Olivier CÔME
Assane SENE
N° étudiant Olivier CÔME: 22110708
N° étudiant Assane SENE: 21615659
olivier.come@etu.umontpellier.fr
assane.sene@etu.umontpellier.fr
Master 2
Statistique et Science des Données
Université de Montpellier

Projet Crowdsourcing

HAX916X : Ateliers-Projet
Rédigé le 3 Janvier 2023 en LaTeX

Sommaire

1 Introduction

Dans le cadre de la labélisation des images ou de la pose d'un diagnostique, plusieurs méthodes sont utilisées pour savoir le vrai label d'une image donnée ou déterminer la véritable pathologie d'un patient. Parmi eux nous pouvons citer l'algorithme EM (Expectation Maximisation). C'est un algorithme itératif, une méthode d'estimation paramétrique qui se base sur le maximum de vraisemblance.

2 Notations

On note $\forall i \in \{1, 2, \ldots, I\}$, $\forall j, l \in \{1, 2, \ldots, J\}$, $\forall k \in \{1, 2, \ldots, K\}$:

- π_{lj}^k : la probabilité que le medecin k donne la réponse j sachant que la vrai réponse est l.
- T_{ij} : une variable de réponse associé au patient i définie par $T_{iq} = 1$ si q est la vrai réponse et $T_{iq} = 0$ si $j \neq q$.
- p_j : la prévalence de la classe j ou la fréquence empirique.

3 Estimation de la vraisemblance

3.1 Cas 1 : 1 medecin et 1 patient

Soit X à valeur dans $\{1, 2, \ldots, M\}$, une variable aléatoire indiquant la maladie du patient.
Soit Y à valeur dans $\{1, 2, \ldots, J\}$, une variable aléatoire correspondant à la maladie du patient indiquée par le medecin.
A la suite de n épreuves identiques c'est-à-dire le patient (sachant qu'il est malade) voit plusieurs fois le medecin pour se faire diagnostiquer, on a :

$$Y|X = x \sim Multinomiale\left[(\pi_{xl}^Y)_l, n\right], avec \ \ l \in \{1, 2, \ldots, J\}.$$

Sa fonction de masse est donnée par :

$$\mathbb{P}\left(n_{i1}^k, \ldots, n_{iJ}^k\right) = \frac{\left[\sum_{j=1}^J n_{ij}^k\right]!}{\prod_{j=1}^J n_{ij}^k!} \prod_{j=1}^J \left(\pi_{lj}^k\right)^{n_{ij}{}^k} \propto \prod_{j=1}^J \left(\pi_{lj}^k\right)^{n_{ij}{}^k}.$$

Ainsi la vraisemblance est :

$$\propto \prod_{j=1}^J \left(\pi_{lj}^k\right)^{n_{ij}{}^k}$$

3.2 Cas 2 : K medecins et I patients

Soit $X = (X_1, \ldots, X_I)$ le vecteur aléatoire indiquant la maladie des I patients.
Soit $Y = (Y_1, \ldots, Y_K)$ le vecteur aléatoire indiquant la maladie des K patients.
On suppose J le nombre maximum de réponses possibles. Si le medecin répond une fois à la question du patient, on a :

$$Y_i^k|X_i = x_i \sim Multinomiale\left[(\pi_{x_ij}^k), 1\right], avec \ \ j \in \{1, 2, \ldots, J\} \ \ et \ \ i \in \{1, 2, \ldots, I\}.$$

Sa fonction de masse est donnée par :

$$\mathbb{P}\left(n_{i1}^k, \ldots, n_{iJ}^k\right) = \frac{\left[\sum_{j=1}^J n_{ij}^k\right]!}{\prod_{j=1}^J n_{ij}^k!} \prod_{j=1}^J \left(\pi_{x_ij}^k\right)^{n_{ij}{}^k}.$$

