`langid.py`: An Off-the-shelf Language Identification Tool

Marco Lui and Timothy Baldwin
NICTA VRL
Department of Computing and Information Systems
University of Melbourne, VIC 3010, Australia
`mhlui@unimelb.edu.au, tb@ldwin.net`

Abstract

We present `langid.py`, an off-the-shelf language identification tool. We discuss the design and implementation of `langid.py`, and provide an empirical comparison on 5 long-document datasets, and 2 datasets from the microblog domain. We find that `langid.py` maintains consistently high accuracy across all domains, making it ideal for end-users that require language identification without wanting to invest in preparation of in-domain training data.

1 Introduction

Language identification (LangID) is the task of determining the natural language that a document is written in. It is a key step in automatic processing of real-world data, where a multitude of languages may be present. Natural language processing techniques typically pre-suppose that all documents being processed are written in a given language (e.g. English), but as focus shifts onto processing documents from internet sources such as microblogging services, this becomes increasingly difficult to guarantee. Language identification is also a key component of many web services. For example, the language that a web page is written in is an important consideration in determining whether it is likely to be of interest to a particular user of a search engine, and automatic identification is an essential step in building language corpora from the web. It has practical implications for social networking and social media, where it may be desirable to organize comments and other user-generated content by language. It also has implications for accessibility, since it enables automatic determination of the target language for automatic machine translation purposes.

Many applications could potentially benefit from automatic language identification, but building a customized solution per-application is prohibitively expensive, especially if human annotation is required to produce a corpus of language-labelled training documents from the application domain. What is required is thus a generic language identification tool that is usable *off-the-shelf*, i.e. with no end-user training and minimal configuration.

In this paper, we present `langid.py`, a LangID tool with the following characteristics: (1) fast, (2) usable off-the-shelf, (3) unaffected by domain-specific features (e.g. HTML, XML, markdown), (4) single file with minimal dependencies, and (5) flexible interface

2 Methodology

`langid.py` is trained over a naive Bayes classifier with a multinomial event model (McCallum and Nigam, 1998), over a mixture of byte n-grams ($1 \leq n \leq 4$). One key difference from conventional text categorization solutions is that `langid.py` was designed to be used *off-the-shelf*. Since `langid.py` implements a supervised classifier, this presents two primary challenges: (1) a pre-trained model must be distributed with the classifier, and (2) the model must generalize to data from different *domains*, meaning that in its default configuration, it must have good accuracy over inputs as diverse as web pages, newspaper articles and microblog messages. (1) is mostly a practical consideration, and so we will address it in Section 3. In order to address (2), we integrate information about the language identification task from a variety of domains by using $\mathcal{LD}$ feature selection (Lui and Baldwin, 2011).

Lui and Baldwin (2011) showed that it is relatively easy to attain high accuracy for language iden-

Dataset	Documents	Langs	Doc Length (bytes)
EUROGOV	1500	10	$1.7\times10^4 \pm3.9\times10^4$
TCL	3174	60	$2.6\times10^3 \pm3.8\times10^3$
WIKIPEDIA	4963	67	$1.5\times10^3 \pm4.1\times10^3$
EMEA	19988	22	$2.9\times10^5 \pm7.9\times10^5$
EUROPARL	20828	22	$1.7\times10^2 \pm1.6\times10^2$
T-BE	9659	6	$1.0\times10^2 \pm3.2\times10^1$
T-SC	5000	5	$8.8\times10^1 \pm3.9\times10^1$

Table 1: Summary of the LangID datasets

tification in a traditional text categorization setting, where we have in-domain training data. The task becomes much harder when trying to perform *domain adaptation*, that is, trying to use model parameters learned in one domain to classify data from a different domain. $\mathcal{LD}$ feature selection addresses this problem by focusing on key features that are relevant to the language identification task. It is based on Information Gain (IG), originally introduced as a splitting criteria for decision trees (Quinlan, 1986), and later shown to be effective for feature selection in text categorization (Yang and Pedersen, 1997; Forman, 2003). $\mathcal{LD}$ represents the difference in IG with respect to language and domain. Features with a high $\mathcal{LD}$ score are informative about language without being informative about domain. For practical reasons, before the IG calculation the candidate feature set is pruned by means of a term-frequency based feature selection.

