Rapport projet classification

Réalisé par :
Maazouzi sabah
Talih aida

Encadré par :
Mr.abdellatif Elafia

Année universitaire 2020 - 2021

Table des matières

Chapitre 1

Classification :

1.1 Visualisation data :

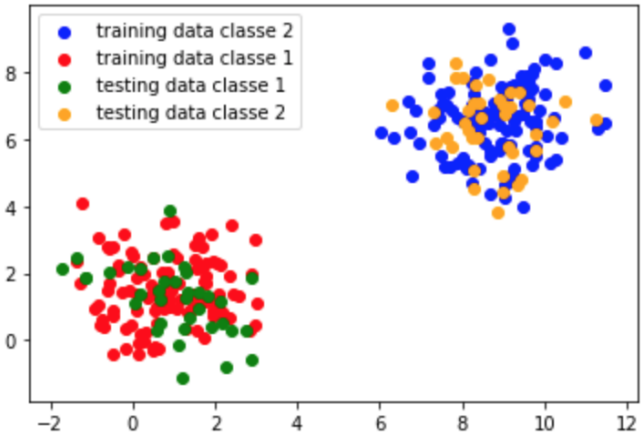


FIGURE 1.1 – data

1.1.1 Description :

L'image ci-dessous représente la moyenne , l'écart type ainsi que la valeur max et min pour chaque variable.

	x_1	x_2	y
count	300.000000	300.000000	300.000000
mean	4.838488	3.961575	0.500000
std	4.080293	2.760414	0.500835
min	-1.729020	-1.121061	0.000000
25%	0.908427	1.342451	0.000000
50%	4.526135	3.928230	0.500000
75%	8.847560	6.649956	1.000000
max	11.484430	9.307988	1.000000

1.1.2 Type du données :

D'après la visualisation du data on voit que chaque x_i est associé à un y_i .
On déduit qu'on a un apprentissage supervise.

1.2 Métriques de performance du modèle

1.2.1 Binary Cross-Entropy Loss

Binary Cross-Entropy Loss est la fonction de perte par défaut à utiliser pour les problèmes de classification binaire.Il est destiné à être utilisé avec une classification binaire où les valeurs cibles sont dans l'ensemble 0, 1. Mathématiquement, il s'agit de la fonction de perte préférée dans le cadre d'inférence du maximum de vraisemblance. C'est la fonction de perte à évaluer en premier et à ne changer que si vous avez une bonne raison.

Binary Cross-Entropy Loss calculera un score qui résume la différence moyenne entre les distributions de probabilité réelle et prévue pour la prédiction de la classe 1. Le score est minimisé et une valeur d'entropie croisée parfaite est 0.

$$H_p = \frac{1}{n}\sum_{i=0}^{n}(y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i)) \tag{1.1}$$

1.2.2 Hinge Loss

Une alternative à cross-entropy pour les problèmes de classification binaire est la fonction de Hinge Loss , principalement développée pour être utilisée avec les modèles SVM (Support Vector Machine).Il est destiné à être utilisé avec une classification binaire où les valeurs cibles sont dans l'ensemble $\{-1, 1\}$
La fonction de Hinge Loss encourage les exemples à avoir le signe correct, en attribuant plus d'erreur lorsqu'il y a une différence de signe entre les valeurs de classe réelles et prévues.Les rapports de performance avec Hinge Loss sont mitigés, ce qui entraîne parfois de meilleures performances que l'entropie croisée sur des problèmes de classification binaire.

$$L(y, y_h) = \frac{1}{n}\sum_{i=0}^{n}\max(0, y_i y_h i) \tag{1.2}$$

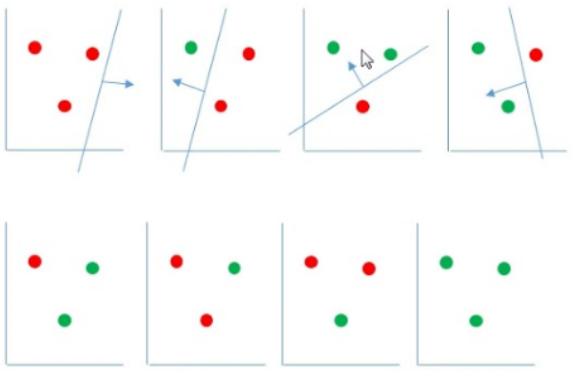
1.2.3 Squared Hinge Loss

La fonction de Hinge Lossa de nombreuses extensions, souvent l'objet d'investigations avec les modèles SVM. Une extension populaire est appelée Squared Hinge Lossqui calcule simplement le carré de la perte de charnière de score. Il a pour effet de lisser la surface de la fonction d'erreur et de la rendre numériquement plus facile à travailler.
Si l'utilisation d'une perte de charnière entraîne de meilleures performances sur un problème de classification binaire donné, il est probable qu'une perte de charnière au carré soit appropriée.

