Middlesex University London

CST3170

Artificial Intelligence

Optical Recognition of Handwritten Digits

Coursework 2 – Final Report

Maaz Chowdhry

M00910300

Abstract

This report explores three distinct machine learning approaches employed to tackle the problem of handwritten digit classification: Euclidean distance categorisation, Multi-Layer Perceptron (MLP), and Support Vector Machines (SVM) optimised using the Sequential Minimal Optimization (SMO) algorithm. I discuss each technique's methodology, performance results, and relative strengths and weaknesses.

1. Euclidean Distance Classification

Methodology: The Euclidean distance categoriser operates on the principle of nearest neighbours. The Euclidean distance is calculated between an unclassified image and each image in the training set. The label of the training image with the smallest distance is assigned to the unclassified image.
Advantages:

- Simplicity in concept and implementation.
- Computationally efficient for small datasets.

Limitations:

- Sensitive to noise, outliers, and scaling in the data.
- Struggles to capture complex, nonlinear relationships within the image data.

2. Multi-Layer Perceptron (MLP)

Methodology: The MLP is a type of feedforward artificial neural network. It consists of an input layer (representing the image pixels), one hidden layer, and an output layer (representing the digit classes). The network is trained using backpropagation to adjust the weights between neurons, minimising a loss function.
Advantages:

- Capacity to model complex nonlinear patterns within the data.

Limitations:

- Requires careful architectural design (number of layers, neurons per layer).
- Training can be computationally intensive, especially for larger datasets.
- Susceptible to overfitting if not properly regularised.
- Although a very good algorithm, isn’t able to beat the results from simpler Euclidean distance categoriser.

3. Support Vector Machine (SVM) with SMO

Methodology: SVMs seek to find the optimal hyperplane that separates data points belonging to different classes, while maximising the margin (distance) between the hyperplane and the nearest data points (support vectors). The SMO algorithm is an efficient technique for solving the SVM optimisation problem, particularly suitable for large datasets.
Advantages:

- Robust to outliers and noise.
- Well-suited for high-dimensional data.
- SMO provides computational efficiency for training.

- Kernel functions allow the modelling of nonlinear decision boundaries.

Limitations:

- Very Mathematically Taxing and hard to implement without using a library like LIBSVM
- Parameter tuning (kernel choice, regularisation parameter) is crucial for good performance.
- Can be less intuitive to interpret compared to some simpler methods.

Implementation

1. **Euclidean distance categorisation**
   Euclidean distance categorisation was the first approach as it is a simple algorithm to implement, where the program just memorises the training set and classifies the test set based on the nearest point in the training set. It gives an average accuracy of 98.25625%

```
Categorizing by Euclidean distance
1st Fold
Confusion Matrix:
Label   0       1       2       3       4       5       6       7       8       9
0       268     0       0       0       0       0       1       0       0       0
1       0       288     0       0       0       0       0       1       0       1
2       0       0       280     0       0       0       0       0       0       0
3       0       1       0       269     0       1       0       0       1       3
4       0       1       0       0       285     0       2       1       0       4
5       0       1       0       1       0       276     0       0       2       5
6       0       2       0       0       0       0       272     0       0       0
7       0       0       0       0       0       0       0       292     0       0
8       0       6       0       0       0       0       0       0       260     0
9       0       2       0       1       2       1       0       3       0       277
Digit   Misclassified   Rate
0       1               0.3717%
1       2               0.6897%
2       0               0.0%
3       6               2.1818%
4       8               2.7304%
5       9               3.1579%
6       2               0.7299%
7       0               0.0%
8       6               2.2556%
9       9               3.1469%
Accuracy: 98.4698%
2nd Fold
Confusion Matrix:
Label   0       1       2       3       4       5       6       7       8       9
0       285     0       0       0       0       0       0       0       0       0
1       0       276     1       1       0       0       2       0       1       0
2       0       3       273     0       0       0       0       1       0       0
3       0       0       0       293     0       0       0       1       1       2
4       0       2       0       0       271     0       1       0       1       0
5       0       0       0       0       1       269     0       0       0       3
6       0       0       0       0       0       0       284     0       0       0
7       0       0       0       1       1       0       0       270     1       1
8       0       12      0       2       0       0       0       0       272     2
9       0       0       0       5       3       3       0       0       3       262
Digit   Misclassified   Rate
0       0               0.0%
1       5               1.7794%
2       4               1.444%
3       4               1.3468%
4       4               1.4545%
5       4               1.4652%
6       0               0.0%
7       4               1.4599%
8       16              5.5556%
9       14              5.0725%
Accuracy: 98.0427%
Accuracy for 1st Fold: 98.4698%
Accuracy for 2nd Fold: 98.0427%
Average Accuracy: 98.25625%
```

2. **Multi-Layer Perceptron (MLP)**
   For implementing it, I started from building a simple 3 layer Artificial Neural Network. An input layer consisting of the 64 inputs, a hidden layer and an output layer of 10 neurons each for a digit(0-9).