Lui and Baldwin (2011) presented empirical evidence that $\mathcal{LD}$ feature selection was effective for domain adaptation in language identification. This result is further supported by our evaluation, presented in Section 5.

3 System Architecture

The full `langid.py` package consists of the language identifier `langid.py`, as well as two support modules `LDfeatureselect.py` and `train.py`.

`langid.py` is the single file which packages the language identification tool, and the only file needed to use `langid.py` for off-the-shelf language identification. It comes with an embedded model which covers 97 languages using training data drawn from 5 domains. Tokenization and feature selection are carried out in a single pass over the input document via Aho-Corasick string matching (Aho and Corasick, 1975). The Aho-Corasick string matching algorithm processes an input by means of a deterministic finite automaton (DFA). Some states of the automaton are associated with the completion of one of the n-grams selected through $\mathcal{LD}$ feature selection. Thus, we can obtain our document representation by simply counting the number of times the DFA enters particular states while processing our input. The DFA and the associated mapping from state to n-gram are constructed during the training phase, and embedded as part of the pre-trained model.

The naive Bayes classifier is implemented using `numpy`,[1] the de-facto numerical computation package for Python. `numpy` is free and open source, and available for all major platforms. Using `numpy` introduces a dependency on a library that is not in the Python standard library. This is a reasonable trade-off, as `numpy` provides us with an optimized implementation of matrix operations, which allows us to implement fast naive Bayes classification while maintaining the single-file concept of `langid.py`.

`langid.py` can be used in the three ways:

Command-line tool: `langid.py` supports an interactive mode with a text prompt and line-by-line classification. This mode is suitable for quick interactive queries, as well as for demonstration purposes. `langid.py` also supports language identification of entire files via redirection. This allows a user to interactively explore data, as well as to integrate language identification into a pipeline of other `unix`-style tools. However, use via redirection is not recommended for large quantities of documents as each invocation requires the trained model to be unpacked into memory. Where large quantities of documents are being processed, use as a library or web service is preferred as the model will only be unpacked once upon initialization.

Python library: `langid.py` can be imported as a Python module, and provides a function that accepts text and returns the identified language of the text. This use of `langid.py` is the fastest in a single-processor setting as it incurs the least overhead.

Web service: `langid.py` can be started as a web service with a command-line switch. This

[1] `http://numpy.scipy.org`

allows language identitication by means of HTTP PUT and HTTP POST requests, which return JSON-encoded responses. This is the preferred method of using `langid.py` from other programming environments, as most languages include libraries for interacting with web services over HTTP. It also allows the language identification service to be run as a network/internet service. Finally, `langid.py` is WSGI-compliant,[2] so it can be deployed in a WSGI-compliant web server. This provides an easy way to achieve parallelism by leveraging existing technologies to manage load balancing and utilize multiple processors in the handling of multiple concurrent requests for a service.

`LDfeatureselect.py` implements the $\mathcal{LD}$ feature selection. The calculation of term frequency is done in constant memory by index inversion through a MapReduce-style sharding approach. The calculation of information gain is also chunked to limit peak memory use, and furthermore it is parallelized to make full use of modern multiprocessor systems. `LDfeatureselect.py` produces a list of byte n-grams ranked by their $\mathcal{LD}$ score.

`train.py` implements estimation of parameters for the multinomial naive Bayes model, as well as the construction of the DFA for the Aho-Corasick string matching algorithm. Its input is a list of byte patterns representing a feature set (such as that selected via `LDfeatureselect.py`), and a corpus of training documents. It produces the final model as a single compressed, encoded string, which can be saved to an external file and used by `langid.py` via a command-line option.

4 Training Data

`langid.py` is distributed with an embedded model trained using the multi-domain language identification corpus of Lui and Baldwin (2011). This corpus contains documents in a total of 97 languages. The data is drawn from 5 different domains: government documents, software documentation, newswire, online encyclopedia and an internet crawl, though no domain covers the full set of languages by itself, and some languages are present only in a single domain. More details about this corpus are given in Lui and Baldwin (2011).

[2]`http://www.wsgi.org`

We do not perform explicit encoding detection, but we do not assume that all the data is in the same encoding. Previous research has shown that explicit encoding detection is not needed for language identification (Baldwin and Lui, 2010). Our training data consists mostly of UTF8-encoded documents, but some of our evaluation datasets contain a mixture of encodings.

