Automatic Identification of Language and Encoding

Graham Russell Guy Lapalme

Abstract

The ability to automatically identify the language and encoding of a written document is a prerequisite to virtually all types of processing in a multilingual environment. Although it is frequently taken for granted in the language engineering community, a general solution poses some significant practical and theoretical problems.

This paper has three objectives. First, it surveys the history and current state of the art from the perspective of desirable system qualities; in addition to efficiency and accuracy, these include generality and extensibility. It then describes the properties of written language and character encodings which give rise to the difficulty of the task. Finally, it presents SILC, a high-performance language and encoding identifier developed at RALI; the current commercial version of SILC is equipped to detect over one hundred language-encoding combinations.

1 Introduction

As far as document processing is concerned, the modern computing environment can be characterized as *multilingual* and *multiencoding*. Increased use of the internet, especially in its world-wide web manifestation, together with growing international communication for commercial, political and social purposes has led to a proliferation of machine-readable texts in multiple natural languages; these texts are encoded in a highly variable manner, depending on language-specific conventions and incompatible standards.[1]

At the same time, the requirement for automated text handling is increasing, with applications ranging from the basic functions of screen display and printing to indexing and search, quality checking and correction, classification, information extraction and translation. All of these presuppose the ability to recognize the language of the text and/or its encoding: detection of encoding is clearly a prerequisite for correctly displaying or printing a document, while more advanced techniques depend at a minimum on identifying the words and other units of which the text is composed. The more sophisticated any higher-level linguistic processing is, the more language-specific the information it employs.

[1]Documents also circulate in numerous formats — as "plain text", PDF, PostScript, HTML, XML, in proprietary word-processor formats, etc. Automatic detection of document format beyond what is indicated by filename, file headers or MIME types, while potentially amenable to some of the methods discussed below, falls outside the scope of this article.

Automatic language and encoding identification (LEI) is therefore a basic task in the modern computing environment, one that at first sight may seem straightforward. Many proposals have appeared over the years (section 3), and impressive performance has frequently been claimed. However, attaining the objective of general, extensible, accurate and efficient LEI still poses significant technical challenges.

In essence, LEI is a classification task. Initially, a number of categories are established, each associated with a distinct language/encoding pair. An input text is examined in order to assign it to the category that it most closely matches; the language and encoding associated with that category are then taken to be those of the text in question.

It may be tempting to think that technological developments will reduce the need for LEI. At first sight, a number of tendencies seem to support this position: the gradual adoption of Unicode, greater use of XML as a format for document exchange (employing either the default UTF-8 encoding or some explicitly indicated alternative), and various proposals for content description schemes capable of indicating language and encoding. However, this ignores two factors: few existing 'legacy' texts will be converted or annotated, and any annotations which are supplied will still have to be treated as potentially erroneous, regardless of whether they are supplied by human users or document creation and management tools. In this connection, it may be instructive to consider the current situation with regard to the rendering of HTML documents in web browsers: even though HTML provides for the necessary annotations (via the `lang` and `http-equiv/content` attributes), users are still obliged to select encodings manually, and still receive documents which display incorrectly.

It is useful to distinguish several senses of the term 'language identification'. This paper confines itself to the domain referred to in the first paragraph, for the type of purpose mentioned in the second: the problems addressed involve written texts in character-based form (rather than speech signals or bitmap images), chiefly composed of one or more natural languages. Language identification in this sense has obvious affinities with the kind of text metrics underlying work on stylistic analysis, authorship ascription, and so on [Kilgariff, 1996].

In the domain of speech technology, language identification has been an important research topic for a number of years, also covering areas such as detection and accommodation of foreign and other non-standard accents. Clearly, spoken utterances are not subject to the same variable encoding methods as text, so recognition of the encoding is not part of the spoken language detection task.

A certain amount of work has also been done [Sibun and Spitz, 1994] on identifying the language of text images, often in the context of establishing correct parameters for subsequent optical character recognition. For this application, detection is based on the distribution of apparently relatively crude features of character shapes: height above and below baseline, presence of ascenders and descenders, placement of curves, etc. Here again, issues related to character encoding are not relevant.

Finally, there is the task of deciding whether an arbitrary data stream contains features of a linguistic nature, regardless of the actual language in question; at least one such project has been carried out in the context of the SETI program [Elliott et al., 2002].

We first describe the different types of language and encoding identification that can be

encountered and we specify the one that is addressed in this paper. Section 2 discusses the relations between encoding and language, details different levels involved in the encoding of characters and describes some problems in distinguishing between similar encodings. Section 3 defines the properties expected from a good language identifier. Section 4 presents SILC, the production quality language and encoding identification system we have developed and its implementation. Section 5 gives the results of an evaluation of SILC and compares it with some other LEI systems.

2 Languages and Encodings

2.1 Language-Encoding Dependencies

We have been presenting the issues in terms of a joint problem: how to identify both the language and the encoding of a text. It is not a priori obvious they need to be handled in the same way or indeed at the same time. Some applications need one but not the other: for example, in a word processing application, language identification can be used for choosing the appropriate spelling checker for a text span; in this case, the word processor knows the encoding it uses so only the language needs to be determined. It is thus necessary to consider whether the two aspects of the problem would not better be approached separately.

It is clear that the need to handle many different languages brings with it the need to handle many encodings. Some encodings are "universal" (UTF-8 and other Unicode realizations), others are specific to a single language (EUC-KR is used only for Korean, ISO 8859-8 for Hebrew), and some are used for a class of languages whose individual orthographies are sufficiently similar. Familiar examples of the last category are some members of the ISO 8859 family: those for West European, Latin-based Central and East European, Cyrillic-based, etc. systems. Ideally, the system should be able to make use of such observations in appropriate cases in order to strengthen its hypotheses; if it were possible to identify the encoding with certainty as EUC-KR, for example, the problem of identifying the language would be solved. Rather less helpful would be the discovery that the encoding was ISO 8859-2; this would narrow the possibilities to several latin-alphabet Eastern and Central European languages, but still leave some work to be done. Knowing that a text is encoded in UTF-8, on the other hand, provides no further information.

