Linting science prose and the science of prose linting

Michael D. Pacer[‡*], Jordan W. Suchow[‡]

https://youtu.be/S55EFUOu4O0

Abstract—The craft of writing is hard despite the abundance of thoughtful advice available in usage guides and other sources. This is partly a problem of medium: amassing advice is not enough to improve writing. Writing would thus benefit if our collective knowledge about best practices in writing were extracted and transformed into a medium that makes the knowledge more accessible to authors.

We built Proselint, a Python-based linter for English prose that identifies violations of style and usage guidelines. Proselint is open-source software released under the BSD license and is compatible with Pythons 2 and 3. It runs as a command-line utility or as a text-editor plugin. Proselint's modules address redundancy, jargon, illogic, clichés, unidiomatic vocabulary, sexism, inconsistency, misuse of symbols, malapropisms, oxymorons, security gaffes, hedging, apologizing, and pretension. Furthermore, Proselint is extensible, enabling creation of domain-specific modules and implementation of house style guides.

Proselint can be seen as both a language tool for scientists and a tool for language science. On the one hand, Proselint can help scientists communicate their ideas to each other and to the public by improving their writing. On the other hand, scientists can use Proselint to measure language usage, to provide style- and usage-based features for tasks such as authorship identification, and to explore the factors that make a linter useful (e.g., a low false discovery rate).

Index Terms—linters, writing tools, copyediting

The problem

Writing is hard even for the best writers, and it's not for lack of good advice — a tremendous amount of knowledge about the craft is strewn across usage guides, dictionaries, technical manuals, essays, pamphlets, websites, and the hearts and minds of great authors and editors. Consider *Garner's Modern English Usage*, an authoritative usage guide with 11,000 entries covering a broad range of advice that can help writers produce clear and idiomatic prose [Gar16]. Or consider the *Federal Plain Language Guidelines*, a guide created by employees of the U.S. federal government to promote writing that is clear, concise, and well-organized [Pla11]. Professional conferences such as the annual meeting of the American Copy Editors Society are dedicated to sharing knowledge about editing prose. And within the academy, organizations such as the American Psychological Association publish manuals whose guidance on style has been adopted as a standard [Ass94].

* *Corresponding author: mpacer@berkeley.edu*
‡ *University of California, Berkeley*

Copyright © 2016 Michael D. Pacer et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Advice on writing touches upon everything from superficial conventions to the deepest reflections of our society and its attitudes. For example, advice concerning the preferred forms of words such as *connote* (vs. *connotate*) may help to prune needless variants in spelling, but is unlikely to affect the reader's understanding of the text and its author. In contrast, advice concerning needlessly gendered language (*woman scientist*, *policeman*) helps to eliminate terms that may perpetuate social inequality [MS01], [Phi04].

Amassing a pile of advice is not enough to make writing better. This is because advice, though it may be principled, thoughtful, and worth following, is hard to apply in new settings once it has been learned [AI00]. Thus even if an author could absorb all the knowledge contained in extant sources of advice on writing, the author would still face the problem of recalling and systematically applying that knowledge during the acts of writing and editing. Furthermore, developing a new habit (linguistic or otherwise) is slow, costly, and effortful [FH10], causing errors to appear even if the author knows the rules.

Today, an author who wishes to improve a piece of writing by applying the collective wisdom of experts must rely on indirect means. Publishers often use a division of labor in which dedicated staff copyedit a piece to their satisfaction. For example, *The New Yorker* employs an editing team of fact checkers, editors, grammarians, and others [Nor]. Individuals often uses software-based tools such as spelling and grammar checkers that mark unrecognized words and purported violations of grammatical rules [HJM+82], [CM83], [Ver00], [Nab03], [Mił10], [PRR12].

Neither approach fully solves the problem of successful adoption of best practices in writing. Few people have the resources needed to outsource editing to external staff. Furthermore, doing so inevitably introduces a delay because copy editors must read the text carefully and are normally unavailable during the act of writing. By the time an editor's notes are received, then, an opportunity to strengthen the writer's craft has passed. Time-sensitivity exacerbates this problem because delays introduced by the editing process may diminish the communication's value. In contrast, software-based tools for writing are automated and relatively fast, but are typically incomplete, imprecise, or inaccessible (see *Proselint's approach*).

The solution

To solve this problem, we built Proselint, a real-time linter for English prose. A linter is a computer program that, like a spell checker, scans through a document and analyzes it, identifying problems with its syntax or style [Joh77]. Proselint identifies violations of expert-endorsed style and usage guidelines[1] and gently

alerts the writer of those violations as they are committed, an ideal opportunity to elicit long-term changes in behavior [FS57]. In doing so, Proselint gives voice to the experts while teaching at a speed and scale unreachable by humans.

