Fouille de données – TP 2

M2 Informatique, Université Paris Diderot
Anne-Claire Haury
2017/2018

L'objectif de ce second TP est de vous essayer au clustering et à la classification supervisée. 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__)
```

Si vous travaillez sur les ordinateurs de l'université, vous devez au préalable vous créer un environnement local pour installer des paquets Python sans être administrateur. Suivez ces instructions pour le créer et y installer les paquets de base.

Récupération et transformation des données

Récupération des données

Pour ce TP nous utiliserons un jeu de données composé de messages SMS que vous pouvez télécharger sur le site du cours.
Il s'agit de construire un anti-spam à partir de ces données.
Chaque ligne du fichier représente un message et son label (spam/ham). Nous utiliserons ces données pour entraîner un algorithme de prédiction qui fournira pour un message une réponse entre `0` (ham) et `1` (spam).
Au préalable, nous allons utiliser ce jeu de données de manière non supervisée (sans regarder les labels) et lui appliquer deux algorithmes de clustering.

Exercice 1 - Transformation des données

1. 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.

```
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. Écrivez une fonction `spams_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 spams_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.
    """
```

3. 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. 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:
        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.
    """
```

5. 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.

```
from scipy.stats import ttest_ind

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 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?