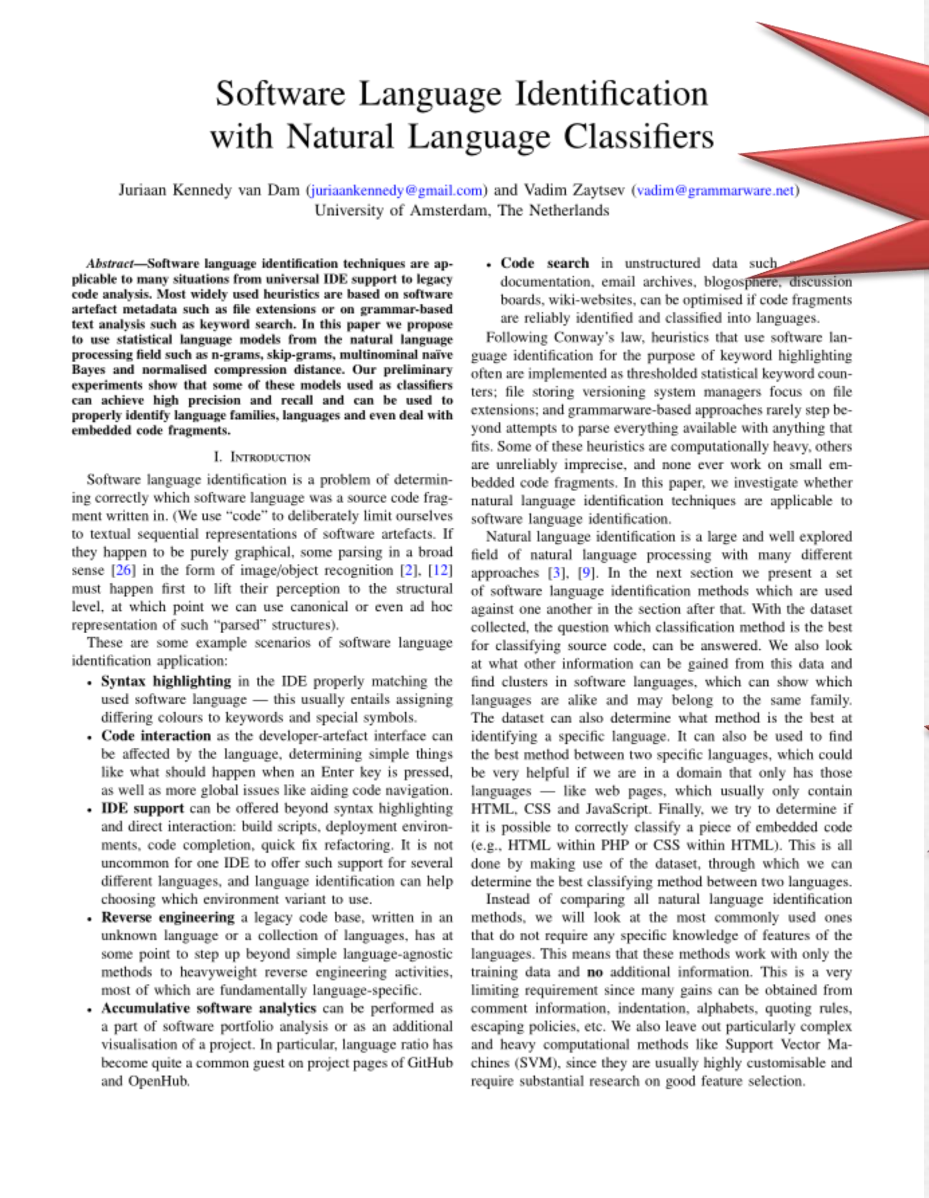
SANER ERA 2016

Software Language Identification with NL Classifiers

Juriaan Kennedy van Dam and Vadim Zaytsev

If You Want to Know What is in the Paper



Software Language Identification with Natural Language Classifiers

Juriaan Kennedy van Dam (juriaankennedy@gmail.com) and Vadim Zaytsev (vadim@grammarware.net)
University of Amsterdam, The Netherlands

Abstract—**Software language identification techniques are applicable to many situations from universal IDE support to legacy code analysis. Most widely used heuristics are based on software artefact metadata such as file extensions or on grammar-based text analysis such as keyword search. In this paper we propose to use statistical language models from the natural language processing field such as n-grams, skip-grams, multinominal naïve Bayes and normalised compression distance. Our preliminary experiments show that some of these models used as classifiers can achieve high precision and recall and can be used to properly identify language families, languages and even deal with embedded code fragments.**

I. Introduction

Software language identification is a problem of determining correctly which software language was a source code fragment written in. (We use "code" to deliberately limit ourselves to textual sequential representations of software artefacts. If they happen to be purely graphical, some parsing in a broad sense [26] in the form of image/object recognition [2], [12] must happen first to lift their perception to the structural level, at which point we can use canonical or even ad hoc representation of such "parsed" structures).

