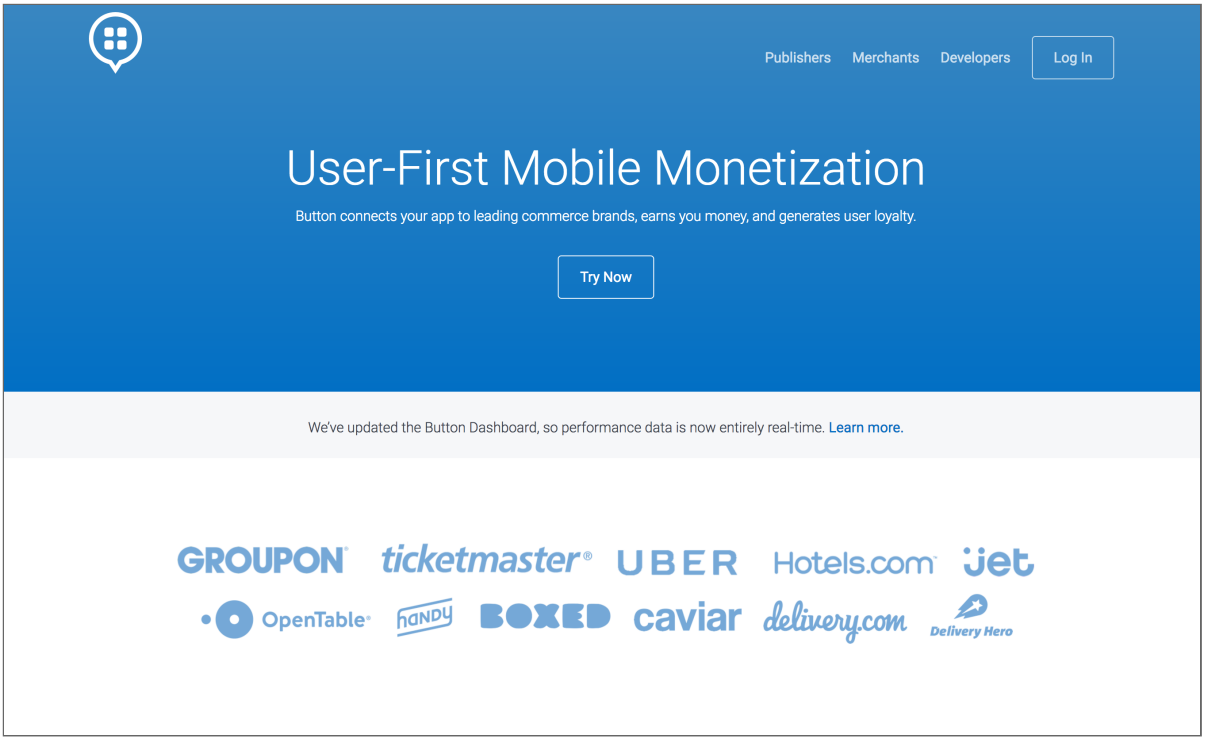
FUZZY SEARCH ALGORITHMS

PYCON 2017

https://github.com/jiaqi216/fuzzy-search-talk

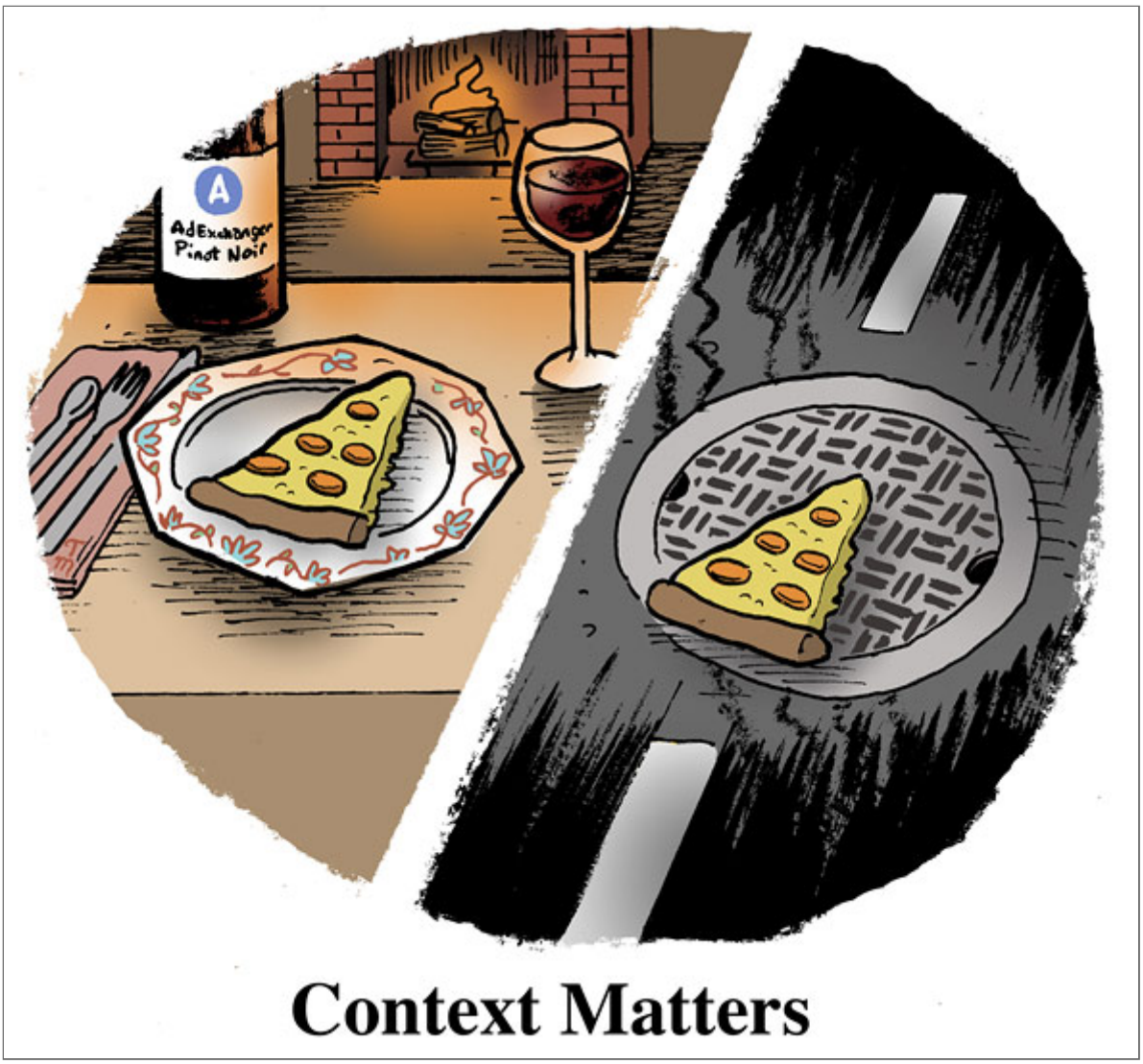
@jiaqicodes



https://blog.usebutton.com/

OVERVIEW

- Common Fuzzy Search Algorithms
- Implementation in Python/Postgres
- Semantic Analysis


A
AdExchanger
Pinot Noir
Context Matters


Entity
Resolution

PHONETIC ALGORITHM - SOUNDEX

AMERICAN SOUNDEX ENCODING

Number	Letter
1	B,F,P,V
2	C,G,J,K,Q,S,X,Z
3	D,T
4	L
5	M,N
6	R

Ignore A,E,I,O,U,H,W,Y

Pros	Cons
Deterministic/Easy to implement	Limited to Language/Dialect
Computationally Fast	Considers Letters Only

HOW DO YOU CALCULATE DISTANCE BETWEEN WORDS?

LEVENSHTEIN DISTANCE

aka Edit Distance

deletion

D[I]STANCE

D[ ]STANCE

insertion

DIST[ ]ANCE

DIST[I]ANCE

substitution

DIST[A]NCE

DIST[M]NCE

Damerau-Levenshtein: Also includes Transposition

DIST[AN]CE

DIST[NA]CE

Comparison

```
>>> j.damerau_levenshtein_distance('recieve', 'receive')
1
>>> j.levenshtein_distance('recieve', 'receive')
2
```

Pros	Cons
Deterministic/Easy to implement	pairwise comparison
Computationally Fast	might need customization

N-GRAM/TRIGRAM

MORE APPLICATIONS

- Leverage trigram as a stepping stone
