TD : K-NN appliqué à la classification de textes

1 Intro

Il s'agit d'implémenter en python un classifieur de type K-NN, appliqué à la classification de textes.
Vous rendrez un seul programme, qui réalise l'apprentissage du vecteur de traits et la classification de nouveaux documents.
Comme d'habitude vos programmes comportent une commande d'aide (--help ou –h), et sont **commentés.**

2 Les données

Collection « reuters21578 » : collection historique de documents en **anglais**, associés à 0, 1 ou n classes (« topics »)
On fournit un sous-ensemble de ce corpus :

- **train = medium.train.examples** = 2000 documents, associés à 1 classe parmi 91 possibles
- **test =** reuters.modapte.test.sgm = 200 documents, associés à 1 classe

2.1 Représentation vectorielle d'un document :

Les fichiers **reuters.train.examples** et **reuters.test.examples** contiennent les couples classe gold + représentation vectorielle du document, pour l'ensemble d'apprentissage et l'ensemble de test. On n'utilise pas ici d'ensemble de développement.

L'espace des traits est constitué des formes fléchies apparaissant dans les documents d'**apprentissage**. Plus précisément, on représente un document par un vecteur avec :

- une dimension par forme fléchie apparaissant au moins 3 fois dans les documents des exemples d'apprentissage, et apparaissant dans moins de 60% des documents
- avec comme valeur, le nombre d'occurrences de la forme fléchie, divisée par le nb total d'occurrences du doc[1]

3 Implémentation

Le but est d'implémenter un programme qui

- lit un fichier d'exemples au format .examples, un fichier de test au format .examples
- applique un classifieur K-NN sur les exemples du fichier de test
- calcule et affiche la précision obtenue, en pourcentage
-

3.1 Etude du canevas fourni

Etudiez le canevas fourni : en particulier l'aide en ligne (-h), les classes Examples, Ovector et KNN.

[1] Plusieurs améliorations IMPORTANTES DEVRAIENT être apportées ici : utiliser les lemmes au lieu des formes fléchies, filtrer les mots outils, utiliser un score TF.IDF …

Toutes les lectures d'options et le main sont déjà implémentés, ainsi que la lecture des exemples.

Il contient des classes pour représenter un vecteur, un exemple (i.e. un vecteur plus la classe associé à l'objet que ce vecteur représente), et un classifieur K-NN.

Ces vecteurs peuvent naïvement être implémentés comme des dictionnaires avec comme clé une forme fléchie (cf. on a une dimension par forme fléchie).

La classe Ovector utilise de manière simpliste un dictionnaire clé=une dimension (ici une forme fléchie), valeur =valeur du trait, avec la convention que **les traits à valeur nulle ne sont pas stockés**.
Cette représentation pose des problèmes d'efficacité (une implémentation plus sérieuse utiliserait un package mathématique, avec gestion de vecteurs creux : voir numpy « sparse matrices »).

Une optimisation possible dans ce cadre est de calculer la distance euclidienne de la manière suivante :

$$dist(\vec{a},\vec{b}) = \sqrt{\sum (a_i - b_i)^2} = \sqrt{\sum a_i^2 + \sum b_i^2 - 2\sum a_i b_i}$$

3.2 Pseudo-code

Ecrivez le pseudo-code de la phase de prédiction pour un exemple donné (voir canevas méthode « classify »)

Ecrivez le pseudo-code pour l'application du classifieur aux exemples de tests, et le calcul de la précision

3.3 Implémentation effective

Il vous reste à remplir les méthodes :
Ovector.distance_to_vector, KNN.classify et KNN.evaluate_on_test_set

En plus : faire un script pour le réglage des hyper paramètres K et d'un booléen (weight_neighbors) utilisant la pondération des voisins (par l'inverse de la distance) lors du vote pour la meilleure classe étant donnés les K plus proches voisins.

En plus : implémenter une validation croisée

NB : durant la mise au point de votre programme, utilisez les petits fichiers small.train.examples et small.test.examples pour ne pas perdre de temps au debug