Optimization and Data Analytics
A research project on image classification

Theo Morales [201703024]

***Abstract*—This research paper treats different optimization algorithms applied on the classical image classification problem. The algorithms are implemented in C++, using the Eigen library for linear algebra, and applied on two different datasets: the MNIST dataset of handwritten digits (grayscale images), and the ORL dataset of faces (grayscale images). The results of the different algorithms are visualized by applying the Principal Component Analysis and plotting the 2D data. Throughout this paper, the algorithms in question will be described, then compared based on their execution times and their success rates.**

***Keywords*—Optimization, machine learning, image classification.**

I. Introduction

The image classification problem is a very common case of study when it comes to machine learning and data analytics. There is a substential amount of different algorithms that can be used to treat this problem, however, the lack of computational power has slowed down the study of those. Nowadays, this is no longer a problem, as computers have become significantly faster than when these algorithms were thought and designed, and it is now a lot easier to compare them in depth. Machine learning is becoming a trend, and as it can be utilized on affordable hardware, it can be applied to a large variety of problems.

This study focuses on classification of two sets of images: the MNIST dataset, which is a collection of 70000 images representing handwritten digits from 0 to 9, in grayscale, and the ORL dataset, containing 400 grayscale facial images, representing a total of 40 different individuals.

The following optimization algorithms will be applied on the said data samples:

- Nearest Centroid Classifier
- Nearest Sub-class Centroid Classifier
- Nearest Neighbour Classifier
- Perceptron trained using Backpropagation
- Perceptron trained using Mean Square Error

In order to establish a benchmark for these classifiers, their computation time and their accuracy will be compared, and they will be applied on the two datasets reduced to two dimensions after applying the Principal Component Analysis (PCA), which will also allow for the plotting of the results.

II. Methods

The methods, or classification algorithms in this case, will be described and explained in this section.

A. Nearest Centroid Classifier

This first algorithm is rudimentary and very simple to understand. Firstly, during the training phase, a mean vector is calculated for each class, by averaging all the training samples of the given class. Then, in the classification phase, each test sample is classified by calculating its euclidean distance to each mean vector: the lowest distance indicates the nearest class for the test element. The euclidean distance formula is:

$$d_E(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i - \mathbf{x}_j{}_2 = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^\top (\mathbf{x}_i - \mathbf{x}_j)} \quad (1)$$

B. Nearest Sub-class Centroid Classifier

This classifier is basically an enhanced version of the Nearest Centroid Classifier, where the K-means algorithm has been applied in the training phase. This results in a clustered class, which holds K mean vectors corresponding to K sub-classes (clusters) instead of just one mean class vector. The number of sub-classes K is, theoretically, giving better and more accurate results for a higher value, as it yields more choices for the testing phase.

The classification of elements is done the same way as for the Nearest Centroid, however each test sample is compared to each mean sub-class vector of each class. The number of sub-classes is to be fine-tuned empirically, depending on the desired precision/speed ratio.

C. Nearest Neighbour Classifier

The last euclidean distance-based classifier, Nearest Neighbour, is the most time-consuming and resource-demanding algorithm, but in theory one of the most accurates. This classification method demands no training phase, as the classification process is very straight-forward. To classify a test sample, its distance to each individual training sample is computed using the euclidean distance, and the lowest is then used to give it the class of the training element, accordingly. This algorithm doesn't introduce any interesting aspects in terms of classification, but is still interesting to use for comparison ends.

D. Perceptron trained using Backpropagation

A perceptron is a linear discriminant method of classification, that uses a discriminant function to determine wether a sample is belonging to a given class or not: it is a binary classifier. As one perceptron can only be used for one given class, several of them will be used to form a neural network, that will them be fed input data such as the aforementioned ORL image vectors, in order to train each perceptron to accept

its given class and to reject every other. The discriminant function can be represented as:

$$g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + \omega_0 \tag{2}$$

with $\mathbf{w}$ being the weights vector, and ω_0 the bias, together constituting the discriminant. When the result of this function is above 0, the given $\mathbf{w}$ vector can be classified as belonging to the perceptron's class, otherwise it doesn't.

