Part 1: Create Nearest Neighbor Classifier

Reshaping training and testing images

Thare are 5000 training images and 500 testing images each with 28x28 pixels. If the training and testing images are visualized as stacked blocks, we can transform the training images into 784x5000 and testing images into 784x500 matrices.

If we proceed to transpose the training and testing matrices into 5000x784 and 500x784 matrices, this way each row of the training and testing matrices contain the information of each image.

```
reshaped_imageTrain = reshape(imageTrain,[28*28,5000])
reshaped_imageTrain = transpose(reshaped_imageTrain) % training matrix is now 5000x784
reshaped_imageTest = reshape(imageTest,[28*28,500])
reshaped_imageTest = transpose (reshaped_imageTest) % testing matrix is now 500x784
```

Initializing parameters

```
K = 3;  % let's consider the 3 nearest neighbors
size_of_test = size (reshaped_imageTest, 1);
size_of_train = size (reshaped_imageTrain, 1);

predictions = zeros(size_of_test, 1);

euclidean_list = zeros(size_of_test, size_of_train);
    % matrix will be updated to contain sorted euclidean distances

euclidean_index = zeros(size_of_test, size_of_train);
    % matrix will be updated to contain the index of sorted euclidean distances
```

Calculate Euclidean distances

- Due to the reshaping, every row now contains information of an image.
- The 2 for loops below will calculate the Euclidean distances by taking every row of testing image matrix and subtract the row of every training images; the values of each subtraction are then summed and square rooted.
- The Euclean values will then be sorted from smallest to largest and stored in 'euclidian_list'. The location of their indexes will be stored in 'euclidian_list'

```
for test_index=1:size_of_test % take every point of testing data for 500 data points

    for train_index=1:size_of_train %every point of training data for 5000 data points
       distance = (reshaped_imageTest(test_index,:)-reshaped_imageTrain(train_index,:)).^2;
       euclidean_list(test_index,train_index)=sqrt(sum(distance));
    end

       [euclidean_list(test_index,:),euclidean_index(test_index,:)] = sort( ...
```

```
            euclidean_list(test_index,:));

end
```

Finding the K nearest neighbors

```
knn=euclidean_index(:,1:K); % takes the first 3 smallest or shortest Euclidean distances

for i=1:size(knn,1)

    most_common=mode(labelTrain(knn(i,:)'));
        % the function 'mode' gives the most occuring value in an array

    predictions(i)=most_common; % the predicted number is stored

end
```

Calculating the error of each class from 0 to 9 by calling function 'calc_error'

A function name 'calc_error' is used to calculate the error of each class to avoid redundancy. Due to how Matlab is structured the function can only be placed at the bottom of the script.

Below is what the function looks like, the real location of the function is at the very bottom.

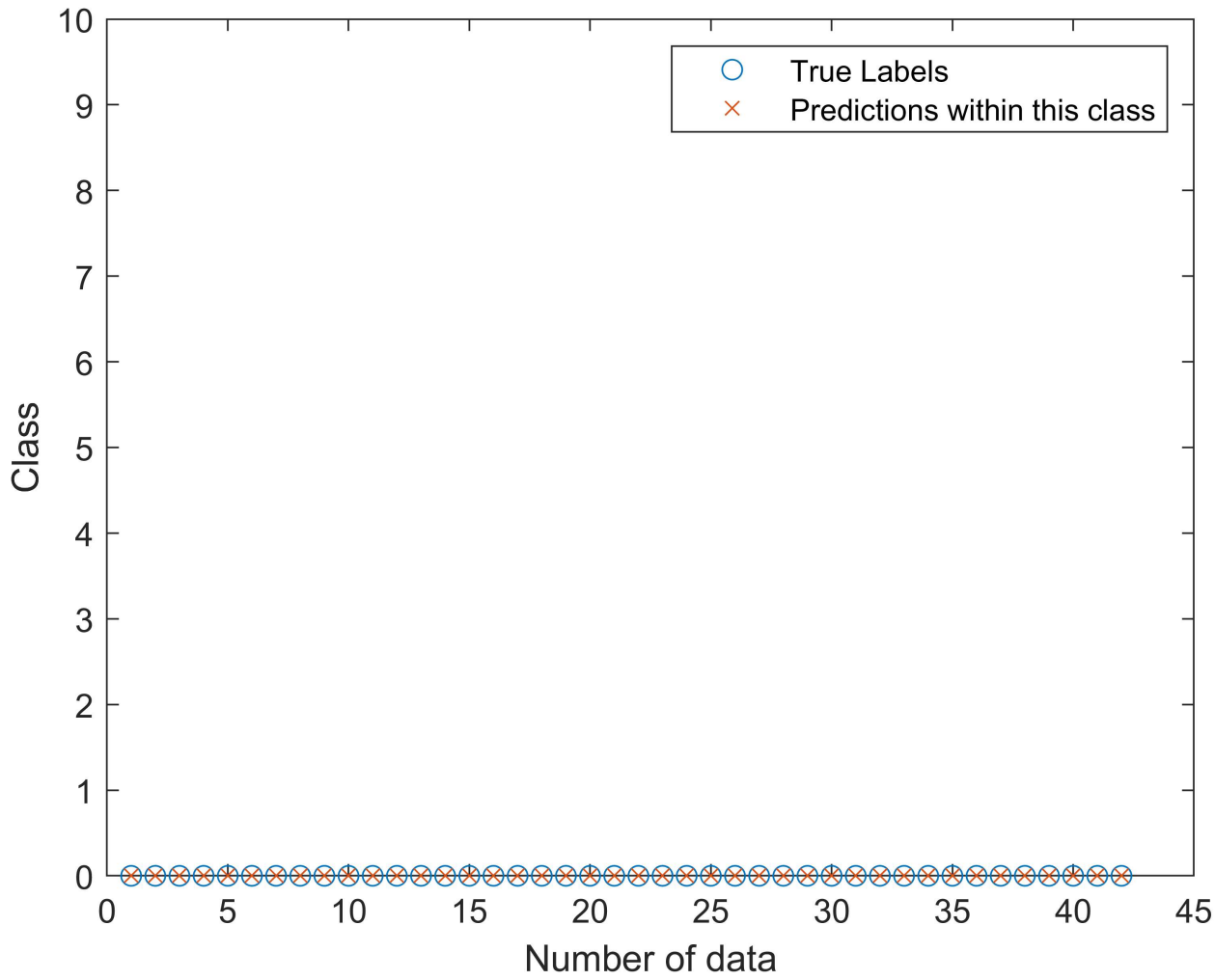
```
% function[error] = calc_error(labelTest, predictions, target_number)
%
%     a = find(labelTest==target_number);
%     error = length(find(labelTest(a)~=predictions(a)))/length(labelTest(a))*100;

%     %error plot for each class
%     figure
%     plot (labelTest(a), 'o');
%     hold on
%     plot (predictions(a),'x');
%     legend ('True Labels', 'Predictions within this class');
%     ylabel('Class');
%     xlabel('Number of data');
%     ylim ([0 10]);

% end
```

```
%To be put into funtion calc_error
true_labels = labelTest;
predicted_labels= predictions;
```

