

# RAG 技术详解与实战应用

第8讲：不止是cosine！匹配策略决定你召回的质量



# 目录



## 1. 上节回顾



## 2. 为什么要自定义Similarity



## 3. 如何自定义 Similarity



## 4. 使用自定义 Similarity 搭建 RAG 应用



## RAG 检索组件评测

### 上下文召回率:

- 评估检索结果是否覆盖所有关键信息
- 值越接近 1，召回越全面；漏检越少。

### 上下文相关性:

- 评估检索内容与查询的语义匹配程度
- 值越高，噪声越少，语义相关性越强。

## 查询重写

### 查询重写:

通过同义词补充、上下文补全、模板转换等方式，使查询更清晰具体，从而提升准确率和召回率。

### 子问题查询:

主问题拆解为多个子问题，有助于系统从不同角度理解问题并生成更全面的答案。每个子问题单独检索，最后合并答案

### 多步骤查询:

利用大模型将复杂查询拆解为多个RAG步骤，每个步骤依赖问题和上一个步骤的答案，层层推进，直至得到最终答案。

## 检索策略优化

### Node Group:

- 创建内置规则节点组和自定义Transform函数创建节点组
- 创建特殊节点组
- 构造复杂的节点组树

### 向量化:

- Embedding的使用
- 多Embedding召回

### 相似度:

- 相似度计算方法
- 相似度阈值过滤

### 重排序



# 目录



## 1. 上节回顾



## 2. 为什么要自定义Similarity



## 3. 如何自定义 Similarity



## 4. 使用自定义 Similarity 搭建 RAG 应用



# Similarity回顾

	Dot Product (点积)	Cosine Similarity (余弦相似度)	BM25 (Best Matching 25)
使用场景	向量空间模型 (如 embedding 检索)	向量空间模型, 消除向量长度影响	信息检索、关键词匹配, 经典的文本检索方法
是否归一化	否 (受向量模长影响)	是 (归一化到单位向量)	否 (基于词频和文档频率建模)
值域范围	取决于向量长度	$[-1, 1]$ (大部分嵌入为非负值时为 $[0, 1]$ )	$\geq 0$ (非负实数)
是否需要嵌入向量	是 (要求输入为稠密向量)	是 (要求输入为稠密向量)	否 (基于稀疏词项统计)
是否考虑词频/TF-IDF	否	否	是 (基于 TF、IDF 以及文档长度归一)
速度表现	快 (适合 ANN 检索)	快 (适合 ANN 检索)	慢 (不适用于大规模 ANN, 适合小规模文本库)
缺点	受向量长度影响, 无法比较不同尺度向量	无法反映词频重要性, 只看方向	不适合处理语义相似但词不同的文本, 如同义词



# 为什么要自定义Similarity



- 在 RAG 服务中，检索模块的效果直接影响生成结果的相关性与准确性。
- 不同场景对“相似度”的定义差异较大：有的任务关注关键词级别的精确匹配（如法律、医疗文档），有的强调语义层次的理解（如通用问答），而在多模态场景中，则可能涉及文本与图像或结构化数据的相似度对齐。

LazyLLM 提供了 [BM25/BM25\\_chinese](#) 和 [余弦相似度](#) 两种比较通用的相似度计算方式。但在实际应用中可能不能满足要求：

- 领域语义理解不足：BM25 基于词频，无法处理语义近义或领域术语归一问题；
- 余弦相似度过于粗粒度：在多句长文档或结构复杂文本中，局部语义匹配容易被稀释；



# 为什么要自定义Similarity

## BM25 无法理解领域语义 (近义词、术语归一)

### ➤ 背景:

用户查询医学相关的资料，使用的是专业术语的同义表达。

### ➤ 查询:

“高血糖的饮食干预方法”

### ➤ 文档候选:

“糖尿病患者应控制碳水化合物摄入，以调节血糖水平。”

### ➤ 我们的期望:

“高血糖” 实际上是 “糖尿病” 的核心症状之一。理想的检索系统应该能识别这两个术语之间的语义关系。

### ➤ 使用 BM25 或 BM25\_chinese的缺点:

BM25 是基于词频和词面匹配的，它无法理解 “高血糖” 和 “糖尿病” 在医学语境下是高度相关的，尽管它们意思接近。因此，这个候选文档的得分很可能较低，因为词面上几乎没有交集。





