TD – Exemple de régression logistique

Présentation générale

Sur un réseau social on a collecté pour un certain nombre d'utilisateurs 3 données :

- ☞ L'âge des utilisateurs (en années) : Age
- ☞ Les revenus annuels (en k€) : Salaire
- ☞ Si ils ont achetés ou non un produit (0 ou 1)

Ces données sont reportées en 2D sur la figure 1 et en 3 D sur la figure 2)

On souhaite prospecter d'éventuels clients en se concentrant toutefois sur les plus susceptibles d'acheter le produit en question.

Pour cela nous aimerions avoir une prédiction (loi prédictive) nous indiquant en fonction de l'âge et des revenus la possibilité qu'une personne achète ou non le produit.

Le but est donc d'établir une régression logistique permettant de prédire si une personne peut acheter le produit suivant son âge et ses revenus :

$$\mathbf{y_p = f(salaire,âge) \text{ avec } y_p \in [0,1]}$$

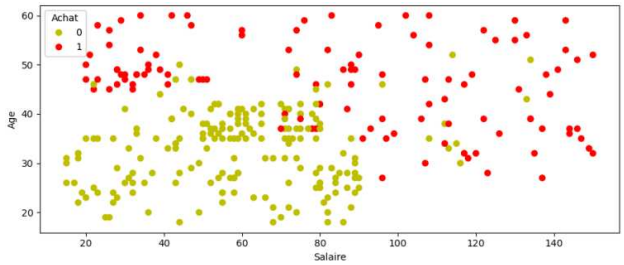

Figure 1 : Représentation graphique 2D des données

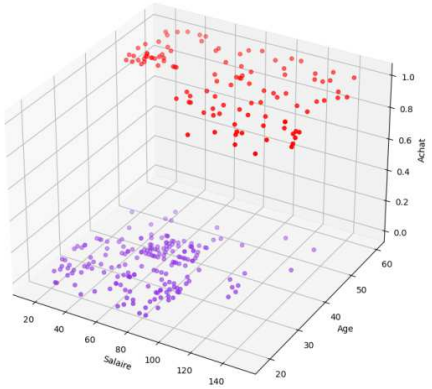

Figure 2 : Représentation graphique 3D des données

Cette loi de prédiction devra donner le bon résultat (La personnes va acheter le produit ou pas) dans plus de 95% des cas de figure.

1- Analyse des données – Mise en place du modèle de prédiction

Exécuter le code puis afficher en 3D l'ensemble des données d'apprentissage et de test. Pour cela lancer les commandes : `affichage_donnees_3D(Entrees,Sorties).`

Observer la répartition des données en tournant la figure de manières à observer la répartition des couleurs (Rouge = 1 = Achat , Violet = 0 = pas d'achat) dans le plan (Salaires,Age).

Question 1 - Une loi de prédiction basée uniquement sur le salaire est-elle satisfaisante ? Sur l'âge ?

Question 2 - Une loi de prédiction uniquement basée sur une fonction affine de l'âge et du salaire peut-elle être satisfaisante ?

Une loi de prédiction uniquement basée sur une fonction affine de l'âge et du salaire n'étant pas forcément suffisante, nous allons établir une loi de prédiction basée sur une fonction polynômiale de l'âge et du salaire. Avec un degré n_S pour le salaire et n_A pour l'âge. Soit sur une fonction du salaire x_S et de l'âge x_A de type : $f(x_S,x_A) = \omega_0 + \omega_1.x_S + \omega_2.x_S^2 + .. + \omega_{n_S}.x_S^{n_S} + \omega_{1+n_S}.x_A + \omega_{2+n_S}.x_A^2 + .. + \omega_{n_A+n_S}.x_A^{n_A}$