These are some example scenarios of software language identification application:

- **Syntax highlighting** in the IDE properly matching the used software language — this usually entails assigning differing colours to keywords and special symbols.
- **Code interaction** as the developer-artefact interface can be affected by the language, determining simple things like what should happen when an Enter key is pressed, as well as more global issues like aiding code navigation.
- **IDE support** can be offered beyond syntax highlighting and direct interaction: build scripts, deployment environments, code completion, quick fix refactoring. It is not uncommon for one IDE to offer such support for several different languages, and language identification can help choosing which environment variant to use.
- **Reverse engineering** a legacy code base, written in an unknown language or a collection of languages, has at some point to step up beyond simple language-agnostic methods to heavyweight reverse engineering activities, most of which are fundamentally language-specific.
- **Accumulative software analytics** can be performed as a part of software portfolio analysis or as an additional visualisation of a project. In particular, language ratio has become quite a common guest on project pages of GitHub and OpenHub.
- **Code search** in unstructured data such documentation, email archives, blogosphere, discussion boards, wiki-websites, can be optimised if code fragments are reliably identified and classified into languages.

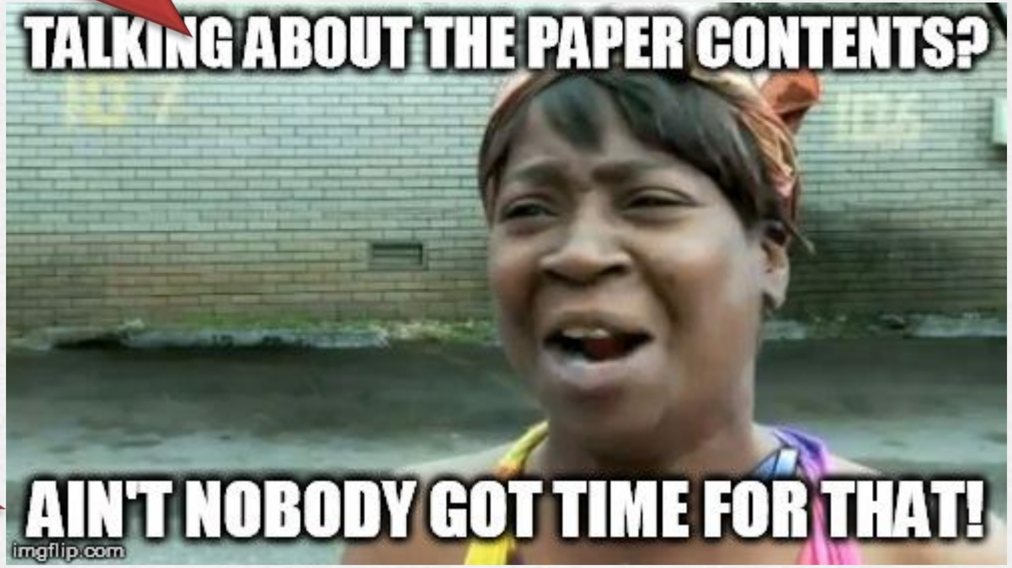
Following Conway's law, heuristics that use software language identification for the purpose of keyword highlighting often are implemented as thresholded statistical keyword counters; file storing versioning system managers focus on file extensions; and grammarware-based approaches rarely step beyond attempts to parse everything available with anything that fits. Some of these heuristics are computationally heavy, others are unreliably imprecise, and none ever work on small embedded code fragments. In this paper, we investigate whether natural language identification techniques are applicable to software language identification.

