BM25 与词向量融合方法在文本检索中的应用与实验分析

沈琨翔

June 2025

1 研究背景与算法原理

1.1 信息检索的核心问题

在大规模文本检索系统中，如何根据用户查询（Query）从海量文档中找到最相关的内容，是信息检索的核心问题。传统的检索方法如 TF-IDF、BM25 等，主要基于词频统计，能够高效地处理大规模数据，但在处理同义词、语义相似等复杂场景时存在局限。

1.2 BM25 简介

BM25（Best Matching 25）是基于概率相关模型的文本检索算法。它通过词频、逆文档频率和文档长度归一化等机制，对文档与查询的相关性进行打分。BM25 在实际搜索引擎中应用广泛，具有高效、可解释等优点。

1.3 词向量与语义检索

近年来，Word2Vec 等词向量模型能够将词语映射为稠密的向量，捕捉词语间的深层语义关系。将词向量与 BM25 结合，可以弥补 BM25 对语义信息捕捉不足的问题，提升检索系统的智能化水平。

2 BM25+Word2Vec 融合算法结构与流程

2.1 算法结构流程图

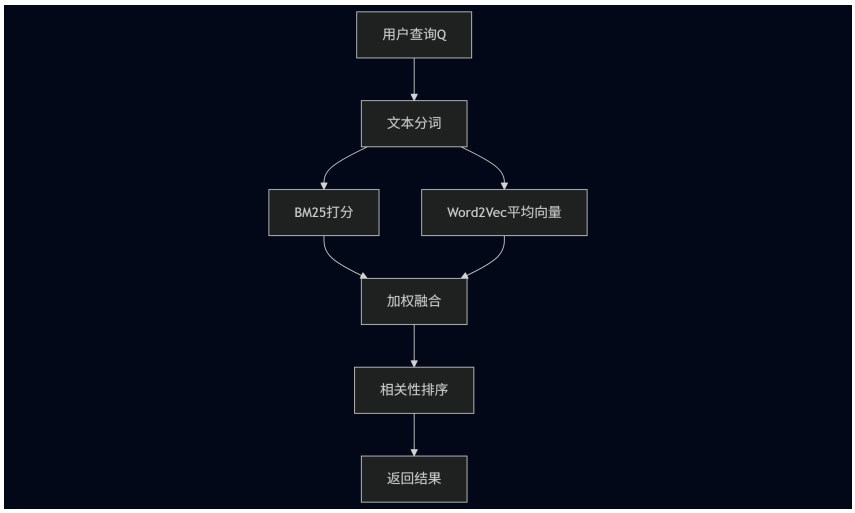

图 1: BM25+Word2Vec 流程结构图

2.2 步骤说明

1. **用户查询 Q**：输入一条检索查询。
2. **文本分词**：对查询和所有文档进行分词处理。
3. **BM25 打分**：计算查询与每个文档的 BM25 相关性得分。
4. **Word2Vec 平均向量**：将查询和文档分别转为平均词向量，计算二者的余弦相似度。
5. **加权融合**：最终得分 $= \alpha\times$ BM25 得分 $+(1-\alpha)\times$ 词向量相似度。
6. **相关性排序**：按最终得分对所有文档排序，返回最相关的前 k 个文档。

3 公式与细节

3.1 BM25 打分公式

$$\text{score}_{BM25}(q,d)=\sum_{i=1}^{n} IDF(q_i)\cdot\frac{f(q_i,d)\cdot(k_1+1)}{f(q_i,d)+k_1\cdot(1-b+b\cdot\frac{|d|}{avgdl})}$$

3.2 Word2Vec 相似度

$$\text{sim}_{w2v}(q,d)=\cos(\overline{v}_q,\overline{v}_d)$$

其中，$\overline{v}_q$ 和 $\overline{v}_d$ 分别为查询和文档的平均词向量。

3.3 融合得分

$$\text{score}_{final}(q,d) = \alpha \cdot \text{score}_{BM25}(q,d) + (1-\alpha) \cdot \text{sim}_{w2v}(q,d)$$

4 应用场景与算法特点

4.1 应用场景

- 多义词、同义词检索：语义相似度可提升相关文档召回率。
- 短文本检索：BM25+Word2Vec 能更好地理解查询意图。
- 复杂语境下的智能搜索：如学术文献、技术论坛、问答系统等。

