# TD 6 Fouille de données : Moteurs de Recommandation

M2 Informatique, Université Paris Diderot

---

**Installation de sklearn: pip install -U scikit-learn**

---

On utilise le dataset Jester 2. Je vous ai préparé une version partiellement nettoyée: jester.tar.xz, à extraire avec la commande: tar -xf jester.tar.xz

- jester_jokes.txt pour les blagues elle-même (les caractères saut de lignes sont remplacés par des tabulations)
- jester_ratings.csv pour les ratings: n° utilisateur, n° de blague, rating.

**Exercice 1: Basé sur le contenu -- Similarité à d'autres contenus**

Dans cette première partie on ignore complètement les ratings. On veut juste, pour une blague donnée, trouver les blagues les plus similaires.
Utilisez `TFidfVectorizer` de sklearn (cf exo 1.3 du TD précédent) pour calculer les TF-IDF de chaque blague. On pourra utiliser `open('filename', 'r').readlines()` pour lire le fichier de blagues et le convertir en listes de blagues (chaque blague = 1 chaîne de caractères).
Ensuite, essayez d'utiliser `cosine similarity` pour mesurer la distance entre blagues.
Finalement, écrivez du code pour, étant donnée la matrice des TF-IDF, et l'index d'une blague, renvoyer les 5 blagues qui lui ressemblent le plus.

Essayez avec la blague #0. Vous devriez notamment trouver que la blague qui lui ressemble le plus est la blague #86. Pour rapidement extraire et visualiser une blague donnée, vous pouvez par exemple faire (dans la console interactive python):

```
>>> data=open('jester_jokes.txt','r').readlines()
>>> data[0]
>>> data[86]
```

**Rendu:** dans un fichier td6.py, écrivez une fonction

```
def most_similar(tfidf, item_index, k):
  """Returns indices of k most similar items to item #item_index (excl. itself)

  Args:
    tfidf: a sparse matrix as returned by TfidfVectorizer.fit_transform(..)
    item_index: an integer, the index of the item (0-based). Eg. a row index.
    k: an integer, the number of similar item indices to return.

  Returns:
    A list of integers representing the items (eg. rows) most similar to item
    #item_index, excluding itself. Use the cosine similarity.
```

```
  """
```

**Exercice 2 : Basé sur le contenu et sur les ratings existants de l'utilisateur.**

On va maintenant prendre en compte les *ratings* existants d'un utilisateur pour pouvoir recommander un contenu parmi ceux qu'il n'a pas encore vu.

Il faut tout d'abord être capable d'**ingérer les données**. Écrivez la fonction suivante:

```
def read_ratings(filename, num_jokes):
  """Parses a file in the same format as jester_ratings.csv.

  Args:
    filename: a string: the name of the file to parse.
    num_jokes: the global number of jokes (some may never appear in the file,
               Which is why this argument is necessary).

  Returns:
    A list of dictionaries: the list of ratings for each user. The ratings for a
    user are a dictionary {joke_id: rating}, where joke_id is an integer in
    0..num_jokes-1 and rating is a float in [-10,10].
    When a joke is not rated by a user, it should not be in the user's dictionary.
  """
```

Aide: si vous ne connaissez pas encore collections.defaultdict, n'hesitez pas a y jeter un oeil, ca peut vous être utile (on peut très bien faire cet exercice sans! Mais cette structure de données est généralement utile en python). De même, la fonction get(key, default) d'un dictionnaire peut être utile.

