CSE 847 Home Assignment 4

Submitted by: Ritam Guha (MSU ID: guharita)

Date: March 31, 2021

1 Logistic Regression: Experiment

As a part of this experiment, we are supposed to build a classifier based on logistic regression. The experimentation is performed on ***Spam Email Detection*** dataset. The dataset is publicly available at: Spam Email Dataset. The dataset consists of 4601 samples classified as 1 (Spam) and 0 (non-Spam).

First, the labels were converted to $+1/-1$ scheme from the existing $+1/0$ scheme. Multiple training sizes were used to track the classification accuracies for varying training load. The test data was always fixed and consisted of the last 2601 entries of the data. On the other hand the training size was increased as: $n = 200, 500, 800, 1000, 1500, 2000$.

1.1 Code

The code used for the experimentation is provided below:

```
data = importdata('Data/Spam Email Detection/data.xlsx');    % import data
labels = importdata('Data/Spam Email Detection/labels.xlsx');    % import labels
labels(labels==0) = -1; % transforming 0/1 labels to -1/+1data = importdata('
    Data/Alzheimers/ad_data.mat');
data(:, size(data,2)+1) = ones(1, size(data,1))';


global cur_train_size;

% initializing traing and test data
train_size = [200, 500, 800, 1000, 1500, 2000];
test_size = 2601;
[num_samples, num_features] = size(data);
test_data = data(num_samples - test_size + 1 : num_samples, :);
test_labels = labels(num_samples - test_size + 1 : num_samples);
accuracy = zeros(1, size(train_size, 2));

for i = 1: size(train_size, 2)
    cur_train_size = train_size(i);

    train_data = data(1 : cur_train_size, :);
    train_labels = labels(1 : cur_train_size);
```

```
22
23      weights = logRegression(train_data, train_labels, 1e-10, 100);
24      accuracy(1, i) = compute_accuracy(test_data, test_labels, weights);
25      fprintf('Training Size = %d: %f\n', cur_train_size, accuracy(1,i));
26  end
27
28  % Plot the variation of accuracy with training size
29  figure;
30  X = train_size;
31  Y = accuracy;
32  fig = plot(X, Y);
33  xlabel('Training Size');
34  ylabel('Classification Accuracy (in %)');
35  title('Variation of Classification Accuracy with Training Size')
36  saveas(fig, strcat('Results/Logistic Train/Accuracy_Variance.jpg'));
37
38
39  function [accuracy] = compute_accuracy(data, labels, weights)
40      global cur_train_size
41
42      % getting the predicted labels and computing accuracy
43      predicted_labels = sigmoid(data * weights);
44      predicted_labels(predicted_labels > 0.5) = 1;
45      predicted_labels(predicted_labels <= 0.5) = -1;
46      correct_predictions = sum(predicted_labels == labels);
47      accuracy = (correct_predictions/size(data,1) * 100);
48
49      % create the confusion matrix
50      fig = confusionchart(labels, predicted_labels);
51      title(strcat('Confusion Chart for Training Size: ', int2str(cur_train_size))
            );
52      saveas(fig, strcat('Results/Logistic Train/Conf_Chart_Train_Size_', int2str(
            cur_train_size), '.jpg'));
53  end
54
55  function [weights] = logRegression(data, labels, epsilon, maxiter)
56
57      % setting the default parameter values
58      if nargin < 4
59          if ~exist('epsilon')
60              epsilon = 1e-6;
61          end
62          if ~exist('maxiter')
63              maxiter=1000;
64          end
65      end
66  %
```

```
% code to train a logistic regression classifier
%
% INPUTS:
%   data    = n * (d+1) matrix withn samples and d features, where
%             column d+1 is all ones (corresponding to the intercept term)
%   labels  = n * 1 vector of class labels (taking values 0 or 1)
%   epsilon = optional argument specifying the convergence
%             criterion - if the change in the absolute difference in
%             predictions, from one iteration to the next, averaged across
%             input features, is less than epsilon, then halt
%             (if unspecified, use a default value of 1e-5)
%   maxiter = optional argument that specifies the maximum number of
%               iterations tp execute (default=1000)
%   train_size = number of samples used for training
%   test_size = number of samples used for testing
%
% OUTPUT:
%    weights = (d+1) * 1 vector of weights where the weights correspond to
%              the columns of "data"


    % initializing parameters
    num_features = size(data, 2);
    w = zeros(num_features, 1);
    iter = 1;
    eta = 0.00001;
    prev_error = Inf;
    cur_error = -Inf;

    % loop running till convergence
    while(iter <= maxiter && (abs(cur_error - prev_error) >= epsilon))
        % compute train error
        z = -labels .* (data * w);
        error(1, iter) = mean(log(1 + exp(z)));

        prev_error = cur_error;
        cur_error = error(1, iter);

        % use the gradient of the loss function wrt the training data to
        % update the weight
        dw = (mean(-exp(-z)./(1+exp(-z)) .* (data .* labels)))';
        w = w - (eta * dw);
        iter = iter+1;
    end

    iter = iter-1;
    weights = w;
```

```
114
115     % plot the train error over the iterations
116     figure;
117     hold on;
118     x = linspace(1,iter,iter);
119     plot(x, error);
120     legend('Train Error');
121     hold off;
122 end
123
124 function[val] = sigmoid(input)
125     % sigmoid function implementation
126     val = 1./(1 + exp(-input));
127 end
```

