TP : Fouille de données de type graphe et applications chémoinformatiques

M2BI – Chloé-Agathe Azencott

6 janvier 2016

1 Similarités entre représentations catégoriques

On désire mesurer la similarité entre des objets représentés par les variables catégoriques décrites sur la Figure 1. Ces variables peuvent par exemple correspondre à des sous-graphes de graphes moléculaires.

FIGURE 1 – Variables catégoriques.

On se donne les trois objets représentés en Figure 2 par l'ensemble de leur variables catégoriques.

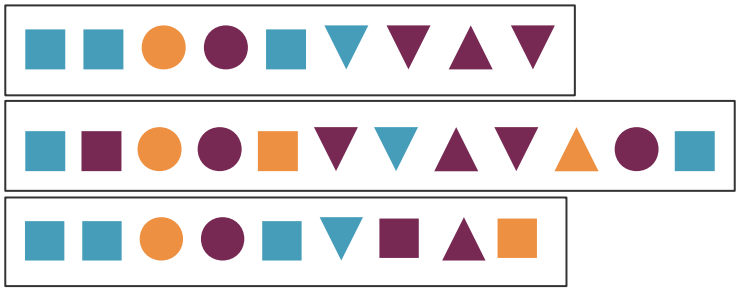


FIGURE 2 – Objets représentés par les variables catégoriques qui les composent. L'ordre n'importe pas, il s'agit d'ensembles.

La distance de Hamming entre deux objets x et z est donnée par le nombre de bits différents entre leurs représentations binaires :

$$d(x, z) = \sum_{j=1}^{p} (x_j \text{ XOR } z_j)$$

1. Calculer la distance de Hamming entre ces objets. Lesquels sont les plus proches ? Les plus distants ?

Solution: Représentations binaires :
A = 100011010110
B = 111011011110
C = 111011010100
D'où les distances de Hamming suivantes :
d(A, B) = 3
d(A, C) = 3
d(B, C) = 2.
B et C sont les objets les plus proches. B et C sont également loin de A.

La similarité de Tanimoto entre deux objets x et z est donnée par le nombre de bits partagés entre les deux objets, normalisé par le nombre de bits à 1 dans l'un ou l'autre de ces objets :

$$s(x, z) = \frac{\sum_{j=1}^{p}(x_j \text{ AND } z_j)}{\sum_{j=1}^{p}(x_j \text{ OR } z_j)}$$

2. Calculer la similarité de Tanimoto entre ces objets. Lesquels sont les plus semblables ? Les moins semblables ?

Solution: En utilisant les représentations binaires ci-dessus :
s(A, B) = 6/9 = 0.67
s(A, C) = 5/8 = 0.63
s(B, C) = 7/9 = 0.78.
B et C sont encore les objets les plus proches, mais A est plus loin de C que de B.

La similarité de Minmax entre deux objets x et z est la version "de compte" de la similarité de Tanimoto :

$$s(x, z) = \frac{\sum_{j=1}^{p} \min(x_j, z_j)}{\sum_{j=1}^{p} \max(x_j, z_j)}$$

3. Calculer la similarité de Minmax entre ces objets. Lesquels sont les plus semblables ? Les moins semblables ?

Solution: Représentations "comptes" :
A = 300011010120
B = 211021011120
C = 311011010100.
D'où les similarités de Minmax suivantes :
s(A, B) = 8/13 = 0.62
s(A, C) = 7/11 = 0.64
s(B, C) = 8/13 = 0.62.
Maintenant, ce sont A et C qui sont les plus proches. Ces deux objets sont également dissimilaires à B.

4. Prenons des fingerprints binaires $\vec{A}$ et $\vec{B}$. Soit a le nombre de bits à 1 dans $\vec{A}$, b le nombre de bits à 1 dans $\vec{B}$, et c le nombre de bits à 1 dans $\vec{A}$ **et** $\vec{B}$.

