Algorithms for AI and NLP (Fall 2014), Problem Set (2b)

Goals

- Implement a Rocchio classifier. This means we need to define functions for
  - computing centroid representations for the classes;
  - classifying unknown words as a function of centroid distance; and
  - evaluation of the classifier on the provided test set.
- Classify IMDB movie reviews into two classes: positive and negative.
- **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) for Problem Set (2a) that you are 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.

Background

This is the second and final part of the second obligatory exercise in INF4820. You can obtain up to 10 points for Problem Set (2b), and you need a minimum of 12 points from (2a) plus (2b) in total. It is a good idea to read through the entire problem set before you start coding (three pages). Send an email to `inf4820-help@ifi.uio.no` to if you have any questions. Your code and answers must be submitted via Devilry by the end of the day (23:59) on Wednesday, October 15.

Files you'll need

This time we will use a new corpus. It is a corpus of sentiment analyzed movie reviews from IMDB comments[1].

To obtain the data files for this problem set, please obtain updates from the SVN repository for the course, e.g. (assuming you opted for the recommended directory location previously):

```
cd ~/inf4820
svn update
```

After the update, there will be a new sub-directory called '`2b/`', containing the files '`sentiwordnet.txt`' and '`readerdata2b.lsp`' and the directories '`test`' and '`train`'.

The '`sentiwordnet.txt`' file contains a copy of a resource called SentiWordNet.[2] The directories '`test`' and '`train`' contain documents created from user comments for movies on the IMDB website classified in two categories "**pos**" (positive) "**neg**" (negative)

Sentiment Analysis

Sentiment analysis (also known as opinion mining) refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. Generally speaking, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall

[1] `http://imdb.com`
[2] `http://sentiwordnet.isti.cnr.it/`

contextual polarity of a document. The attitude may be his or her judgment or evaluation (see appraisal theory), affective state (that is to say, the emotional state of the author when writing), or the intended emotional communication (that is to say, the emotional effect the author wishes to have on the reader). [3] In out experiments we will experiment with a binary sentiment analysis task. The task consists in classifying a movie review in one of two categories:

- **pos** (positive) - The review is expresses a generally favorable option about a movie. Example: "*kolya is one of the richest films i've seen in some time.*"
- **neg** (negative) - The review is expresses a generally negative option about a movie. Example: "*this film is extraordinarily horrendous and i'm not going to waste any more words on it .*"

Your task will be to build a simple classifier using the Rocchio algorithm that assign a category to a review.

SentiWordNet To help you with the task we provide a list of emotionally charged words from SentiWordNet. SentiWordNet is a lexical resource for opinion mining. The resource contains words that express positive or negative emotion. These words are used in natural language to reflect the speakers opinion about an object. For example:

- **Positive** good (good food, good day) , excellent etc.
- **Negative** dreadful (dreadful weather), bad etc.

Some of these words can express both positive and negative emotion. This depends on the context in which they are used. For example the adjective *plain* can be interpreted as both positive (simple) and negative (not complex enough). Example:

- **Positive**: *He presented a plain solution*—interpret it as a simple solution;
- **Negative**: *He presented a plain dish*—interpret it as a dish without much taste.

Having this in mind to each of the words in SentiWordNet is assigned a degree of positivity and negativity that varies from 0 to 1. Example:

- *excellent*—positive 1.0, negative 0.0
- *plain*—positive 0.25, negative 0.3
- *dreadful*—positive 0.0, negative 0.7

This degree is calculated by taking into consideration the usages of each word in a large text collection.

Reading in the corpus data

The provided source file '`readerdata2b.lsp`' includes an implementation of the function '`read-reviews`' taking one argument, viz. the directory which contains the IMDB reviews. We have separated the corpus in two parts:

- `train`—a directory that contains 800 positive and 800 negative reviews. You can use this part to train your classifier.
- `test`—directory that contains 200 positive and 200 negative reviews. You can use this part to test your classifier.

If your data is stored in the directory '`~/inf4820/2b/`' you can use the following commands to load the data into the variables **train** and **test**.

```
(defparameter *train*
    (read-reviews "~/inf4820/2b/train"))
(defparameter *test*
    (read-reviews "~/inf4820/2b/test"))
```

[3] http://en.wikipedia.org/wiki/Sentiment_analysis

These functions reads into each variables a list of reviews that correspond to the corpus part. A review is represented using the following structure:

```
(defstruct review
  (class nil)
  (tokens nil)
)
```

where *class* is the category of the review—`:pos` or `:neg`—and tokens is a list of the words of the review. Example:

```
#S(REVIEW :CLASS :pos :TOKENS ("Matrix" "is" "great"))
```

The SentiWordNet words are loaded using the function `read-sentiwordnet-words`:

```
(defparameter *sentiwordnet-words*
    (read-sentiwordnet-words "~/inf4820/2b/sentiwordnet.txt"))
```

