PAO : machine learning et clicks prediction

Octave QUEFFELEC et Sandratra RASENDRASOA

11/06/2017

Contents

List of Figures

List of Algorithms

Part I
Introduction

> "We are drowning in information and starving for knowledge."
> Rutherford D. Roger[1]

> January 2009 Hal Varian, Google's Chief Economist, tells the McKinsey Quarterly: "I keep saying the sexy job in the next ten years will be statisticians. People think I'm joking, but who would've guessed that computer engineers would've been the sexy job of the 1990s? The ability to take data—to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it—that's going to be a hugely important skill in the next decades..."[2]

Avec l'avènement des technologies de l'information, le Data Science (ou science des données) est une discipline ayant pour objectif de répondre au besoin d'analyser des immenses ensembles de données pour en extraire des informations -potentiellement- utiles. Les applications sont nombreuses, que ce soit dans le secteur privé (la recommandation de séries par Netflix, les moteurs de recherche, l'analyse financière), le domaine médical (l'aide aux diagnostics) et bien d'autres encore.

Le but de notre PAO est de s'initier aux méthodes et algorithmes issus du Machine Learning. Pour cela, nous avons à notre disposition un jeu de données issu de la base de données en ligne Kaggle, qui a été créé pour répondre à la problématique suivante : déterminer un modéle pour estimer le taux de clicks(ou CTR) qui est une unité de mesure importante pour évaluer les publicités en ligne. Nous avons utilisé comme base un projet de l'université de Washington, en réalisant la partie pratique du projet et en ayant une réunion toutes les deux semaines avec nos responsables pédagogiques M.Gasso et M.Gauzere pour aborder ces différents points et les parties théoriques s'y référant.

L'ensemble du projet a été réalisé en langage Python qui est un des langages proposé dans le sujet de l'université de Washington. Nous avons notamment utilisé les librairies Numpy et Scipy pour nos algorithmes. Nous avons aussi dans un premier temps réalisé quelques algorithmes sous Matlab pour se familiariser avec les jeux de données.

Dans ce rapport, nous allons vous présenter tout d'abord les éléments théoriques nécessaires pour ce projet, puis l'ensemble des travaux pratiques que nous avons

[1] T.Hastie, R. Tibshrani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Springer Verlag)

[2] Gil Press, "A Very Short History Of Data Science",Forbes

réalisé ainsi que les résultats qui y sont liés. Nous terminerons par une interprétation de ces résultats et une conclusion globale.

Part II
Partie théorique

1 Supervised learning

Il existe plusieurs algorithmes d'appprentissage automatique(ou machine learning) qui se différencient par le mode d'apprentissage qu'ils utilisent. Dans le cadre de cette initiation, nous ne verrons qu'un mode d'apprentissage, mais il nous semble important de mentionner que d'autres techniques existent pour répondre aux différents besoins en machine learning.
L'apprentissage supervisé (ou supervised learning) est la technique d'apprentissage la plus commune pour répondre à des problèmes de machine learning. L'objectif est le suivant :

> A partir des données $(x_i, y_i)\epsilon X \times Y, i = ..., N$, estimer les dépendances entre X et Y.

On parle ici d'apprentissage supervisé (en opposition à l'apprentissage non-supervisé) car les y_i (en pratique, il s'agira des cas déja traités et validés) permettent de guider le processus d'estimation. Un exemple concret pour différencier l'apprentissage supervisé de l'apprentissage non-supervisé est présenté dans la figure 1.