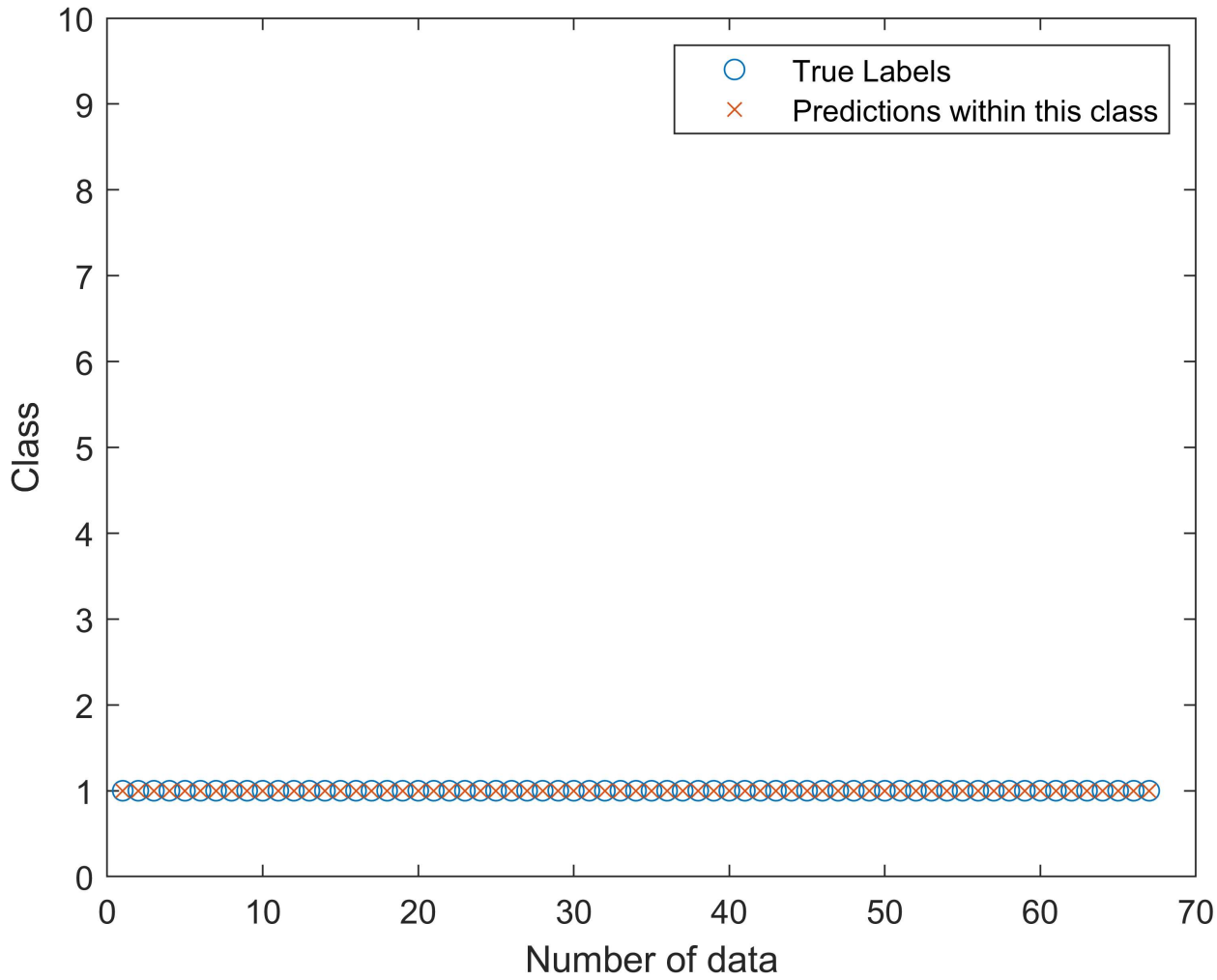
```
error_class_0 = calc_error(true_labels, predicted_labels,0);
```




```
error_class_0
```

```
error_class_0 = 0
```