Proselint is open-source software released under the BSD license and compatible with Pythons 2 and 3. It runs as a command-line utility or editor plugin for Sublime Text, Atom, Emacs, vim, etc. It outputs advice in JSON and the standard linting format (SLF), promoting integration with external services [Was90] and providing human-readable output. Proselint includes modules on a variety of usage problems, including redundancy, jargon, illogic, clichés, sexism, misspelling, inconsistency, misuse of symbols, malapropisms, oxymorons, security gaffes, hedging, apologizing, pretension, and more (see Tables 1 and 2 for a fuller listing).

Proselint is both a language tool for scientists and a tool for language science. On the one hand, it can help scientists communicate their ideas to each other and to the public by improving their writing. On the other hand, scientists can use Proselint to study language and linting.

A language tool for scientists

Scientists use the written word to communicate to each other and to the public. Proselint improves writing across a number of dimensions relevant to science communication, including consistency in terminology & typography, concision, and elimination of redundancy. For example, Proselint detects the letter x used in place of the multiplication symbol × (e.g., 1440 x 900), misspecified p values resulting from data-analysis software that truncates small numbers (e.g., $p = 0.00$), and colloquialisms that obscure the mechanisms of science-based technology (e.g., "lie detector test" for the polygraph machine, which measures arousal, not lying per se).

A tool for language science

Linguistics is largely descriptivist, tending to describe language as it is used rather than prescribe how it ought to be used [Gar16]. Errors are considered mostly in the context of language learning (especially children's) because those errors reveal the structure of the language-learning mechanism (see, e.g., overregularization by young English speakers [MPU+92]). Though linting prose is implicitly prescriptivist because its detection of norm violations presupposes the existence of norms [Gar16], even so, language science can benefit from Proselint's advice without making normative claims. Linguists can use Proselint to detect patterns in usage and style in corpora of written text, to identify authors by their usage, and to enrich standard Natural Language Processing (NLP) techniques with features beyond word frequencies and syntactic structures [BKL09].

The advice

Proselint is built around advice derived from works by Bryan Garner, David Foster Wallace, Chuck Palahniuk, Steve Pinker, Mary Norris, Mark Twain, Elmore Leonard, George Orwell, Matthew Butterick, William Strunk, E.B. White, Philip Corbett, Ernest Gowers, and the editorial staff of the world's finest literary magazines and newspapers, among others.[2]

Our standard for including a new rule is that it should be accompanied by a citation to a recognized expert on language usage who has defined the rule clearly. Though we have no explicit criteria for what makes a citation appropriate, in practice we have given greater weight to works from well-established publishers and those widely cited as reliable sources of advice. The choice of which rules to implement is ultimately a question of feasibility of implementation, utility, and preference. Our guiding preference is to make Proselint widely useful by default. In the case of unresolved conflicts between advice from multiple sources, our default is to exclude all forms of the advice because we find it unreasonable to hold users to a higher standard than we hold the experts, at least one of whom supports the user's choice. Because we aim for excellent defaults without hampering customization, Proselint can be extended by adding new rules or filtered by excluding existing rules through a configuration file.

Tables 1 and 2 list much of the advice that Proselint currently implements. That advice is organized into modules.

Rule modules

Proselint's rules are organized into modules that reflect the structure of usage guides [Gar16]. For example, the `terms` module encourages expressive vocabulary by flagging use of unidiomatic and generic terms. The module has submodules for categories of terms found as entries in usage guides. The submodule `terms.venery` pertains to venery terms, which arose from hunting tradition and describe groups of animals of a particular species — a *pride* of lions or an *unkindness* of ravens. Similarly, the submodule `terms.denizen_labels` pertains to demonyms, which are used to describe people from a particular place — *New Yorkers* (New York), *Mancunians* (Manchester), or *Novocastrians* (Newcastle).

Organizing rules into modules is useful for two reasons. First, it allows for a logical grouping of similar rules, which often require similar computational machinery to implement. Second, it allows users to include and exclude rules at a higher level of abstraction than the individual word or phrase.

Converting a rule to code: rule templates

Suppose a developer wanted to implement the following entry from *Garner's Modern English Usage* as a rule in Proselint:

> **decimate.** Originally this word meant "to kill one in every ten," but this etymological sense, because it's so uncommon, has been abandoned except in historical contexts. Now *decimate* generally means "to cause great loss of life; to destroy a large part of." ... In fact, though, the word might justifiably be considered a SKUNKED TERM. Whether you stick to the original one-in-ten meaning or use the extended sense, the word is infected with ambiguity. And some of your readers will probably be puzzled or bothered. [Gar16]

In general, a rule's implementation need only be a function that takes in a string of text, applies logic identifying whether the rule has been violated, and then returns a value identifying the violation in the correct format. Weak requirements and Python's

1. Proselint differs from a spell-checker in that its recommendations do not specifically counter spelling errors, but rather errors of style and usage. The two occasionally overlap, e.g. in the malapropism "attacking your voracity", where it is not that "voracity" is a spelling error per se but that the appropriate word is its phonetic neighbor "veracity". Compare this to "attacking your verqcity", almost certainly a typo.

