Information Retrieval Algorithms: BM25, HNSW, and MMR

Techniques for Finding and Ranking Relevant Documents

Introduction to Information Retrieval

Imagine searching for recipes online. You type "chocolate cake" and expect the most relevant recipes at the top. Information retrieval (IR) algorithms power this experience, scoring and ranking documents based on relevance to your query. In IR, we care about:

- **Relevance**: How well a document matches the query
- **Efficiency**: Fast retrieval from large collections
- **Diversity**: Ensuring varied results to cover different aspects of a query

This guide covers three cornerstone techniques:

1. **BM25**: A robust term-weighting ranking function
2. **HNSW**: A graph-based approximate nearest neighbor search
3. **MMR**: Maximal Marginal Relevance for diversified retrieval

BM25: Term-Weighting Ranking Function

Concept

BM25 scores documents by summing contributions of query terms, adjusting for term frequency and document length. Think of it as weighing how often keywords appear, while penalizing overly long documents.

Formula

$$\text{score}(D, Q) = \sum_{t \in Q} \text{IDF}(t) \cdot \frac{f(t, D) \cdot (k_1 + 1)}{f(t, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})}$$

- $f(t, D)$: term frequency in document D
- $|D|$: document length; avgdl: average length
- k_1, b: hyperparameters (common defaults: 1.2, 0.75)
- $\text{IDF}(t) = \ln \frac{N - n_t + 0.5}{n_t + 0.5}$: inverse document frequency

Short Example (Pseudo-Code)

```
def bm25_score(query_terms, doc, index):
    score = 0
    doc_len = len(doc)
    avgdl = index.avg_doc_len
    for t in query_terms:
        f = doc.term_freq(t)
        idf = log((index.N - index.doc_freq(t) + 0.5)/(index.doc_freq(t) + 0.5))
        denom = f + k1*(1 - b + b*doc_len/avgdl)
        score += idf * (f*(k1+1))/denom
    return score
```

Output Discussion

BM25 ranks documents by descending score. Higher scores indicate stronger relevance, balancing frequent term occurrence against document length.

Reflection

- **Key:** Strong baseline for text retrieval.
- **Pitfalls:** Needs careful tuning of k1 and b; ignores term proximity and semantics.
- **Applications:** Search engines, QA retrieval.

HNSW: Approximate Nearest Neighbor Search

Concept

Hierarchical Navigable Small World (HNSW) graphs enable ultra-fast approximate nearest neighbor search by connecting points in a multi-layer graph where higher layers have sparser connections.

Structure and Search

1. **Graph build**: Insert points one by one, linking to nearest neighbors at each layer.
2. **Search**: Start at top layer with an entry point, greedily move to closer neighbors, then descend layers to refine.

Short Example (Pseudo-Code)

```
from hnswlib import Index

# Build index
d = 128  # embedding dimension
index = Index(space='l2', dim=d)
index.init_index(max_elements=N, ef_construction=200, M=16)
index.add_items(data_vectors)

# Query
```

```
ef=50
index.set_ef(ef)
labels, distances = index.knn_query(query_vector, k=10)
```

Output Discussion

`labels` are indices of nearest neighbors; `distances` measure their distances. HNSW trades slight accuracy loss for massive speed gains.

Reflection

- **Key:** Scalable to millions of vectors with sub-millisecond queries.
- **Pitfalls:** Index parameters (M, ef_construction) require tuning; memory overhead.
- **Applications:** Vector search in semantic retrieval, recommendation.

MMR: Maximal Marginal Relevance

Concept

Maximal Marginal Relevance (MMR) balances relevance and diversity. When retrieving documents, MMR avoids redundancy by penalizing similarity to already selected items.

Formula

$$\text{MMR} = \arg\max_{D_i \in R \setminus S} \left[\lambda \, \text{Sim}(D_i, Q) - (1 - \lambda) \max_{D_j \in S} \text{Sim}(D_i, D_j) \right]$$

- R: candidate set; S: selected set
- λ: trade-off parameter between relevance and diversity

Short Example (Pseudo-Code)

```
selected = []
candidates = initial_ranked_list(query)
f = lambda x,y: cosine_similarity(x,y)
for _ in range(k):
    mmr_scores = {}
    for doc in candidates:
        if doc not in selected:
            relevance = f(doc, query)
            diversity = max(f(doc, sel) for sel in selected) if selected else 0
            mmr_scores[doc] = lambda_*relevance - (1-lambda_)*diversity
    next_doc = argmax(mmr_scores)
    selected.append(next_doc)
```

Output Discussion

MMR returns `selected`, a list balancing top relevance and diversity, improving user experience by covering varied aspects.

Reflection

- **Key:** Simple yet effective diversification technique.
- **Pitfalls:** Requires similarity function; λ tuning crucial.
- **Applications:** Search result diversification, recommendation lists.

Conclusion

BM25, HNSW, and MMR address core IR needs:

- **BM25**: Accurate term-based ranking.
- **HNSW**: Fast vector retrieval at scale.
- **MMR**: Balances relevance and diversity.

Together, these techniques form a powerful toolkit for building efficient, effective, and user-friendly search and recommendation systems.

Download the PDF above for a fully formatted, beginner-friendly chapter on Information Retrieval algorithms.