During the training phase, the weight vectors have to be computed in the optimal way, so that they will be able to classify an input with the best success rate. In order to do so, a perceptron is fed with input vectors belonging to all the classes of the training set, and the following criterion function is used to update the weights vector of one perceptron:

$$\mathcal{J}_p(\mathbf{w}) = \sum_{x_i \in \mathcal{X}} -l_i \tilde{\mathbf{w}}^\top \tilde{\mathbf{x}}_i \tag{3}$$

with χ being the set of misclassified samples for the perceptron, l_i being the binary label of the misclassified sample, $\tilde{\mathbf{w}}$ corresponding to the augmented weights vector, and $\tilde{\mathbf{x}}_i$ the augmented misclassified sample vector. This optimization problem can be solved using the gradient descent method as follow:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \sum_{x_i \in \chi} -l_i \mathbf{x}_i \tag{4}$$

with η being the learning rate, which defines the step size of the gradient descent: it is a hyperparameter of the algorithm.

Finally, during the classification phase, the aforementioned function $g(x)$ is applied to each input vector, and the binary result is used to determine which class the input sample belongs to.

E. Perceptron trained using Mean Square Error

This last classifier is essentialy similar to the previous perceptron, except that it treats the criterion function as a quadratic problem and directly tries to solve it. This training method for the perceptron yields better results for cases where the data classes are not linearly separable. The criterion function can be written as:

$$\mathcal{J}_p(\mathbf{w}) = ||\mathbf{X}^\top \mathbf{w} - \mathbf{b}||_2^2 \tag{5}$$

with $\mathbf{X}$ the matrix of training sample vectors, and $\mathbf{b}$ containing the output labels for the training samples. The actual solution of this function is the following:

$$\mathbf{X}^\dagger = \lim_{\epsilon \to 0} (\mathbf{X}\mathbf{X}^\top + \epsilon \mathbf{I})^{-1} \mathbf{X} \tag{6}$$

In order to make sure that the result of the matrices product is invertible, a neglectibly small value is added to each element of the matrix.

F. Principal Component Analysis

Dimensional reduction of the datasets can be useful for proving that a high amount of dimensions will result in an equaly high classification accuracy, by comparing the original data with a 2D version of it. Moreover, it makes the data plottable on an orthogonal space, and easy to visualize by emphasizing on the variation, which shows the strong patterns. In order to obtain the same data reduced to two dimensions, the Principal Component Analysis is applied to linearly transform it while keeping the most informative bits. The PCA transformation can be expressed as:

$$y_i = \mathbf{W}^i (7)$$

with $\mathbf{W}$ being a matrix of ordered eigenvectors of the original matrix, that can be used partially to reduce the number of dimensions equal to the number of eigenvectors used, and with more or less variation depending on the order of the chosen latter.

Fig. 1. MNIST dataset after applying PCA

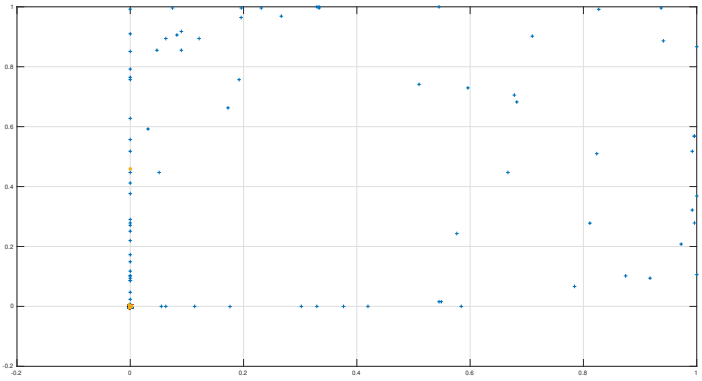
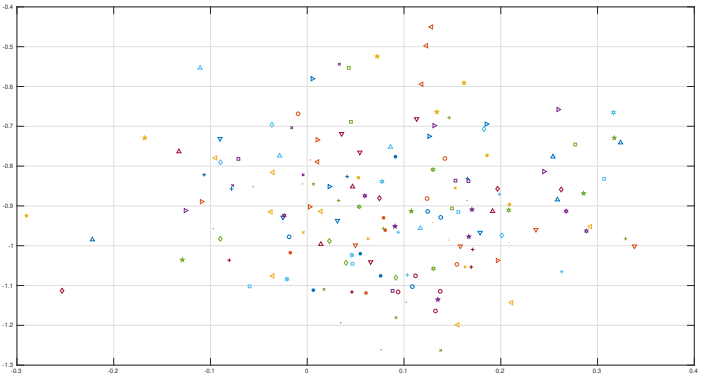