# 为什么要自定义Similarity

## BM25 无法理解领域语义 (近义词、术语归一)

### ➤ 背景:

用户想查询苹果的价值，但系统里面有苹果手机，也有苹果电脑。

### ➤ 查询:

“请告诉我苹果的营养价值”

### ➤ 文档候选:

“苹果可以生津止渴、美容养颜、润肠通便”

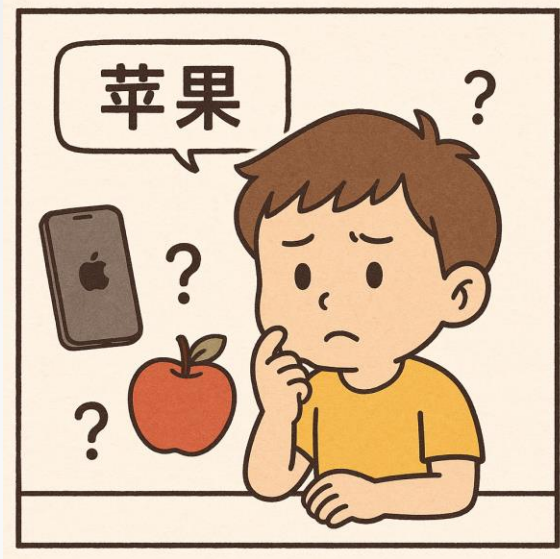
“苹果手机在外观设计上的不断探索和创新，不仅满足了用户对于美的追求，更体现了其对用户体验的极致关注”

### ➤ 我们的期望:

当问到“苹果”的营养价值时，实际上我们已经默认了它作为“水果”去搜索

### ➤ 使用 BM25 或 BM25\_chinese的缺点:

BM25 是基于词频和词面匹配的，它无法理解“苹果”同时具备“水果”和“手机”两重含义，也就是在处理多义词时会出现一些偏差。





# 为什么要自定义Similarity



## 余弦相似度在长文档中被稀释

### ➤ 背景:

用户想找一段描述“张三的项目管理能力”的内容。

### ➤ 查询:

“谁具备优秀的项目管理能力”

### ➤ 候选文档（长文档摘要）:

“张三毕业于某重点高校，具有多年软件开发经验。他参与多个大型项目的研发与部署，包括某知名 Agent 应用开发框架。他的主要职责包括代码实现、团队协作与任务分配。此外，他还在多个关键节点中承担项目负责人的角色，体现出良好的组织能力和推进力。”

### ➤ 我们的期望:





用户查询和文档中“项目负责人”，“组织能力”等部分高度相关，理想的系统应该能够识别出局部相关性，并提升该文档排名。

### ➤ 使用余弦相似度（例如 TF-IDF + 向量化）的缺点:

- 长文档中的所有词语都会被纳入向量计算，项目管理部分在整个文档中占比小；
- 余弦相似度可能被“毕业”、“开发经验”、“代码实现”等无关信息稀释，从而导致相似度偏低。



# 目录

-  1. 上节回顾
-  2. 为什么要自定义Similarity
-  3. 如何自定义 Similarity
-  4. 使用自定义 Similarity 搭建 RAG 应用



# 自定义similarity: TF-IDF

TF-IDF (Term Frequency - Inverse Document Frequency) 是一种在信息检索和文本挖掘中广泛使用的关键词提取算法。它的核心思想是：一个词如果在**一篇文档中**频繁出现，但在**所有文档中**不常见，那么它很可能是这篇文档的重要关键词。

**词频 (TF)**：表示某个词在一篇文档中出现的频率，衡量“**词对该文档的重要性**”。

$$TF(t, d) = \frac{t \text{ 在文档 } d \text{ 中出现的次数}}{\text{文档 } d \text{ 的总词数}}$$

**逆文档频率 (IDF)**：表示一个词在整个语料库中有多“稀有”，衡量“**词对所有文档的区分能力**”。

$$IDF(t) = \log\left(\frac{N}{1+df(t)}\right) \text{ 或 } IDF(t) = \log\left(\frac{1+N}{1+df(t)}\right) + 1$$