2. Proselint has not been endorsed by these individuals; we have merely implemented their words in code.

ID	Description
airlinese.misc	Avoiding jargon of the airline industry
annotations.misc	Catching annotations left in the text
archaism.misc	Avoiding archaic forms
cliches.misc	Avoiding clichés
consistency.spacing	Consistent sentence spacing
consistency.spelling	Consistent spelling
corporate_speak.misc	Avoiding corporate buzzwords
cursing.filth	Avoiding cursing
cursing.nfl	Avoiding words banned by the NFL
dates_times.am_pm	Using the right form for time
dates_times.dates	Stylish formatting of dates
hedging.misc	Not hedging
hyperbole.misc	Not being hyperbolic
jargon.misc	Avoiding miscellaneous jargon
lexical_illusions.misc	Avoiding lexical illusions
links.broken	Linking only to existing sites
malapropisms.misc	Avoiding common malapropisms
misc.apologizing	Being confident
misc.back_formations	Avoiding needless backformations
misc.bureaucratese	Avoiding bureaucratese
misc.but	Avoiding starting a par. with "But..."
misc.capitalization	Capitalizing correctly
misc.chatspeak	Avoiding lolling and other chatspeak
misc.commercialese	Avoiding commerical jargon
misc.currency	Avoiding redundant currency symbols
misc.debased	Avoiding debased language
misc.false_plurals	Avoiding false plurals
misc.illogic	Avoiding illogical forms
misc.inferior_superior	Superior to, not than
misc.latin	Avoiding overuse of Latin phrases
misc.many_a	Many a singular
misc.metaconcepts	Avoiding overuse of metaconcepts
misc.narcisissm	Talking about the subject, not its study
misc.phrasal_adjectives	Hyphenating phrasal adjectives
misc.preferred_forms	Miscellaneous preferred forms

TABLE 1
What Proselint checks.

ID	Description
misc.pretension	Avoiding being pretentious
misc.professions	Calling jobs by the right name
misc.punctuation	Using punctuation assiduously
misc.scare_quotes	Using scare quotes only when needed
misc.suddenly	Avoiding the word suddenly
misc.waxed	Waxing poetic
misc.whence	Using "whence"
mixed_metaphors.misc	Not mixing metaphors
mondegreens.misc	Avoiding mondegreens
needless_variants.misc	Using the preferred form
nonwords.misc	Avoid using nonwords
oxymorons.misc	Avoiding oxymorons
psychology.misc	Avoiding misused psychological terms
redundancy.misc	Avoid redundancy & saying things twice
redundancy.ras_syndrome	Avoiding RAS syndrome
skunked_terms.misc	Avoid using skunked terms
spelling.able_atable	-able vs. -atable
spelling.able_ible	-able vs. -ible
spelling.athletes	Spelling of athlete names
spelling.em_im_en_in	-em vs. -im and -en vs. -in
spelling.er_or	-er vs. -or
spelling.in_un	in- vs. un-
spelling.misc	Spelling words corectly
security.credit_card	Keeping credit card numbers secret
security.password	Keeping passwords secret
sexism.misc	Avoiding sexist language
terms.animal_adjectives	Animal adjectives
terms.denizen_labels	Calling denizens by the right name
terms.eponymous_adjs	Calling people by the right name
terms.venery	Call groups of animals by the right name
typography.diacritics	Using dïacríticâl marks
typography.exclamation	Avoiding overuse of exclamation
typography.symbols	Using the right symbols
uncomparables.misc	Not comparing uncomparables
weasel_words.misc	Avoiding weasel words

TABLE 2
What Proselint checks (cont.).

expressiveness allow developers to build detectors for all computable usage and style requirements, but provide little guidance for implementing new rules.

To provide guidance for implementing new rules, we wrote helper functions that follow the protocol and provide some common logical forms of rules. These include checking for the existence of a given word, phrase, or pattern (`existence_check()`); for intra-document consistency in usage (`consistency_check()`); and for use of a word's preferred form (`preferred_forms_check()`).

