UMEÅ UNIVERSITET **October 20, 2022**
Computer Science
Assignment rapport

5DV124 - OU2, V1

Fundamentals of Artificial Intelligence

Classfication of Handwritten Digits

Name	Gustaf Söderlund	Per Sondell
CS Email	`et14gsd@cs.umu.se`	`c19psl@cs.umu.se`
Umu Email	`gusa0038@student.umu.se`	`peso0131@student.umu.se`

Teacher
Ola Ringdahl

Contents

1 Introduction

This assignment gives an introduction to using two different tools for classifying handwritten digits given from the MINST data set. The first one is K-Nearest Neighbour and for short *K-NN* which we'll refer to in this report, where we implemented the *fit* and *predict* methods. The other method is using neural network wich we'll refer as *NN* in the report. For both methods we did a hyper-parameter search to find the best model to reach the goal accuracy of 94% for the K-NN and 97,7% for the NN.

2 K-Nearest Neighbour

This section presents the implementation, result and discussion of K-NN.

2.1 Implementation

In order to complete the assignment and classify the digits we had to make two implementations. We were using the python library *sklearn* for the K-NN method. However we had to implement the *fit* and *predict* methods. The other implemention was the hyper-parameter search

2.2 Implementation of methods

The *fit* method were trivial since all it does is setting the x and y training data from the input parameters.

The *predict* implementation follows the generic K-NN algorithm. It calculates the distance between a new point and the rest of the data set. It then sorts the list to get the shortest distances between the new point and the rest. We then only retrieve the k nearest neighbours from the sorted list and return the most common element in that list. In that way we have classified the new point as that returned element.

2.3 Implementation of hyper-parameter search

The hyper-parameter search for K-NN only uses one variable as it search. That is the k value. In our implementation we try different k values ranging from 1 to 10 when we train our model. The reason for choosing our range of k values is because if we only were to use $k = 1$, that is very unstable since we are only considering one neighbour. If we were to go over $k = 10$ we get neighbours that is not related to the new point and would result in a lower accuracy. See Algorithm 1 for pseudo-code for our parameter search.

Algorithm 1 Hyper-Parameter Search

```
Require: $k > 0$
  $k_{max} \leftarrow 10$
  $errorvalues \leftarrow emptylist$
  $k \leftarrow 1$
  while $K_{max} \geq k$ do
    $model \leftarrow train(data, k)$
    $error \leftarrow model.accuracy$
    $errorvalues \leftarrow error$
    $k+ = 1$
  end while
```

2.4 Result

Our K-NN implementation with our hyper-parameter search gave a satisfactory result. The validation score landed at **97.23428571428572%** which implies that our implementation is vaild.

2.5 Discussion

The result of our validation score is a fine value in relation to our goal (94%). We noticed that our hyper-parameter took a while to finish. In the future it would be interesting to compare our implementation with that of *sklearn*.

3 Neural Network

This section presents the implementation, result and discussion of Neural network.

3.1 Implementation

The implementation of the neural network (NN) hyper-parameter search were more advanced than the one of K-NN. In NN we have more variables to search over. We did not search over all possible variables in our implementation. Instead we only need to do a hyper-parameter search for the number of neurons in two hidden layers to achieve the needed result of a accuracy of (97.7%). We choose a range of 200 neurons in the first layer and 100 in the second layer, to 300 neurons in the first layer and 200 in the second divided by 10 iterations. We choose our range of neurons, because it was reasonable size, since if we were to choose a higher range it would take to long of a time to compute. However if we were to choose a smaller range we would get a worse accuracy. See Algorithm 2 for pseudo-code for our parameter search.

Algorithm 2 Hyper-Parameter Search

```
Require: $Numberofneurons > 0$
  $errorvalues \leftarrow emptylist$
  $num\_neurons\_1 \leftarrow 200$
  $num\_neurons\_2 \leftarrow 100$
  while $(num\_neurons\_1 \leq 300) AND ((num\_neurons\_2 \leq 200)$ do
    $hidden\_layer\_sizes \leftarrow (num\_neurons\_1, num\_neurons\_2)$
    $model \leftarrow neural\_network.MLPclassifier(Settings)$
    $model.fit(data, data)$
    $error \leftarrow model.score(data, data)$
    $errorval \leftarrow error$
    $num\_neurons\_1 += 10$
    $num\_neurons\_2 += 10$
  end while
```

3.2 Result

Our NN implementation with our hyper-parameter search gave a satisfactory result. The validation score landed at **97.95428571428572%** which implies that our parameter search is valid. In Figure 1 we can see how our loss evolved during training for our best model found by the hyper-parameter search. We can se that the loss decreases as the iterations of the training increases. This is expected since the model gets better accuracy from more iterations.

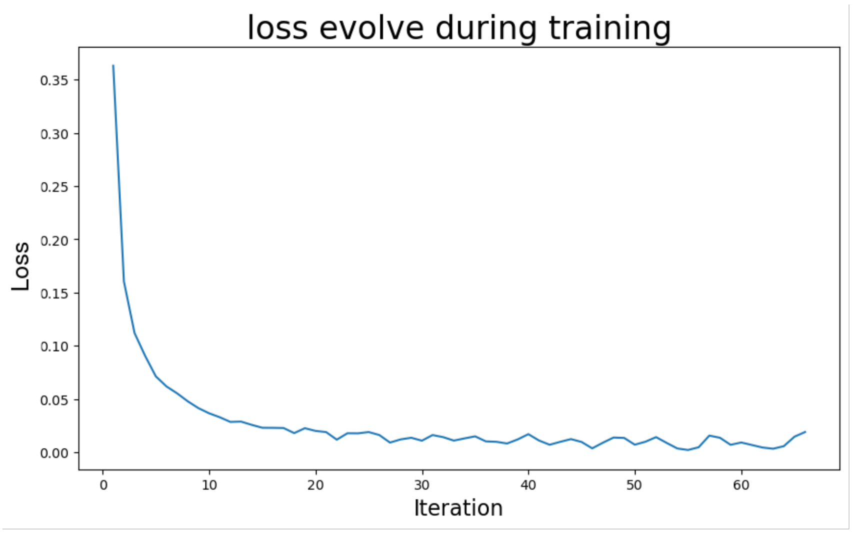

Figure 1: Loss evolve during training for our best model

3.3 Discussion

The main part to discuss here is that we did not search all parameters in our search. We still got a good result, however a more thourgh search, e.x changing number of layers and playing more with the learning rate might have given us a better accuracy. But as a consequence of a more thourgh search would result in longer execution time. The result we got which is above 97.7% is good considering our relative low execution time.

4 Reflection

During the development of the system we worked using the 'Pair-programming' technique to make sure that both of us were actively involved in the development.
We didn't face any huge problems that interrupted our work except the execution time of the whole program. We had to at least do multiple over night runs to test if the program passed the accuracy requirements. As a result of long waiting time between executions, in the case we got "bad" results, lead to longer development time for the whole system to work as intended.
In addition the implementation for the *predict* method took a bit of time. We needed to understand the theory behind the algorithm and how to implement it. The hyper-parameter search was significantly easier to understand and implement.