Multilingual E-mail Text Processing for Speech Synthesis

Daniela Oria, Akos Vetek

Audio-Visual Systems Laboratory
Nokia Research Center, Helsinki, Finland
daniela.oria@nokia.com, akos.vetek@nokia.com

Abstract

An integrated method of text pre-processing and language identification is introduced to deal with the problem of mixed-language e-mail messages in a speech-enabled e-mail client. Our method can confidently distinguish between the supported languages and switch between several TTS engines or languages to read the portions of the text in the appropriate language. This is achieved by making use of the combined information from a text pre-processor and a language identifier that relies on both statistical information and linguistic features indicative of a particular language.

1. Introduction

One of the problems we encountered when designing our speech-enabled e-mail client ([1]) was the fact that a considerable portion of the e-mail traffic in our research laboratory consists of messages where the text is either written in languages other than English or different languages are mixed within the same message. This is often the case of Finnish e-mail messages, where the text is mainly in Finnish, but English words or whole sentences in English (e.g.: quoted from other messages) are inserted in the Finnish text. In such a situation, it is quite obvious that offering only an English text-to-speech (TTS) engine for reading out the e-mails would be a considerable limitation. Quite often users would end up having, for example, Finnish text read out using the English synthesizer, which would greatly hinder both the usability and the utility of the application. These concerns led to the introduction of an integrated text processing and language identification module, which is now one of the core features of our multilingual e-mail client.

When designing the language identification module of our system, one of the biggest challenges was finding an algorithm that would perform best keeping in mind the special nature of the text whose language was going to be identified. E-mail text is extremely irregular. It contains both very short sentences (such as salutations or signatures, where often languages are mixed, e.g. “Hi Jean-Francois” or “Best regards, Matti Karjalainen”) and longer sentences or paragraphs that resemble a more “conventional” kind of text. It may also contain lots of abbreviations (also “unconventional”, such as BR, BTW and the like), acronyms and jargon specific for a restricted community (e.g. a particular company) that might make language recognition difficult. As a consequence, we needed to find a solution that would perform well not only on long and regular text, but also on very short text chunks.

2. Text Pre-Processor Algorithm

The task of the pre-processor in our system is two-fold, and is tightly integrated with the subsequent language identification phase. First, it is used to identify the high level structure of the e-mail message, grouping its contents into larger blocks of text, called regions (header, plain text, quoted regions, etc), which is followed by identifying utterance level boundaries within each region (Figure 1). The subsequent language identification phase will combine the score of both the smaller and larger pieces of text to come up with the final decision as to what language to use for speaking the text (see Section 4).

The first phase of the analysis is loosely based on [2], whereby chunks of the input text are grouped together to form regions. A region is defined as a block of contiguous lines that are separated from other regions by: one or more blank lines; or some other unambiguous separator, like a forwarded message indicator (e.g. “>” or “.”). Each region is considered to be of one single type. As noted in [2], in some cases the detection of a region is straightforward (e.g. e-mail headers), while in other cases a more elaborate detection mechanism might be required. Therefore the detection of the region types is also based on the distribution of different character classes (e.g. whitespace, digits, alphabetic, “>”).

The procedure goes as follows: first each character in the input is classified to belong to one of the previously mentioned character classes, and the individual lines of each block resulting from this encoding process are matched in turn against the set of eight character class N-gram models, one for each text region type ([2]). The models were trained on a few dozen real-world e-mail messages using an N-gram based algorithm implemented after [3]. Matched against each line, the models give a value indicating how strongly the given line resembles the particular type the model was trained

---

1 Eight in total as in [2].
on. Finally, the unique classification of each line is chosen by picking the model with the best match, while the uniform classification of the block is achieved by selecting the model that was assigned to the most number of lines. Our experiments show an error rate of 9%.

The second part of the analysis follows that of utterance chunking described in the Festival Tutorial ([4]). Once the main regions of text are identified, the regions themselves get sub-divided into smaller chunks, roughly corresponding to utterances or sentences. These not only aid the quick and smooth synthesis of text but are also used by the language identifier to arrive at the local score (see Section 4).

Even though at a first glance one might presume that the assumption that a sentence starts with a capitalized word and ends with a full stop should work fine, as it turns out, especially in e-mail messages, these rules often simply do not hold (e.g. confusing use of periods, abbreviations). In order to deal with these, our system employs a hand-crafted decision tree to determine what signifies a sentence break. The rules, amongst other things, take into account the presence of blank lines, certain punctuations based on the current input token, its immediate surroundings, and a minimum length criterion (arrived at in our language identification experiments). Even though the system performs quite well, some unseen abbreviations, invalid capitalization or spacing might still occasionally pose problems.

