600.466 - ASSIGNMENT 3
Lexical Ambiguity Resolution

Due date: Tuesday, April 9, 2013 in class

Overview

The goal of this assignment is build a general-purpose classifier for resolving various forms of lexical ambiguity. The two instances addressed here are word sense disambiguation and the classification of a proper name as a person, place, etc. Both are significant tasks in information extraction and text understanding, yet they are both remarkably similar and can borrow heavily from the vector models covered in Assignment 2.

Data

You have been provided with 3 data sets. The first consists of example sentences of the word *plant* in context, where each instance is labelled as one of the following senses:

plant/1 = *a manufacturing plant or factory*

plant/2 = *a living plant*

The second data set consists of example sentences of the word *tank* in context, each labelled as either:

tank/1 = *a military vehicle*

tank/2 = *a container*

In both cases, the labelled words are limited to just nouns (i.e. the verb forms *to plant a tree* or *to plant evidence* are excluded. Part of speech taggers are much more appropriate for making plant/verb vs. plant/noun distinctions than vector models are.

The third data set consists of proper names such as *Madison*, *Paris* and *Villahermosa* labelled as either sense 1=PERSON or 2=PLACE. Distinguishing between these two possibilities is a subset of the larger *named entity classification task*.

The actual format of the data is nearly identical to the vector representations used in Assignment 2, where each training sentence is equivalent to a "document". The only differences is that the start of document header (e.g. .I 2407) has a second numeric field (e.g. e.g. `.I 2407 1` or `.I 2407 2`) indicating whether that sentence is an example of sense 1 or 2 of the target word. Also, the target word itself is marked with an `.X`. For example:

```
.I 2407 1
Abrams
M1A1
.X-tank
divisions
were
...
```

```
.I 2408 2
Leaks
in
the
liquid
oxygen
.X-tank
threatened
....
```

By treating each of the example sentences as documents, sense classification can be treated exactly like a document routing problem in the vector model, where tagged training examples are used to create vector "profiles" of each of the target senses.

There are 5 data files associated with each target word, with formats essentially identical to the cacm files in Assignment 2:

```
tank.tokenized
tank.tokenized.hist
tank.stemmed
tank.stemmed.hist
tank.titles
```

`tank.tokenized` and `tank.stemmed` are contain the example sentences treated as documents, one word per line, as shown above. The first file is unstemmed, the second is stemmed. The `.hist` files contain corpus and document frequencies for each word, as before. And the `tank.titles` file contains one line for each example represented horizontally and labelled, for use like a document title when showing output. For example:

```
2407  1  MILITARY                ... Abrams M1A1 *tank* divisions were
2408  2  CONTAINER    Leaks in the liquid oxygen *tank* threatened ...
```

Finally, each data file contains 4000 training examples/"documents". The first 3600 are for training and the last 400 are for testing, as explained below.

Part 1 - Implementing the Vector Classification Model

The basic steps in the classification model are as follows:

1. Initialize the "document" vectors as in Assignment 1, where each example sentence is its own document. The weighting options will be discussed below. When encountering the start of a new example (e.g. `.I 2408 2`), store its assigned sense number in an array for later use (e.g. `$sensenum[2408]=2`).

2. From the first 3600 vectors (the "training" examples), create two new vectors $V_{profile1}$ and $V_{profile2}$, where $V_{profile-i}$ is the average (or *centroid*) of all of the training vectors labelled as sense i. The details of this process will be discussed in class.

3. For each of the remaining 400 "test" vectors (i=3600 to 3699), compute `$sim1 = &similarity($v_profile1, $vecs[$i])` and `$sim2 = &similarity($v_profile2, $vecs[$i])`. If $sim1 > sim2$ then label test example i as Sense 1, else Sense 2. As output, you should print the horizontal "title" line for the test document (`$titles[$i]`), along with $sim1$, $sim2$, the algorithm's choice of sense number, the correct sense number, and a `+` or `*` indicating if the algorithm's choice was correct or not. For ease of evaluation, you may also wish to print `$sim1-$sim2` and sort by this value. Large positive numbers will indicate examples that are strongly sense 1, large negative numbers will indicate examples that are ambiguous.

4. In Step 3, keep a running count of the total number of the test examples that your program classifys correctly and incorrectly. At the end, print out the percent correct ($\frac{total\_correct}{total\_correct+total\_incorrect}$).

Part 2 - Variations on the Model

Implement and explore the following variations on this basic model:

1. **Position weighting:** The base model assumes that all words surrounding an ambiguous word like *tank* contribute equal weight to its disambiguation, regardless of position. This is clearly not the case. Implement a weighting function sensitive to distance from the ambiguous word (e.g. `.X-tank`) when initializing the training vectors, much like you implemented region weighting in Assignment 2. Specifically, implement (1) a smooth exponential distance decay, where `$vecs[$docn]$term += 1/($distance_of_term_to_ambiguous_word)` - i.e. adjacent words have weight 1, words 2 away have weight 1/2, words 3 away have weight 1/3, etc. (2) Implement a stepped weighting function, where adjacent words are given weight 6.0, words 2-3 away are given weight 3.0 and where all other words are given weight 1.0. (3) Implement a weighting scheme of your own choice.

