Master SID
Année universitaire 2020-2021

TP2 Apprentissage Automatique 2

Régression logistique

simon.bernard@univ-rouen.fr

1 Problème jouet

Pour ce premier exercice, vous allez estimer et appliquer une modèle de regression logistique sur un problème "jouet". Ce problème comprend des points étiquetés $(\mathbf{x}, y_i) \in \mathbb{R}^2 \times \{\lambda_1, \lambda_2\}$, et les distributions des classes suivent une loi normal, i.e. $\lambda_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ et $\lambda_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$.

- ▷ Générez cette base de données synthétiques telle que :
  - chaque classe contient 300 points
  - $\lambda_1 \sim \mathcal{N}((1,1), 0.7)$ et $\lambda_2 \sim \mathcal{N}((-1,-1), 0.7)$

- ▷ Affichez le nuage de points pour obtenir un résultat similaire à celui de la figure 1

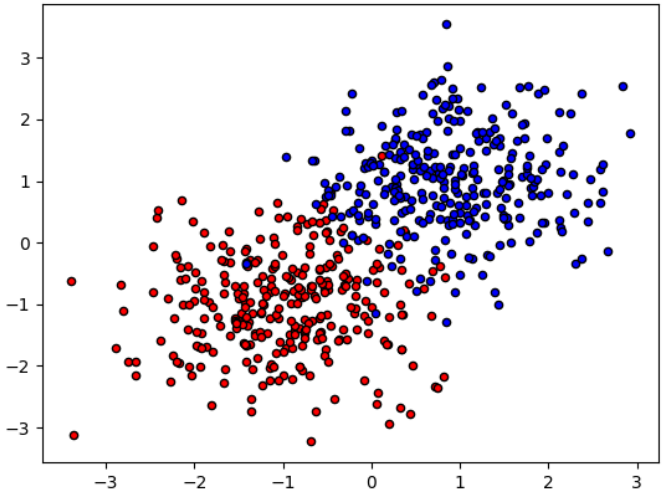

FIGURE 1

Avant d'implémenter votre propre estimation du modèle de regression logistique, vous allez tester celui de *scikit-learn* :

- ▷ Importez la modèle de régression logistique de *scikit-learn* [1] et entrainez le sur la base synthétique que vous venez de créer
- ▷ Affichez la fonction de décision du modèle, comme montré en figure 2. Pour cela, il faut :
  1. Générer une grille de points 2D à l'aide de la fonction `mesh` que nous donnons ci-dessous.
  2. Rassembler les prédictions du modèle pour chacun des points de cette grille.
  3. Afficher le nuage de points de la base d'apprentissage
  4. Afficher les frontières de décisions en utilisant la fonction `contourf` de *Matplotlib*.

```
def mesh(X, h = 0.02):
   x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
   y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
   xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
   return np.c_[xx.ravel(), yy.ravel()], xx, yy
```

N'hésitez pas à vous aider des exemples qui sont fournis dans la documentation de *Scikit-learn*.

2 Régression logistique sur le problème jouet

Vous allez maintenant implémenter votre propre modèle de regression logistique ; Pour cela :

1. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

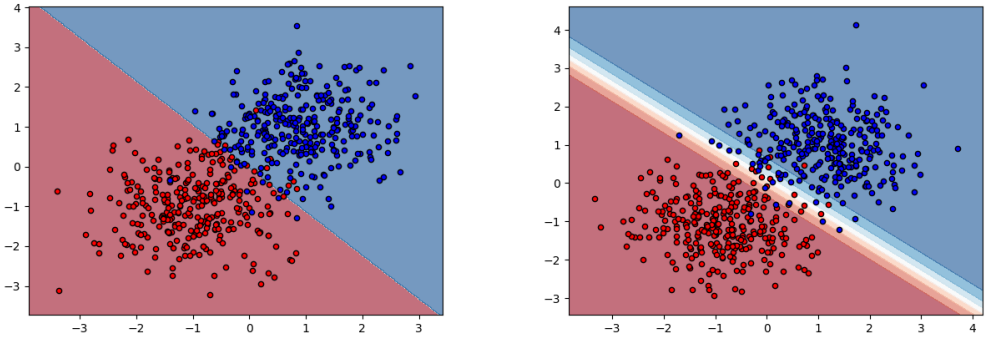

FIGURE 2 – À gauche : la fonction de décision si les prédictions sont des classes. À droite : la fonction de décision si les prédictions sont des probabilités d'appartenance à la classe λ_1

▷ Implémentez le calcul du coût en complétant la fonction ci-dessous ci-dessous :

```
def logreg_cost(X, y, w):
    z = X@w
    # insérer votre code ici
    return cost
```

▷ Implémentez l'estimation des paramètres du modèle de regression logistique en utilisant la méthode de descente de gradient :

```
def logreg_grad_desc(X, y):
    w = np.random.randn(d)
    nb_iter = 100  # to tune
    pas = 0.01  # to tune
    beta = 0.9  # for backtracking if needed - to tune
    for i in range(nb_iter):
        z = X @ w
        # insérer votre code ici
        w = w - pas * grad
    print(w)
```

▷ Affichez la fonction de décision et proposez un protocole judicieux permettant de tester la performance de votre modèle

3 Diagnostique médical

On cherche à prédire la survenue d'un cancer à partir de paramètre médical. Pour cela, vous allez utiliser la base de données *breast_cancer* de *Scikit-learn*.

▷ Proposez une méthodologie valide pour comparer votre algorithme de régression logistique à l'implémentation de *scikit-learn*. Les résultats seront présentés avec des taux de bonne classification moyen et des écart-types.

▷ Comparez maintenant avec un algorithme la regression logistique régularisée en utilisant l'implémentation de *Scikit-learn*

▷ Poursuivez les expérimentations en 'jouant' avec les autres paramètres de l'algorithme de *Scikit-learn*

▷ Analysez et commentez les résultats de chaque test