Régression linéaire :

* Fonction de prédiction :

predict(x) = theta0 + (theta1 \* x)

* Fonction de coût :

(m = la len des données)

m

J(theta0, theta1) = 1/2m E (predict(xi) - yi)2

i=0

predict(x) étant la fonction qui va prendre les thetas pour prédire :

Donc la fonction de coût:

m

J(theta0, theta1) = 1/2m E ((theta0 + (theta1 \* xi)) - yi)2

i=0

Les dérivées partielles sont :

(lr = alpha)

m

lr/lr\*theta\_0 J(theta\_0, theta\_1) = 1/m E (h(xi) - yi)

i=0

m

lr/lr\*theta\_1 J(theta\_0, theta\_1) = 1/m E (h(xi) - yi) \* xi

i=0

Donc calcul theta\_0, theta\_1:

m

theta\_0 = lr \* 1/m E (predict(xi) - yi)

i=0

m

theta\_1 = lr \* 1/m E (predict(xi) - yi) \*xi

i=0

Par comparaison, régression logistique

* Fonction de prédiction:

predict(x) = g(theta.T \* x)

avec:

g(z) = 1 / (1+e\*\*-z)

Donc **predict(x) = 1 / (1 + exp(-(theta.T \* x))**

-> theta: un par classifier classifier.

**-> T correspond au transpose d’une matrice comme le fait numpy**

**-> e means np.exp**

* Fonction de coût:

m

J(theta) = -1/m E yi \* log(predict(xI)) + (1 - yi)log(1 - predict(xi))

i=1

-> pour 4 thetas.

* La dérivée partielle est :

m

lr/lr\*thetaj J(theta) = 1/m E (predict(xi) - yi) \* xj \*\*i

i=1

j représente ici le classifieur d’une classe : thetaj et xi ==> classifieur gryffondor et data gryffondor.

* donc pour calculer un theta:

m

thetaj = lr \* 1/m E (predict(xi) - yi) \* xj \*\*i

i=1

4 x 13 thetas: 1 theta par classe par feature.

On crée un classifieur par feature pour chacune des maisons. C’est la méthode du one vs all.

Nos classifiers sont des régressions logistiques :

f1(x) = regression log sur x.

f2(x) = regression log sur x.

…

Il est logique de prédire que x appartient à la classe pour laquelle cette prédiction est la plus élevée.

On prédit la classe pour laquelle la fonction de décision retourne la valeur la plus élevée

C’est ce qu’on va dans le predictor : pour chaque student, on envoie nos 13 thetas par maison dans la fonction de prediction, donc on envoie 4 fois la prédiction avec set de 13 thetas différent.

Puis on choisit la max : f1, f2, f3, f4.

Avant cela, il faut avoir défini les 4 set de thetas.

Pour ça on fait donc un entraînement.

Pour faire l’entraînement, on normalise les données:

* pour chaque student
  + pour chaque cours
    - on prend les notes et on normalize selon mean / std du course.
    - on ajoute cela à l’array X qui va contenir toutes les notes.
* en parallèle, on ajoute l’index de la maison du student dans un array Y.

on remplace tous les nans par 1.

(il faut faire pareil pour la prédiction, à l’exception des Y).

Puis on lance le training :

On va avoir un nb de thetas qui correspond au nb de course.

Puis, on va lancer 4 fois un training (1 par maison) ->

- on reconstruit un tableau de Y qui ne pas plus contenu le numéro de la maison, mais un booleen qui nous informe si tel student appartient à la maison pour laquelle on a lancé le training (1), ou l’autre.  
 - Puis on lance 200 fois le training et à chaque fois on réajuste des thetas temporaires

- chacune de ces itérations va constituer un gradiant descent -> à chaque fois on va déduire des thetas temporaires un gradiant qu’on calcule comme ça :

- on run la prédiction avec les thetas temporaires m fois, avec à chaque fois X[1/m]

- on déduit de ces prédictions les valeurs Y qu’on a trouvées : en fonction de si c’est la maison pour laquelle on run l’algorithme ou non - c’est la partie xi - yi dans l’algo.

- on multiplie matrice de X avec matrice de prédictions (plutôt que de faire un par un à chaque itération et d’additionner : on peut faire comme ça avec les matrices, ça revient à additionner chaque couple de multiplication XY).

- on inverse on divisant par m.

Puis on multiplie par learning rate.

Percentile :

Ordonner data.

Trouver un rank index qui correspond au % en fonction de la taille du dataset.

on arrondi au dessus et en dessous -> si c’est le mêmerésultat, alors on renvoie la valeur du dataset qui est à l’index rank.

sinon on prend les valeurs qui correspondent aux deux indexs des arrondis, on les mulitplie par la différence de ces arrondis avec le rank, et on les additionne.