The entry on *decimate* bans a word and so can be implemented using the `existence_check` template:

```
def check_for_decimate(text):
    err = "skunked_terms.decimate"
    msg = (u"'{}' is a skunked term -- impossible to
           "use without someone taking issue. Find"
           "another way to say it")
    regex = "decimat(?:e|es|ed|ing)?"
    return existence_check(
        text, [regex], err, msg, join=True)
```

First the function defines an error code, an error message, and a regular expression that matches the word *decimate* in its various forms. Then it applies the existence check.

Using Proselint

Installation

Proselint is available on the Python Package Index and can be installed using pip:

```
pip install proselint
```

Alternatively, developers can retrieve the Git repository from GitHub (https://github.com/amperser/Proselint) and then install the software using setuptools:

```
pip install --editable
```

Command-line utility

Proselint is a command-line utility that reads in a text file:

```
proselint text.md
```

Running this command prints a list of suggestions to stdout, one per line. The GNU Error Message Formatting standard [S+16] is the basis for the format of displaying these suggestions. We further require that the error code (here, the `check_name`) is separated from the error message by a space. Because this format is used by many linters, we call it the Standard Linting Format (SLF). An SLF-formatted suggestion has the form:

```
text.md:<line>:<column>: <check_name> <message>
```

For example,

```
text.md:0:10: skunked_terms.misc 'decimate' is ...
a skunked term -- impossible to use without ...
someone taking issue. Find another way to say it."
```

This message suggests that, at column 10 of line 0, the module `skunked_terms.misc` detected the presence of the skunked term *decimate*. The command-line utility can instead print the list of suggestions in JSON through the `--json` flag. In this case, the output is considerably richer:

```
{
    // The check originating this suggestion
    "check": "uncomparables.misc",

    // The line where the error starts
    "line": 1,

    //The column where the error starts
    "column": 1,

    // Index in the text where the error starts
    "start": 1,

    // the index in the text where the error ends
    "end": 18,

    // start - end
    "extent": 17,

    // Message describing the advice
    "message": "Comparison of an uncomparable: ...
    'very unique\n' is not comparable.",

    // Possible replacements
    "replacements": null,

    // Importance("suggestion", "warning", "error")
    "severity": "warning"
}
```

Text editor plugins

Proselint is available as a plugin for popular text editors, including Emacs, vim, Sublime Text, and Atom. Embedding linters within the tools that people already use to write removes a barrier to adoption the linter and thereby promotes adoption of best practices in writing [Was90].

Proselint's approach

In the following sections, we describe Proselint's approach and its greatest points of departure from previous attempts to lint prose. As part of this analysis, we curated a list of known tools for automated language checking. The dataset contains the name of each tool, a link to its website, and data about its basic features, including languages and licenses (link). The tools are varied in their approaches and coverage, but typically focus on grammar versus usage and style; are unsystematic in choosing sources of advice; or have been abandoned. In general, we regard the tools as being imprecise, incomplete, and inaccessible:

Imprecise. Even the best software-based tools for editing are riddled with false positives. We evaluated many of the tools in our dataset on an earlier version of the corpus. Proselint's false discovery rate of 1 false positive to 10 true positives was 20× better than the next best tool, Microsoft Word, which had a false discovery rate of 2 false positives to 1 true positive.

Incomplete. All software-based tools for editing are incomplete; not one frees our collective knowledge about best practices in writing from its bindings. Completion is likely an unattainable goal, which inspires Proselint's open-source, community-participation model.

Inaccessible. Many existing tools are inaccessible because they cost money, are closed source, or are inextensible. Thus we designed Proselint to be free, open source, and extensible.

What to check: usage, not grammar

Proselint does not detect grammatical errors because it is both too easy and too hard:

Detecting grammatical errors is too easy in the sense that most native speakers can readily identify and easily fix them. The errors that leave the greatest negative impression in the reader's mind are often glaring to native speaker. On the other hand, more subtle errors, such as a disagreement in number set apart by a long string of intermediary text, escapes even a native speaker's notice.

Detecting grammatical errors is too hard in the sense that its most general form is AI-hard, requiring at least human-level artificial intelligence and a native speaker's ear [Yam13]. Modern NLP techniques that detect grammatical errors are unavoidably statistical and produce many false positives [BKL09] [LCGT10]. This is in part because syntax parsers used in grammatical error detection must tolerate grammatical errors, a problem that is compounded in writing by English-language learners [LCGT10]. Once a grammatical error has been detected, determining the correct replacement hinges on the intended meaning. Occasionally, the intended meaning will determine even *whether* a grammatical error is present: e.g., is "Man bites dog" a headline about canine aggression, or are the subject and object swapped in error? In the general case, the problem of determining the intended meaning of a sentence is AI-hard [Yam13].

Instead of focusing on grammatical errors, Proselint addresses errors of usage and style.