   Exprimer les quantités suivantes en termes de a, b et c seulement :

   (a) Similarité cosinus :

$$s(\vec{A}, \vec{B}) = \frac{\vec{A}.\vec{B}}{||\vec{A}||.||\vec{B}||}$$

   (b) Distance de Hamming / Manhattan :

$$d(\vec{A}, \vec{B}) = \sum_{j=1}^{p} |A_j - B_j|$$

   (c) Similarité de Tanimoto :

$$s(\vec{A}, \vec{B}) = \frac{\sum_{j=1}^{p}(A_j \text{ AND } B_j)}{\sum_{j=1}^{p}(A_j \text{ OR } B_j)} = \frac{\vec{A}.\vec{B}}{||\vec{A}||^2 + ||\vec{B}||^2 - \vec{A}.\vec{B}}$$

Solution:

— Similarité cosinus : $s(\vec{A}, \vec{B}) = \frac{\vec{A}.\vec{B}}{||\vec{A}||.||\vec{B}||} = \frac{c}{\sqrt{ab}}$

— Distance de Hamming : $d(\vec{A}, \vec{B}) = \sum_{i=1}^{p} |A_i - B_i| = a + b - 2c$

— Similarité de Tanimoto : $s(\vec{A}, \vec{B}) = \frac{\vec{A}.\vec{B}}{||\vec{A}||^2+||\vec{B}||^2-\vec{A}.\vec{B}} = \frac{c}{a+b-c}$

2 Chaînes SMILES

Les règles d'écriture de la représentation SMILES d'une molécule sont les suivantes :

— Les atomes sont représentés par leur élément atomique.

— Les atomes dans des cycles aromatiques sont représentés en lettres minuscules.

— Les atomes H et charges partielles sont liés à un atome par des crochets.
Ex. `[Fe+2]` = `[Fe++]`

— On peut négliger de noter les H : l'absence de crochets indique une valence normale.
Ex. `[CH4]` = `C`

— Liaisons atomiques : simples (`-`), doubles (`=`), triples (`#`), aromatiques (`:`)
Ex. `C=O`, `C#N`, `C-C` = `CC`

— Les branchements sont indiqués par des parenthèses.
Ex. `CCN(CC)CC`

— On "casse" les cycles et on utilise des numéros pour indiquer où.
Ex. `c1ccccc1`

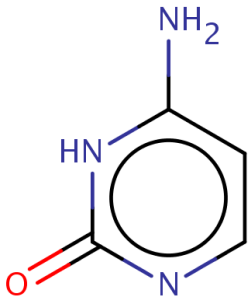
1. Dessiner les molécules représentées par les SMILES suivantes :

(a) Cytosine : `c1cnc(=O)[nH]c1N`

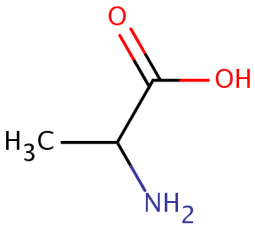
(b) Alanine : `O=C(O)C(N)C`

(c) Acetate : `CC([O-])=O`

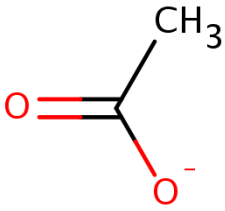
Solution:



`c1cnc(=O)[nH]c1N`
cytosine

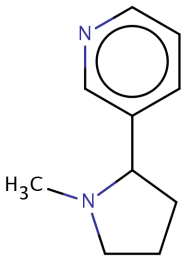


`O=C(O)C(N)C`
alanine

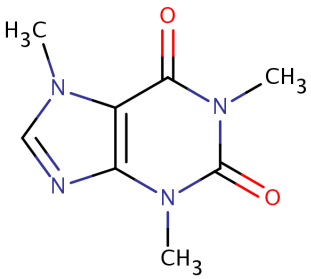


`CC([O-])=O`
acetate

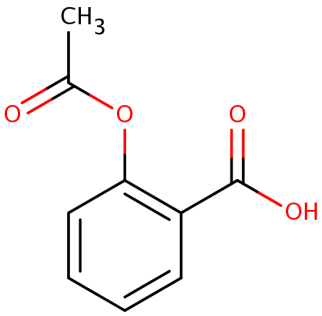
2. Écrire une représentation SMILE pour les molécules suivantes :