5 Evaluation

In order to benchmark `langid.py`, we carried out an empirical evaluation using a number of language-labelled datasets. We compare the empirical results obtained from `langid.py` to those obtained from other language identification toolkits which incorporate a pre-trained model, and are thus usable *off-the-shelf* for language identification. These tools are listed in Table 3.

5.1 Off-the-shelf LangID tools

`TextCat` is an implementation of the method of Cavnar and Trenkle (1994) by Gertjan van Noord. It has traditionally been the de facto LangID tool of choice in research, and is the basis of language identification/filtering in the ClueWeb09 Dataset (Callan and Hoy, 2009) and CorpusBuilder (Ghani et al., 2004). It includes support for training with user-supplied data.

`LangDetect` implements a Naive Bayes classifier, using a character n-gram based representation without feature selection, with a set of normalization heuristics to improve accuracy. It is trained on data from Wikipedia,[3] and can be trained with user-supplied data.

`CLD` is a port of the embedded language identifier in Google's Chromium browser, maintained by Mike McCandless. Not much is known about the internal design of the tool, and there is no support provided for re-training it.

The datasets come from a variety of domains, such as newswire (TCL), biomedical corpora (EMEA), government documents (EUROGOV, EUROPARL) and microblog services (T-BE, T-SC). A number of these datasets have been previously used in language identification research. We provide a

[3]`http://www.wikipedia.org`

Test Dataset	langid.py		LangDetect		TextCat		CLD	
	Accuracy	docs/s	ΔAcc	Slowdown	ΔAcc	Slowdown	ΔAcc	Slowdown
EUROGOV	0.987	70.5	+0.005	1.1×	−0.046	31.1×	−0.004	0.5×
TCL	0.904	185.4	−0.086	2.1×	−0.299	24.2×	−0.172	0.5×
WIKIPEDIA	0.913	227.6	−0.046	2.5×	−0.207	99.9×	−0.082	0.9×
EMEA	0.934	7.7	−0.820	0.2×	−0.572	6.3×	+0.044	0.3×
EUROPARL	0.992	294.3	+0.001	3.6×	−0.186	115.4×	−0.010	0.2×
T-BE	0.941	367.9	−0.016	4.4×	−0.210	144.1×	−0.081	0.7×
T-SC	0.886	298.2	−0.038	2.9×	−0.235	34.2×	−0.120	0.2×

Table 2: Comparison of standalone classification tools, in terms of accuracy and speed (documents/second), relative to `langid.py`

Tool	Languages	URL
`langid.py`	97	`http://www.csse.unimelb.edu.au/research/lt/resources/langid/`
`LangDetect`	53	`http://code.google.com/p/language-detection/`
`TextCat`	75	`http://odur.let.rug.nl/vannoord/TextCat/`
`CLD`	64+	`http://code.google.com/p/chromium-compact-language-detector/`

Table 3: Summary of the LangID tools compared

brief summary of the characteristics of each dataset in Table 1.

The datasets we use for evaluation are different from and independent of the datasets from which the embedded model of `langid.py` was produced. In Table 2, we report the accuracy of each tool, measured as the proportion of documents from each dataset that are correctly classified. We present the absolute accuracy and performance for `langid.py`, and relative accuracy and slowdown for the other systems. For this experiment, we used a machine with 2 Intel Xeon E5540 processors and 24GB of RAM. We only utilized a single core, as none of the language identification tools tested are inherently multicore.

5.2 Comparison on standard datasets

We compared the four systems on datasets used in previous language identification research (Baldwin and Lui, 2010) (EUROGOV, TCL, WIKIPEDIA), as well as an extract from a biomedical parallel corpus (Tiedemann, 2009) (EMEA) and a corpus of samples from the Europarl Parallel Corpus (Koehn, 2005) (EUROPARL). The sample of EUROPARL we use was originally prepared by Shuyo Nakatani (author of `LangDetect`) as a validation set.

`langid.py` compares very favorably with other language identification tools. It outperforms `TextCat` in terms of speed and accuracy on all of the datasets considered. `langid.py` is generally orders of magnitude faster than `TextCat`, but this advantage is reduced on larger documents. This is primarily due to the design of `TextCat`, which requires that the supplied models be read from file for each document classified.

