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 **nomenminuscule_knn.py** . Il comportera une commande d'aide (--help ou –h), et sera **commenté.**

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, avec 1 seule classe par document :

train = medium.train.examples = 2000 docs classés (parmi 91 classes possibles)
test = medium.test.examples = 200 documents classés

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.

Espace vectoriel : L'espace des traits est constitué des formes fléchies apparaissant dans les documents d'**apprentissage**[1], avec D le nombre de formes fléchies distinctes du vocabulaire ainsi défini.

Pour un document donné d, son vecteur représentatif a D composantes
- chaque composante correspond à une forme fléchie du vocabulaire,
- la valeur de la composante pour la forme fff vaut le nb d'occ de fff dans d, divisée par le nb total d'occ dans d

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

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

3.1 Astuce de calcul de la distance euclidienne

Le K-NN repose sur le calcul de la distance euclidienne avec tous les exemples d'apprentissage. Cela peut être très long si l'ens. d'apprentissage est grand. Une optimisation possible est d'utiliser :

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

Dans le TD (voir infra), on va utiliser pour représenter les vecteurs des dictionnaires où les traits à valeur nulle ne sont pas stockés. Etant donné ce type de représentation, et les représentations vectorielles des documents, **expliquez pourquoi ce mode de calcul sera plus rapide**.

En outre, dans le cadre du K-NN, **ce mode de calcul permet de pré-calculer certaines parties, lesquelles**?

3.2 Etude du canevas fourni

Etudiez le canevas, en particulier l'aide en ligne (-h), les classes Example et KNN.
Toutes les lectures d'options et le main sont déjà implémentés, ainsi que la lecture des exemples.

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

Le membre v de la classe Example 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 optimisé, voir TD ultérieur avec **numpy**).

[1] Plus précisément, on a dans l'espace vectoriel une composante par forme fléchie apparaissant au moins 3 fois dans les documents des exemples d'apprentissage, et apparaissant dans moins de 60% des documents. Plusieurs améliorations IMPORTANTES DEVRAIENT être apportées ici : utiliser les lemmes au lieu des formes fléchies, utiliser un **score TF.IDF** au lieu des nbs d'occurrences, ce qui améliore nettement les résultats.

3.3 Pseudo-code

Ecrivez le pseudo-code de la phase de prédiction pour un exemple donné (voir canevas méthode « classify »), en supposant que vous avez déjà stocké la norme de chaque vecteur au carré (membre norm_square de example).
En cas d'égalité de classe, vous rendrez la première classe par ordre alphabétique.
Ecrivez le pseudo-code pour l'application du classifieur aux exemples de tests, et le calcul de la précision.

3.4 Implémentation effective

Voir les parties "TODO" du canevas:

- Distance euclidienne : commencez par le calcul des normes au carré des vecteurs d'apprentissage et de test. **Où est-il le plus judicieux de faire ce calcul? Implémentez-le.**
- Méthode de prédiction (classify) pour un document donné : rendre la liste des classes [c1, c2, … cK] correspondant à la prédiction pour k=1, k=2 … k=K
- Evaluation sur un ensemble de test : idem, rendre une liste de précisions (accuracies) pour k=1, k=2 …. k=K

3.5 Résultats attendus

En utilisant le corpus medium (train / test), voici les performances attendues pour quelques valeurs de K:

```
ACCURACY FOR K =  1 = 61.50 (123 / 200) (weight = False dist_or_cos = dist)
ACCURACY FOR K =  2 = 60.00 (120 / 200) (weight = False dist_or_cos = dist)
ACCURACY FOR K =  3 = 61.50 (123 / 200) (weight = False dist_or_cos = dist)
ACCURACY FOR K =  4 = 59.50 (119 / 200) (weight = False dist_or_cos = dist)
ACCURACY FOR K =  5 = 60.00 (120 / 200) (weight = False dist_or_cos = dist)
```

3.6 Pour aller plus loin: hyperparamètres

Implémentez

- option : utilisation de la **similarité cosinus** versus la distance euclidienne
- option : pondération ou pas des voisins:
    - par l'inverse de la distance si distance euclidienne
    - ou par le cosinus sinon

Il s'agit ensuite d'implémenter le lancement des 4 combinaisons, avec un nb K grand (cf. toutes les valeurs inférieures de K seront aussi testées) et d'étudier les résultats pour chercher la meilleure combinaison d'hyperparamètres.