1.2 Experimental Outcome

From the experimentation, it was observed that the classification accuracy varied over the increasing number of training samples as shown in Figure 1. From the Figure, it is visible that at first, increasing the number of samples in the training schedule was improving the perfromance. But when the training size exceeded 1000, the classification accuracy started dropping beyond that point. The best classification accuracy obtained by this classifier was 91.080354 for training size of 1000.

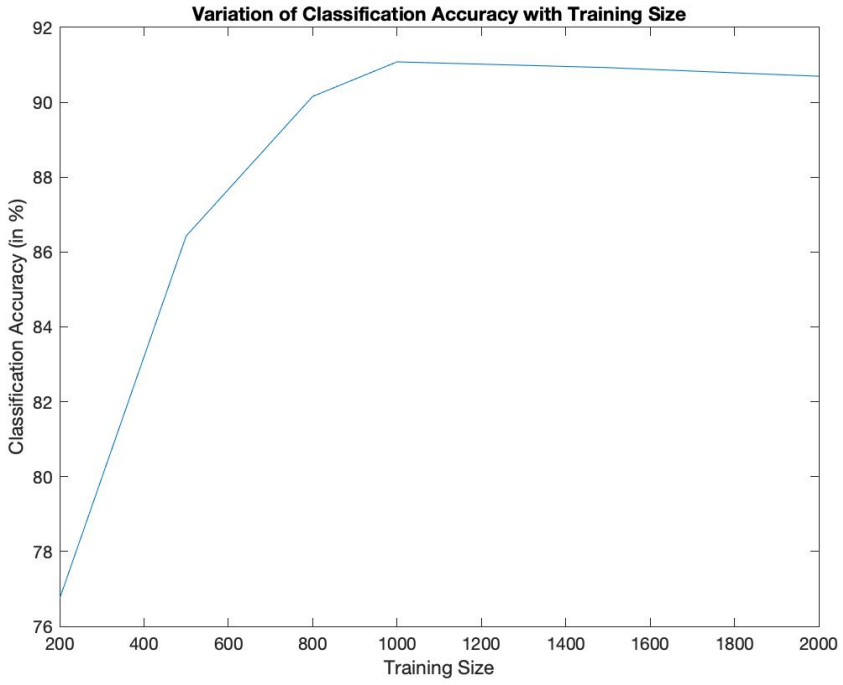

Figure 1: Variation of Classification Accuracy with the Number of Samples available for Training

In addition to this, the confusion matrices for these experimentation were also plotted. The matrices are

provided in Figure 2. False postives are really crucial for spam email detection services. If an important mail gets marked as spam, it may be fatal for the users. So, classifcationa accuracy cannot be always used as a evaluation metric. The reason is that classification accuracy works on the total false cases and does not differentiate between false positives and false negatives. But, in some cases, false positives may be worse than false negatives. In case of spam email detection, that is the case. So, we should always check the confusion matrices to arrive at the final model based on the business aspect of the service. For this reason, the confusion matrices for the given experiments were checked and it was observed that $n = 1000$ gives the least number of false positives as well. So, it can be considered as the best model provided by the experimentation.