Fig. 2. ORL Data set after applying PCA



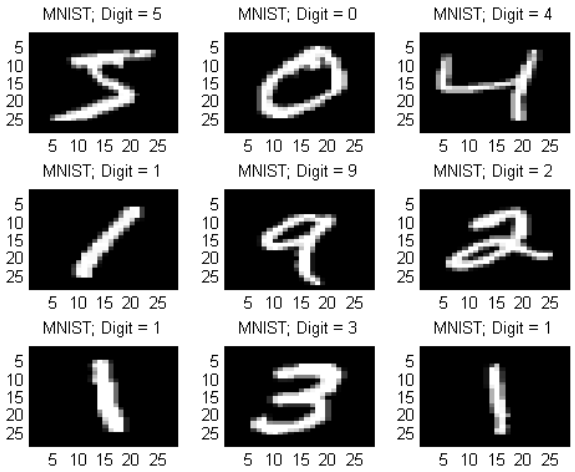
III. Results

As stated previously, each algorithm has been applied to the MNIST and ORL datasets. It is important to know that the MNIST dataset comes already split in training data (60000 samples) and test data (10000 samples), whereas the ORL

dataset had to be manually and randomly split in 70% training samples and 30% test samples.

The MNIST dataset is composed of 70000 pictures of handwritten digits in grayscale, from 0 to 9, and with an image resolution of 28x28.

Fig. 3. MNIST dataset sample



The ORL dataset consists of 400 facial pictures, for it contains 10 pictures for each of the 40 persons, thus giving 40 classes.

Fig. 4. ORL dataset sample

Each algorithm has been compared by execution time, and success rate (or accuracy). It can be seen that changing the hyperparameters of one can drastically improve its success rate, as the Nearest Sub-class Centroid Classifier shows for instance, with the given hyperparameters {2,3,5} corresponding to the number of sub-classes. The learning rate for the backpropagation algorithm has been set to 0.1, after experimentation and fine-tuning.