`langid.py` generally outperforms `LangDetect`, except in datasets derived from government documents (EUROGOV, EUROPARL). However, the difference in accuracy between `langid.py` and `LangDetect` on such datasets is very small, and `langid.py` is generally faster. An abnormal result was obtained when testing `LangDetect` on the EMEA corpus. Here, `LangDetect` is much faster, but has extremely poor accuracy (0.114). Analysis of the results reveals that the majority of documents were classified as Polish. We suspect that this is due to the early termination criteria employed by `LangDetect`, together with specific characteristics of the corpus. `TextCat` also performed very poorly on this corpus (accuracy 0.362). However, it is important to note that `langid.py` and `CLD` both performed very well, providing evidence that it is possible to build a generic language identifier that is insensitive to domain-specific characteristics.

`langid.py` also compares well with `CLD`. It is generally more accurate, although `CLD` does better on the EMEA corpus. This may reveal some insight into the design of `CLD`, which is likely to have been tuned for language identification of web pages. The EMEA corpus is heavy in XML markup, which `CLD` and `langid.py` both successfully ignore. One area where `CLD` outperforms all other systems is in its speed. However, this increase in speed comes at the cost of decreased accuracy in other domains, as we will see in Section 5.3.

5.3 Comparison on microblog messages

The size of the input text is known to play a significant role in the accuracy of automatic language identification, with accuracy decreasing on shorter input documents (Cavnar and Trenkle, 1994; Sibun and Reynar, 1996; Baldwin and Lui, 2010).

Recently, language identification of short strings has generated interest in the research community. Hammarstrom (2007) described a method that augmented a dictionary with an affix table, and tested it over synthetic data derived from a parallel bible corpus. Ceylan and Kim (2009) compared a number of methods for identifying the language of search engine queries of 2 to 3 words. They develop a method which uses a decision tree to integrate outputs from several different language identification approaches. Vatanen et al. (2010) focus on messages of 5–21 characters, using n-gram language models over data drawn from UDHR in a naive Bayes classifier.

A recent application where language identification is an open issue is over the rapidly-increasing volume of data being generated by social media. Microblog services such as Twitter[4] allow users to post short text messages. Twitter has a worldwide user base, evidenced by the large array of languages present on Twitter (Carter et al., to appear). It is estimated that half the messages on Twitter are not in English. [5]

This new domain presents a significant challenge for automatic language identification, due to the much shorter ‘documents’ to be classified, and is compounded by the lack of language-labelled in-domain data for training and validation. This has led to recent research focused specifically on the task of language identification of Twitter messages. Carter et al. (to appear) improve language identification in Twitter messages by augmenting standard methods with language identification priors based on a user’s previous messages and by the content of links embedded in messages. Tromp and Pechenizkiy (2011) present a method for language identification of short text messages by means of a graph structure.

Despite the recently published results on language identification of microblog messages, there is no dedicated off-the-shelf system to perform the task. We thus examine the accuracy and performance of using generic language identification tools to identify the language of microblog messages. It is important to note that none of the systems we test have been specifically tuned for the microblog domain. Furthermore, they do not make use of any non-textual information such as author and link-based priors (Carter et al., to appear).

We make use of two datasets of Twitter messages kindly provided to us by other researchers. The first is T-BE (Tromp and Pechenizkiy, 2011), which contains 9659 messages in 6 European languages. The second is T-SC (Carter et al., to appear), which contains 5000 messages in 5 European languages.

We find that over both datasets, `langid.py` has better accuracy than any of the other systems tested. On T-BE, Tromp and Pechenizkiy (2011) report accuracy between 0.92 and 0.98 depending on the parametrization of their system, which was tuned specifically for classifying short text messages. In its *off-the-shelf* configuration, `langid.py` attains an accuracy of 0.94, making it competitive with the customized solution of Tromp and Pechenizkiy (2011).