Published expertise as primary sources

People have such strong shared intuitions about grammar that a common experimental measure in linguistics is the grammaticality of a sentence as measured by the intuitions of native speakers [Kel00]. But style and usage inspire a multitude of intuitions. Authors of usage guides have done much of the work of hashing out these conflicting intuitions to arrive at sensible everyday advice [Gar16]. Proselint thus defers to these experts, and in doing so embodies our collective understanding about the craft of writing with style.

Levels of difficulty

In a loose analogy to Chomsky's hierarchy of formal grammars [Cho56], usage errors vary in the difficulty of detecting and correcting them:

1) AI-hard

2) NLP, beyond state-of-the-art
3) NLP, state-of-the-art
4) Syntax-dependent rules
5) Regular expressions
6) One-to-one replacement rules.

At the lowest levels of the hierarchy are usage errors that a linter can reliably detect and correct through one-to-one replacement rules. At the highest levels are usage errors whose detection and correction are such hard computational problems that it would require at least human-level intelligence to solve in the general case, if a solution is possible at all [Yam13]. Consider usage errors pertaining to placement of the word *only*, which depends on the intended meaning. For example, in "John hit Peter in his only nose", is the *only* misplaced or is it unusual that Peter has only one nose? Usage errors at this highest level of the hierarchy are hard to detect without introducing false positives and determining the correct replacement requires understanding the intended meaning. Development of Proselint begins at the lowest levels of the hierarchy and builds upwards.

Signal detection theory and the lintscore

Any new tool, for language or otherwise, faces a challenge to its adoption: it must demonstrate that the utility the tool provides outweighs the cost of learning to use it [Was90]. The utility of a prose linter comes in part from its ability to detect usage and style errors. Each issue flagged might be an error, but it might instead be a false positive. Let T be the number of true errors and F be the number of false positives, thus making $T+F$ the total number of flags raised by the tool. An approach that attempts to maximize T by flagging many errors without adequately considering F will identify many genuine errors, but raise so many false positives that writers must evaluate each proposed error.

With Proselint, we aim for a tool precise enough that users can adopt its recommendations unquestioningly and still come out ahead. To achieve this, we penalize the number of false positives F by evaluating Proselint in terms of its *empirical lintscore*. The lintscore gives one point for every true positive T and penalizes on the basis of the false discovery rate $\alpha = \frac{F}{T+F}$. The lintscore is given by

$$l(T,F;k) = T(1-\alpha)^k,$$

where the parameter :math:kgeq1‘ controls the strength of the $1-\alpha$ penalty. Notably, the lintscore does not reflect the number of true and false negatives; we reason that it is more important to be quiet and authoritative than to be loud and risk being untrustworthy (cf. the metrics discussed in [CDIT12]).

The lintscore can be computed exactly if an evaluator can classify each error flagged by the linter as a true or false positive. However, many corpora are large enough to preclude this kind of exhaustive assessment. In these cases, the lintscore can be estimated from the total number of issues flagged and an estimate of the false discovery rate.

Note that the lintscore is not a readability metric because it evaluates linters, not prose. Given a set of documents, signal detection theory makes it possible to estimate a linters' trustworthiness through the lintscore.

Speed via Memoization

Proselint must be efficient for use as a real-time linter. Avoiding redundant computation by storing the results of expensive function calls ("memoization") improves efficiency. Because most paragraphs do not change from moment to moment during editing of a sizable document, memoizing Proselint's output over paragraphs and recomputing only when a paragraph has changed (otherwise returning the memoized result) reduces the total amount of computation and thus improves the running time.

A proof of concept

As a proof of concept, we used Proselint to make contributions to several documents. These include the White House's Federal Source Code Policy; The Open Logic Project textbook on advanced logic; Infoactive's Data + Design book; and many of the other papers submitted to SciPy 2016. In addition, we evaluated Proselint's false discovery rate on a corpus of essays from well-edited magazines such as *Harper's Magazine*, *The New Yorker*, and *The Atlantic* (full list). We then measured the lintscore. Because the essays included in our corpus were edited by a team of experts, we expect Proselint to remain mostly silent. By design, Proselint should comment only on the rare error that slips through unnoticed by the editors or, more commonly, on finer points of usage, about which the experts sometimes disagree. When run over v0.1.0 of our corpus, we achieved a lintscore (k = 2) of 98.8.

Future development and possible applications

We see a number of directions for future development of Proselint that improve the tool and its utility for science:

Context-sensitive rule application and machine learning

Many rules apply better to some kinds of documents than to others. For example, in most cases *extendable* is preferable to *extensible*, but in software development the opposite is true. Applying these rules without consideration of the context will systematically introduce false positives.

