A Language Identification Application Built on the Java Client/Server Platform

Gary Adams
Sun Microsystems Laboratories
Two Elizabeth Drive
Chelmsford, MA 01824-4195, USA
gary.adams@east.sun.com

Philip Resnik
Dept. of Linguistics and UMIACS
University of Maryland
College Park, MD 20742, USA
resnik@umiacs.umd.edu

Abstract

We describe an experimental system implemented using the Java(TM) programming language which demonstrates a variety of application-level tradeoffs available to distributed natural language processing (NLP) applications. In the context of the World Wide Web (WWW), it is possible to provide value added functionality to legacy documents in a client side browser, a document server or an intermediary agent. Using a well-known ngram-based algorithm for automatic language identification, we have constructed a system to dynamically add language labels for whole documents and text fragments. We have experimented with several client/server configurations, and present the results of tradeoffs made between labelling accuracy and the size/completeness of the language models .

1 Introduction

In the 1.1 release of the Java Developers Kit(TM), a wide selection of text processing and internationalization interfaces have been added to the base Java package[1] making the package usable for multilingual text processing. The Java programming language, the portable Java virtual machine and the basic web infrastructure of client web browsers and document resource protocols provide a widely deployed platform suitable for distributing NLP applications. Our research is targeted at shallow machine translation and summarization of multilingual web pages. [2] To properly bootstrap this technology we require appropriate language labels on documents. Language labels may be present at a whole document or collection of documents level for large granularity applications or at a structural component (SGML entity level) for fine grained uses. (Yergeau et al., 1997) Using an ngram language model (Dunning, 1994), we have explored a variety of mechanisms for adding language labels to legacy documents as part of the normal end user experience of the World Wide Web. Three obvious places the language labels could be added to legacy documents are within an end user web browser, within a document repository server and within an intermediary proxy server. We have experimented with several client/server configurations, and present the results of tradeoffs made between labelling accuracy and the size/completeness of the language models .

2 Automatic Language Identification

Although the general framework will support a variety of algorithms for automatic language identification, our implementation is based on Dunning's (1994) character ngram approach, which is conceptually quite simple and achieves good performance even given relatively small training sets (50K of training text is more than enough, and one can make do with as little as 1-2K), and even when the strings to be classified are quite short (e.g., 50 characters). Essentially, the method involves constructing a probabilistic model (or "profile") based on character ngrams for each language in the classification set, and then performing classification of an

[1] A few pointers to online Java resources.
Java Developer Kit :
<URL:http://www.javasoft.com/products/jdk/1.1/index.html>
Java Server(TM) Product Family :
<URL:http://jeeves.javasoft.com/products/java-server/index.html>
HotJava(TM) Browser :
<URL:http://www.javasoft.com/products/HotJava/index.html>
Java internationalization :
<URL:http://java.sun.com/products/jdk/1.1/docs/guide/intl/index.html>
Java Workshop :
<URL:http://www.sun.com/workshop/index.html>
Java JIT :
<URL:http://www.sun.com/workshop/java/jit/index.html>

[2] Sun Microsystems Laboratories efforts in International Linguistic Applications:
<URL:http://www.sunlabs.com/research/ila/>.

unknown string by selecting the model most likely to have generated that string.

Although the model itself is quite simple, some subtle and not-so-subtle issues do arise in putting the algorithm into practice. First among these is the problem of matching the character set of the input text to the character set assumed by the language profiles — for example, Shift-JIS and EUC-Japan are frequently used to encode Japanese documents in the PC and Unix worlds, respectively. Documents currently found on the WWW are often insufficiently labeled to indicate the language of text or the encoding of the characters within the document. We believe that use of Unicode will become increasingly widespread, obviating this problem, although for the time being we avail ourselves of the reasonable tools available[3] for identifying and converting among character sets.

Second, sparse training data is a significant factor in ngram modeling, even at the level of character co-occurrences, since we consider character sequences up to 5 characters in length. We have experimented with simple add-k smoothing and, noting known problems with that method (Gale and Church, 1990), we have also experimented with Good-Turing smoothing (Good, 1953) – finding, to our surprise, that the simple "add $\frac{1}{2}$" is only slightly less accurate.

Table 2 shows the performance of the language identification algorithm when run on Dunning's (1994) English/Spanish test set, using language profiles for English and Spanish constructed from training sets of 50K characters each and varying the size of the ngrams and length of the test strings. This experiment used Good-Turing smoothing and also adopted a simplified approximation of conditional probabilities used by Dunning (personal communication) in his experiments. Each row of the table shows the language and length of the test strings, the ngram size, the number of test strings classified correctly and incorrectly, and the percentage correct.

