Learning a Rule

Neural Networks and Computational Intelligence - Practical Assignment II

Samuel Giacomelli
Student Number: S3546330
s.giacomelli@student.rug.nl

Davide Pedranz
Student Number: S3543757
d.pedranz@student.rug.nl

***Abstract*—Perceptrons are devices used to solve binary classification problems. Multiple algorithms were proposed in the literature to train a perceptron. In this assignment, we implement the MinOver training algorithm and measure its generalization error. We run simulations for datasets of different sizes and with different grades of noise. We compare the performances of the MinOver algorithm with the Rosenblatt one implemented in the last assignment.**

I. Introduction

Perceptrons are devices used to solve binary classification problems. A perceptron tries to learn a hyperplane that divides the training examples in 2 groups, depending on their label. In general, if the dataset is linear separable, there is an infinite number of hyperplanes that perfectly separates the examples based on their labels. The MinOver algorithm [1] tries to find the hyperplane with the maximum stability, i.e. maximize the distance of the closest examples from the hyperplane.

In this assignment, we implement the MinOver training algorithm and measured its generalization error on a linearly separable dataset as a function of the rate between the number of examples and their dimension. We also compare its performances with the Rosenblatt algorithm [2] implemented in the previous assignment. Finally, we introduce different degrees of noise in the datasets and observe the performances of both algorithms.

Section II introduces the MinOver training algorithm. Section III describes the implementation of the various experiments. Section IV discusses the obtained results. Finally, Section V summarizes the most interesting results.

II. Theory

The aim of the MinOver algorithm is to find the perceptron of maximum stability, i.e. to maximize the margin between the weight vector $\mathbf{w}$ and the closest input example ξ^μ. Formally, the stability κ^μ is defined in Equation (1):

$$\kappa^\mu = \frac{\mathsf{w} \cdot \xi^\mu S_R^\mu}{|\mathsf{w}|} \tag{1}$$

During the training phase, the MinOver algorithm looks for the input vector with the lowest stability at the current time t (Equation (2)) and performs a Hebbian update using its value (Equation (3)):

$$\kappa^{\mu(t)} = \min_\nu \left\{ \kappa^{\nu(t)} = \frac{\mathsf{w}(t) \cdot \xi^\nu S_R^\nu}{|\mathsf{w}(t)|} \right\} \tag{2}$$

$$\mathsf{w}(t+1) = \mathsf{w}(t) + \frac{1}{N} \xi^{\mu(t)} S_R^{\mu(t)} \tag{3}$$

The procedure is repeated for a certain number of epochs n_{max} or until convergence, i.e. when $\mathsf{w}(t+1) = \mathsf{w}(t)$.

Like in the Rosenblatt perceptron, the output $S^\mu \in \{+1, -1\}$ for a given examples is computed by evaluating the sign of the dot product between the example ξ^μ and the weights vector w, as shown in Equation (4):

$$S^\mu = sign(\mathsf{w} \cdot \xi^\mu) \tag{4}$$

III. Implementation

A. Dataset Generation

We use an artificially generated dataset with P examples and N features with normally distributed values. The labels are not chosen randomly, but determined using a teacher weights vector $\mathbf{w}^*$ as $S^\mu = sign(\mathsf{w}^* \cdot \xi^\mu)$. The obtained dataset is by construction linearly separable for $\mathbf{w} = \mathbf{w}^*$.

We chose $\mathbf{w}^*$ such that its norm is constant, for example $||\mathbf{w}^*||^2 = N$. Since we initialize $\mathbf{w} = \overrightarrow{0}$, we have that the distance between the student and the teacher perceptron is constant:

$$\mathbf{w} = \overrightarrow{0} \in \mathbb{R}^N$$

$$distance(\mathbf{w}, \mathbf{w}^*) = ||\mathbf{w} - \mathbf{w}^*|| = ||\mathbf{w}^*|| = \sqrt{N}$$

For simplicity, we chose $\mathbf{w}^* = \overrightarrow{1} \in \mathbb{R}^N$.

