To boost fields like titles in your blog search engine, giving them a higher score compared to content or keywords, you can adjust the relevance scoring process to prioritize matches in specific fields. Since you're using Python and SQLite with FTS5 and fuzzy scoring (as in the previous example), I'll show how to boost the title field in both the SQLite FTS5 search and the fuzzy scoring component. This ensures that matches in the title contribute more to the final relevance score, improving accuracy for queries like "Clients Accounts Payable."

Why Boost Fields?

- **Relevance**: Titles often summarize the main topic, so matches in titles are typically more relevant than matches in content or keywords.
- **User Experience**: Prioritizing title matches aligns results with user expectations, as users often expect title keywords to indicate a strong match.
- **Flexibility**: Boosting allows you to fine-tune the importance of fields (e.g., title > keywords > content).

Approaches to Boost Fields

1. **SQLite FTS5 with Custom Ranking**:
   - FTS5 uses the BM25 algorithm for ranking, but you can post-process results to boost matches in the title field.
   - Check if query terms appear in the title and assign a boost factor to increase the score.
2. **Fuzzy Scoring with Field Weighting**:
   - In the fuzzy scoring step (using `rapidfuzz`), compute separate similarity scores for each field (title, content, keywords) and apply weights to combine them.
   - Assign a higher weight to title matches.
3. **Hybrid Approach**:
   - Combine FTS5's BM25 scores with fuzzy scores, applying a boost to titles in both components for a balanced ranking.

Implementation Strategy

- Use SQLite FTS5 for efficient initial search and BM25 ranking.
- Post-process FTS5 results to boost title matches by checking for query term presence.
- Enhance fuzzy scoring with `rapidfuzz` by computing field-specific similarities and weighting the title higher.
- Combine scores for final ranking, ensuring title matches increase the overall score.

Python Example with Title Boosting

Below is an updated version of the previous example, modified to boost the title field in both FTS5 and fuzzy scoring. The title is given a higher weight (e.g., 2x for FTS5 and 50% in fuzzy scoring) to prioritize title matches for the query "Clients Accounts Payble."

blog_search_with_title_boost.py
python

Editar em arquivos • Mostrar o inline

Key Changes for Title Boosting

1. **FTS5 Title Boost**:
   - The `title_boost` parameter (set to 2.0) multiplies the FTS5 rank (BM25 score, which is negative) if any query term appears in the title.
   - Example: If "clients" or "accounts" is in the title, the FTS5 score is doubled, increasing its contribution to the final score.
   - Implementation: Checks if stemmed query terms appear in the title (`title_match`).
2. **Fuzzy Scoring Title Weight**:
   - The `title_weight` parameter (set to 0.5) gives the title's fuzzy score 50% of the total fuzzy score, with content and keywords splitting the remaining 50% equally.
   - Example: If title_score=90, content_score=80, keywords_score=70, the weighted fuzzy score is `0.5 * 90 + 0.5 * 0.5 * (80 + 70) = 82.5`.
   - This ensures title matches have a stronger influence.
3. **Combined Score**:
   - The final score combines weighted fuzzy score (60%) and boosted FTS5 score (40%), adjustable via weights.
   - Titles with query matches get higher scores due to both `title_boost` and `title_weight`.

How It Works with "Clients Accounts Payble"

- **Query Preprocessing**: Stemmed to "client account paybl" for FTS5 and fuzzy matching.
- **FTS5 Search**: Retrieves documents matching the stemmed query, ranked by BM25.
- **Title Boost in FTS5**: If "clients" or "accounts" appears in the title, the BM25 score is multiplied by `title_boost` (2.0).
- **Fuzzy Scoring**: Computes `token_sort_ratio` for title, content, and keywords separately. The title score is weighted higher (50%) than content and keywords (25% each).
- **Ranking**: Combines scores, prioritizing documents with title matches (e.g., "Managing Client Accounts" ranks higher than "General Finance Tips").

Example Output

For the query "Clients Accounts Payble" with the sample data:

text Recolher Encapsular Copiar

```text
Title: Managing Client Accounts
Content: Best practices for handling accounts payable and receivable for cli
Keywords: clients, accounts payable, accounts receivable, finance
Score: 94.20

Title: Accounts Payable Guide
Content: Tips for managing accounts payable for small businesses...
Keywords: accounts payable, clients, business
Score: 89.50

Title: General Finance Tips
Content: Overview of financial strategies for businesses...
Keywords: finance, business
Score: 65.30
```

- **Explanation**: "Managing Client Accounts" and "Accounts Payable Guide" rank higher because their titles contain "clients" and "accounts," boosted by `title_boost` and `title_weight`. The misspelling "Payble" is handled by fuzzy scoring.

Tuning Parameters

- **title_boost**: Increase (e.g., 3.0) for stronger title preference in FTS5, or decrease (e.g., 1.5) for subtler boosts.
- **title_weight**: Adjust between 0 and 1 (e.g., 0.7 for stronger title emphasis, 0.3 for balanced fields).
- **similarity_threshold**: Lower to 70 for more results, or raise to 90 for stricter matches.
- **Score Weights**: Adjust the 0.6 (fuzzy) and 0.4 (FTS5) weights in `combined_score` based on testing.

Performance Considerations

- **Efficiency**: Title boosting adds minimal overhead, as it's a simple check in FTS5 and weighted sum in fuzzy scoring.
- **Scalability**: For large blogs, limit fuzzy scoring to top N FTS5 results (e.g., top 50) to reduce computation.
- **Indexing**: Preprocess and index stemmed text in FTS5 to avoid query-time stemming.

Additional Tips

- **Field Weights**: Extend to boost keywords (e.g., `keywords_weight=0.3`) if they're more important than content.
- **Synonyms**: Add a synonym dictionary (e.g., "payable" = "payment") to improve fuzzy matching.
- **Testing**: Experiment with `title_boost` and `title_weight` on your dataset to balance precision and recall.

Dependencies

- Install: `pip install rapidfuzz nltk`
- SQLite FTS5: Included in Python's `sqlite3` (ensure your SQLite version supports FTS5).

If you have specific requirements (e.g., different boost values, additional fields, or a larger dataset), let me know, and I can further customize the code!