




# Nostril: A nonsense string evaluator written in Python

Michael Hucka<sup>1</sup>

<sup>1</sup> Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125, USA

DOI: [00.00000/joss.00000](https://doi.org/10.0000/joss.00000)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Submitted: 00 January 0000

Published: 00 January 0000

## Licence

Authors of JOSS papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC-BY).

## Summary

Nostril (*Nonsense String Evaluator*) is a Python 3 module that can infer whether a given word or text string is likely to be nonsense or meaningful text. A “meaningful” string of characters is one constructed from real or real-looking English words or fragments of real words (even if the words are *runtogetherlikethis*). The main use case for Nostril is to decide whether short strings returned by source code mining methods are likely to be program identifiers (of classes, functions, variables, etc.), or random or other non-identifier strings.

Nostril is easy to use. It provides a Python function named `nonsense()`; this function takes a single text string as an argument and returns a Boolean value as a result. Here is an example of its use. The following code,

```
from nostril import nonsense
for s in ['bunchofwords', 'xywinlist', 'faiwtlwexu', 'asfgtqwafazfy']:
    if nonsense(s):
        print("{} is nonsense".format(s))
    else:
        print("{} is real".format(s))
```

produces the following output:

```
bunchofwords is real
xywinlist is real
faiwtlwexu is nonsense
asfgtqwafazfy is nonsense
```

Nostril also includes a command-line program named `nostril`; it will evaluate strings provided on the command line or in a file, and is useful for experimenting with Nostril or using it in command-oriented workflows.

## The need for detecting nonsense

A number of research efforts have investigated extracting and analyzing textual information contained in software artifacts (e.g., Dit et al. 2011; Linstead et al. 2009). However, source code files can contain meaningless text, such as random text used as markers or test cases, and code extraction methods can also sometimes make mistakes and produce garbled text. When used in processing pipelines without human intervention, it is often important to include a data cleaning step before passing tokens extracted from source code to subsequent analysis or machine learning algorithms. Thus, a basic (and often unmentioned) step is to filter out nonsense tokens.

Discerning real identifiers from nonsense is a surprisingly difficult problem, because program identifiers often consist of words, acronyms and word fragments jammed together

(e.g., `ioFlXFndrInfo`). The resulting strings can challenge even humans. Nostril uses a combination of (1) a prefilter that detects simple positive and negative cases using heuristic rules and (2) a custom TF-IDF (Manning, Raghavan, and Schütze 2009) scoring scheme that uses letter 4-grams as features. The software includes a precomputed table of n-gram weights derived by training the system on a large set of strings constructed from concatenated American English words, real text corpora, and other inputs. Parameter values were optimized using the evolutionary algorithm NSGA-II (Deb et al. 2000).

By default, Nostril is tuned to reduce false positives – it is more likely to say something is *not* gibberish when it really might be. This bias is motivated by Nostril’s original purpose of filtering source code identifiers for machine-learning applications, where false positives would cause real identifiers to be filtered out and potentially-useful features to be missed. However, the bias and other parameters (and the table of n-grams) can also be retrained if applications require it.

Nostril is reasonably fast: once the package is loaded, a string evaluation takes 30–50 microseconds on average on a 4 Ghz Apple macOS computer. Nostril is accurate: it achieves 99.76% on the the Ludiso identifier oracle (Binkley et al. 2013) and 91.70% on a test set of 1,000,000 machine-generated random strings.

## Acknowledgments

This material is based upon work supported by the [National Science Foundation](#) under Grant Number 1533792. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## References

- Binkley, David, Dawn Lawrie, Lori Pollock, Emily Hill, and K Vijay-Shanker. 2013. “A Dataset for Evaluating Identifier Splitters.” In *Proceedings of the 10th Working Conference on Mining Software Repositories*, 401–4. IEEE Press. <http://dl.acm.org/citation.cfm?id=2487085.2487158>.
- Deb, Kalyanmoy, Samir Agrawal, Amrit Pratap, and T Meyarivan. 2000. “A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II.” In *Parallel Problem Solving from Nature PPSN VI*, 849–58. Springer. [https://doi.org/10.1007/3-540-45356-3\\_83](https://doi.org/10.1007/3-540-45356-3_83).
- Dit, B, L Guerrouj, D Poshyvanyk, and G Antoniol. 2011. “Can Better Identifier Splitting Techniques Help Feature Location?” In *2011 IEEE 19th International Conference on Program Comprehension*, 11–20. <https://doi.org/10.1109/ICPC.2011.47>.
- Linstead, Erik, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. 2009. “Sourcerer: Mining and Searching Internet-Scale Software Repositories.” *Data Mining and Knowledge Discovery* 18 (2):300–336. <https://doi.org/10.1007/s10618-008-0118-x>.
- Manning, Christopher D, Prabhakar Raghavan, and Hinrich Schütze. 2009. *Introduction to Information Retrieval*. [Online edition]. Cambridge University Press. <https://doi.org/10.1017/CBO9780511809071>.