Machine Learning

Digit Classification with Kernel Perceptron

Lukas Hermans

Università degli Studi di Milano
lukas.hermans@studenti.unimi.it

June 6, 2021

Abstract

- MNIST dataset common benchmark for (new) machine learning algorithms
- here: multiclass kernel perceptron (well-known, less complex than deep neural networks) - three variants applied - comparison between them: who is the best? (state best error rates) - comparsion with other alogrithms, drawbacks

Contents

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work.
I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying.
This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Milan, June 6, 2021

1 Introduction

Recently, the application of computer vision techniques has gained increasing importance in a broad variety of fields, e.g. for the development of autonomeous vehicles, for facial recognition, and for the detection of illnesses [1, 2, 3]. This process is especially driven by the growing field of machine learning.

A common dataset to benchmark and compare machine learning algorithms for image recognition as a subfield of computer vision is the MNIST database of handwritten digits. The dataset contains 60000 training examples and 10000 test examples of handwritten digits from 0 to 9. Originally, the dataset was published by Y. LeCun et al [4]. In this paper, in order to simplify the work with the images, a revision of the original dataset is adopted [5]. In the following, the revised dataset is simply refered to as MNIST dataset.

The objective of the present work is to train a multiclass kernel perceptron algorithm for the classification of handwritten digits using the training examples in the MNIST dataset.

In Section 2, the MNIST dataset is presented in more detail, and some basic definitions and notations are clarified. Then, in Section 3, the theoretical foundation of the multiclass kernel perceptron algorithm is summarized. There, three different approaches to retrieve a multiclass predictor from the multiclass kernel perceptron algorithm are distinguished. Section 4 describes the details of the implementation of the multiclass kernel perceptron algorithm in the present work and gives a list of the applied software. The results of the training of predictors using the multiclass kernel perceptron algorithm on the training part of the MNIST dataset are presented in Section 5. In doing so, the hyperparameters of the multiclass kernel perceptron algorithm are optimized, in order to obtain three multiclass predictors with the lowest error rates on the test part of the MNIST dataset. In Section 6, the main advantages as well as the drawbacks of the application of the multiclass kernel perceptron algorithm for the application on the MNIST dataset are discussed. The retrieved predictors are compared with the test error rates of different learning algorithms that are applied to the MNIST dataset in the literature. Finally, Section 7 summarizes the present work and gives an overview of current and future objects of research regarding the recognition of handwritten digits.

2 Dataset & Definitions

The MNIST dataset contains a total of 70000 images of handwritten digits labeled with a number from 0 to 9. A predifined division of the dataset distinguishes 60000 training examples and 10000 test examples. Each image contains $28 \times 28 = 784$ pixels, where each pixel encodes the brightness in 8 bit. The brightness of each pixel can be described by a variable $x_i \in \{0, 1, \ldots, 255\}$ for $i \in \{1, 2, \ldots, 784\}$. Hence, each image is completely defined by a vector of *features*

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{pmatrix}. \tag{1}$$

The space of all possible images $\mathcal{X} = \{0, 1, \ldots, 255\}^{784}$ is called *feature space*. Each image $\vec{x}$ in the MNIST dataset has a label $y \in \{0, 1, \ldots, 9\}$ that represents the handwritten digit. The *label space* is thus given by $\mathcal{Y} = \{0, 1, \ldots, 9\}$. The labels were added by humans. In the present work, they are regarded as the *true labels* in contrast to labels $\hat{y}$ predicted by an algorithm. So, every *example* in the dataset has the form $(\vec{x}, y)$.

In Figure 1, an examplary ensemble of ten different handwritten digits from 0 to 9 in the dataset is shown. The decompistion of the 70000 examples into training and test examples can be described by two sets S (*training set*) and D (*test set*). The training set S contains all training examples, while the test set D contains all test examples. There is no intersection between the sets S and D, such that the test set contains images that are different from those in the training set. This allows the evaluation of the performance of a classification algorithm that was trained on the training set S.