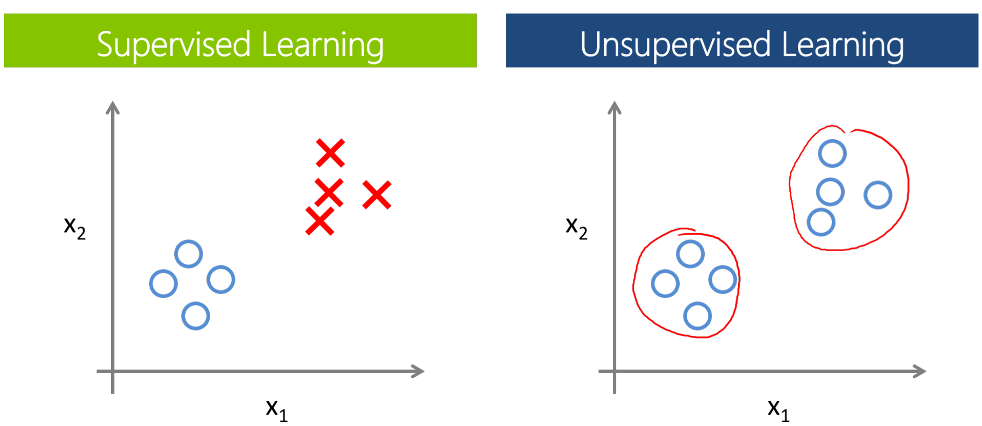

Figure 1: Apprentissage supervisé VS non-supervisé

En apprentissage supervisé, on sait d'avance qu'il y a exactement deux catégories(cercle bleu et croix rouge), tandis qu'en apprentisage non supervisé, on essaie de partitionner et classer les données dans des groupes homogènes (en anglais, on parle aussi de clustering). Dans le cadre de notre projet, nous travaillons en apprentissage supervisé, nous possédons ainsi plusieurs jeux de données correspondants à X et Y, que nous préciserons dans la partie pratique.

2 Regression Lineaire

La regression lineaire est une des méthodes d'apprentissage supervisé la plus simple. Une base de données d'apprentissage est un ensemble de couples entrée-sortie $(x_i, y_i), x \in X, y \in Y, i = 1..n$ qui suit une loi inconnu. Un apprentissage est supervisé lorsque l'on connais les sorties y_i et qu'elles nous permettent de guider le processus d'estimation. Le but d'un algorithme d'apprentissage supervisé est donc de généraliser pour des entrées inconnues ce qu'il a pu « apprendre » grâce aux données déjà traitées, ceci de façon « raisonnable ». On dit que la fonction de prédiction apprise doit avoir de bonnes garanties en généralisation. Le but d'un probleme de regression est de predire grace a cette fonction de prediction la sortie y d'une entrée x dont on ne connais pas par avance la veritable sortie.

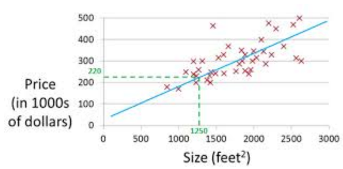

Figure 2: Prix des maisons dans l'etat d'Oregon

Voici un graphe avec le prix des maisons dans l'etat d'Oregon en fonction de leur superficie en mètre carré. Le dataSet nous est donné. Le probleme posé est le suivant : pour une superficie donnée, quel serais le prix le plus adéquat de la maison ?
Pour cela, il nous faut modeliser ce jeux de données. On pourrais par exemple tracer une droite en minimisant chaque distances entre la droite et un point des données pour representer le modele au mieux.
Pour ce faire, on dispose d'un training set, un ensemble de couple (x_i, y_i).

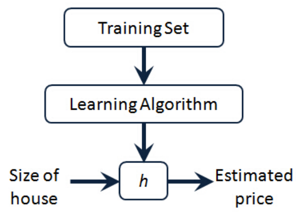

Figure 3: Modélisation du jeu de données

A l'aide du training set, l'algorithme d'apprentissage doit determiner une fonction h. Cette fonction dois predire un y pour chaque entree x passé en entré de la fonction.

Generalement, cette fonction est defini dans l'absolu comme :

$h(x) =< w, x >, x \in R^n, w \in R^n$

Dans notre cas de regression lineaire, on aurais quelquechose de la forme :

$h(x) = w_0 + w_1 * x, x \in R^2, w \in R^2$

En somme, il "suffit" que notre algo determine correctement les coefficients w_0 et w_1.

3 Regression Logistique

La regression logistique est un cas particulier de la regression lineaire, c'est un modele de regression binomiale. Ainsi, la variable de sortie prends 2 modalités : $y = 0, 1$

On cherche à predire une probabilité d'appartenance à une classe ou non.