Etant donné que l'on souhaite une réponse binaire (0 ou 1) on va déterminer une valeur de prédiction y_P comprise entre 0 et 1 par la fonction sigmoïde : $y_P = \dfrac{1}{1+e^{-f(x_S,x_A)}}$ D'où la loi de prédiction :

$$y_P = \frac{1}{1+e^{-(\omega_0 + \omega_1.x_S + \omega_2.x_S^2 + .. + \omega_{n_S}.x_S^{n_S} + \omega_{n_S+1}.x_A + \omega_{n_S+2}.x_A^2 + .. + \omega_{n_S+n_A}.x_A^{n_A})}}$$

Remarque $y_P \in]0,1[$

Ensuite si $y_P \geq 0,5$ on considèrera que la personne achète : Prédiction = 1. Sinon Prédiction = 0

2- Formatage des données pour l'algorithme de la descente de gradient

Nous avons vu que la détermination des paramètres ω_i de la fonction polynômiale peut se faire avec un algorithme de la descente de gradient très similaire à celui des régressions linéaires ou polynômiales. Nous allons donc utiliser un code très proche du code du TD précédent pour cette régression logistique.

Description du code :

1ière partie du code : Structure des données brutes

Dans une première partie on importe les données à partir du fichier texte « Achat.txt ». Elles sont stockées dans les variables **`Entrees`** et **`Sorties`**. La variable **`Entrees`** contient les paramètres d'entrée (Salaire et âge) La variable **`Sortie`** contient la cible (label) : 0 = pas d'achat, 1 = achat.

Ces données sont séparées en deux parties. Les entrées et sorties d'apprentissage : **`Entr_Appr`** et **`Sort_Appr`** d'une part et les entrées et sorties de test : **`Entr_Test`** et **`Sort_Test`** d'autre part.

Les données d'apprentissage permettront de construire la régression logistique et les données de test d'évaluer la pertinence de cette régression.

Une fonction **`affichage_donnees_3D(entrees,sorties)`** permet d'afficher la répartition des données dans un graphique 3D. Plan (Salaires,Age) pour les entrées et axe (Achat) les sorties.

2ième partie du code : Mise en forme des données pour la régression polynômiale

Dans cette partie on retrouve la fonction **`Mat_donnees(entrees_brutes,L_degres)`** quasi identique à celle du code de la régression linéaire et qui permet d'obtenir la matrice des m données qui permettra de construire une régression polynômiale :

$$Xd = \begin{bmatrix} 1 & x_s^0 & (x_s^0)^2 & .. & (x_s^0)^{n_S} & x_a^0 & (x_a^0)^2 & .. & (x_s^0)^{n_A} \\ 1 & x_s^1 & (x_s^1)^2 & .. & (x_s^1)^{n_S} & x_a^1 & (x_a^1)^2 & .. & (x_a^1)^{n_A} \\ 1 & x_s^2 & (x_s^2)^2 & .. & (x_s^2)^{n_S} & x_a^2 & (x_a^2)^2 & .. & (x_a^2)^{n_A} \\ .. & \dots & \dots & .. & \dots & \dots & \dots & .. & \dots \\ 1 & x_s^m & (x_s^m)^2 & .. & (x_s^m)^{n_S} & x_a^m & (x_a^m)^2 & .. & (x_a^m)^{n_A} \end{bmatrix} \qquad \text{dimensions} = m\times(1 + n_S + n_A)$$

On a également la fonction **`normalisation_donnes(donnees)`** qui permet d'avoir une matrice **`Xn`** de même dimension que **`Xd`** . Matrice **`Xn`** dont toutes les données sont normalisées par colonne. Chaque colonne (sauf la première qui n'a que des 1) à une moyenne égale à zéro et un écart type égal à 1.

Et enfin la fonction **`Init_vecteur_parametres(X)`** qui permet d'initialiser le vecteur colonne **`Winit`** des paramètres ω_i pour i allant de 0 à $n_S + n_A$.