Intuitively, the problem in the general case is the following: in order to determine the language of a text, we must know which characters are represented by each byte of the text; this depends on the character encoding of the text, and this is in turn strongly related to its language. This is why we adopted the principle that *simultaneity is the remedy for circularity.*

2.2 Character Encodings

In general, a LEI system has no direct access to even the character-level content of a text; all that is available is a sequence of bytes within some file or input stream. The relation

between bytes and characters is complex to a degree that often surprises those unfamiliar with the problem. In this section we spell out some of the reasons for this complexity.

There is a rich literature on character encoding issues. Dürst [Dürst et al., 2002] give a good overview, and [Lunde, 1999] covers East Asian encodings in detail. The following presentation draws heavily on Whistler and Davis [Whistler and Davis, 2000].

We consider a text T as a sequence of N characters $[C_1, \ldots, C_i, \ldots, C_N]$ to be encoded as B, a sequence of L bytes $[b_1, \ldots, b_j, \ldots, b_L]$. What we refer to as a character encoding is termed by Whistler and Davis a 'Character Map': a mapping from an 'abstract character repertoire' to a 'serialized stream of bytes'. This can be defined extensionally as a relation between characters and byte sequences:

$$CM : \{\langle C, [b_1, \ldots, b_l]\rangle_k\}$$

This relation is mediated by four levels of representation.[2]

(1) Abstract Character Repertoire: the set of characters to be encoded, as normally conceived of, i.e. objects such as '*F*', '*š*', '*!*'; these are abstract in that they typically have varying graphical representations.

$$ACR : \{C_k\}$$

In formal language theory terms, this set of symbols is the *alphabet*; naturally, it typically contains elements not normally thought of as 'alphabetic', e.g. numerals, punctuation, and certain blank characters representing spaces, newlines, etc.

(2) Coded Character Set: a mapping from an ACR to a set of non-negative integers.

$$CCS : \{C_k\} \rightarrow \{c_k\},\ c_k \geq 0$$

Note that the range of a CCS generally exceeds what can be represented in a single byte. Examples include ISO 8859-1 and Unicode Standard Version 3.0; some instances of a CCS are known as 'code pages'. Chinese, Japanese and Korean character inventories are customarily presented in a tabular 'row-cell' form; here, each $\langle \text{row}, \text{cell}\rangle$ index pair corresponds to a distinct integer.

(3) Character Encoding Form: a mapping from a CCS integer set to a set of sequences of code units of some specified width.

$$CEF : \{c_k\} \rightarrow \{[u_1, \ldots, u_l]_k\},\ l \geq 1$$

For a given CEF, the width of the code units may be fixed or variable. Fixed-width options include seven, eight, sixteen and thirty-two bits. Variable-width CEFs include UTF-8 (one to four eight-bit units) and UTF-16 (one or two sixteen-bit units).

[2]Whistler and Davis also present a fifth level, not used in this paper, the **Transfer Encoding Syntax** which is a reversible transform of encoded data that may or may not contain textual data (e.g. base64, uuencode, BinHex, gzip).

LC	C_k	*CM*	hexadecimal b_j
en	Hello world	ISO 8859-1	48 65 6C 6C 6F 20 77 6F 72 6C 64
		UTF-8	48 65 6C 6C 6F 20 77 6F 72 6C 64
fr	Bonjour les dégats	ISO 8859-1	42 6F 6E 6A 6F 75 72 20 6C 65 73 20 64 E9 67 61 74 73
		UTF-8	42 6F 6E 6A 6F 75 72 20 6C 65 73 20 64 C3 A9 67 61 74 73
de	Grüß Gott	ISO 8859-1	47 72 FC DF 20 47 6F 74 74
		UTF-8	47 72 C3 BC C3 9F 20 47 6F 74 74

Figure 1: Some examples of texts, showing the hexadecimal codes in latin-1 and UTF-8 encodings.

(4) Character Encoding Scheme: a mapping from CEF code units to a serialized sequence of bytes.

$$CES : \{[u_1, \ldots, u_l]_k\} \rightarrow \{[b_1, \ldots, b_m]_k\},\ l, m \geq 1$$

For our purposes, there are essentially two reasons for distinguishing the notions of CEF and CES. First, each CEF applies to a single CCS, while a CES may accommodate two or more; such 'compound' CESs include those based on ISO 2022, in which escape sequences indicate the CCS in use, and the EUC variants, in which the mapping is controlled by specified shift operators. The second reason is that code units wider than eight bits are subject to considerations of differential byte-ordering. Thus there exist both UTF-16BE ('big-endian') and UTF-16LE ('little-endian').

In certain cases, the relation *CES* is trivial: each c in the codomain of the CCS translates as the byte whose numerical value is c. Making the customary assumption of 8-bit bytes, this clearly applies only to CCSs assigning codes $0 \leq c \leq 255$.

The character map can then be defined as the composition of *CES*, *CEF* and *CCS*:

$$CM : CCS \cdot CEF \cdot CES \equiv \{C_k\} \rightarrow \{[b_0, \ldots, b_L]_k\}$$

for which we can consider $C(E)$ and $B(E)$ to be the domain and range respectively of the encoding E, i.e.

$$C(E) = \left\{c : \exists b^+ \langle c, b^+ \rangle \in CM\right\} \quad (1)$$

$$B(E) = \left\{b^+ : \exists c\, \langle c, b^+ \rangle \in CM\right\} \quad (2)$$

Figure 1 shows some examples of encoded characters in different languages.

