Minimax de l'entropie conditionnelle régularisée pour le crowdsourcing

Anne Bernard

15 Septembre 2023

Sommaire

1. Introduction

Introduction

Crowdsourcing = externalisation par la foule.

De plus en plus de services de crowdsourcing à moindre coût... mais à quel prix?

$\Longrightarrow$ Méthode de Dawid et Skene

2. Dawid et Skene

Méthode de Dawid et Skene

Modèle d'agrégation probabiliste paramétrant le niveau d'expertise par des matrices de confusions.

- σ_w : matrice de confusion du travailleur w de taille $n_c \times n_c$ où n_c est le nombre de classes

Méthode de Dawid et Skene

Modèle d'agrégation probabiliste paramétrant le niveau d'expertise par des matrices de confusions.

- σ_w : matrice de confusion du travailleur w de taille $n_c \times n_c$ où n_c est le nombre de classes
- Y_i la vraie étiquette de l'item i

Méthode de Dawid et Skene

Modèle d'agrégation probabiliste paramétrant le niveau d'expertise par des matrices de confusions.

- σ_w : matrice de confusion du travailleur w de taille $n_c \times n_c$ où n_c est le nombre de classes
- Y_i la vraie étiquette de l'item i
- X_{wi} la classe que le travailleur w a attribué à l'item i

Méthode de Dawid et Skene

Modèle d'agrégation probabiliste paramétrant le niveau d'expertise par des matrices de confusions.

- σ_w : matrice de confusion du travailleur w de taille $n_c \times n_c$ où n_c est le nombre de classes
- Y_i la vraie étiquette de l'item i
- X_{wi} la classe que le travailleur w a attribué à l'item i
- p le vecteur de probabilité des classes à priori, $P(X_{wi} = c) = p[c]$

Méthode de Dawid et Skene

Modèle d'agrégation probabiliste paramétrant le niveau d'expertise par des matrices de confusions.

- σ_w : matrice de confusion du travailleur w de taille $n_c \times n_c$ où n_c est le nombre de classes
- Y_i la vraie étiquette de l'item i
- X_{wi} la classe que le travailleur w a attribué à l'item i
- p le vecteur de probabilité des classes à priori, $P(X_{wi} = c) = p[c]$

Problème : même confusion pour tous les items

3. Principe du minimax

Principe du minimax

Chaque travailleur est indexé par w, chaque item par i et les classes par c ou v. Soit x_{wi} l'étiquette observée que le travailleur w à assigner à l'item i, X_{wi} la variable aléatoire correspondante et Y_i la variable associée à la classe de l'item i.

Principe du minimax

Chaque travailleur est indexé par w, chaque item par i et les classes par c ou v. Soit x_{wi} l'étiquette observée que le travailleur w à assigner à l'item i, X_{wi} la variable aléatoire correspondante et Y_i la variable associée à la classe de l'item i.

Soit $Q(Y_i = v)$ la vraie probabilité non-observée que l'item i soit dans la classe v. On note $P(X_{wi} = c | Y_i = v)$ la probabilité que le travailleur w étiquette l'item i dans la classe c sachant que la vraie classe est v. On cherche à estimer Q.

Principe du minimax

Chaque travailleur est indexé par w, chaque item par i et les classes par c ou v. Soit x_{wi} l'étiquette observée que le travailleur w à assigner à l'item i, X_{wi} la variable aléatoire correspondante et Y_i la variable associée à la classe de l'item i.

Soit $Q(Y_i = v)$ la vraie probabilité non-observée que l'item i soit dans la classe v. On note $P(X_{wi} = c | Y_i = v)$ la probabilité que le travailleur w étiquette l'item i dans la classe c sachant que la vraie classe est v. On cherche à estimer Q.

On ajoute τ_i dans le problème dual permettant de mesurer la difficulté pour un item d'être étiqueter.

Principe du minimax

BUT : minimiser sur Q le maximum sur P de l'entropie de X conditionnée par Y

$$\min_Q \max_P H(X|Y)$$

C'est optimal lorsque Q est déterministe. Mais en pratique on préfère avoir des étiquettes probabilistes. Et parfois le nombre de données n'est pas suffisant pour avoir un résultat proche de la réalité car il y a trop de bruit.