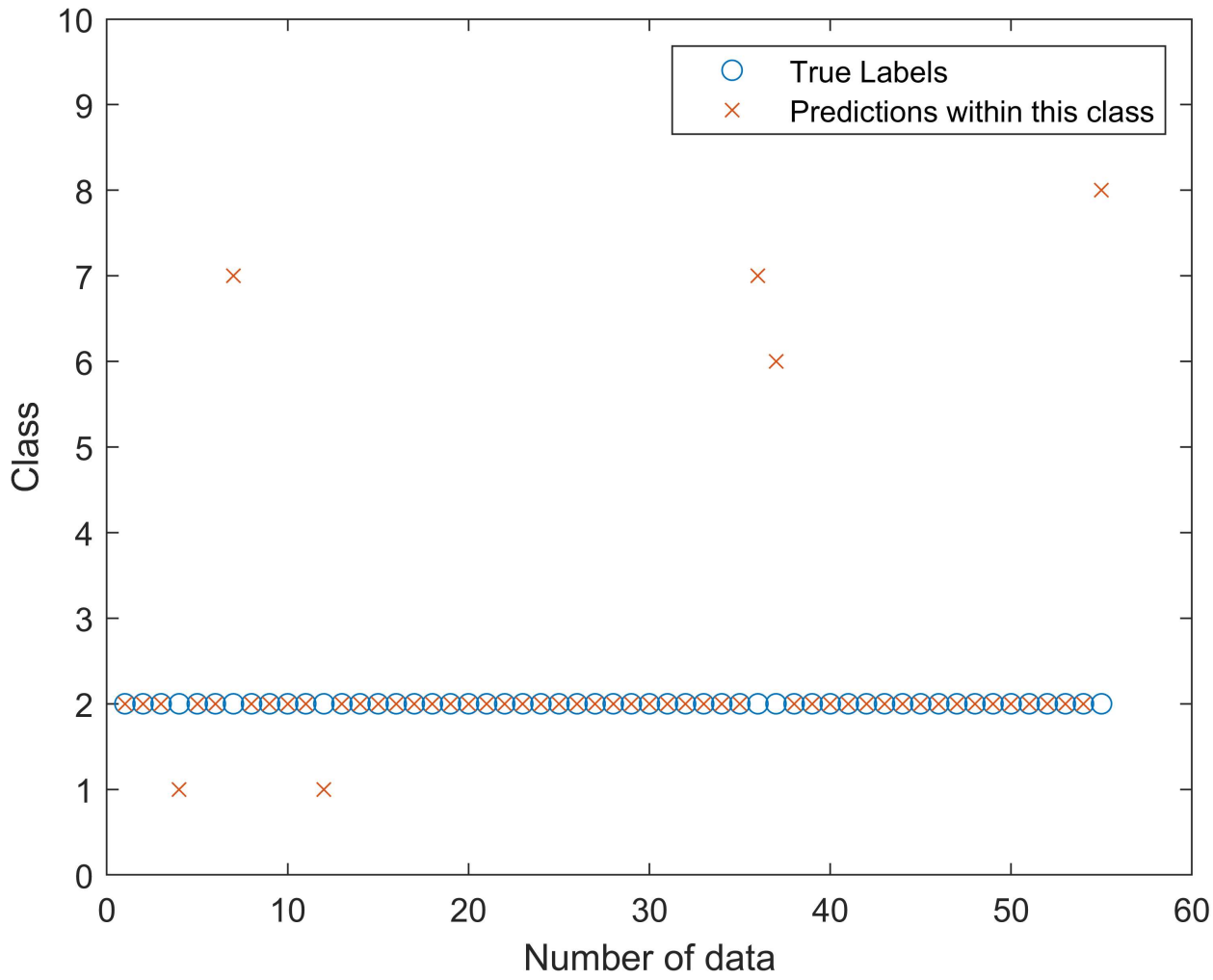
```
error_class_1 = calc_error(true_labels, predicted_labels,1);
```




```
error_class_1
```

```
error_class_1 = 0
```

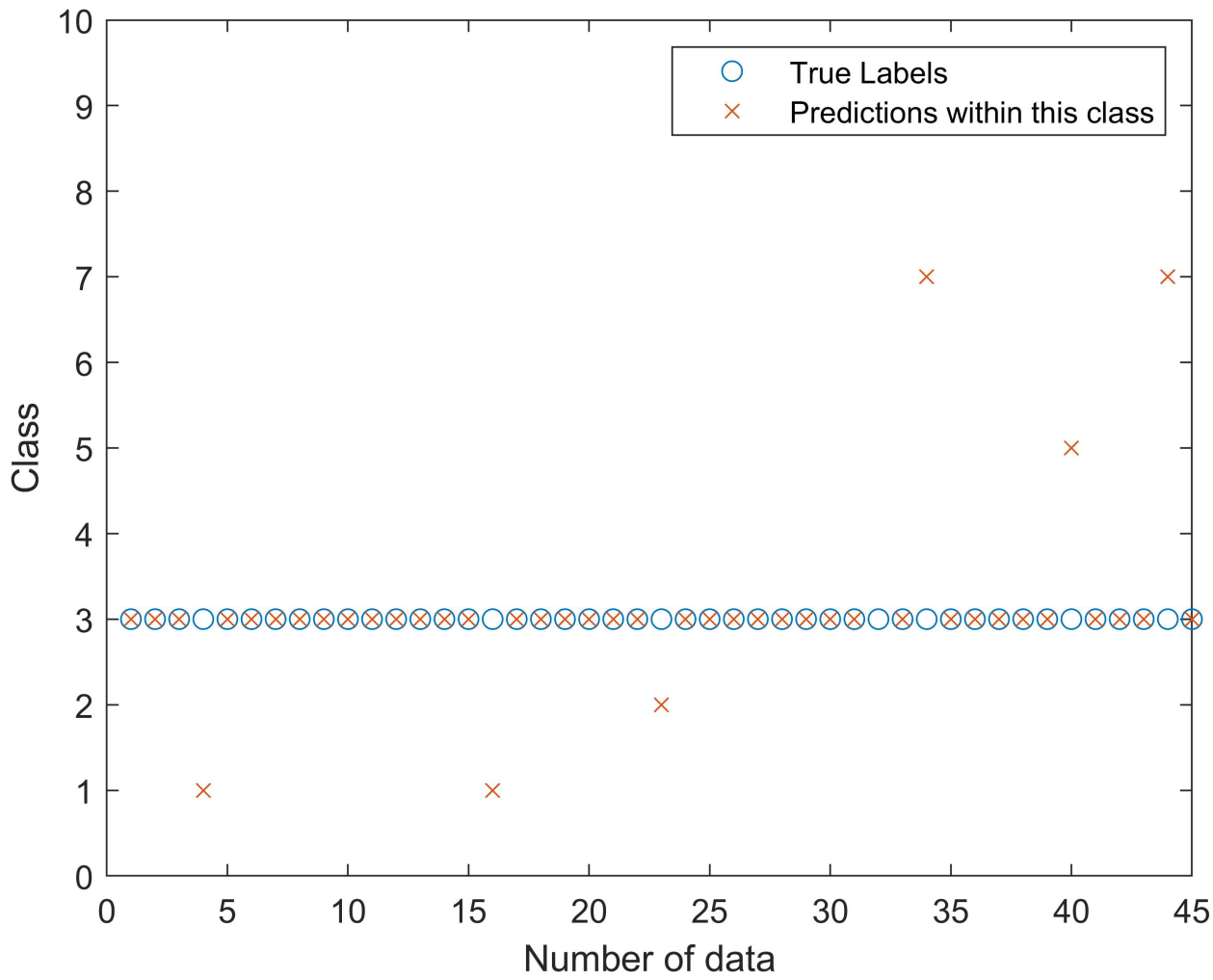
```
error_class_2 = calc_error(true_labels, predicted_labels,2);
```




```
error_class_2
```

```
error_class_2 = 10.9091
```

