Lab session : K-NN for document classification

1 Intro

The objective is to build a document classifier that uses the K-NN algorithm. Deliver a single python file via moodle, named your-last-name-in-lower-case.py
YOU ARE KINDLY REQUESTED TO COMMENT YOUR CODE ...

2 Data

The « reuters21578 » collection is a famous set of documents in English, associated to 0, 1 or n classes (« topics »). We provide a sub-set of this corpus, with a single class per document (single label multiclass classification):

train = medium.train.examples = 2000 documents (among 91 potential classes)
dev = medium.dev.examples = 200 documents
(we won't use here any dev set, which is wrong)

2.1 Vector representation of a document

Files **reuters.train.examples** et **reuters.dev.examples** contain pairs (gold class + vector representing the document), for the training and dev sets.
Vector space : each position in the vector space corresponds to an inflected form belonging to the vocabulary seen in the **training set**[1]. Let D be the size of this vocab.

For a given document d, its vector representation has size D

- the component at position i, corresponding to word form f_i, is the number of times f_i appears in d, divided by the total number of occurrences in d

3 Implementation

The target program will

- read a train and dev files in .examples format
- apply a K-NN classifier on dev examples using the train examples
- compute and print the resulting accuracy (percentage of dev examples that are correctly classified by our K-NN)

[1] More precisely, we have one component per inflected form appearing at least 3 times in the documents of the training set, and appearing in less than 60% of the documents. Several improvements could be made here: using lemmas instead of inflected forms, using a TF.IDF score instead of the number of occurrences.

NB : when developing your program, use the small.train.examples and small.dev.examples files to avoid wasting time on debugging

3.1 Distance calculation trick

K-NN relies on calculating the Euclidean distance with all the training examples. This can be very long if the training set is large. A possible optimization is to use:

$$dist(a,b)=\sqrt{\sum_{i=1}^{D}(a_i - b_i)^2}$$

$$=\sqrt{\sum_{i=1}^{D}a_i^2+\sum_{i=1}^{D}b_i^2-2\sum_{i=1}^{D}a_i\,b_i} = \sqrt{\|a\|^2+\|b\|^2-2a\cdot b}$$

In this first version, we will use dictionaries to represent the vectors, where the null-valued features are not stored. **Given this type of implementation, and considering the fact that document vectors are sparse, explain why this trick can be faster**.

Also, in the K-NN framework, this mode of computation allows some parts to be pre-computed, which ones?

3.2 Program to fill

Study the provided python program, in particular the online help (-h), the Example and KNN classes. The main of the program is provided, as well as a method for loading .examples files.

In this version of the program, we will use dictionaries to represent vectors: in such dicts, keys are the inflected forms. **Absent keys correspond to null values**. There is no actual ordering of the components of the vectors.

This representation has efficiency problems, we will use a matrix implementation in the next session.

3.3 Pseudo-code

Write down the pseudo-code for the prediction phase for a given example (you will then implement it in the KNN.classify method). Youi will suppose that square norms of each vector in the examples have already been computed (member norm_square in Ovector class).
In case of equality of number of neighbors for several classes, you will return the first class in alphabetic order.

Write down the pseudo-code for running the classifier on the test examples, and computing the accuracy.

3.4 Code

See the TODO parts in the program to fill.

- Euclidian distance: start by computing the square norms for the vector of all train examples and all test examples (to do once and for all). **Make sure you position this computation at the most appropriate step.**
- KNN.classify method, for a given document:
  - NB : whatever the value of K, the prediction for an example requires to compute the distances between this example and all the training examples, which can be VERY long
  - So to test several values of K, it is better to do this calculation only once
  - => the method classify(x, K) not only returns the predicted class using the K nearest neighbors, but returns the list of classes [c1, c2, ... cK] predicted using respectively k=1, k=2 ... k=K neighbors
- Evaluation on a test set: in the same vein, will return a list of accuracies for k=1, k=2 .... k=K

3.5 Expected results

When using the medium corpus (train / dev), here is the expected accuracy for a few k values:

```
ACCURACY FOR k =  1 = 61.50 (123 / 200)
ACCURACY FOR k =  2 = 60.00 (120 / 200)
ACCURACY FOR k =  3 = 61.50 (123 / 200)
ACCURACY FOR k =  4 = 59.50 (119 / 200)
ACCURACY FOR k =  5 = 60.00 (120 / 200)
```

3.6 To go further: hyperparameters

Implement various options:

- option cos_or_dist: use cosine similarity versus Euclidean distance
- option weight: with or without any weighting of the neighbors:
  - by the inverse of the distance if Euclidean distance
  - or by the cosine otherwise

Implement the launching of the 4 combinations, with a large nb K
(cf. all lower values of K will also be tested) and study the results to identify the best combination of hyperparameters (k, cos_or_dist, weight).