3ième partie du code : Le code de la descente de gradient

On retrouve comme pour le code de la régression linéaire du TD précédent la fonction **`Regression(Entrees,Sorties,L_degre,Nbr_iter,alpha,Epsilon)`**. Cette fonction applique l'algorithme de la descente de gradient. Elle prend en argument les données d'apprentissage : **`Entr_Appr`** et **`Sort_Appr`** la liste de des degrés du polynôme n_S et n_A : **`L_degre`** le nombre maximal d'itérations : **`Nbr_iter`** , la vitesse d'apprentissage : **`alpha`** et la valeur minimale de variation du coût en deça de laquelle l'algorithme de la descente de gradient sera stoppée : **`Epsilon`** .

On retrouve dans cette fonction de la ligne de code qui constitue le cœur de l'algorithme de la descente de gradient (1ière ligne de la boucle **`while`**) : **`Wn=Wn-alpha*gradient(Xn,Wn,Sorties)`** et la boucle **`for`** qui permet de retrouver les paramètres ω_i de la régression logistique (vecteur **`W`**) s'appliquant sur la matrice des données d'apprentissage **`Xd`** à partir des paramètres $\overset{*}{\omega}_i$ de la régression logistique (vecteur **`Wn`**) construite avec la matrice des données normalisées **`Xn`**.

Cette fonction utilise les trois fonctions : **`sig(X,W)`** pour le calcul de $\sigma(X,\theta)$, **`Cout(X,W,Y)`** pour le calcul le coût $J(X,\theta)$ d'une régression avec le vecteur paramètres θ (vecteur **`W`**) la matrice des données normalisées (matrice **`X`**) et le vecteur des cibles (vecteur **`Y`**) ainsi que la fonction **`gradient(X,W,Y)`** qui calcule le gradient de descente : $\frac{\partial J(\theta)}{\partial \theta}$. **Nous allons compléter le code des ces trois fonctions.**

4ième partie du code : Affichages du résultat de la régression

On a deux fonctions **`Ecriture_regression`** et **`trace_regression_3D`** qui permettent d'afficher les résultats de la régression dans le Shell (paramètres ω_i) et de tracer dans un graphique 3D la régression logistique (surface représentative de la fonction $f(x_S,x_A)$) et la répartition des données test.

5ième partie du code : Evaluation de la régression sur les données test

Dans cette partie on a une fonction qui va permettre de déterminer la matrice de confusion pour évaluer les performances d'une régression logistique à partie des données test. **Nous allons compléter le code des cette dernière fonction**.

Implémentation du code manquant :

Question 3 - Compléter dans Pyzo le code python des trois fonctions : **`sig(X,W)`** , **`Cout(X,W,Y)`** et **`gradient(X,W,Y)`**.

Test de l'algorithme :

Tester votre code avec les commandes :

1 : **`Regression(Entr_Appr,Sort_Appr,[1,0],50000,0.5,1e-7)`**

2 : **`Regression(Entr_Appr,Sort_Appr,[0,1],50000,0.5,1e-7)`**

Question 4 - Que prend en compte chacune de ces régressions pour prédire l'achat (âge ou/et salaire) ? Quelles sont les limites d'âge ou de salaire qui déclenche l'achat ?

Question 5 - Ces deux régressions vous paraissent-elle suffisamment performantes ? (On rappelle qu'on souhaite avoir une réponse exacte dans plus de 95% des cas et on a 100 données test)

Tester le code avec les commandes :

3 : **`Regression(Entr_Appr,Sort_Appr,[1,1],50000,0.5,1e-7)`**

Question 6 - Que prend en compte cette régression pour prédire l'achat ? Donner l'inéquation sur les paramètres x_S et x_A qui permet de prédire le déclenchement de l'achat ?

Question 7 - Cette régression vous parait-elle suffisamment performantes ?

Tester le code avec les commandes :

4 : **`Regression(Entr_Appr,Sort_Appr,[2,2],50000,0.5,1e-7)`**

Question 8 - Cette régression vous parait-elle suffisamment performantes ?

