Fouille de données – TD 5: TF-IDF, Clustering

Notions abordées: Statistiques 101, TF-IDF, Clustering

L'objectif de ce TD est de **manipuler des jeux de données** et appliquer de l'**extraction de features** et des débuts d'**analyse statistique**.
À partir d'un jeu de données que vous allez transformer, vous allez pouvoir tester différents modèles et comprendre comment choisir le meilleur.
Votre alliée dans cette affaire est la page d'aide de l'API sklearn. Toutes les fonctions utilisées ici y sont expliquées. Vérifiez au préalable quelle version de sklearn est installée sur votre ordinateur :

```
import sklearn
print(sklearn.__version__)
```

Installation des paquets requis:

`[sudo] pip3 install numpy scipy scikit-learn tensorflow`
Si vous travaillez sur les ordinateurs de l'université, tous devrait être pré-installé. Sinon, vous devrez **au préalable** vous créer un environnement local pour installer des paquets Python sans être administrateur. Suivez ces instructions pour le créer.

Si ca ne marche pas:

- **Linux**: essayez ça:
  ```
  sudo apt-get install build-essential python3-dev python3-setuptools\
     python3-numpy python3-scipy python3-pip libatlas-dev libatlas3gf-base
  sudo pip3 install scikit-learn
  ```
- **Mac**: ces choses peuvent être utile :
  - `brew link --overwrite python3 # Si soucis de conflits de versions de python3`
  - `Le flag --ignore-installed: pip install scipy` **`--ignore-installed scipy`**
- **Autres plateformes**, ou si ca ne marche pas: essayez d'abord virtualenv (cf stackoverflow), puis demandez de l'aide si toujours pas

Récupération et transformation des données

Pour ce TD nous utiliserons SMSSpamCollection.

Exercice 1: Échauffement/Révisions : Transformation des données

1. read_dataset

   Écrivez une fonction `read_dataset(filename)` qui prend en argument le chemin vers le fichier et retourne une liste de paires type/texte. Le type est un entier `0` pour les hams (non-spams) et `1` pour les spams. On pourra utiliser open().

```
def read_dataset(filename):
  """ Reads the file at the given path that should contain one
   type and one text separated by a tab on each line, and returns
   pairs of type/text.

   Args:
      filename: a file path.
   Returns:
      a list of (type, text) tuples. Each type is either 0 or 1.
  """
```

2. spam_count

   Écrivez une fonction `spam_count(pairs)` qui prend en argument la liste renvoyée par la fonction précédente et retourne le nombre de spams dans le jeu de données.

```
def spam_count(pairs):
  """ Returns the number of spams from a list of (type, text) tuples.

   Args:
      pairs: a list of (type, text) tuples.
   Returns:
      an integer representing the number of spams.
  """
```

   Vous pourrez vérifier les 2 premières fonctions en commande interactive:

```
$> python3
>>> import td5
>>> td5.spam_count(td5.read_dataset('SMSSpamCollection'))
```

   La valeur renvoyée devrait être 379.

3. Préparation/Documentation (rien à écrire)

   Importez la classe `sklearn.feature_extraction.text.TfidfVectorizer`. Celle-ci vous permet d'appliquer l'algorithme TF-IDF vu en cours. Le constructeur accepte un certain nombre de paramètres optionnels que vous pouvez donner pour adapter

l'algorithme à vos besoins, et retourne un objet capable d'appliquer cet algorithme. Vous trouverez ici la documentation de cette classe. Observez tous les arguments et choisissez des valeurs pertinentes. Il n'y a pas une seule bonne réponse, mais il faut être capable de justifier vos choix.

4. transform_text

Écrivez une fonction `transform_text(pairs)` qui retourne une matrice `X` (les textes au format TF-IDF) et un vecteur `y` (0 si le message est un ham, 1 s'il s'agit d'un spam).

```
from sklearn.feature_extraction.text import TfidfVectorizer

def transform_text(pairs):
  """ Transforms the pair data into a matrix X containing TF-IDF values
   for the messages and a vector y containing 0s and 1s (for hams and
   spams respectively).
   Row i in X corresponds to the i-th element of y.

   Args:
      pairs: a list of (type, message) tuples.
   Returns:
      A pair (X, Y) with:
      X: a sparse TF-IDF matrix where each row represents a message and
         each column represents a word.
      Y: a vector whose i-th element is 0 if the i-th message is a ham,
         else 1.
  """
```

Inspectez la valeur de retour. Vous pourrez utiliser le module `scipy`. Pour itérer efficacement sur une "sparse matrix", on peut appeler la fonction `nonzero()` sur la matrice, en passant par les très pythoniques `zip` et `*`:

```
# Exemple: X est une sparse matrix.
for x,y in zip(*X.nonzero()):
  print("[%d][%d] = %s" % (x, y, X[x, y]))
```

Exercice 2: Apprentissage non supervisé (clustering)

1. Un simple KMeans

Trouvez dans la bibliothèque `sklearn` la fonction qui vous sera utile pour appliquer un algorithme du KMeans. Lancez sur l'ensemble `X` de l'exercice précédent un KMeans avec K = 10 (on va donc former des clusters de SMS "similaires"). Vous verrez un exemple de script sur la page d'aide de l'algorithme. Inspirez-vous en !