$$L(y, y_h) = \frac{1}{n}\sum_{i=0}^{n}\max(0, y_i y_(hi))^2 \tag{1.3}$$

1.3 VC dimension :

vu que le nombre de features est 2 alors vc dimension = 3



Chapitre 2

Perceptron

2.1 Introduction :

Le perceptron prend en entrée un vecteur à plusieurs dimensions (1 par neurone) et opère une séparation entre ces données pour fournir une sortie. Grâce à cette séparation qu'il a construite entre les données, il sait, pour un nouvel exemple, quelle doit être la réponse.Par exemple, si on a 2 classes en sortie (chat ou chien), on va entraîner le réseau à comprendre la différence entre les deux à partir des entrées.
Un neurone possède des entrées

- Chaque entrée possède un poids
- La sortie est une fonction du poids et des entrées
  Y = f(W1 ∗ X1 + W2 ∗ X2)

2.2 Erreur empirique :

$$L_s(w) = \frac{1}{n} \sum 1_(signe((w^t x_i))y_i) < 0 \tag{2.1}$$

2.3 algorithme

while (ls $\neq$ 0)
for i= 1,...n :
if signe(w$^t x_i$)$y_i < 0$
$w = w + y_i x_i$
$computeL_s(w)$
$returnw, L_s$

2.4 Résultat :

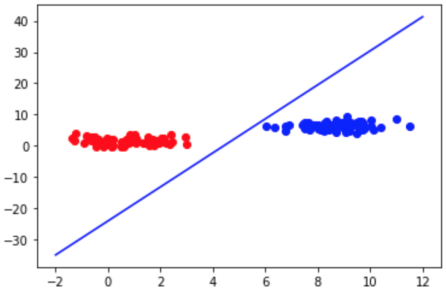


FIGURE 2.1 – resultat perceptron

2.5 Interprération :

Vu que les données des deux classes sont linéairement séparables et la conditions d'arret du perceptron est ls = 0 alors nous voyons que la droite sépare parfaitement les données .

Chapitre 3

Pocket

3.1 Introduction

L'algorithme de pocket est considéré comme capable de fournir pour tout problème de classification le vecteur de poids qui satisfait le nombre maximum de relations d'entrée-sortie contenues dans l'ensemble d'apprentissage. Un théorème de convergence approprié garantit l'obtention d'une configuration optimale avec une probabilité de un lorsque le nombre d'itérations augmente indéfiniment.

3.1.1 Transformation de Data

Algorithme de Pocket exige de travailler avec une data de bruit, donc premièrement nous avons ajouté un peu de bruit en prenant 7% de nos donnée et remplacer leurs labels en des labels opposés. voici le plot de la nouvelle data :

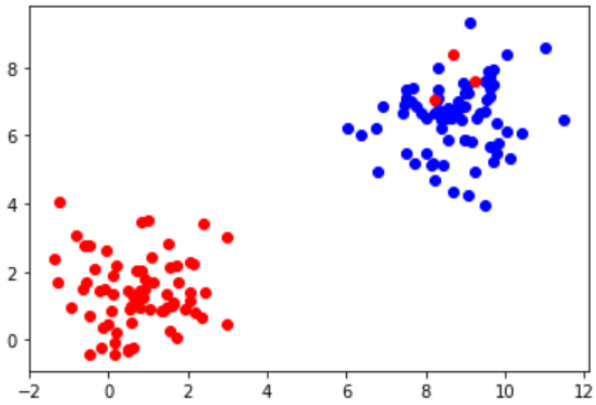


FIGURE 3.1 – visualisation data

3.1.2 Algorithm

Pocket learning algorithm

Input: $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and w_0.
Output: w^*, t and $L_S(w^*)$
Start: $w(0) \leftarrow w_0$
Initialize the weight vector of pocket by the weight vector of PLA.