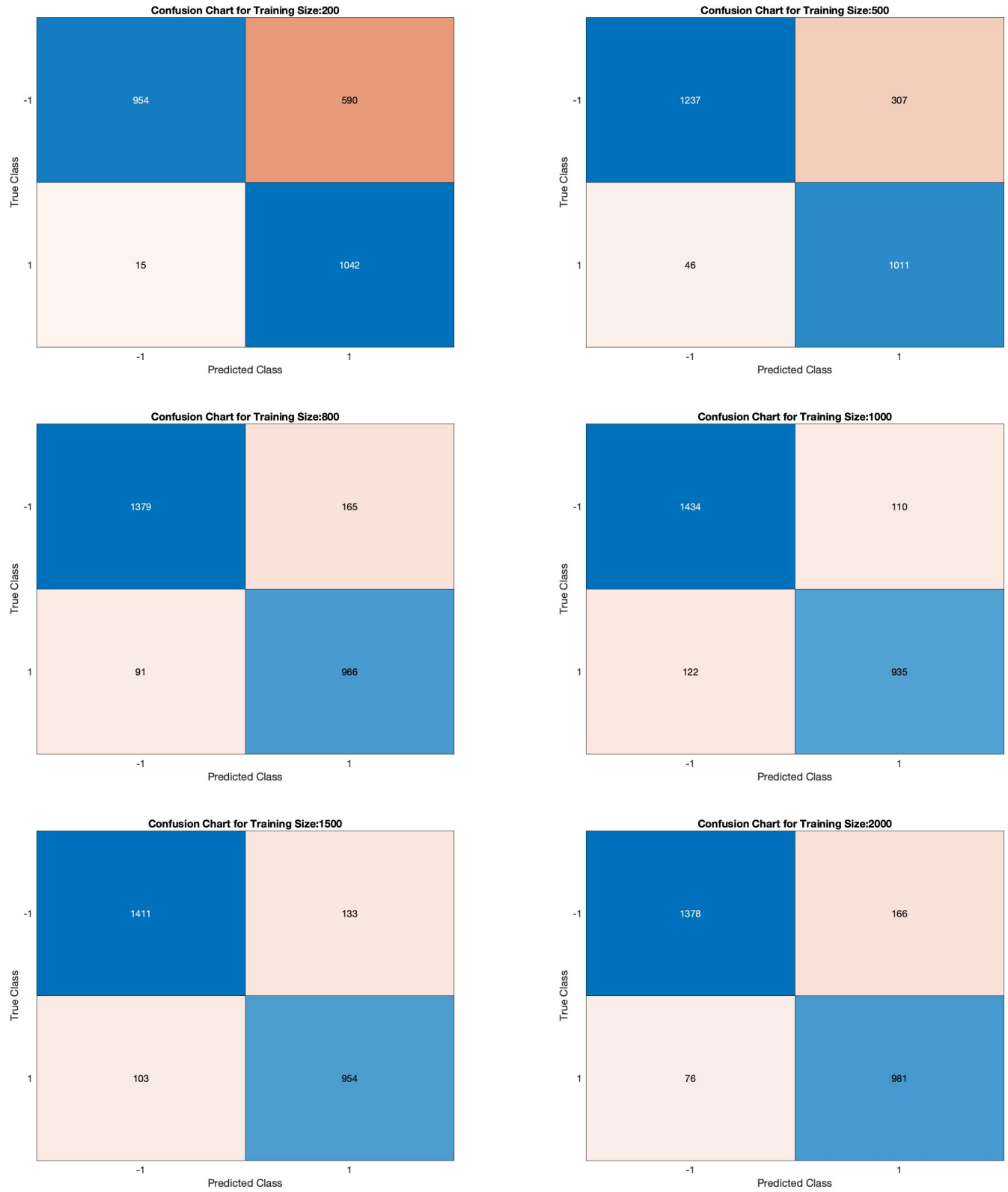

Figure 2: Plot of Confusion Matrices obtained for different training sizes

2 Sparse Logistic Regression: Experiment

The goal of this experiment is to add sparse regularization to the logistic regression framework and observe how it helps to reduce the number of features used for the classification (feature selection). For the simulation of the experimentation, ***Alzheimers*** dataset is used as an application which is publicly available at: Alzheimers Dataset.