Third, in an environment where computation may more efficiently be done on the client side rather than the server side, the size of the language profiles becomes relevant, since computation cost must be traded off against communication cost for the data needed in order to perform the classification. Given that probabilities for low frequency items can be poorly estimated anyway, we have experimented with eliminating low-frequency items from the profile — e.g., treating singletons (ngrams appearing just once in the training data) as if they never appeared at all and using the smoothed-zero value for them instead, thus trading model size against classification accuracy. Again to our surprise, we have found that quite reasonable classification performance is sustained even when filtering out not only singletons but even ngrams that appear twice and even three times in the training set. (see Tables 3-5).

Table 1 shows the dramatic size reduction that takes place as smaller window sizes are used in training the language models. In the current implementation plain text files are used for maximum portability of the resources. An application that uses a 5-gram model without filtering any of the training data would use a 220K model containing 13K observed ngrams, with an average accuracy of 98.68% for 100-500 character length strings. If the same application can function effectively with a marginally lower accuracy rate of 98.32%, then the same training data can be used to produce a profile a full order of magnitude smaller (a 23K profile containing only 1.6K ngrams), by using a trigram model and filtering out those trigrams whose observed frequency is less than 4. This 10X reduction in size for this particular resource could mean supporting 10 times as many languages with the same memory footprint or delivering the linguistic resource 10 times as fast for a client side computation.

Finally, standardization of language labels must be addressed; this work follows the ISO standards for language and country codes for internationalization (ISO, 1988b; ISO, 1988a; Yergeau et al., 1997; Alvestrand, 1995).

3 Client/Server Architecture

In designing a distributed application several decisions can be built into the architecture of the product or left as runtime decisions. By using a Java virtual machine as the target platform, the same code can run on a server machine or within the client graphical user interface. A sophisticated program can determine at startup whether the computation resources (memory and CPU) on the server machine or on the client workstation are better for the more complex algorithms.(In our testing we work with a SparcStation(TM) 10 file server, an Ultra1(TM) with 500M of memory as a high end client and a SparcStation 2 remote client over a 28.8 Kb modem connection as a low end client.)

In addition to compute resources, it is also important to consider the network bandwidth resources. The local area network configuration can make some simplifying assumptions that may not be appropriate for wide area network and remotely connected clients. We have explored the possibility of degraded application performance in exchange for reasonable response times for the remotely connected client, i.e., we have allowed a higher error rate on language labels of short text fragments in exchange for smaller language models which can easily be down-loaded over slower network connections by remote sites. In

[3] Xemacs internationalization : <URL:http://www.xemacs.org/xemacs-faq.html#internationalization>

this section, we discuss the incorporation of language identification at three possible locations: the client's Web browser, the document server, and between them at a proxy Web server.

3.1 Client Web Browser

We have experimented with two extreme client configurations. Our high end client has fast CPU and a large memory pool. Our low end client has both a slow CPU and small memory footprint. The high end client easily caches large language profiles and is capable of computing the best possible language labels. When the network resources are available to the high end client, it makes the most sense to perform the language labeling within the client browser.

On the low end client, the available network bandwidth was the driving architectural consideration. When high bandwidth was available, delegating computations to the server system provides the best language labels and the best throughput to end users. In disconnected or low bandwidth situations, the client must perform its own labeling. In these situations, less accurate language labels with reasonable responsiveness is preferred over slow but more correct results.

Three primary techniques were used to improve the responsiveness of the client side language labelling interfaces. Basically, they all attempt to minimize the work that is performed and to overlap the work whenever possible with other end user interactions.

- *Asynchronous processing for perceived responsiveness.* End users perceive system responsiveness in terms of its ability to react to their requests when they are presented to the system. Within our application there are clear points during system initialization and end user parameter selection when large amounts of network bandwidth and computation resources are needed. Using the builtin threading capabilities of the Java environment, we start the resource intensive operations when they are indicated, but allow the user to continue interacting with the user interface. If the user requests an operation that requires an uninitialized resource a message is presented and the application blocks until the resource is available.
- *Degraded language profiles for smaller footprint.* Our language identification profiles have been built with 3, 4 and 5 character ngram windows. In addition to varying the ngram window size we have experimented with removal of singleton and doubleton observations in the training data. While this amplifies the sparse data problem, it does not significantly impact the end user perceived error rates for large granularity text objects, e.g., labeling a typical webpage with 1000 characters of textual information.
- *Subset of language profiles for specific user needs.* We have been working with a mixture of western European and Asian languages[4]. For remote clients it is worth the extra effort to preselect the languages that will be most beneficial to distinguish on the client machine. For demonstration purposes we use a dozen western languages and preload a few profiles during the initialization of the sample configuration.

