doctag

Tagging and parsing tag queries in Python

Dan Turkel

June 10th, 2019

Background

Tags

A **tag** is a common form of metadata for organizing files (or **documents**).

Tags are a convenient supplement to a file system because they are not hierarchical like folders are.

Tags and documents have a **many-to-many** relationship: one document may have many tags, and one tag may be applied to many documents.

Functionality

Tagging systems naturally motivate several additional actions beyond tagging and untagging:

Functionality

Tagging systems naturally motivate several additional actions beyond tagging and untagging:

show tags:	display all tags for a given document
show docs:	display all documents with a given tag

Functionality

Tagging systems naturally motivate several additional actions beyond tagging and untagging:

show tags:	display all tags for a given document
show docs:	display all documents with a given tag
merge tags:	replace all instances of `old` tag with `new`
delete tag:	remove all usages of a given tag
clean doc:	remove all tags from a given doc

Functionality

Tagging systems naturally motivate several additional actions beyond tagging and untagging:

show tags:	display all tags for a given document
show docs:	display all documents with a given tag
merge tags:	replace all instances of `old` tag with `new`
delete tag:	remove all usages of a given tag
clean doc:	remove all tags from a given doc
query docs:	display all documents matching a tag query

Data Structures: Database

One way to represent a set of tagged documents is through relational tables.[1]

[1] http://howto.philippkeller.com/2005/04/24/Tags-Database-schemas/

Data Structures: Database

One way to represent a set of tagged documents is through relational tables.[1]

docs
doc_id
doc_name

tag_map
tag_map_id
tag_id
doc_id

tags
tag_id
tag_name

[1] http://howto.philippkeller.com/2005/04/24/Tags-Database-schemas/

Data Structures: Database (Example)

docs

doc_id	doc_name
1	movies.txt
2	books.txt
3	school.txt

tag_map

tag_map_id	tag_id	doc_id
1	1	1
2	1	2
3	2	2
4	2	3

tags

tag_id	tag_name
1	list
2	learning

We can tag/untag with `INSERT`/`DELETE`, delete tags or docs with `DELETE`, and show tags or docs `SELECT`. Merging tags can be done with a targeted `INSERT` followed by a `DELETE`, and building tag queries is simply a matter of building `WHERE` clauses for `SELECT` statements.

Data Structures: Inverted Index

Another option for representing tagged documents is with an **index** and an **inverted index**.[2]

The index maps documents to tags, while the inverted index maps tags to documents.

Inverted indexes are used in NLP[3] and search[4] for quickly finding documents which contain specific user-defined content.

[2] https://stackoverflow.com/a/24993487

[3] https://nlp.stanford.edu/IR-book/html/htmledition/a-first-take-at-building-an-inverted-index-1.html

[4] https://www.elastic.co/guide/en/elasticsearch/guide/current/inverted-index.html

Data Structures: Inverted Index (Example)

index

doc	tags
movies.txt	list
books.txt	list, learning
school.txt	learning

inverse index

tag	docs
list	movies.txt, books.txt
learning	books.txt, school.txt

Tag and untag operations require writing to both indexes, but *show tags and show docs operations become trivial*. Deleting a tag is roughly the same process as in the tabular solution, and merging tags can be done by re-tagging in the index and unioning in the inverted index.

Querying has to be implemented through a series of intersections, unions, and negations.

doctag

doctag

doctag is a Python library for building index/inverted index tagging systems and performing actions on those systems.

The library includes a `TagIndex` class which stores the index and inverted index and implements methods for tagging and retrieval.

ultrajson[5] is used to (optionally) serialize and deserialize the `TagIndex` to disk *really fast*.

boolean.py[6] is used to parse arbitrarily complex tag queries, like:

“(list and learning) or (not work)”

[5] https://github.com/bastikr/boolean.py

[6] https://github.com/esnme/ultrajson

Features

See `notebooks/features.ipynb`

Performance

See `notebooks/performance.ipynb`

Query Parsing: Text to Expressions

The boolean.py library turns plaintext strings with a flexible syntax into nested **expression** objects, each with one or more arguments and an optional operation. E.g. “a or b” is evaluated to

```
OR(Symbol('a'), Symbol('b'))
```

where a **symbol** is the simplest expression, having no operation at all.

Similarly, a more complex query like “not a or (b and c)” evaluates to

```
OR(
    NOT(Symbol('a')),
    AND(Symbol('b'), Symbol('c'))
)
```

Query Parsing: Example

doctag recurses through expression arguments until it hits a bare symbol, at which point it fetches all documents tagged with that symbol. When all arguments of an operation expression are evaluated, it uses set logic to evaluate the entire expression.

If we let {list} represent the set of all documents tagged with “list,” we can follow simple set logic to parse morecomplex expressions:

- “a and b” is the set intersection of {a} and {b}
    - (featuring short-circuiting if {a} evaluates to the empty set)
- “a or b” is the set union of {a} and {b}
- “not a” is the set complement of {a} (all existing tags that are not in {a})

Links

doctag is on Github: https://github.com/daturkel/doctag