It is worth emphasizing three problems implicit in this discussion that will be faced by a LEI system intended to handle the full range of encoding types:

(i) it will not generally know in advance whether a given byte in the input represents an entire character, part of a character, or even, in a modal encoding, a shift operator or

part of a control sequence; in this last case, the byte in question is an artifact of the encoding rather than a constituent of the text.

(ii) it will not be able to determine with certainty the location within the text of elements such as spaces, line breaks, punctuation, etc., and so will be unable to carry out many kinds of linguistically-based processing.

(iii) discovering the correct encoding of a text will not always assist in discovering its language, and vice-versa.

Naturally, these problems can be solved by an exhaustive search, but that is an unattractive option from the point of view of both run-time efficiency and maintenance: assembling the necessary data for all supported encodings is not a trivial undertaking.

It may be noted that several additional issues are strongly related to that of character encoding. These include conventions for transliteration and romanization, and the various ad hoc informal methods of representing a larger character set within a smaller: `e2` for *é*, `\"u` for *ü*, etc. These fall outside the definition of encoding adopted above, but they may nevertheless be amenable to treatment by the some of the methods described below.

2.3 Distinguishability of Encodings

Using the notation of equations 1 and 2, for any two encodings CM_1 and CM_2, any text in $C(CM_1 \cap CM_2)^+$ will map to identical byte sequences, and any byte sequence in $B(CM_1 \cap CM_2)^+$ will represent the same text, under both CM_1 and CM_2. This is not merely a theoretical concern. Although different encodings are normally thought of as distinct, they often overlap considerably, in the sense that their intersection is a non-empty set of $\langle C, [b_1, \ldots, b_l] \rangle$ pairs. To the extent that two such encodings are applicable to the same language, this imposes inherent limits on their distinguishability.

Examples of encoding intersection abound. Consider the characters associated with bytes whose hexadecimal values are between `0x20` (space) and `0x7E` (tilde) inclusively in the ASCII encoding. These pairs also occur in the ISO 8859 family, CP850, CP1252, and UTF-8, among others. Consequently, it is impossible in principle to decide whether the intended encoding of an English text containing only these characters is ASCII, ISO 8859-1, CP850, CP1252, or UTF-8.

Further, ISO 8859-1 is, as far as its non-control characters are concerned, a proper subset of CP1252, the differences being confined to bytes in the range `0x80`–`0x9F`, control characters in the ISO standard but used for punctuation and accented characters in CP1252. A text containing none of the distinctive characters in question cannot be classified unambiguously. It is important to realize that this remains true even when the text has been produced by an application claiming to generate the CP1252 encoding. Nor is the appearance of the printed text a reliable guide. Visually, the two encodings in question differ largely in the presence in CP1252 of a wider range of punctuation symbols: opening and closing 'curved' double quotation marks, dashes of different lengths, and so on. It might be thought that these could serve as distinctive markers. However, many fonts provide close visual equivalents of these

characters within the intersection of the two encodings: pairs of apostrophes or backquotes resemble the quotation marks, sequences of hyphens may merge to produce dashes, etc., and it is by no means uncommon for users to make these substitutions. As a result, even a text whose printed form strongly suggests that its encoding is CP1252 may remain unclassifiable.

A slightly more subtle manifestation of encoding intersection occurs when distinctions between encodings designed for a family of languages are, in linguistic jargon, 'neutralized' in one member of the family. For most of the languages for which they are intended, ISO 8859-2 and CP1250 are quite easily distinguishable; at least some of the accented characters occurring frequently in Polish, Czech, Slovakian, etc. texts are mapped to different byte codes. In the case of Romanian, however, this is not the case, and in order to distinguish between the two encodings as applied to Romanian one must rely on the punctuation characters referred to in the preceding paragraph. Here, two encodings that pose no problems in most of their languages become much more similar when applied to another.

Since it is so often impossible to identify the precise *intended* encoding, a better approach is to select an *appropriate* encoding. It is usually reasonable to envisage a scenario in which a text such as the current paragraph, which can be written using only characters in the ASCII range, is to be subjected to further processing (display, printing, indexing, linguistic analysis, etc.) by some other program. In this case, identifying it as ISO 8859-1, CP850, CP1252 or UTF-8 rather than ASCII will have no practical consequences, since any program capable of treating e.g. CP1252 text correctly will also be able to handle ISO 8859-1 and ASCII text (the reverse is not generally true, of course). A sensible strategy would then be to make the most 'inclusive' identification possible.

It is important to note that accurate evaluation of LEI systems needs to take these factors into account. Perhaps the only realistic application where it would matter that a text should be identified as having the ISO 8859-1 encoding rather than CP1252 would be a usage survey on the lines of those occasionally conducted in order to estimate the relative importance of different languages on the world-wide web; as far as we are aware, no such survey has ever been performed. To be a useful indicator of system performance, therefore, evaluation must be based on the content of test data rather than its nominal encoding.

2.4 Distinguishability of Languages

Like encodings, certain pairs of languages overlap in ways that can make it difficult to distinguish them.

One well-known example is Norwegian: this name informally designates at least two widely used and relatively distinct varieties, Bokmål and Nynorsk, the former of which resembles Danish. Norwegian language policy lies outside the scope of this article, but some of its consequences have implications for LEI, especially of the data-driven sort. Government documents and public radio and television broadcasts appear in both, and in some cases 25% of the language produced at various institutions should be Nynorsk. Most of the differences are in the inflections and pre/suffixes and also in the selection of words. Some systems use a predefined list of frequent words and select the variant having the highest frequency in the text under study.

Serbian and Croatian are similar languages when spoken but the former is usually written with the latin alphabet while the former is written in cyrillic. Malay and Indonesian are also very similar but their denomination often depends on where the text was published or written.

