

# Homework Assignment 4

By: Pujan Thakrar

## PART 1

**Question 1:** codes provided in main\_ha4

We are given a dataset that contains images with the numbers 0-9, written in different ways. Our task is to apply machine learning techniques to the data in order to accurately read the numbers from images.

First, I load in and split the data into testing and training samples using the 'mnist.load\_data()' function. After which, I transform each image from a 28x28 to a 784-pixel vector. Then, I normalize the inputs from gray scale to be values between 0-1.

I then further split the data into training and validation data. I use the new training data to build 3 models using different sets of hyperparameters.

Here are the hyperparameters for each model:

### *Model 1*

- i) Input layer contains 150 neurons, with activation function 'sigmoid';
- ii) Hidden layer 1 contains 200 neurons, with activation function 'relu';
- iii) Hidden layer 2 contains 250 neurons, with activation function 'sigmoid';
- iv) Output layer contains 10 neurons, with activation function 'softmax';
- v) Default learning rate for RMSprop() is used

### *Model 2*

- i) Input layer contains 100 neurons, with activation function 'tanh';
- ii) Hidden layer 1 contains 300 neurons, with activation function 'relu';
- iii) Hidden layer 2 contains 240 neurons, with activation function 'softmax';
- iv) Output layer contains 10 neurons, with activation function 'sigmoid';
- v) Default learning rate for RMSprop() is used

### *Model 3*

- i) Input layer contains 250 neurons, with activation function 'relu';
- ii) Hidden layer 1 contains 250 neurons, with activation function 'softmax';
- iii) Hidden layer 2 contains 250 neurons, with activation function 'relu';
- iv) Output layer contains 10 neurons, with activation function 'softmax';
- v) Default learning rate for RMSprop() is used

I then convert training data to integers and fit models to my training set. After this, I use the model to make predictions of the testing set. Using the function 'func\_confusion\_matrix()' from util.py, I am able to get the confusion matrices and evaluation metrics including accuracy score, per-class recall and precision rates. I get the following output for each model:


Model 1:

```
In [123]:  #confusion matrix and evaluation metrics
```

```
#model 1  
func_confusion_matrix(y_test, y_pred1)
```

```
Out[123]: (array([[ 960,    0,    3,    0,    0,    2,   12,    1,    2,    0],  
                  [    0, 1124,    5,    0,    0,    1,    2,    0,    3,    0],  
                  [    2,    0, 1019,    1,    1,    0,    3,    3,    3,    0],  
                  [    0,    1,   15,  974,    0,    5,    1,    7,    7,    0],  
                  [    0,    0,    4,    0,  965,    0,    9,    3,    0,    1],  
                  [    2,    0,    0,    5,    1,  869,   10,    1,    3,    1],  
                  [    1,    3,    0,    0,    3,    6,  945,    0,    0,    0],  
                  [    2,    5,   24,    0,    0,    0,    0,  993,    2,    2],  
                  [    1,    1,   11,    3,    1,    5,    7,    5,   938,    2],  
                  [    1,    6,    1,    7,   25,   10,    1,   12,    7,   939]]),  
0.9726,  
array([0.97959184, 0.99030837, 0.9874031 , 0.96435644, 0.98268839,  
       0.97421525, 0.98643006, 0.96595331, 0.96303901, 0.93062438]),  
array([0.99071207, 0.98596491, 0.94177449, 0.98383838, 0.9688755 ,  
       0.96770601, 0.95454545, 0.96878049, 0.97202073, 0.99365079]))
```

Model 2:

```
In [119]:  #model 2  
func_confusion_matrix(y_test, y_pred2)
```

```
Out[119]: (array([[ 974,    0,    0,    1,    0,    1,    1,    0,    2,    1],  
                  [    0, 1122,    2,    2,    0,    0,    2,    2,    5,    0],  
                  [  11,    2,  974,    4,    3,    0,   22,    4,   10,    2],  
                  [    1,    0,    2,  904,    0,   84,    0,    2,   14,    3],  
                  [    3,    0,    2,    1,  723,    0,    5,    0,    0,  248],  
                  [    5,    0,    0,  354,    1,  515,    4,    0,    8,    5],  
                  [  15,    3,   16,    1,    3,    3,  910,    0,    7,    0],  
                  [    3,    3,   10,    6,    2,    0,    0,  979,    7,   18],  
                  [  12,    0,    3,   19,    1,   11,    1,    0,  921,    6],  
                  [    5,    2,    0,    6,   20,    9,    0,    4,    4,   959]]),  
0.8981,  
array([0.99387755, 0.98854626, 0.94379845, 0.8950495 , 0.73625255,  
       0.57735426, 0.94989562, 0.95233463, 0.94558522, 0.95044599]),  
array([0.94655005, 0.99116608, 0.96531219, 0.69645609, 0.96015936,  
       0.82664526, 0.96296296, 0.98789102, 0.94171779, 0.77214171]))
```