B. Perceptron Training

To implement the MinOver algorithm we started from the code written for the Rosenblatt perceptron in the last assignment. As for the Rosenblatt perceptron, we initialize the weights vector of the student perceptron with zeros. Then, we run the training procedure for number of epochs (where $epochs = n_{max} * P$). At each epoch, we compute the stability of each example in the dataset (for performance reasons and to avoid the division by the norm of $\mathbf{w}$, we parallelize the computation as matrix operations instead of iterating over the elements in the dataset) and update the weights using the example with the minimum stability. We terminate the training if the last update of the weights (normalized by the current norm) is smaller than a certain threshold (we chose $min_{update} = 0.001$), since the update is in practice never equal to zero for numerical problems.

The algorithm differs from the Rosenblatt's one since it performs an update at each step, whether the perceptron is already giving the correct classification or not (it could not have reached the optimum stability).

C. Experiments

For a fixed value of P and N, n_D independent datasets are generated. A new perceptron is trained on each dataset for at most $epochs = n_{max} * P$ epochs. For each dataset, we compute the generalization error $\epsilon_g(t_{max})$ as:

$$\epsilon_g(t_{max}) = \frac{1}{\pi} \arccos\left(\frac{\mathbf{w}(t_{max}) \cdot \mathbf{w}^*}{||\mathbf{w}(t_{max})|| \cdot ||\mathbf{w}^*||}\right) \tag{5}$$

$\epsilon_g(t_{max})$ measures the distance of the student from the teacher perceptron and gives an indication on the accuracy for the classification of previously unseen data.

Multiple experiments are run for different values of P and N in order to compute $\epsilon_g(t_{max})$ as a function of $\alpha = P/N$. In other words, we study the accuracy of the perceptron in classifying previously unseen inputs as a function of the rate between the number of examples and the dimension of the input.

D. Noisy dataset

In real classification problems the data contains some noise. It is therefore interesting to investigate the performances of the MinOver and Rosenblatt training algorithms on noisy data. The implementation consists in a small change of the function that generates the dataset (Section A-C). We generate P random values and then define a noise vector which contains a label -1 if the the randomly generated number is smaller than λ and $+1$ otherwise. The real labels are then multiplied element-wise with the noise vector.

IV. Evaluation

All the results presented in this section are obtained with the following parameters:

- $n_D = 50$
- $n_{max} = 250$
- $N = 200$

A. MinOver

Figure 1 shows the generalization error of the perceptron trained with the MinOver algorithm on a linearly separable dataset (with no noise, $\lambda = 0$). Since the number of dimensions of the dataset N is fixed, $\alpha = P/N$ is proportional to the number of examples P used for the training. The generalization error decreases for higher values of α, i.e. when the number of examples increases. In other words, it seems to be the case:

$$\lim_{\alpha \to \infty} \mathbf{w} = \mathbf{w}^*$$

The examples are distributed in all the space and the boundary between the classes is fixed and determined by the teacher vector $\mathbf{w}^*$. Intuitively, for a higher number of examples P, the maximum margin between the 2 classes decreases, since it is more likely that some example is really close to the boundary hyperplane determined by $\mathbf{w}^*$. Since the MinOver algorithm tries to find some separation hyperplane, it is easier to get closer to the teacher vector $\mathbf{w}^*$ if the maximum possible margin is smaller.

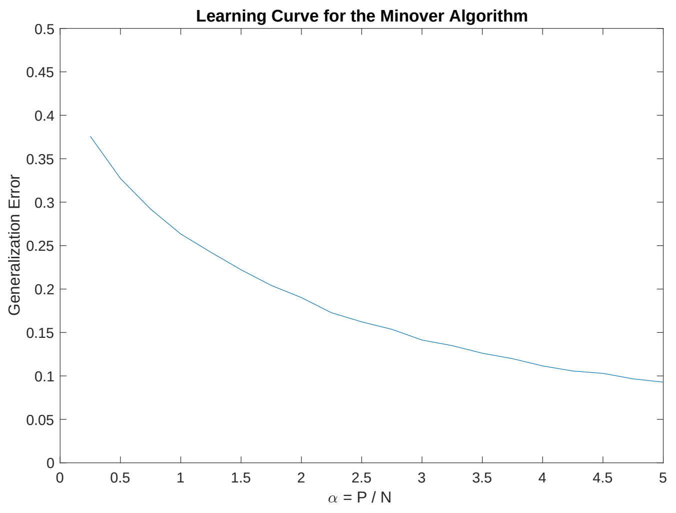

Figure 1. Generalization error of a perceptron trained with the MinOver algorithm on a linearly separable dataset.