Unfortunately, the issue of language identification raises emotions that are (normally) absent from discussions of encoding. For example, Dutch spoken in the Netherlands and Flemish spoken in Belgium which are the same language when written and spoken but they are nevertheless often differentiated. Moreover, as [Langer, 2002] observes, it is precisely in cases of very similar languages that feelings tend to run particularly high.

This distinguishability issue implies that training data must be carefully selected and the results interpreted with caution by users who often judge the output from the computer on the basis of their own world knowledge.

3 Language and Encoding Identification

The range of approaches to LEI can be viewed from the perspective of two questions:

(i) What are the cues that will be exploited in identification?

(ii) How are those cues exploited?

Answers to these questions strongly influence the following properties of a LEI system:

Generality: what assumptions are built into the system relating to the nature of the input — does it handle unrestricted languages and encodings, or just some predefined subset?

Extensibility: related to the above, what is involved in adding a new language-encoding pair to the system?

Efficiency: the amount of space occupied by the profiles, how much time is required to process an input, and how these quantities depend on the number of profiles and the size of the input;

Accuracy: the performance of the system, including robustness when presented with noisy, insufficient, or out-of-domain input.

When presented with the LEI problem, the first solution proposed by most laymen, and not a few language technology practitioners, is some variant of the 'manually derived distinctive feature' approach. Here, each of the languages to be recognized (different encodings rarely figure in discussions of this method) is associated with a number of items (words, single characters or sequences of characters) whose presence in a text has been determined, either by intuition or by visual inspection, to be good evidence that it is written in that language. Further, frequent items such as function words are normally preferred. For example,

a text containing instances of *the*, *does* and *and* is almost certain to be English, while *les*, *dans* and *et* are excellent indicators of French.

This is intuitively attractive, largely because it corresponds to the way we tend to recognize text ourselves: French text is French precisely because it is composed of French words. Indeed, tables encoding such features (generally characters) have been created to assist translators and librarians in manually determining the language of a text [Ingle, 1976]. For automatic processing, however, these methods have a number of disadvantages, as noted by Dunning [Dunning, 1994]:

- short inputs may contain no instance of a cue item;
- it is surprisingly difficult to produce cues which will unambiguously indicate the correct language (if only because of the possible presence of foreign place-names and other words);
- the non-word strings proposed as language cues cover only a very small proportion of possible inputs, and so provide only weak evidence;
- moreover, that evidence is categorial in nature and does not support weighting for comparison of hypotheses.

The importance of the last point is that any input of substantial size is likely to contain conflicting cues, leaving a choice to be made between the corresponding languages.

A further drawback of this class of solutions lies in its combinatorial nature; each time a language or encoding is added to the repertoire recognized, the potential for interference means that the distinctiveness of all existing features is called into question and must therefore be reevaluated.

Using n-grams statistics for identifying languages has been used for many years by cryptographers for separating random strings from what looked like natural language strings in automatically decoded messages. Kahn [Kahn, 1967] even traces back to the 14th century the idea of using frequencies of occurrences of sequence of two, three, four, five or more letters to characterize and distinguish between natural languages. [Cavnar and Trenkle, 1994] is recent survey on the use of n-grams for different tasks of text categorization: they give the results of extensive experimentation on these tasks, one of them being language identification.

A second consideration is that any linguistically-informed approach to LEI relies on knowledge of encoding to perform the basic operations of tokenization and segmentation on which more advanced processing typically depends. This is because these operations demand the ability to distinguish punctuation and white-space (space, tab, newline, etc.) characters from those occurring within words and other tokens. In practical terms, information allowing this must be available for all encodings known to the system.

Beesley [Beesley, 1988] was one of the first to explain in detail how to implement and use a language identifier in the context of NLP especially a Machine Translation environment. This algorithm seems to have been reinvented many times, Damashek [Damashek, 1995] is one of the most cited work in the area following this approach. [Poutsma, 2002] gives an

excellent overview of the different techniques both linguistic and statistical for identifying languages; he then shows an interesting approach based on Monte Carlo sampling. Dunning [Dunning, 1994] explains in details the mathematical machinery behind identification of languages. None of these works directly deal with the notion of detection of coding which seems to be taken for granted. Kikui [Kikui, 1996] addresses this problem for Asian languages but takes a special purpose approach based on the detection of special bytes that indicates escape codes that can appear only in certain encodings.

Language identification can also be related with the output of the Unix file program which determines the type of file (binary, dictionnary, C or C++source file, etc.) of its arguments. file uses all kinds of special tricks (magic numbers, choice of extension in the file name, detection of distinctive bytes for many types of files); SILC for the moment, treats all files uniformly according to trigrams of bytes values encountered during model building.

4 The SILC System

We now describe the approach to LEI we have chosen in SILC (Système d'Identification de Langues et de Codages)[3], the LEI system we have developed during the last few years and which is now used in a number of commercial applications.

The central task performed by SILC is classification: it places a text T, with unknown language and character encoding, in a category whose members it most closely resembles, and whose language and encoding are known. The language and encoding of T are then taken to be those of that category. More concretely, SILC adopts a probabilistic approach to classification. Each category is represented by a model, a mathematical structure whose role is to assign a probability to any text which is submitted to it. The correct model for a text is then the one that assigns it the highest probability. SILC is based on the classic Noisy Channel statistical approach [Manning and Schütze, 1999, section 2.2.4] employing the approach of Bayesian decision theory [Manning and Schütze, 1999, section 2.1.10].