Model 3:

In [120]: ▶

```
#model 3
func_confusion_matrix(y_test, y_pred3)
```

```
Out[120]: (array([[ 969,    0,    2,    0,    1,    1,    2,    1,    3,    1],
 [    0, 1130,    2,    0,    0,    0,    1,    0,    2,    0],
 [    3,    8, 1003,    2,    0,    0,    4,    5,    7,    0],
 [    0,    0,    0, 993,    0,    7,    0,    4,    4,    2],
 [    1,    1,    3,    0, 947,    0,    6,    2,    4,   18],
 [    2,    0,    0,    6,    1, 872,    4,    0,    6,    1],
 [    6,    3,    0,    1,    6,    9, 931,    0,    2,    0],
 [    2,    6,    8,    5,    0,    0,    0, 994,    6,    7],
 [    1,    0,    2,    6,    3,    3,    2,    2, 951,    4],
 [    1,    2,    0,   12,    3,    6,    0,    3,    2, 980]]),
0.977,
array([0.98877551, 0.99559471, 0.97189922, 0.98316832, 0.96435845,
       0.97757848, 0.97181628, 0.96692607, 0.97638604, 0.97125867]),
array([0.98375635, 0.9826087 , 0.98333333, 0.96878049, 0.98543184,
       0.97104677, 0.98      , 0.98318497, 0.96352584, 0.96742349]))
```

From these results, we can see that Model 3 performed the best with an accuracy score of 0.977. Model 1 was a close second with a score of 0.973 and Model 2 did the worst over all with a score of 0.898.

**Question 2:** codes provided in main\_ha4

After having determined that Model 3 performed the best overall, I apply it to the validation data and get the following confusion matrix and evaluation metrics:

In [121]: ▶

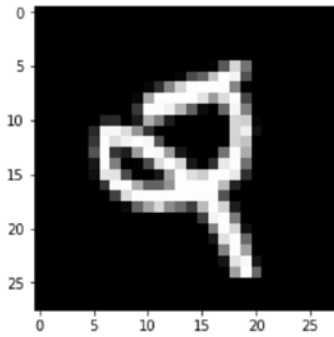
```
#Question 2
#running model 3 over validation set
y_pred3_val=mod3.predict_classes(x_valid)

func_confusion_matrix(y_valid, y_pred3_val)
```

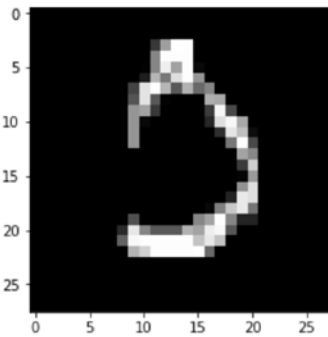
```
Out[121]: (array([[ 969,    1,    5,    0,    0,    2,    0,    0,    1,    2],
 [    0, 1126,    1,    1,    1,    0,    0,    3,    2,    0],
 [    2,    8, 943,    6,    1,    3,    1,    4,    5,    1],
 [    2,    2,    4, 1014,    0,   17,    0,    2,    5,    1],
 [    0,    2,    1,    1, 968,    0,    4,    3,    3,   34],
 [    0,    3,    0,    8,    2, 846,    2,    0,    6,    2],
 [    3,    4,    2,    1,    3,    1, 975,    0,    1,    0],
 [    1,    2,    8,    2,    2,    0,    0, 1055,    0,   10],
 [    3,    0,    5,    6,    1,    6,    4,    2, 936,    6],
 [    1,    2,    0,   12,    4,    3,    0,    3,    3, 913]]),
0.9745,
array([0.98877551, 0.99294533, 0.96817248, 0.96848138, 0.95275591,
       0.9735328 , 0.98484848, 0.97685185, 0.96594427, 0.97024442]),
array([0.98776758, 0.97913043, 0.97316821, 0.96479543, 0.98574338,
       0.96355353, 0.98884381, 0.98414179, 0.97297297, 0.94220846]))
```

We can see the accuracy is still quite high for the validation set. I then built a for loop that would print out the first ten cases where my model did not successfully predict the right value. Here was the output:

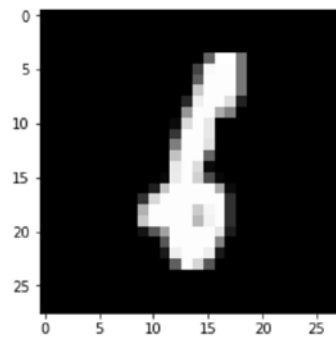
Failed case at index # 54 :  
predicted: 9 true: 2



