LAB 1: Linear Models for Binary Classification

Hamza Ba-mohammed, Hicham Filali, Bouchra Sahri
{*hamza_bamohammed, hicham_filali2, bouchra_sahri*}*@um5.ac.ma*
Professor: Dr. Abdellatif El Afia

***Abstract*—This report resumes the results of the LAB1 of our Introduction to AI course. The section enumeration follows the same order as the LAB1 assignment.**
The source code of this work is available in a separate Python package.

***Key-words*—Artificial Intelligence, Binary Classification, Uniform Convergence, Perceptron, Adaline, Pocket Algorithm, Delta Rule, Gradient, Linear Data.**

I. Introduction

One of the oldest forms of Artificial Intelligence known in the history of modern computer science is the Perceptron, first proposed by psychologist Dr. Frank Rosenblatt in 1957. This model opened the door for future innovations in the field of machine learning algorithms. Among them, we discussed in this report:

1) Perceptron Learning Algorithm
2) Pocket Learning Algorithm
3) Adaline Algorithm: Δ rule variant

II. The 3 algorithms under scoop

A. Python code

The Python script of the 3 mentioned algorithms is available in the GitHub.
We used only the libraries `numpy` and `matplolib` for this package.

B. Datasets

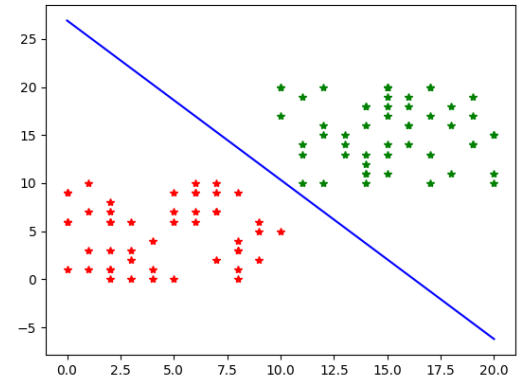
The Python package contains also a `data.py` file used mainly to generate 2 types of random datasets (with the library `random`) for our binary classification:

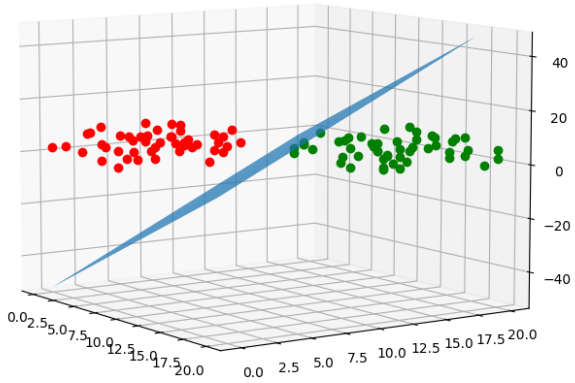
1) *Linear Separable Data*: the data could be separated using a single hyperplane. This is the only case of data that the PLA could process efficiently and where convergence criteria are guaranteed. However, this is also the less common type of data we face in real-life situations.
2) *Linear Non-Separable Data*: the data presents noise and could not be totally separated. This is the most frequent case for which we use either Adaline or Pocket algorithms.

The purpose of this method is to avoid the development of AI models with over-fitting issues: we use it later to simulate a

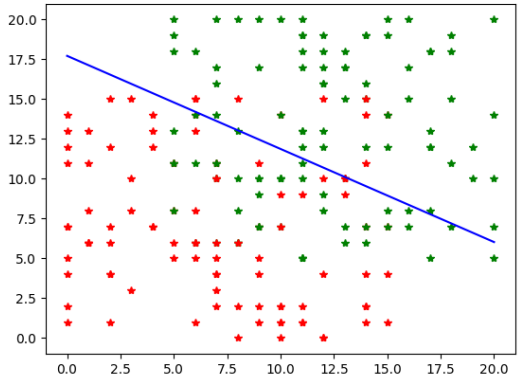
C. Plot results of each algorithm on a random dataset

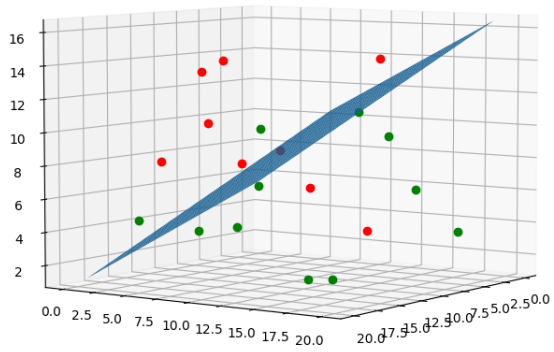
For this plots, we used $T_{max} = 100$.