3. Language Identification Algorithms

3.1. Background

Our approach in performing language identification follows some of the conclusions presented in [5] and [6]. [5] suggests a method for identifying the language of sentences that is based only on the characteristic features of languages, without having any dependence on training sets. The only clues used for guessing the language of a text are grammatical words and the letters of the alphabet. One limitation of this approach is that it works well on sentences that are longer than 8 words, but it is very weak on shorter sentences, which do not contain any (or few) grammatical words and therefore provide insufficient clues for the language identifier. Alphabets also do not seem suitable for performing language identification on short sentences.

[6] is based on the assumption that frequent phenomena that can be identified as specific of a particular language differ according to the length of the text whose language has to be identified. The proposal is to combine linguistic knowledge (grammatical words and alphabet) and statistical knowledge (N-gram models based on frequent endings of words), as they are able to act as language identifiers on different levels of a sentence. Their combinations should therefore be able to classify every kind of input.

In our approach we also combine statistical and linguistic information. The statistical knowledge is given by N-gram models and is used as the main source of information for performing language identification. When the statistical information seems to be insufficient for making a reliable decision, we enhance it with the use of linguistic knowledge. We decided to use only the letters of the alphabet as linguistic clues, as we believe that building reliable grammatical word lists for a considerable number of languages can be a relatively long process that might also imply quite some knowledge of the languages in question. Another reason for dropping the use of grammatical words is that we wanted to keep the whole process as automatic as possible, involving no or very little human knowledge and manual checking (the manual work required for identifying the letters of the alphabet specific to certain languages can still be considered very minimal).

The main difference between our approach and that of [6] lies in the structure of the decision tree employed in the algorithm. The decision tree on which the algorithm presented in [6] is based utilizes the alphabet as first source of information for performing language identification and is as follows:

1. Call the Alphabet method to check if the word belongs to the language
2. If so,
   a. Call the Grammatical Word method to check if the word is a grammatical word.
   b. If not, Call the Frequent Ending method to check if the word morphology indicates that it belongs to that specific language.

Since we have to identify the language of text chunks of different length, we instead base our algorithm on the assumption that the statistical information is most reliable in the case of short input text, and that this information can be reinforced by alphabetical knowledge in the case of long text. This is the decision tree we base our algorithm upon:

1. Perform language identification using the statistical language identifier and obtain an n-best list
2. If this result is not confident enough (i.e., the difference between 1st and 2nd best falls below a specific threshold), use the alphabet to check to which language the text chunk belongs
   a. If the alphabet provides useful clues, rescore the n-best list
   b. If the alphabet provides no clues (i.e. the text chunk does not contain any of the special characters that can be used to discriminate between languages), leave the n-best list unchanged and rely only on statistical information.

3.2. Implementation of the language identification algorithms

As we already mentioned, our language identification module is a statistical, N-gram based module. We trained N-gram models for 10 languages (Danish, Dutch, English, Finnish, French, German, Italian, Portuguese, Spanish and Swedish). Since our target was to have models that would be as balanced as possible across languages, we collected a training

---

2 According to [5], grammatical words can be used to discriminate languages because they are a) proper to each language and in a whole different from one language to another, b) short, and c) not numerous, so it is easy to build exhaustive lists.

3 We use N-gram models built on whole words (we also take word boundaries into consideration) rather than only word endings, as suggested in [6].
corpus that contains the very same material translated in all 10 languages).

In order to select the most appropriate model to be used in our language identification module, we ran several tests using different N-gram sizes (N=3, 4, 5) and different training set sizes (1000, 2000, 5000, 10000, 50000, 75000 and 100000 bytes). We then tested the resulting models on different sizes of input text (10, 20, 50, 100, 200 and 500 bytes). We also ran the same experiments with two other algorithms.

The first algorithm is partly based on the approach suggested in [6], and it uses statistical information based only on the last three characters of each word (but no linguistic information).

The second algorithm follows the proposal of [3] and is based on calculating and comparing profiles of N-gram frequencies. In this approach the language identifier first computes profiles of training set data, i.e. it creates a list of common character sequences (of N length) derived from the training data, sorted into reverse order by the number of occurrences. When performing language identification, the system computes a profile of the text that is to be classified and it then computes a distance measure between the text’s profile and each of the category profiles it has at disposal. The smallest distance measure from all the category (i.e. language) profiles to the text profile indicates the category to which that specific text belongs.