nicotine

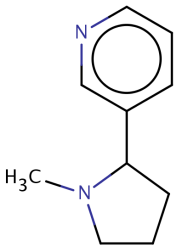


caffeine

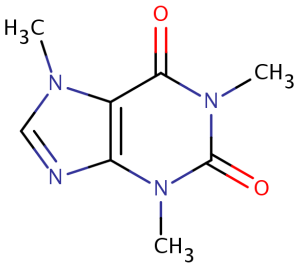


aspirin

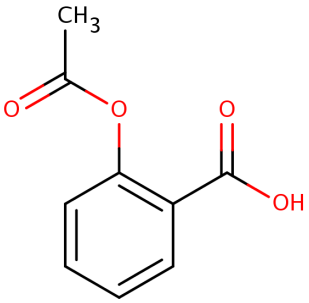
Solution:



`c1ncccc1C1CCCN1C`
nicotine



`CN1C=NC2=C1C(=O)N(C(=O)N2C)C`
caffeine



`CC(=O)OC1=C(C=CC=C1)C(O)=O`
aspirin

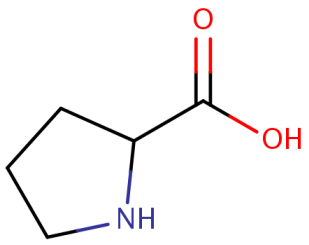
Vous pouvez vous aider de `http://cdb.ics.uci.edu/cgibin/Smi2DepictWeb.py` pour vérifier la validité de vos solutions.

3 SMILES canoniques : Étiquetage de Morgan

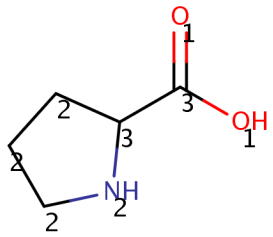
L'algorithme de Morgan permettant l'étiquetage de graphes moléculaires est le suivant :

1. Donner à chaque atome non-H sa valence (nombre d'atomes autres que H auquel il est lié).
2. Ré-étiqueter chaque atome par la somme des étiquettes de ses voisins.
3. Répéter jusqu'à obtenir des étiquettes aussi uniques que possible : s'arrêter quand une nouvelle itération n'augmente pas le nombre d'étiquettes différentes.
4. Donner l'étiquette 1 à l'atome avec le plus grand numéro. Puis 2 au voisin de 1 avec le plus grand numéro, 3 au voisin suivant, et ainsi de suite.
   - Pour départager les valeurs égales, donner l'étiquette la plus petite à l'atome dont la liaison à l'atome "initial" est de cardinalité la plus élevée. Par exemple, si l'atome étiquetté 1 est lié à deux atomes de même étiquette, l'un par une liaison simple et l'autre par une liaison double, celui lié par double liaison aura le numéro 2 et celui lié par liaison simple aura le numéro 3.
   - Les atomes invariants par symmétrie reçoivent la même étiquette. Par ailleurs, il existe aussi des cas dans lesquels l'algorithme de Morgan n'est pas capable de différencier des atomes qui ne sont *pas* invariants par symmétrie.