**TF-IDF**：TF 和 IDF 相乘，得到某词对文档的最终权重。

$$TF-IDF(t, d) = TF(t, d) \times IDF(t)$$

1. "大模型 正在 改变 世界"
2. "大模型 是 人工 智能 的 关键 技术"
3. "我 喜欢 编程"

我们的查询： "大模型 技术"

**TF (以文档2为例)：**

- "大模型" 出现1次  $\rightarrow TF = 1/7 \approx 0.143$
- "技术" 出现1次  $\rightarrow TF = 1/7 \approx 0.143$

**IDF (以左边公式为例)：**

- "大模型"： 出现在 文档1 和 文档2  $\rightarrow df = 2$   
 $\rightarrow IDF = \log(3 / (1+2)) = \log(1) = 0$
- "技术"： 只出现在 文档2  $\rightarrow df = 1$   
 $\rightarrow IDF = \log(3 / (1+1)) = \log(1.5) \approx 0.405$

**TF-IDF (以文档2为例)：**

- $TF-IDF(\text{"大模型"}, \text{文档2}) = 0.143 \times 0 = 0$
- $TF-IDF(\text{"技术"}, \text{文档2}) = 0.143 \times 0.405 \approx 0.058$



# 自定义similarity 代码示例

这里以 tfidf 算法来实现一个 similarity 计算的例子来说明怎么定义及应用。

```
1. from sklearn.feature_extraction.text import TfidfVectorizer
2. from sklearn.metrics.pairwise import cosine_similarity

3. def tfidf_similarity(query: str, nodes: List[DocNode], **kwargs) -> List[Tuple[DocNode, float]]:
4.     def add_space(s):
5.         return ' '.join(list(s))
6.     corpus = [add_space(node.get_text()) for node in nodes]
7.     query = add_space(query)
8.     topk = min(len(nodes), kwargs.get("topk", sys.maxsize))
9.     cv = TfidfVectorizer(tokenizer=lambda s: s.split())
10.    tfidf_matrix = cv.fit_transform(corpus+[query])
11.    query_vec = tfidf_matrix[-1]
12.    doc_vecs = tfidf_matrix[:-1]
13.    similairties = cosine_similarity(query_vec, doc_vecs).flatten()

14.    indexes = heapq.nlargest(topk, range(len(similairties)), similairties.__getitem__)
15.    results = [(nodes[i], similairties[i]) for i in indexes]
16.    return results
```

```
query = "今天天气怎么样"
```

```
candidates = [
    DocNode(text="今天阳光明媚"),
    DocNode(text="衬衫的价格是100元"),
    DocNode(text="今天天气非常好"),
    DocNode(text="我喜欢吃苹果"),
    DocNode(text="今天天气真糟糕")
]
```

```
results = tfidf_similarity(query, candidates)
```

Query: 今天天气怎么样

Scores:

