PatternCounter

Joao Felipe Pimentel

Sep 15, 2022

CONTENTS

CHAPTER

ONE

FEATURES

This tool allows to count patterns in lists of sequential groups using *rules* and *variables*.

For the following examples, consider the following file (`sequences.txt`):

```
A -1 -2
B -1 -2
A B -1 -2
A -1 B C -1 -2
B -1 A B -1 A -1 C -1 -2
```

Example 1: Count how many sequences contain both the elements A and B:

```
$ patterncounter count "A B" -n -f sequences.txt
Supp((A B)) = 0.6 | 3 lines: 2, 3, 4
```

Example 2: Count how many sequences contain elements A and B at the same group:

```
$ patterncounter count "A & B" -n -f sequences.txt
Supp(A & B) = 0.4 | 2 lines: 2, 4
```

Example 3: Count how many sequences have an element B that after after A:

```
$ patterncounter count "A -> B" -n -f sequences.txt
Supp(A -> B) = 0.2 | 1 lines: 3
```

Example 4: Count in how many sequences the element B was removed within an interval of A:

```
$ patterncounter count "[A OutB]" -n -f sequences.txt
Supp([A OutB]) = 0.2 | 1 lines: 4
```

Example 5: Count in how many sequences there is an element C that occurs after an interval of A:

```
$ patterncounter count "[A] -> C" -n -f sequences.txt
Supp([A] -> C) = 0.4 | 2 lines: 3, 4
```

Example 6: Show results even when the pattern does not exist:

```
$ patterncounter count "Z" -n -f sequences.txt -z
Supp(Z) = 0.0 | 0 lines
```

In addition to using simple rules, it is possible to define multiple rules and calculated association rules metrics among them:

Example 7: Count both how many sequences have an interval of A, and how many sequences have an interval of A with an element B inside:

```
$ patterncounter count "[A]" "[A B]" -n -f sequences.txt
Supp([A], [A B]) = 0.4 | 2 lines: 2, 4
Association Rule: [A] ==> [A B]
  Supp([A]) = 0.8 | 4 lines: 0, 2, 3, 4
  Supp([A B]) = 0.4 | 2 lines: 2, 4
  Conf = 0.5
  Lift = 1.25
Association Rule: [A B] ==> [A]
  Supp([A B]) = 0.4 | 2 lines: 2, 4
  Supp([A]) = 0.8 | 4 lines: 0, 2, 3, 4
  Conf = 1.0
  Lift = 1.25
```

It is also possible to define variables.

Example 8: Count how many sequences have groups with two distinct elements:

```
$ patterncounter count "x & y" -v "x" -v "y" -n -f sequences.txt -z
Supp(x & y) = 0.6 | 3 lines: 2, 3, 4

[BINDING: x = B; y = A]
  Supp(B & A) = 0.4 | 2 lines: 2, 4

[BINDING: x = A; y = B]
  Supp(A & B) = 0.4 | 2 lines: 2, 4

[BINDING: x = B; y = C]
  Supp(B & C) = 0.2 | 1 lines: 3

[BINDING: x = A; y = C]
  Supp(A & C) = 0.0 | 0 lines

[BINDING: x = C; y = B]
  Supp(C & B) = 0.2 | 1 lines: 3

[BINDING: x = C; y = A]
  Supp(C & A) = 0.0 | 0 lines
```

Note that the result first shows the combined metrics (union).

Finally, given a file of sequences, it is also possible to select its lines (0-indexes):

```
$ patterncounter select -f sequences.txt -n 4
0| A -1 -2
2| A B -1 -2
4| B -1 A B -1 A -1 C -1 -2
```

CHAPTER

TWO

INSTALLATION

You can install *PatternCounter* via pip from PyPI:

```
$ pip install patterncounter
```

CHAPTER

THREE

USAGE

Please see the *Command-line Reference* for details.

CHAPTER

FOUR

CONTRIBUTING

Contributions are very welcome. To learn more, see the *Contributor Guide*.

CHAPTER

FIVE

LICENSE

Distributed under the terms of the *MIT license*, *PatternCounter* is free and open source software.