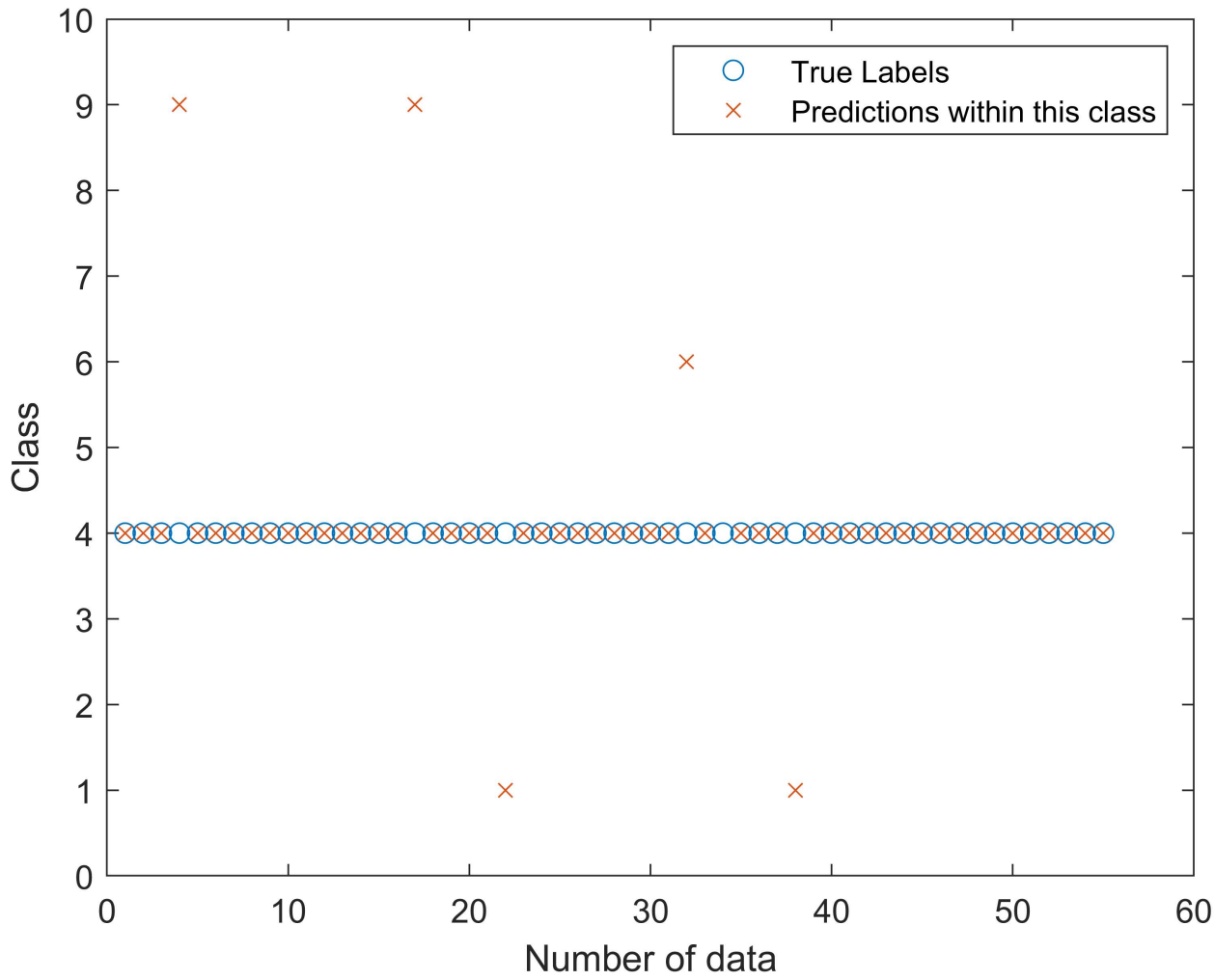
```
error_class_3 = calc_error(true_labels, predicted_labels,3);
```




```
error_class_3
```

```
error_class_3 = 15.5556
```