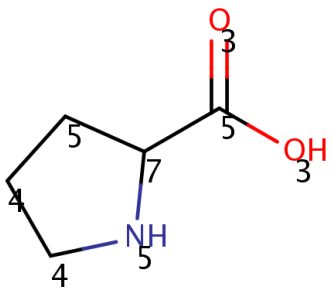
1. Calculer l'étiquetage de Morgan de la proline :



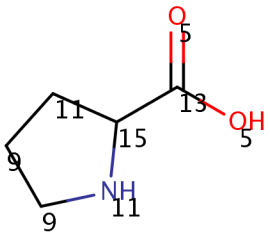
Solution: First, we label each heavy atom by its number of heavy atom neighbors. There are 3 different labels.



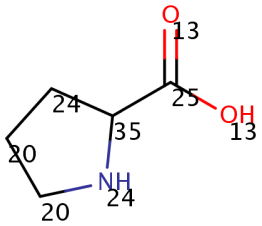
We then replace each atom's label by the sum of its neighbors' labels. There are 4 different labels.



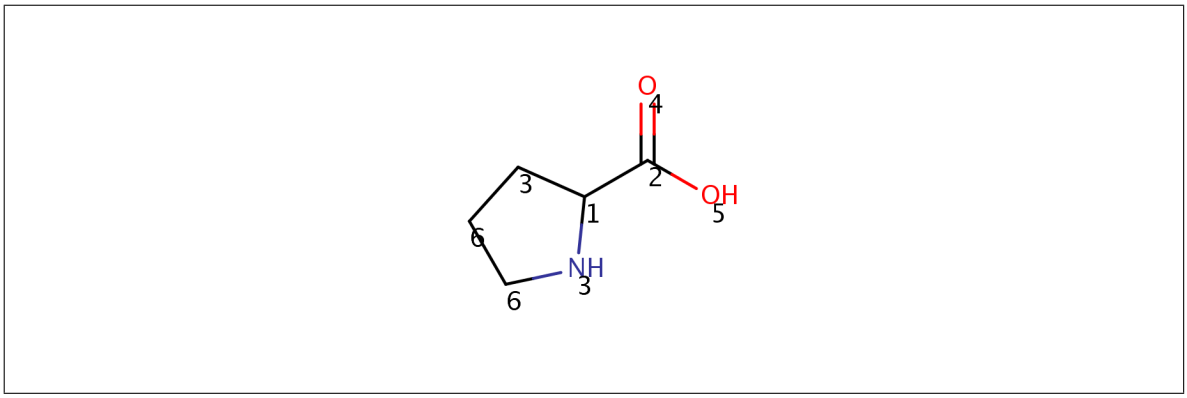
And again. There are 5 different labels.



If we repeat the process again, we still do not get more than 5 different labels. We stop iterating here.



(e) Finally, we relabel as 1 the atom with highest label (35). Then we assign 2 to its neighbor with highest label (25) and 3 to both other neighbors, which are undistinguishable (symmetry equivalent). We then move to atom 2 and label its neighbors. The double-bonded O, having the highest bond order from the issuing atom, gets label 4, and the other O gets label 5. Finally the neighbors of the nodes labeled 3 get label 6.



4 Analyse du jeu de données Mutag

Le but de ce TP est d'analyser le jeu de données de mutagénicité Mutag. Ce jeu de données contient 188 molécules étiquettées 1 ou -1 selon qu'elles sont mutagènes ou non. Ce TP est conçu en Python.

Données Téléchargez le jeu de données Mutag depuis `http://tinyurl.com/mutag2015`.
Le fichier `mutag_188_data.smi` contient les 188 molécules au format SMILES.
Le fichier `mutag_188_target.txt` contient leurs étiquettes, dans le même ordre.

Fingerprints Les fichiers `mutag_188_*.fingerprints` contiennent des représentations des molécules du jeu de données Mutag précalculées. Le format est le suivant :

— Chaque ligne commençant par `#` représente une sous-structure

— Chaque ligne ne commençant pas par `#` correspond à une molécule, représentée par sa chaîne de caractères SMILES, suivie de son nom, suivie de sa fingerprint dans un format type dictionnaire (`<indice>:<valeur>` pour tous les indices pour lesquels la valeur n'est pas 0.)

