http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (4): 534-540, 2007
ISSN 1812-5638

Unicode Aided Language Identification across Multiple Scripts and Heterogeneous Data

Farheen Hanif, Fouzia Latif and M. Sikandar Hayat Khiyal
Department of Computer Science, International Islamic University, Sector H/10, Islamabad, Pakistan

Abstract: With growing explosion of multi-lingual data on the Internet and other informational and communicational fields, the requirement of having effective automated language identifiers has increased further. More information finds its way into the computer systems and the web and using manual methods to categorize the information is becoming increasingly in-feasible. In this study we discuss improvements we have achieved in existing language identification methods. Couple of new areas that were not explored before is the inclusion of non-Roman scripts and active usage of Unicode information about scripts to enhance the language detection process.

Key words: Language identification, unicode, multi-lingual documents, n-grams, internationalization, AI, language script

INTRODUCTION

The fundamental purpose of language identifiers is to indicate distinctions related to linguistic properties and specifically distinctions that are relevant for IT purposes. There are a wide variety of distinctions pertaining to several distinct linguistic parameters that have been suggested as potentially relevant for language identification: languages, language families, dialects, country variants, other regional-based variants, script variants, style variants and modality variants, time based variants, typographic variants, etc. Many different orthogonal parameters could be used in meta-data attributes and the potential combinations and permutations are daunting. In actual practice many of the potential distinctions are not needed for most realistic usage scenarios.

Application areas can probably be divided into two general types: cataloging and retrieval of content and resources for localization and language enabling of software.

Present research provides Unicode support and multi-script support (e.g., Roman, Chinese, Arabic). It also provides language detection for multilingual documents.

IDENTIFICATION TECHNIQUES

Dictionary based identification methods had been the most used methods in the early stages. In the 80's Cavnar and Trenkle presented their n-gram based algorithm as used by Kornai and Richards (2002) for language detection that solved many problems that persisted with previous language identification techniques.

In this research we have tried to improve on what already has been done. We have implemented improved variations of dictionary based and n-gram based algorithms.

Dictionary based identification: Dictionary Based Identification (DBI) is one of the most tried and implemented methods for language identification Kuhn (2004). Though simple, its effectiveness in certain areas cannot be denied. One of its biggest advantages is its performance. DBI is much faster during training and detection than other algorithms used in this research. DBI gives good results when comparing languages belonging to different scripts. If used wisely this technique becomes more useful than it seems at first glance. Joined with Unicode based inter-language document identification, to detect and extract multiple languages data chunks from a document; dictionary based identification is applied on these chunks individually.

Dictionary based method has been previously used but in different ways (Hedlund, 2002). Dictionary based method used in our work is more efficient as compared to previously implemented methods. That is as compared to Hedlund (2002) method with our method (DBI) is more efficient due to the following reasons:

- DBI uses N-grams (1 g) as compared to the compounds (Hedlund, 2002). One gram cover all the linguistic features.

Corresponding Author: M. Sikander Hayat Khiyal, Department of Computer Science, International Islamic University, Sector H/10, Islamabad, Pakistan

- DBI requires less memory
- DBI is not a lengthy process as compared to Hedlund (2002) method. In Hedlund (2002), the compounds are first split into components, normalized and then query structuring for compounds and components. In DBI n-grams are not normalized. They are generated and stored in the dictionary.
- DBI requires no knowledge about the language to be identified.

Like most techniques used in this research, dictionary based identification is also a two phase process. The first phase being training and the second is identification. Before going on to identifying documents, the system must first be trained for the languages that we wish to detect. Remember, it only detects the languages it has been trained for. However, if a language belongs to a script for which another language has already been trained is found in an input document, that language is identified as the language that was trained for the same script. For example, say in the Arabic script, we have trained DBI for Arabic, Urdu and Persian languages. If we try to detect a Pushto, Punjabi or similar document, all Arabic, Urdu and Persian will come up as close matches giving a hint to the script the language belongs to.

The drawback in dictionary based implementation is that it doesn't perform well when comparing between languages belonging to the same script. Also document containing too few characters give much less information to DBI to detect languages effectively. Also, in case of documents like web pages etc containing multiple languages, the results are not able to clearly distinguish the language. However, if one language occupies the majority portion of the document, it has higher match weight-age hinting its identification.

Algorithm:

Training Phase:

Inputs: 1) Training document
2) Dictionary file path
Output: Unique characters list

- Open training document.
- Read all data from the document.
- Generate a list of unique characters present in the training document.
- Sort the unique characters list.
- Dump the list into the dictionary file specified.

Identification Phase:
Input: 1) Document to be identified
Output:1) Language match count

- Open input document and read all data.
- Generate a list of unique characters in the document.
- Sort the generated list.
- Get a list of script folders in the lang_dicts folder
- for each script_folder in lang_dicts folder:
  - Get a list of language files in the script folder
  - For each language:
    - Load the language list of unique characters
    - Get match count of characters present in document list that are also present in language list.
- Display match count statistics for each language.

