Information Retrieval Course
CSE 435/535 - Fall 2009
Project 1
Document tokenizer in C++ for English and foreign language documents; Simple n-gram language identification

Project Description An IR Engine should include at least the following major components: Text parser, Indexer and Retrieval System. Your first project is to build the first component, the text parser which will be used by subsequent projects. It must be coded in C++. The purpose of this project is to let you get familiar with C++ STL (Standard Template Library).

Due Date **Online submission through UBLearns' Digital Dropbox by 25^{th} September, 2009 11:59 P.M.**

1 Parsing/tokenizing standard english text

1.1 In memory tokenization and dictionary creation

- **Tokenization** Read documents into memory, tokenize to separate words/tokens in the documents, process the tokens and store the "terms" in a suitable STL container. Tokenization rules will be discussed in class and recitation.
- **Term Dictionary** Build a "dictionary," which assigns each processed word/token to a numerical ID and keeps this correspondence information. Each unique processed token is assigned to the same unique numerical ID. The dictionary should be implemented such that element lookups are very fast.
- **File Dictionary** You also need to keep a Dictionary to map each document name to a unique numerical ID

NOTE: you don't need to invent a new STL container for this class project.
Document preprocessing steps typically include:

- Tokenization to handle numbers, **hyphens**, punctuation marks, and the case of letters (upper/lower)
- Elimination of stopwords
- Stemming of the remaining words

The dictionary will be a simple text file with each line formatted as <TermID>=<TermLabel> without the <'s and the >'s.

1.2 Data

We are using the TREC data, which contains multiple documents in a file and tags them separately. So you cannot treat each file as a single document, you need to parse them to separate documents. Files will be available from the course website through UBLearns.

Testing: Store the file dictionary and the term dictionary in two different files named fileIDName.txt and termIDLabel.txt These files will be inspected if you are asked for a demo.

2 Parsing/tokenizing Unicode UTF-8 text

2.1 In memory tokenization and dictionary creation

- **Tokenization** As before, read documents into memory, tokenize to separate words/tokens in the documents and store the tokens in a suitable STL container. You don't need to perform token processing using the rules for standard english text.
- **Dictionaries** As before, you need to create the token and file dictionaries. For UTF-8 encoded Unicode text, we are interested in obtaining the proper Unicode integer codes for each of the characters in a token. Each line of the token dictionary in this case should be formatted as <UnicodeTokenID>=< $UnicodeToken_{[0]}$: ... : $UnicodeToken_{[l]}$ > where $UnicodeToken_{[i]}$ is the proper Unicode value (decimal integer) for the i^{th} Unicode charater in the token. < $UnicodeToken_{[0]}$: ... : $UnicodeToken_{[l]}$ > may be stored as a string again without the <'s and the >'s.
- **Byte format and codes** You can find the UTF-8 Unicode byte format at `http://en.wikipedia.org/wiki/UTF-8`. A full listing of Unicode charater codes can be found at `http://www.buffalo.edu/~pdas3/activities/utf8unicodes.html`.

2.2 Data

Multilingual data from multiple sources. The French, Spanish and German data are collected from `http://www.statmt.org/europarl/` and thanks to Smruthi Mukund for sharing the cleaned Urdu text files. This dataset is organized into 4 top level folders one each for German, Spanish, French and Urdu documents Further each folder has 2 subfolders called Training and Validation.

Testing: Store the file dictionary and the unicode term dictionary in two different files named fileIDName.txt and utermIDLabel.txt These files will be inspected if you are asked for a demo.

3 Language Identification Module

3.1 Unigram and Bigram word and character probabilities

In this section we will evaluate a basic language model using simple frequency estimates. Let us suppose that in a document, $word_i$ follow $word_{i-1}$ and that label END follows the last word in the document. We could calculate

$$P(w_i|w_{i-1}) = \frac{P(w_{i-1}w_i)}{P(w_{i-1})}$$

and predict that for e.g. "I am" is more likely than "I the." If there are N words in a document, there can be N word bigrams *in the document* obtained by moving a sliding window of 2 words from the beginning to the end of the document. The total number of possible word bigrams can be V^2 where V is the size of the word vocabulary.

We could also calculate $P(c_{i-1}c_i)$ in a corpus as

$$\frac{\#\,of\ c_{i-1}c_i}{\#of\ all\ possible\ c_{j-1}c_j\ s}$$

, where c_{i-1} and c_i are the $i-1^{th}$ and the i^{th} characters in a space separated word. Note that c_i could be a space character if c_{i-1} happens to be the last character in a word. Since for foreign languages, the size of each character code may not be 7 bits as for ascii characters, we will encode characters in foreign language in UTF-8 Unicode integer values for e.g. (105 243) is a proper character bigram in Spanish denoting i*ó* as in "sesión". Your Unicode tokenizer should already have taken care of this.