3.2 Document Server

A typical document server is designed to service a large number of end user requests. While they are usually configured with large amounts of disk storage, they are not always the best computational resources available on the network. For static webpages, it is easy to include a language labeling tool for off-line document management. The labeling tool would be used to convert *text/html* file into *message/http* files.[5]

For real time information such as news wires or other database generated replies the same off line language labeling tools could be used with Common Gateway Interface (CGI)[6] scripts to automatically add language labels to dynamically generated webpages.

3.3 Proxy Web Server

A proxy server is an intermediary program that provides value added functionality to documents as part of the transmission process. A proxy server could be configured close to the end user or close to the data source depending on the network topology. Proxy servers may also be employed as a shared workgroup or enterprise wide facility, e.g., department level proxies can share cached webpages, or an enterprise wide proxy could add an extra level of access controls. By introducing language labeling at a proxy server it is possible to combine the benefits of webpage caching and transparent content negotiation to reuse previously computed headers.[7]

[4] We selected 200K of sample text from the WWW for the following languages: Chinese, Czech, Danish, Dutch, English, Finnish, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Spanish, Swedish, and Turkish. We use 50K of text for training the models, 50K for use in entropy calculations and 100K heldout for testing purposes. Preliminary experiments indicate that performance comparable to what we have seen with the English/Spanish test set will also be achieved with other language pairs.

[5] Apache server documentation for variant files:
<URL:http://www.apache.org/docs/mod/mod_asis.html>
<URL:http://www.apache.org/docs/mod/mod_negotiation.html>

[6] Common Gateway Interface 1.1 :
<URL:http://hoohoo.ncsa.uiuc.edu/cgi/>

[7] Tarnsparent content negotiation in HTTP :
<URL:http://gewis.win.tue.nl/

4 Java Classes

The primary reusable Java module written for language labeling in our system is called a *frequency table class*. A *main()* routine is provided in the class to provide a stand-alone interface for generating new language profiles from training data. The generated language profiles are self documenting text files indicating the parameters used in creating the language model and the algorithms used for smoothing and filtering the training data. Methods are provided in the *frequency table class* for saving and loading the profiles to disk and for scoring individual strings from a loaded profile.

Specialized classes were also written to provide connections within a client environment (in Java lingo an "applet") and within a proxy HTTP server (again in Java lingo a "servlet"). In both the servlet and applet applications of the language labeling class the Java platform provided the basic class loading infrastructure to allow a common shared module and the distributed platform for running those algorithms transparently on a client or server system.

5 Discussion

In this paper, we have described a Java[8] implementation of a character ngram language labeling algorithm. This NLP module was successfully reused in a client side Java application, in an offline document management system and embedded within an HTTP proxy server. With the rapid deployment of the globally available Java infrastructure, a tremendous opportunity exists for resusable NLP components.

The distributed nature of our particular application, led us to explore possible tradeoffs between the accuracy needed for client side language labeling and the size of the language models. By selecting smaller ngram windows sizes and by disgarding infrequently observed ngrams from our language profiles we can reduce the size of the models by an order of magnitude with an insignificant loss of precision for our target application.

The tradeoffs we have explored in the context of automatic language identification are relevant more generally to natural language processing in the distributed setting made possible by the Java infrastructure. At a minimum, our observations with respect to character-based language models are likely to be applicable to the word-based language models used in other statistically-driven NLP applications. Beyond that, similar client/server tradeoffs are likely to be important even in strictly knowledge based systems. Part-of-speech tagging and phrase identification, foreign word translation, and topic labeling are among the operations that promise to enhance intelligent search and browsing on the Web, and the present paper represents a beginning step toward making decisions about where to locate these operations' computations and data.