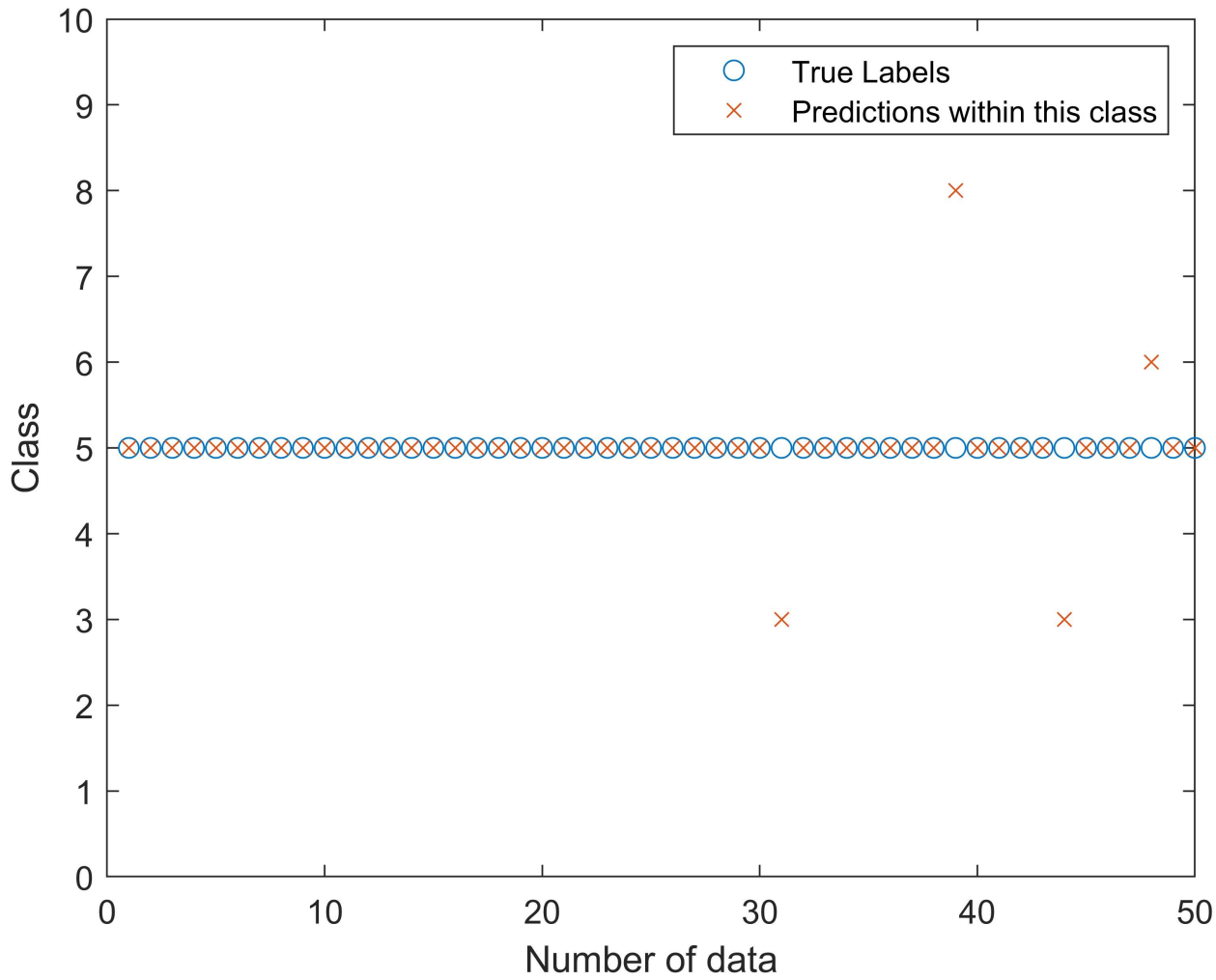
```
error_class_4 = calc_error(true_labels, predicted_labels,4);
```




```
error_class_4
```

```
error_class_4 = 10.9091
```

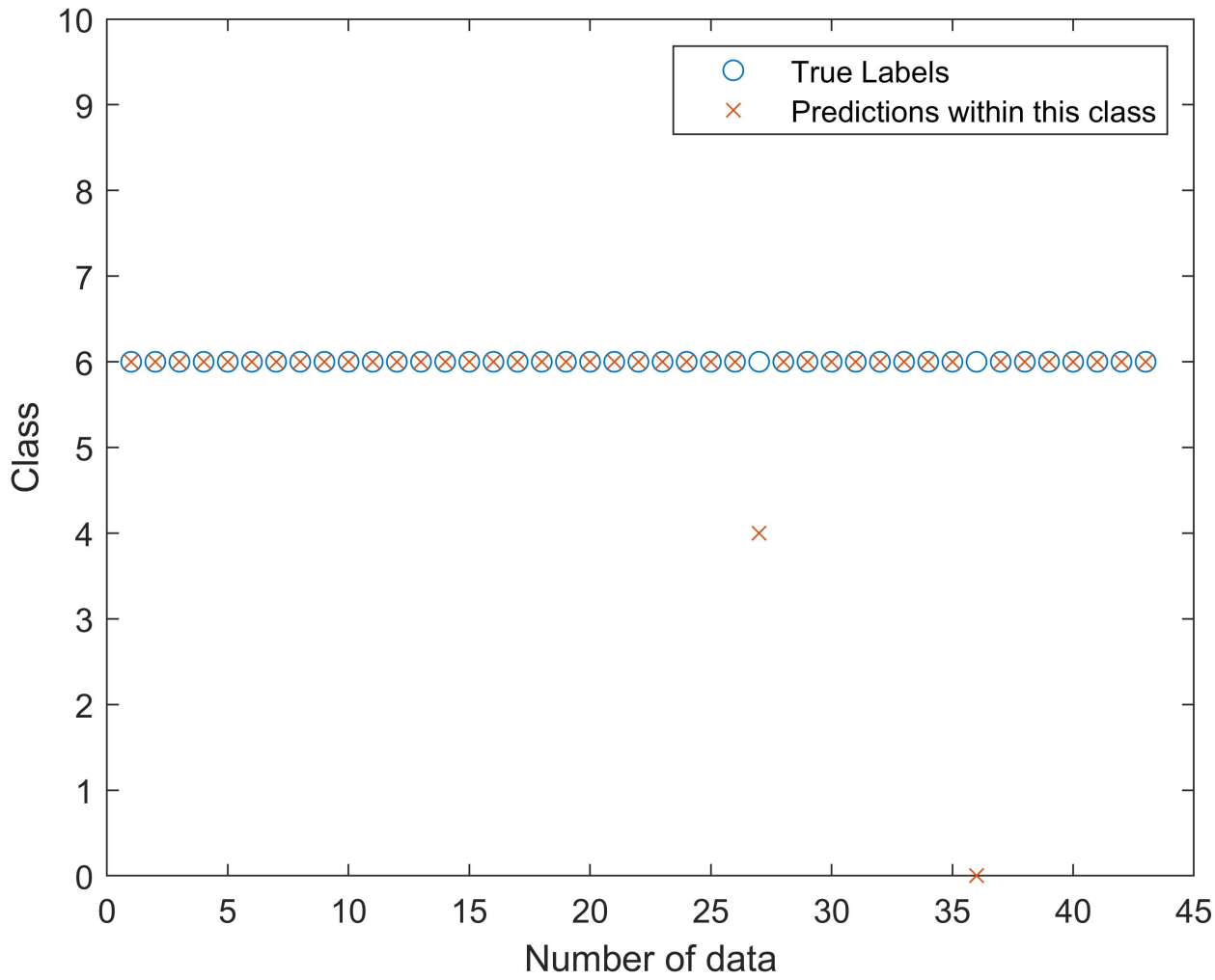
```
error_class_5 = calc_error(true_labels, predicted_labels,5);
```




```
error_class_5
```

```
error_class_5 = 8
```