The language unique character lists are stored in a file folder hierarchy as displayed below:

- lang_dicts/(root folder)
  - script folder 1/
    - language1 list file
    - language2 list file
  - script folder 2/
    - language 3 list file

Unicode based inter-document language/script identification: The use of Unicode in our research has proved to be very useful. The global acceptance of Unicode and the well thought out placement of different languages in the Unicode code-set has given us more opportunities to detect languages effectively. Each script/language in Unicode has defined code ranges. This information is provided in the form of Unicode Database giving type, language, code point and other info for every character of every language represented in Unicode (JWS home page, 2005) and (Hedlund, 2002). The use of this database enabled us to detect different scripts in a document by querying the Unicode database for info about any character.

Unicode based identification is the only technique in our research that does not require an explicit training phase. Only using updated versions of the Unicode database provided at www.unicode.org is sufficient for improving the technique.

This technique is the fastest technique for language detection that we have implemented. As added benefit, this technique also allows us to identify different script portions within a document.

Like all, this technique also has its drawbacks. First, for those scripts that have many languages in them (Roman, Arabic, etc.) this technique can only detect the script and not the language. Secondly for inter-document language identification, detection of different language chunks is effective if consecutive language chunks belong to different scripts. For example, it accurately

separates Arabic and English language chunks repeating one after another but has problems if English and French or German language chunks start repeating after one another.

Algorithm:

Input: 1) Input Document
Output: List of languages and their byte ranges in the input document.

- Read all data from input document.
- set current script = ""
- For each character in data:
  - Get Unicode category of the character (Letter, Digit, Punctuation, etc.)
  - If category = Letter
    - Get Unicode character name
    - Get the script name portion from the character name
  - if script_name != current_script:
    - Add last script to language chunks with start and end byte positions
    - set current script = new script name
- Add last script to the language chunks list
- Remove first empty chunk from the list.

Cavnar's and Trenkle's algorithm: C&T algorithm (Kornai and Richards, 2002) is the most accurate algorithm around for language detection. This algorithm concentrates on alphabet-combination characteristics of languages. Because of this property, this algorithm excels where other algorithms fail. It more accurately identifies languages belonging to the same script.

This algorithm performs its calculations on n-grams. The value of n can be any digit (1,2,3, etc.) The number of n-grams in a document is equal to the number of characters in that document. For example, take the text HELLO WORLD. 1-grams of this text are: 'H', 'E', 'L', 'L', 'O', ' ', 'W', 'O', 'R', 'L', 'D'. 2-grams for the same text are: 'HE', 'EL', 'LL', 'LO', 'O ', ' W', 'WO', 'OR', 'RL', 'LD', 'DH'. The advantage of using n-grams is that they highlight the language properties. Finding common alphabetic combinations of a language is the purpose of processing n-grams.

Over the years modifications and filtering techniques have been applied to further improve the performance of this technique. The most recent successful addition to the technique was by Hayati (2004), which included Fisher Discriminant function to give more importance to n-grams that were more unique across languages. We have further implemented another modification to the process by that serves as a replacement of Fisher discriminant function which was too time consuming for n-grams with n greater than 3.

The training phase for this technique spans across multiple steps that are:

- Document filtering to remove formatting information from documents.
- Document normalization for removing multiple white spaces.
- Training multiple documents per language so that sufficient training data per language is present to make the calculations more real.
- Sorting the resulting statistics and calculating top C n-grams according to weight for each language. In this research C = 1000 was found to be a good value.

The identification phase involves generating n-grams for the input document and then comparing top C n-grams from the document again top C n-grams for each language to find out match value for that language.

By default the algorithm runs with our custom modifications (called FF modifications here) included. Since comparing the statistics against fisher discriminant implementation was also required to get validated results, so a fisher discriminant implementation of the algorithm is also provided.

Drawbacks of this method involve slower detection and more time required to train documents. Additionally adding a new language to the corpus of languages means doing all the calculations again for each language.

Algorithm (with FF modification):

Training (Step 1):
Inputs: 1) Language Name
2) Training Document
Output: NILL

- If the lang folder exists inside the data folder
  - Load previously trained data for n-grams (n = 1-4) from the data files present in the lang folder into n-gram lists.
- If lang folder does not exist:
  - Create lang folder
  - Initialize n-grams lists to blank
- Read all data from training document
- Normalize the data by replacing multiple white spaces with a single space.
- Generate n-grams from input data.

- Calculate weights (frequency) for each n-gram in input document.
- Update weight of each n-gram in the n-gram lists (loaded from previous training data files).
- Dump n-grams lists to lang folder inside the data folder.

