Analyse de résumés de données

Introduction

Ce projet consiste à implémenter une approche d'exploration de données et d'extraction de connaissances. Vous allez réaliser ce projet en binôme et vous disposez de trois séances de deux heures encadrées. Vous rendrez un rapport de projet ainsi que le code de votre implémentation.

Principe de l'approche

L'objectif de votre travail est de développer des méthodes d'extraction de connaissances à partir de données brutes (des objets décrits dans un fichier *csv* e.g.). La particularité de l'approche est de reposer sur une première étape de réécriture des données à l'aide d'un vocabulaire utilisateur. Ce vocabulaire, comme l'illustre la figure 1, est composé de partitions floues. Une partition floue est une discrétisation d'un domaine de définition à l'aide d'ensembles flous.

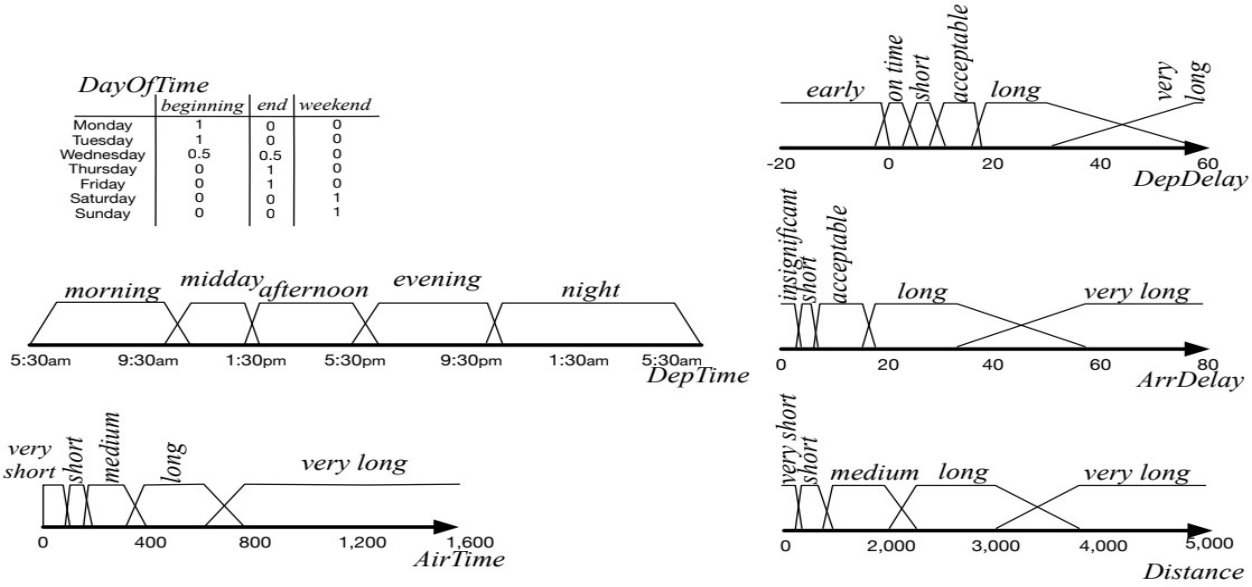

Figure 1 Extrait d'un vocabulaire flou décrivant des vols d'avions

Préparation

Récupérez l'archive *baseProject_BDA.tgz* sur l'ENT *:*

`https://perso.univ-rennes1.fr/pierre.nerzic/BDDA/Projet_2022.zip`

Cette archive contient un répertoire *Data/* dans lequel vous trouverez deux jeux de données de tests (*test.csv* et *flights2008extract.csv*). Une fois votre programme finalisé, vous pourrez le tester sur un jeu complet *flights2008.csv* compressé dans le fichier (700Mo décompressé) :