Natural language identification is a large and well explored field of natural language processing with many different approaches [3], [9]. In the next section we present a set of software language identification methods which are used against one another in the section after that. With the dataset collected, the question which classification method is the best for classifying source code, can be answered. We also look at what other information can be gained from this data and find clusters in software languages, which can show which languages are alike and may belong to the same family. The dataset can also determine what method is the best at identifying a specific language. It can also be used to find the best method between two specific languages, which could be very helpful if we are in a domain that only has those languages — like web pages, which usually only contain HTML, CSS and JavaScript. Finally, we try to determine if it is possible to correctly classify a piece of embedded code (e.g., HTML within PHP or CSS within HTML). This is all done by making use of the dataset, through which we can determine the best classifying method between two languages.

Instead of comparing all natural language identification methods, we will look at the most commonly used ones that do not require any specific knowledge of features of the languages. This means that these methods work with only the training data and **no** additional information. This is a very limiting requirement since many gains can be obtained from comment information, indentation, alphabets, quoting rules, escaping policies, etc. We also leave out particularly complex and heavy computational methods like Support Vector Machines (SVM), since they are usually highly customisable and require substantial research on good feature selection.

papers/1855a624.pdf

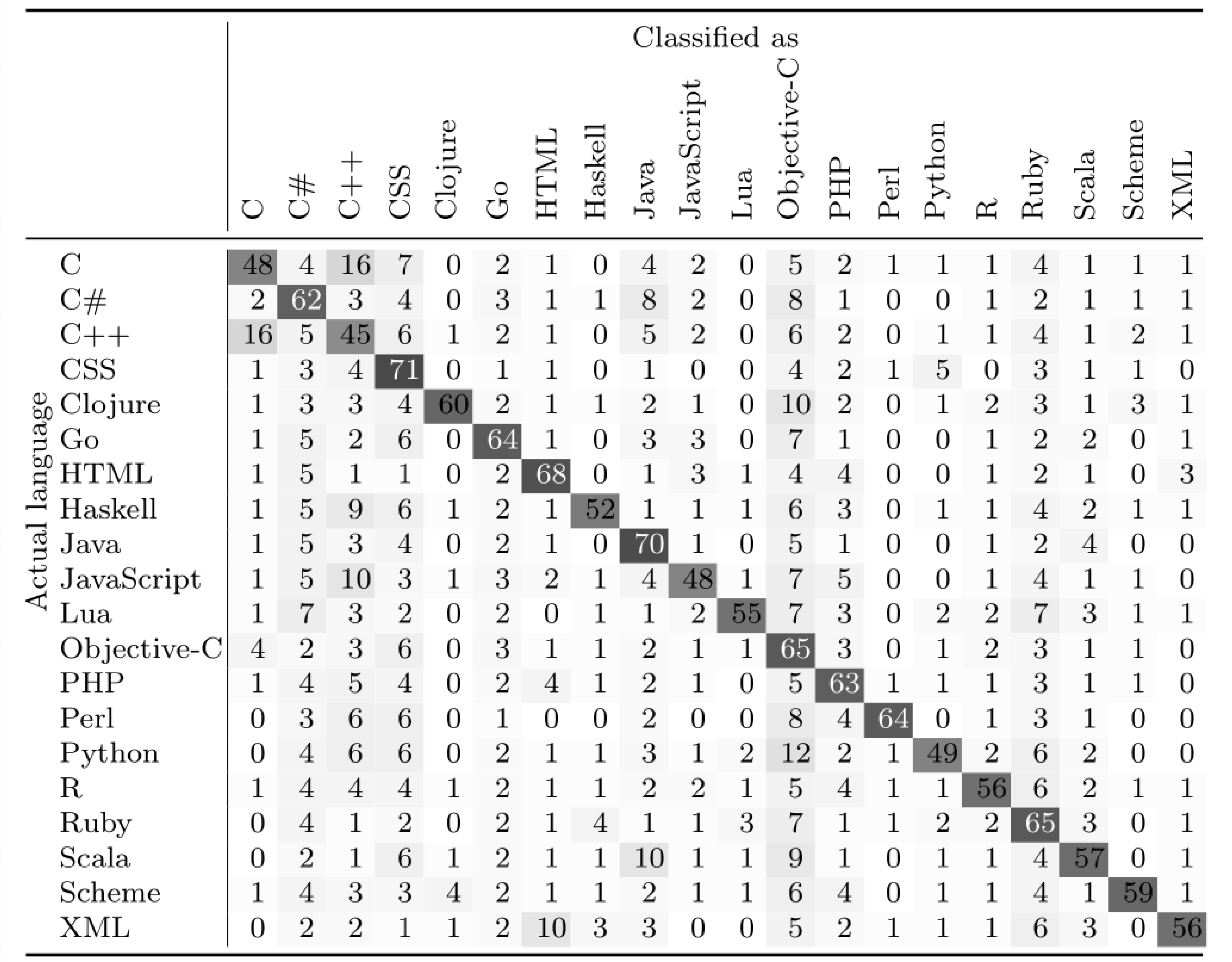
p.624-628


TALKING ABOUT THE PAPER CONTENTS?
AIN'T NOBODY GOT TIME FOR THAT!
imgflip.com

Natural Language Classifiers

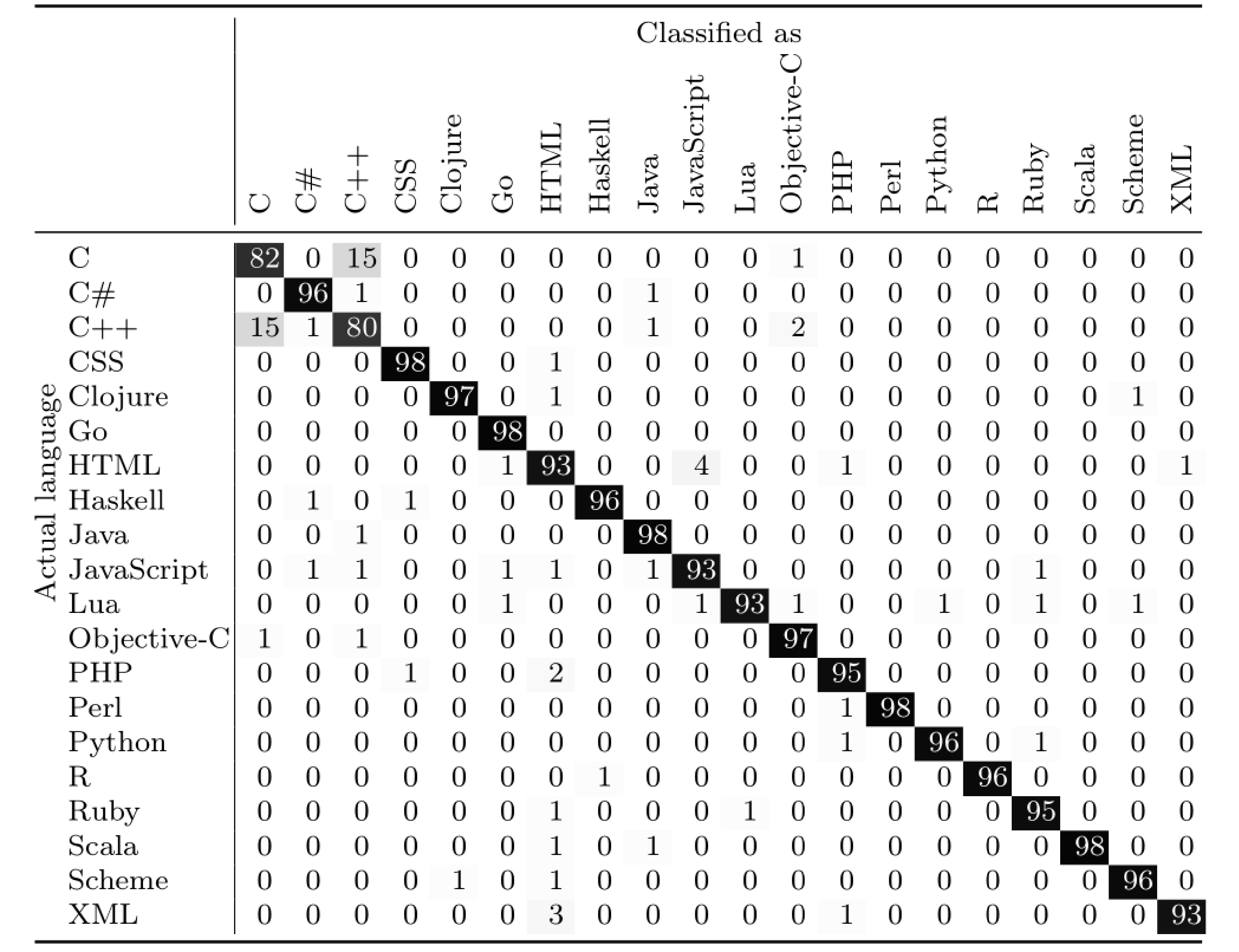
- Features?
- Multinomial Naïve Bayes
  - fight naïveté with n-grams & skip-grams
    - smooth zero-probability n-grams
    - skip or hack unknown tokens
  - implement with SRILM
- Vary n, tokenisers, smoothers, …
- Train on 10000 files of 5-10kb in 20 languages
  - C, C++, C#, Closure, CSS, Go, Haskell, HTML, Java, JS, Lua, ObjC, Perl, PHP, Python, R, Ruby, Scala, Scheme, XML
- Run experiment on 200 files in 20 languages
- Classify lines in mixed files