Silencing rules that are predicted to be irrelevant because of the context allows a greater variety of rules to be included without introducing false positives. Consider the advice that, when specifying a decade, an apostrophe is unnecessary: Eisenhower was president in the 50s, not the 50's. However, not all instances of *50's* are problematic: one can validly write *50's manager* to refer to 50's manager without making a usage error about decades. To account for this context sensitivity, Proselint detects whether a document's topic is 50 Cent, identifying *50's* as a usage error only when the topic is not detected.

The 50 Cent topic detector was hand-crafted in the fashion of expert knowledge systems [Jac86]. Machine-learning techniques for identifying the topic of a document (e.g., topic models [BL09]) can generalize this ability and will be crucial to safely growing Proselint's coverage of usage errors. Once incorporated, extending this to hierarchical nonparametric topic models will enable document sub-structure to be taken into account as a form of context [BGJ10].

Evaluating linters by testing on multiple corpora

In our internal evaluations of Proselint, we calculate the empirical lintscore manually on a corpus of professionally edited documents, which presumably have few errors. This efficiently alerts us to false positives that are introduced by new rules, but tells us little about its performance in other settings. A major improvement would be to compute the lintscore on corpora such as student essays, which are more likely to have true positives and will thus

improve our estimates of Proselint's positive utility for a more typical user.

Corpora of documents drawn from different content-based categories (technical papers, scientific articles, software documentation, fiction, journalism, etc.) will help in evaluating Proselint's performance in evaluating prose from different fields. Certain rules may be relevant to some fields more than others and testing with diverse corpora will ensure that Proselint can be used by a diverse range of individuals. Furthermore, this will allow us to learn which rule sets are relevant in which contexts.

Observing how a document is modified in accordance with Proselint's suggestions affords new opportunities for evaluation of Proselint, tracking the acceptance of its advice and any effects on the rate of new errors introduced between drafts.

File formats and markup languages for documents (e.g, reStructuredText, LaTeX, Markdown, HTML, etc.) often rely on syntactical conventions that Proselint falsely identifies as errors. Similar concerns arise for documentation written as docstrings or code comments in a variety of programming languages. Corpora focusing on individual formats and languages will aid in identifying and filtering these errors, enabling development targeted at addressing these problems.

Stylometrics and machine learning

The field of stylometrics has extensively studied the problem of identifying the authors of documents [ZLCH06]. Many of these studies focus on the relative frequencies with which individual words are used, especially function words. For example, Mosteller & Wallace inferred the authorship of twelve essays in the *Federalist Papers* on the basis of the frequency of common function words such as *to* and *by* [MW63]. Proselint provides new measures that could be used to improve this kind of stylometric analysis.

Several applications follow from authorship identification:

One application uses Proselint to detect ghost-written documents, which could also have benefits for identifying academic dishonesty (e.g., purchasing and selling of ghost-written essays). This application assumes that there is a ground-truth corpus with samples of the author's writing. On the other hand, someone may be able to use Proselint to *escape* identification by avoiding features that distinguish the author's writing from those of others.

A second application inverts and generalizes the process of identifying authors by selectively introducing, changing, or removing usage choices to obfuscate or encrypt messages. With some modifications and a protocol for establishing usage-based keys, Proselint could become a system for designing content-aware steganographic systems that convey hidden messages through their choice of words and style [BK06]. Encryption would require modifying the Proselint infrastructure to identify when more than one acceptable choice exists.

The errors Proselint can detect are rare compared to the typical linguistic features used in stylometry [ZLCH06], [MW63], [Rud97]. Sparse measures pose difficulty for methods like those in Mosteller & Wallace (1963) [MW63]. Machine-learning techniques for inferring identity from sparse data will thus be particularly applicable. Furthermore, this endeavor will benefit from an approach that considers the cross product of authors and topics [RZGSS04].

Automated usage and style metrics

Readability metrics such as the Flesch–Kincaid Grade Level and the Gunning fog index do not capture usage and style because they measure reading ease rather than conventionality [Fle48]. Proselint could be used to create automated metrics for the consistency and stylishness of prose. Such metrics may also find use as part of automated essay-grading tools [VNC03].

Tracking historical trends in usage

An application of Proselint as a tool for language science is in tracking historical trends in usage. Corpora such as Google Books have been useful for measuring changes in the prevalence of words and phrases over several hundred years [MSA+11]. Our tool can be used in a similar way because it provides a feature set for usage. For example, one might study the prevalence of airlinese (including, e.g., use of "momentarily" to mean "in a moment", as in the phrase "we are taking off momentarily") and its alignment with the rise of that industry.

An unsolved problem: foreign languages

We have no immediate plans for extending Proselint to other languages. This is in part because building a linter for style and usage errors in both American and British English is challenging enough for a native speaker, and in part because attempting to build a linter for languages in which the creators lack fluency would seem to be an exercise in folly. An open problem is how to extend Proselint to become a universal linter for prose.