   It initializes the weights and biases using random numbers from a Gaussian distribution of with a mean of 0 and a standard deviation of 0.1
   To minimize the error I used the backpropagation algorithm to update the weights and biases after each epoch
   A softmax function is used to calculate the probabilities for the output layer

   The initial implementation was bad, and gave about **10%** accuracy which can be achieved from randomly choosing any digit, or classifying all inputs as 0(Since there are 10 digits 1/10= 10%).

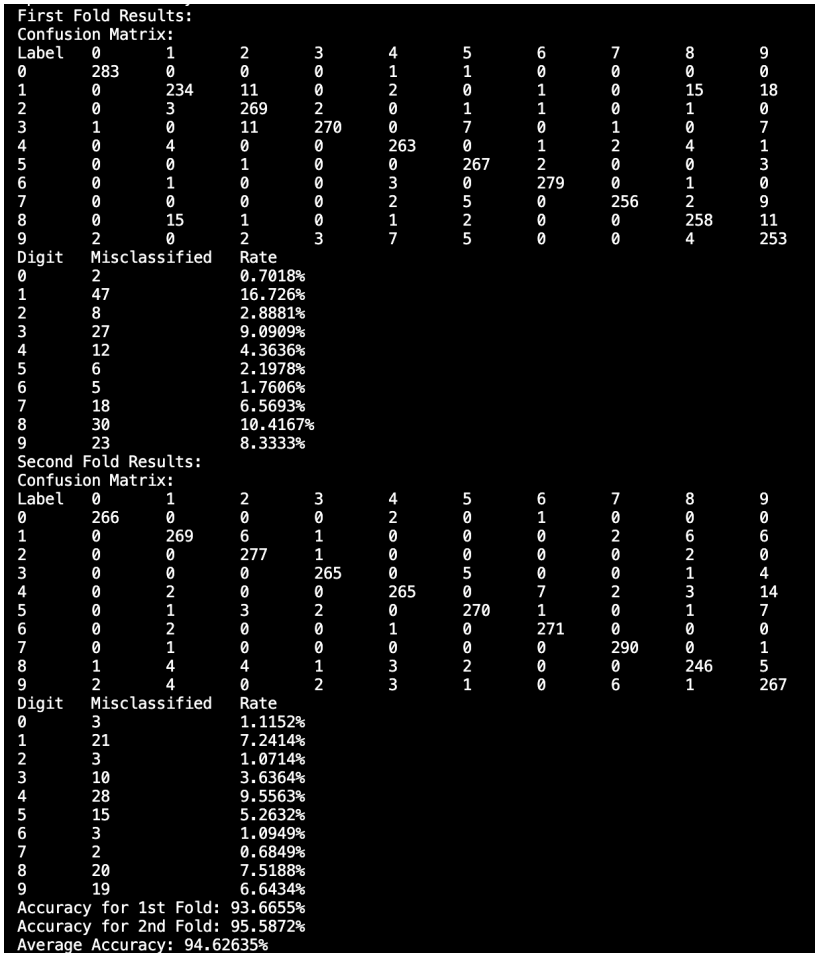
   After thorough research I tried normalising the inputs. As all pixel values are between 0-16, I divided each pixel value by 16. This resulted in **25%** accuracy. An improvement.

   I then implement Relu and sigmoid derivative and tried playing around with the learning rate and number of epochs and started seeing improvements with accuracy reaching as high as **50%-80%.**

   The consistent good results started when I used a tanh activation function for the hidden layer, fixed an issue with the hidden bias calculation. All this resulted in getting a **91%** accuracy, but there was a problem, I could see the accuracy improving with more epochs but then suddenly the accuracy would fall to 10%. The problem was that the algorithm would start classifying all digits as 0 after certain number of epochs.

   To fix this ovefitting to one class issue, I implemented dropout technique to prevent overfitting by randomly setting the activations of some hidden perceptrons to 0, and using a cosine decay learning rate schedule to gradually decrease the learning rate from .2 to 0 with more epochs. I also implemented a cache, to store the weights, biases and Activation values for the best accuracy on training set, and implemented a stop early mechanism, to stop the program if the accuracy ever goes more than 10% below the best accuracy.
   These techniques helped improved the average accuracy to **94.62635%**