Natural Language Classifiers



		Classified as																			
		C	C#	C++	CSS	Clojure	Go	HTML	Haskell	Java	JavaScript	Lua	Objective-C	PHP	Perl	Python	R	Ruby	Scala	Scheme	XML
Actual language	C	48	4	16	7	0	2	1	0	4	2	0	5	2	1	1	1	4	1	1	1
	C#	2	62	3	4	0	3	1	1	8	2	0	8	1	0	0	1	2	1	1	1
	C++	16	5	45	6	1	2	1	0	5	2	0	6	2	0	1	1	4	1	2	1
	CSS	1	3	4	71	0	1	1	0	1	0	0	4	2	1	5	0	3	1	1	0
	Clojure	1	3	3	4	60	2	1	1	2	1	0	10	2	0	1	2	3	1	3	1
	Go	1	5	2	6	0	64	1	0	3	3	0	7	1	0	0	1	2	2	0	1
	HTML	1	5	1	1	0	2	68	0	1	3	1	4	4	0	0	1	2	1	0	3
	Haskell	1	5	9	6	1	2	1	52	1	1	1	6	3	0	1	1	4	2	1	1
	Java	1	5	3	4	0	2	1	0	70	1	0	5	1	0	0	1	2	4	0	0
	JavaScript	1	5	10	3	1	3	2	1	4	48	1	7	5	0	0	1	4	1	1	0
	Lua	1	7	3	2	0	2	0	1	1	2	55	7	3	0	2	2	7	3	1	1
	Objective-C	4	2	3	6	0	3	1	1	2	1	1	65	3	0	1	2	3	1	1	0
	PHP	1	4	5	4	0	2	4	1	2	1	0	5	63	1	1	1	3	1	1	0
	Perl	0	3	6	6	0	1	0	0	2	0	0	8	4	64	0	1	3	1	0	0
	Python	0	4	6	6	0	2	1	1	3	1	2	12	2	1	49	2	6	2	0	0
	R	1	4	4	4	1	2	1	1	2	2	1	5	4	1	1	56	6	2	1	1
	Ruby	0	4	1	2	0	2	1	4	1	1	3	7	1	1	2	2	65	3	0	1
	Scala	0	2	1	6	1	2	1	1	10	1	1	9	1	0	1	1	4	57	0	1
	Scheme	1	4	3	3	4	2	1	1	2	1	1	6	4	0	1	1	4	1	59	1
	XML	0	2	2	1	1	2	10	3	3	0	0	5	2	1	1	1	6	3	0	56