On T-SC, Carter et al. (to appear) report overall accuracy of 0.90 for `TextCat` in the off-the-shelf configuration, and up to 0.92 after the inclusion of priors based on (domain-specific) extra-textual information. In our experiments, the accuracy of `TextCat` is much lower (0.654). This is because Carter et al. (to appear) constrained `TextCat` to output only the set of 5 languages they considered. Our results show that it is possible for a generic language identification tool to attain reasonably high accuracy (0.89) without artificially constraining the set of languages to be considered, which corresponds more closely to the demands of automatic language identification to real-world data sources, where there is generally no prior knowledge of the languages present.

[4]`http://www.twitter.com`

[5]`http://semiocast.com/downloads/Semiocast_Half_of_messages_on_Twitter_are_not_in_English_20100224.pdf`

We also observe that while `CLD` is still the fastest classifier, this has come at the cost of accuracy in an alternative domain such as Twitter messages, where both `langid.py` and `LangDetect` attain better accuracy than `CLD`.

An interesting point of comparison between the Twitter datasets is how the accuracy of all systems is generally higher on T-BE than on T-SC, despite them covering essentially the same languages (T-BE includes Italian, whereas T-SC does not). This is likely to be because the T-BE dataset was produced using a semi-automatic method which involved a language identification step using the method of Cavnar and Trenkle (1994) (E Tromp, personal communication, July 6 2011). This may also explain why `TextCat`, which is also based on Cavnar and Trenkle's work, has unusually high accuracy on this dataset.

6 Conclusion

In this paper, we presented `langid.py`, an off-the-shelf language identification solution. We demonstrated the robustness of the tool over a range of test corpora of both long and short documents (including micro-blogs).

Acknowledgments

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Alfred V. Aho and Margaret J. Corasick. 1975. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, June.

Timothy Baldwin and Marco Lui. 2010. Language identification: The long and the short of the matter. In *Proceedings of NAACL HLT 2010*, pages 229–237, Los Angeles, USA.

Jamie Callan and Mark Hoy, 2009. *ClueWeb09 Dataset*. Available at `http://boston.lti.cs.cmu.edu/Data/clueweb09/`.

Simon Carter, Wouter Weerkamp, and Manos Tsagkias. to appear. Microblog language identification: Overcoming the limitations of short, unedited and idiomatic text. *Language Resources and Evaluation Journal*.

William B. Cavnar and John M. Trenkle. 1994. N-gram-based text categorization. In *Proceedings of the Third Symposium on Document Analysis and Information Retrieval*, Las Vegas, USA.

Hakan Ceylan and Yookyung Kim. 2009. Language identification of search engine queries. In *Proceedings of ACL2009*, pages 1066–1074, Singapore.

George Forman. 2003. An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research*, 3(7-8):1289–1305, October.

Rayid Ghani, Rosie Jones, and Dunja Mladenic. 2004. Building Minority Language Corpora by Learning to Generate Web Search Queries. *Knowledge and Information Systems*, 7(1):56–83, February.

Harald Hammarstrom. 2007. A Fine-Grained Model for Language Identication. In *Proceedings of iNEWS07*, pages 14–20.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. *MT summit*, 11.

Marco Lui and Timothy Baldwin. 2011. Cross-domain feature selection for language identification. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 553–561, Chiang Mai, Thailand.

Andrew McCallum and Kamal Nigam. 1998. A comparison of event models for Naive Bayes text classification. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, Madison, USA.

J.R. Quinlan. 1986. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, October.

Penelope Sibun and Jeffrey C. Reynar. 1996. Language determination: Examining the issues. In *Proceedings of the 5th Annual Symposium on Document Analysis and Information Retrieval*, pages 125–135, Las Vegas, USA.

Jörg Tiedemann. 2009. News from OPUS - A Collection of Multilingual Parallel Corpora with Tools and Interfaces. *Recent Advances in Natural Language Processing*, V:237–248.

Erik Tromp and Mykola Pechenizkiy. 2011. Graph-Based N-gram Language Identification on Short Texts. In *Proceedings of Benelearn 2011*, pages 27–35, The Hague, Netherlands.

Tommi Vatanen, Jaakko J. Vayrynen, and Sami Virpioja. 2010. Language identification of short text segments with n-gram models. In *Proceedings of LREC 2010*, pages 3423–3430.

Yiming Yang and Jan O. Pedersen. 1997. A comparative study on feature selection in text categorization. In *Proceedings of ICML 97*.