4.1 SILC Architecture

SILC consists of three components: the matching component, named apply, evaluates the text in order to make the decision, while the data used as a basis for comparison are produced by a pair of programs, train and merge. In addition to their status as self-contained programs, apply, train and merge also exist in the form of application programming interfaces (API). Figure 2 shows the architecture of the system,

The organization illustrated in Figure 2 makes the addition of new models a relatively simple matter. Starting with the file containing the training corpus, a unitary model is produced using train, and is saved in a `.mod` file. This model file is then combined with existing unitary models to produce a new grouped model file.

[3]Language and Encoding Identification System, also commercialized as ¿QUE? by Alis Technologies

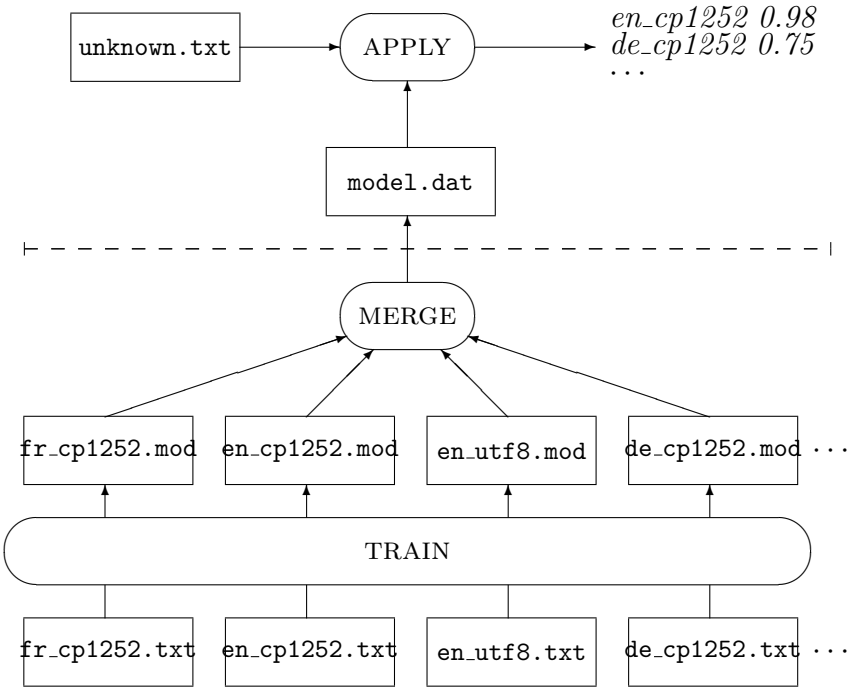

Figure 2: SILC system architecture. Individual model files `*.mod` are generated by train from text files `*.txt`, then combined into a single grouped model file `model.dat` by merge. This file is then used by apply to determine the language and encoding (here English and cp1252) of the content of `unknown.txt`.

SILC is delivered with a set of prepared models: we currently supply unitary models for 31 languages each in 3 or 4 encodings. Using figures of March 2003 from Global Reach[4], we estimate that the languages covered by SILC are the ones used by 632 million people with internet access and that they represent about 99.6% of the total online population.

The intended usage scenario is for the apply program (or its library equivalents) to be used together with a single, stable model file to evaluate large numbers of texts; if the models supplied with the system are insufficient, or if the developer wishes to experiment, alternate or replacement model files can be created using train and merge. Typically, therefore, apply will be called many times for each model training and merging operation; most end users need not be aware of the model building process.

4.2 Probabilities

Let $\mathcal{M}$ be a set of models, each of which represents a distinct language-encoding pair, and T be a text to be classified. The task is to find the member of $\mathcal{M}$ which provides the best description of T; in the SILC framework, this will be the most probable model, given T:

$$\text{argmax}_{M \in \mathcal{M}} P(M \,|\, T).$$

We exploit the Bayesian equivalence

$$P(M \,|\, T) = \frac{P(T \,|\, M) P(M)}{P(T)},$$

and since T is constant for all $M \in \mathcal{M}$, ignore it:

$$P(M \,|\, T) = P(T \,|\, M) P(M).$$

We also ignore the prior model probability $P(M)$. In effect, we are making the false assumption that the true distribution of models is uniform, i.e. that all language-encoding pairs are equally likely. In realistic applications this is clearly not the case, see [Poutsma, 2002] for an interesting discussion about this issue. Rather than attempting to estimate $P(M)$ empirically (note that the distribution is likely to change drastically over time), we employ the approximation

$$P(M \,|\, T) \approx P(T \,|\, M).$$

For $T = [b_1, \ldots, b_n]$,

$$P(T \,|\, M) = \prod_{i=1}^{n} P(b_i \,|\, M),$$

in which $P(b_i \,|\, M)$ is estimated via an interpolated trigram/unigram model

$$P(b_i \,|\, M) = \lambda P_3(b_i \,|\, b_{i-2}, b_{i-1}) + (1 - \lambda) P_1(b_i),$$

the parameter λ being set empirically, in such a way that most weight is assigned to the trigram.

[4] `http://www.glreach.com/globstats/index.php3`

4.3 Confidence

For some text T, each model $M = \langle L, E \rangle$ is assigned $P(M|T)$, i.e. the probability that L and E are the actual language and encoding of T. The validity of a decision based on this probability depends on three assumptions:

(i) All $\langle L, E \rangle$ pairs known to the system are *a priori* equally likely;

(ii) The models are statistically independent;

(iii) The true language and encoding of T are represented by one of the models known to the system.

In practice, none of these assumptions is entirely valid:

(i) Most real applications are more likely to receive texts in some $\langle L, E \rangle$ pairs than others;

(ii) Models are not statistically independent, since it is frequently the case that several encodings for a given language overlap;

(iii) There is no way of ensuring that the range of models known to the system exhaustively covers all input possibilities.

Little can be done about (i) and (iii), but (ii) can be partially compensated for. The problem is that different encodings for a language are often related, and so the probability of a model for some language depends on the number of encodings it is associated with.