Etant donné que les (Y_i^k) sont indépendants $\forall \ \ k \in \{1, 2, \ldots, K\} \ \ et \ \ i \in \{1, 2, \ldots, I\}$, Donc :

$$\mathbb{P}\left(n_{i1}^k, \ldots, n_{iJ}^k\right) = \frac{\left[\sum_{j=1}^J n_{ij}^k\right]!}{\prod_{j=1}^J n_{ij}^k!} \prod_{j=1}^J \left(\pi_{x_ij}^k\right)^{n_{ij}{}^k} \propto \prod_{k=1}^K \prod_{j=1}^J \left(\pi_{x_ij}^k\right)^{n_{ij}{}^k}.$$

Par conséquent la vraisemblance est :

$$\propto \prod_{k=1}^{K}\prod_{j=1}^{J}\left(\pi_{x_i j}^{k}\right)^{{n_{ij}}^{k}}$$

Soit $p_j = \mathbb{P}(X_i = j) \ \forall \ i \in \{1, 2, \ldots, I\} \ et \ j \in \{1, 2, \ldots, J\}$.

$$\mathbb{P}(\cap Y_i^k | X_i = x_i) = \frac{\mathbb{P}\left((\cap Y_i^k) | \cap (X_i = x_i)\right)}{\mathbb{P}(X_i = x_i)}$$

$$\Longrightarrow$$

$$\mathbb{P}\left((\cap Y_i^k) | \cap (X_i = x_i)\right) = \mathbb{P}(\cap Y_i^k | X_i = x_i)\mathbb{P}(X_i = x_i)$$

Donc

$$\mathbb{P}\left((\cap Y_i^k) | \cap (X_i = x_i)\right) = p_{x_i}\prod_{k=1}^{K}\prod_{j=1}^{J}\left(\pi_{x_i j}^{k}\right)^{{n_{ij}}^{k}}$$

D'une manière générale si T_{ij} est une variable de réponse associé au patient i définie par $T_{iq} = 1$ si q est la vrai réponse et $T_{iq} = 0$ si $j \neq q$, on a :

$$\mathbb{P}\left((\cap Y_i^k) | \cap (X_i = x_i)\right) = \prod_{i=1}^{I}\left[p_{x_i}\prod_{k=1}^{K}\prod_{j=1}^{J}\left(\pi_{ij}^{k}\right)^{{n_{ij}}^{k}}\right]^{T_{ij}}$$

Si on se base sur toutes les données c'est-à-dire les réponses de tous les medecins et les questions de tous les patients, on a :

$$\mathbb{P}\left((\cap\cap Y_i^k) | \cap (X_i = x_i)\right)\mathbb{P}\left(\cap (X_i = x_i)\right) = \mathbb{P}\left((\cap\cap Y_i^k) \cap (\cap (X_i = x_i))\right).$$

Par indépendance des (Y_i^k) et (X_i),

$$\begin{aligned}
\mathbb{P}\left((\cap\cap Y_i^k) \cap (\cap (X_i = x_i))\right) &= \prod_{i=1}^{I}\mathbb{P}\left(\cap Y_i^k | \cap (X_i = x_i)\right)\mathbb{P}(X_i = x_i) \\
&= \prod_{i=1}^{I}\left(\mathbb{P}(X_i = x_i)\prod_{k=1}^{K}\mathbb{P}\left(Y_i^k | \cap (X_i = x_i)\right)\right) \\
&= \prod_{i=1}^{I}\left(p_{x_i}\prod_{k=1}^{K}\mathbb{P}\left(Y_i^k | (X_i = x_i)\right)\right)
\end{aligned}$$

D'où

$$\mathbb{P}\left((\cap\cap Y_i^k) \cap (\cap (X_i = x_i))\right) \propto \prod_{i=1}^{I}\prod_{l=1}^{J}\left[p_{x_i}\prod_{k=1}^{K}\prod_{j=1}^{J}\left(\pi_{ij}^{k}\right)^{{n_{ij}}^{k}}\right]^{T_{ij}}.$$