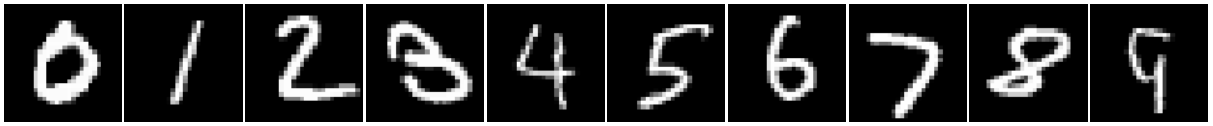
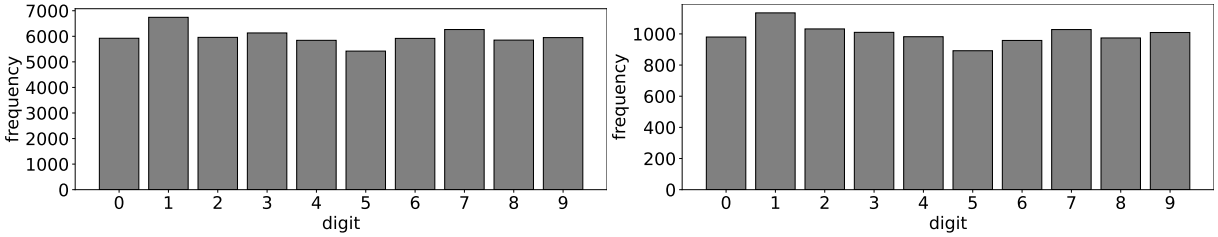


Figure 1: Examplary ensemble of ten different handwritten digits from 0 to 9 in the MNIST dataset. Each image can be described by a feature vector $\vec{x}$, and has a human-decided label y. The images consist of $28 \times 28 = 784$ pixels, where each pixel encodes an integer brightness level between 0 and 255.

Figure 2a and Figure 2b show histograms of the frequency, with which each digit occurs in the training and test set. Both the training and test set are balanced because the different digits appear with a similar frequency. In the following, this is important for the training of classifiers as there is no bias towards a certain digit.



(a) training examples

(b) test examples

Figure 2: The histogram shows, how often each digit occurs in (a) the training set S and (b) the test set D. In both sets, all of the digits appear with a similar frequency, such that the training and test sets are balanced.

3 Theory

With the definitions from Section 2 in mind, the problem in the present paper consists in finding a multiclass predictor $f_S : \mathcal{X} \to \mathcal{Y}, \vec{x} \mapsto \hat{y}$ that predicts a label $\hat{y} \in \mathcal{Y}$ for every possible image $\vec{x} \in \mathcal{X}$ using the multiclass kernel perceptron algorithm. The subscript S of f_S indicates that the classifier is trained on the training set S.

From Multiclass to Binary: One-vs-All Encoding

The basis of the multiclass kernel perceptron algorithm is the reduction of this multiclass classification problem to several binary classification problems - one for each of the ten digits - using the so-called *one-vs-all encoding*. Applying one-vs-all encoding, the initial label $y \in \{0, 1, \ldots, 9\}$ of each example in the training set S is transformed into a binary label $z \in \{-1, 1\}$ by fixing a digit $a \in \{0, 1, \ldots, 9\}$. z is only 1 for those examples for which $y = a$, otherwise z is set to -1. In doing so, a new binary training set $S^{(a)}$ with examples of the form $(\vec{x}, z)$ is generated. As can be seen immediately, the transformation only regards the labels, but leaves the images $\vec{x}$ themselves unchanged [6].

Binary Kernel Perceptron Algorithm

Now, for each of the ten binary training sets $S^{(a)}$ obtained via one-vs-all encoding, the goal is to train a binary classifier that predicts $\hat{z} = 1$ when a given image $\vec{x}$ shows the digit a and $\hat{z} = -1$ when it shows another digit. For this purpose, the *binary kernel perceptron algorithm* - that was first presented in 1984 by Aizerman et al. - can be applied [7], see Algorithm 1.
The binary kernel perceptron algorithm is an online learning algorithm as the training examples are processed sequentially.
In the version of the binary kernel perceptron algorithm that is applied in the present work, the training examples are processed in n_{epochs} epochs. Each epoch is a loop over n_{sample} training examples randomly drawn with replacement from the complete binary training set $S^{(a)}$.

The binary kernel perceptron trains a predictor of the form $h_{S^{(a)}} : \mathcal{X} \to \mathcal{Z}, \vec{x} \mapsto \hat{z}$, where $\mathcal{Z} = \{-1, 1\}$ is the binary label space. The predictor $h_{S^{(a)}}$ has the following form:

$$h_{S^{(a)}} = \text{sgn}\left(\sum_{s:\alpha_s \neq 0} \alpha_s z_s K_p(\vec{x_s}, \vec{x})\right). \tag{2}$$