Regle de decision de Bayes :

$D(x) = C1$si $P(C1|X)/P(C2|X) > 1$, $C2$ sinon

Avec $P(C1|X) = exp(wTx)/(1 + exp(wTx))$. La fonction logistique associé donne donc :

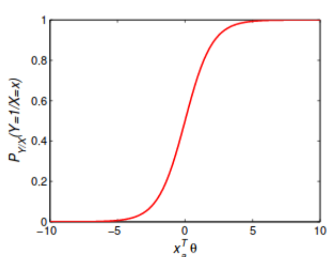

Figure 4: Fonction logistique associée à D(x)

4 Fonction de coût

En apprentissage supervisé, l'objectif est de trouver une fonction $f : X \rightarrow Y$ qui permet d'estimer la valeur y associée à x. On peut alors se poser la question suivante : la fonction f que nous avons construite est-elle optimale pour résoudre notre problème ?
On introduit alors la notion de de coût $L(Y, f(X))$, dont l'objectif et d'évaluer la pertinence de la prédiction réalisée par f, et de pénaliser les erreurs. Dans l'idéal, on chercherait donc une fonction f qui prédit au mieux y; en d'autres termes, une fonction f qui minimise l'erreur entre la vérité et la valeur estimée. D'où la fonction f telle que :

$$R(f) = E_{X,Y}[L(Y, f(X))]$$

R(f) où R est appelé le risque moyen ou erreur de généralisation.
Un exemple de fonction de coût et de risque moyen associé est le coût quadratique (ou moindres carrés) :

$$L(Y, f(X)) = (Y - f(X))^2$$

$$R(f) = E[(Y - f(X))^2] = \int (y - f(x))^2 p(x, y) dx dy$$

La fonction de coût est associé à l'ensemble Y dont on souhaite prédire les valeurs, et on peut distinguer deux types de problèmes en fonction de cet ensemble Y.

1. Si l'ensemble Y est un sous-ensemble de R^d (donc continu), on traitera le problème comme étant un problème dit de régression, où les moindres carrés sont la fonction de coût usuelle.

2. Dans le cadre de notre projet, Y est un ensemble discret non-ordonné (clic ou non clic, 0 ou 1 dans le jeu de données), il s'agit d'un problème dit de classification. Ici, on cherchera plutôt à approcher la fonction de coût suivante : $\theta(-yf(x))$où θ est la fonction échelon.

5 Perceptron

Le perceptron est un classifieur lineaire de la forme :

$$f(x) = \begin{cases} 1 & si < w, x > +b > 0 \\ 0 & sinon \end{cases}, w \epsilon R^n$$

$w$le vecteur des poids, $b \in R$, le biais, $< w, x >$, le produit scalaire entre le vecteur de poids w et le vecteur d'entrée x. $< w, x > +b = 0$ est alors l'équation de l'hyperplan affine qui sépare l'espace en 2 classes. Le but est donc d'entrainer notre vecteur de poids w afin de connaître par la suite notre fonction f qui nous permettra de prédire la classe d'un vecteur d'entrée x donné.

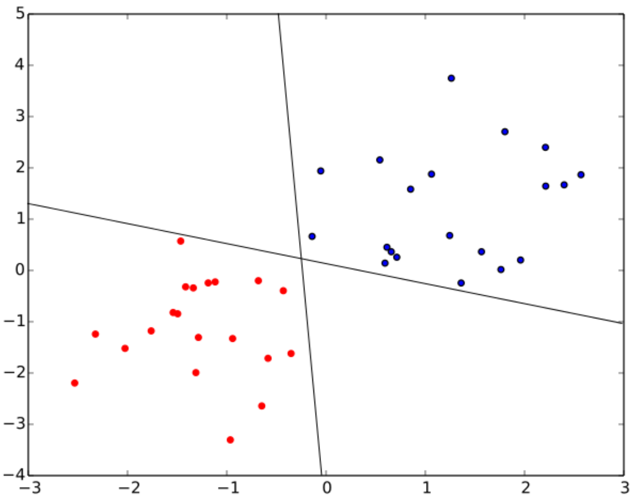

Figure 5: Perceptron

L'équation séparatrice des 2 classes n'est pas unique. Ici, 2 droites sont quasi-perpendiculaires l'une à l'autre mais le perceptron n'a aucun moyen d'en privilégier l'une au profit de l'autre.
Voici l'algorithme du perceptron, qui nous permet de mettre à jour à chaque iteration les poids w :

Algorithme d'apprentissage du Perceptron
Entrée : $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$, un échantillon complété linéairement séparable de $\mathbb{R}^{n+1} \times \{-1, 1\}$
$\mathbf{w}_0 = 0 \in \mathbb{R}^{n+1}, k = 0$
Répéter
    Pour $i = 1$ **à** l
        Si $y_i \langle \mathbf{w}_k, \mathbf{x}_i \rangle \leq 0$ **alors**
            $\mathbf{w}_{k+1} = \mathbf{w}_k + y_i \mathbf{x}_i$
            $k = k + 1$
    FinPour
Jusqu'à ce qu'il n'y ait plus d'erreurs
Sortie : $\mathbf{w}_k$

Figure 6: Algorithme Perceptron

Le perceptron peut etre vu comme un type de réseau de neurones simplifié.