Missing corpora

To evaluate Proselint's false discovery rate, we built a corpus of text from well-edited magazines believed to contain low rates of usage errors. In the course of assembling this corpus, we discovered a lack of annotated corpora that provide false discovery rates for style and usage violations[3]. The Proselint testing framework is an excellent opportunity to develop such a corpus. Unfortunately, because our current corpus derives from copyrighted work, it cannot be released as part of open-source software. Developing an open-source corpus of style and usage errors will be necessary if these tools are to be made available for NLP research outside internal testing of Proselint.

A critique of normativity in prose styling, and a response

One critique of Proselint [hac] is a concern that introducing any kind of linter-like process to the act of writing diminishes the ability for authors to express themselves creatively. These arguments suggest that authors will find themselves limited by the linter's rules and that, as a result, this will have a shaping or homogenizing effect on language.

In response to this critique, we note that our goal is not to homogenize text for the sake of uniformity (though perhaps there is value there, too), but rather to detect instances of language use that have been identified by experts as problematic. Creative use of language is not flagged unless it has been previously identified as problematic, furthering our aim of a quiet and authoritative tool. And even an author who intentionally flouts conventions for creative reasons will benefit from a thorough understanding of them [Bri04].

3. Editor [edi] has built a corpus which compares the performance of various grammar checkers. Their corpus contains "real-world examples of grammatical mistakes and stylistic problems taken from published sources". A corpus made of errors will maximize true positives, but misestimate false discovery rates in real-world documents. Their corpus is not publicly available, and they do not provide a standard format for describing corpora annotated with false positives and negatives.

Furthermore, technical writing of all kinds is often characterized by consistent language use and precise terminology. Even an author who views all writing as inextricably creative must sometimes direct that creativity toward a particular aim. Software documentation, technical manuals, and legal briefs, and pedagogical writing all feature this need and are improved when the author follows the conventions of a field.

Lastly, science demands consistency to promote clarity and replication. At the same time, scientists are in the business of expressing ideas that challenge even the greatest of minds, and their success depends on conveying those ideas to people who then use the ideas in their own work. When an idea is hard to grasp, simplicity and clarity will further its proliferation.

Contributing to Proselint

The primary avenue for contributing to Proselint is by contributing code to its GitHub repository. In particular, we have developed an extensive set of Issues that range from trivial-to-fix bugs to lofty features whose addition are entire research projects in their own right. To merit inclusion in Proselint, contributed rules should be accompanied by a citation to a recognized expert on language usage who has defined the rule clearly. This is not because language experts are the only arbiters of language usage, but because our goal is explicitly to aggregate best practices as put forth by the experts.

A secondary avenue for contributing to Proselint is through discovery of false positives: instances where Proselint flags well-formed idiomatic prose as containing a usage error. In this way, people with expertise in editing, language, and quality assurance can make a valuable contribution that directly improves the metric we use to gauge success.

Acknowledgments

Proselint is supported in part by the Berkeley Center for Technology, Society and Policy through the CTSP Fellows program, specifically for applying it to the problem of improving governmental communications as laid out in the Federal Plain Language Guidelines. We thank several reviewers who gave feedback on the manuscript, including Dan Lewis, David Lippa, Scott Rostrup, and Stéfan van der Walt. This work was presented as a talk at *SciPy* 2016 (YouTube).

References

[AI00] L. Argote and P. Ingram. Knowledge transfer: A basis for competitive advantage in firms. *Organizational Behavior and Human Decision Processes*, 82:150–169, 2000.

[Ass94] American Psychological Association. *Publication Manual of the American Psychological Association*. American Psychological Association Washington, 1994.

[BGJ10] David M Blei, Thomas L Griffiths, and Michael I Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*, 57(2):7, 2010.

[BK06] Richard Bergmair and Stefan Katzenbeisser. Content-aware steganography: about lazy prisoners and narrow-minded wardens. In *International Workshop on Information Hiding*, pages 109–123. Springer, 2006.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Sebastopol, CA, 2009. 504 pages.

[BL09] David M Blei and John D Lafferty. Topic models. *Text mining: classification, clustering, and applications*, 10(71):34, 2009.

[Bri04] Robert Bringhurst. *The Elements of Typographic Style. 3rd revision*. Canada, USA: Hartley & Marks, 2004.

[CDIT12] Martin Chodorow, Markus Dickinson, Ross Israel, and Joel R Tetreault. Problems in evaluating grammatical error detection systems. In *COLING 2012*, pages 611–628, 2012.