The dataset is pre-divided into training and test samples. There are 172 training samples and 74 test samples in the dataset. Each sample has an associated label of either $+1$ (Alzheimer's Disease patient) or -1 (Mild Cognitive Impairment patient).

2.1 Code

The code used to implement the sparse logistic regression is presented below:

```
% import train and test data
data = importdata('Data/Alzheimers/ad_data.mat');
train_data = data.X_train;
train_labels = data.y_train;
test_data = data.X_test;
test_labels = data.y_test;
features = importdata('Data/Alzheimers/feature_name.mat');

% possible values for the regularization parameter
par = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1];

global cur_par

% perform sparse regularized logistic regression for all possible
% parameters
for i=1:size(par, 2)
    cur_par = par(i);
    [w, c] = logL1Regression(train_data, train_labels, cur_par);
    scores = sigmoid(test_data * w + c);
    [~, ~, ~, AUC(1,i)] = perfcurve(test_labels, scores, 1); % get the AUC
    accuracy(1, i) = compute_accuracy(test_data, test_labels, w, c);
    num_nz_weights(1, i) = nnz(w);
%     fprintf('Par= %d: Non-zero weights: %d, Accuracy: %f, AUC: %f\n', cur_par,
    num_nz_weights(1, i), accuracy(1, i), AUC(1,i));
    fprintf('%d\t %f\t %f\n', num_nz_weights(1, i), accuracy(1, i), AUC(1,i));
end

% Plot the variation of accuracy, AUC and No. of non-zero weights with training
   size
figure;
hold on;
X = par;
Y1 = normalize(accuracy, 'norm');
Y2 = normalize(AUC, 'norm');
```

```
Y3 = normalize(num_nz_weights, 'norm');
fig = plot(X, Y1);
plot(X, Y2)
plot(X, Y3);
legend('Accuracy', 'AUC', 'No. of Non-zero Weights')
xlabel('Regularization Parameter');
ylabel('Normalized Metric Scores');
title({'Variation in Classification Accuracy, AUC and Number of non-zero weights', 'with Regularization Parameter'})
saveas(fig, strcat('Results/L1 Logistic Train/Metric_Variance.jpg'));
hold off


function [accuracy] = compute_accuracy(data, labels, weights, c)
    global cur_par

    % getting the predicted labels and computing accuracy
    predicted_labels = sigmoid(data * weights + c);
    predicted_labels(predicted_labels > 0.5) = 1;
    predicted_labels(predicted_labels <= 0.5) = -1;
    correct_predictions = sum(predicted_labels == labels);
    accuracy = (correct_predictions/size(data,1) * 100);

    % create the confusion matrix
    fig = confusionchart(labels, predicted_labels);

    title(strcat('Confusion Chart for Regularization Parameter: ', string(cur_par)));
    saveas(fig, strcat('Results/L1 Logistic Train/Conf_Chart_Reg_Par_', string(cur_par), '.jpg'));
end


function [w, c] = logL1Regression(data, labels, par)
% OUTPUT w is equivalent to the first d dimension of weights in logistic train
% c is the bias term, equivalent to the last dimension in weights in logistic train.
% Specify the options (use without modification).
opts.rFlag = 1; % range of par within [0, 1].
opts.tol = 1e-6; % optimization precision
opts.tFlag = 4; % termination options.
opts.maxIter = 5000; % maximum iterations

[w, c] = LogisticR(data, labels, par, opts);
end


```

```
function[val] = sigmoid(input)
    % sigmoid function implementation
    val = 1./(1 + exp(-input));
end
```

2.2 Experimental Outcome

As a part of the experiment, the regularization parameters was varied as:
$par = 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1$ Every different model was judged analyzed by 3 imporant metrics: Classification Accuracy, AUC and Number of non-zero weights. $L1$-regularization leads to sparse models, so it is really interesting to see how change in the regularization parameter can lead to a change in the number of non-zero weights. As number of zero-weights increase in the model, the number of features used to build the classification model reduces. Thus $L1$-regularization helps to perform Feature Selection.