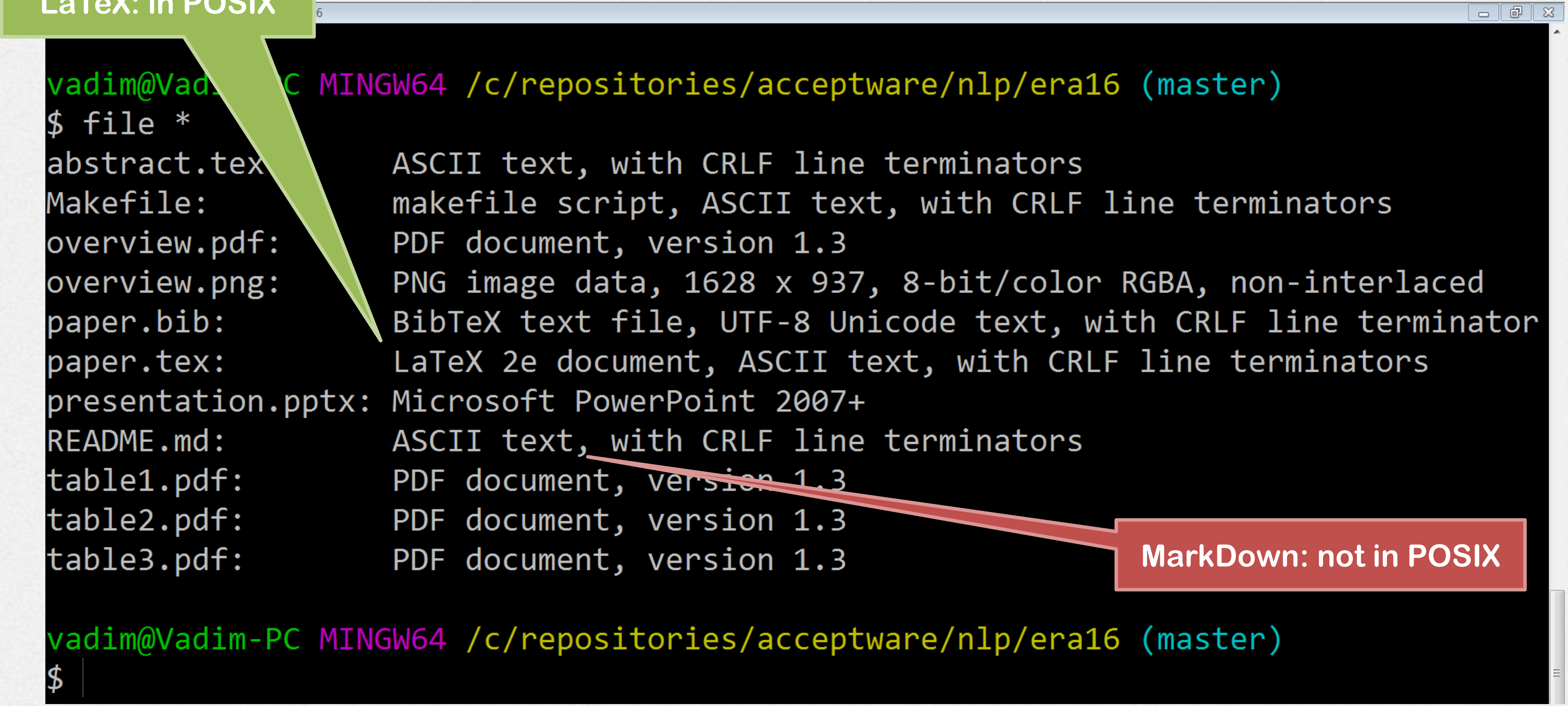
Natural Language Classifiers



		Classified as																			
		C	C#	C++	CSS	Clojure	Go	HTML	Haskell	Java	JavaScript	Lua	Objective-C	PHP	Perl	Python	R	Ruby	Scala	Scheme	XML
Actual language	C	82	0	15	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	C#	0	96	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	C++	15	1	80	0	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0
	CSS	0	0	0	98	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Clojure	0	0	0	0	97	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
	Go	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	HTML	0	0	0	0	0	1	93	0	0	4	0	0	1	0	0	0	0	0	0	1
	Haskell	0	1	0	1	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0
	Java	0	0	1	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0	0	0
	JavaScript	0	1	1	0	0	1	1	0	1	93	0	0	0	0	0	0	1	0	0	0
	Lua	0	0	0	0	0	1	0	0	0	1	93	1	0	0	1	0	1	0	1	0
	Objective-C	1	0	1	0	0	0	0	0	0	0	0	97	0	0	0	0	0	0	0	0
	PHP	0	0	0	1	0	0	2	0	0	0	0	0	95	0	0	0	0	0	0	0
	Perl	0	0	0	0	0	0	0	0	0	0	0	0	1	98	0	0	0	0	0	0
	Python	0	0	0	0	0	0	0	0	0	0	0	0	1	0	96	0	1	0	0	0
	R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	96	0	0	0	0
	Ruby	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	95	0	0	0
	Scala	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	98	0	0
	Scheme	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	96	0
	XML	0	0	0	0	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	93