[Cho56] Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[CM83] Lorinda L Cherry and Nina H Macdonald. The unix writers workbench software. *Byte*, 8(10):241, 1983.

[edi] Comparing grammar checkers: Holding grammar scammers' feats to the fire. http://www.serenity-software.com/pages/comparisons.html. Accessed: 2016-07-11.

[FH10] B. J. Fogg and J. Hreha. Behavior wizard: a method for matching target behaviors with solutions. *International Conference on Persuasive Technology*, pages 117–131, 2010.

[Fle48] Rudolph Flesch. A new readability yardstick. *Journal of Applied Psychology*, 32(3):221, 1948.

[FS57] Charles B Ferster and Burrhus Frederic Skinner. Schedules of reinforcement. 1957.

[Gar16] Bryan Garner. *Garner's Modern English Usage*. Oxford University Press, 2016.

[hac] Hacker news: Proselint (proselint.com). https://news.ycombinator.com/item?id=11232882. Accessed: 2016-07-05.

[HJM+82] George E. Heidorn, Karen Jensen, Lance A. Miller, Roy J. Byrd, and Martin S Chodorow. The epistle text-critiquing system. *IBM Systems Journal*, 21(3):305–326, 1982.

[Jac86] Peter Jackson. Introduction to expert systems. 1986.

[Joh77] S. Johnson. Lint, a C program checker. *Computer Science Technical Report 65, Bell Laboratories*, December 1977.

[Kel00] Frank Keller. *Gradience in grammar: Experimental and computational aspects of degrees of grammaticality*. PhD thesis, 2000.

[LCGT10] Claudia Leacock, Martin Chodorow, Michael Gamon, and Joel Tetreault. Automated grammatical error detection for language learners. *Synthesis Lectures on Human Language Technologies*, 3(1):1–134, 2010.

[Mił10] Marcin Miłkowski. Developing an open-source, rule-based proofreading tool. *Software: Practice and Experience*, 40(7):543–566, 2010.

[MPU+92] Gary F Marcus, Steven Pinker, Michael Ullman, Michelle Hollander, T John Rosen, Fei Xu, and Harald Clahsen. Overregularization in language acquisition. *Monographs of the Society for Research in Child Development*, pages i–178, 1992.

[MS01] Casey Miller and Kate Swift. *The handbook of nonsexist writing*. iUniverse, 2001.

[MSA+11] Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K Gray, Joseph P Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, et al. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014):176–182, 2011.

[MW63] Frederick Mosteller and David L Wallace. Inference in an authorship problem: A comparative study of discrimination methods applied to the authorship of the disputed Federalist Papers. *Journal of the American Statistical Association*, 58(302):275–309, 1963.

[Nab03] Daniel Naber. *A rule-based style and grammar checker*. PhD thesis, 2003.

[Nor] Copy Editing at The New Yorker with Mary Norris. https://andyrossagency.wordpress.com/2009/09/20/. Accessed: 2016-07-11.

[Phi04] S. U. Philips. *Language and social inequality. A Companion to Linguistic Anthropology*. 2004.

[Pla11] Plain Language Action and Information Network. Federal plain language guidelines. http://www.plainlanguage.gov/howto/guidelines/bigdoc/fullbigdoc.pdf, 2011.

[PRR12] Fabrizio Perin, Lukas Renggli, and Jorge Ressia. Linguistic style checking with program checking tools. *Computer Languages, Systems & Structures*, 38(1):61–72, 2012.

[Rud97] Joseph Rudman. The state of authorship attribution studies: Some problems and solutions. *Computers and the Humanities*, 31(4):351–365, 1997.

[RZGSS04] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 487–494. AUAI Press, 2004.

[S+16] Richard Stallman et al. GNU coding standards. https://www.gnu.org/prep/standards/html_node/Errors.html/, 2016. Accessed: 2016-07-18.

[Ver00] Alex Vernon. Computerized grammar checkers 2000: Capabilities, limitations, and pedagogical possibilities. *Computers and Composition*, 17(3):329–349, 2000.

[VNC03] Salvatore Valenti, Francesca Neri, and Alessandro Cucchiarelli. An overview of current research on automated essay grading. *Journal of Information Technology Education*, 2:319–330, 2003.

[Was90] Anthony I Wasserman. Tool integration in software engineering environments. In *Software Engineering Environments*, pages 137–149. Springer, 1990.

[Yam13] Roman V Yampolskiy. Turing test as a defining feature of ai-completeness. In *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, pages 3–17. Springer, 2013.

[ZLCH06] Rong Zheng, Jiexun Li, Hsinchun Chen, and Zan Huang. A framework for authorship identification of online messages: Writing-style features and classification techniques. *Journal of the American Society for Information Science and Technology*, 57(3):378–393, 2006.