One way of obtaining a degree of certainty which is independent of the number of encodings per language is to find for each language L the most probable model $\langle L, E \rangle$. The degree of certainty c is then the ratio of the best score achieved for L to the sum of the best scores for all languages:

$$c = \frac{\text{argmax}_L P(\langle L, E \rangle)}{\sum_{l \in \mathcal{L}} \text{argmax}_l P(\langle l, E \rangle)}.$$

A sound decision is then based not only on the probability assigned to a text by a model, but also on the value of the confidence ratio for that model.

An additional measure of confidence can be obtained by taking the log-likelihood ratio between best and second-best models M_1 and M_2:

$$c = \log \frac{P(M_1)}{P(M_2)}$$

An alternative would be the use of confidence intervals as suggested in [Elworthy, 1998] in which the probabilities for each language are gradually computed as the input is processed. The algorithm terminates when enough evidence has been gathered to make a clear decision. In this approach, the model for each language contains not only the probability of each unit but also the upper and lower limits of a range containing this probability for a specific level of confidence.

4.4 Training Text Requirements

SILC makes its decision based on what the system has learned to recognize during its training. The data used to train a model for a language L and an encoding E must be large enough for a high proportion of the characters and character sequences of L to be assigned a probability that reflects their true behaviour in texts of that language and encoding. Training data must therefore be *representative*: for example, text consisting largely of personal names and telephone numbers will produce a model with high accuracy when applied to telephone directories but lower on more general documents. *Coverage* must also be adequate; positive decisions are based largely on trigrams observed in the training text. Any trigram in the input which is not present in a model, either because it was absent from the training text or because it was removed due to low frequency in the model truncation process, greatly lowers the probability of that model. If too little training text is used, many useful trigrams will fail to reach the threshold required for them to be retained, and the model will assign too low a probability to input texts it is intended to identify.

For these reasons, it is important to choose adequate training data for each model. The models included in the standard SILC distribution were built from texts ranging in size from less than 1Mb to over 8Mb; the largest texts tend to be those for Chinese and Japanese, which have a very large character inventory, and for the less compact encodings such as UTF-8. In our case, we built one corpus for each language in one encoding and then used the well known `recode` program [Pinard, 2002] to transform this corpus into different encodings.

As the identification depends on statistics computed from the training corpus, it is very important that identification be made on similar corpora. In our current distribution, we have built model with *text only* corpora, so identification is not reliable (often false) on different types of files (HTML, pdf, MS-Word documents, program source files, binaries, etc) which have to be transformed to plain files before being processed by SILC. For example, at one time, SILC was applied to C++ files with english comments but they had been identified as being written in malaysian,

4.5 Model Compression

Two principal factors contribute to the size of a SILC model: the number of trigram probabilities and the number of models.

Since each of the three bytes represents 256 distinct possibilities, an exhaustive model would need to record 256^3 trigrams, which, given probabilities expressed as four-byte floats, would produce unitary models over 65 Mb in size, even without the necessary structures for storage and indexing of trigrams.

The first and simplest means of reducing the size of a model is to restrict it to the n most frequent trigrams, where n is supplied as a parameter to the model-building process. As, in practice, we are only interested in the relative rank of the models for a given text, we can discretize the probabilities into 256 distinct values which are stored in single bytes. More details on the impact of these compressions are reported later in this paper in section 5 dealing with the evaluation.

As we have shown in section 2.4, many encodings share a part of their byte values for the same abstract character, so certain trigrams receive the same probability in more than one model but they can be represented just once. The process allowing this is known as factorization, and it is conducted under the control of a specification file read when the models are merged. For example, if $\{CM_1, CM_2, CM_3\}$ are to be factorized. Any trigram assigned identical probabilities by all of CM_1, CM_2 and CM_3 is removed and added to a new model created automatically for this purpose. When the global probabilities of CM_1, CM_2 and CM_3 are calculated, these extracted probabilities are redistributed correctly. Currently, we have used factorization only for encodings within the same language but this process could be applied to any combination of encodings sharing a part of their byte sequences. Factorization does not incur any loss of precision in the detection process.

5 Evaluation

5.1 Impact of parameters on precision

We carried out a number of experiments in order to measure the identification precision of SILC on a random sample of texts in different languages and encodings. Precision is measured by counting the number of times SILC has returned the correct language and encoding as its first choice. Tables 1 to 3 give the mean precision on 10 randomly selected extracts. As discussed in section 2.3, this measure is a lower bound because often different encodings of the same language cannot always be distinguished (e.g. English text encoding bytes are the same for both CP1252 and UTF-8 if only ASCII characters are used). Given the number of tests we carried out, we used an automatic scoring which only counted if the first answer returned by SILC matched exactly the one of the original text and not another possible one.

Table 1 gives the precision as a function of the number of models to choose from. As expected, precision increases with the length of the texts to analyse and decreases with the number of models to choose from. We see that for all models, we need about 200 characters in order to get reliable enough results (better than 90%). Precision is almost perfect with 5 or less models even with a text of as less as 10 characters.

Table 2 uses the same set-up as of Table 1 but for only distinguishing between languages given that the encoding is known as it occurs in many cases when he user uses the same computer system for entering the text and for identifying the language. Precision is then much better and almost perfect with as less as 50 characters.

