Finding and Identifying Text in 900+ Languages

Ralf D. Brown
Carnegie Mellon University
Language Technologies Institute

6 August 2012
Finding and Identifying Text in 1000 Languages

Ralf D. Brown
Carnegie Mellon University
Language Technologies Institute

6 August 2012
Executive Summary

- Open-source (GPLv3), trainable tool to extract textual strings and identify their languages
  - http://la-strings.sourceforge.net/
- False alarm rate < 0.4%, miss rate < 0.01%
- Language identification accuracy >99% on a 1000-language evaluation set
Overview

• Why yet another string-extraction tool?
• Language models
• Identifying character encodings and languages
• Where to get language data
• Experimental results
• Future work
The Need for String Extraction

- Damaged files
- Text hidden inside non-text data
- Disk images
Existing “Strings” Utilities

- Limited support for non-ASCII text
- No knowledge of language
  - Extract every sequence that is valid in the specified encoding
  - Thus have a high false alarm rate
Desirable Features for a Text Extractor

- Support as many character encodings as possible
- Automatically identify the encoding(s) used
- Filter out non-text sequences
- Language identification to permit intelligent downstream processing
Language Models

- Statistics for variable-length byte sequences found in training data
- One model (or more) for each language/encoding pair we want to identify
The “Secret Sauce”

- Selection of the most useful n-grams
- Use of negative evidence ("stop-grams")
- Inter-string score smoothing
  - Assumption is that consecutive strings are most likely in the same language
Picking the Most Useful N-grams

- Collect the most frequent byte n-grams up to some maximum length
- Filter out high-frequency n-grams which don't add much information
  - If the n-gram is a substring of another with at least 90% as many occurrences
Using Negative Evidence

• If an n-gram is never seen in the training data but is common in another, similar language
  – Give it a negative weight proportional to its frequency in the other language and the degree of similarity between the two languages
Inter-String Smoothing

- Add a portion of the previous string's score to current string
- Use exponential decay
  - New smoothing value = \text{curr\_score} + 0.25 \times \text{prev\_value}
- Relative weight of current string's score adjusted by string length
  - Longer strings have more reliable scores
Identifying Languages

- Given an input string and a set of language models:
  - At each offset in the input, find the matching n-grams in the models and increment the corresponding scores by the n-gram's weight
  - At the end of the string, sort the models by total score
  - Output the top K languages which have scores at least 0.85 times the highest score
Identifying Character Encoding

- Same as identifying languages, but instead of looking at the language associated with each model, use the encoding
  - Remove lower-scoring duplicates before selecting
  - Use encodings with score at least 0.3 times highest, and above a predefined threshold
Extracting Strings

- Begin by identifying probably character encodings for fixed-size blocks of bytes
- At every byte position within a block,
  - Attempt to extract a string in each identified encoding
  - Longest string at a position is taken as correct
- Identify the language of each extracted string
  - Discard if confidence score is too low
Scanning for Encodings

Input Data

Latin-1
UTF-8 / Latin-1
UTF-8
UTF-8
Extracting Strings

Input Data

Latin-1

UTF-8 / Latin-1

UTF-8

En
1.3

Fr
7.7

De
4.8

Ru
9.1

Es
3.5

Es
2.9
Obtaining Training Data

- **Wikipedia**
  - 285 languages, ~200 with useful amounts of text

- **Bible translations**
  - Full Bible has been translated into 475 languages
  - New Testament in 1240 languages
  - Hundreds have been made available online since 2010
Experiments: Data

- Built models for 1026 languages, several in multiple writing systems
- For the majority of languages, the training data was a translation of the New Testament
  - Median training data size of 1.4 million bytes (quartiles 1.0 million and 2.0 million)
- Held out ~3% of training data for evaluation
Experiments: Test Conditions

• Varied three different parameters
  – Amount of training data (use only first B bytes)
  – Number of highest-frequency n-grams in model
  – Maximum length of n-gram in model

• Computed micro- and macro-average error rates with and without inter-string smoothing
  – Also micro-average without discriminative training when restricting training data
Experiments: Results

- Error rate decreases smoothly as training data increases and as the number of n-grams in each model increases
- Increasing maximum n-gram length eventually starts increasing error rate again
- Inter-string smoothing cuts errors by about half
- Discriminative training reduces error rates with more than 250k training data per model
Performance by Training Data Size

![Graph showing performance by training data size](image)
Performance by Training Data Size
(Detail: low data)

![Graph showing error rate vs. training data size.](image-url)

- Raw errors (no discriminative training), micro-average
- Raw errors, micro-average
- Raw errors, macro-average
- Smoothed errors (no discriminative training), micro-average
- Smoothed errors, micro-average
- Smoothed errors, macro-average
Performance by Training Data Size
(Detail: high data)
Performance by N-gram Count

![Graph showing performance by N-gram count. The graph plots error rate (%) on the y-axis against number of n-grams per language on the x-axis. Various lines and markers represent different error rates and database sizes.]
Performance by Max. N-gram Length
(topK = 3000)
Performance by Max. N-gram Length
(topK = 9000)

![Graph showing error rate vs. maximum n-gram length]

- Raw errors, micro-avg
- Raw errors, macro-avg
- Smoothed errors, micro-avg
- Smoothed errors, macro-avg
- n-gram database size

Error Rate (%) vs. Maximum n-gram length

Database size (millions of bytes)
Performance on Top Languages

![Graph showing the relationship between error rate and number of n-grams per language, with different lines representing various error metrics and database sizes.](image-url)
Missed and Falsely Identified Text

● Miss vs False Alarms on running text
  – Low threshold: 0.002% miss / 0.34% false alarm
  – High threshold: 0.009% miss / 0.012% false alarm

● Miss vs False Alarms for isolated strings not fully characterized yet
  – Seems to average about one (short) false-alarm string following each true string as a result of smoothing
Other Measures of Performance

- **Speed (full database of 3397 models)**
  - ~1.7MB/s on random bytes
  - ~160 KB/s on running text

- **Speed (restricted database of 454 models)**
  - ~3.5MB/s on random bytes
  - ~800 KB/s on running text

- **RAM**
  - Database is shared memory, only 4MB private RAM
Future Work

- Improved discriminative training
- Increased speed
  - Even 3.5 MB/s is too slow for terabyte disk images
Conclusion

- Presented a trainable open-source tool to extract textual strings and identify their language
- High accuracy on both string extraction and language identification
- Reasonable speed
- Available from http://la-strings.sourceforge.net/
  - Includes pre-trained models for 1026 languages
  - Training data for over 500 languages available (Creative Commons licenses)
Thank You.

Questions?