CHAPTER

SIX

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

SEVEN

CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

7.1 Usage

7.1.1 patterncounter

PatternCounter. This tool counts patterns from lists of sequences.

```
patterncounter [OPTIONS] COMMAND [ARGS]...
```

Options

--version
: Show the version and exit.

convert

Convert spmf dictionary to sequence list.

```
patterncounter convert [OPTIONS]
```

Options

-l, --line-sep <line_sep>
: line separator

-i, --interval-sep <interval_sep>
: interval separator

-e, --element-sep <element_sep>
: element separator

-r, --remove <remove>
: remove elements that start with prefix

-f, --file <file>
load input from file

-s, --stop-on-failures
stop on failures

count

Count patterns in file.

```
patterncounter count [OPTIONS] [PATTERN_DEFINITIONS]...
```

Options

-l, --line-sep <line_sep>
line separator

-i, --interval-sep <interval_sep>
interval separator

-e, --element-sep <element_sep>
element separator

--in-prefix <in_prefix>
prefix that is added to elements when they enter a slice

--out-prefix <out_prefix>
prefix that is added to elements when they exit a slice

-z, --show-support-zero
show rule when support is zero

-b, --hide-bindings
hide bindings

-n, --line-number
show line numbers

-t, --line-text
show line text

-f, --file <file>
load input from file

-c, --csv
display result as csv

Arguments

PATTERN_DEFINITIONS
Optional argument(s)

select

Select lines from file.

```
patterncounter select [OPTIONS] [LINES]...
```

Options

-l, --line-sep <line_sep>
line separator

-f, --file <file>
load input from file

-n, --line-number
show line numbers

Arguments

LINES
Optional argument(s)

show

Show sequence list with In and Out.

```
patterncounter show [OPTIONS]
```

Options

-l, --line-sep <line_sep>
line separator

-i, --interval-sep <interval_sep>
interval separator

-e, --element-sep <element_sep>
element separator

--in-prefix <in_prefix>
prefix that is added to elements when they enter a slice

--out-prefix <out_prefix>
prefix that is added to elements when they exit a slice

-n, --line-number
show line numbers

-f, --file <file>
load input from file

7.2 Rules

This document defines the syntax of the pattern rules. There are 4 types of constructs in the syntax: Elements, Unary Operators, Binary Operators, and Groups.

7.2.1 Elements

To define a pattern with elements, just write the element name. For instance to find the element `A`, the rule is just `A`. Suppose the sequence `A ... A B ... C D ... A`, this rule matches the first, second, and last groups.

In addition to simple elements, it is possible to find the group where the element was inserted. In this case, the rule that matches the insertion of an element `A` is `InA`. Suppose the sequence `A ... A B ... C D ... A`, this rule matches the first and last groups.

Similarly, it is possible to find the first group that the element does not appear anymore after appearing once. In this case, the rule that matches the removal of an element `A` is `OutA`. Suppose the sequence `A ... A B ... C D ... A`, this rule matches the third group.

Summary:

- Element: `A`
- Insertion: `InA`
- Removal: `OutA`

7.2.2 Unary Operators

Unary operators can be applied to rules. Currently, three operators are supported: First (`^`), Last (`$`), and Not (`~`).

The operator First (`^`) makes the rule only match if it is at the beginning of the sequence. The rule `^A` only matches the first group in the sequence `A ... A B ... C D ... A`.

The operator Last (`$`) is similar, but it only matches at the end of the sequence. The rule `$A` only matches the last group in the sequence `A ... A B ... C D ... A`.

The operator Not (`~`) inverts matches. The rul `~A` does not match the sequence `A ... A B ... C D ... A`, because `A` has at least one match. Howerver `~E` matches the sequence.

Summary:

- First: `^A`
- Last: `$A`
- Not: `~A`

7.2.3 Binary Operators

Binary operators apply to sets of rules. Currently, five operators are supported: And ( ), Intersect (&), Or (|), Sequence (->), LooseSequence (=>).

Every rule separated by space is treated as using the And operator. Hence, the rule `A C` matches the sequence `A ... A B ... C D ... A`. Note that the individual rules do not need to match in the same group: `C` matches the third group, while `A` matches the other. Since, both rules matches the sequence, `A C` also matches it. However, rules such as `A E` or `A $C` do not match the sequence.

If you want to match the rule at the same position, it is possible to use the Intersect operator (&). In this case, `A & B` matches only the second group of the sequence `A ... A B ... C D ... A`.