今天天气非常好 -> 相似度: 0.4274

今天天气真糟糕 -> 相似度: 0.4274

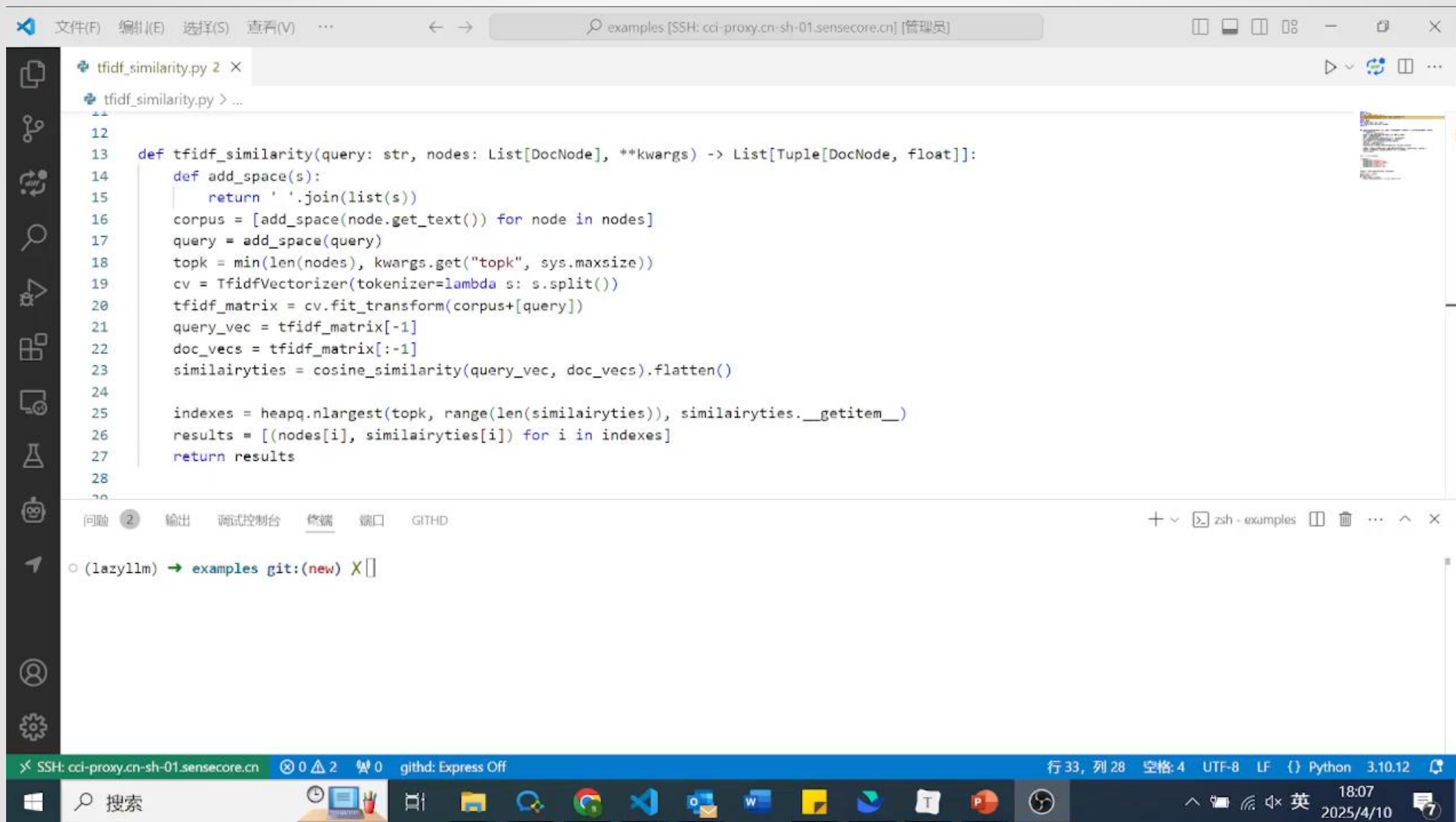
今天阳光明媚 -> 相似度: 0.2127

衬衫的价格是100元 -> 相似度: 0.0000

我喜欢吃苹果 -> 相似度: 0.0000



# 自定义similarity 代码示例



The image shows a VS Code editor window with a Python file named `tfidf_similarity.py`. The code defines a function `tfidf_similarity` that takes a query string and a list of document nodes, and returns a list of tuples containing the document node and its similarity score. The function uses `TfidfVectorizer` to create a corpus from the document nodes and the query, then calculates the cosine similarity between the query vector and the document vectors. The top `topk` results are returned.





```
12
13 def tfidf_similarity(query: str, nodes: List[DocNode], **kwargs) -> List[Tuple[DocNode, float]]:
14     def add_space(s):
15         return ' '.join(list(s))
16     corpus = [add_space(node.get_text()) for node in nodes]
17     query = add_space(query)
18     topk = min(len(nodes), kwargs.get("topk", sys.maxsize))
19     cv = TfidfVectorizer(tokenizer=lambda s: s.split())
20     tfidf_matrix = cv.fit_transform(corpus+[query])
21     query_vec = tfidf_matrix[-1]
22     doc_vecs = tfidf_matrix[:-1]
23     similairyties = cosine_similarity(query_vec, doc_vecs).flatten()
24
25     indexes = heapq.nlargest(topk, range(len(similairyties)), similairyties.__getitem__)
26     results = [(nodes[i], similairyties[i]) for i in indexes]
27     return results
28
```

Below the code editor, there is a terminal window showing the command `git:(new) X[]` being executed in a shell.

The status bar at the bottom indicates the file is `SSH: cci-proxy.cn-sh-01.sensecore.cn`, the editor is using `Python 3.10.12`, and the current position is `行 33, 列 28`.



