JUL 8TH, 2011

FuzzyWuzzy: Fuzzy String Matching in Python

seatgeek open sourced seatgeek/fuzzywuzzy
Fuzzy String Matching in Python

We've made it our mission to pull in event tickets from every corner of the internet, showing you them all on the same screen so you can compare them and get to your game/concert/show as quickly as possible.

Of course, a big problem with most corners of the internet is labeling. One of our most consistently frustrating issues is trying to figure out whether two ticket listings are for the same real-life event (that is, without enlisting the help of our army of interns).

To pick an example completely at random, Cirque du Soleil has a show running in New York called "Zarkana". When we scour the web to find tickets for sale, mostly those tickets are identified by a title, date, time, and venue. Here is a selection of some titles we've actually seen for this show:

```
Cirque du Soleil Zarkana New York
Cirque du Soleil-Zarkana
Cirque du Soleil: Zarkanna
Cirque Du Soleil - Zarkana Tickets 8/31/11 (New York)
Cirque Du Soleil - ZARKANA (Matinee) (New York)
Cirque du Soleil - New York
```

As far as the internet goes, this is not too bad. An normal human intern would have no trouble picking up that all of these listings are for the same show. And a normal human intern would have no trouble picking up that those listings are **different** than the ones below:

```
Cirque du Soleil Kooza New York
Cirque du Soleil: KA
Cirque du Soleil Zarkana Las Vegas
```

But as you might imagine, we have far too many events (over 60,000) to be able to just throw interns at the problem. So we want to do this programmatically, but we also want our programmatic results to pass the "intern" test, and make sense to normal users.

To achieve this, we've built up a library of "fuzzy" string matching routines to help us along. And good news! We're open sourcing it. The library is called "Fuzzywuzzy", the code is pure python, and it depends only on the (excellent) **difflib** python library. It is available on **Github** right now.

String Similarity

The simplest way to compare two strings is with a measurement of edit distance. For example, the following two strings are quite similar:

```
NEW YORK METS
NEW YORK MEATS
```

Looks like a harmless misspelling. Can we quantify it? Using python's difflib, that's pretty easy

```
from difflib import SequenceMatcher
m = SequenceMatcher(None, "NEW YORK METS", "NEW YORK MEATS")
m.ratio() ⇒ 0.962962962963
```

So it looks like these two strings are about 96% the same. Pretty good! We use this pattern so frequently, we wrote a helper method to encapsulate it

```
fuzz.ratio("NEW YORK METS", "NEW YORK MEATS") ⇒ 96
```

Great, so we're done! Not quite. It turns out that the standard "string closeness" measurement works fine for very short strings (such as a single word) and very long strings (such as a full book), but not so much for 3-10 word labels. The naive approach is far too sensitive to minor differences in word order, missing or extra words, and other such issues.

Partial String Similarity

Here's a good illustration:

```
fuzz.ratio("YANKEES", "NEW YORK YANKEES") ⇒ 60
fuzz.ratio("NEW YORK METS", "NEW YORK YANKEES") ⇒ 75
```

This doesn't pass the intern test. The first two strings are clearly referring to the same team, but the second two are clearly referring to different ones. Yet, the score of the "bad" match is higher than the "right" one.

Inconsistent substrings are a common problem for us. To get around it, we use a heuristic we call "best partial" when two strings are of noticeably different lengths (such as the case above). If the shorter string is length m, and the longer string is length n, we're basically interested in the score of the best matching length-m substring.

In this case, we'd look at the following combinations

```
fuzz.ratio("YANKEES", "NEW YOR") ⇒ 14
fuzz.ratio("YANKEES", "EW YORK") ⇒ 28
fuzz.ratio("YANKEES", "W YORK ") ⇒ 28
fuzz.ratio("YANKEES", " YORK Y") ⇒ 28
...
fuzz.ratio("YANKEES", "YANKEES") ⇒ 100
```

and conclude that the last one is clearly the best. It turns out that "Yankees" and "New York Yankees" are a perfect partial match…the shorter string is a substring of the longer. We have a helper function for this too (and it's far more efficient than the simplified algorithm I just laid out)

```
fuzz.partial_ratio("YANKEES", "NEW YORK YANKEES") ⇒ 100
fuzz.partial_ratio("NEW YORK METS", "NEW YORK YANKEES") ⇒ 69
```

That's more like it.

