False Alarm Rate	14.30	9.14	48.79	56.24	42.54	112.79	18.08	34.97	88.89	80.10
Specificity	98.60	98.86	96.36	95.90	96.76	94.71	98.25	96.90	94.97	94.93
Precision	87.12	90.97	65.91	62.58	69.03	44.03	83.80	72.75	51.17	51.91

(a) $\hat{f}_{(k)}(\texttt{train\_im})$

	Value (%)									
Metric	0	1	2	3	4	5	6	7	8	9
Error Rate	1.57	1.67	4.16	3.98	3.33	5.31	2.60	3.56	5.14	5.14
Sensitivity	95.47	93.92	95.83	93.84	95.01	88.32	91.25	92.21	92.28	89.35
False Alarm Rate	12.82	9.07	57.74	50.86	41.19	101.06	21.99	36.93	86.76	71.07
Specificity	98.72	98.88	95.84	96.18	96.80	95.00	97.97	96.81	95.01	95.23
Precision	88.16	91.19	62.40	64.85	69.76	46.64	80.58	71.40	51.54	55.70

(b) $\hat{f}_{(k)}(\texttt{test\_im})$

Table XVI: Performance metrics for each Boolean digit classifiers operating on training (a) and testing (b) data.