Nous avons donc voulu pour nous entrainer, implémenter en matlab puis en python l'algorithme.

5.1 Descente de gradient

De manière générale, la descente de gradient est un algorithme cherchant à minimiser une fonction $J(\omega_0, \omega_1)$. Il pourrait dons notre cas s'agir de la fonction de coût que nous utilisons en régression logistique. Le principe général est le suivant : Soit une fonction

$$J(\omega_0, \omega_1)$$

on veut

$$min_{\omega_0,\omega_1}(J(\omega_0, \omega_1))$$

Etapes:

1. on débute à ω_0, ω_1
2. on modifie les paramètres ω_0, ω_1 pour minimiser notre fonction et arriver à un minimum.

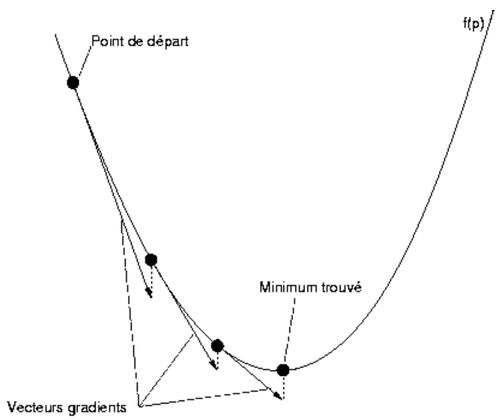

Figure 7: Descente de gradient

Part III
Implémentation

Le CTR (ou taux de clic) est une mesure cohérente pour évaluer la popularité d'une publicité. En pratique, notre objectif sera d'entraîner notre modèle pour espérer prédire le CTR sur un jeu de données d'environ un million d'exemples.

1 Présentation du jeu de données

1.1 Présentation générale

Les données que nous manipulons sont issues de la compétition 2012 KDD Cup Track 2. Nous avons à notre disposition 3 fichiers CSV : un fichier pour l'apprentissage "train.txt" et deux fichiers pour la validation "test.txt" et "test_label.txt".Pour mieux visualiser les données, prenons l'exemple d'un untilisateur ayant réalisé une requête sur moteur de recherche et nous allons observer sur quelles publicités l'utilisateur a cliqué :