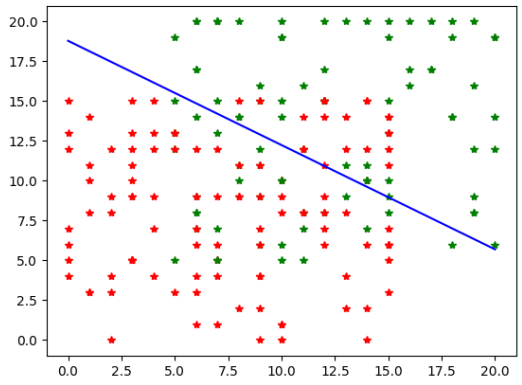


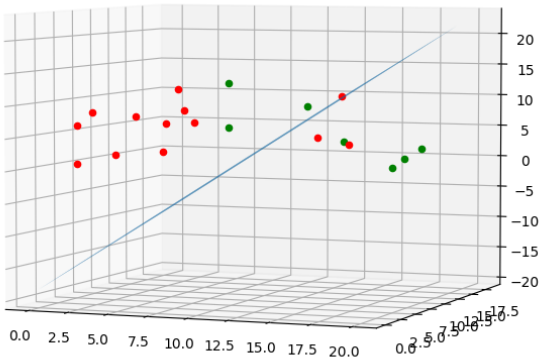
PLA with linearly separable randomized data





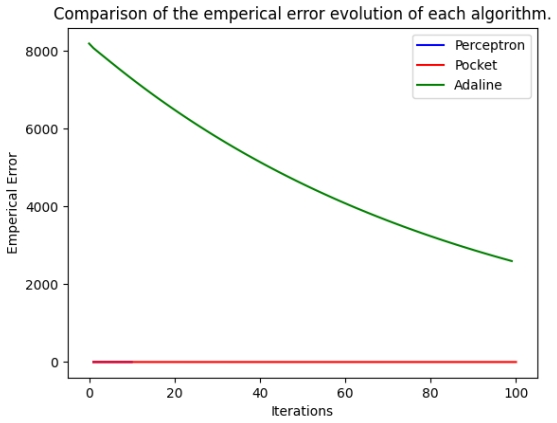
Pocket with linearly non-separable randomized data

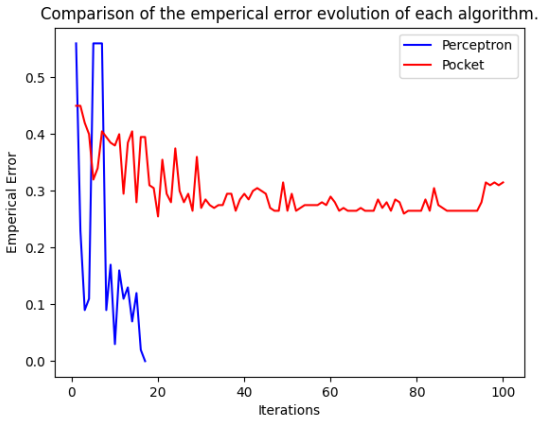




Adaline (delta rule variant) with linearly non-separable randomized data ($\alpha = 0.0001$)

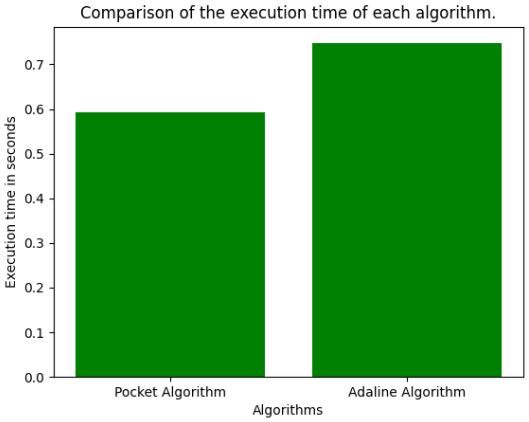
D. Empirical Error Evolution plots





E. Comparing the 3 algorithms

1) Number of iterations: The PLA is the fastest algorithm in a matter of the number of iterations. Actually, it is the only algorithm that always converges since it works on separable data while the other algorithms generally don't converge to 0, but finish by oscillating around a small value $\epsilon \sim 0.1$. Thus, generally, we notice that the number of iterations of the Pocket and Adaline algorithms is the same T_{max} set up in the start configuration.