2. **Improvements on the bag of words model**: Regardless of the weighting scheme used, the basic vector approach is still a bag of words model. The examples ... *pesticide *plant** ... and ... **plant* pesticide* ... will be represented as the same vector. Options for capturing word sequence, at least locally, will be discussed in class. The simplest case would be to treat adjacent words (to the left and right) as special tokens:

```
1329  1  FACTORY    statements by the L-pesticide *plant* R-manager ...
1330  2  LIVING    is a very effective L-tropical *plant* R-pesticide ...
1331  1  FACTORY   pesticide often found on L-the *plant* R-roots and stems
```

   This would allow *pesticide* to contribute to the collective sense profiles for plant in different counters, depending on whether it occured immediately to the left, right or in another position. Alternately, those implementing a bigram model in Assignment 2 could modify this slightly to include only bigrams adjacent to the target word. Details will be discussed in class.

Part 3 - Evaluation

In order to test the effectiveness of the two variations above, as well as the effectiveness of the use of stemming in this case, you should test your system on the 6 different permutations of parameters, given below. For each permutation, print the accuracy of each (i.e. % correct) on the *plant*, *tank* and PERSON/PLACE data, as shown. Note that the values given below are only samples, and should not be consided to be values that your system should obtain. Descriptions of the 4 types of position weighting methods and the 2 kinds of proposed local collocation modeling methods are given in Part 2 of the assignment.

		Position	Local Collocation	ACCURACY		
	Stemming	Weighting	Modelling	tank	plant	pers/place
1	unstemmed	#0-uniform	#1-bag-of-words	.90	.84	.83
2	stemmed	#1-expndecay	#1-bag-of-words	.84	.80	.78
3	unstemmed	#1-expndecay	#1-bag-of-words			
4	unstemmed	#1-expndecay	#2-adjacent-separate-LR			
5	unstemmed	#2-stepped	#1 or #2 (specify)			
6	unstemmed	#3-yours	#1 or #2 (specify)			
7	Extra Credit: The results of your implemented extension					

Part 4 - Extensions to the Classification Model

For up to 50 points extra credit, students are strongly encouraged to implement one or more of the specified extensions to the vector model for sense disambiguation, with details provided in the hw3 class subdirectoryn the `NOTES` file. The total number of points of extra credit will be commensurate with the substance and quality of the implementation.

What to Turn in:

Please hand in your assignment both in hard copy and electronically. For hard copy submissions, please provide printouts of your code, commented sufficiently so that I can understand what you did solely from reading the comments. It is not necessary to include printouts of any large secondary data files, such as new class members.

To simplify the electronic submission of code and data files, you are required to submit a compressed tar file of your homeworks via a new automatic submission program.

Here are the steps:

1. Make a directory called hw3
2. Copy all the files of your program into hw3, including write-up, log files, etc.
3. Back up one level (cd ..) so that you are not inside the hw3 directory, but instead its parent directory.
4. Type: `tar -cvf - hw3 | gzip > jdoe.hw3.tar.gz` .. replacing "jdoe" with your user name.

Now, load your favorite web browser. We suggest using Netscape. Go to:
`http://www.ugrad.cs.jhu.edu/cgi-bin/cgiwrap/cs466/submit.pl`

Your password is the same as for previous assignment submissions.

If you have not selected a pasword for some reason, to request your password, go to option #3 and enter your login on the CS network. So if your e-mail address is catalina@cs.jhu.edu, your login is catalina and that is all that you have to type into the form.

Then click Continue. You will see a confirmation screen that says your password was sent via e-mail. Check your e-mail in a minute and you should have your new password.

Now go back to the web browser, your new password in hand. Go to the first form (submit a program). You need to enter your login again, your new password, choose the assignment you are submitting (Homework 3), pick the file that you are submitting (it will pop up a file browser and you have to find the myusername.hw3.tar.gz file that you just created), and then type the name of the file again in the last box. Then click Continue. It will now suck the file off your computer and drop it into your directory on our account.

Finally, you will probably want to check the file(s) you have submitted. To do this, go to the second form (view files I submitted) and enter your login, password and the assignment you want to check on. It will show you a listing of all the files that you have submitted for that assignment. Additionally, if for each file that is .tar or .tar.gz it will show you all the files that are inside that compressed file, so you will know that you compressed it correctly.

If you submit the same file more than once, it will over-write the old file, unless you give it a new name. So if you want to submit a correction or update and it is after the deadline, submit just the file(s) that have changed and not the whole tar.gz file again so we will know what you submitted and when.

Please contact one of the TAs if you have any problems with or questions about the new submission program.