The part inside the sgn-function will be called $g_{S^{(a)}}$. In Eq. 2, K_p is a polynomial kernel of degree p that has the functional form

$$K_p(\vec{x_i}, \vec{x_j}) = (1 + \vec{x_i} \cdot \vec{x_j})^p,$$

for all $\vec{x_i}, \vec{x_j} \in \mathcal{X}$. Other kernels are of course possible, but the present work focuses on polynomial kernels. The binary classifier in Eq. 2 corresponds to a decision surface of degree p in the feature space $\mathcal{X}$. For $p = 1$, the surface is a hyperplane that is adjusted using the binary training set $S^{(a)}$ in order to make predictions. This corresponds to the original perceptron algorithm [8]. Note, that the predictor $h_{S^{(a)}}$ depends only on a vector $\vec{\alpha}$ that - as can be seen in the algorithm panel below - is updated by the binary kernel perceptron algorithm only if a training example is missclassified. Thus, $\vec{\alpha}$ counts the number of missclassifications during the training process for all examples in the training set $S^{(a)}$. The dependence of $h_{S^{(a)}}$ on the specific $\vec{\alpha}$ is not explicitly denoted in order to maintain readability. However, from the context the particular vector $\vec{\alpha}$ that specifies the predictor should be obvious.

The binary kernel perceptron algorithm stores the $\vec{\alpha}$ for all iterations of the algorithm in a set A and outputs these (technically, A is a multiset as it can and most likely will contain the same $\vec{\alpha}$ more than once, when the prediction for an iteration is correct). As every $\vec{\alpha}$ defines a binary classifier $h_{S^{(a)}}$, the result of the algorithm is a large set A of binary predictors. In the present paper, both the function $h_{S^{(a)}}$ and the corresponding vector $\vec{\alpha}$ are called binary predictor as both contain the same information.

Algorithm 1: Binary Kernel Perceptron

Input : n_{epoch}, n_{sample}, p, $S^{(a)}$
let $A = \emptyset$, $\vec{\alpha} = \vec{0}$;
for *all* $1, \ldots, n_{epoch}$ **do**
  let $T^{(a)} = \emptyset$;
  draw n_{draw} examples from $S^{(a)}$ with replacement and store them in $T^{(a)}$;
  for *all* $t = 1, \ldots, n_{sample}$ **do**
    compute $\hat{z} = \text{sgn}\left(\sum_{s:\alpha_s \neq 0} \alpha_s z_s K_p(\vec{x_s}, \vec{x_t})\right)$;
    if $\hat{z} \neq z_t$ **then**
      $\alpha_i \leftarrow \alpha_i + 1$ where i is the index of example $(\vec{x_t}, z_t)$ in $S^{(a)}$;
    end
    store $\vec{\alpha}$ in A;
  end
end
Output: A

Choice of Binary Predictors

The remaining problem regards the choice of a predictor from the set A of $\vec{\alpha}$-vectors. A straightforward approach is to simply take the last predictor $\vec{\alpha}_{fin}$ after $n_{epoch} \times n_{sample}$ iterations as in the original perceptron algorithm by Rosenblatt. In the present work, also two somewhat more sophisticated approaches are considered.

First, another possible choice is to compute the average $\langle\vec{\alpha}\rangle$ over all predictors in A. This leads to the following vector:

$$\langle\vec{\alpha}\rangle = \frac{1}{n_{epoch} \cdot n_{sample}} \sum_{\vec{\alpha} \in A} \vec{\alpha}.$$

The second approach makes use of the binary training error $\ell_{S^{(a)}}$ of a particular predictor $\vec{\alpha}$ that is defined

as follows:

$$\hat{\ell}_{S^{(a)}}(\vec{\alpha}) = \frac{1}{|S|} \sum_{(\vec{x},z)\in S^{(a)}} \ell(h_{S^{(a)}}(\vec{x}), z).$$

Here, ℓ is the binary *zero-one loss*

$$\ell(\hat{z}, z) = \mathbb{1}(\hat{z} = z),$$

where $\mathbb{1}$ is the indicator function. The binary training error is the error rate of a particular predictor $\vec{\alpha}$ on the training set $S^{(a)}$. Now, the second predictor is extracted by computing the $\vec{\alpha}_{min}$ in A, for which the training error $\ell_{S^{(a)}}$ is minimized:

$$\vec{\alpha}_{min} = \underset{\vec{\alpha}\in A}{\operatorname{argmin}}\, \hat{\ell}(\vec{\alpha}).$$

In the present work, the predictors are refered to as final predictor $\vec{\alpha}_{fin}$, average predictor $\langle\vec{\alpha}\rangle$, and minimizing predictor $\vec{\alpha}_{min}$.

Multiclass Kernel Perceptron Algorithm

The binary kernel perceptron algorithm just presented can be applied for all of the ten digits $a \in \{0, 1, \ldots, 9\}$, using the corresponding binary training sets $S^{(a)}$. If a specific $\vec{\alpha}$-type is chosen, the result are ten binary predictors $h_{S^{(a)}}$ that predict $\hat{z} = 1$ when a given image $\vec{x}$ shows the handwritten digit a and $\hat{z} = -1$ when not.

