模糊匹配

In [ ]:
```
import json
import requests
import time
from tqdm import tqdm
from difflib import get_close_matches
from opencc import OpenCC

API_KEY = "2a2299dd95944956b69397a89113d5a7.GW5jR7NYhANOuGES"
API_URL = "https://open.bigmodel.cn/api/paas/v4/chat/completions"
HEADERS = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {API_KEY}"
}

# 创建OpenCC对象，设置从繁体到简体的转换
cc = OpenCC('t2s')

def build_prompt(question, answer, block_content=None):
    return f"""
你是一位保险领域的资深专家，请对以下问答对进行评分，满分为5分。

【参考文本块内容】：
{block_content if block_content else "无相关文本块"}

【问题】：{question}
【回答】：{answer}

请你从保险领域专业视角，根据参考文本块内容，从以下几个方面评估给定的保险领域 QA 对的质量：

1.准确性：判断问题和答案是否与保险专业知识、相关条款及给定文档内容高度一致，杜绝事实性错误。若答案与权威保险资料、行业标准规范相
2.完整性：检查问题和答案是否全面覆盖关键要点。对于保险条款解读类问题，需涵盖条款核心内容、适用条件、限制范围等；理赔流程类问题，
3.清晰度：评估问题和答案是否清晰易懂，逻辑结构是否合理。在解释保险责任判定时，推理过程应依据合理逻辑从事故情况推导到责任归属；

请按以下格式回答：
分数：X.X
评语：......
"""
```

```python
def score_qa_pair(question, answer, block_content=None):
    prompt = build_prompt(question, answer, block_content)
    data = {
        "model": "glm-4",
        "messages": [{"role": "user", "content": prompt}],
        "temperature": 0.2
    }
    try:
        response = requests.post(API_URL, headers=HEADERS, json=data)
        if response.status_code == 200:
            result = response.json()
            return result["choices"][0]["message"]["content"]
        else:
            return f"请求失败: {response.status_code} - {response.text}"
    except Exception as e:
        return f"异常: {str(e)}"

def find_best_matching_block(question, text_blocks, threshold=0.15):

    question = cc.convert(question)

    block_texts = [cc.convert(block["content"]) for block in text_blocks]
    # 使用difflib找到最匹配的文本
    matches = get_close_matches(question, block_texts, n=1, cutoff=threshold)

    if matches:
        best_match = matches[0]

        index = block_texts.index(best_match)
        return text_blocks[index]["uuid"], text_blocks[index]["content"]
    return "no_match", None

def process_testset(qa_file, text_blocks_file, output_file):

    with open(qa_file, "r", encoding="utf-8") as f:
        qa_list = json.load(f)


    with open(text_blocks_file, "r", encoding="utf-8") as f:
        text_blocks = json.load(f)
    results = []

    print("\n开始处理QA对评分和映射...\n")

    for item in tqdm(qa_list, desc="处理进度", unit="对"):
        question = item.get("question", "")
        answer = item.get("answer", "")

        # 找到最匹配的文本块
        best_block_uuid, block_content = find_best_matching_block(question, text_blocks)

        feedback = score_qa_pair(question, answer, block_content)
        item["score_feedback"] = feedback

        item["best_block_uuid"] = best_block_uuid

        results.append(item)

        time.sleep(2)  # 控制API请求频率

    with open(output_file, "w", encoding="utf-8") as f:
        json.dump(results, f, ensure_ascii=False, indent=2)

    print("\n所有问答评分和映射已完成，结果保存在：", output_file)


process_testset(
    r"D:\火力全开的项目实践\宏利 pdf 文件\测试集\测试集 2.json",
    r"D:\火力全开的项目实践\宏利 pdf 文件\数据清洗与分块\all_docs_split_400_40(2).json",
    "qa_scored_with_blocks6667.json"
)
```

```
开始处理QA对评分和映射...

处理进度：100%|██████████| 97/97 [19:01<00:00, 11.77s/对]
所有问答评分和映射已完成，结果保存在： qa_scored_with_blocks6667.json
```

