5/19/2020

Quang Le

**Executive summary:** The objective of this exercise is to think of a better naming scheme for machine learning runs. A good naming convention will give us: the ability to organize files in a manner that is easy to find and retrieve, give us quick information on the runs’ content without opening it and make it easier for us to query our graph database. While the current naming scheme *dataset-algorithm-feature-tune* earns a high score when subjected to the criteria for success, perhaps we can modify it in a way that is much more concise while not sacrificing the many advantages that it brings. By looking at 3 different ways to modify our current naming convention, using letters and numbers is the most efficient choice as it is concise and also intuitive enough for a person who is familiar with the system to understand the content of the file.

**Objective**: Naming schemes for machine learning results.

**Ideas**:

* Words and numbers (What we have right now)
* Numbers only
* First letters and numbers

**Criteria for success**

* It must be concise and easy to read
* No obvious bad practices such as whitespace or special characters (besides – and \_)
* Must provide a unique name for every single machine learning run
* It must give the user some information about the content without opening it
* If there are rules, they must be simple, easy to follow and remember.

**Brainstorm:**

Characteristics of a naming scheme we would want:

* The rules are clear and consistent
* It needs to create a unique name for every case
* The naming scheme must contain some information about the machine learning results whether it’s the name itself or its rules

The naming scheme we have right now - *dataset-algorithm-feature-tune* - contains just the right amount of information to make it unique for every case. However, I think we can change the way it is represented.

Firstly, I’ll be order *dataset-algorithm-feature-tune* to *algorithm \_dataset\_ feature\_tune* (lexicographic order).

**Words and numbers (What we have right now)**

Example: ESOL-rf-0-tuned

Pros

* Most intuitive -> Mostly don’t need any naming scheme info to understand.

Cons

* Slowest and Least memory efficient
* Too long -> Won’t fit very well into our graph

**Numbers Only:**

1. Random Number Generator (RNG)

Pros:

* No rules needed
* Easiest to create and apply

Cons:

* Contains no information about the machine learning results -> Every file is a mystery box that needs to be opened to know its component
* **Only apply when all else fails**

1. Apply current naming scheme:

**Rule**: Every component in *dataset-algorithm-feature-tune* will be represented by their lexicographic order with respect to their own set.

For example:

Currently, we have 5 elements in the dataset category: Lipophilicity, ESOL, water-energy, jak2\_pic50 and logP14k. Applying the rule, we get

|  |  |
| --- | --- |
| Lexicographic Order | Number |
| ESOL | 0 |
| jak2\_pic50 | 1 |
| Lipophilicity\* | 2 |
| logP14k | 3 |
| water-energy | 4 |

\* In lexicographic order, upper case letters come before lower case ones (<https://chortle.ccsu.edu/java5/Notes/chap92/ch92_2.html>)

By applying the rule, rf-ESOL-0-tuned, will become 4\_0\_5\_0 (I chose 5 for feature arbitrarily since it depends on the set of feature combination we want to use. rdkit2d starts with an r so it is unlikely to be a small number since we also have other features that starts with characters that is earlier alphabetically like morgancount or atomcount). Without the underscore in the middle, it will be impossible to tell which number belong to which component when the components in *algorithm \_dataset\_ feature\_tune* have more than 9 things in them. I also chose the underscore instead of (–) because (-) is also the subtraction operator.

\*\* As of right now, it seems like only *dataset* and *feature* have the potential to surpass 9 so let’s try and remove the underscores that are separating each number. Let’s say the run number is 311180. Following the *algorithm \_dataset\_ feature\_tune* scheme, since we know algorithm and tune will never surpass 9, we can deduce that the two numbers 3 and 0 at each end must be *algorithm* and *tune*, so the run uses the 11th dataset and the 18th feature combination in their respective list. However, it gets hairy once the order of magnitude surpasses 2. For the run number 3111180, the number in between is 1111. There is no way to tell which part of 1111 belongs to *dataset* and which belongs to *feature*.

***Pros****:*

* Most efficient in terms of speed and memory since integers are mutable.
* The rule is simple enough to easily apply and automate.

***Cons****:*

1. Need to decide whether we need to add a separator or not depending on how big this project is, \*\* might not be applicable.

**First Letters and Numbers:**

**Rule:** We use some of the letters in the file name to represent the elements in *algorithm* and *dataset* while using a numbering system for *feature* and *tune*.

For *algorithm* specifically, regressors will be in upper case and classifiers will be in lower case

Example:

rf-ESOL-0-tuned will become RE51 where: R is the first letter in Random Forest Regressor, E is the first letter in ESOL, 5 is for rdkit2d (Again, I chose 5 for feature arbitrarily since it depends on the set of features and the combinations we want to use. rdkit2d starts with an r so it is unlikely to be a small number since we also have other features like morgancount or atomcount that are earlier alphatically), and 1 is for tuned.

Let’s say we have Lipophilicity-ID (1) and Lipophilicity14k (2) as two of the datasets that have similar names.

We can do one of the followings:

1. We can represent (1) as -ID and (2) as 14k
2. We can represent and (1) as “–“ and (2) as “1”
3. We can represent (1) as “L-“ and (2) as “L1”
4. We can represent (1) as LD and (2) as Lk
5. We can represent (1) as “L1” and (2) as “L2”

For option (a), the algorithm removes the parts that are duplicates and keep the difference. But if the situation was Lipophilicity-ID (1) and Lime (2), they will be represented as ophilicity-ID for (1) and “me” for (2). While this is easy to implement computationally, it can still lead to very long names.

For option (b), the algorithm compares the two names until it finds the first character that they differ from each other and use it to represent them. Therefore, resulting in – for (1) and 1 for (2). This is much more ideal than option (a) but at first glance, we won’t know what “–“ and “1” actually represent.

Option (c) is similar to option (b) but it keeps the first different character and the first character of each name. Since they both starts with “L”, it will be “L-“ for (1) and “L1” for (2). While this more intuitive than option (c) – since someone who knows about the dataset can probably deduce that both of them start with an L – the second part does not provide any useful information.

Option (d) keeps the first and last character of each dataset. I think this is as intuitive as it’s going to get. If we have names that contain the same first and last letter, the algorithm will add more letter starting from the last character. If we have Lipophilicity-ID (1) and Lipophilicity14kD (2), they will be represented as LID and LkD respectively.

Option (e) uses the first letter and the lexicographic order of each name in its own set. Lexicographically, Lipophilicity-ID places before Lipophilicity14k, so Lipophilicity-ID will be L1 and Lipophilicity14k will be L2.

**Overall,** I think (d) is the clear winner. While all 5 options are valid - and they will all need some explanation in the code to help the user – option (d) is the most intuitive for users who are familiar with the content of the dataset.

*Pros:*

* Fast and memory efficient
* It can be quite intuitive depending on the situation
* It RE51 sounds cooler than 4050 (Most important pros)

*Cons:*

* Lots of ways to go about it. There maybe even better options that I have not thought of.