Finally, the multiclass classifier f_S that predicts a digit from a given image $\vec{x}$ is the combination of all of the ten binary classifiers:

$$f_S(\vec{x}) = \underset{a\in\{0,1,\ldots,9\}}{\operatorname{argmax}}\, g_{S^{(a)}}(\vec{x}).$$

Note, that the functions $g_{S^{(a)}}$ inside the sgn-function in Eq. 2 enter the computation of f_S, and not the binary predictors $g_{S^{(a)}}$ themselves. f_S predicts the digit of the binary classifier that is most secure that the image $\vec{x}$ contains a certain digit by chosing the largest $g_{S^{(a)}} \in \mathbb{R}$. The algorithm is called *multiclass kernel perceptron algorithm* as it makes use of the kernelized perceptron algorithm and expands it to predictions beyond the binary case[6].

Evaluation: Training and Test Error Rate

To evaluate the performance of a multiclass predictor f_S, two error rates have to be distinguished.
One the one hand, there is the *training error rate* that is given by:

$$\ell_S = \frac{1}{|S|} \sum_{(\vec{x},y)\in S} \ell(f_S(\vec{x}), y). \tag{3}$$

Here, ℓ is the binary zero-one loss

$$\ell(f_S(\vec{x}) = \mathbb{1}(\hat{y} = y).$$

The training error rate is the relative number of images that the multiclass predictor f_S (that of course depends on the binary predictors choosen previously) missclassifies among all the images $\vec{x}$ in the training set S.
On the other hand, the *test error rate*

$$\ell_D = \frac{1}{|D|} \sum_{(\vec{x},y)\in D} \ell(f_S(\vec{x}), y) \tag{4}$$

is the relative number of missclassified images $\vec{x}$ in the test set D. Note, that these images do not enter the training of f_S.

4 Implementation & Software

For the present work, the multiclass kernel perceptron algorithm is implemented in the programming language Python (version 3.8.6). On top of vanilla Python, the following software is used:

- pip (version 21.1.2): managing of Python modules
- matplotlib (version 3.4.2) & seaborn (version 0.11.1): modules for visualization & plots
- pandas (version 1.2.4): module for loading data
- numpy (version 1.20.3): module for array computing (such as dot products)

All of the Python code used in this paper can be found in the following GitHub repository: `https://github.com/lukher98/digit-classification`. To ensure the reproducability of the numerical results, seeds for the random number generator of the numpy module are set whenever random numbers enter the computation. In contrast to Algorithm 1, the predictors of each iteration of the binary kernel perceptron algorithm are not collected but the average and minimizing predictors are computed on the fly, in order to obtain higher efficiency and less memory usage. In addition to the Python code, the repository contains the whole MNIST dataset (using Git Large File Storage), as well as the LaTeX code of this paper.
The Python code is executed on a Linux machine, but it should also work on a Mac machine. For Windows, the user might have to change some paths inside the code.

5 Results

6 Discussion

a

7 Conclusion & Outlook

a

References

[1] Arien P. Sligar. "Machine Learning-Based Radar Perception for Autonomous Vehicles Using Full Physics Simulation." In: *IEEE Access* 8 (2020), pp. 51470–51476. DOI: 10.1109/ACCESS.2020.2977922.

[2] Deepak Gupta Anvita Saxena Ashish Khanna. "Emotion Recognition and Detection Methods: A Comprehensive Survey." In: *Journal of Artificial Intelligence and Systems* 2 (2020), 53–79. DOI: https://doi.org/10.1038/s41746-020-00376-2.

[3] A. Esteva, K. Chou, and S. Yeung. "Deep learning-enabled medical computer vision." In: *npj Digit. Med.* 4.5 (2021). DOI: https://doi.org/10.1038/s41746-020-00376-2.

[4] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. *The MNIST database of handwritten digits*. 2021. URL: http://yann.lecun.com/exdb/mnist/ (visited on 06/02/2021).

[5] Dariel Dato-on (User) on Kaggle. *MNIST in CSV*. 2021. URL: https://www.kaggle.com/oddrationale/mnist-in-csv (visited on 06/02/2021).

[6] Jianhua Xu and Xuegong Zhang. "A multiclass kernel perceptron algorithm." In: *2005 International Conference on Neural Networks and Brain*. Vol. 2. 2005, pp. 717–721. DOI: 10.1109/ICNNB.2005.1614728.

[7] M. A. Aizerman, E. A. Braverman, and L. Rozonoer. "Theoretical foundations of the potential function method in pattern recognition learning." In: *Avtomat. i Telemekh.* 25.6 (1967), pp. 917–936.

[8] Rosenblatt. "The Perceptron: A Perceiving and Recognizing Automaton." In: *Conrell Aeronautical Laboratory* 85-460-1 (1957).