The variable `*sentiwordnet-words*` contains a hash-table with words from SentiWordNet as keys and values the following structures:

```
(defstruct sw
  (word "")
  (positive 0.0)
  (negative 0.0)
)
```

where field `word` represent the word and the fields `positive` and `negative` represent the degree of positivity and negativity of the word. Example:

```
#S(SW :WORD "excellent" :POSITIVE 1.0 :NEGATIVE 0.0)
```

1 Creating a Feature Vector

In this exercise we will experiment with creation of feature vectors. A feature vector is an n-dimensional vector of numerical features that represent some object. When approaching a classification task building a feature vector for each object to be classified is an important part of building a classifier. In exercise *2a* we have used the words c_i in a context of a word w as features to represent its semantics. For each feature i feature value was the frequency with which we find c_i in the context of the word w. In our sentiment analysis experiment the features should represent the option expressed in the movie review.

In this exercise we will implement functions that calculate feature vectors for each review. Each of this function will have as arguments a review and the `*sentiwordnet-words*` hash-table. The output of each function should be structure that contains feature vector. We will leave the implementation of this structure to your preference. The options to use are array, list, hash-table or association list. Check your or our solution to problem set (2a) for help.

(a) Write a function `make-word-feature-vector` that for each review uses as feature the frequency of the SentiWordNet words found. The approach is similar to the one we have used to build vectors for words. But this time we use the emotion charged words from SentiWordNet found in the review to represent the option expressed by the author.

For example if we consider as features the words '*good*', '*excellent*' and '*bad*' for the review:

> Good even excellent movie but with some bad actors.

the feature vector is:

```
good 1
bad 1
excellent 1
```

For the review

Good movie but with some bad actors.

the feature vector is:

```
good 1
bad 1
excellent 0
```

For the review

One of the bad movies this year.

the feature vector is:

```
good 0
bad 1
excellent 0
```

(b) Write a function `make-count-feature-vector` that for each review calculates the following features:

- **Feature1** - number of positive words found in the review (SentiWordNet words with degree of positivity more than 0.0) devided by the number of words in the review.
- **Feature2** - number of negative words found in the review (SentiWordNet words with degree of negativity more than 0.0) devided by the number of words in the review.
- **Feature3** - The average degree of positivity of all SetiWordNet words found in the review.
- **Feature4** - The average degree of negativity of all SetiWordNet words found in the review.

For example for the review:

Good even excellent movie but with some bad actors.

if the degree of positivity and negativity of the words is the following:

```
#S(SW :WORD "good" :POSITIVE 0.6 :NEGATIVE 0.0)
#S(SW :WORD "bad" :POSITIVE 0.0 :NEGATIVE 0.6)
#S(SW :WORD "excellent" :POSITIVE 1.0 :NEGATIVE 0.0)
```

The output of the function be the following:

```
feature1 - 2/9 // good & excellent
feature2 - 1/9 // bad
feature3 - (0.6 + 0.0 + 1.0)/3
feature4 - (0.0 + 0.0 + 0.6)/3
```

(c) Write a function `make-combined-feature-vector` that combines the features calculated in `make-count-feature-vector` and `make-word-feature-vector`.

2 Implementing a Rocchio Classifier

To classify out movie reviews we are going to use the Rocchio classifier.

(a) 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`' as its arguments the list of '`review`' structures, a function that will be used to calculate the feature vectors for each review and the `*sentiwordnet-words*` hash-table. The function should compute the length normalized centroids for the `pos` and `neg` class and stores into a structure of your choice.

(b) We now have all the pieces we need in order to write a Rocchio classifier and classify movie reviews as positive or negative. Recall that a Rocchio classifier assigns each word to the class which has the nearest centroid. Write a function '`rocchio-classify`' that for review 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, just as in the previous exercise. Example:

```
CL-USER(73): (setf count-classifier
  (compute-class-centroids
    *train*
    #'make-count-feature-vector
    *sentiwordnet-words*))

CL-USER(74): (rocchio-classify
    (nth 0 *test*) count-classifier)
"neg"
```

(c) Create three models using the `*train*` reviews and the three feature vector calculation functions. Calculate the accuracy of the obtained models on the `*test*` reviews. The accuracy of a model is calculated by dividing the number of correctly classified reviews by the number of all classified reviews. For example if we classify 4 reviews and our classifier correctly classifies 3 of them the accuracy is $3/4$ or 0.75.

3 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) *evaluation.* In a few sentences, discuss what do you consider a good accuracy result for the sentiment analysis task implemented in this exercise. Take into account the number of classes and the distribution of the examples in each class. done in order to evaluate our classifier here. Include some comments on the different strategies we could use for computing our evaluation scores.