**Test:** Si vous exécutez les instructions suivantes, vous devriez voir les sorties correspondantes:

```
>>> import td6
>>> r = td6.read_ratings('/tmp/jester_ratings.csv', 150)
>>> r[0]
{3: 0.219, 5: -9.281, 6: -9.281, 11: -6.781,  ……
>>> sum([r[0][x] for x in r[0]])
129.871
```

Ensuite, on va coder le moteur de recommandation à proprement parler. L'idée qu'on va utiliser ici est de calculer les similarités des blagues entre elles (avec la similarité cosinus), et de les utiliser comme poids pour le calcul d'une moyenne pondérée:

$$rating(b) = \frac{\sum_{i\ pair\ |\ b_i \text{ a un rating}} rating(b_i) * similarity(b, b_i)}{\sum_{i\ pair\ |\ b_i \text{ a un rating}} similarity(b, b_i)}$$

On calculera cette formule pour tous les blagues “impaires” (d’index impair), et on proposera donc les K meilleures.
L’idée est donc de se baser sur des ratings connus (les blagues paires) pour classer des blagues inconnues (les blagues impaires), en se basant uniquement sur une analyse de similarité fondée sur le contenu.

**Indice**: on peut calculer d’un seul coup une matrice de similarité à partir d’une matrice TF-IDF, par exemple, avec:
`from sklearn.metrics.pairwise import cosine_similarity`
`cosine_similarity(tfidf)`

Écrivez la fonction suivante:

```
def content_recommend(similarity_matrix, user_ratings, k):
  """Recommends k best jokes for a given user.

  This recommendation takes as input the ratings of a single user, but only
  takes into account the ratings of even-numbered jokes, while it only recommends
  Odd-numbered jokes.

  Args:
    similarity_matrix: A similarity matrix of size NxN.
    user_ratings: a dictionary {joke id: rating} containing the known joke
                  ratings of a given user.
    k: an integer, the number of odd-indexed jokes to recommend.

  Returns:
    A list of odd joke indices recommended for this user, based on the joke
    similarities and using only the user's ratings of even-indexed jokes.
  """
```

**Tip: “even” = pair, “odd” = impair**

**Test:** Essayez-là!
Vous pouvez par exemple récupérer une liste `user_ratings` (pour l’utilisateur #0) avec r[0] dans le code déjà mentionné ci-dessus, et calculer `similarity_matrix` avec la commande donnée en indice.
Il est encore plus intéressant de l’essayer sur vous-même: ratez vous-même quelques blagues “paires” (en indexant à 0, i.e. la toute première blague du fichier à l’index 0), et voyez quelles blagues votre fonction vous recommande!

**Exercice 3: Collaborative filtering**

Vous vous en doutiez, on va maintenant utiliser la puissance du collaborative filtering.

L'idée est, étant donné un nouvel utilisateur (i.e. potentiellement absent du dataset!), des ratings partiel des blagues de notre dataset par cet utilisateur, et la donnée de jester_rating.csv, de pouvoir donner une recommandation à ce nouvel utilisateur (lui proposer des blagues qu'il n'a pas encore vues).

On peut reprendre la formule vue en cours, à savoir estimer le rating d'une blague (pour notres nouvel utilisateur) en faisant la moyenne pondérée des ratings de cette blague par les autres utilisateurs, où les poids de la pondération correspondent au coefficient de corrélation avec chaque autre utilisateur.
On propose donc simplement les K meilleures blagues selon cette formule.

Indices:

- utilisez numpy.corrcoeff.
- Vous devrez traiter les `None` correctement. Cf cours : les enlever, ou les remplacer par une moyenne?

Écrivez la fonction suivante:

```
def collaborative_recommend(ratings, user_ratings, k):
  """Recommends the k best unknown jokes based on the ratings of some know jokes.

  Args:
    ratings: The output of read_ratings(), see above.
    user_ratings: A dictionary of joke:rating pairs, where joke is a 0-based
                  integer and rating a float.
    k: an integer, the number of jokes to recommend.

  Returns:
    A list of joke indices recommended for this user, of size k, which will not
    contain any jokes that were already known by this user (eg. those whose index
    is in the user_ratings dictionary).
  """
```

**Test:** De même, essayer votre fonction à la main. Vous pouvez utiliser un utilisateur existant, ou, mieux, essayer sur vous-même!