TF-IDF

In [ ]:
```python
import json
import requests
import time
import re
from tqdm import tqdm
from opencc import OpenCC
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

API_KEY = "2a2299dd95944956b69397a89113d5a7.GW5jR7NYhANOuGES"
API_URL = "https://open.bigmodel.cn/api/paas/v4/chat/completions"
HEADERS = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {API_KEY}"
}

# 创建OpenCC对象，设置从繁体到简体的转换
cc = OpenCC('t2s')

def clean_text(text):
    """文本清洗函数：繁体转简体、移除标点、转换为小写"""
    text = cc.convert(text)
    text = re.sub(r"[^\w\s]", "", text)
    text = text.lower()
    return text

def build_prompt(question, answer, block_content=None):
    return f"""
你是一位保险领域的资深专家，请对以下问答对进行评分，满分为5分。

【参考文本块内容】：
{block_content if block_content else "无相关文本块"}

【问题】：{question}
【回答】：{answer}

请你从保险领域专业视角，根据参考文本块内容，从以下几个方面评估给定的保险领域 QA 对的质量：

1.准确性：判断问题和答案是否与保险专业知识、相关条款及给定文档内容高度一致，杜绝事实性错误。若答案与权威保险资料、行业标准规范相
2.完整性：检查问题和答案是否全面覆盖关键要点。对于保险条款解读类问题，需涵盖条款核心内容、适用条件、限制范围等；理赔流程类问题，
```

```
3.清晰度：评估问题和答案是否清晰易懂，逻辑结构是否合理。在解释保险责任判定时，推理过程应依据合理逻辑从事故情况推导到责任归属；阐

请按以下格式回答：
分数：X.X
评语：......
"""

def score_qa_pair(question, answer, block_content=None):
    prompt = build_prompt(question, answer, block_content)
    data = {
        "model": "glm-4",
        "messages": [{"role": "user", "content": prompt}],
        "temperature": 0.2
    }
    try:
        response = requests.post(API_URL, headers=HEADERS, json=data)
        if response.status_code == 200:
            result = response.json()
            return result["choices"][0]["message"]["content"]
        else:
            return f"请求失败: {response.status_code} - {response.text}"
    except Exception as e:
        return f"异常: {str(e)}"

def find_best_match_with_tfidf(question, text_blocks, threshold=0.1):
    """使用TF-IDF和余弦相似度找到最匹配的文本块"""
    # 准备语料库：所有文本块内容
    corpus = [clean_text(block["content"]) for block in text_blocks]
    question_cleaned = clean_text(question)

    vectorizer = TfidfVectorizer()

    try:
        tfidf_matrix = vectorizer.fit_transform([question_cleaned] + corpus)

        similarities = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:])

        best_idx = similarities.argmax()
        best_score = similarities[0, best_idx]

        if best_score > threshold:
            return text_blocks[best_idx]["uuid"], text_blocks[best_idx]["content"]
        return "no_match", None
    except ValueError:
        return "no_match", None

def process_testset(qa_file, text_blocks_file, output_file):
    with open(qa_file, "r", encoding="utf-8") as f:
        qa_list = json.load(f)

    with open(text_blocks_file, "r", encoding="utf-8") as f:
        text_blocks = json.load(f)

    results = []

    print("\n开始处理QA对评分和映射...\n")

    for item in tqdm(qa_list, desc="处理进度", unit="对"):
        question = item.get("question", "")
        answer = item.get("answer", "")

        best_block_uuid, block_content = find_best_match_with_tfidf(question, text_blocks)

        feedback = score_qa_pair(question, answer, block_content)
        item["score_feedback"] = feedback

        item["best_block_uuid"] = best_block_uuid

        results.append(item)

        time.sleep(2)  # 控制API请求频率

    with open(output_file, "w", encoding="utf-8") as f:
        json.dump(results, f, ensure_ascii=False, indent=2)

    print("\n所有问答评分和映射已完成，结果保存在：", output_file)

process_testset(
    r"D:\火力全开的项目实践\宏利 pdf 文件\测试集\测试集 2.json",
    r"D:\火力全开的项目实践\宏利 pdf 文件\数据清洗与分块\all_docs_split_400_40(2).json",
    "qa_scored_with_tfidf.json"
)
```

```
开始处理QA对评分和映射...

处理进度: 100%|██████████| 97/97 [17:24<00:00, 10.77s/对]
所有问答评分和映射已完成，结果保存在： qa_scored_with_tfidf.json
```

