Machine Learning

Digit Classification with Kernel Perceptron

Lukas Hermans

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

June 2, 2021

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

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 2, 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 test 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 two different approaches of the multiclass kernel perceptron algorithm using the training examples in the MNIST database. Their performance for different choices of hyperparameters is compared with the help of the test examples. The result are two implementations of the kernel perceptron for the recognition of handwritten digits, and their corresponding test errors. In the last step, their performance is compared with other approaches that can be found in the literature.
- structure of paper

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*

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

Each image $\vec{x}$ in the dataset is labeled with $y \in \{0, 1, \ldots, 9\}$ that represents the handwritten digit displayed in the figure. The labels were decided 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.

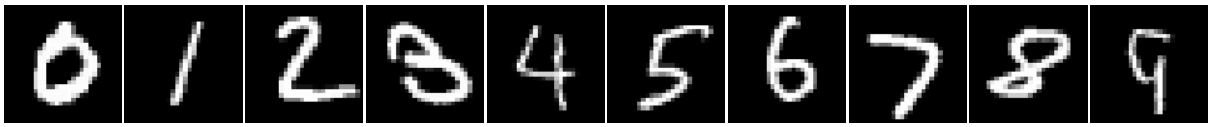
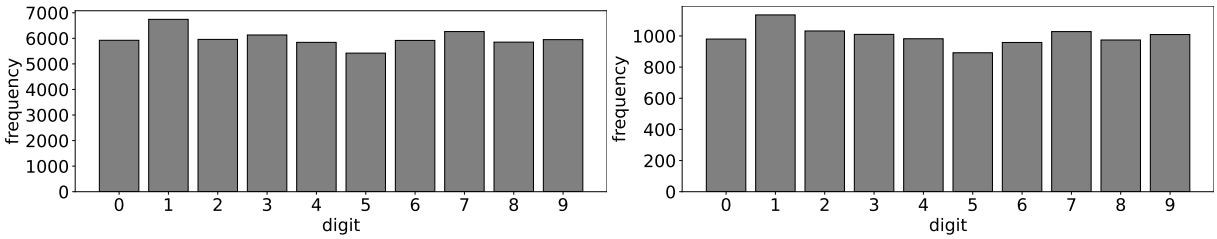


Figure 1: Examplary ensemble of ten different handwritten digits from 0 to 9 in the MNIST dataset. Each image can be described by a feature vector $\vec{x}$, and has a human-decided label y. The images consist of $28 \times 28 = 784$ pixels, where each pixel encodes an integer brightness level between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

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 $(\vec{x}, y)$, while the test set contains all test sets. 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 2a and Figure 2b show a histogram of the frequency, with which each digit occurs in the training and test examples. 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 for 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 above in mind, the problem in the present paper consists in finding a multiclass predictor $f_S : U \to \{0, 1, \ldots, 9\}, \vec{x} \mapsto \hat{y}$. The subscript S of f_S indicates that the classifier is trained on the training set S. In principal, U contains all possible image vectors $\vec{x}$ of the form described above. In other words, for all possible images $\vec{x}$ the classifier outputs a label $\hat{y}$. However, first, the images should be digits from 0 to 9 (though they can be blurry, for instance) as the classifier is not trained to identify other objects, such as letters. Second, here, the predicted labels $\hat{y}$ for the images in the test set D are of particular interest because they are used to evaluate the performance of a trained classifier on new images by comparing them to the true labels y. The images in the test set D are "new" in the sense, that they are not included in the training set S that is used to train the classifier f_S.

From the multiclass classification problem stated above, ten binary classification problems can be derived using *one-vs-all encoding*. For the application of one-vs-all encoding, the initial label $y \in \{0, 1, \ldots, 9\}$ is transformed into a binary label $z \in \{-1, 1\}$ by fixing a digit $a \in \{0, 1, \ldots, 9\}$. By setting $z = 1$ if $y = a$, and $z = -1$ else, the multiclass training set S is transformed into a binary training set $S^{(a)}$, where a refers to the fixed digit in the one-vs-all encoding. In doing so, the images themselves remain unchanged, only the labels are transformed from multiclass to binary.

For the ten transformed training sets $S^{(a)}$, the goal is to train a binary classifier $h_{S^{(a)}} : U \to \{-1, 1\}, \vec{x} \mapsto \hat{z}$. Here, $\hat{z}$ is the predicted binary label for an arbitrary image $\vec{x}$. A well-known procedure to train $h_{S^{(a)}}$ using $S^{(a)}$ is the kernel perceptron algorithm that was first presented by Aizerman et al in 1964 [6]. In the present work, the algorithm is used in the following form:

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

The algorithm has three inputs. n_{epochs} describes the number of epochs for which the binary predictor should be trained. Each epoch considers a subset $T^{(a)} \subset S^{(a)}$ of the training examples that is generated by drawing examples at random with replacement from the complete training set $S^{(a)}$. This - instead of using the complete training set in one epoch - is basically done to insure computational feasability. In

general, the binary predictor in the kernel perceptron algorithm at each step has the form

$$h_{S^{(a)}} = \text{sgn}(g_{S^{(a)}}),$$

with

$$g_{S^{(a)}} = \sum_{s:\alpha_s \neq 0} \alpha_s z_s K_p(\vec{x_s}, \vec{x_t}),$$

and is completely described by a vector $\vec{\alpha}$. The binary kernel perceptron algorithm updates $\vec{\alpha}$, whenever its prediction $\hat{z}$ errs on a training example. Thus, $\vec{\alpha}$ counts how often each training example in $S^{(\alpha)}$ is missclassified. The output of the algorithm is the set A that contains all vectors $\vec{\alpha}$ that were produced by the binary kernel perceptron algorithm during the training phase.

After the application of the binary kernel perceptron algorithm, a predictor inside of all predictors A has to be choosen. One could simply take the predictor at the end of the algorithm. However, there are more refined choices. Among those are the average predictor $\langle\vec{\alpha}\rangle$ over all predictors in A as well as the predictor $\vec{\alpha}_{min}$ that minimizes the training error

$$\hat{\ell}(\vec{\alpha}) = \frac{1}{n_{epoch} n_{sample}} \sum_{(\vec{x},z)\in S^{(a)}} \ell(h_{S^{(a)}}(\vec{x}), z),$$

where ℓ is the zero-one loss

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

where $\mathbb{1}$ is the indicator function. The binary kernel perceptron algorithm can be applied for all ten digits. define kernel -> connection to classical perceptron algorithm expansion to multiclass prediction online algorithm

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): module for visualization & plots
- 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, a seed for the random number generator of the numpy module is set at the beginning of the Python code. In addition to the Python code, the repository contains the whole MNIST dataset (using Git Large File Storage), the LaTeX code of this paper, as well as all of the cited papers.
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] 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.