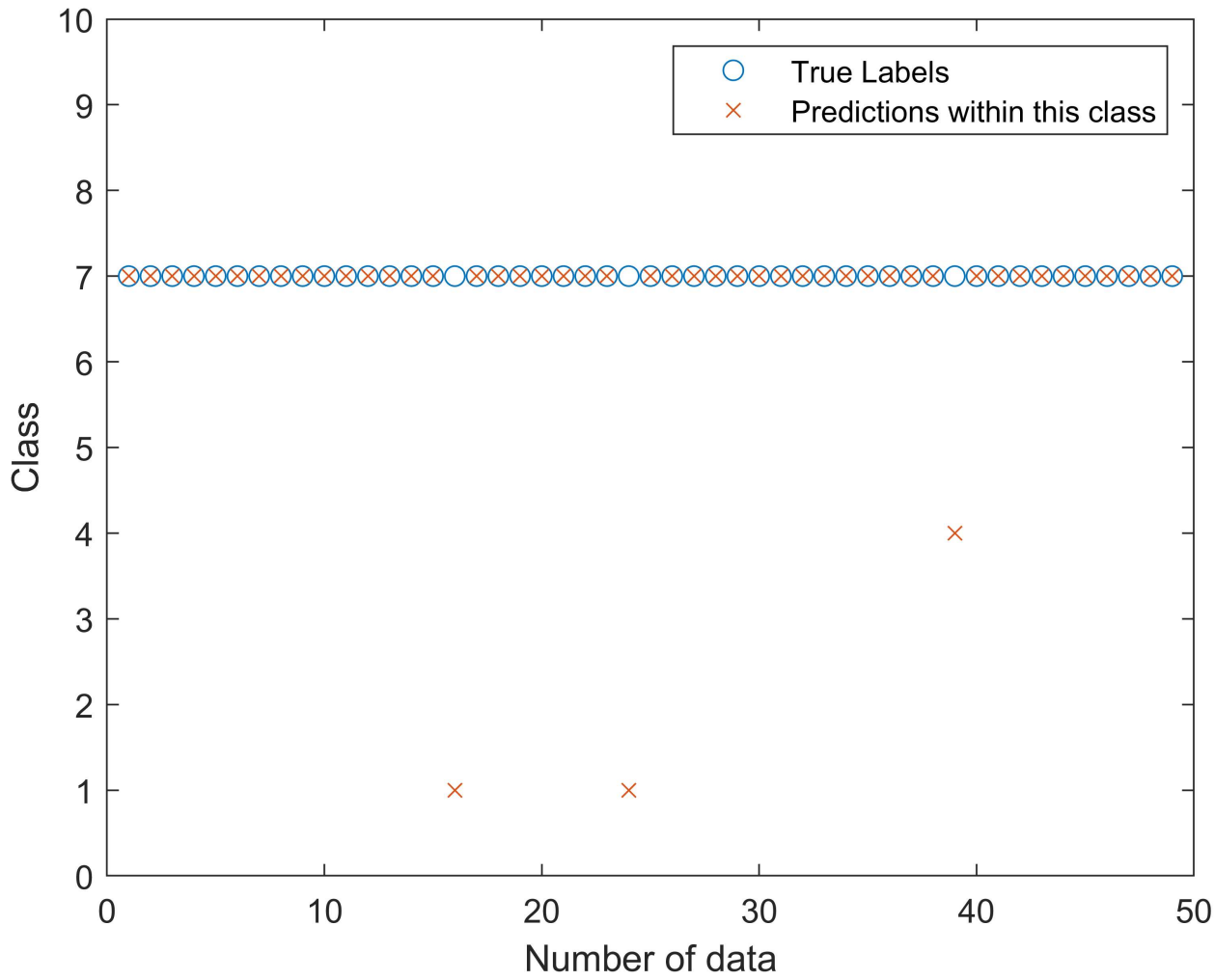
```
error_class_6 = calc_error(true_labels, predicted_labels,6);
```




```
error_class_6
```

```
error_class_6 = 4.6512
```