`https://perso.univ-rennes1.fr/pierre.nerzic/BDDA/flights2008.csv.7z`

Il décrit plusieurs millions de vols commerciaux aux USA en 2008. Attention au temps de calcul. Restez sur les extraits tant que ça ne fonctionne pas parfaitement. Vous trouverez également un fichier nommé *FlightsVoc.txt* contenant la spécification d'un vocabulaire flou sur les vols d'avions. Enfin, dans le répertoire Src/, vous trouverez une base de code pour effectuer la réécriture des données (vols) selon un vocabulaire. Vous pouvez tester ce code de la façon suivante :

```
$ python rewriterFromCSV.py ../Data/FlightsVoc.txt ../Data/test.csv
```

Étape 1 : Réécriture des données

En partant de *rewriterFromCSV.py*, ajoutez une méthode de réécriture des n-uplets selon le vocabulaire. L'objectif est d'aboutir à un vecteur (un dictionnaire Python) qui décrit pour chaque terme à quel degré 0..1 il couvre l'ensemble des objets. Il s'agit d'une association entre chaque terme flou et la moyenne des degrés d'appartenance de tous les n-uplets à ce terme.

Actuellement, dans ce qui vous est fourni, il y a le calcul de la réécriture de chaque n-uplet individuellement. Il vous reste à cumuler tous ces degrés, terme par terme et à en calculer les moyennes. Donc, à la fin, on n'obtient qu'un seul vecteur résumant la satisfaction moyenne des termes flous sur l'ensemble des données. Ajoutez ce qu'il faut pour enregistrer le vecteur de réécriture dans un fichier JSON.

Dans votre rapport, vous analyserez rapidement ces informations : en prenant quelques exemples, quels sont les termes qui semblent déséquilibrés, c'est à dire dont l'une des modalités regroupe une grande majorité des n-uplets au détriment des autres modalités, et quels sont les termes qui semblent équilibrés. Il se peut que ces équilibres ou non montrent une mauvaise définition du vocabulaire, et il se peut aussi que les données soient réellement déséquilibrées.

Étape 2 : Exploration de données

Définissez une nouvelle méthode de réécriture des données permettant de ne résumer que les données satisfaisant un ensemble de termes avec un degré de satisfaction strictement supérieur à un seuil de satisfaction fixé, 0 par défaut. La méthode que vous devez écrire prend donc en paramètre une liste de termes flous à considérer de manière conjonctive, suivie d'un seuil de satisfaction optionnel.

Cette méthode vous permettra par exemple de résumer uniquement les vols partant le soir « *DepTime.evening* » et couvrant une longue distance « *Distance.long* ». Mettez plusieurs exemples dans votre rapport, mais attention au temps de calcul sur les données complètes.

Techniquement, pour extraire des paramètres optionnels, voici ce qu'on peut programmer en Python :

```
            terms = sys.argv[3:]
            try:
                alpha = float(terms[-1])
                terms = terms[:-1]
            except ValueError:
                alpha = 0.0
            Rterms = rw.readAndRewrite(terms, alpha)
```

Ajoutez ce qu'il faut pour enregistrer ce vecteur de réécriture dans un second fichier JSON, qui sera à distinguer du précédent en construisant un nom approprié, basé sur les termes choisis.

Étape 3 : Extraction de connaissances

À partir du vecteur de réécriture du jeu de données complet, désigné par *R*, et d'un vecteur de réécriture d'un sous-ensemble d'objets, noté *Rv*, satisfaisant une certaine condition *v*, il est possible

d'extraire des connaissances permettant de mieux comprendre les propriétés des n-uplets qui satisfont *v*.