The operator Or (|) matches the sequence if either one of its subrules matches the sequence. The rule `E | C` matches the sequence `A ... A B ... C D ... A` because `C` matches it, even though `E` does not match it.

The Sequence operator (->) only matches the sequence when the second rule appears after the first one. The rule `A -> C` matches `A ... A B ... C D ... A` because `C` appears after a group that contains `A`. Note that `C -> D` does not match the sequence, since `C` and `D` are on the same group. However, `A -> B` matches the sequence because `A` matches at least a group in the sequence (the first) that appears before `B`.

The LooseSequence operator (=>) allows to match elements as "sequence" when they are also on the same groups. It matches everythin that the Sequence operator matches in addition to the same-group situations. In this case, `C => D` matches the sequence `A ... A B ... C D ... A`. The order within the same group does not matter. Hence, `D => C` also matches the sequence.

Summary:

- And: `A B C`
- Intersect: `A & B`
- Or: `A | B`
- Sequence: `A -> B`
- LooseSequence: `A => B`

7.2.4 Groups

Finally, the last type of construct in the rules are groups. There are two types of groups: Parentheses (`()`) and Slices (`[]`, `{}`, `[}`, `{]`).

Parentheses allow nesting operators and building complex rules without worrying about operator precedences. For instance, the rule `(E | C) -> A` matches the sequence `A ... A B ... C D ... A` because it first evaluates `(E | C)` and founds that `C` matches the sequence, and then it found that `C -> A` matches the sequence. Parentheses with internal rules can be used in place of any other rule.

Slices matches sequential groups in the sequence. The rule `[A]` matches the first two groups of the `A ... A B ... C D ... A` and the last group too (slices can have size 1). In this simple form, the slice is not much more interesting than element rules. However, slices also allow finding rules within them. The rule `[A B]` matches the sequence because it finds a slice of `A` that contains `B` as well.

Is it possible to have complex rules with slices. For instance the rule `[A $B ~C]` matches a slice that does not have `C` and that ends with `B`. Hence, it matches the sequence `A ... A B ... C D ... A`. Note that the within slices, the unary operators refer to the slice groups. Hence `^` matches the first element of the slice, `$` matches the last, and `~` only considers the slice in the negative.

Square bracket slices consider all of the groups of the slices. It is also possible to match "open" slices using braces. The rule `{A B]` only matches slices of `A` that contain `B` in a position that is not the first. Similarly, the rule `[A B}` matches

slices of `A` that contain `B` in a position that is not the last. Finally, the rule `{A B}` matches slices of `A` that contain `B` in a position that is neither the first nor the last.

Summary:

- Parenthesis: `(A B)`
- Closed slice: `[A B]`
- Left-open slice: `{A B]`
- Right-open slice: `[A B}`
- Open slice: `{A B}`

7.3 Variables

This document describes the syntax of variable definitions (argument `-v` of command line).

There are three types of variable definitions:

- **Simple** (`-v "X"`): If you just define the name of the variable, the tool will try to bind it to all names that appear in your input file
- **Exclusive** (`-v "X~A,B"`): If you define a list of comma-separated elements after `~`, the tool will try to bind the variable to all names that appear in your input file, **except** the ones on the list
- **Inclusive** (`-v "X:A,B"`): If you define a list of comma-separated elements after `:`, the tool will **only** bind the variable to the elements you defined.

It is not necessary to define `In` and `Out` variables. If you have a variable `X` bound to `Element`, not only the rule `X` will match `Element`, but the rule `InX` will match `InElement` and the rule `OutX` will match `OutElement`.

7.4 Reference

7.4.1 patterncounter

Pattern Counter.

`patterncounter.`**`parse`**(*text*)

Parses text and returns a Rule.

Parameters
`text` (*str*) –

Return type
Rule

7.5 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

7.5.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

7.5.2 How to request a feature

Request features on the Issue Tracker.

7.5.3 How to set up your development environment

You need Python 3.8+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run patterncounter
```

7.5.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

7.5.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

7.6 Contributor Covenant Code of Conduct

7.6.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

7.6.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

7.6.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

7.6.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

7.6.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at joaofelipenp@gmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

7.6.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

7.6.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

7.7 License

PYTHON MODULE INDEX

INDEX

Symbols

L

M

P