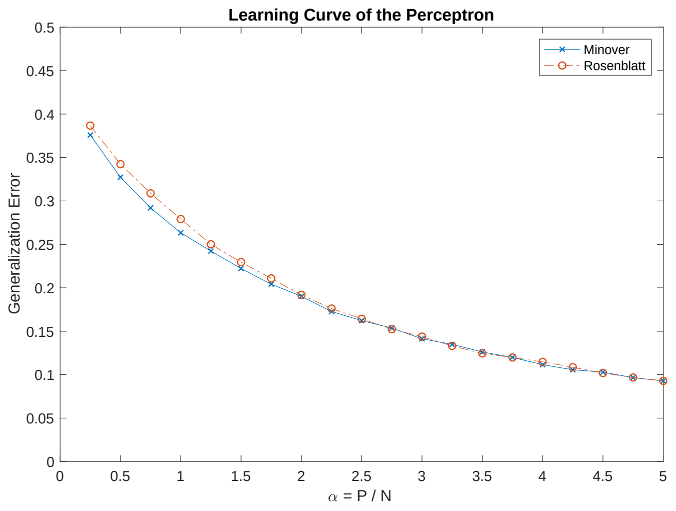

Figure 2. Generalization error for the MinOver and the Rosenblatt algorithms on a linearly separable dataset.

B. MinOver vs Rosenblatt

Figure 2 shows the generalization error of the perceptron trained with the MinOver and the Rosenblatt algorithms on a linearly separable dataset (without noise, $\lambda = 0$). For both algorithms the generalization error decreases with higher numbers of examples for the training (see the previous section). In absence of noise, the performances of the 2 algorithms are quite close, with MinOver performing slightly better than Rosenblatt for low values of α.

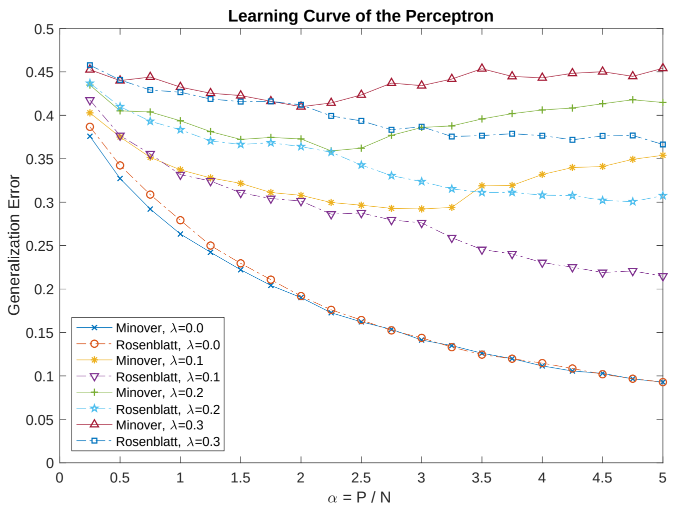

Figure 3. Generalization error for the MinOver and the Rosenblatt algorithms on a noisy dataset. The dataset is generated starting from a linearly separable set of examples and swapping the labels of each example with probability λ. The lines for $\lambda = 0$ (no noise) are left as a reference.

C. *Noise*

Figure 3 shows the generalization error of the perceptron trained with the MinOver and the Rosenblatt algorithms on a noisy dataset. For both algorithms, the generalization error increases with the noise λ.