Termes corrélés

Vous allez tout d'abord identifier les termes corrélés aux termes *v*, c'est-à-dire les propriétés *v'* impliquées par *v*. La corrélation entre *v* utilisé pour sélectionner des données et obtenir *Rv*, et un autre terme *v'* du vocabulaire est quantifié par la formule suivante :

$$assoc(v, v') = \begin{cases} 0 & \text{if } dep(v, v') \leq 1 \\ 1 - \frac{1}{dep(v,v')} & \text{otherwise} \end{cases}$$

où

$$dep(v, v') = \frac{cover(v', R_v)}{cover(v', R)}.$$

cover(v', Rv) est appelé « couverture du terme *v'* dans le vecteur de réécriture *Rv* ». C'est la somme des degrés de satisfaction des termes de *v'* divisée par le nombre de termes résumés dans *Rv* (la taille du vocabulaire). Évidemment, *cover(v', R)* doit être strictement positif. Généralement, c'est une valeur assez petite, mais le principe, c'est de la comparer à une autre.

Écrire un programme qui relit les deux vecteurs de réécriture, *R* et *Rv* et qui calcule *assoc(v,v')* pour chaque terme *v'* du vocabulaire. On part donc de la réécriture *Rv* d'une conjonction *v* et de *R* et on obtiendra une énumération de termes *v'* et de degrés de corrélation, *assoc(v, v')* compris entre 0 et 1. Commentez les deux ou trois résultats les plus marquants : que peut-on en tirer comme information ?

Termes atypiques

Vous allez désormais identifier les termes atypiques dans *Rv*, c'est-à-dire les termes surprenants, c'est à dire qu'ils apparaissent dans la réécriture mais pas majoritairement, et ils sont éloignés des termes majoritaires. Un terme *v''* est considéré comme surprenant si :

- Il couvre une minorité des données. C'est exprimé par *1-cover(v'', Rv)*,
- Il est éloigné des autres termes *v'* de la même partition qui couvrent de manière majoritaire l'ensemble des données concerné. C'est exprimé par *cover(v', Rv)*.

La distance entre deux termes *v''* et *v'* d'une même partition est représentée par *d(v'',v')*. Si l'attribut concerné par *v''* et *v'* est numérique alors *d(v'',v')* est égal au nombre d'éléments de partition qui les séparent divisé par le nombre d'éléments de partition moins un. Toutefois, il y a un piège pour calculer le nombre d'éléments d'écart, quand la partition se referme sur elle-même. Par exemple, l'attribut *DepTime* varie dans [0, 24] et il est représenté par 5 modalités telles que la dernière *night* correspond à la première *morning*. Donc il faut trouver le plus court « chemin » entre les modalités pour déterminer la distance qui les sépare.

Si l'attribut est catégoriel, la distance est 1 dès que les termes sont différents, 0 sinon.

Pour ce calcul, il faut ajouter des méthodes aux classes qui gèrent le vocabulaire. La classe *Partition* représente la liste des modalités d'un terme flou. Les modalités sont triées dans l'ordre de la partition. Il suffit donc d'obtenir les indices de *v''* et *v'*, et tester si la dernière modalité correspond à la première, pour en déduire la distance qui les sépare.

Le traitement général consiste :

1. dans un premier temps, à définir une condition *v* et à calculer *R* et *Rv*, c'est à dire les réécritures des données sans et avec la condition *v*.

2. Ensuite, comme précédemment, on recherche les corrélations, c'est à dire les *v'* qui sont davantage présents dans *Rv* que dans *R* : *assoc(v, v')>0*.

3. Puis, pour chacun de ces *v'*, on parcourt tous les *v''* qui sont dans la même partition que *v'*. Notez que *v'* en fait partie, mais ce n'est pas grave.

4. Sur chacun de ces *v'* et *v''*, on calcule le degré suivant :

   $atyp(v', v'') = min(d(v', v''), cover(v', Rv), 1-cover(v'', Rv))$.

5. Enfin, il faut afficher le *v''* qui possède le plus grand degré $atyp(v', v'')$ strictement positif.

Pour programmer tous ces traitements, il faut chercher les méthodes appropriées dans les classes qui représentent le vocabulaire : *voc.getPartition(attrname)*, *partition.getLabels()*, *partition.getAttName()*, etc. Ces méthodes permettent de parcourir les listes de partitions et les listes de modalités de chaque partition.