Obligatory exercise 2b (INF4820, Fall 2013)

This is the second and final part of the second obligatory exercise in INF4820. You can obtain up to 10 points for 2b, and you need a minimum of 12 points for 2a + 2b in total. It's a good idea to read through the entire problem set before you start coding (3 pages). Send an email to `inf4820-help@ifi.uio.no` to if you have any questions. Answers must be submitted via Devilry by the end of the day (23:59) on Thursday, October 17th.

Summary of goals for this exercise:

- Add more functionality for the vector space model from the previous assignment.
- Implement a Rocchio classifier. This means we need to define functions for
  - reading in information about predefined classes (the training data);
  - computing centroid representations for the classes; and
  - classifying unknown words as a function of centroid distance.
- **NB**: The programming we need to do for this assignment naturally extends on what we did for the previous one. To ensure that everyone is starting out from a working and sufficiently efficient implementation, we provide a solution (source code) that you're free to use as you want.
- The first half of the problem set comprises practical coding exercise, while the second half contains some theoretical questions.

Files you'll need

We will be re-using the data sets from exercise 2a; 'brown20000.txt' and 'words.txt'. The former contains the first 20.000 sentences of the Brown corpus, comprising close to half a million words. The latter contains the list of words we'll be modeling. In addition, you'll need the file 'classes.txt', containing lists of predefined classes and their members. Copy the file to your home directory:

```
cp ~erikve/inf4820/classes.txt ~/
```

An implementation of the vector space functionality from exercise 2a can be copied from the same directory:

```
cp ~erikve/inf4820/vs.lsp ~/
```

Reading in the corpus data

The provided source file 'vs.lsp' includes an implementation of the function '`read-corpus-to-vs`' taking two arguments; a corpus file and a file specifying the words to model. The function returns a vector space model, defined in terms of the Lisp structure '`vs`'. Feel free to modify it or use it as is (of course, you're also free to extend/write your own code for this). Make sure to always use *compiled* (rather than just interpreted) code when dealing with large data sets like we'll do here, as this makes the code run faster.

The following call will create a length-normalized vector space model for the words in the file 'words.txt':

```
CL-USER(35): (setf space
               (length-normalize-vs
                 (read-corpus-to-vs "brown20000.txt" "words.txt")))

#S(VS :STRING-MAP (:FEATURE ...
                   :WORD ... )
      :ID-MAP (:FEATURE ...
               :WORD ...)
      :MATRIX ...
      :SIMILARITY-FN ...
      :CLASSES NIL)
```

As you'll see, the structure definition of '`vs`' in 'vs.lsp' has been extended with a few more slots to accommodate the extensions to the vector space model that we will be implementing here. Please take some time to familiarize yourself with the code in the provided source file, and try to convince yourself that it does what it should.

1 Implementing a Rocchio classifier

(a) In this exercise we'll implement a *Rocchio classifier*. The first thing we need to do is read in information about which words are associated with which classes. Have a look at the file 'classes.txt'. This file contains lists specifying the class membership of the different words in our model. The first element in each list specifies the class name (given as a Lisp keyword, e.g. '`:foodstuff`'. The second element is a list specifying the words associated with the given class, e.g. '`(potato food bread fish ...)`'. Some of the words are *unclassified*, however, and these are listed with the dummy class '`:unknown`'. The unknown words are the words we want to classify. The other words define our *training data*. *Note*: the words found in the file 'classes.txt' are the same as those in the file 'words.txt'. This means that all the relevant feature vectors, both for the training items and the test items, are already available in our vector space model.

Write a function '`read-classes`' that reads the lists from the file 'classes.txt' and stores the information about class-membership in the slot '`classes`' in our '`vs`' structure. Exactly how to store and organize that information is up to you. (But you'll want to make it easy to retrieve information about the members of each class, and perhaps also make it possible to add more information about classes later, such as the corresponding centroid representation. You probably also want to take care to convert all the words to lowercase strings.) Remember that the function '`read`' is very handy for reading s-exps (like lists).

(b) The Rocchio classifier represents classes by their *centroids* $\vec{\mu}$. For a given class c_i, the centroid vector $\vec{\mu}_i$ is simply the average of the vectors of the class members,

$$\vec{\mu}_i = \frac{1}{|c_i|} \sum_{\vec{x}_j \in c_i} \vec{x}_j$$

where $|c_i|$ denotes the cardinality of the class (i.e., the number of class members as observed in the training data). The class centroids are often not normalized for length, but in order to avoid bias effects for classes with different sizes (classes with many members will typically have less sparse centroids and a larger norm), we will here define our centroids to have unit length. By this we mean that their Euclidean length should be one; $\|\vec{\mu}_i\| = 1$. Luckily we implemented functionality for normalizing vectors in the exercise 2a.

Write a function '`compute-class-centroids`', expecting only a '`vs`' structure as its argument. The function should compute the length normalized centroids for each class. Store the centroids somewhere within the vector space structure (for example adding it to the already existing '`classes`' slot).

(c) We now have all the pieces we need in order to write a Rocchio classifier and classify the unlabeled words (i.e., the words that were listed after '`:unknown`' in the file 'classes.txt'). Recall that a Rocchio classifier assigns each word to the class which has the nearest centroid. Write a function '`rocchio-classify`' that for each unclassified word in our model assigns it to its nearest class.

There are many similarity functions that could potentially be used when performing the classification, i.e., for measuring how close a given feature vector is to a given centroid. We'll here continue to use the dot-product, however, just as in the previous exercise. Recall also that the particular similarity function assumed in the space is stored in the slot named '`similarity-fn`'. As an example, the output of '`rocchio-classify`' could look something like this:

```
CL-USER(73): (read-classes space "classes.txt")
NIL

CL-USER(74): (compute-class-centroids space)
NIL

CL-USER(75): (rocchio-classify space)
(("fruit" :FOODSTUFF 0.36678165) ("california" :PERSON_NAME 0.29967937)
 ("peter" :PERSON_NAME 0.3088614) ("egypt" :PLACE_NAME 0.30741462)
 ("department" :INSTITUTION 0.54971284) ("hiroshima" :PLACE_NAME 0.23045596)
 ("robert" :PERSON_NAME 0.59566414) ("butter" :FOODSTUFF 0.384543)
 ("pepper" :FOODSTUFF 0.36385757) ("asia" :PLACE_NAME 0.3986899)
 ("roosevelt" :TITLE 0.2597395) ("moscow" :PLACE_NAME 0.48412856)
 ("senator" :TITLE 0.3563019) ("university" :INSTITUTION 0.5805098)
 ("sheriff" :TITLE 0.22804888))
```

Among other things, this would mean that we found the centroid of the class '`:place_name`' to be the one closest to the feature vector representing '`egypt`', and that the dot-product of these two vectors is approximately 0.31. (Your results might differ.)

(d) So far we haven't said anything about the particular *data type* used for implementing the *centroid vectors*. We have silently assumed that the data type is the same as what we've used for the feature vectors of individual words. In a few sentences, discuss whether or not you believe this is a wise choice.

2 Classification theory

(a) Give an outline of the main differences between Rocchio classification and kNN classification. Limit your discussion to no more than half a page. (Some keywords to get you started: decision boundaries, complexity of training vs testing, class representation, etc.)

(b) In just a couple of sentences, compare and contrast Rocchio classification and k-means clustering. What sets these algorithms apart, and in what ways can they be considered similar? (We're interested in rather general and basic traits here.)

(c) One missing piece in the implementation above is *evaluation*. In a few sentences, outline what would need to be done in order to evaluate our classifier here. Include some comments on the different strategies we could use for computing our evaluation scores.