Spelling Correction using Lucene FSTs

FOSDEM 2015

Anna Ohanyan (anna.ohanyan@here.com)
Christian Ziech (christian.ziech@here.com)
Users mistype their queries
The Task

Goal 1: Spelling robustness - return correct results even if the user mistyped the query

Goal 2: Precision - return as few as possible irrelevant results
Difficulties

- User queries are corrected word by word
- Spelling robustness means more complex search queries and more irrelevant matches
- Include the word the user actually meant
- Include as few as possible additional words in the search query
- Results have to be delivered quickly
Agenda

- Matching
- Ranking
Lucene's FuzzyQuery (4.x)

- Enables spelling robustness per term
- Searches for top-n terms per user term
- Requires an indexed field as dictionary
  - Dictionary comes for free
- Ranks terms by Levenshtein Distance
What is the Problem?

• By default 50 candidates per term
  – Slow query execution
  – Inaccurate
• High memory footprint due to CompiledAutomaton
• Candidates sorted by Levenshtein Distance
  – Limited customizability due to a lack of information
How do we solve it?

- Build a custom dictionary and store it in a Lucene FST (Finite State Transducer)
  - Meta information can be stored for each term
  - No automaton compilation needed
  - Flexibility for how to rank candidates
FST Based Solution

- Spelling corrections ordered using the meta information stored in the FST
  - Flexibility on what information to store
- Additional effort to build the FST
- FSTs are intersected with the automaton representing the user term
How Automaton Intersection Works

FST

Automaton
How Automaton Intersection Works

FST

Automaton
How Automaton Intersection Works

FST

Automaton
How Automaton Intersection Works

FST

Automaton
How Automaton Intersection Works

FST

Automaton
How Automaton Intersection Works

FST

Automaton
How Automaton Intersection Works
How Automaton Intersection Works

FST

Automaton
How Automaton Intersection Works

FST

Automaton
How Automaton Intersection Works

FST

Automaton
How Automaton Intersection Works

FST

Automaton
Implementation

- Intersecting FST with plain Automaton inspired by lucene-suggest's FSTUtil
  - Optimizations to reduce object allocations
  - No prefix matching
- Each match triggers a hit collector
- Terms scored by their meta information and Levenshtein distance
Performance
(“Avenue Franklin D. Roosevelt”)

FuzzyQuery
- Building automatons
  - <1ms
- Compiling automatons
  - 2ms
- Finding terms
  - 23ms
- Total
  - 26ms

FST Intersection
- Building automatons
  - <1ms
- Compiling automatons
  - Not needed
- Finding terms
  - 12ms
- Total
  - 13ms
Agenda

- Matching
- Ranking
Term Similarity

- Levenshtein distance
  minimum number of single-character insertions, deletions or substitutions

  Berlin → Belgien
  Berlin → Belin → Belgin → Belgien
Term Similarity

- Phonetic algorithms
  
  words encoded by their pronunciation

- Metaphone (Double Metaphone, Metaphone 3) for English and Germanic languages

  Tchaikovsky → XKFS
  Chaikowski → XKFS or XKS
  Chaykovsky → XKFS
Choosing best spelling corrections

Features that we store:
- Term frequency
- Term geo location

Features that we compute:
- Edit distance
- Phonetic distance
- Common typos
- First letter rule
Choosing best spelling corrections

- Ranking function to score spelling candidates
- Linear weighted function over all the features

\[ f(t) = \sum_{i \in \text{features}} w_i \times f_i \]

- Train weights
- Take top-n spelling corrections (n << 50)
Terms and spelling corrections ("Rue Chance Mailly Clichy")

- rue
- chance
- mailly
- clichy
- rues
- france
- milly
- cache
- rua
- crane
- mills
- cliche
- chan
- mill
- lich
- cachy
Term co-occurrence
(“Rue Chance Mailly Clichy”)
Term co-occurrence

Features that we store:

- Term co-occurrence likelihood
- Same language
- Same context
Dense subgraph
Densest at most $k$ subgraph problem

- Extract the subgraph with at most $k$ vertices and maximal density
- NP-complete
- Various definitions of density
- Various approximation algorithms available
Choosing density function

- Classical definition

\[ f(S) = \frac{\text{number of edges}}{\text{number of nodes}} \]

- For weighted graphs

\[ f(S) = \frac{\text{number of edges}}{\sum \text{node weights}} \]

- Our case

\[ f(S) = f\left(\sum_S w_s, \sum_E w_e, N\right) \]
Choosing minimum degree vertex

- Classical definition: number of edges

- For weighted graphs: averaged number of edges
Greedy 2-approximation algorithm

Graph(E, V)
density function f(V), minimum degree vertex selection function min(S)
S = V
while (S not empty) {
    v = min(S)
    remove v from S
    compute f(S)
}
return max f(S)

M Charikar. Greedy approximation algorithms for finding dense components in a graph. 2000
Final Result
Conclusions

Using FST + Automaton has many advantages:

- performance
- meta information stored for each term
- flexible term ranking

Using ranking/filtering techniques

- minimizes number of terms actually searched for
- comes at a cost of time
Do you have any questions?