The error for the MinOver algorithm initially decreases by adding new examples, while it starts to increase again if the dataset gets bigger: the algorithm diverges from the teacher vector $\mathbf{w}^*$. The more noise is present, the sooner the algorithm starts to diverge. At each step, the algorithm selects the example with the lower stability $\kappa^\mu \propto \mathbf{w} \cdot \xi^\mu S_R^\mu$: this quantity is positive for correctly classified examples and negative for wrongly classified ones. If there are misclassified examples, the MinOver algorithm chooses one of them to update the weights vector $\mathbf{w}$. If the dataset is non linearly separable, the MinOver algorithm stacks forever: each update can correct the classification for some point, but not for all of them; in the following epochs, only the misclassified examples cause updates and create new wrongly classified point. For the way it is constructed, the probability that the dataset is not linearly separable is non-zero (for $\lambda > 0$) and increases with α. For $\lambda = 0.3$, the algorithms has a generalization error of around 0.45, very close to the error for a random guess. The MinOver algorithm does not seem to be suitable for the classification of noisy data.

The Rosenblatt algorithm has similar performances, but it does not diverge when α increases. In other words, the Rosenblatt algorithm is up to a certain degree able to generalize even for noisy training data.

V. Conclusion

The MinOver algorithm tries to maximize the minimum stability of the dataset. In absence of noise, MinOver yields to a generalization error similar to the Rosenblatt algorithm. For both algorithms the generalization error decreases for a high $\alpha = P/N$ (i.e. with a higher number of examples P) and approaches zero in the limit for $\alpha \to \infty$. In presence of noise, the MinOver algorithm diverges for big datasets, where the likelihood of the data to be non linearly separable is very high.

In the original formulation, MinOver does not give any advantages over the Rosenblatt algorithm in terms of generalization error, both in absence and presence of noise in the data. In addition, the MinOver algorithm requires more computation power than the Rosenblatt one, since it needs to compute the stability of all examples in the dataset at each epoch.

VI. Individual Workload

The workload of this assignment was divided as follows.
Samuel Giacomelli:

- Generation of the datasets with noise.
- Basic implementation of the MinOver algorithm.
- Experiments on datasets with different degrees of noise, both for the MinOver and Rosenblatt training algorithms.

Davide Pedranz:

- Generation of the datasets without noise.
- Stopping criteria and optimization for the MinOver training algorithm.
- Experiments for the MinOver algorithm and comparison with the Rosenblatt one in absence of noise in the data.

We worked together on the report.

References

[1] W. Krauth and M. Mezard, "Learning algorithms with optimal stability in neural networks," *Journal of Physics A: Mathematical and General*, vol. 20, no. 11, p. L745, 1987.

[2] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

Appendix A
Matlab code

A. *MinOver algorithm*

```
function [w, n_updates] = minover(X, y, n_max)
    % TRAIN_PERCEPTRON Train a perceptron on the dataset X, y for at most
    % (n_max * P) epochs using the MinOver algorithm.

    % stop the training unless at least one component of the weights
    % vector does not change significantly
    min_update = 0.001;

    % extract the number of examples P and number of dimensions (N)
    P = size(X, 1);
    N = size(X, 2);

    % initialize the weights to zero
    w = zeros(N, 1);

    % repeat training for any epochs
    epochs = n_max * P;
    for epoch = 1:epochs

        % compute stabilities k^{v(t)}
        % NB: we do NOT divide by |w| since it is a constant for all
        % examples and we only want to take the examples with the lowest
        % stability
        stabilities = (X * w) .* y;

        % keep track of the old weight we use this to stop the training
        % if the weights do not change significantly for a training step
        old_w = w;

        % extract the example with lowest stability
        [~, index] = min(stabilities);

        % perform a Hebbian update step of the weights
        example = X(index, :);
        label = y(index);
        w = old_w + (example' * label) / N;

        % stopping criteria: stop if there is no significant update
        if all(norm(old_w - w) / norm(w) < min_update)
            break
        end
    end

    % we return the effective number of updates to complete the training
    n_updates = epoch;
end
```

B. Dataset generation basic implementation

```
function [X, y] = generate_dataset(P, N, w_star)
   X = randn(P, N);
   y = sign(X * w_star);
end
```

C. Dataset generation noisy data

```
function [X, y] = generate_dataset(P, N, w_star, lambda)
    prob = rand(P, 1);
    noise = iff(prob < lambda, -1, +1);
    X = randn(P, N);
    y = sign(X * w_star) .* noise;
end
```