1. Nicolas réalise une recherche sur le "très utilisé" moteur de recherche Bing, où il a écrit la requête "perruque".

2. En plus des résultats, Bing affiche plusieurs publicités contenant des images et du texte (description, titre, etc).

3. Nicolas clique alors sur la première publicité.

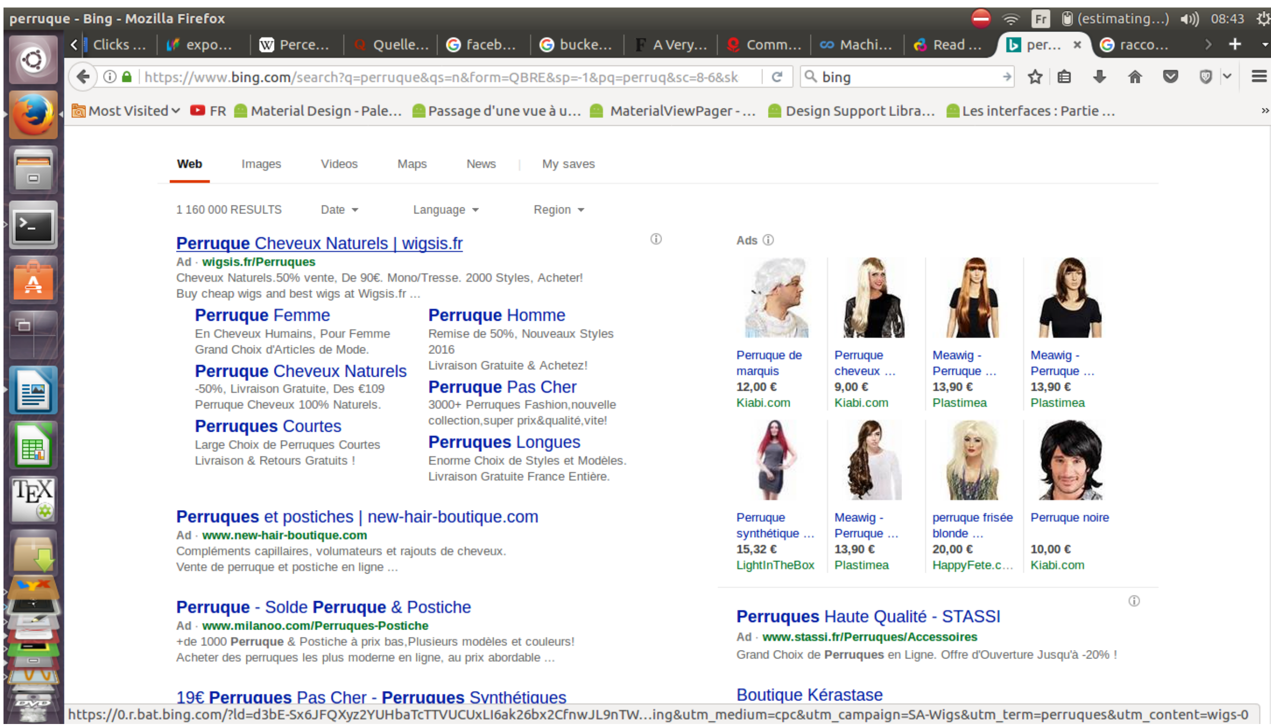

Figure 8: Exemple de session

Ces 3 étapes correspondent à une session. A la fin de cette session, Bing a receuillli un certain nombre d'enregistrement :

$Clicked = 1|Depth = 10|Position = 1|Nicolas|TextAd1$
$Clicked = 0|Depth = 10|Position = 2|Nicolas|TextAd2$
$Clicked = 0|Depth = 10|Position = 3|Nicolas|TextAd3$
etc...

En plus de ces d'enregistrements, Bing possède d'autres informations sur l'utilisateur. Le format complet d'une ligne du jeu de données est présenté ici :

Clicked | Depth | Position | Userid | Gender | Age | Text Tokens of Ad

Les caractéristiques sont délibérément laissé en anglais afin de rester cohérent avec le code présenté ci-après.
Nous allons maintenant présenter chacune des caractéristiques :

Clicked: 0 ou 1, indique si l'utilisateur a cliqué ou non sur la publicité (0 non clic et 1 clic).

Depth: prend une valeur dans $\{1, 2, ...\}$, indique le nombre de publicités affichées dans chaque session.

Position: prend un valeur dans $\{1, 2, ..., Depth\}$,indique le rang/position de la publicité parmi les publicités affichées.

Userid: l'identifiant d'un utilisateur.

Gender: le genre $\{-1, 0, 1\}$d'un utilisateur : -1 homme, 1 femme et 0 inconnu.

Age: prend une valeur dans $\{0, 1, 2, 3, 4, 5, 6\}$,indique la tranche d'age d'un utlisateur : 0 si inconnu, 1 si entre $(0, 12]$,2 si entre $(12, 18]$,3 si entre $(19, 24]$,4 si entre $(24, 30]$,5 si entre $(30, 40]$et 6 si au-delà de 40.

TextTokensofAd: une liste séparée par des virgules des "clés" de chaque mot d'une publicité. Par exemple "9,30,151" correspond aux clés des mots "mot#9","mot#30","mot#151" du dictionnaire utilisé (la correspondance avec des mots réels" n'est pas réalisé pour des soucis de confidentialité).

Voici donc un échantillon complet d'une ligne du jeu de données "train.txt" :

0|2|1|490234|1|3|0,133,1374,164,17005,1891,208,211,246,2527,3917,459,5705,625,850,8726,888,906,93

Comme on peut le constater, l'ensemble des attributs correspond ici à des scalaires pour faciliter le traitement.La seule différence entre le fichier d'apprentissage et de validation est que la première caractéristisque du fichier de validation "test.txt" est séparée du reste du fichier et enregistrée dans le fichier "test_label.txt".