# 目录

-  1. 上节回顾
-  2. 为什么要自定义Similarity
-  3. 如何自定义 Similarity
-  4. 使用自定义 Similarity 搭建 RAG 应用





# 使用自定义 Similarity 搭建 RAG 应用



1. @register\_similarity(mode= "text" , batch=True)
2. def tfidf\_similarity(query: str, nodes: List[DocNode], \*\*kwargs):

...

## # 定义 prompt

1. prompt = ('You will play the role of an AI Q&A assistant and complete a dialogue task. '  
'In this task, you need to provide your answer based on the given context and question.' )

## # 指定数据源

2. documents = Document(dataset\_path=os.path.join(os.getcwd(), "rag\_data"),  
embed=OnlineEmbeddingModule(), manager=False)

## # 定义 rag 数据流， 将定义的 tfidf\_similarity 注册为 similarity

3. with pipeline() as ppl:
4. ppl.prl = Retriever(documents, group\_name= "CoarseChunk" , similarity= "tfidf\_similarity" , # 注册为similarity  
similarity\_cut\_off=0.003, topk=3)
5. ppl.reranker = Reranker("ModuleReranker",  
model=OnlineEmbeddingModule(type="rerank"),  
topk=1, output\_format='content', join=True) | bind(query=ppl.input)
6. ppl.formatter = (lambda nodes, query: dict(context\_str=nodes, query=query)) | bind(query=ppl.input)
7. ppl.llm = lazyllm.OnlineChatModule(stream=False).prompt(lazyllm.ChatPrompter(prompt, extra\_keys=["context\_str"]))

## # 进行问答

8. print(ppl("人工智能的应用"))





# 使用自定义 Similarity 搭建 RAG 应用



```
文件(F) 编辑(E) 选择(S) 查看(V) ...  examples [SSH: cci-proxy.cn-sh-01.sensecore.cn] [管理员]

rag_tfidf_similarity.py 2  test_param_mode.py  test_param_descend.py  test_param_batch.py 1  rag_tfidf_similarity.py 1 X

rag_tfidf_similarity.py > ...
30     return results
31
32     prompt = ('You will play the role of an AI Q&A assistant and complete a dialogue task. '
33             'In this task, you need to provide your answer based on the given context and question.'
34             'If the given context is not sufficient to answer the question, please do not answer and explain the situation')
35
36     documents = Document(dataset_path=os.path.join(os.getcwd(), "rag_data"),
37                         embed=OnlineEmbeddingModule(), manager=False)
38     documents.create_node_group(name="sentences", transform=SentenceSplitter, chunk_size=1024, chunk_overlap=100)
39
40     with pipeline() as ppl:
41         ppl.prl = Retriever(documents, group_name="CoarseChunk", similarity="tfidf_similarity",
42                           similarity_cut_off=0.003, topk=3)
43         ppl.reranker = Reranker("ModuleReranker",
44                                model=OnlineEmbeddingModule(type="rerank"),
45                                topk=1, output_format='content', join=True) | bind(query=ppl.input)
46         ppl.formatter = (lambda nodes, query: dict(context_str=nodes, query=query)) | bind(query=ppl.input)
47         ppl.llm = lazyllm.OnlineChatModule(stream=False).prompt(lazyllm.ChatPrompter(prompt, extra_keys=["context_str"]))
48     print(ppl("全球变暖问题"))
49
```

问题 4 输出 调试控制台 终端 端口 GITHD

o (lazyllm) → examples git:(new) X

SSH: cci-proxy.cn-sh-01.sensecore.cn 0 4 0 githd: Express Off 行 43, 列 46 空格: 4 UTF-8 LF {} Python 3.10.12

搜索 9:49 2025/4/11



# 查看Retriever的Similarity得分



```
1. documents = Document(dataset_path=os.path.join(os.getcwd(), "rag_data"),  
    embed=OnlineEmbeddingModule(), manager=False)
```

# similarity 设置为 tfidf\_similarity, 可调整similarity\_cut\_off查看效果

```
2. ppl = Retriever(documents, group_name="CoarseChunk", similarity="tfidf_similarity", similarity_cut_off=0.003, topk=3)
```