The result of the experimentation is presented in Figure 3 and Table 1. From the Figure, it can be observed that the number of non-zero weights decreases as the value of regularization parameter is increased. This means that the model is becoming more sparse with increase in the value of the parameter. On the other hand, it can be seen that accuracy is increasing as opposite to the variation in number of non-zero weights. This observation was pretty interesting. Particularly, in the later stages of the experimentation, the model was using only one non-zero weight and it was still giving almost 75% accuracy. A careful look at the dataset made the idea clear. The test data consisted of imbalanced samples where approx. 75% samples were having label -1. So, the model was simply classifying every sample to -1 and that is the reason why it was getting such a high accuracy using only 1 feature. But it did not produce our intended model.

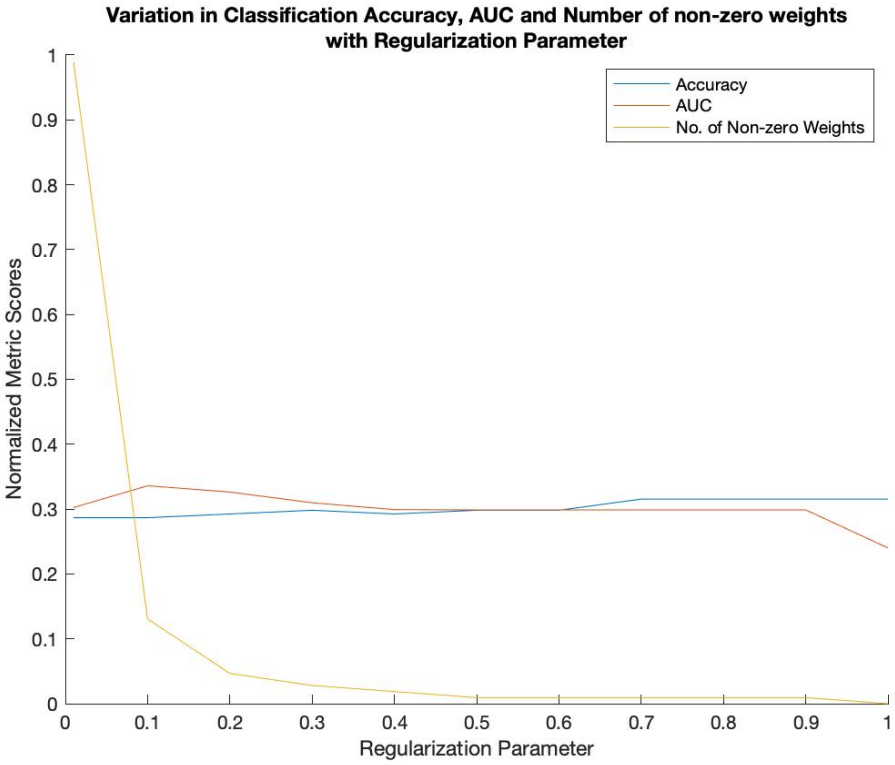

Figure 3: Plot of different metrics obtained for different values of regularized parameters

Par	**No. of Non-zero Weights**	**Accuracy**	**AUC**
0.01	106	67.567568	0.629665
0.1	14	67.567568	0.699522
0.2	5	68.918919	0.679426
0.3	3	70.27027	0.644976
0.4	2	68.918919	0.622967
0.5	1	70.27027	0.62201
0.6	1	70.27027	0.62201
0.7	1	74.324324	0.62201
0.8	1	74.324324	0.62201
0.9	1	74.324324	0.62201
1	0	74.324324	0.5

Table 1: Different Metric Scores for different values of the regularization parameter

In order to get some more insights to the problem, in the next stage, the confusion matrices for the experiments were plotted. The matrices are presented in Figure 4. After the parameter value reaches 0.7, it starts assigning all the labels to -1, so there's no false negatives or true positives. Although it is able to achieve good classification accuracy, it is not a good model. For this reason, in case of imbalanced data, classification accuracy cannot serve as an appropriate metric. AUC (Area Under Curve) is a better metric compared to accuracy in such a scenario. Intuitioanlly, AUC denotes the probability of a model assigning higher value to a random positive example in comparison to a random negative example. In terms of AUC, the second model of the experimentation, i.e. parameter value of 0.1, can be considered to be the best model out of all as it achieved the highest AUC score.