Training (Step 2):
Inputs: 1) Source n-gram list
2) Target n-gram list
3) C (Top No. of n-grams to store)
Output: NILL

- Load Source n-gram list.
- Sort the list in descending order.
- Trim the list to top C items.
- Dump the new list to Target n-gram list path.

Training (Step 3)-n-gram Uniqueness Calculation:
Input: 1) Lang
Output: NILL

- Load n-gram lists from lang folder in the data_finalizer folder.
- Get list of languages in data_finalizer folder.
- For each n-gram in n-gram lists:
  - set u_ng = C * total_langs
  - For each lang:
    - if n-gram found in lang list:
      - idx = index of n-gram in current lang list.
      - u_ng = u_ng-idx
  - Add uniqueness value of n-gram to uniqueness list
- Dump n-gram uniqueness lists to lang folder.

Identification phase:
Input: 1) Input Document
Output: Language match count

- Read the data from document
- Normalize the data.
- Generate n-grams (1-4) for the data.
- Calculate n-gram weights (Frequencies) in data.
- Sort document n-grams list in descending order.
- Trim list to top C items.
- Get list of language folders in the data_finalizer folder.
- For each language:
  - Load lang n-gram lists.
  - Load n-gram uniqueness values.
  - For each document n-gram list (n= 1-4):
    - set diff = 0
    - For each n-gram in list:
      - If n-gram found in lang list:
      - set i1 = index of n-gram in doc list
      - set i2 = index of n-gram in lang list
      - diff += i1-i2
      - uv = unique-ness value of n-gram
      - diff- = (uv/100)
        - if n-gram not found in lang list
          - diff += C+1
          - uv = unique-ness value of n-gram
          - diff-= (uv/100)
- return/display match statistics

Folder structure used in this algorithm is given below:

- data/
  - lang1/
    - 1grams.dat
    - 2grams.dat
    - 3grams.dat
    - 4grams.dat
  - lang2/
    - 1grams.dat
    - 2grams.dat
    - 3grams.dat
    - 4grams.dat

    so on
- data_finalizer/
  - lang1/
    - 1grams.dat
    - 2grams.dat
    - 3grams.dat
    - 4grams.dat
    - u_1grams.dat
    - u_2grams.dat
    - u_3grams.dat
    - u_4grams.dat
  - lang2/
    - 1grams.dat
    - 2grams.dat
    - 3grams.dat
    - 4grams.dat
    - u_1grams.dat
    - u_2grams.dat
    - u_3grams.dat
    - u_4grams.dat

Algorithm (Fisher discriminant):

Training phase:
Pre-condition: All training documents are present in training documents folder.

- Generate a list all unique n-grams found in all training documents of each language. Let's call this list A.
- Get a list of language in the training documents folder.
- For each lang:
- Get a list of training docs in that lang.
- For each doc:
  - For each n-gram in A:
    - Find frequency of n-gram in doc.
    - Find normalized frequency of n-gram in doc.
- For each n-gram in A:
  - Calculate mean frequency of n-gram
- Sort the list of mean frequencies in descending order of frequency.
- Trim this list to top 1000 items. Lets call this list R.
- For each lang:
  - Calculate lang fisher values
    - For each n-gram in A
      - Calculate lang mean frequency.
      - For each n-gram in R:
      - fR = mean frequency of n-gram in R.
      - fL = mean frequency of n-gram in current lang.
      - n-gram fisher value = FL/fR
- Save lang fisher values.

Identification phase:
Input: Input document

- Read document data.
- Normalize data.
- Get a list of unique n-grams present in the document.
- For each n-gram in document n-grams:
  - Calculate frequency of n-gram in document.
  - Load R (Top thousand n-grams by mean frequency, generated during training phase).
  - For each n-gram in R:
    - If n-gram present in document n-grams:
    - Calculate n-gram normalized frequency in document
  - else:
    - Set normalized frequency of n-gram in doc to 0.00.
- Get a list of Languages available for identification
- For each lang:
- Load lang fisher values.
- set diff = 0.00
- For each n-gram in document normalized frequencies list:
  - set nFV = lang fisher value for ngram
  - set nNF = document normalized frequency for ngram
  - diff += nNF * nFV
- Display diff as lang match value for document.

IMPLEMENTATION DETAILS

All the algorithms in our research were tested on 4 scripts containing a total of 10 languages.
The scripts used are:

- Arabic
- Chinese
- Japanese
- Roman

The languages used are:

- Arabic
- Chinese
- English
- French
- German
- Italian
- Japanese
- Persian
- Spanish
- Urdu

Training documents of roughly equal data size were used for training each language to ensure that all languages get equal share of training. All training documents were manually cleaned. Portions of languages other than the desired one were removed from each training document and all the documents were saved in UTF-8 encoding.