# 进行召回并打印相似度分数

```
3. nodes = ppl("人工智能给传统行业带来哪些机遇与挑战")
```

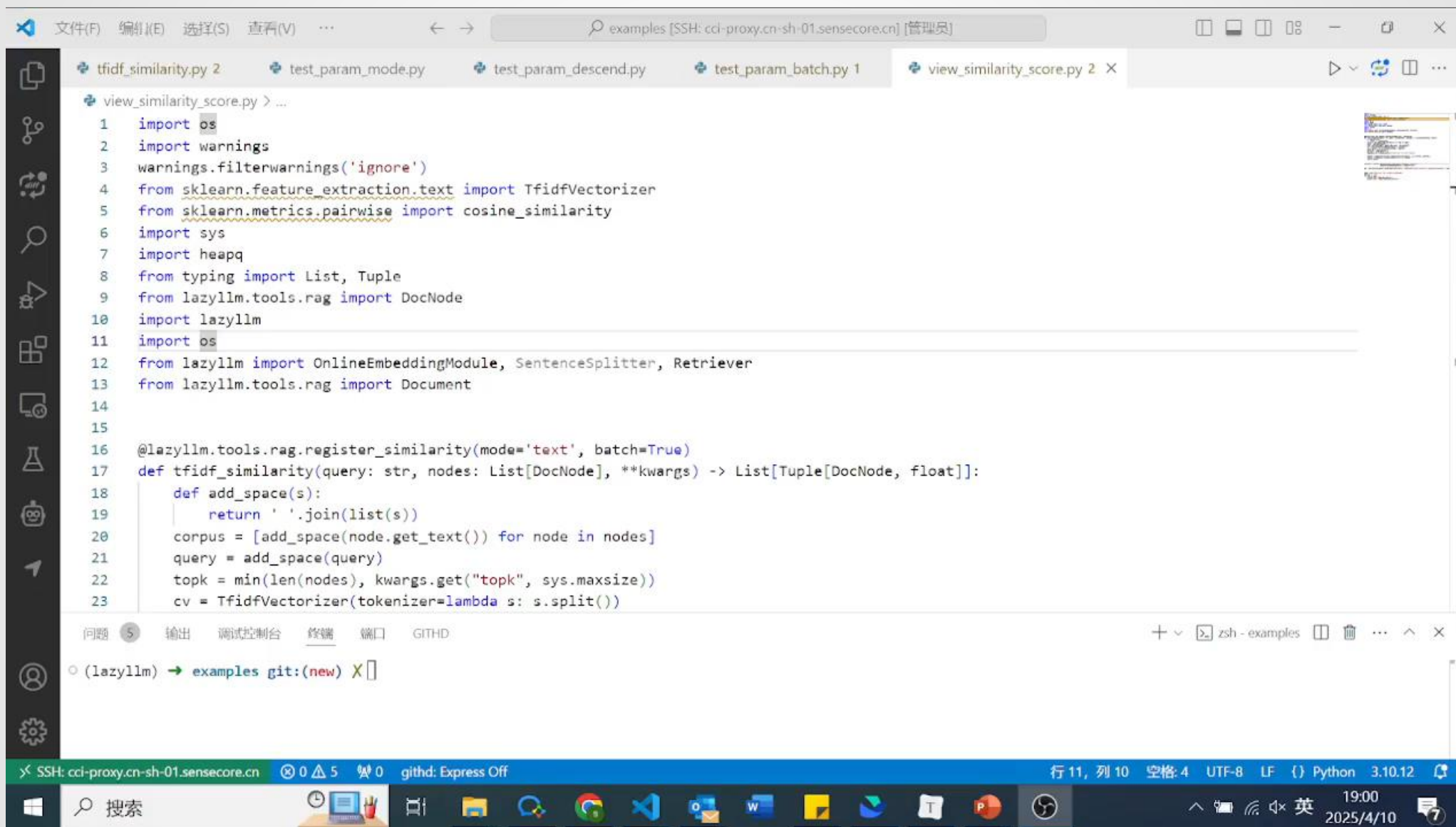
```
4. for node in nodes:
```

```
5.     print(f"node: {node.get_text()}")
```

```
6.     print(f"score: {node.similarity_score}\n")
```