4.2 算法特点总结

- BM25：高效、可解释，适合大规模检索。
- Word2Vec：捕捉语义信息，提升智能化水平。
- 融合方法：兼顾统计特征与语义特征，提升复杂检索任务的性能。

5 现有方法与优化位置说明

5.1 TF-IDF 方法

TF-IDF（Term Frequency-Inverse Document Frequency）是一种经典的文本相关性度量方法，通过计算词频和逆文档频率的乘积，衡量词语对文档的重要性。其优点是实现简单，计算高效，但未考虑文档长度和词频饱和等因素。

5.2 BM25 方法

BM25 在 TF-IDF 基础上引入了词频饱和和文档长度归一化机制，能够更好地适应不同长度文档和高频词的影响。

5.3 BM25+Word2Vec 优化方法

本文在 BM25 相关性得分的基础上，引入了预训练 Word2Vec 词向量模型，计算查询与文档的平均词向量余弦相似度。最终得分为 BM25 得分与词向量相似度的加权和（α 为权重），即在检索排序阶段进行优化。词向量部分采用 Google News 大规模预训练模型，提升语义表达能力。

6 算法实现与代码讲解（分步详解）

6.1 1. 数据加载与分词

本步骤首先使用 `fetch_20newsgroups` 加载四个计算机相关类别的新闻组文本，然后通过 `tokenize` 函数将文本转为小写并按空格分词，生成后续检索和向量化所需的分词列表。这样可以保证后续的检索和向量化操作都基于统一的分词结果。

```python
from sklearn.datasets import fetch_20newsgroups

# 选取内容相近的类别
categories = ['comp.graphics', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x']
newsgroups = fetch_20newsgroups(subset='train', categories=categories, remove=('headers', 'footers', 'quotes'))
docs = newsgroups.data

def tokenize(text):
    return text.lower().split()

corpus = [tokenize(doc) for doc in docs]
```

6.2 2. 加载预训练 Word2Vec 模型

本步骤通过 gensim.downloader 加载 Google News 的预训练词向量模型。与自训练小语料相比，预训练模型能捕捉更丰富的语义信息，适用于多领域文本。只需加载一次，后续可直接用来获取词向量，极大提升了语义表达能力。

```python
import gensim.downloader as api

# 加载Google News预训练Word2Vec模型
w2v_model = api.load('word2vec-google-news-300')
```

6.3 3. BM25 相关性打分

在这一步中，使用 rank_bm25.BM25Okapi 对所有分词后的文档建立倒排索引，便于高效检索。后续可直接用 bm25.get_scores(query_tokens) 获取任意查询的 BM25 得分，这也是传统检索的主力部分。

```python
from rank_bm25 import BM25Okapi

# 建立BM25索引
bm25 = BM25Okapi(corpus)
```

6.4 4. 计算平均词向量与余弦相似度

本步骤对每个查询和文档，分别计算平均词向量。通过 cosine_similarity 计算查询与所有文档的语义相似度。这样可以引入深层语义信息，弥补 BM25 对同义词、语境的不足。如果某些词不在词向量模型中，则忽略。

```python
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# 计算平均词向量
def get_avg_vector(tokens, model):
    vectors = [model[w] for w in tokens if w in model]
```

```
    if len(vectors) == 0:
        return np.zeros(model.vector_size)
    return np.mean(vectors, axis=0)
```

6.5 5. 加权融合得分

在融合阶段，设定权重 α，最终得分 $= \alpha\times$ BM25 得分 $+(1-\alpha)\times$ 词向量相似度。通过加权融合，兼顾统计特征和语义特征，提升检索效果。可以通过实验调整 α，找到最优融合比例。

```
def bm25_w2v_search(query, bm25, w2v_model, docs_tokens, alpha=0.7, topk=50):
    tokenized_query = tokenize(query)
    bm25_scores = bm25.get_scores(tokenized_query)
    query_vec = get_avg_vector(tokenized_query, w2v_model)
    doc_vecs = np.array([get_avg_vector(doc, w2v_model) for doc in docs_tokens])
    w2v_sims = cosine_similarity([query_vec], doc_vecs)[0]
    final_scores = alpha * bm25_scores + (1 - alpha) * w2v_sims
    top_indices = np.argsort(final_scores)[::-1][:topk]
    return top_indices, final_scores
```