   I tried other techniques and played around with the hyperparameters, but they all resulted in lower results.



```
First Fold Results:
Confusion Matrix:
Label   0       1       2       3       4       5       6       7       8       9
0       283     0       0       0       1       1       0       0       0       0
1       0       234     11      0       2       0       1       0       15      18
2       0       3       269     2       0       1       1       0       1       0
3       1       0       11      270     0       7       0       1       0       7
4       0       4       0       0       263     0       1       2       4       1
5       0       0       1       0       0       267     2       0       0       3
6       0       1       0       0       3       0       279     0       1       0
7       0       0       0       0       2       5       0       256     2       9
8       0       15      1       0       1       2       0       0       258     11
9       2       0       2       3       7       5       0       0       4       253
Digit   Misclassified   Rate
0       2               0.7018%
1       47              16.726%
2       8               2.8881%
3       27              9.0909%
4       12              4.3636%
5       6               2.1978%
6       5               1.7606%
7       18              6.5693%
8       30              10.4167%
9       23              8.3333%
Second Fold Results:
Confusion Matrix:
Label   0       1       2       3       4       5       6       7       8       9
0       266     0       0       0       2       0       1       0       0       0
1       0       269     6       1       0       0       0       2       6       6
2       0       0       277     1       0       0       0       0       2       0
3       0       0       0       265     0       5       0       0       1       4
4       0       2       0       0       265     0       7       2       3       14
5       0       1       3       2       0       270     1       0       1       7
6       0       2       0       0       1       0       271     0       0       0
7       0       1       0       0       0       0       0       290     0       1
8       1       4       4       1       3       2       0       0       246     5
9       2       4       0       2       3       1       0       6       1       267
Digit   Misclassified   Rate
0       3               1.1152%
1       21              7.2414%
2       3               1.0714%
3       10              3.6364%
4       28              9.5563%
5       15              5.2632%
6       3               1.0949%
7       2               0.6849%
8       20              7.5188%
9       19              6.6434%
Accuracy for 1st Fold: 93.6655%
Accuracy for 2nd Fold: 95.5872%
Average Accuracy: 94.62635%
```

Using tanh Derivative

I have attached an excel file with this report where you can find the different results, by changing the hyperparameters. It contains all the confusion matrices.

3. **Support Vector Machine (SVM) with SMO**

Support Vector Machines are hard to implement because of the extensive Math that needs to be implemented to do the Sequential minimal optimization. It is seen as a black box in programming as all the modern usecases use the LIBSVM library to get the results. I tried to create my own SVM but failed. I get only 10% accuracy and which means that it doesn't work. I tried to reverse engineer the LIBSVM library to understand what is going on, and minimised the code from 4000+ lines to 1000 and after normalizing the inputs got 98.58% accuracy. I am only attaching it in this report to show that I failed to do it after trying multiple attempts and spending weeks trying to understand it correctly to implement my own.
I am attaching the results that I get using the LIBSVM library below

```
Fold 1:
Confusion Matrix:
Label   0       1       2       3       4       5       6       7       8       9
0       284     0       0       0       0       0       1       0       0       0
1       0       280     2       1       0       0       0       0       4       0
2       0       0       274     0       0       0       0       0       0       0
3       0       0       0       287     0       0       0       1       0       1
4       1       0       0       0       274     0       2       1       0       0
5       0       0       0       3       0       270     0       0       0       1
6       0       0       0       0       1       0       280     0       0       0
7       0       0       1       0       0       0       0       267     0       0
8       0       0       0       4       0       0       1       0       279     2
9       0       1       0       2       0       3       0       5       5       272
Digit   Misclassified   Rate
0       1               0.3509%
1       7               2.439%
2       0               0.0%
3       2               0.692%
4       4               1.4388%
5       4               1.4599%
6       1               0.3559%
7       1               0.3731%
8       7               2.4476%
9       16              5.5556%
Fold 2:
Confusion Matrix:
Label   0       1       2       3       4       5       6       7       8       9
0       265     0       0       0       0       0       0       0       0       0
1       0       287     0       0       0       0       2       0       3       1
2       0       0       280     1       0       0       0       0       0       0
3       1       0       0       271     0       2       0       0       1       2
4       1       0       0       0       284     0       0       0       0       1
5       0       0       0       1       0       280     0       0       0       0
6       1       0       0       0       2       0       272     0       0       0
7       0       1       0       0       1       0       0       292     0       2
8       1       1       0       1       0       0       0       0       262     0
9       0       1       0       1       6       3       0       0       0       280
Digit   Misclassified   Rate
0       0               0.0%
1       6               2.0478%
2       1               0.3559%
3       6               2.1661%
4       2               0.6993%
5       1               0.3559%
6       3               1.0909%
7       4               1.3514%
8       3               1.1321%
9       11              3.7801%
Accuracy for model trained on dataset1 and tested on dataset2: 98.4698%
Accuracy for model trained on dataset2 and tested on dataset1: 98.6833%
Average Accuracy: 98.57655%
```

Average Accuracy: 98.57651245551602% using LIBSVM code taken from this github rCWepository

Conclusion

This project underscored the importance of selecting an algorithm suited to the dataset and the specific problem as there is no free lunch. The Euclidean distance classifier's success demonstrates that even seemingly basic methods can be effective under the right conditions. The MLP's improvement through experimentation highlights the iterative nature of machine learning development. Finally, the challenges with SVM implementation reinforce the value of leveraging well-optimized libraries for complex algorithms.