Note that no matter how large the corpus is, the number of character bigrams again will always be $C \times C$ where C is the total number of characters in the alphabet set. (why?)

3.2 Estimation of bigram character probabilities from Unicode data

From the section 2, create the following dictionaries:

- *for *all* foreign languages training set folders*: character bigram label to unique id mapping. The bigram label should be stored in the text format as $< UnicodeTokenCharacter_{[0]} : UnicodeTokenCharacter_{[1]} >$. The last Unicode character in a token can have the second member in the bigram to be a space character. As before $< UnicodeTokenCharacter_{[0]} : UnicodeTokenCharacter_{[1]} >$ may be stored as a string.
- *for *each* foreign language training set folder*: character bigram ID to count mappings. Normalize the counts to a real number between 0 and 1.
- *for *each* foreign language validation set folder*: character bigram ID to count mappings. Normalize the counts to a real number between 0 and 1. For Unicode characters not found in the training set, assign the label UNKNOWN (any special value you like for e.g. 0). You could then encounter 3 possibilities like $< UnicodeTokenCharacter_{[0]} : 0 >$, $< 0 : UnicodeTokenCharacter_{[0]} >$ and $< 0 : 0 >$. Compute the normalized counts of these counts also.

To generate a global dictionary of character bigrams, you will need to access all the training and validation files of **all** foreign languages. Alternately you can copy the files in one single directory and do the processing.

We thus have the following at our disposal: for each foreign language training set corpus, we could compute a histogram of normalized counts over character bigrams. These counts in turn represent the empirical probability mass function (p.m.f) estimates of the character bigrams. Let, for a particular foreign language L, $p_{L,T}$ denote the p.m.f. of all character bigrams in the training set of documents in language L. Also denote q_{L,V_d} to be the p.m.f. of all character bigrams in **a particular document** in the validation set of documents in language L. The Kullback-Leibler (KL) divergence is an information theoretic measure calculated by the formula,

$$\sum_{all\_character\_bigrams(.)} p(.) \log p(.)/q(.)$$

where $p(.)$, wherever defined, is the value of the p.m.f at a certain character bigram and $q(.)$, wherever defined is the value of the p.m.f. at the **same** character bigram. Treat $\log(0.0)$ or $\log p(.)/0.0$ to be 0 so that the term has no effect on the summation. Intuitively, KL divergence gives us an estimate of the expected number of questions that must be asked to get from the guessed $q(.)$ to the true $p(.)$.

3.3 Simple Language Identification module

- For **each document** in the **validation set folder** of the one foreign languages e.g. German, compute the probability mass function estimate of the character bigrams. Compute the KL divergence score of the document in the German test set with respect to each of the four sets of p's in the other foreign languages including it's own language. So for each validation document, you will obtain 4 possible scores. Create four tables for these scores. *Note that the summation in the KL divergence formula in this case is over all matching character bigrams. For e.g. German and*

French can have the same character bigrams that has the same ID in the global character bigram dictionary.

- Comment on which score is the lowest for each validation set document in each foreign language. How would you use this score to determine the language of a document?
- Save all the character bigram IDs including unknown bigrams in a single dictionary and also save all the normalized bigram ID to count mappings for each language in different files. Write a module that accepts a single test file in UTF-8 and outputs "German", "Spanish", "French" or "Urdu" based on the language determined using the comparison of the KL divergence values from the four training sets. This module will be tested during your demo.

Please ask at least one your project members to attend recitation for information on which tools and IDE to use for developing your programs. I personally use Eclipse CDT on Ubuntu. If you use Eclipse, the makefiles will automatically be generated for you. If you prefer to use windows, install Cygwin (`http://www.cygwin.com/`) and then extract Eclipse CDT for windows. In the latter case, the Cygwin GCC toolchain should automatically be selected for compilation. All programs must be coded in C++ and **should not be** in only 1 or 2 (one header and one source) file(s). Points will be allocated for clean and modular design. Heavy use of virtual functions (if any) is discouraged for these class projects. More on this on recitations.

4 Submission

Submit a tar or zip file named <your-team-name>.tar or <your-team-name>.zip using the Digital Dropbox in UBLearns. The contents of the compressed file is as follows:

−− <your-team-name>.tar
|+ english_tokenizer.tar (All source and make files + dictionary files)
|+ multilingual_tokenizer.tar (All source and make files + dictionary files)
|+ document_language_identifier.tar (All source and make files + a pdf file showing the four tables from section 3.3)

Make sure all your programs at least compile on at at least one CSE machine: (e.g. nickelback.cse.buffalo.edu [running Linux]) During the day of the demo, the TAs will download your most recent tar files (upto the submission date) from UBLearns and run it.