Par conséquent la vraisemblance de toutes les données est :

$$\propto \prod_{i=1}^{I}\prod_{l=1}^{J}\left[p_{x_i}\prod_{k=1}^{K}\prod_{j=1}^{J}\left(\pi_{ij}^{k}\right)^{{n_{ij}}^{k}}\right]^{T_{ij}}$$

4 Estimation du maximum de vraisemblance

Etant donné que la vraisemblance de toutes nos données est connue, nous pouvons trouver l'estimation du maximum de vraisemblance associé. Si on suppose que les T_{ij} sont connus et que en pratique nous avons les ${n_{ij}}^{k}$ (grâce à nos données), on a les estimateurs du maximum de vraisemblance suivants :

$$\hat{\pi}_{jl}^{k} = \frac{\sum_{i=1}^{I} T_{ij}{n_{il}}^{k}}{\sum_{l=1}^{J}\sum_{i=1}^{I} T_{ij}{n_{il}}^{k}}.$$

$$\hat{p}_j = \frac{\sum_{i=1}^{I} T_{ij}}{I}.$$

5 Applications numériques

Dans cette section, nous allons présenter les applications numériques qui ont été réalisées dans le cadre de ce projet. Cela nous permettra de voir comment les outils mathématiques théoriques, présentés dans le papier " Maximum Likelihood Estimation of Observer Error-rates using the EM Algorithm" de David et Skene, peuvent s'appliquer au problème du "crowdsourcing". Mais tout d'abord donnons le contexte de ce projet. Dans le cadre du service de micro-travail Turc mécanique d'amazon, lancé par la compagnie en 2005, un certain nombre de personnes ont annoté en échange d'une rémunération, différentes images issues d'une base de données nommée cifar10 (nous la décrirons dans le prochain paragraphe). Cependant, ces derniers ont potentiellement commis des erreurs d'étiquetage. Les vraies étiquettes (labels) des images sont ici connues mais ce n'est généralement pas le cas dans la vraie vie. Le but de ce projet est donc d'utiliser l'algorithme EM afin d'estimer les vrais labels associés à ces dernières. Comme mentionné précédemment, les vrais labels sont ici connus. Cela nous permettra donc de vérifier la justesse des étiquettes choisies par les annotateurs ainsi que la précision des prédictions faites par l'algorithme EM. Pour ce faire, nous allons générer des matrices de confusion. Ensuite, nous montrerons que l'utilisation de l'algorithme EM est pertinent dans ce cas, et nous comparerons les matrices générées à partir des prédictions de l'algorithme EM, avec celles générées à partir des labels choisis par les annotateurs. Par ailleurs, le nom Turc mécanique a été donné en mémoire du célèbre canular de l'automate joueur d'échec. Dans ce canular, l'automate soit disant doté de la capacité de jouer aux échecs était en réalité un joueur humain caché dans le mécanisme de la machine.

5.1 Le jeu de données

La base de données que nous avons utilisé "cifar10h" est une base qui contient 10 différentes classes d'images, que nous vous présentons ci dessous :

- classe 0 : airplane
- classe 1 : automobile
- classe 2 : bird
- classe 3 : cat
- classe 4 deer
- classe 5 : dog
- classe 6 : frog
- classe 7 : horse
- classe 8 : ship
- classe 9 : truck

Les données ont été stockées dans un fichier csv. Ce dernier comprend un certain nombre de colonnes décrivant les data mais nous ne nous intéresserons qu'à celles essentielles à notre étude, qui sont les suivantes :

- "annotator_id" colonne composée d'entiers allant de 0 à 2570. Ces chiffres jouant le rôle d'identifiant pour les annotateurs (il y a 2571 annotateurs).
- "true_label" cette colonne est composée d'entiers allant de 0 à 9. Ces chiffres jouent le rôle des vraies étiquettes (labels) des images.
- "chosen_label" cette colonne est composée d'entiers allant de 0 à 9. Ces chiffres correspondent aux étiquettes (labels) choisies par les annotateurs (qui ne sont pas toujours corrects).