N	Filter	Lines	Bytes	% Correct
5	0	13423	221996	98.68
4	0	8196	127917	98.66
3	0	3557	52144	98.38
5	1	6003	95858	98.64
4	1	4607	70496	98.54
3	1	2393	34688	98.36
5	2	3899	62704	98.58
4	2	3377	50973	98.52
3	2	1952	27646	98.32
5	3	2863	45782	97.84
4	3	2685	39464	98.50
3	3	1693	23527	98.32

Table 1: Language profile sizes

References

H. Alvestrand. 1995. RFC 1766: Tags for the identification of languages, March.

Ted Dunning. 1994. Statistical identification of language. Computing Research Laboratory Technical Memo MCCS 94-273, New Mexico State University, Las Cruces, New Mexico.

W. Gale and K. Church. 1990. What's wrong with adding one? *IEEE transactions on Acoustics, Speech, and Signal Processing.*

I.J. Good. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3 and 4):237–264.

ISO. 1988a. ISO3166 - codes for the representation of names of countries.

ISO. 1988b. ISO639 - code for the representation of names of languages.

F. Yergeau, G. Nicol, G. Adams, and M. Duerst. 1997. RFC 2070: Internationalization of the hypertext markup language, January.

koen/conneg/>
IETF - HTTP Working Group :
<URL:http://www.ics.uci.edu/
pub/ietf/http/>

[8]Sun, Java, Java Developers Kit, HotJava, and Ultral are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Lang	Length	N	Correct	Wrong	% Correct
En	100	3	983	17	98.3
Sp	100	3	973	7	99.3
En	200	3	984	16	98.4
Sp	200	3	981	19	98.1
En	500	3	498	2	99.6
Sp	500	3	500	0	100.0
En	100	4	988	12	98.8
Sp	100	4	976	4	99.6
En	200	4	986	14	98.6
Sp	200	4	983	17	98.3
En	500	4	500	0	100.0
Sp	500	4	500	0	100.0
En	100	5	991	9	99.1
Sp	100	5	973	7	99.3
En	200	5	986	14	98.6
Sp	200	5	984	16	98.4
En	500	5	500	0	100.0
Sp	500	5	500	0	100.0

Table 2: Language identification performance using full profiles

Lang	Length	N	Correct	Wrong	% Correct
En	100	3	986	14	98.60
Sp	100	3	972	8	99.18
En	200	3	979	21	97.90
Sp	200	3	981	19	98.10
En	500	3	498	2	99.60
Sp	500	3	500	0	100.00
En	100	4	989	11	98.90
Sp	100	4	973	7	99.29
En	200	4	983	17	98.30
Sp	200	4	982	18	98.20
En	500	4	499	1	99.80
Sp	500	4	500	0	100.00
En	100	5	989	11	98.90
Sp	100	5	973	7	99.29
En	200	5	982	18	98.20
Sp	200	5	985	15	98.50
En	500	5	500	0	100.00
Sp	500	5	500	0	100.00

Table 4: Language identification performance using reduced profiles (filtered doubletons)

Lang	Length	N	Correct	Wrong	% Correct
En	100	3	986	14	98.60
Sp	100	3	973	7	99.29
En	200	3	980	20	98.00
Sp	200	3	981	19	98.10
En	500	3	498	2	99.60
Sp	500	3	500	0	100.00
En	100	4	989	11	98.90
Sp	100	4	973	7	99.29
En	200	4	982	18	98.20
Sp	200	4	983	17	98.30
En	500	4	500	0	100.00
Sp	500	4	500	0	100.00
En	100	5	990	10	99.00
Sp	100	5	974	6	99.39
En	200	5	984	16	98.40
Sp	200	5	984	16	98.40
En	500	5	500	0	100.00
Sp	500	5	500	0	100.00

Table 3: Language identification performance using reduced profiles (filtered singletons)

Lang	Length	N	Correct	Wrong	% Correct
En	100	3	984	16	98.40
Sp	100	3	973	7	99.29
En	200	3	979	21	97.90
Sp	200	3	981	19	98.10
En	500	3	499	1	99.80
Sp	500	3	500	0	100.00
En	100	4	990	10	99.00
Sp	100	4	972	8	99.18
En	200	4	983	17	98.30
Sp	200	4	981	19	98.10
En	500	4	499	1	99.80
Sp	500	4	500	0	100.00
En	100	5	979	21	97.90
Sp	100	5	961	19	98.06
En	200	5	974	26	97.40
Sp	200	5	985	15	98.50
En	500	5	497	3	99.40
Sp	500	5	496	4	99.20

Table 5: Language identification performance using reduced profiles (filtered tripletons)