2) Empirical error evolution: The speed of convergence of the Adaline algorithm is the slowest among the 3 algorithms.

Also, the value of its empirical error is the biggest among them, with a very big marge of difference. This could be explained by the nature of the loss function in the Adaline case: summing the square of the differences results in a very big value.
The PLA algorithm also has generally a faster speed of convergence and a more accurate result compared to the Pocket algorithm as we can see in the previous plot. Furthermore, the Pocket algorithm seems to stop its convergence starting from a certain iteration and enters a random oscillation character.

3) Accuracy: Could be deduced from the empirical error value: it's the complementary function of the loss function. In other words, the lower the empirical error, the higher the accuracy, and vice-versa.

III. Discussion and Synthesis

A. Proof of Convergence: linear separable data

1) Assumptions:

- Data is linearly separable:$\exists\, w^*$ such that $y_i(x_i^T w^*) > 0\ \forall (x_i, y_i) \in D$
- We re-scale each data point and the w^* such that:
  - $\|w^*\| = 1\, and\, \|x_i\| \leq 1\ i = 1, 2...., n$
  - All inputs x_i live within the unit sphere
  - w^* lies on the unit sphere

- we define the margin of a hype-plane, denoted by γ, as $\gamma = min_{(x_i,y_i)\in D} \left|x_i^T w^*\right|$
  - γ is the distance from the hyper-plane to the closest data point.

2) Theorem: Under these assumptions, the Perceptron algorithm makes at most $\frac{1}{\gamma^2}$ misclassifications.

3) Proof:

- In the Perceptron algorithm, when $y_i(x_i^T w^*)0$, we update as:
  $w_{new} = w + y_i x_i$
- consider the effect of an update on $w_{new}^T w^*$:
  $w_{new}^T w^* = (w + y_i x_i)^T w^* = w^T w^* + y_i(x_i^* w^*) w^T w^* + \gamma\ (1)$
  The inequality follows from the fact: w^*, the distance from the hyperplane defined by w^* to x_i must be at least γ
  (ie $y_i(x_i^T w^*) = \left|x_i^T w^*\right| \gamma$)
  This means that for each update, $w^T w^*$ grows by at least γ
- Consider the effect of an update on $w_{new}^T w_{new}$:
  $w_{new}^T w_{new} = (w + x_i y_i)^T (w + y_i x_i)$
  $= w^T w + 2y_i(w^T x_i) + y_i^2(x_i^T x) w^T w + 1\ (2)$
  The inequality follows from the fact $2y_i(w^T x_i)0$ as we had to make an update $y_i^2(x_i^T x_i)1$ as $y_i^2 = 1$ and all $x_i^T x_i 1$(because $\|x_i\|\, 1$)
  This means that for each update, $w^T w$ grows by at most 1.
  $w_{new} w^* w^T w^* + \gamma\ (1)$
  $w_{new} w_{new} w^T w + 1\ (2)$

  After M updates, we have:
  - $M\gamma w^T w^*$ from (1): each update increases,at least by γ
  - $M\gamma w^T w^* = \left|w^T w^*\right| \|w\| \|w^*\|$ (By Cauchy-Schwartz inequality)
  - $M\gamma \|w\| \|w^*\| = \|w\|$ (Unit sphere assumption)
  - $M\gamma \|w\| = \sqrt{w^T w}\sqrt{M}$
    $M\gamma\sqrt{M} \Rightarrow M^2\gamma^2 M \Rightarrow M\frac{1}{\gamma^2}$

  The theorem is proved since the number of updates is equal to the number of misclassifications.

B. Discussion about T_{max}

The learning rate and iteration choice are pretty arbitrary. This is essentially unavoidable in the context of training neural networks with gradient descent methods. Nowadays, there are several "tricks" that can be applied to search for parameters like the learning rate α more efficiently, but the problem of searching for those kinds of parameters persists.