En ce qui concerne les noms des fichiers, `PATH_d<depth>` correspond à l'extraction de tous les sous-chemins de taille allant jusqu'à `<depth>` et `CIRC_d<depth>` correspond à l'extraction de tous les sous-arbres de profondeur allant jusqu'à `<depth>`.

1. Quelle est la longueur des fingerprints pour chacun des fichiers ?

Solution:

```
grep "#" mutag_188_PATH_d8.fingerprints | tail -1
# 2308 C:C:C:C:C=C=C~C~N
grep "#" mutag_188_PATH_d4.fingerprints | tail -1
# 364 C=C=C~C~N
grep "#" mutag_188_CIRC_d2.fingerprints | tail -1
# 352 {C|={C|=C|=C|~C}|~{C|=C|~C|~N}}
grep "#" mutag_188_CIRC_d3.fingerprints | tail -1
```

```
# 1149 {C|={C|={C|:C|:C|=C}|={C|=C|~C}|~{C|=C|~C|~N}}|~{C|={C|=C
   |~C}|~{C|=C|~C}|~{N|=O|~C|~O}}}
```

2. Créer une fonction Python `read_data` qui lise un fichier de fingerprints et un fichier d'étiquettes (aux formats ci-dessus) et crée une liste `samples` de dictionnaires (pour les fingerprints) et une liste `labels` d'étiquettes.

```
def read_data(in_fingerprint_f, in_labels_f):
  ...
  return samples, labels
```

3. **Calcul de similarités (Tanimoto)**
   (a) Créer une fonction Python `tanimoto` qui calcule le coefficient de Tanimoto entre deux dictionnaires.

$$\text{Tanimoto}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

```
def tanimoto(dictA, dictB):
  ...
  return tan
```

   (b) Vérifier que la similarité de Tanimoto est toujours comprise entre 0 et 1.
   (c) Vérifier que la similarité de Tanimoto entre les deux premières fingerprints de `mutag_188_CIRC_d2.fingerprints` est 0.535714 :

```
samples, labels = read_data("mutag_188_CIRC_d2.fingerprints",
                            "mutag_188_target.txt")
print tanimoto(samples[0], samples[1])
```

4. **Calcul de similarités (MinMax)**
   (a) Créer une fonction Python `tanimoto` qui calcule le coefficient minmax entre deux dictionnaires.

$$\text{Minmax}(A, B) = \frac{\sum_{i=1}^{N} \min(A_i, B_i)}{\sum_{i=1}^{N} \max(A_i, B_i)}$$

```
def minmax(dictA, dictB):
  ...
  return mma
```

   (b) Vérifier que la similarité Minmax est toujours comprise entre 0 et 1.
   (c) Vérifier que la similarité Minmax entre les deux premières fingerprints de `mutag_188_CIRC_d2.fingerprints` est de 0.4754098 :

```
samples, labels = read_data("mutag_188_CIRC_d2.fingerprints",
                            "mutag_188_target.txt")
print minmax(samples[0], samples[1])
```

5. **Algorithme du plus proche voisin**
   (a) Créer une fonction Python `nearest_neighbor` qui calcule, étant donnés un dictionnaire `dictA`, une liste `samples` de dictionnaires et une similarité (`'tanimoto'` ou `'minmax'`), l'indice du plus proche voisin de `dictA` dans `samples`.

```
def nearest_neighbor(dictA, samples, similarity):
    ...
    return similarities.index(max(similarities))
```

   (b) Vérifier que le plus proche voisin de la première fingerprint dans `mutag_188_CIRC_d2.fingerprints`, parmi toutes les autres fingerprints du même fichier, est celle qui a pour index 32 (selon Tanimoto) et celle qui a pour index 9 (selon Minmax) :

```
samples, labels = read_data("mutag_188_CIRC_d2.fingerprints",
                            "mutag_188_target.txt")
print nearest_neighbor(samples[0], samples[1:], 'tanimoto')
print nearest_neighbor(samples[0], samples[1:], 'minmax')
```