Out Of Order

Substrings aren't our only problem. We also have to deal with differences in string construction. Here is an extremely common pattern, where one seller constructs strings as "<HOME_TEAM> vs <AWAY_TEAM>" and another constructs strings as "<AWAY_TEAM> vs <HOME_TEAM>"

```
fuzz.ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs New York Mets
fuzz.partial_ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs New Y
```

Again, these low scores don't pass the intern test. If these listings are for the same day, they're certainly referring to the same baseball game. We need a way to control for string construction.

To solve this, we've developed two different heuristics: The "token_sort" approach and the "token_set" approach. I'll explain both.

Token Sort

The token sort approach involves tokenizing the string in question, sorting the tokens alphabetically, and then joining them back into a string. For example:

```
"new york mets vs atlanta braves"   →→  "atlanta braves mets new vs york"
```

We then compare the transformed strings with a simple ratio(). That nicely solves our ordering problem, as our helper function below indicates:

```
fuzz.token_sort_ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs Ne
```

Token Set

The token set approach is similar, but a little bit more flexible. Here, we tokenize both strings, but instead of immediately sorting and comparing, we split the tokens into two groups: intersection and remainder. We use those sets to build up a comparison string.

Here is an illustrative example:

```
s1 = "mariners vs angels"
s2 = "los angeles angels of anaheim at seattle mariners"
```

Using the token sort method isn't that helpful, because the second (longer) string has too many extra tokens that get interleaved with the sort. We'd end up comparing:

```
t1 = "angels mariners vs"
t2 = "anaheim angeles angels los mariners of seattle vs"
```

Not very useful. Instead, the set method allows us to detect that "angels" and "mariners" are common to both strings, and separate those out (the set intersection). Now we construct and compare strings of the following form

```
t0 = [SORTED_INTERSECTION]
t1 = [SORTED_INTERSECTION] + [SORTED_REST_OF_STRING1]
t2 = [SORTED_INTERSECTION] + [SORTED_REST_OF_STRING2]
```

And then compare each pair.

The intuition here is that because the SORTED_INTERSECTION component is always exactly the same, the scores increase when (a) that makes up a larger percentage of the full string, and (b) the string remainders are more similar. In our example

```
t0 = "angels mariners"
t1 = "angels mariners vs"
t2 = "angels mariners anaheim angeles at los of seattle"
fuzz.ratio(t0, t1) ⇒ 90
fuzz.ratio(t0, t2) ⇒ 46
fuzz.ratio(t1, t2) ⇒ 50
fuzz.token_set_ratio("mariners vs angels", "los angeles angels of anaheim at s
```

There are other ways to combine these values. For example, we could have taken an average, or a min. But in our experience, a "best match possible" approach seems to provide the best real life outcomes. And of course, using a set means that duplicate tokens get lost in the transformation.

```
fuzz.token_set_ratio("Sirhan, Sirhan", "Sirhan") ⇒ 100
```

Conclusion

So there you have it. One of the secrets of SeatGeek revealed. There are more tidbits in the library (available on Github), including convenience methods for matching values into a list of options. Happy hunting.

Posted by Adam Cohen • Jul 8th, 2011 • Development

« Announcing Soulmate

How SeatGeek Measures PR Coverage »

Comments

steven c shepard • 5 years ago

This is a super good introduction to using Python to understand how to properly process web-based text. More, please!

4 | Reply • Share ›

har_shaji • 6 years ago

Excellent work !

3 | Reply • Share ›

infomaven • 5 years ago

Wow, my mind just got a huge expansion on this topic, and this is just what I was looking for. Thanks for the great writeup.

1 | Reply • Share ›

bipul kumar • a year ago

its very helpful for me,thanks

| Reply • Share ›

Kuziwa Sachikonye • 2 years ago

Amazing work! Thank you so much for the value that you add to the community. I'd love to see how i can use some of what i've learnt here in my development. Definitley going to take a look into NFS and fuzzy learning!

| Reply • Share ›

Jordi Camps • 2 years ago

Perfectly explained, thanks!