Dans les sections suivantes, nous allons décrire les fonctions les plus importantes que nous avons implémentées pour répondre à la problématique de ce projet.

5.2 Extraction des données utiles

5.2.1 La fonction *CreateSubDf*

Cette fonction créé un nouveau dataframe à partir de celui d'origine. Ce sous dataframe contiendra les labels choisis par un annotateur spécifique (choisi en argument) ainsi que les vrais labels associés aux images.

```
def CreateSubDf(df, an_id):
    sub_df = df[(df["annotator_id"] == an_id)]
    df_labels = sub_df[['cifar10_test_test_idx', 'true_label', 'chosen_label']]
    return df_labels
```

La fonction CreateSubDf prend en argument les paramètres suivants :

— df : le dataframe des données (Il s'agit du dataframe d'origine)

— an_id : entier désignant l'identifiant de l'annotateur (an_id sera compris entre 0 et 2571 dans notre cas)

Cette fonction renvoie un dataframe composé de deux colonnes. Dans la première seront stockés les labels choisis par un annotateur spécifique (choisi en argument). Dans la colonne deuxième colonne seront stockés les vrais labels associés aux images.

5.3 Customisation et affichage des matrices de confusion

5.3.1 La fonction *custom_confusion_matrix*

Cette fonction permet d'améliorer l'esthétique de la représentation graphique de la matrice de confusion afin d'avoir une visualisation agréable et claire de cette dernière.

```
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True labels')
```

```
    plt.xlabel('Chosen labels')
    plt.tight_layout()
```

La fonction custom_confusion_matrix prend en argument les paramètres suivants :

— cm : la matrice de confusion

— classes : la liste stockant les noms de chaque classe

— normalize(Booléen) : si True la matrice sera normalisée, et si False elle ne le sera pas

— title : le titre du graphique

— cmap : la palette de couleur du graphique

5.3.2 La fonction *plot_confusion_matrix*

Cette fonction va calculer la matrice de confusion puis afficher la représentation de cette dernière.

```
def matrice_confusion(y_true, y_predict, class_names):
    conf_matrix = confusion_matrix(y_true, y_predict)
    np.set_printoptions(precision=2)
    plt.figure(figsize=(15,15))
    plot_confusion_matrix(conf_matrix, normalize=True, classes=class_names,
                          title='Matrice de confusion')
    plt.show()
```

La fonction plot_confusion_matrix prend en argument les paramètres suivants :

— y_true : un numpy array dans lequel sont stockés les vrais labels

— y_predict : un numpy array dans lequel sont stockés les labels choisis par l'annotateur

— class_names : la liste stockant les noms de chaque classe

Cette fonction retourne le graphique de la matrice de confusion ("customisé" grâce à la fonction "*custom_confusion_matr*
décrite dans la sous section précédente).
Dans la partie suivante, nous allons expliquer comment nous procédons pour utiliser le modèle de David et Skene (inventeur de l'algorithme EM) afin d'estimer les matrices de confusion.

5.4 Le modèle de David Skene

L'implémentation du modèle de David-Skene que nous avons utilisé a été réalisée par Monsieur Michael P. J. Camilleri et est intégré dans son package *isar.models*. Après de nombreuses tentatives (installation dans un environnement python 3.9, puis 3.8, puis 3.7 et enfin 3.6), nous n'avons pas réussi à installer le package en utilisant la commande "python setup.py install" dans le prompt anaconda. Nous avons donc opté pour une autre solution moins conventionnel consistant à copier le dossier ISAR-Inter_Schema_AdapteR du chercheur dans notre git. Ce dossier contient les scripts nécessaires à l'utilisation du package *isar.models* et nous l'avons copié afin de pouvoir réutiliser ses fonctions dont les plus importantes : $DawidSkeneIID$ et fit. Nous avons gardé les metadata de monsieur Michael P. J. Camilleri au début de son code python pour montrer qu'il s'agit bien de son implémentation.
Dans la sous section suivante nous allons décrire brièvement les fonctions $DawidSkeneIID$ et fit.

