TP1 : apprentissage artificiel avec Numpy, pandas et Scikit-Learn, Ndeye Fatou NGOM (DIC1 2018-2019)

Exercice 1 *Equation Normale*
Considérons l'équation de prédiction d'un modèle de régression linéaire suivante

$$\hat{y} = h_\theta(x) = \theta^T.x = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$$

où θ est le vecteur des paramètres du modèle, θ^T transposé de θ, x vecteur des valeurs d'une observation, $\theta^T.x$ est le produit scalaire de θ^T et x et h_θ est la fonction hypothèse utilisant les paramètres θ.

1. *Pour tester ce modèle, générer les données à l'allure linéaire suivante*

   $$y = 4 + 3x + \textit{bruit gaussien}$$

   où x est un jeu de données aléatoires
2. *Nous souhaitons entraîner le modèle en utilisant la librairie **Numpy** et la fonction coût suivante*

   $$MSE(x, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T.x^i - y^i\right)^2$$

   où la solution analytique de la valeur $\hat{\theta}$ qui minimise la fonction coût est donnée par l'équation normale suivante

   $$\hat{\theta} = (X.X)^{-1}.X^T.y$$

   avec y le vecteur des valeurs cibles y^1 à y^m.
   (a) *Calculer θ à l'aide de l'équation normale.*
   (b) *En déduire le temps de calcule et les paramètres θ de ce modèle.*
   (c) *Interpréter les résultats obtenus.*
   (d) *Réaliser des prédictions du modèles.*
   (e) *Représenter graphiquement les prédicitions du modèle.*
   (f) *Maintenant générer des données non linéaires.*
      i. *Faite une régression linéaire,*
      ii. *Afficher le résultat sur un graphe.*
      iii. *Interpréter le résultat obtenu.*
3. *On considère pour la suite la libraire Scikit-Learn.*
   (a) *Donner le code équivalent de la régression linéaire pour le modèle précédent.*
   (b) *Générer un jeu de données non linéaire avec l'équation qui suit*

   $$y = 0.5x_1^2 + 1.0x_1 + 2.0 + \textit{bruit gaussien}$$

   (c) *Implementer et tester sur des données aléatoires la régression polynomiale : faire une polynomialisation (tranformation des données d'apprentissage : ajout des puissances de chacune des variables comme nouvelles variables) avec PolynomialFeatures, puis entraîner un modèle linéaire sur ce nouvel ensemble de données.*

4. *Les courbes d'apprentissage sont des diagrammes représentant les résultats obtenus par le modèle sur le jeu d'entraînement et le jeu de validation en fonction de la taille du jeu d'entraînement. Pour générer ces graphiques, il suffit d'entraîner le modèle plusieurs fois sur des sous-ensembles de différentes tailles du jeu d'entraînement.*
   (a) *Définir une fonction qui trace les courbes d'apprentissage,*
   (b) *Générer les courbes d'apprentissage pour un modèle de régression linéaire simple et pour un modèle polynomial.*
5. *Trouver la polynomialisation qui permet d'éviter le sur-apprentissage tout en garantissant un apprentissage s'appelle la* ***régularisation d'un modèle****. Parmi les modèles de régularisation figure la régression Ridge ou régularisation de Tikhonov, la régression LASSO (least absolute Shrinkage and Selection Operator Regression), l'Elastic Net ou le Early Stopping. Pour chacune de ces méthodes*
   (a) *rappeler l'idée générale,*
   (b) *rappeler la fonction coût,*
   (c) *implémenter et tester le modèle.*

Exercice 2 *Descente de gradient*
La recherche d'optimum avec des solutions directes comme l'équation normale est souvent une tâche très fastidieuse. Une solution le plus souvent plus efficace est la Descente de Gradient (DG). La descente de gradient est un algorithme d'optimisation qui permet de trouver des solutions optimales en corrigeant petit à petit les paramètres dans le but de minimiser une fonction coût.

1. *Discuter sur l'impact de la dimension des pas (taux d'apprentissage) et de la normalisation des données pour la DG.*
2. *Générer des données aléatoires X.*
3. *Normaliser les données en vous inspirant du script suivant*

```
# Numpy
X_norm=(X-X.mean(axis=0))/X.std(axis=0)
# StandardScaler de ScikitLearn
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_norm=scaler.fit_transform(X)
```

4. *Nous considérons la descente de gradient ordinaire avec la fonction coût d'un modèle de régression linéaire. Nous supposons que le vecteur gradient de la fonction coût est donnée par le vecteur qui suit*

$$\nabla_\theta MSE(\theta) = \begin{pmatrix} \frac{\partial}{\partial\theta_0} MSE(\theta) \\ \frac{\partial}{\partial\theta_1} MSE(\theta) \\ . \\ . \\ \frac{\partial}{\partial\theta_n} MSE(\theta) \end{pmatrix} = \frac{2}{m} X^T.(X.\theta - y)$$

où

$$\frac{\partial}{\partial\theta_j} MSE(\theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^T.x^i - y^i\right) x_j^i$$

Une fois que l'on a le vecteur de gradient, il suffit d'aller dans la direction opposée pour descendre. Ainsi le pas de descente de gradient est donnée par

$$\theta_{EtapeSuivante} = \theta - \eta\nabla_\theta MSE(\theta)$$

où η est le taux d'apprentissage.

   (a) *Implémenter l'algorithme de descente de gradient ordinaire avec un nombre d'itération de l'algorithme fixé à 1000.*

   (b) *En vous basant sur les graphes et les temps de calcul, interpréter les résultats obtenus avec les taux d'apprentissage suivantes : $\eta = 0.02$, $\eta = 0.1$ et $\eta = 0.5$.*

   (c) *En testant plusieurs nombre d'itération différentes, interpréter l'impact du nombre d'itération sur le résultat obtenu.*

   (d) *Modifier l'algorithme de manière à choisir un très grand nombre d'itérations, et interrompre l'algorithme lorsque le vecteur de gradient devient très petit ie quand sa norme devient inférieure à une tolérance epsilon ; ce qui signifie que la descente de gradient atteint presque son minimum.*

   (e) *Rappeler les avantages et inconvénient, puis implémenter et tester les méthodes de descente de gradient suivantes*

      i. *Descente de gradient stochastique,*

      ii. *Descente de gradient par mini-lots.*