6. **Validation croisée** Créer une fonction Python qui coupe le jeu de données en k jeux d'entraînement et de validation selon le principe de la validation croisée, prédit la classe des molécules du jeu de validation conformément à celle du plus proche voisin (dans le jeu d'entraînement), et retourne la liste des prédictions (correctement ordonnée).

```
predictions = cross_validate_nearest_neighbor(samples, labels, num_folds)
```

7. **Évaluation des performances** Pour chacun des fichiers de fingerprints, évaluer la performance de l'algorithme du plus proche voisin en termes de : accuracy, précision, rappel.

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

$$\text{precision} = \frac{\text{number of true positives}}{\text{number of predicted positives}}$$

$$\text{recall} = \frac{\text{number of true positives}}{\text{number of positives}}$$

8. **Algorithme des k plus proches voisins** Remplacer l'algorithme du plus proche voisin par celui des k plus proches voisins. Faire varier k.

5 Informations complémentaires

scikit-learn La bibliothèque `scikit-learn` [1] fournit un grand nombre d'outils permettant d'effectuer les tâches que nous venons d'implémenter. De nombreux algorithmes de classification, régression et clustering y sont implémentés, ainsi que des méthodes facilitant la validation croisée, l'évaluation des performances, la normalisation des données, la sélection de variables...

scikit-learn permet par exemple aussi de tracer des courbes ROC (taux de vrais positifs, ou sensibilité, en fonction du taux de faux positifs, ou 1-sensitivité, pour tous les seuils possibles) et de calculer l'aire sous cette courbe.

scikit-learn repose sur `NumPy` (Numerical Python), une extension de Python destinée à la manipulation de matrices et tableaux multidimensionels, et `SciPy` (Scientific Python), une collection de bibliothèques de calcul scientifique, ainsi que sur `matplotlib`, une bibliothèque de visualisation de données sous forme graphiques.

OpenBabel Open Babel [2] est une toolbox libre conçue pour manipuler des données chimiques. OpenBabel est capable de calculer un certain nombre de représentations "fingerprints" pré-définies.

Le paquet **Pybel** [3] est un wrapper Python pour OpenBabel :

```
import pybel
```

Pour lire le fichier de molécules et créer une liste d'objets de la classe `pybel.Molecule` :

```
chemicals = list(pybel.readfile("smi", "mutag_188_data.can"))
```

Nous pouvons ensuite extraire une représentation vectorielle ("fingerprint") pour chacune des molécules. Par défaut, Pybel propose d'indexer les fragments linéaires (ou sous-chemin) d'une longueur variant de 1 à 7 atomes. Pour plus d'information, voir : `http://openbabel.org/docs/2.3.1/Fingerprints/fingerprints.html?highlight=fp2`.

```
fingerprints = [pybel.Molecule.calcfp(chem, "FP2") for chem in chemicals]
```

`fingerprints` est une liste d'objets de type `pybel.Fingerprint`. Pour accéder aux indices des bits à 1 dans ces fingerprints, on peut utiliser l'attribut `bits`. Par exemple :

```
fingerprints[0].bits
```

Pour calculer d'autres types de fingerprints, il faut utiliser d'autres logiciels, tels que ChemCPP [4], ou les implémenter soi-même. Noel O'Boyle propose un exemple d'implémentation de fingerprints circulaires (ECFP) ici :
`http://baoilleach.blogspot.fr/2008/02/calculate-circular-fingerprints-with.html`.

1. `http://scikit-learn.org/`
2. `http://openbabel.org`
3. `http://openbabel.org/docs/2.3.1/UseTheLibrary/Python_Pybel.html`
4. `http://chemcpp.sourceforge.net/html/index.html`