Training documents were mostly gathered from the web primarily in HTML format. The other formats used were UTF-8 encoded text, MS. WORD DOC format and PDF.

All the code except for the web interface is coded in Python programming language. The web interface is coded in PHP and HTML running on Apache web server. Operating system used for development and testing is Linux Kernel 2.4 (Slackware 10.2) on a Pentium IV (2.4 GHz) machine with 256 MB of RAM.

FEATURES

Unicode based script/language identification: The system is not only able to read/write Unicode files; it also uses Unicode extensively for language identification. Use of Unicode makes language identification much more flexible, accurate and faster.

Unicode based script separation within a single document: Unicode code point information is used to detect the presence of multiple scripts in a document. Data of each script is then processed separately and language identification results for each script chunk are returned.

Dictionary based (fast) language detection method: Alphabets in the input document are compared against the alphabets of different languages and based on that the language of the input document is guessed. This method is the most used method for language identification. Although simple, the results returned from this method were found to be satisfactorily accurate and quite fast.

N-gram based (more accurate) modified method: This is a modified (improved) version of Cavnar's and Trenkle's algorithm. The modifications give higher weights to n-grams that are less common in other languages.

Automatic removal of document formatting information: The system handles input documents like web pages very well and automatically removes all tags and other formatting info from the page using only the page content for language identification.

Support for easily adding more languages into the system: More languages can be easily trained using a couple of training programs to expand the system to support more languages.

RESULTS

Detection of multiple languages within a single script using Unicode Meta data is most useful for multi-lingual documents. Language chunks belonging to different scripts were always identified and extracted correctly by this method. However, detecting language chunks belonging to same script is not possible with this technique and requires further work.

Dictionary based identification serves well for script identification. Language identification results give close match values for languages belonging to the same script as of the original documents.

The N-gram based implementations (CTA with FF modifications and Fisher Discriminant Function) were found to be most accurate giving above 90% accuracy. Training phase of CTA with FF was observed to take much less time than Fisher Discriminant. Identification times and results for both variants were observed to be similar.

For very short documents (less those 30 characters) all algorithms were observed to perform badly for language detection. However, accurate script detection of these documents using Unicode Meta info or Dictionary Based Identification was still accurate.

STATISTICS

Total size of cleaned training documents: 2.7 MB.

Technique	Operation	Time
DBI	Training	0 m 0.117 s
	Identification	0 m 0.334 s
CTA With FF Modification	Training (Generating n-grams[1 to 4])	80 m 1.002 s (1 h:20 m)
	Training (Finalizing)	160 m 19.854 s (2 h:40 m: 90 s)
	Identification	0 m 17.303 s
CTA With Fisher Discriminant (for 3-grams)	Training (Generating n-grams[3 g])	3926.53322601 (1 h: 5 m)
	Training (Calculating Frequencies)	30 m 1.119 s
	Training (Calculating fisher values)	96 m 9.879 s
	Identification	0 m 18.134 s

FUTURE ENHANCEMENTS

With the ever changing face of information in the computer world, more possibilities for improving language detection are just around the corner. There are many standards that save meta-data along with documents. The increasing adoption of such standards (like XML) will allow use of document Meta data for language identification. To do so detail study of different document formats needs to be done along with widespread use of open file formats instead of closed specifications formats.

Automatic generation of training documents without having the need to check them manually is also a field that requires more research. Currently the trainer requires good knowledge of the languages that he/she wishes to train for. Training documents need to contain only the language that they will be used to train. This process involves manual cleaning of documents. Some technique

can be developed to use online digital archives like digital libraries to automatically obtain books or other documents for different languages, clean them and then use them for training the system.

Detection of multiple languages belonging to the same script within a document is another field that needs improvement. Currently this is being done using Unicode script ranges that allow only script detection. This poses problems detecting multiple languages belonging to the same script occurring in tandem. New ways need to be developed based language properties to detect this.

REFERENCES

Hayati, K., 2004. Language identification on the world wide web. Report of degree of master of science in computer science, University of California, Santa Cruz.

Hedlund, T., 2002. Compunds in dictionary based cross language information retrieval. University of Tampere, Finland. Informa. Res., Vol. 7 [Available at http://www.informationR.net/ir/7-2/paper128.html]

Kornai, A. and J.M. Richards, 2002. Linear Discriminant Text Classification in High Dimension In: Hybrid Information System Physica, Abraham, A. and M. Kooeppen (Eds.). Verlag, Heidelberg, pp: 527-538.

Kuhn, M., 2004. UTF-8 and Unicode FAQ for Unix/Linux. Report of Computer Laboratory, University of Cambridge, UK.

JWS web site. Multi-lingual text on Linux. 2005. (http://www.jw-stumpel.nl/stestu.html).