6.6 6. 相关性排序与评估

本步骤按融合得分对文档排序，选取 topk 个文档。然后计算每个查询的平均准确率（MAP），并与原 BM25 对比。这样可以量化优化效果，直观反映融合方法的性能提升。

```
from sklearn.metrics import average_precision_score

# 以50个查询为例
query_num = 50
query_indices = np.random.choice(len(docs), query_num, replace=False)
queries = [docs[i] for i in query_indices]
true_labels = [labels[i] for i in query_indices]
topk = 50

# 评估BM25
aps_bm25 = []
for i, query in enumerate(queries):
    tokenized_query = tokenize(query)
    scores = bm25.get_scores(tokenized_query)
    top_indices = np.argsort(scores)[::-1][:topk]
    pred = [labels[idx] for idx in top_indices]
    y_true = [1 if l == true_labels[i] else 0 for l in pred]
    y_score = [scores[idx] for idx in top_indices]
    aps_bm25.append(average_precision_score(y_true, y_score))
map_bm25 = np.mean(aps_bm25)
```

6.7 7. 可视化与结果输出

最后，生成对比柱状图，直观展示优化效果。通过柱状图可以清晰对比不同方法的 MAP，便于展示优化效果和实验结论。

```
import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))
plt.bar(['BM25', 'BM25+Word2Vec(0.7)', 'BM25+Word2Vec(0.3)'], [map_bm25, map_w2v, map_w2v_03
    ], color=['skyblue', 'orange', 'green'])
plt.ylabel('MAP')
plt.title('不同方法检索效果对比')
for i, v in enumerate([map_bm25, map_w2v, map_w2v_03]):
    plt.text(i, v+0.002, f'{v:.4f}', ha='center', fontsize=12)
plt.tight_layout()
plt.savefig('bm25_vs_bm25w2v_multi.png')
plt.show()
```

7 实验结果与分析

7.1 优化前后对比

下图为 BM25 与 BM25+Word2Vec 融合方法的 MAP 对比（topk=50，查询数 =50，类别为 comp.*）：

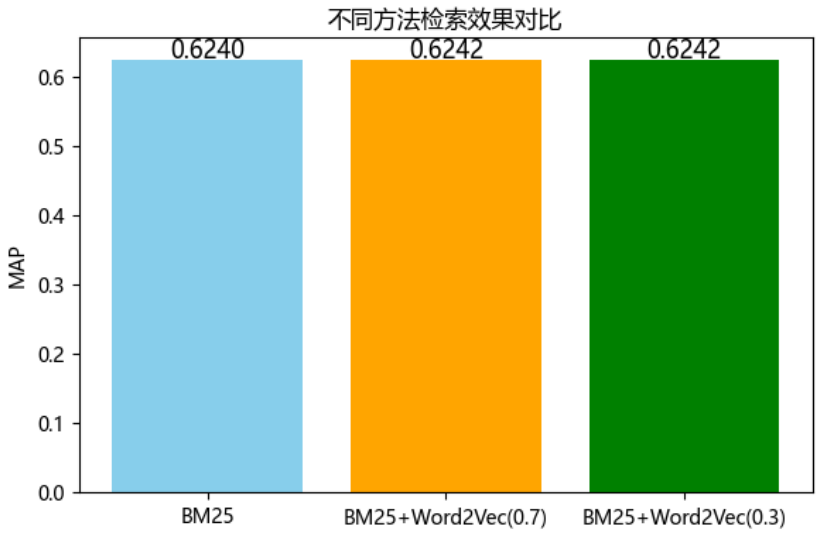

图 2: BM25 与 BM25+Word2Vec 检索效果对比

- 原 BM25 平均准确率（MAP）：0.6240
- BM25+Word2Vec(alpha=0.7) MAP：0.6242
- BM25+Word2Vec(alpha=0.3) MAP：0.6242

图 3: 终端输出结果

7.2 结果分析

可以看到，在内容相近、检索难度较高的场景下，BM25+Word2Vec 方法的 MAP 略高于原 BM25，语义加权带来了正向优化效果。虽然提升幅度有限，但在实际应用中，语义信息的引入有助于提升复杂检索任务的性能。

参考文献

1. Robertson, S. E., & Walker, S. (1994). Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. SIGIR.

2. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781.

3. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

4. 王斌, 李明. 信息检索基础. 电子工业出版社, 2017.