Machine Learning TP2: Régression logistique

Melvin DUBEE - Tanguy ROUDAUT FIPASE 24

16 octobre 2023

Vous pouvez trouver nos codes dans les dossiers "code" et "code2" et les résultats dans le dossier "output" qui se trouve à la racine de l'archive.

1 Régression logistique

1.1 Affichage des données

Une première fonction *plotData()*, qui permet d'afficher un graphique 2D avec les axes représentant les deux notes des examens, et les exemples positifs et négatifs affichés avec différents marqueurs.
L'objectif de cette partie sera de construire un modèle de régression logistique pour prédire si un étudiant est admis dans une université en comprenant sa méthodologie.

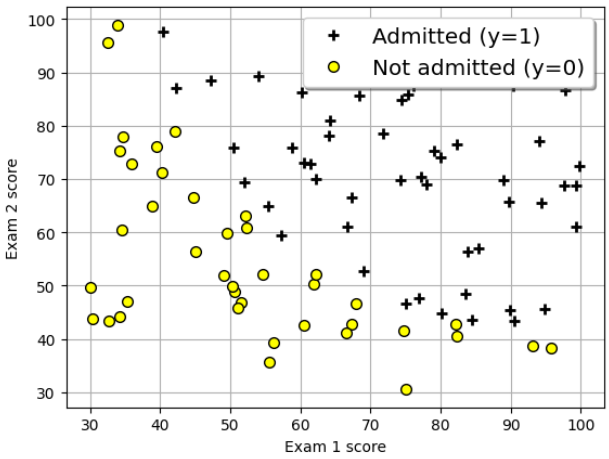

FIGURE 1 – Diagramme de dispersion des données d'entraînement

```
1    def plotData(X,y):
2            pos = X[(y==1).flatten(),:]
3            neg = X[(y==0).flatten(),:]
4            plt.plot(pos[:,0], pos[:,1], '+', markersize=7, markeredgecolor='black',
             ↪  markeredgewidth=2)
5            plt.plot(neg[:,0], neg[:,1], 'o', markersize=7, markeredgecolor='black',
             ↪  markerfacecolor='yellow')
6            plt.legend(['Admitted (y=1)', 'Not admitted (y=0)'], loc='upper right', shadow=True,
             ↪  fontsize='x-large', numpoints=1)
7            plt.grid()
8            plt.xlabel('Exam 1 score')
9            plt.ylabel('Exam 2 score')
```

FIGURE 2 – Fonction plotData

1.2 Descente de gradient

Le modèle de régression linéaire est représenté par l'équation 1. Cette équation nous permet d'obtenir une prédiction en fonction d'une entrée x et de θ.

$$h_\theta(x) = g(x^T\theta) \qquad \text{avec} \qquad g(z) = \frac{1}{1 - e^{-z}} \tag{1}$$

```
1         def sigmoid(z):
2         """computes the sigmoid of z."""
3             g = 1 / (1 + np.exp(-z))
4
5         return  g
```

FIGURE 3 – Fonction sigmoïd

Pour que cette prédiction soit optimale, il est important de déterminer correctement les paramètres de notre modèle : θ. Pour cela, nous devons réaliser deux étapes : Le calcul du coût $J(\theta)$ et une descente de gradient.

1.2.1 Calcul du coût $J(\theta)$

De la même manière que dans le TP1, le calcul du coût $J(\theta)$ permet de mesurer la qualité de la prédiction, si le coût est faible alors notre prédiction est proche des valeurs réelles et inversement si le coût est important. La formule utilisée pour calculer ce coût n'est pas la même pour une régression linéaire que pour une régression logistique.

$$J(\theta) = \frac{1}{m}\sum_{i=0}^{m-1}[-y^{(i)}log(h_\theta(x^{(i)})) - (1 - y^{(i)})log(1 - h_\theta(x^{(i)}))] \tag{2}$$

On remarque ici avec l'équation 3 qui utilise l'équation 1, que la seule valeur qui puisse influencer notre coût est θ. Effectivement, les valeurs restantes : x, y et m ; sont les valeurs de notre problème qui sont déterminées et non modifiables.

Mise en application

```
1  def costFunction(theta, X, y):
2      # Initialize some useful values
3      m,n = X.shape
4      theta = theta.reshape((n,1))
5
6      predictions = sigmoid(X @ theta)
7      J = (1/m) * np.sum(-y * np.log(predictions) -
       ↪  (1 - y) * np.log(1 - predictions))
8
9      return  J
```

Listing 1 – Fonction computeCost

```
Cost at initial theta (zeros): 0.693147
Expected cost (approx): 0.693
```

Listing 2 – Output fonction computeCost

1.2.2 Descente de gradient

La descente de gradient permet de minimiser le coût et donc d'obtenir les bonnes valeurs de theta pour réaliser une prédiction optimale.

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{3}$$

Cette formule pour calculer le gradient de la régression logistique comme pour le coût n'est pas identique à celle de la régression linéaire, en effet cela est dû aux définitions différentes de $h_\theta(x)$.

Mise en application

```python
def gradientFunction(theta, X, y):
        m = X.shape[0]
        n = X.shape[1]
        theta = theta.reshape((n,1))
        grad = 0
        for i in range(m):
            grad += (1/m) * (sigmoid(X[i] @
            ↪  theta) - y[i]) * X[i]

        return grad
```

Listing 3 – Fonction gradientFunction

```
theta: ['-25.1613', '0.2062', '0.2015']
Expected theta (approx): -25.161 0.206 0.201

 --------------------------

For a student with scores 45 and 85, we
↪  predict an admission probability of
↪  0.776291
Expected Proba (approx): 0.776

 --------------------------

Train Accuracy: 89.000000
Expected accuracy (approx): 89.0%
```

Listing 4 – Output fonction gradientFunction