Master SID
Année universitaire 2021-2022

TP3 Apprentissage Automatique 2

Régression logistique multiclasse

simon.bernard@univ-rouen.fr

Le but de ce TP est d'implementer un coût *softmax*, qui sert à étendre la régression logistique pour la classification multiclasse. Dans ce contexte, la modélisation de la probabilité d'appartenance à une classe est donnée par :

$$p(y = \lambda_k|\mathbf{x}) = \frac{\exp(\mathbf{x}^\top \mathbf{w}_k)}{\sum_{j=1}^{C} \exp(\mathbf{x}^\top \mathbf{w}_j)}$$

où C est le nombre de classes, et $\mathbf{w}_k$ est un vecteur de $\mathbb{R}^d$. On appelle cette fonction, la fonction *softmax* [1].

En considérant que $y_{i,j}$ représente la probabilité d'appartenance à une classe λ_j de l'exemple $\mathbf{x}_i$, la fonction de coût *cross-entropy* s'écrit :

$$L(y_i, \hat{y}_i) = -\sum_{j=1}^{C} y_{i,j} \log(\hat{y}_{i,j})$$

où $\hat{y}_{i,j} = p(y = \lambda_k|\mathbf{x}_i)$ est modélisé par la fonction *softmax* ci-dessus.

Vous allez implémenter dans ce TP, un modèle linéaire multiclasse appris tel que :

$$\min_{\mathbf{W}} \quad \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i) + \frac{\lambda}{2}\sum_{i,j} \mathbf{w}_{i,j}^2$$

où $\mathbf{W}$ représente une matrice de taille $d \times C$ contenant l'ensemble des $\{\mathbf{w}_k\}$

▷ Commencez par charger les données relatives à un problème de reconnaissance de chiffres manuscrits :

```
import numpy as np
from sklearn.datasets import load_digits
n_class = 10
X,y = load_digits(n_class=n_class, return_X_y=True)
```

▷ Implémentez les fonctions permettant :

- de transformer les étiquettes des données en vecteurs de probabilités d'appartenance aux classes, à l'aide d'un encodage *one-hot* [2]. Bien entendu, pour les données d'apprentissage, ces probabilités sont soit égales à 1 (pour la classe à laquelle appartient la donnée), soit égale à 0 (pour les autres classes).
- d'évaluer une fonction *softmax* étant donné un $\mathbf{z} = \mathbf{x}^\top \mathbf{w}$
- d'estimer la probabilité d'appartenance d'un ensemble de données étant donné $\mathbf{W}$.

```
def oneHotEncodage(y,n_class):
   # insérer votre code ici
   return y_one

def softmax(z):
   z -= np.max(z)  # computational trick for numerical stability
   # insérer votre code ici
   return sm

def get_prob_pred(X,W):
   # insérer votre code ici
   return probs, preds
```

▷ Implémentez maintenant une fonction qui calcule la fonction de coût et le gradient.

```
def get_loss_grad(W,X,y,lam,n_class):
   y_mat = oneHotEncodage(y,n_class) # convert the integer class coding into a
       one-hot representation
   scores = X@W  # compute raw class scores given our input and current weights
   prob = softmax(scores) # perform a softmax on these scores to get their
       probabilities
   # insérer votre code ici
   return loss, grad
```

1. https://fr.wikipedia.org/wiki/Fonction_softmax
2. https://fr.wikipedia.org/wiki/Encodage_one-hot

▷ Implémentez une descente de gradient avec *backtracking* pour optimiser les paramètres $\mathbf{W}$.

▷ Évaluez l'erreur en classification sur les données d'apprentissage après entrainement

▷ Affichez l'évolution de la fonction objective après chaque itération

Aide pour le calcul du gradient

Pour l'exercice ci-dessus, il vous faut calculer le gradient du coût softmax. Ce coût s'exprime comme :

$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{i,j}\log(p_{i,j})$$

où

- n est le nombre d'instances d'apprentissage
- C est le nombre de classes
- $p_{i,k}$ est la probabilité de l'instance i d'appartenir à la classe k exprimé par la fonction *softmax* :

$$p_{i,k} = p(y = \lambda_k|\mathbf{x}_i) = \frac{\exp(z_{i,j})}{\sum_{l=1}^{C}\exp(z_{i,l})}$$

avec $z_{i,j} = \mathbf{x}_i^\top \mathbf{w}_j$

On s'intéresse au gradient de L en fonction de $\mathbf{w}_k$, les paramètres du modèle pour la classe λ_k :

$$\begin{aligned}
\nabla_{\mathbf{w}_k} L &= -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} \frac{\partial y_{i,j}\log(p_{i,j})}{\partial \mathbf{w}_k} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{i,j}\frac{\partial \log(p_{i,j})}{\partial \mathbf{w}_k} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{i,j}\frac{1}{p_{i,j}}\frac{\partial p_{i,j}}{\partial \mathbf{w}_k}
\end{aligned}$$

Le plus simple ici est d'utiliser la règle de dérivation en chaîne [3] qui nous donne

$$\nabla_{\mathbf{w}_k} L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{i,j}\frac{1}{p_{i,j}}\frac{\partial p_{i,j}}{\partial z_{i,k}}\frac{\partial z_{i,k}}{\partial \mathbf{w}_k}$$

On obtient facilement :

$$\frac{\partial z_{i,k}}{\partial \mathbf{w}_k} = \mathbf{x}_i$$

Et pour $\frac{\partial p_{i,j}}{\partial z_{i,k}}$ il faut distinguer deux situations : quand $j \neq k$ (1) et quand $j = k$ (2).

Il suffit pour finir d'assembler toutes les pièces du puzzle :

$$\nabla_{\mathbf{w}_k} L = -\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\left((2) + \sum_{j=1, j\neq k}^{C} (1)\right)$$

Charge à vous de calculer (1) et (2).

3. `https://en.wikipedia.org/wiki/Chain_rule`