We also investigated the influence of the number of the most frequent trigrams kept in the trained model for tallying the frequencies of occurrences as described in section 4.5. This is has a strong influence on the size of the grouped model (shown in the last column). Table 3 shows that about 3000 trigrams are needed in order to get a reliable enough precision.

nb models	10 ch	50 ch	100 ch	200 ch	500 ch	1000 ch	5000 ch
2	90 %	100 %	100 %	100 %	100 %	100 %	100 %
5	100 %	100 %	100 %	100 %	100 %	100 %	100 %
10	87 %	98 %	99 %	99 %	99 %	99 %	99 %
20	94 %	94 %	97 %	97 %	100 %	100 %	100 %
50	59 %	83 %	89 %	89 %	97 %	97 %	97 %
85	54 %	79 %	85 %	92 %	93 %	93 %	96 %
mean	81 %	92 %	95 %	96 %	98 %	98 %	99 %

Table 1: Precision of SILC on texts of different number of characters (colums) given a random sample of number of models to choose from (rows). For example, SILC has a precision of 94 % when it has to choose between 20 models for a text of 10 chars.

nb langs	10 ch	50 ch	100 ch	200 ch	500 ch	1000 ch	5000 ch
2	100 %	100 %	100 %	100 %	100 %	100 %	100 %
5	80 %	98 %	98 %	100 %	100 %	100 %	100 %
10	74 %	97 %	98 %	99 %	99 %	99 %	99 %
15	75 %	97 %	98 %	99 %	99 %	99 %	99 %
20	73 %	95 %	97 %	97 %	98 %	98 %	99 %
27	72 %	95 %	98 %	98 %	98 %	98 %	99 %
mean	79 %	97 %	98 %	99 %	99 %	99 %	99 %

Table 2: Precision of SILC on texts of different number of characters (colums) given a number of languages to choose from (rows). The languages were chosen by taking the most used languages on the web as given in the table available at `http://www.glreach.com/globstats/index.php3`. All texts were coded in UTF-8. For example, SILC has a precision of 97 % when it has to choose between 15 languages for a text of 50 chars.

nb tricar	10 ch	50 ch	100 ch	200 ch	500 ch	1000 ch	5000 ch	Model size
1	21 %	28 %	30 %	31 %	37 %	36 %	38 %	60,8 K
5	26 %	39 %	44 %	49 %	52 %	53 %	57 %	64,1 K
10	28 %	45 %	51 %	58 %	62 %	63 %	67 %	68,1 K
15	30 %	49 %	53 %	62 %	65 %	66 %	69 %	72,1 K
20	32 %	52 %	57 %	65 %	69 %	70 %	70 %	76,1 K
25	32 %	53 %	60 %	66 %	71 %	74 %	74 %	80,1 K
50	36 %	61 %	69 %	76 %	81 %	81 %	80 %	99,9 K
100	40 %	68 %	75 %	82 %	85 %	86 %	86 %	138,8 K
300	46 %	73 %	81 %	89 %	93 %	94 %	95 %	290,9 K
500	46 %	76 %	86 %	90 %	94 %	94 %	95 %	441,6 K
1000	51 %	82 %	88 %	92 %	93 %	94 %	95 %	817,3 K
3000	57 %	84 %	89 %	93 %	93 %	94 %	95 %	2200,0 K
5000	59 %	85 %	90 %	94 %	94 %	93 %	95 %	3700,0 K

Table 3: Mean precision of SILC for 850 texts (10 extracts for all language and encodings) of different number of characters (columns).The first number of each row gives the number of trigrams kept and the last number gives the size of the resulting grouped model for 85 models.

5.2 Comparison with other systems

We have compared SILC with other public domain language identifiers whose source code was available: TextCat, Lingdet_suite and Stochastic Language Identifier; we also compared with two demonstration systems available on the web Xerox Research Centre Europe and Natural Language Identification Tool. Tests were made for the 30 languages identified by SILC and latin for which SILC was not trained. One sentence by language encoded many times gave a sample of 100 sentences. These sentences were chosen randomly from a corpus outside of the original training and testing sets of SILC or of other language identifiers. Table 4 shows the results we obtained. SILC is by far the fastest system and one of the most precise taking into account that it can also identify the encoding.

6 Future work

We see four areas where modifications to the current system would be worth pursuing:

(i) treatment of mixed-language input;

(ii) investigation of alternative confidence measures;

(iii) mechanisms addressing the model intersection problem;

(iv) extension of coverage to non-text input.

System	langs	tested	prec	CPU	url
XRCE	47	52	90%	web	http://www.xrce.xerox.com/competencies/content-analysis/tools/guesser
NL Ident Tool	4	10	100%	web	http://users.info.unicaen.fr/~giguet/diagnostic.html
Stochastic LID	13	38	100%	2,5 min	http://www.dougb.com/ident.html
Lingdet_suite	2	4	100%	2 min	ftp://crl.nmsu.edu/pub/misc/lingdet_suite.tar.gz
Lexitech	260	21	18	86%	http://www.languageidentifier.com/
Textcat	70	62	77%	0,65 sec	http://odur.let.rug.nl/~vannoord/TextCat/
SILC	30	65	100%	0,01sec	http://www-rali.iro.umontreal.ca/ProjetSILC.en.html

Table 4: Comparison of some language identifiers: following each system name, we give the number of languages it recognizes, the number of test sentences we used, the precision in the identification of the test sentences, the CPU time it took on a 300MHz Linux machine and the URL where we found the system.

As normally used, SILC provides a global analysis of the entire input. This makes it difficult to manage input containing two or more languages. Indeed, results on such input can be counterintuitive: it is quite possible for a text containing roughly equal quantities of French and Spanish to be classified as Catalan, for example. Here, the presence of large amounts of each language has reduced the probability of the other sufficiently to allow a third to emerge as the favourite. The SILC API contains facilities for assigning $P(t_i, \ldots, t_j \,|\, M)$ to arbitrary subsequences $t_i, \ldots, t_j$ of T, and this may well be of use in solving the mixed-language problem. Nevertheless, it is arguable that the onus for providing a solution should be on the suppliers of SILC rather than its users. One plausible assumption that could be exploited in this connection is that while a given input may be composed of more than one language, it is unlikely to involve multiple encodings[5]. We have done some experiments with a recursive separation approach in which parts of a text are analyzed: based on the comparative scores for the languages, SILC judges if a part of a text contains more than one language; in this case, its separates the text in half and reapplies this procedure to each part.

