Lab session : K-NN for document classification

1 Intro

The objective is to build a document classifier that uses the K-NN algorithm.
You will turn in a single python file via moodle, named YOUR-LAST-NAME-IN-UPPER-CASE_whatever.py
Please use *meaningful variable names* + comments so that your code reads easily.

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, in which we made sure to have a single gold class for each document (to perform monolabel multiclass classification):

More precisely:

- **train = medium.train.examples** = 2000 documents with their gold class
- **dev = medium.dev.examples** = 200 documents with gold class
    - which you will use to test and evaluate the algorithm

- **Document representation**: Documents are represented using BOW vectors, in which the numbers of occurrences word forms have been divided by the total number of tokens in the document[1].
- **Format**: But NB, the provided *.examples files only contain the components with non-null values, identified using the word form itself instead of an integer position in the vector space.
- **Vocabulary**: the total vocabulary is the union of all features (words) in all **training** documents.
- **"Unknown words"**: when applying the K-NN to an input document, its features (words) that are not in the training documents will just be ignored

3 Pseudo-code

The target program will have the following methods/steps:

[1] More precisely, we have one component per inflected form appearing at least 3 times in 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.

1. `read_examples` (already provided) reads a .examples file and stores the examples into a single instance of the `Examples` class.
2. **Indices for words**:
   - build indices for the word vocabulary (as union of vocab of all training documents)
   - assign an integer identifier to each word of the vocabulary
3. **Turn training set and test set into matrices**
   - o build_matrix X_train (resp. X_dev) ndarrays, in which rows are the BOW vectors representing training (resp. dev) documents
   - o build Y_train and Y_dev as lists of the gold classes of training / dev examples
4. **K-NN classification**:
   - o use matrix operations to compute the **cosines** of each row in X_dev with each row in X_train (cf. LAB1)
     - ▪ shape will be (nb_test, nb_train)
   - o predict a class for each of the **dev** examples, using the **train** examples and K-NN algorithm, using cosine to identify the neighbors.
5. **Evaluation**: Compute and print the resulting **accuracy** (percentage of dev examples for which the predicted class is the correct one)
   - o The optimal value of K is unknown, all you can do is test some k values and choose the best one (cf. methodology in machine learning : this is "tuning the hyperparameters")

4 Program to fill in

Study the provided python program, run it with -h option to see the online help (obtained via the argparse module). The main of the program is provided, as well as a read_examples method for loading .examples files (step 1 above).

TODO: study how read_examples work, and the Examples class

4.1 Indices for words

To map the words to ids and vice-versa we can use:
- a **dict** `w2i` : from the word string to its id
- a simple **list** `i2w` : at rank k in the list, we store the word having id=k

This can also be done for the vocabulary of class names (dict c2i and list i2c).

4.2 Turn training set and test set into matrices

Represent the T=2000 training examples [vector, gold_class] using two structures:
- **X_train**: a T x V matrix (ndarray), whose i-th row is the vector for the i-th example

- **Y_train**: a vector of size T, for the gold classes of the T examples (actually it can be a plain python list for now).

Similarly, represent the test data as **X_test** and **Y_test.**

➔ fill in the `build_matrices` method, which takes as input:
  - an `Examples` instance
  - the w2i correspondance (or a more sophisticated user-defined instance for indices)

  and outputs the corresponding X ndarray and Y list or ndarray

Tips : start by building an X matrix with desired shape, filled with zeros, and while looping over the examples, you can assign the non-null cells

5 Efficiency

Write down the pseudo-code for step number 4 above (prediction phase), supposing the previous steps are already done, with K=3.

If needed, improve your algorithm:

- don't recompute the training norms for all test example
- *identify which steps are shared for all possible K values as opposed to steps that depend on K.*
  - you could make your K-NN work for all values from k=1 to k=K, sharing steps not depending on k, and output a list of K accuracies, for k=1, k=2 .... k=K

6 Expected results

When using the **small corpus** (train / dev), here is the expected cos_matrix and the accuracies for a few k values:

```
Reading train examples...
READ 5 EXAMPLES, vect length 67
Reading test examples...
READ 3 EXAMPLES, vect length 67
Evaluating on test...
Matrix of cosine similarities (rows = test, columns = train):
[[0.1864542  0.30658331 0.25518441 0.05080005 0.01133659]
 [0.         0.009787   0.01176674 0.70624201 0.66805963]
 [0.18749253 0.08808303 0.24710158 0.22135944 0.        ]]

ACCURACY FOR K =  1 = 100.00 (3 / 3) (use_weight = False)
ACCURACY FOR K =  2 = 66.67 (2 / 3) (use_weight = False)
ACCURACY FOR K =  3 = 66.67 (2 / 3) (use_weight = False)
```

On medium, here are the expected results for the first k values:
```
ACCURACY FOR K =  1 = 78.50 (157 / 200) (use_weight = False)
ACCURACY FOR K =  2 = 76.00 (152 / 200) (use_weight = False)
ACCURACY FOR K =  3 = 77.50 (155 / 200) (use_weight = False)
ACCURACY FOR K =  4 = 81.00 (162 / 200) (use_weight = False)
ACCURACY FOR K =  5 = 79.50 (159 / 200) (use_weight = False)
```

NB: It is ok even if you have not exactly the same results, but if accuracy is close to these values.

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

NB: In case of equality of number of neighbors for several classes, you will return the first class in alphabetic order