5.4.1 La fonction *DawidSkeneIID*

Cette fonction permet d'initialiser le modèle de David Skene.
Elle prend en paramètres de nombreux arguments mais nous décrirons uniquement ceux que nous avons utilisés. *La fonction DawidSkeneIID prend en paramètres les arguments suivants :*

— dims : un couple sous la forme (nombre de classes, nombre d'annotateurs) qui indique les dimensions du modèle

— max_iter : Le nombre d'iterations souhaité de l'algorithme EM

— predict_tol : un seuil de tolérance pour la prédiction. Si inférieur à ce seuil, la fonction retournera une valeur $np.NAN$. Le seuil par défaut est 0

5.4.2 La fonction *fit*

Cette fonction permet d'alimenter le modèle $DawidSkeneIID$.
Elle prend en paramètres de nombreux arguments mais nous décrirons uniquement ceux que nous avons utilisés.
La fonction fit prend en paramètres les arguments suivants :

— U : les données (dans notre cas le dataframe contenant uniquement les labels choisis par les annotateurs). U est de dimension (N,K) ou N est le nombre de lignes du dataframe et K le nombre d'annotateurs choisis

— priors : les probabilités du prior pour π_z et $\psi^k_{u,z} = p\big(u_k = u | Z = z\big)$ en reprenant les notations de l'article de monsieur Michael P. J. Camilleri

— starts : un tuple contenant les matrices (π et ψ) des paramètres initiaux de l'algorithme EM

5.4.3 La fonction *plot_cm_ds*

Cette fonction affiche la représentation graphique de la matrice de confusion estimée à partir des labels prédits par l'algorithme EM.

```
def plot_cm_ds(df, an_id, labels):
    sub_df = CreateSubDf(df, an_id)
    U_df = sub_df['chosen_label'].values.astype(float)
    U2D_array = np.reshape(U_df, (-1, 1))
    pi = np.random.dirichlet(np.ones(10))
    psi = np.tile(np.eye(10)[:, np.newaxis, :]*0.93, [1, 1, 1]) + 0.01
    dsmodel = DawidSkeneIID((10, 1), predict_tol=0.6)
    dsmodel.fit(U2D_array, priors=(np.ones(10), np.ones([10, 1, 10])),
    starts=[(pi, psi)])
    y_pred = dsmodel.predict(U2D_array)
    y_true = sub_df['true_label'].to_numpy()
```

La fonction plot_cm_ds prend en argument les paramètres suivants :

— df : le dataframe des données

— an_id : entier désignant l'identifiant de l'annotateur (an_id sera compris entre 0 et 2571 dans notre cas)

— labels : la liste dans laquelle sont stockés les noms des labels des images

Elle retourne le graphique de la matrice de confusion estimée.

6 Conclusion

En résumé de ce projet, nous avons pu voir que l'algorithme EM prédit les labels des images de manière assez précise. En revanche, les matrices de confusion estimées à partir de cet algorithme ne sont pas forcément meilleure que celles des annotateurs. Cependant, même si les matrices de confusion des

annotateurs ne montre pas beaucoup d'erreurs (elles sont presque diagonales), elles restent tout de même généralement très proche de celles estimées à partir de l'algorithme EM.

7 Lien git du TP

Vous pourrez accéder au code python complet (fichier intitulé $code_crowdsourcing.py$) que nous avons implémenté afin de réaliser les expériences numériques via le lien git suivant :
`https://github.com/SENEAssane/Projet_Crowdsourcing.git`