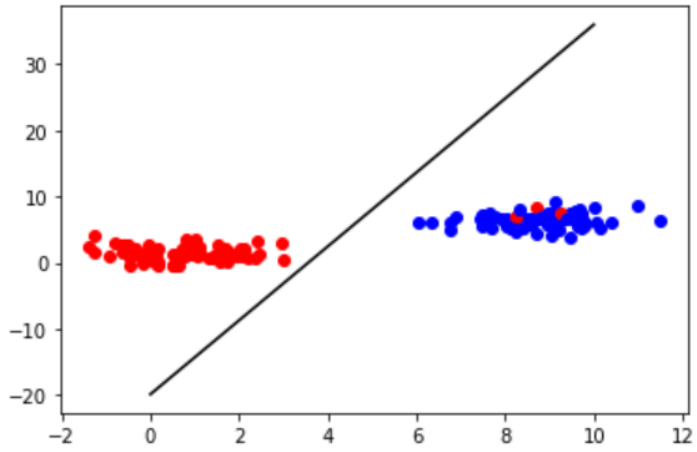
$$w_s \leftarrow w_0$$

for t = 1 , … , T_{max} :
  Execute PLA for one weight update to obtain w(t).
  Evaluate $L_S(w(t))$
  if $L_S(w(t)) < L_S(w_s)$
    $w_s \leftarrow w(t)$
  endif
  Return $w^* \leftarrow w_s$, t and $L_S(w^*)$
endfor

end

FIGURE 3.2 – pocket algorithm

3.1.3 Implémentation



3.1.4 Résultat

On voit que la droite sépare bien notre donnée avec des coefficients : [-86 , 26.96397725 , -1.47350537], Et avec une erreur de généralisation de 0.0333333333 Ce qui implique que notre model et bien entrainer.

Chapitre 4

Addaline :

4.1 Introduction :

Une illustration de l'ADAptive LInear NEuron (Adaline) - un neurone linéaire artificiel monocouche avec une unité de seuil.Le classificateur Adaline est étroitement lié à l'algorithme de régression linéaire des moindres carrés .Linear Regression implémente un modèle de régression linéaire pour effectuer une régression des moindres carrés ordinaires, et dans Adaline, nous ajoutons une fonction de seuil g pour convertir le résultat continu en une étiquette de classe catégorielle

4.1.1 Erreur empirique :

$$L_s(w) = \frac{1}{n}\sum(y_i - w^t x_i) \tag{4.1}$$

4.2 Algorithme :

for t = 1,.....T$_m$
$fori = 1,,,,n$
$if(e = y_i - w^t x_i)! = 0$
$w = w + 2ex_i$
$retun$w* , $l_s(w^*)$

4.2.1 Transformation de Data

Algorithme de Adaline exige de travailler avec une data de bruit, donc premièrement nous avons ajouté un peu de bruit en prenant 7% de nos donnée et remplacer leurs labels en des labels opposés. voici le plot de la nouvelle data :

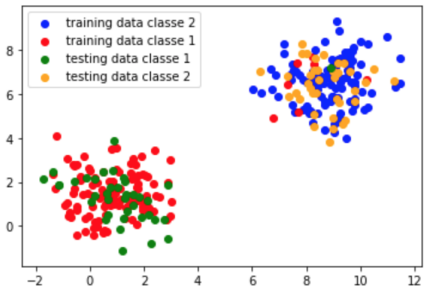


FIGURE 4.1 – Noise

4.2.2 Implémentation

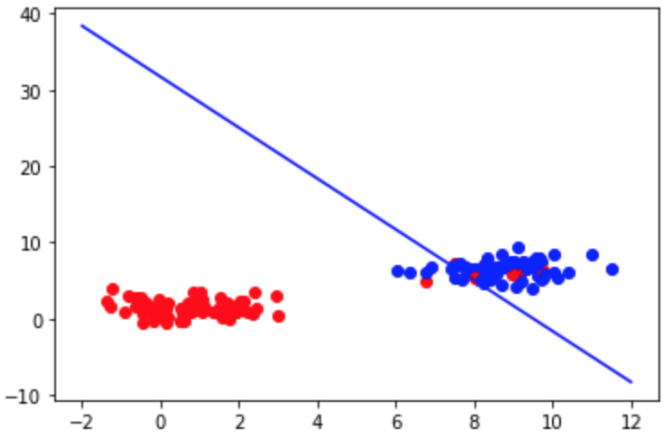


FIGURE 4.2 – resultat adaline

4.2.3 Interprétation :

D'après les résultats on voit que adaline prédit mieux que le perceptron et pocket vu que Perceptron utilise les étiquettes de classe pour apprendre les coefficients du modèle alors que Adaline utilise des valeurs prédites continues (à partir de l'entrée nette) pour apprendre les coefficients du modèle, ce qui est plus «puissant» car il nous indique par «combien» nous avions raison ou tort