BM25算法

In [ ]:
```python
import json
import requests
import time
from tqdm import tqdm
from opencc import OpenCC
from rank_bm25 import BM25Okapi
import jieba

API_KEY = "2a2299dd95944956b69397a89113d5a7.GW5jR7NYhANOuGES"
API_URL = "https://open.bigmodel.cn/api/paas/v4/chat/completions"
HEADERS = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {API_KEY}"
}

# 创建OpenCC对象，设置从繁体到简体的转换
cc = OpenCC('t2s')

def tokenize(text):
    """中文分词函数"""
    return list(jieba.cut(text))

def build_prompt(question, answer, block_content=None):
    return f"""
你是一位保险领域的资深专家，请对以下问答对进行评分，满分为5分。

【参考文本块内容】：
{block_content if block_content else "无相关文本块"}

【问题】：{question}
```

```
【回答】：{answer}

请你从保险领域专业视角，根据参考文本块内容，从以下几个方面评估给定的保险领域 QA 对的质量：

1.准确性：判断问题和答案是否与保险专业知识、相关条款及给定文档内容高度一致，杜绝事实性错误。若答案与权威保险资料、行业标准规范相忄
2.完整性：检查问题和答案是否全面覆盖关键要点。对于保险条款解读类问题，需涵盖条款核心内容、适用条件、限制范围等；理赔流程类问题，
3.清晰度：评估问题和答案是否清晰易懂，逻辑结构是否合理。在解释保险责任判定时，推理过程应依据合理逻辑从事故情况推导到责任归属；阐

请按以下格式回答：
分数：X.X
评语：......
"""


def score_qa_pair(question, answer, block_content=None):
    prompt = build_prompt(question, answer, block_content)
    data = {
        "model": "glm-4",
        "messages": [{"role": "user", "content": prompt}],
        "temperature": 0.2
    }
    try:
        response = requests.post(API_URL, headers=HEADERS, json=data)
        if response.status_code == 200:
            result = response.json()
            return result["choices"][0]["message"]["content"]
        else:
            return f"请求失败：{response.status_code} - {response.text}"
    except Exception as e:
        return f"异常：{str(e)}"

def find_best_match_with_bm25(question, text_blocks, threshold=0.1):
    """使用BM25算法找到最匹配的文本块"""
    # 准备语料库：所有文本块内容（分词后）
    corpus = [tokenize(cc.convert(block["content"])) for block in text_blocks]
    question_tokenized = tokenize(cc.convert(question))

    bm25 = BM25Okapi(corpus)

    scores = bm25.get_scores(question_tokenized)
    best_idx = scores.argmax()
    best_score = scores[best_idx]
    if best_score > threshold:
        return text_blocks[best_idx]["uuid"], text_blocks[best_idx]["content"]
    return "no_match", None

def process_testset(qa_file, text_blocks_file, output_file):
    with open(qa_file, "r", encoding="utf-8") as f:
        qa_list = json.load(f)

    with open(text_blocks_file, "r", encoding="utf-8") as f:
        text_blocks = json.load(f)

    results = []

    print("\n开始处理QA对评分和映射...\n")

    for item in tqdm(qa_list, desc="处理进度", unit="对"):
        question = item.get("question", "")
        answer = item.get("answer", "")

        best_block_uuid, block_content = find_best_match_with_bm25(question, text_blocks)

        feedback = score_qa_pair(question, answer, block_content)
        item["score_feedback"] = feedback

        item["best_block_uuid"] = best_block_uuid

        results.append(item)

        time.sleep(2)  # 控制API请求频率

    with open(output_file, "w", encoding="utf-8") as f:
        json.dump(results, f, ensure_ascii=False, indent=2)

    print("\n所有问答评分和映射已完成，结果保存在：", output_file)

process_testset(
    r"D:\火力全开的项目实践\宏利 pdf 文件\测试集\测试集 2.json",
    r"D:\火力全开的项目实践\宏利 pdf 文件\数据清洗与分块\all_docs_split_400_40(2).json",
    "qa_scored_with_bm25.json"
)
```

```
开始处理QA对评分和映射...

处理进度: 100%|██████████| 97/97 [16:29<00:00, 10.20s/对]
所有问答评分和映射已完成，结果保存在:  qa_scored_with_bm25.json
```