1.2 Zoom sur la représentation réelle des attributs

De manière théorique, il est facile représenter le vecteur des attributs :

$$x^t = [x_1^t, ..., x_d^t]$$

où chaque attribut est un élément du vecteur x^t.
En pratique, la contruction de ce vecteur nécessite plusieurs prérequis :

1. L'attribut "Userid" ne sera pas considéré comme une caractéristique du vecteur; il ne sera donc pas pris en compte.

2. Les nombres présents dans la liste des tokens étant assignés de manière arbitraire, il est inutile de les considérer dans notre modèle. Il est conseillé de visualiser la liste des tokens $L = l_1, l_2, ...$comme étant la représentation compacte d'un vecteur creux $b$où $b(i) = 1 \forall i \epsilon L$ .Ainsi les valeurs non-nulles de ce vecteur correspondent aux clés des mots utilisés pour décrire une publicité.

3. Le reste des attributs sera utilisé en tant que tel.

2 Présentation des différents fichiers/classes

L'enoncé à mis a disposition un package avec differentes classes python afin de gerer assez facilement l'acces aux données.

BasicAnalysis.py Permet de faire le warm up, on y calcule entre autre les unique users, unique tokens, ctr. . .

DataInstance.py La classe DataInstance sectionne une instance des données en plusieurs attributs qui sont les differents features : clicked, depth,position,id,gender,age,tokens. . . Cela facilite par la suite l'acces au features.

DataSet.py La classe DataSet permet de charger un jeu de données et de naviguer entre les differentes instances à l'aide de fonction comme : hasNext(), nextInstance(), reset()...

DummyLoader.py Sert à verifier si les données chargent correctement. Il print un nombre defini d'instances.

LogisticRegression.py Permet d'entrainer les poids grace au training set et de predire sur le testing set.

3 Warm-up

Notre premier traitement sur les données correspondra à plusieurs opérations simples, pour s'assurer qu'on peut correctement accéder et manipuler les données. En somme, les notions utilisés ici sont la manipulation des fichiers CSV en Python et l'utilisation basique des fonctionnalités de la librairie Numpy.
Nous profitons des opérations déja fournis avec le template de l'université de Washington pour accéder aux attributs qui nous intéressent.

3.1 Taux de clic moyen

Algorithm 1 taux de clic moyen

```
def average_ctr(self, dataset):
temp = 0
x = []
while dataset.hasNext():
  temp = dataset.nextInstance().clicked
  x.append(temp)
return numpy.sum(x)/dataset.size
```

Résultat : *Average CTR = 3.36552848438 %*

3.2 Nombre de tokens unique dans les données d'apprentissage

Algorithm 2 Nombre de tokens unique

```
def uniq_tokens(self, dataset):
X =[]
instance = dataset.nextInstance()
while dataset.hasNext():
  y = instance.tokens
  # tokens uniques de la ligne i
  x = np.unique(y)
  X.append(x)
  instance = dataset.nextInstance()
X = np.array(x)
# tokens uniques de X = [x1' x2' ... xi' ... xn']'
X = np.unique(X)
return X
```

Résultat : *there are 24 unique tokens*

X_{unique} = [1 3 26 144 149 178 255 440 466 582 685 771 1181 1691 1766 2854 5838 5977 6800 8511 8833 9930 10948 15645]

4 Implémentation du perceptron

4.1 Matlab

Nous avons implementé differents systemes assez simple.
Pour chaque systeme, nous avons gardé la meme notation :

1. l:le nombre d'entree

2. n:la dimension des entrées

3. $X = [x1, x2, ..., xl]$, on concatene les l données dans X

4. $x_i = [a_1, a_2, ..., a_n, 1]$on concatene le biais a la fin de chaque vecteur d'entrée

5. $y = [1, -1, 1, ...] \in R^l$

Algorithm 3 perceptron sous matlab

```
w1=zeros(n+1,1);
k=0;
K=67;
for compteur=1:K
  indice=randperm(l);
    for i=1:l i=indice(i);
        if(y(i)*(x(i,:)*w1)<=0)
            w1=w1+y(i)*x(i,:)';
            k=k+1;
        end
    end
end
```

4.2 Python

Nous avons ensuite implementé le perceptron en python, afin de nous familiariser avec les librairies numpy, math, et matplotlib. Les résultats obtenus sont identiques à ceux en matlab.