- Can Build Statistical Model leveraging similarity score as feature

THINGS TO NOTE

- the data set i'm working with is not large - 100K rows

But even so ...
soundex query on 3 records - `Execution time: 0.485 ms`
soundex query on 100K records - `Execution time: 44.249 ms`

```
pycon=# explain analyze     SELECT
                a.description,
                similarity(lower(a.description), 'fuzz
    FROM        pypi as a
    WHERE       lower(a.description) % 'fuzzysearch'
    ORDER BY    similarity DESC;
```

```
 Planning time: 0.037 ms
 Execution time: 867.635 ms
(8 rows)
```

indices gist and gin indexes for trigrams

```
CREATE INDEX trgm_idx ON table_name USING gist (column_t gist_
```

or

```
CREATE INDEX trgm_idx ON table_name USING gin (column_t gin_tr
```

Pros	Cons
More Context! Yay!	Slower - need to calculate n-gram for each string
Proper Unit Analysis (1-gram, 2-gram etc)	

SEMANTIC SEARCH

NLTK: WORDNET

```
from nltk.corpus import wordnet

word1 = wordnet.synsets("blue")
word2 = wordnet.synsets("green")

word1[0].wup_similarity(word2[0])
```

```
0.875
```

WORD2VEC

.

WORD ANALOGY

SUMMARY

- fuzzy search - powerful for adding context to qualitative data
- use a stepping stone to broader analysis
- customize for own use case/scenario