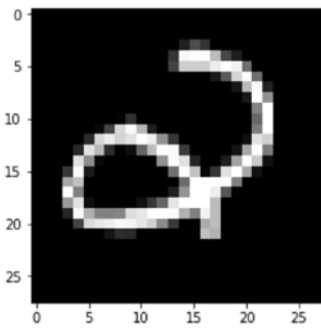
Failed case at index # 74 :  
predicted: 2 true: 0



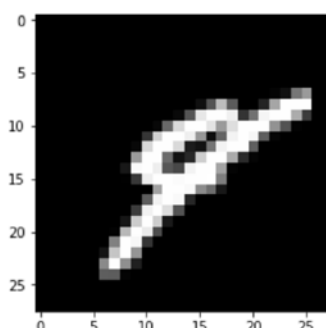
Failed case at index # 75 :  
predicted: 1 true: 6



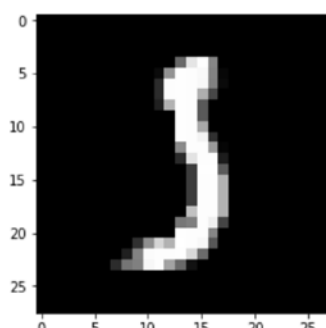
Failed case at index # 113 :  
predicted: 5 true: 2



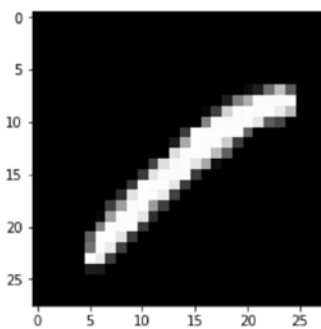
Failed case at index # 155 :  
predicted: 8 true: 4



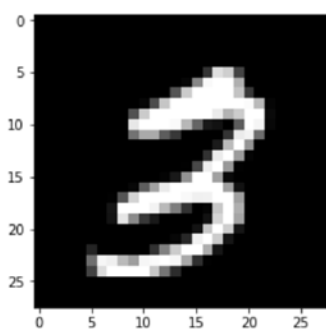
Failed case at index # 165 :  
predicted: 1 true: 5



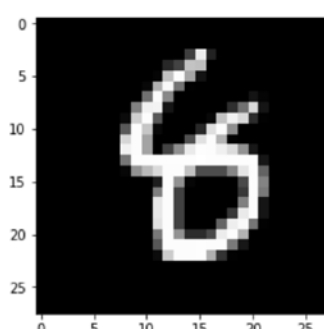
Failed case at index # 200 :  
predicted: 8 true: 1



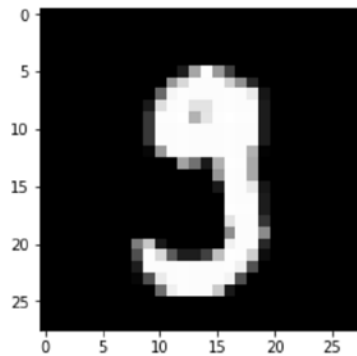
Failed case at index # 212 :  
predicted: 2 true: 3



Failed case at index # 295 :  
predicted: 6 true: 8



Failed case at index # 332 :  
predicted: 3 true: 9



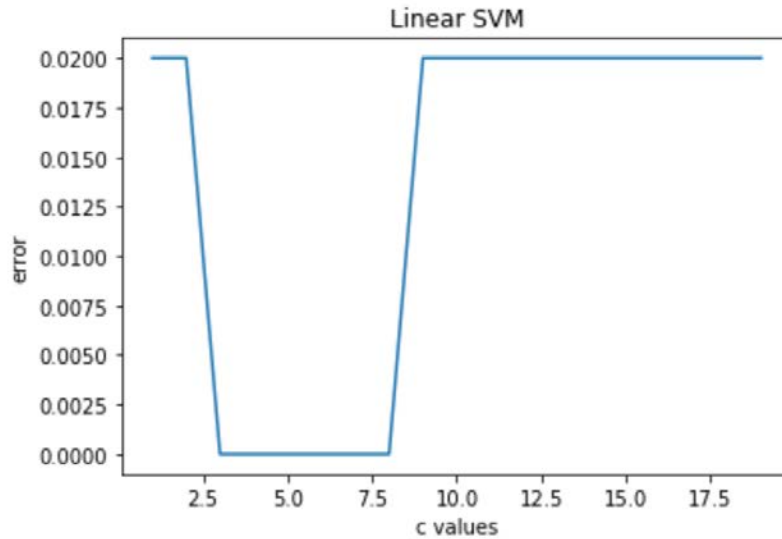
From these errors, we can see that curvy numbers, numbers that were written on angles, and numbers that are not completed were harder to predict. For example, the model had a harder time predicting 2s from this sample as they are curvy and can be written differently and mistaken for other numbers. For the failed case at index 200, we see that because the number 1 was written on an angle, it was harder for the model to predict. Finally, with the failed case at 295, we can see that because the 8 wasn't fully closed, the model thought it was a 6.