There is growing interest in the application of techniques for estimating confidence which have been developed in the speech processing community to other areas of language technology.

The issues concerning distinguishability outlined in section 2.3 have obvious implications for the interpretation of system results. Although the present behaviour is fully justifiable on practical grounds, it may not be reasonable to expect users of SILC to appreciate why a text they correctly believe to be in ASCII is not in fact recognized as such. Fortunately,

[5]Of course, this may not hold for applications where multiple texts are received through an undifferentiated input stream.

there seem to be a variety of table-based approaches that would permit a finer-grained set of distinctions to be made, without a significant run-time penalty.

In this paper, we have shown that SILC works well on plain text files; we are currently extending its coverage to many other types of files (PDF, MS-word, HTML, binaries, etc.) without the need for preprocessing in other to extract the text. As discussed in section 3, as SILC and the Unix `file` program share a common goal, we have integrated SILC function calls within a version of `file` but we have not yet evaluated its performance. We also have developed a Java version of SILC in which we can dynamically add or remove language models to a grouped model before the identification process.

Quite separate from these issues is that of maintaining the model inventory of SILC. Even though the current standard distribution covers 31 languages, this is small in comparison with the total number of written languages in the world; at some stage many of these will figure in a significant proportion of potential SILC inputs. Just as coverage may need to be extended to accommodate new languages, so new encodings will need to be added, in particular those Unicode encodings not already handled. And, to a far greater extent than languages, encodings fall out of use, and will therefore be removed from the inventory.

7 Conclusion

This paper has described SILC a state of the art implementation of a language detection engine. SILC is fast and accurate on more than 30 languages each having 3 or 4 encodings. Although this problem does not seem to be *high flying NLP* and that it is considered by many to be a solved one, we have shown that it involves many subtle aspects that must be dealt with in a practical system especially when it has to be combined with other NLP modules.

Acknowledgments

SILC has been developed of the years by many members of RALI especially: Pierre Isabelle who saw the potential of this technology, many years ago and decided to *push* it. We also thank George Foster and Elliott Macklovitch who helped develop and market the system. SILC benefited from the contribution of Véronique Bouffard, Guillaume Benny, Ann-Sophie Pépin and Ngoc Nguyen Tran who developed and tested many alternatives. Christian Descoteaux and François Yergeau from Alis Technologies also suggested many improvements and prompted us to go further.

References

[Beesley, 1988] Beesley, K. (1988). Language identifier: A computer program for automatic natural-language identification of on-line text. In *Languages at Crossroads: Proceedings of the 29th Annual Conference of the American Translators Association*, pages 47–54.

[Cavnar and Trenkle, 1994] Cavnar, W. B. and Trenkle, J. M. (1994). N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, US.

[Damashek, 1995] Damashek, M. (1995). Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848.

[Dunning, 1994] Dunning, T. (1994). Statistical identification of language. Technical Memo MCCS 94-273, Computing Research Laboratory, New Mexico State University, Las Cruces.

[Dürst et al., 2002] Dürst, M. J., Yergeau, F., Ishida, R., Wolf, M., Freytag, A., and Texin, T. (2002). Character Model for the World Wide Web 1.0.

[Elliott et al., 2002] Elliott, J., Atwell, E., and Whyte, B. (2002). Language identification in unknown signals. In *Proceedings of COLING'2000, 18th International Conference on Computational Linguistics*, pages 1021–1026.

[Elworthy, 1998] Elworthy, D. (1998). Language identification with confidence limits. In *Proceedings of Sixth Workshop on Very Large Scale Corpora*, pages 94–101, Montreal. COLING-ACL'98.

[Ingle, 1976] Ingle, N. C. (1976). A language identification table. *The Incorporated linguist*, 15(4):98–101.

[Kahn, 1967] Kahn, D. (1967). *The Codebreakers*. Macmillan.

[Kikui, 1996] Kikui, G. (1996). Identifying the coding system and language for on-line documents on the internet. In *Proceedings of COLING'96*, pages 652–657, Copenhagen.

[Kilgariff, 1996] Kilgariff, A. (1996). Which words are particularly characteristic of a text? a survey of statistical approaches. In *Proceedings of the AISB Workshop on Language Engineering for Document Analysis and Recognition*, pages 33–40, Sussex.

[Langer, 2002] Langer, S. (2002). Grenzen der Sprachenidentifizierung. In *Proceedings of Konvens 2002: 6. Konferenz zur Verarbeitung natürlicher Sprache*, pages 96–106, Saarbrücken.

[Lunde, 1999] Lunde, K. (1999). *CJKV Information Processing*. O'Reilly.

[Manning and Schütze, 1999] Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA.

[Pinard, 2002] Pinard, F. (2002). recode 3.5 available from `http://www.iro.umontreal.ca/~pinard/recode/HTML`.

[Poutsma, 2002] Poutsma, A. (2002). Applying Monte Carlo techniques to language identification. In Hondorp, H., Nijholt, A., and Theune, M., editors, *Computational Linguistics in the Netherlands 2001: Selected Papers from the Twelfth CLIN Meeting*, volume 45 of *Language and Computers*, pages 179–189. Rodopi, Amsterdam.

[Sibun and Spitz, 1994] Sibun, P. and Spitz, A. L. (1994). Language determination: Natural language processing from scanned document images. In *Proceedings of the Fourth Conference on Applied Natural Language Processing*, pages 15–21, Stuttgart.

[Whistler and Davis, 2000] Whistler, K. and Davis, M. (2000). Character encoding model. Unicode Technical Report 17.