Algorithm 4 perceptron en python

```
l = np.size(x[:,1])
for count in range(1,k):
  indice = np.random.permutation(range(l))
  for i in range(1,l):
    i = indice[i]
    if(np.dot(y[i],np.inner(x[i,:],w))<=0):
        w += np.dot(y[i],x[i,:])
```

Résultats pour les différents systèmes :

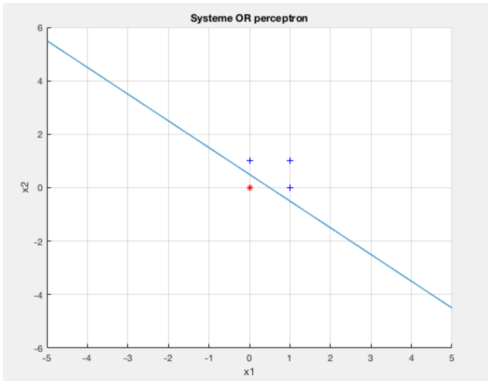

Figure 9: Système OR

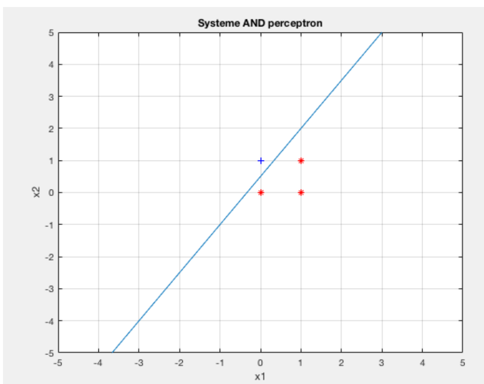

Figure 10: Système AND

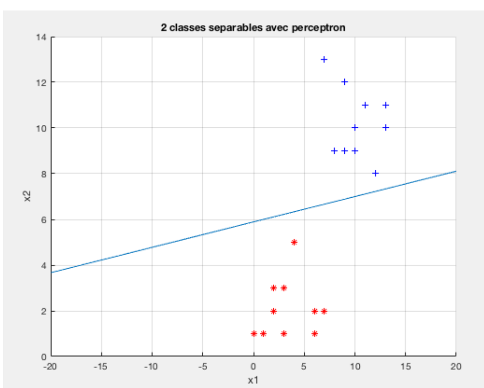

Figure 11: 2 classes séparables

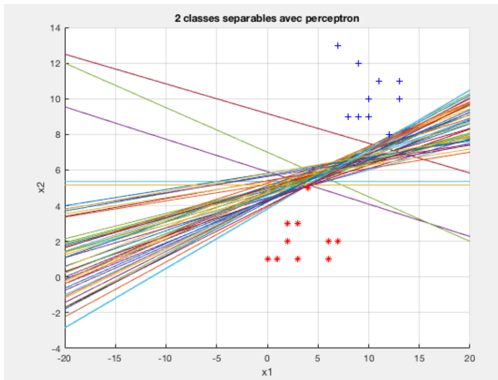

Figure 12: plusieurs solutions pour 2 classes séparables

5 Descente de gradient stochastique (SGD)

L'objectif de la descente de gradient stochastique est de réaliser une estimation du "vrai" gradient en ne mettant à jour que ce qui nous intéresse. Le principe est le suivant :

1. Choisir un vecteur initial de paramètres ω, et un taux d'apprentissage η.
2. Répéter jusqu'à ce qu'un minimum approché (assez précisément) soit obtenu :
3. Mélanger aléatoirement les échantillons de l'ensemble d'apprentissage.
4. Pour $i = 1, 2, ..., n$ faire :
5. $\omega = \omega - \eta \nabla Qi(w)$

Dans le projet, le vecteur de paramètresω est une classe nommée Weights, possédant qui possèdent plusiers paramètres :

Algorithm 5 Classe Weights

```
# This class represents the weights in the logistic regression model.
class Weights:
def __init__(self):
self.w0 = self.w_age = self.w_gender = self.w_depth = self.w_position = 0
# token feature weights
self.w_tokens = np.zeros(shape=(1070659,1))
# to keep track of the access timestamp of feature weights.
# use this to do delayed regularization. self.access_time = {}
def __str__(self):
formatter = "{0:.2f}"
string = ""
string += "Intercept: " + formatter.format(self.w0) + "\n"
string += "Depth: " + formatter.format(self.w_depth) + "\n"
string += "Position: " + formatter.format(self.w_position) + "\n"
string += "Gender: " + formatter.format(self.w_gender) + "\n"
string += "Age: " + formatter.format(self.w_age) + "\n"
string += "Tokens: " + str(self.w_tokens) + "\n"
return string
```

- La mise à jour des paramètres ωest réalisé dans la fonction *train(self, dataset, lambduh, step, avg_loss).* Cette fonction retourne les paramètres mis à jour ainsi que la courbe d'erreur cumulative.