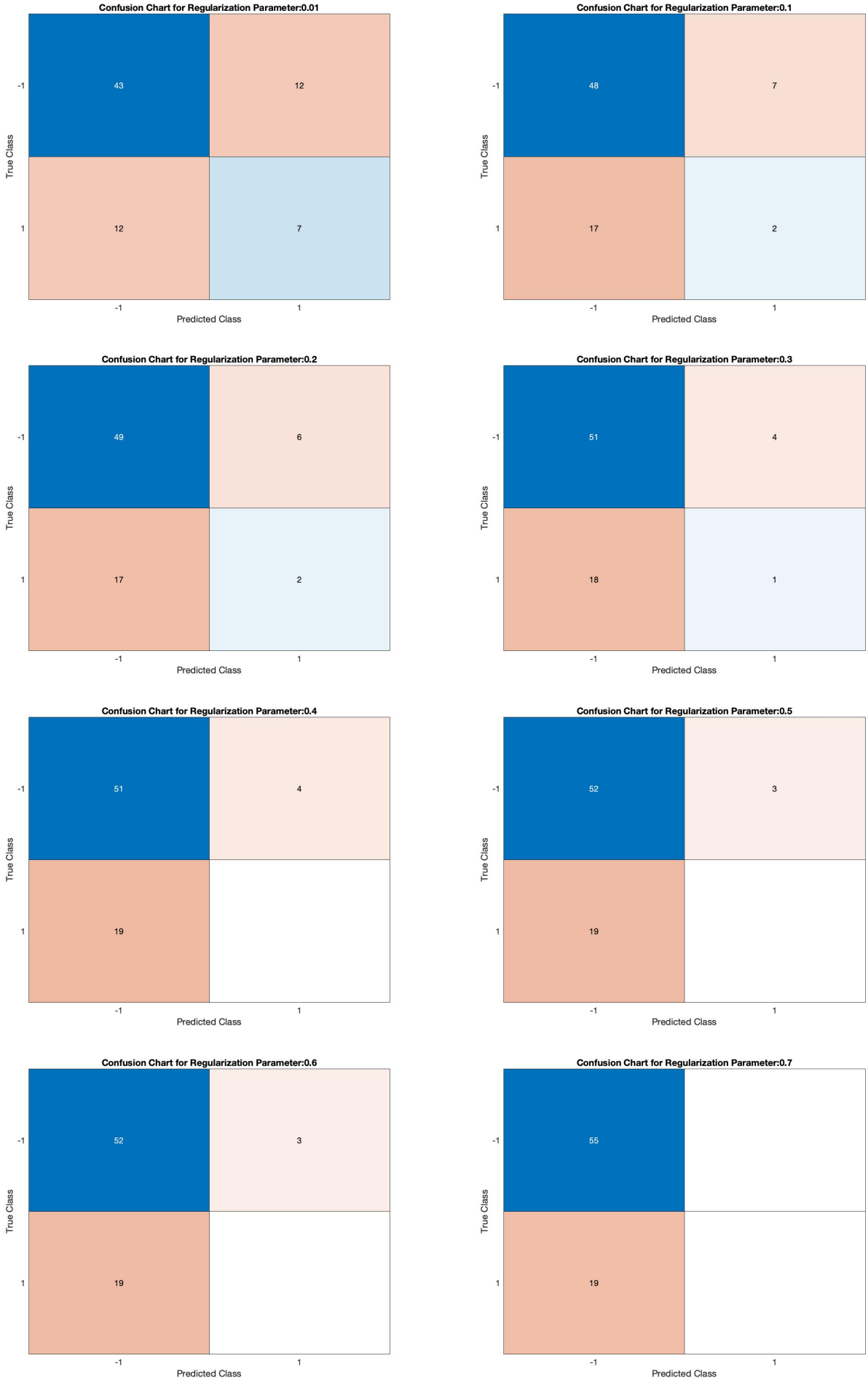

Figure 4: Plot of Confusion Matrices obtained for different values of the Regularization Parameter