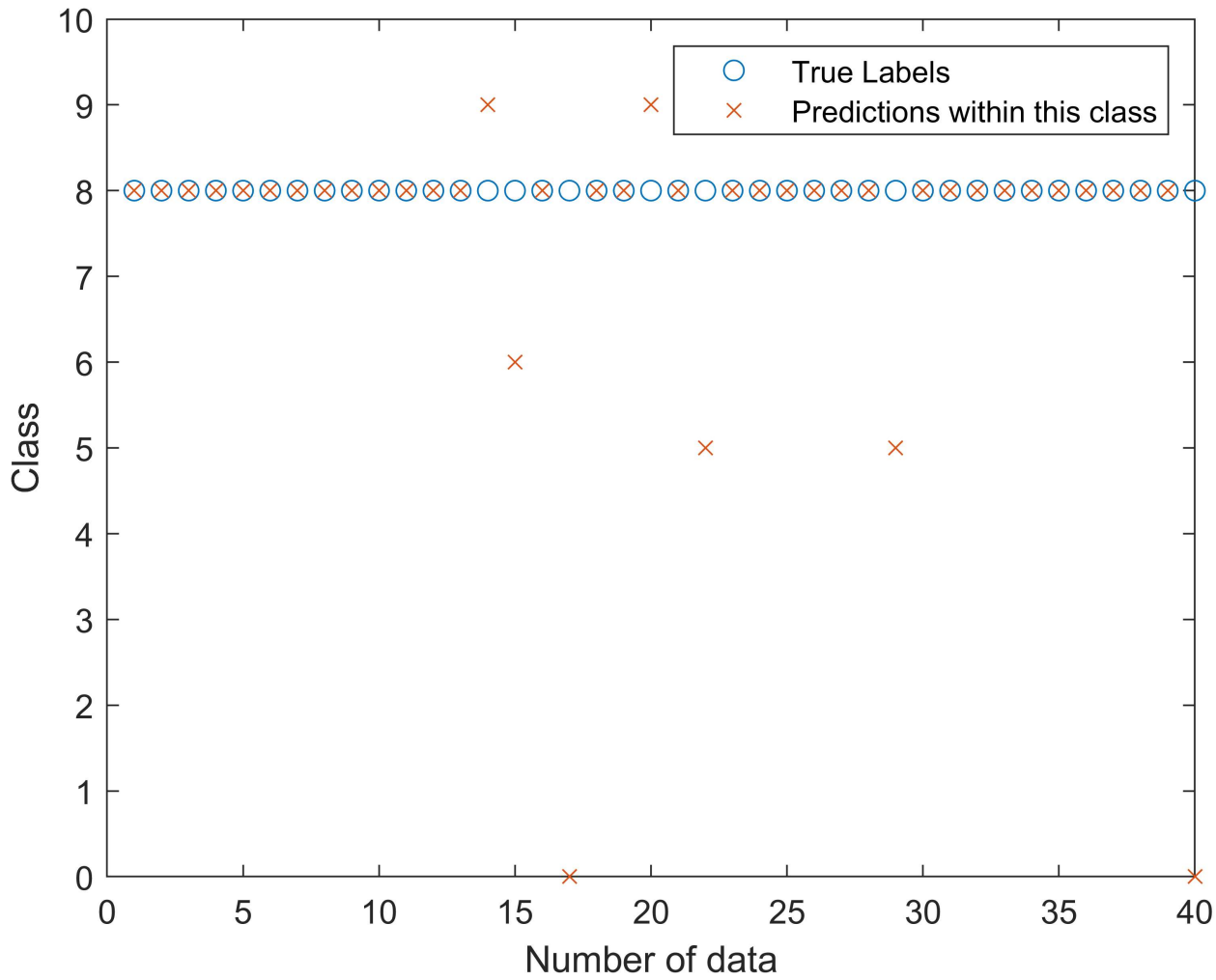
```
error_class_7 = calc_error(true_labels, predicted_labels,7);
```




```
error_class_7
```

```
error_class_7 = 6.1224
```

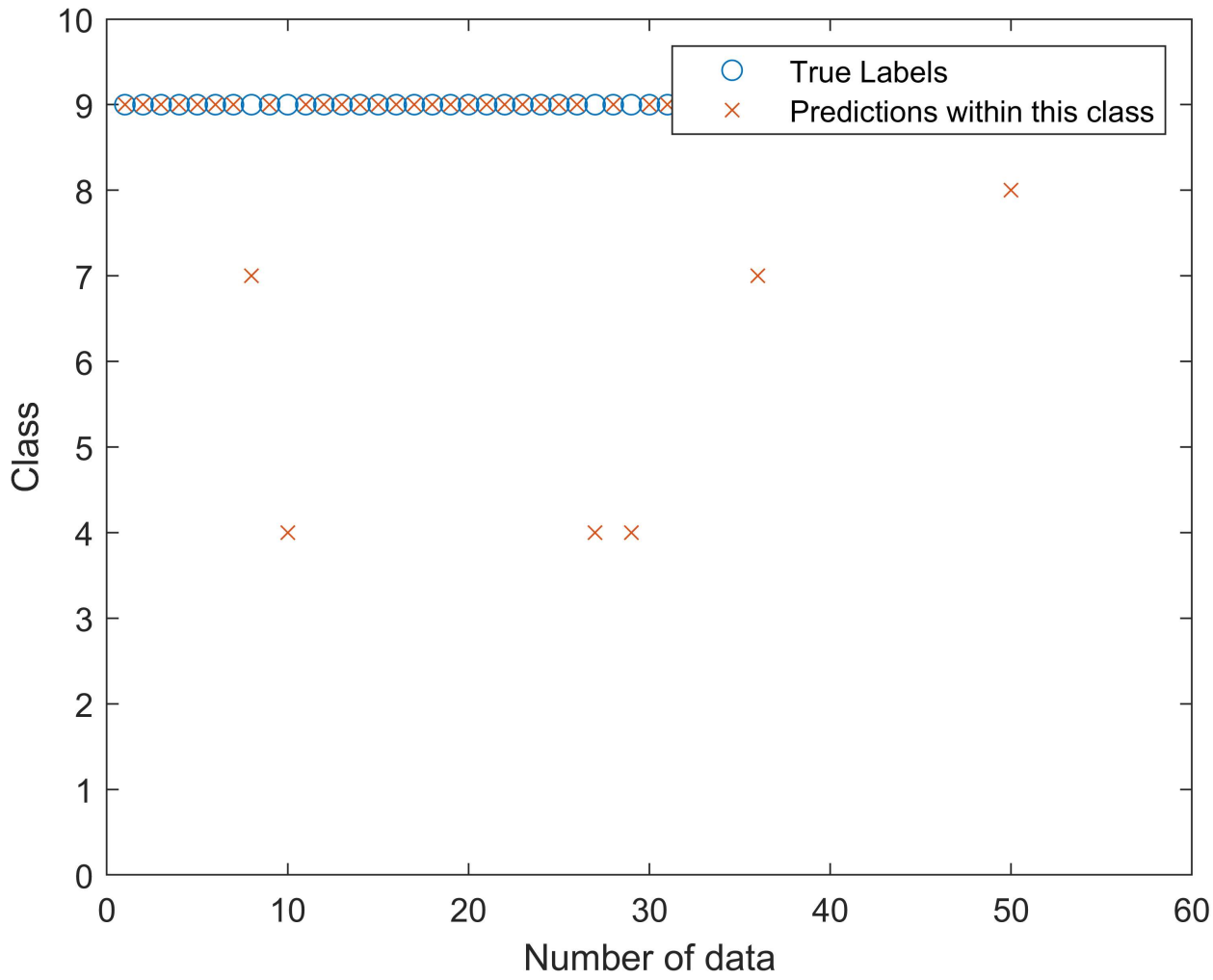
```
error_class_8 = calc_error(true_labels, predicted_labels,8);
```




```
error_class_8
```

```
error_class_8 = 17.5000
```

```
error_class_9 = calc_error(true_labels, predicted_labels,9);
```




```
error_class_9
```

```
error_class_9 = 11.1111
```

