# pubrunner Documentation

***Release =0.5.5***

**Jake Lever**

**Jul 11, 2020**

# Contents

CHAPTER 1

# Tutorial

## 1.1 The Simple Problem

We'll work with a deliberately simple text mining tool that counts every use of the phrase "text mining". We want to apply this to the entirety of PubMed. The code for the tool is in the examples/TextMiningCounter. It consists of two Python scripts. Count.py takes in a text file and outputs the frequency of "text mining" as an integer. Sum.py takes in a directory of frequency counts and adds them up.

## 1.2 Basic Metadata

The pubrunner.yml file is the key for this example to run on the Pubmed corpus. We first define some information about the tool. This may be used on the pubrunner.org website and when uploading the output of the text mining tool.

```
name: TextMiningCounter
version: 0.1
url: https://github.com/jakelever/pubrunner/tree/master/examples/TextMiningCounter
```

## 1.3 Adding Resources

Next we need to say which resources to use. The code below would tell PubRunner that the latest version of Pubmed is required for the full analysis. The full analysis is the default one that is run. PubRunner will manage the download of PUBMED and symlink it into the working directory. So to access the files, our commands (shown later) will look in the PUBMED directory.

```
resources:
    full:
        - PUBMED
```

However, the default format of Pubmed is the Pubmed XML format. Our text mining tool just works with plain text files. So we need to specify that using the format parameter for our resource. Note the added colon after PUBMED to show that we're going to add parameters.

```
resources:
    full:
       - PUBMED:
           format: txt
```

## 1.4 Adding Commands

Now we want to put in the shell commands to actually run our scripts on all of PubMed. We will use a certain notation to denote the input and output files. First we want to run Count.py on all the individual files in PubMed. We will use the percent wildcard system for this. The PUBMED resource is composed of a large set of files, and we want to run a Count.py instance for each file. The wildcard system allows us to generalise a command for all the files that fit a particular pattern.

The input and output annotations are shown as {IN:filename} and {OUT:filename}. PubRunner makes use of these to figure out what commands need to be rerun when files are updated. This is very useful for reducing the amount of computation when dealing with the regular updates to PubMed. It should be noted that the files in the IN and OUT tags will be in a separate working directory and not the directory where the code resides. The Count.py script takes in a text file (containing PubMed abstracts) and outputs a text file (containing the count). To execute it against all files in the PUBMED directory (which is a symlink of the PUBMED resource), you can use the input: {IN:PUBMED/%}. When matched with a output of {OUT:counts/%.txt}, the wildcards are filled in for both and all files in the PUBMED directory are processed. So the Count.py command is included in the pubrunner.yml file as below.

```
run:
   - python Count.py --inFile {IN:PUBMED/%} --outFile {OUT:counts/%.txt}
```

We then want to execute the Sum.py file on the counts directory. Using the same IN and OUT annotation (but without the wildcards), we can add it as following:

```
run:
   - python Count.py --inFile {IN:PUBMED/%} --outFile {OUT:counts/%.txt}
   - python Sum.py --inDir {IN:counts} --outFile {OUT:textminingcount.txt}
```

## 1.5 Output files

We can see that the final output of the tool is the textminingcount.txt file. We add that as an additional bit of information to the pubrunner.yml file. At the moment, we haven't set up PubRunner to actually upload results anywhere, but when we do this information will be used to upload only the specific results file.

```
output: textminingcount.txt
```

## 1.6 Running It

We now have enough information that we could run the TextMiningCounter project against all of PubMed. If we were in the TextMiningProject directory (and had made the pubrunner.yml file), we could execute the full analysis with the command below. The period denotes the current directory, so that's where pubrunner looks for the pubrunner.yml file. PubRunner makes use of a .pubrunner.settings.yml file in your home directory, so if you don't have one, it will prompt you with some settings questions before starting. This can be very useful to control where resources and intermediate files (which can be very large) are stored and how to manage file uploads.

```
pubrunner . # Don't run this. It will take time and disk space.
```

The command above would take a long time to execute. Initially, it may be preferable to be able to test it on a small set. The test command is below. But if you run it now, it will state that no test resources are specified.

```
pubrunner --test .
```

## 1.7 Adding a test mode

For this, we need to replace the PUBMED resource with a mini PUBMED resource for the test environment. There is a PUBMED_TWOFILES resource that uses only two files from PUBMED and can be executed quite quickly. We add that in as an extra resource under "test:" and include the rename parameter. This means that it will be symlinked as PUBMED instead of PUBMED_TWOFILES and so the run commands can be the same for the test mode.

```
resources:
   full:
      - PUBMED:
         format: txt
   test:
      - PUBMED_TWOFILES:
         format: txt
         rename: PUBMED
```

That completes the pubrunner.yml file for this project.

# CHAPTER 2

# Overview

PubRunner solves the problem of regularly running and testing a text mining tool. It is primarily designed for biomedical text mining tools that use Pubmed or Pubmed Central. It follows the four steps outlined below.

1. Fetch the latest version of the corpus to be mined (such as PubMed or PMC) and any additional corpora/resources needed
2. Intelligently execute the text mining tool using the corpora
3. Upload the output of the text mining tool to an appropriate location (e.g. FTP or Zenodo)
4. Update the PubRunner website with the location of the latest output and code

# CHAPTER 3

# Installation

PubRunner can be installed using pip and requires Python 3.

```
pip install pubrunner
```

# CHAPTER 4

# Running PubRunner

To run PubRunner, use the terminal command ‘pubrunner’. The main argument should be the location of the text mining tool. This can be a local directory or Github repo. The command will attempt to run the test example of the Bio2Vec project.

```
pubrunner --test https://github.com/jakelever/Ab3P
```

To run the full example, omit the “–test” flag. Important: depending on the project, this may required downloading the entirety of PubMed (~180GB). It is recommended to use a cluster for large projects like this.

CHAPTER 5

# Getting Started

A first place to start would be the *Tutorial* page. Then you could check out the example projects to see a few different use cases. And check out the *Ab3P project* that scans for abbreviations across PubMed and PubMed Central.

If you have any questions, ideas, bugs, please create an issue.