Chapitre 5

Régression Logistique

5.1 Introduction

La régression logistique est l'un des algorithmes d'apprentissage automatique les plus couramment utilisés pour la classification . C'est un modèle statistique qui utilise une fonction logistique pour modéliser une variable dépendante binaire. Essentiellement, il prédit la probabilité qu'une observation appartienne à une certaine classe ou étiquette.

5.1.1 Hypothèse :

notre modèle prédise la probabilité d'une observation appartenant à une certaine classe ou étiquette. En tant que tel, nous voulons une hypothèse h qui satisfait la condition suivante $0 \leq h(x) \leq 1$, où x est une observation,avec :

$$h(x) = g(w^T x) \tag{5.1}$$

et g est une fonction sigmoïde et w sont les paramètres entraînables. À ce titre, nous avons :

$$h(x) = \frac{1}{1 + e^{-(w^T x)}} \tag{5.2}$$

5.1.2 Le coût d'une observation

maintenant que nous pouvons prédire la probabilité d'une observation, nous voulons que le résultat ait l'erreur minimale. Si l'étiquette de classe est y , le coût (erreur) associé à une observation x est donné par :

$$cost(h(x), y) = \begin{cases} -\log(h(x)) \,; if y = 1 \\ -\log(1 - h(x)) \,; if y = 0 \end{cases} \tag{5.3}$$

5.1.3 Fonction de coût :

Le coût total de toutes les m observations d'un ensemble de données est :

$$j(w) = \frac{1}{m} \sum_{i=1}^{m} cost(h(x^i), y^i) \tag{5.4}$$

5.1.4 Gradient Descent :

Gradient Descent est un algorithme d'optimisation utilisé pour minimiser certaines fonctions en se déplaçant de manière itérative dans la direction de descente la plus raide tel que défini par le négatif du pente. Nous mettrons à jour chacun des paramètres w_i en utilisant le modèle suivant :

$$w_i = w_i - \alpha \frac{\partial J(w)}{\partial J(w_i)} \tag{5.5}$$

avec :

$$\frac{\partial J(w)}{\partial J(w_i)} = \frac{1}{m} \sum_{j=1}^{m} (h(x^j) - y^j)^2 x_i^j \tag{5.6}$$

cette étape nous aidera à trouver un ensemble de paramètres W_i, qui nous aideront ensuite à trouver $h(x)$ pour résoudre notre tâche de classification binaire.

5.2 Régularisation

La régularisation est une technique pour résoudre le problème de surapprentissage dans un algorithme d'apprentissage automatique en pénalisant la fonction de coût. Pour ce faire, il utilise un terme de pénalité supplémentaire dans la fonction de coût. Il existe deux types de techniques de régularisation :

— Régularisation Lasso ou L1
— Régularisation Ridge ou L2

5.2.1 Fonction de coût :

La nouvelle fonction de coût s'écrit :

$$j(w) = \frac{1}{m}\sum_{i=1}^{m} cost(h(x^i), y^i) + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2 \tag{5.7}$$

λ est appelé le paramètre de régularisation. Il contrôle le compromis entre deux objectifs : bien ajuster les données d'entraînement contre garder les paramètres petits pour éviter le surajustement.

5.2.2 Gradient Descent :

Le gradient de $J(w)$ devient :

$$\frac{\partial J(w)}{\partial J(w_i)} = \frac{1}{m}[\sum_{j=1}^{m}(h(x^j) - y^j)^2 x_i^j + \lambda w_i] \tag{5.8}$$

Le terme de régularisation pénalisera fortement les grands w_i . L'effet sera moindre sur les w_i plus petits . En tant que tel, la croissance de w est contrôlée. Le $h(x)$ que nous obtenons avec ces paramètres contrôlés w sera plus généralisable.