Fig. 5. ORL success rates

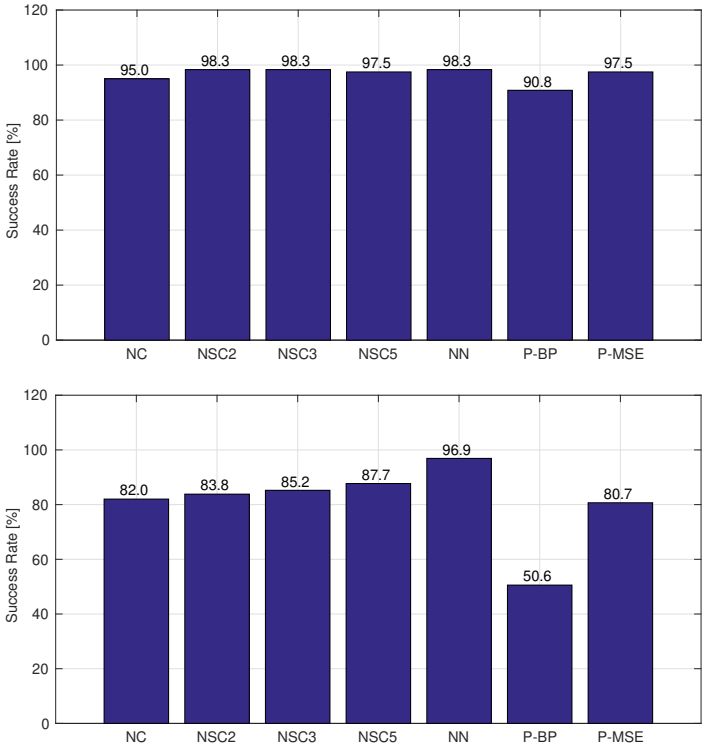


Fig. 6. MNIST success rates

The bar diagrams shown in figure 5 and 6 clearly demonstrate that most of the algorithms are more efficient and give slightly better results on the ORL dataset. It is also clear that the distance-based algorithms perform better than the perceptrons, and it is especially true for the Nearest Neighbour classifier, even though it is the slowest algorithm as shown in the following bar diagrams.

As it can be seen on figure 7 and 8 of the conclusion section, the execution time of the Nearest Neighbour is the highest, as stated above, while the other distance-based classifiers globally remain the fastest to compute. In general, each algorithm is slightly slower to execute on the MNIST dataset, compared to the ORL dataset, and this is due to the fact that it is substantially larger than the latter.

The follong figures show the tremendous accuracy and processing time reduction after applying PCA to the datasets, thus reducing their dimensions to only 2. It goes without saying that applying PCA only to reduce the data's dimensions to 2D doesn't serve any interest other than constating the huge loss of accuracy. However, as the reduction of dimensions greatly reduces the execution time, it can be interesting to experiment with the amount of dimensions that the data can be reduced to, in order to obtain acceptable success rates while profiting from the great time gain.

IV. Conclusion

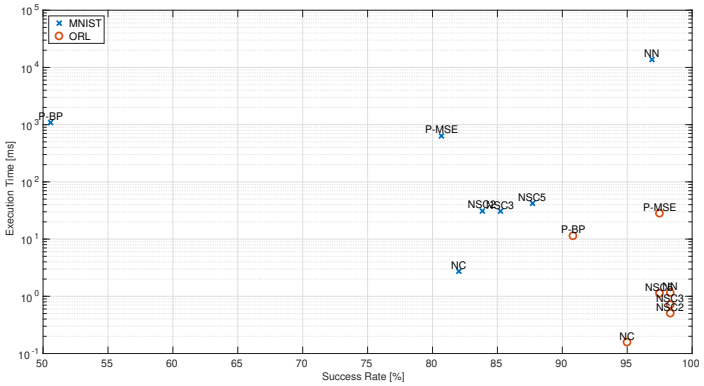
Finally, the algorithms' performances for both datasets can be visualized on the following graphs:

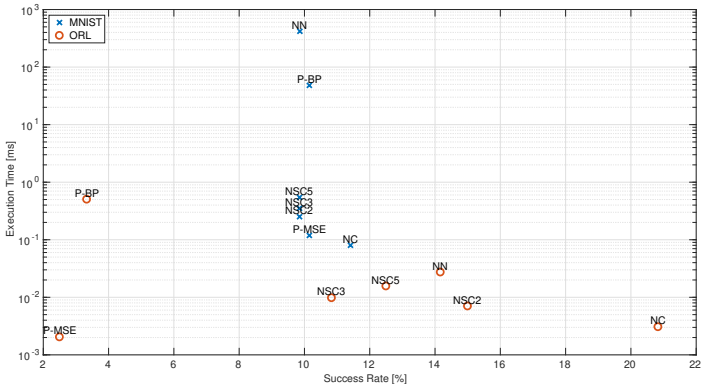
Fig. 7. Comparison of performance

Fig. 8. Comparison of performance after applying PCA

These figures represent the execution time and the success rates of each algorithm for each dataset (see legend). Of course, the higher accuracy the better, but a low execution time is very important as well, for real life applications.

If more time was given for this project, it would have been a good thing to compare several iterations of the same algorithms for the ORL dataset for example, as the training samples and test samples are split randomly, thus resulting in very different results each time. Furthermore, this project was implemented





in C++, and therefore took a lot more time than expected to be implemented, that is why the optimization has been set aside in order to obtain results quickly, and it would add a good value to work on the optimization, for example to make more use of threads (as it can be seen in the Nearest Neighbour Centroid) in every algorithm.