The target of our tests was to find the most suitable model (with respect to both the employed algorithm and the size of the training set) that would perform reasonably well on both short and long input texts. Some of the results of our experiments are shown in Table 1:

<table>
<thead>
<tr>
<th>Testing size</th>
<th>3gram</th>
<th>4gram</th>
<th>5gram</th>
<th>Profiles of n-gram freq.</th>
<th>Last 3 char</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 b</td>
<td>3.56</td>
<td>1.52</td>
<td>3.0</td>
<td>2.36</td>
<td>4.01</td>
</tr>
<tr>
<td>200 b</td>
<td>3.04</td>
<td>0.18</td>
<td>0.24</td>
<td>0.14</td>
<td>0.55</td>
</tr>
</tbody>
</table>

Table 1: Language identification error rates on 50 and 200-byte input, training size 75000 bytes

From the results in Table 1 we can deduce that the 5-gram models we employed were likely undertrained. Adding new training data would probably have boosted performance even further, so that the 5-gram models would have most likely outperformed the 4-gram ones. On the other hand, this would have meant putting a considerable amount of extra effort in collecting new training material. Since the results we obtained with the 4-gram models and with other algorithms were already very satisfactory, the minimal increase in performance would not have justified the extra effort needed.

Taking into consideration the results of all the tests across all input sizes, we came to the conclusion that the best model for performing language identification in our application specific domain is a 4-gram model trained on a 75000 bytes corpus.

In order to check how our baseline algorithm performed in comparison to other language identification methods, we compared its performance with that of the algorithm presented in [7]. For this purpose we trained and tested our algorithm with the same testing and training corpora that Dunning used in his research. The results we obtained were in line with those presented by Dunning in [7].

4 Our corpus is made up of texts taken from http://europa.eu.int

4 Integrated Text-Processing and Language Identification Module

The algorithm we are using consists of 3 steps. First all competing language models are uploaded and language identification is performed both on what has been identified as a “sentence” and on what has been classified as a “region” (or paragraph) by the text processing module (see Section 2). We therefore get two language identification scores:

- a “local” language identification score, which refers to the language of the shorter text chunk, and
- a “global” language identification score, which identifies the language of the bigger text region (see Figure 1).

If the scores of the first and second best candidates (both in the case of local and in the case of global language identification) are too close, we perform a second pass using linguistic constraints. If the incoming text string contains any of the special characters which are specific to the language that has been identified (we use only the first and second best candidates), we put more weight on that language. If no special character is found, then the results remain the same.

After getting the results of the second pass, a further check is still performed. We assume that paragraphs are usually written in (or largely dominated by) one single language. Therefore, in order to avoid too frequent language switching and increase the language identification accuracy, the “local” score is influenced by the “global” score. After getting the results of the first (and eventually the second) pass, we check whether the first best candidates of both global and local scores are the same. If the results do not match, we perform a rescoring based on the global language identification score. The result gives the final decision for what is identified as that specific chunk’s language. The “global rescoring” is done as follows:

\[
S_f = \alpha S_l + (1-\alpha) S_g \quad (0 \leq \alpha \leq 1) \quad (1)
\]

where \(S_f\), \(S_l\) and \(S_g\) are the final score, local score and global score respectively, and \(\alpha\) is a weighting coefficient.

We tuned and tested our algorithm on a corpus of about 30 real-life mixed-language e-mail messages taken from various Internet newsgroups and obtained accurate results. The tests included experiments with several weighting coefficients, after which we came to the conclusion that \(\alpha=0.35\) provides the best performance for the algorithm we are proposing.

Some examples of how our algorithm works are provided in Figure 2 and Figure 3.

In the example shown in Figure 2, the language of the whole paragraph is identified as Swedish. The language of the utterance “Hejsan” is identified as English. Since the language of the utterance (local score) does not match with the language of the paragraph (global score), we perform the global rescoring. Hence, applying our formula for global rescoring, we obtain:

\[
Utt_conf = 0.001 \cdot 0.35 + 0.0174 \cdot (1-0.35) = 0.01166
\]
5. Conclusions

In this article we focused our attention on the treatment of e-mail messages containing texts from several languages. We have shown that making use of the combined information from a text pre-processor and a language identifier that relies on both statistical information and linguistic features indicative of a particular language we can confidently switch between several TTS engines to read the portions of text in different languages.

6. References