Écrivez ensuite une fonction qui fait tout cela pour un **K** quelconque, et qui renvoie les **P** mots les plus caractéristiques (au sens de TF-IDF) dans chaque cluster:

```
def kmeans_and_most_common_words(pairs, K, P):
  """Applies TF-IDF and then the K-Means algorithm with the given K on the
   pair data. Then returns the most common P words in each cluster.

   Args:
      pairs: a list of (type, message) tuples.
   Returns:
      A list of K lists, each containing P strings: the "most
      characteristic" words of each cluster. By "most characteristic, we
      mean the highest TF-IDF score, averaged over the entire cluster.
  """
```

Test: a faire vous-même: lancez votre fonction sur les paires données par read_dataset() sur SMSSpamCollection, avec K=10 et P=5 par exemple, et regarder les résultats. Vous semblent-ils logiques?

2. Choisir le meilleur "K"

Cette fois, vous ne connaissez pas la valeur de K. Comment trouver la meilleure valeur de K ? On va utiliser la fonction `sklearn.metrics.silhouette_score`.
Attention, vous devez être capable d'expliquer ce que représente cette valeur.

Implémentez cette solution:

```
def best_k(pairs):
  """Applies TF-IDF to the given pairs, then returns the K that seems best
   for K-means, meaning the one that yields the highest silhouette score.

   Args:
      pairs: a list of (type, message) tuples.
   Returns:
      A pair (best_k, silouhette_score): the best possible K, and the
      silhouette score for that value.
  """
```

3. **Comparer deux algorithmes de clustering**

   Vous n'avez presque rien à faire pour lancer cette fois un algorithme de classification hiérarchique sur ces données. L'algorithme s'appelle `AgglomerativeClustering` dans la bibliothèque `sklearn`.

4. **Moyenne des clusters**

   Une fois que vous avez obtenu des clusters, il s'agit de calculer la "moyenne" de chaque clusters, pour pouvoir ensuite mesurer la similarité d'un nouveau SMS aux différents clusters obtenus.

5. **Classification**

   Écrivez la fonction suivante:

```
def classify_batch(train_pairs, test):
  """Classification algorithm using clustering, and nearest (euclidian)
   distance: use some clustering algorithm from the previous questions,
   and decide on a way to use them to classify all the messages in "test".

   Args:
      pairs: see previous questions
      test: a list of messages to classify, e.g. ['Hello world',
            'Did you receive my last call ?', …]

   Returns:
      A list of the (estimated) types of the messages in "test": this list
      Will contain as many elements as "test", each element will be 0 or 1.
  """
```

Exercice 3: Test statistiques [A faire seulement si vous aimez les stats]

En reprenant les valeurs TF-IDF faites dans l'exo 1, vous allez maintenant classer les mots par ordre de "pouvoir discriminatif" grâce aux tests statistiques. Commençons par une fonction qui, pour chaque mot, nous renvoie la p-value liée au test: "Le mot apparaît plus souvent dans les spams que dans les hams". On va utiliser pour cela la fonction du module `stats` de `scipy` nommée `ttest_ind` dont le but est de comparer les moyennes de deux échantillons.
(si vous avez sauté le dernier paragraphe "Inspectez …" de la question précédente, vous allez galérer: regardez-le)

```
import scipy

def test_word_means(X, Y, word_index):
  """ Performs a two-means t-test on the tf-idf values of a given word
   represented by its index in the matrix X. The test checks whether
   the word is over-represented (in terms of their TF-IDF value) in
   spammy messages and returns its p-value. The smaller the p-value, the
   more over-represented the word is within spams compared to hams.

  Args:
      X: the TF-IDF matrix where each line represents a document and each
         column represents a word, typically obtained by running
         transform_text().
      Y: a binary vector where the i-th value indicates whether the i-th
         document is a spam, typically obtained by running
         transform_text().
      word_index: an int representing a column number in X.
  Returns:
      A double that corresponds to the p-value of the test (the
      probability that the word is NOT over-represented in the spams).
  """
```

Appliquez maintenant cette fonction à tous les mots et repérez les N = 20 mots les plus représentés dans les spams. A partir de quelle valeur de N cette liste n'est-elle plus statistiquement significative?