4- Evaluation des performances de l'algorithme

Pour s'assurer que la prédiction est exacte dans plus de 95% des cas nous allons compter le nombre de prédictions exactes et afficher les résultats dans une matrice de confusion. Le principe est le suivant :

On dispose d'un jeu de données d'apprentissage (**`Entrees_Appr`** et **`Sorties_Appr`**) permettant d'établir un modèle de prédiction. Nous allons maintenant utiliser le jeu de données test (**`Entrees_Test`** et **`Sorties_Test`**). Sur ce jeu de test nous allons comparer les prédictions du modèle établi y_p et les sorties réelles y (les labels ou cibles). Nous allons pour cela construire la matrice de confusion.

Pour les prédictions y_p on utilise des entrées des données test (**`Entrees_test`**) et on calcule avec le modèle établi (vecteur θ) :

Si $\sigma(x,\theta) >= 0{,}5$ alors $y_p = 1$ sinon $y_p = 0$.

Pour les labels y on a le vecteur **`Sorties_test`**.

		Prédictions	
		$y_p = 0$	$y_p = 1$
labels	y = 0	n_{00}	n_{01}
	y = 1	n_{10}	n_{11}

Dans cette matrice les n_{ij} sont les nombres d'occurrences sur les données test où les valeurs des sorties (labels) sont i et les valeurs des prédictions sont j.

C'est la fonction **`Matrice_de_confusion(entrees_t,sortie_t,L_degre,W)`** qui permet de construire cette matrice de confusion.

Description du code de cette fonction de construction de la matrice de confusion :

Cette fonction prend en argument les données test : **`entrees_t`** et **`sortie_t`** , la liste des degrés de la fonction polynômiale $f(x_S,x_A)$: **`L_degre`** et le vecteur des coefficients ω_i des la régression logistique : **`W`** . Elle retourne la matrice de confusion **`MatConf`** sous la forme d'un tableau numpy.

On a en premier lieu une sous fonction **`polynome(xs,xa)`** qui prend en argument le salaire **`xs`** et l'âge **`xa`** et qui retourne la valeur de la fonction polynômiale $f(x_S,x_A)$. Cette sous fonction utilise les paramètres ω_i du vecteur colonne W .

On initialise la matrice de confusion : **`MatConf=np.zeros((2,2))`** . Puis on parcourt toutes les données test. Pour chacune des ces données :

On mémorise dans la variable **`label`** la valeur y de sortie de la données test. Attention cette valeur doit impérativement être un entier 0 ou 1 (utiliser la fonction Python : **`int`**).

On calcule et on mémorise dans la variable **`proba`** la valeur $y_P = \sigma((x_S,x_A),\theta)$. Puis si y_P (variable **`proba`**) est supérieure ou égale à 0,5 on mémorise dans la variable **`predi`** la valeur entière 1 : **`predi=int(1)`** sinon on mémorise **`predi=int(0)`**.

On incrémente la valeur correspondante de la matrice de confusion (ligne y = 0 ou 1 et colonne $y_P = 0$ ou 1) avec la commande : **`MatConf[label,predi]+=1`**.

Question 10 - Compléter les deux premières lignes de la boucle **`for`** : « **`label =`**… » et « **`proba =`**… ».

Question 11 - .Décommenter l'avant dernière ligne de la fonction Regression qui permet de construire la matrice de confusion et l'afficher dans le Shell. Puis tester la fonction **`Matrice_de_confusion`** en lançant la commande **`Regression(Entr_Appr,Sort_Appr,[`**n_S**`,`**n_A**`],50000,0.5,1e-7)`** avec plusieurs valeurs de n_S et n_A de 1 à 4. Conclure sur les performances des régressions obtenues.

Question 11 - Proposer des solutions pour améliorer les performances de l'algorithme.