Part 2: Calculating accuarcy rate and error rate

Below is the total accuracy rate of the classifier

```
total_accuracy = size (find(predictions==labelTest),1)/size(labelTest,1)*100;
disp(total_accuracy);
```

```
   91.8000
```

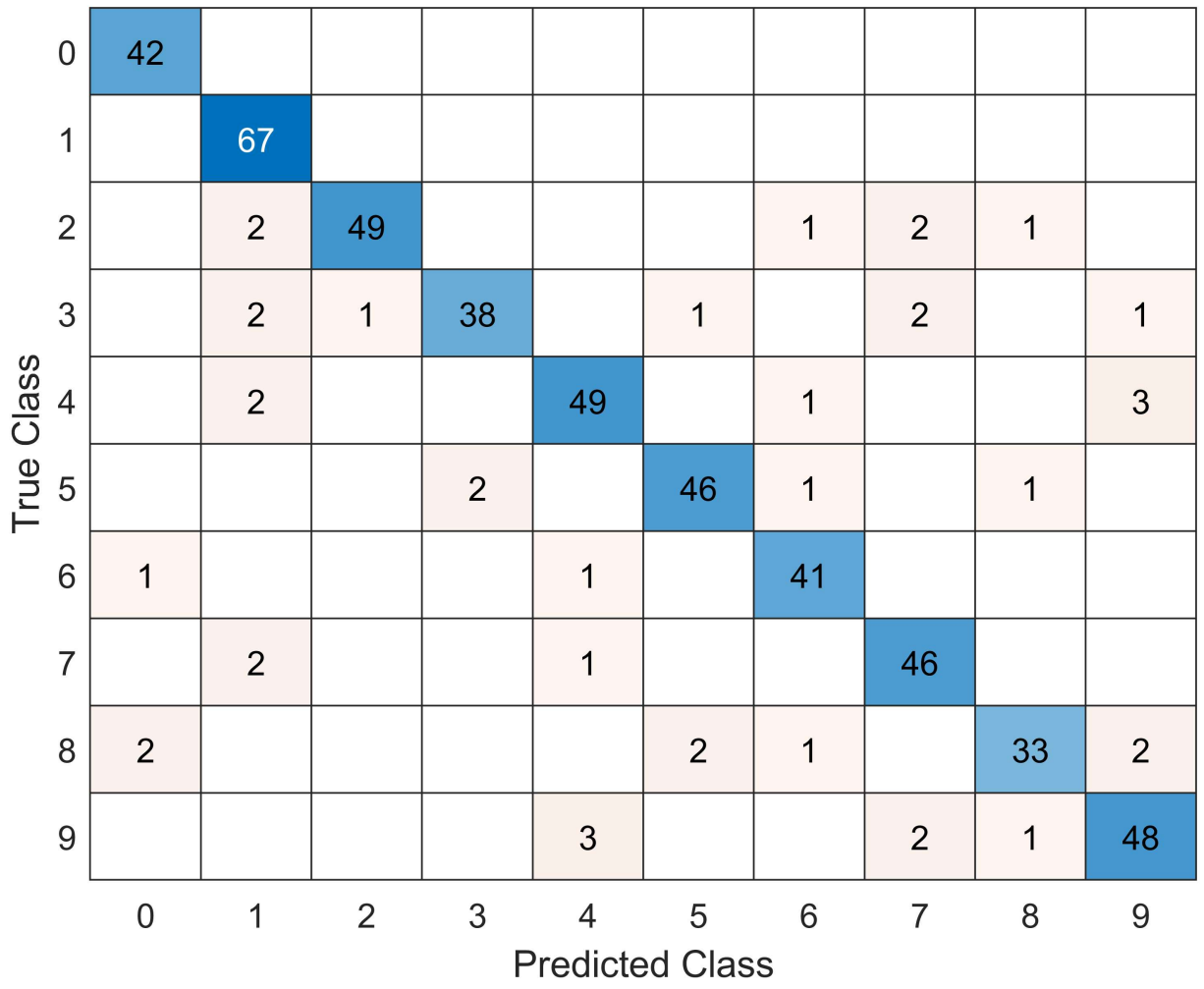
Below is the total error rate of the classifier

```
total_error = size (find(predictions~=labelTest),1)/size(labelTest,1)*100;
disp(total_error);
```

```
   8.2000
```

Confusion matrix to visualize data

```
figure
Confusion_matrix = confusionchart(labelTest,predictions);
```


True Class
0 42
1 67
2 2 49 1 2 1
3 2 1 38 1 2 1
4 2 49 1 3
5 2 46 1 1
6 1 1 41
7 2 1 46
8 2 2 1 33 2
9 3 2 1 48
0 1 2 3 4 5 6 7 8 9
Predicted Class