Principe du minimax

Il faut donc régulariser le minimax et on obtient :

Principe du minimax

Il faut donc régulariser le minimax et on obtient :

$$\max_{\sigma,\tau,Q} \sum_{i,v} Q(Y_i = v) \sum_{w} \log P(X_{wi} = x_{wi} | Y_i = v) + H(Y) - \alpha\Omega^*(\sigma) - \beta\Psi^*(\tau)$$

où

$$\Omega^*(\sigma) = \frac{1}{2}\sum_{w}\sum_{v,c}[\sigma_w(v, c)]^2,$$

$$\Psi^*(\tau) = \frac{1}{2}\sum_{i}\sum_{v,c}[\tau_i(v, c)]^2.$$

4. Implementation

Implementation

input : $\{x_{wi}\}, \alpha, \beta$

initialize :

$$Q(Y_i = v) \propto \sum_w 1(x_{wi} = v)$$

repeat :

$$\{\sigma, \tau\} = \arg\max_{\sigma,\tau} \sum_{w,i,v} Q(Y_i = v) \log P(X_{wi} = x_{wi} | Y_i = v) - \alpha\Omega^*(\sigma) - \beta\Psi^*(\tau)$$

$$Q(Y_i = v) \propto \prod_w P(X_{wi} = x_{wi} | Y_i = v)$$

output : Q

Implementation

Méthode de montée de gradient donc calcul des gradients :

$$\frac{\partial F}{\partial \sigma_w(v,c)} = \sum_i Q(Y_i = v)[1(x_{wi} = c) - P(X_{wi} = c | Y_i = v)] - \alpha\sigma_w(v,c),$$

$$\frac{\partial F}{\partial \tau_i(v,c)} = \sum_w Q(Y_i = v)[1(x_{wi} = c) - P(X_{wi} = c | Y_i = v)] - \beta\tau_i(v,c),$$

Etape de code

- Initialisation de Q avec les données

Etape de code

- Initialisation de Q avec les données
- Calcul des gradients

Etape de code

- Initialisation de Q avec les données
- Calcul des gradients
- Méthode des gradients pour σ et τ

Etape de code

- Initialisation de Q avec les données
- Calcul des gradients
- Méthode des gradients pour σ et τ
- Mise à jour de Q avec les nouveaux σ et τ

5. Exemples

Bluebirds

Jeu de données avec 38 travailleurs, 107 items et 2 classes.

10 itérations suffisent pour converger. On obtient une nouvelle distribution Q que l'on compare avec la vraie.

On a obtenu un pourcentage d'erreur de 15%. Plus faible que le résultat obtenu sur l'article (36%)

Bluebirds

```
sigma = torch.zeros((n_workers, n_classes, n_classes))
tau = torch.zeros((n_items, n_classes, n_classes))
num_iterations = 10
Q = dist_Q(bluebirds, n_classes)

for _ in range(num_iterations):
    sigma_up, tau_up = gradient_ascent(
        n_workers, n_classes, n_items, bluebirds, sigma, tau, Q)
    Q = update_Q(bluebirds, n_workers, n_classes, n_items,
                 sigma_up, tau_up)
    sigma = sigma_up
    tau = tau_up
```

Hammer/Spammer

Simulation d'un jeu de données grâce à **peerannot** avec 10 travailleurs, 100 items et 5 classes.

10 itérations également pour celui-ci. Nous obtenons un pourcentage d'erreur de 0%. En effet dans le jeu de données quasiment la totalité des travailleurs ont donné la bonne réponse, il a été facile pour l'algorithme d'éliminer les mauvaises réponses.

Hammer/Spammer

```
sigma = torch.ones((n_workers, n_classes, n_classes))
tau = torch.ones((n_items, n_classes, n_classes))
num_iterations = 10
Q = dist_Q(hammer_spammer, n_classes)

for _ in range(num_iterations):
    sigma_up, tau_up = gradient_ascent(
        n_workers, n_classes, n_items, hammer_spammer, sigma, tau, Q)
    Q = update_Q(hammer_spammer, n_workers, n_classes, n_items,
                 sigma_up,tau_up)
    sigma = sigma_up
    tau = tau_up
```

Conclusion

Conclusion

Les résultats obtenus sont bons mais le code a ses limites :

- très chronophage lorsque le nombre de données est élevé
- ne convient pas si tous les travailleurs n'ont pas étiqueté tous les items