file

LaTeX: in POSIX



```
vadim@Vadim-PC MINGW64 /c/repositories/acceptware/nlp/era16 (master)
$ file *
abstract.tex:       ASCII text, with CRLF line terminators
Makefile:           makefile script, ASCII text, with CRLF line terminators
overview.pdf:       PDF document, version 1.3
overview.png:       PNG image data, 1628 x 937, 8-bit/color RGBA, non-interlaced
paper.bib:          BibTeX text file, UTF-8 Unicode text, with CRLF line terminator
paper.tex:          LaTeX 2e document, ASCII text, with CRLF line terminators
presentation.pptx: Microsoft PowerPoint 2007+
README.md:          ASCII text, with CRLF line terminators
table1.pdf:         PDF document, version 1.3
table2.pdf:         PDF document, version 1.3
table3.pdf:         PDF document, version 1.3

vadim@Vadim-PC MINGW64 /c/repositories/acceptware/nlp/era16 (master)
$
```

MarkDown: not in POSIX

IN UR UNIX SINCE 1973
IDENTIFYING UR FILES

COBOL, PL/I, REXX, JCL, SQL
ASCII text, with CRLF line terminators

State Why the Problem is a Problem

- **Legacy IT Portfolio Assessment**
  - 10k+ files
  - 50M+ LOC
  - names like “WMGCLP9M”
  - no extensions
- **State of the art**
  - file, grep, …
  - vocabulary
  - parsing/resolution attempts
- **SLI can provide clustering**

#TODO #afterSANER16

- Find falsely classified files by
  - file and other lexical classification programs
  - GitHub, SearchCode and other services
  - SyntaxHighlighter.js and other libraries
  - looking at reactions to polyglots & quines
- Validate by realistic case studies
  - COBOL+PL/I+REXX+JCL+…
  - HTML+JS+CSS+PHP+…
- Compare to related methods of telling code from non-code
- Explain the lack of evidence for language families
  - and all other peculiarities

Software Language Identification

- SLI is a thing.
  - basic fact extractor
  - not all methods work
- Practice is full with imperfection
  - deal with it
- Swimming with the ~~data~~ code
  - unstructured ~~data~~ code
- Read the paper.
- Follow @grammarware!

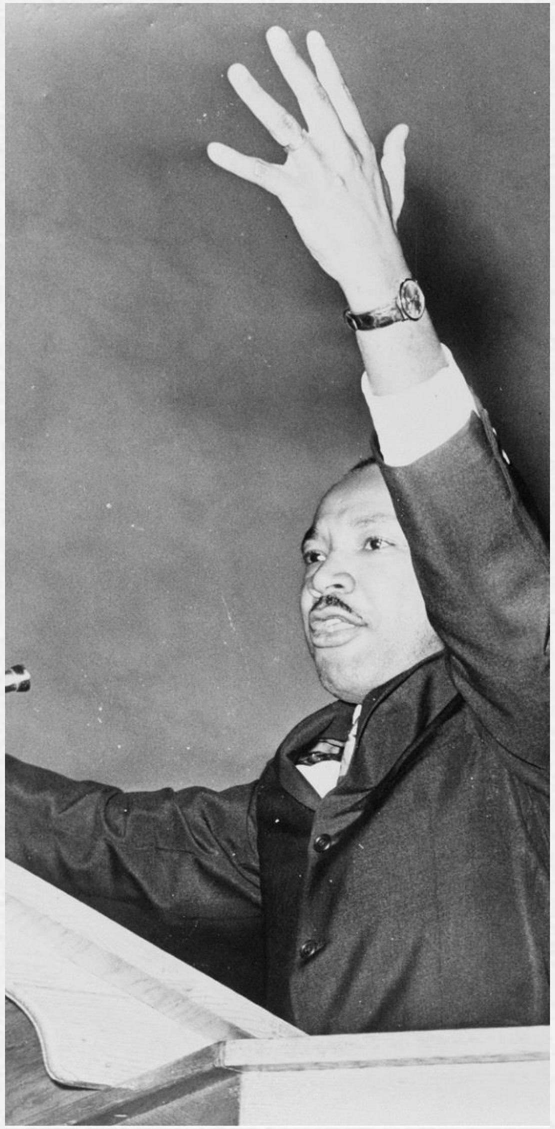


Photo in public domain, source: http://loc.gov/pictures/resource/cph.3c11157/