5.2.3 Transformation de Data

Algorithme de Logistic regression exige de travailler avec une data de bruit, donc premièrement nous avons ajouté un peu de bruit en prenant 7% de nos donnée et remplacer leurs labels en des labels opposés. voici le plot de la nouvelle data :

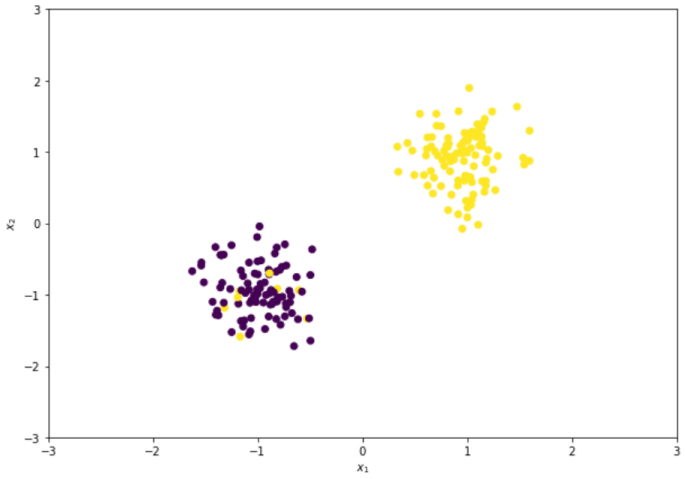
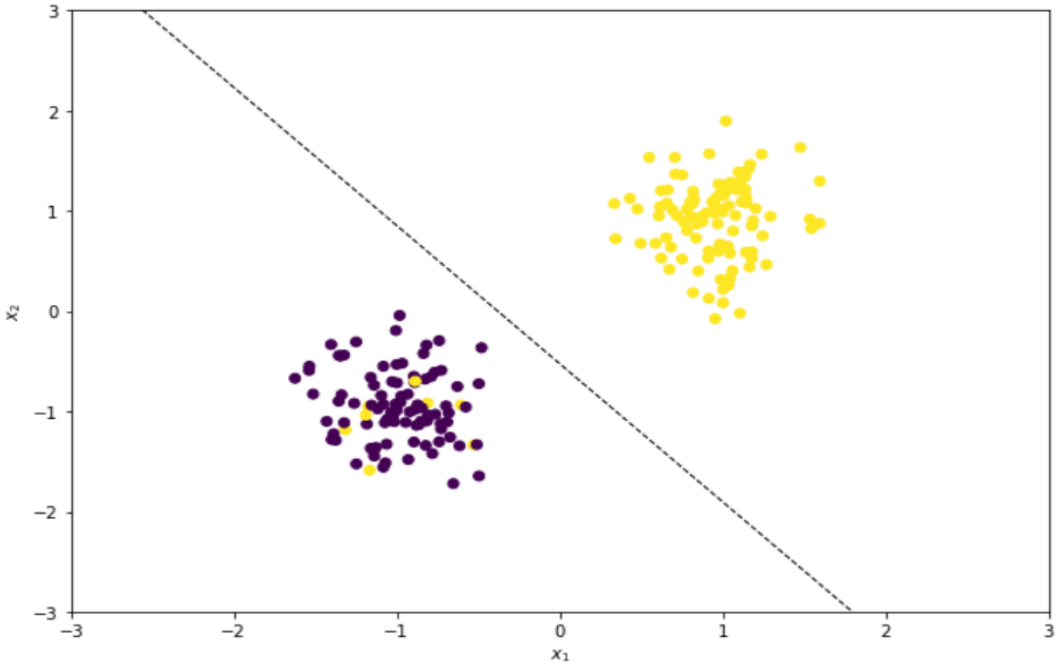


FIGURE 5.1 – visualisation data

5.2.4 Implémentation



Comme nous pouvons le voir, notre modèle est capable de très bien classer les observations. La frontière est la ligne de décision.

5.2.5 Résultat

Pour la partie régularisations on a pris lambda=0.1 ,nous avons obtenu une erreur de généralisation de 0.05 ce qui explique que notre modèle a bien entraîner. Aussi on a déduit que la régularisation n'affecte pas l'erreur empirique vu que on a eu la même valeur avec et sans régularisation. Cette dernier a un impact aussi sur notre coefficient de la droite de classification :
Avant la régularisation :[0.91153192 2.11729229 1.42095706]
Après la régularisation :[0.71888135 1.89095934 1.37161725]