Algorithm 6 simple train

```
def train(self, dataset, lambduh, step, avg_loss):
N = dataset.size
weights= Weights()
n_epoch = 1
N00=0
N01=0
N10=0
N11=0
count = 0
nbStep = 100
T = np.linspace(0,N,N/nbStep)
for epoch in range(n_epoch):
  while (dataset.hasNext()):
    instance = dataset.nextInstance()
    prediction = self.predict(weights, instance)
    error = instance.clicked - prediction
    if(error==0):
      if(prediction==0):
        N00+=1
      else:
        N11+=1
    else:
        if(prediction==0):
          N10+=1
        else:
          N01+=1
# if (error!=0):
  weights.w0 = weights.w0 + step * error
  weights.w_age = weights.w_age + step * error * instance.age
  weights.w_gender = weights.w_gender + step * error * instance.gender
  weights.w_depth = weights.w_depth + step * error * instance.depth
  weights.w_position = weights.w_position + step * error * instance.position
  for indice in instance.tokens:
    weights.w_tokens[indice]= weights.w_tokens[indice]+step*error
# record the average loss for each step 100
  avg_loss[0] = (1 / 2) * (error * error)
  j = count % nbStep
  if (j == 0 and count / nbStep != 0):
     avg_loss[int(count / nbStep)] = (1 / (2 * count)) * (error * error) +
avg_loss[int(count / nbStep) - 1]
    count += 1
  print("train DONE")
return weights,N00,N10,N01,N11,T,avg_loss
```

Algorithm 9 logreg

```
def regularised_train(self, dataset, lambduh, step, avg_loss):
#—même chose que le regularized train—
prediction =self.sigmoid(self.compute_weight_feature_product(weights,instance))
#—même chose que le regularized train—
```

- Régularisation

Algorithm 7 regularized train

```
def regularised_train(self, dataset, lambduh, step, avg_loss):
#—même chose que le simple train—
  weights.w0 = (1-step*lambduh/N)*weights.w0 + step * error
  weights.w_age = (1-step*lambduh/N)*weights.w_age + step * error * instance.age
  weights.w_gender = (1-step*lambduh/N)*weights.w_gender + step * error * instance.gender
  weights.w_depth = (1-step*lambduh/N)*weights.w_depth + step * error * instance.depth
  weights.w_position = (1-step*lambduh/N)*weights.w_position + step * error * instance.position
#—même chose que le simple train—
```

- logreg

Algorithm 8 sigmoïde

```
def sigmoid(self,z):
return 1/(1+math.exp(-1*z))
```

Part IV
Evaluation du modèle

L'évaluation est réalisée pour $\lambda = \{0.01\}$pour la régularisation et le logreg.

0.1 Matrice de confusion

0.1.1 Apprentissage simple

Prédiction ŷ / Vérité y	Positifs	Négatifs
Positifs	45978	1906
Négatifs	1916	200

- Ratio de reussite pour le training 0.92356
- Average ctr for training 0.04212
- Average ctr for training predicted 0.04232

0.1.2 Régularisation

Prédiction ŷ / Vérité y	Positifs	Négatifs
Positifs	45989	1898
Négatifs	1905	208

- Ratio de reussite pour le training 0.92394
- Average ctr for training 0.04212
- Average ctr for training predicted 0.04226

0.1.3 LogRegression

Prédiction ŷ / Vérité y	Positifs	Négatifs
Positifs	45989	1898
Négatifs	1905	208

- Ratio de reussite pour le training 0.92394
- Average ctr for training 0.04212
- Average ctr for training predicted 0.04226

<Courbes d'erreur moyenne>

1 Interprétation des résultats

2 Répartition des tâches

Tâches	Octave	Sandratra
taux de clic moyen		X
unique tokens	X	
unique users	X	
Descente de gradient stochastique	X	X
Régularisation	X	X
Evaluation du modèle	X	X
Rédaction du rapport	X	X

3 Conclusion