## PART 2

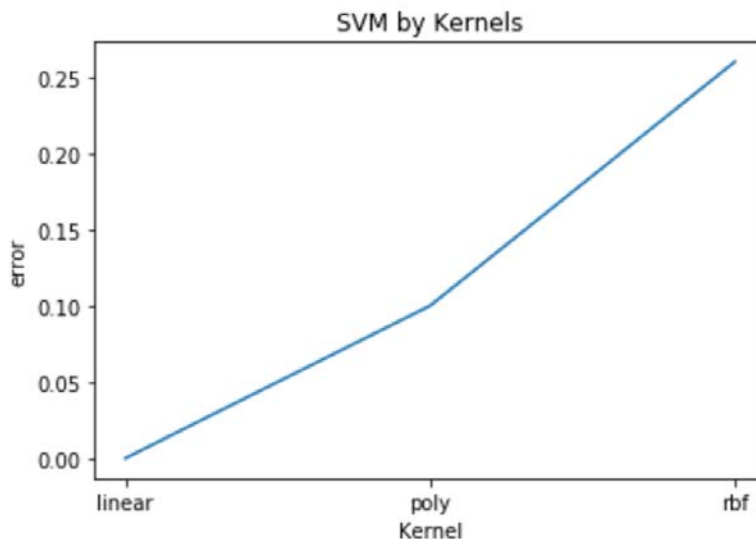
In this mini-project we use 6 features of a crab to determine if the crab is either male or female. There are 200 samples with gender labels provided.

We start by loading in the data and split the samples into two even subsets of length 100. One dataset will be used for training and validation, while the other is used for testing. After, I randomly divide the first dataset into two even subsets of length 50. One is used from training and the other is used for validation.

I then consider different weighting parameters with a linear kernel within an SVM model, in order to determine which weighting parameter yields the lowest error. We apply this model to the validation set. The result is shown in the plot below.



We can see that the lowest error occurs between approximately 2.6 and 7.6. Given this, I choose the weighting parameter value of 5 and then test for different kernel types. Again, I apply the models to the validation set. The result is given by the plot below.



We see that the linear kernel provides the lowest error in this case and therefore, I take the best model (linear kernel and weighting value = 5) and apply it over the testing set. I yield the following evaluation metrics:

```
In [32]: ► ## step 5 evaluate your results in terms of accuracy, real, or precision.

#####placeholder 5: metrics #####

y_pred = model.predict(X_test)
conf_matrix, accuracy, recall_array, precision_array = func_confusion_matrix

print("Confusion Matrix: ")
print(conf_matrix)
print("Average Accuracy: {}".format(accuracy))
print("Per-Class Precision: {}".format(precision_array))
print("Per-Class Recall: {}".format(recall_array))

#####placeholder end #####
```

Confusion Matrix:  
[[45 3]  
 [ 2 50]]  
Average Accuracy: 0.95  
Per-Class Precision: [0.95744681 0.94339623]  
Per-Class Recall: [0.9375 0.96153846]

We see that the accuracy is quite high and therefore our model performed well. Finally, I created two for loops that would print out both the successful cases and failure cases from the data. I have provided a subset of the successful examples and all the failure examples below.

#### Successful examples

```
Y_test: 1.0 y_pred: 1.0
Y_test: -1.0 y_pred: -1.0
Y_test: -1.0 y_pred: -1.0
Y_test: 1.0 y_pred: 1.0
Y_test: -1.0 y_pred: -1.0
Y_test: -1.0 y_pred: -1.0
Y_test: 1.0 y_pred: 1.0
Y_test: -1.0 y_pred: -1.0
Y_test: 1.0 y_pred: 1.0
Y_test: -1.0 y_pred: -1.0
Y_test: -1.0 y_pred: -1.0
Y_test: 1.0 y_pred: 1.0
Y_test: -1.0 y_pred: -1.0
Y_test: -1.0 y_pred: -1.0
Y_test: -1.0 y_pred: -1.0
Y_test: -1.0 y_pred: -1.0
Y_test: -1.0 y_pred: -1.0
Y_test: -1.0 y_pred: -1.0
Y_test: -1.0 y_pred: -1.0
Y_test: 1.0 y_pred: 1.0
Y_test: -1.0 y_pred: -1.0
Y_test: 1.0 y_pred: 1.0
```

#### Failure examples

```
Y_test: 1.0 y_pred: -1.0
Y_test: 1.0 y_pred: -1.0
Y_test: -1.0 y_pred: 1.0
Y_test: 1.0 y_pred: -1.0
Y_test: -1.0 y_pred: 1.0
```

In total, there were 95 successful cases, and 5 failure cases.