I made a small adjustment when detecting the intersection, considering not only 'exact matching words' but also 'matches above a certain Levenshtein-threshold'. This makes mispellings not so critical in the algorithm. What if, in your example, the second "angels" was typed "angles"? The algorithm should still consider it an intersection token:

s1 = "mariners vs angels"
s2 = "los angeles ANGLES of anaheim at seattle mariners"

Cheers and thanks again,

| Reply • Share ›

Prashant Goyal • 2 years ago

Can it return the index of the matched string i.e index in s2 where s1 is matched the most?

| Reply • Share ›

Jiayu Zhang ➔ Prashant Goyal • 2 years ago

I'm searching for this too, so far I use fuzz function and record the max similarity index. But as for process function, i don't know how to find the exact index.

| Reply • Share ›

haha ttpro • 2 years ago

thanks

| Reply • Share ›

Karlotcha Hoa • 2 years ago

This line is wrong:

```
fuzz.ratio("YANKEES", "NEW YOR") ⇒ 14
```

The actual score should be 28 :)

| Reply • Share ›

Steve Hanov • 2 years ago

I wonder if this is related to Jacaard distance? It seems to be very similar to the ratio between sets of words. Jacaard distance is already a well studied natural language processing technique.

| Reply • Share ›

Y-Mi Wong ➔ Steve Hanov • 3 months ago

The token set ratio is the Jaccard distance

^ | v • Reply • Share ›

Justin Boylan-Toomey ➔ Steve Hanov • 2 years ago

Hi Steve, according to the FuzzyWuzzy documentation it uses Levenshtein distance.

^ | v • Reply • Share ›

Shen Ni • 3 years ago

This is great! Sorry if I misunderstood something, but was there a typo in the Token_set description, right below the line "We'd end up comparing:"?
You said that "t2 = "anaheim angeles angels los mariners of seattle vs"

But I don't think the string "vs" is in s2. I might have missed something - regardless, this is super clear and easy to understand, thanks!

^ | v • Reply • Share ›

Nick Mavridis • 3 years ago

does anyone have an alternative to this ? that they have used successfully?

^ | v • Reply • Share ›

Lilienfa Li • 3 years ago

Great explanation! Thank you! however, what caught my attention was the following...
An abbreviation for take back scheme is tbs. However, if I do:
fuzz.ratio('tbs', 'TBS') it results in a ratio of 0. Am I missing something?

Love the library so far :) Thank you for the effort ^^

^ | v • Reply • Share ›

Hansen D'silva ➔ Lilienfa Li • 2 years ago

Try fuzz.WRatio('tbs','TBS')

^ | v • Reply • Share ›

Ethan Furman ➔ Lilienfa Li • 3 years ago

The comparisons are case sensitive.

^ | v • Reply • Share ›

Doomboss • 3 years ago

Thank you so much for sharing this. Very useful!!!

^ | v • Reply • Share ›

Farhan Khan • 4 years ago

Gives a good introduction to the topic, great!!

^ | v • Reply • Share ›

Farhan Khan • 4 years ago

Very informative, appreciate the work!!

^ | v • Reply • Share ›

geekunlimited • 4 years ago

Amazing. Really helpful. Thanks

^ | v • Reply • Share ›

Kássio Machado • 4 years ago

this work is awesome! Thank you for sharing!

^ | v • Reply • Share ›

Maxim Romanov • 4 years ago

Very useful!

^ | v • Reply • Share ›

TRWells1 • 4 years ago

Awesome!

^ | v • Reply • Share ›

jubin kuriakose • 5 years ago

really interesting read!!

^ | v • Reply • Share ›

Robert Hodgson • 5 years ago

Awesome dude - was looking for something similar this is amazing

^ | v • Reply • Share ›

Vaibhav Jain • 5 years ago

Can you tell which research papers are used to implement this library?

^ | v • Reply • Share ›

Robert Hodgson • 5 years ago

Extremely useful - thank you!

^ | v • Reply • Share ›

A.Erkan CELIK • 5 years ago • edited

good works!
can I learn, which string part of long text has best ratio score like partial_ratio?

^ | v • Reply • Share ›

vishnuatrai • 6 years ago

I am searching for the same kind of tool.

Thanks!!

^ | v • Reply • Share ›