# 查看Retriever的Similarity得分



```
view_similarity_score.py > ...
1  import os
2  import warnings
3  warnings.filterwarnings('ignore')
4  from sklearn.feature_extraction.text import TfidfVectorizer
5  from sklearn.metrics.pairwise import cosine_similarity
6  import sys
7  import heapq
8  from typing import List, Tuple
9  from lazyllm.tools.rag import DocNode
10 import lazyllm
11 import os
12 from lazyllm import OnlineEmbeddingModule, SentenceSplitter, Retriever
13 from lazyllm.tools.rag import Document
14
15
16 @lazyllm.tools.rag.register_similarity(mode='text', batch=True)
17 def tfidf_similarity(query: str, nodes: List[DocNode], **kwargs) -> List[Tuple[DocNode, float]]:
18     def add_space(s):
19         return ' '.join(list(s))
20     corpus = [add_space(node.get_text()) for node in nodes]
21     query = add_space(query)
22     topk = min(len(nodes), kwargs.get("topk", sys.maxsize))
23     cv = TfidfVectorizer(tokenizer=lambda s: s.split())
```

问题 输出 调试控制台 终端 端口 GITHD

o (lazyllm) → examples git:(new) X

SSH: cci-proxy.cn-sh-01.sensecore.cn 0 5 githd: Express Off 行 11, 列 10 空格: 4 UTF-8 LF {} Python 3.10.12

搜索 19:00 2025/4/10



# 引申：让函数支持作为装饰器使用

**装饰器 (Decorator)** 是 Python 中的一种高级功能，它允许你 **在不修改原函数代码的情况下**，给函数添加额外的功能（比如日志记录、性能测试、权限验证等）。

其本质上是一个 **“接受函数作为参数，并返回新函数”** 的函数。你可以把它想象成 **“函数的包装盒”** ——把函数放进去，它就会自动获得新能力。

```
def register_similarity(
    func: Optional[Callable] = None,
    mode: Optional[Literal['text', 'embedding']] = None,
    descend: bool = True, batch: bool = False
) -> Callable:
    @functools.wraps(func)
    def wrapper(query, nodes, **kwargs):
        ... ..
        return similarity
    return wrapper
```

# 使用方式1：作为函数使用

1. func1 = lazyllm.tools.rag.register\_similarity(tfidf)

# 使用方式2：作为无参数的装饰器使用

2. @lazyllm.tools.rag.register\_similarity

3. def tfidf(query: str, nodes: List[DocNode],  
 \*\*kwargs) -> List[Tuple[DocNode, float]]:

.....



# 引申：让函数支持作为装饰器使用

**装饰器 (Decorator)** 是 Python 中的一种高级功能，它允许你 **在不修改原函数代码的情况下**，给函数添加额外的功能（比如日志记录、性能测试、权限验证等）。

其本质上是一个 **“接受函数作为参数，并返回新函数”** 的函数。你可以把它想象成 **“函数的包装盒”** ——把函数放进去，它就会自动获得新能力。

```
def register_similarity(
    func: Optional[Callable] = None,
    mode: Optional[Literal['text', 'embedding']] = None,
    descend: bool = True, batch: bool = False
) -> Callable:
    def decorator(f):
        @functools.wraps(f)
        def wrapper(query, nodes, **kwargs):
            ...
            return similarity
        return wrapper
    return decorator(func) if func else decorator
```

# 使用方式1：作为函数使用

1. func1 = lazyllm.tools.rag.register\_similarity(tfidf)

# 使用方式2：作为无参数的装饰器使用

1. @lazyllm.tools.rag.register\_similarity

2. def tfidf(query: str, nodes: List[DocNode],  
 \*\*kwargs) -> List[Tuple[DocNode, float]]:

.....

# 使用方式3：作为有参数的装饰器使用

1. @lazyllm.tools.rag.register\_similarity(mode= 'text' )

2. def tfidf(query: str, nodes: List[DocNode],  
 \*\*kwargs) -> List[Tuple[DocNode, float]]:





# 自定义similarity – 参数mode



Mode, 可选的字面类型, 取值为 "text" 或 "embedding", 表示相似度计算的模式。

- "text" 表示该相似度计算方法主要是针对文本进行计算的
- "embedding" 表示该相似度计算方法主要是针对嵌入向量进行计算的

结论：输入、mode和参数三者类型必须完全一致才可以使用。

```
1. from lazyllm.tools.rag import register_similarity

# 定义一个用于计算embedding相似度的计算方法
2. def euclidean_distance(query: List[float], node: List[float]) -> float:
3.     point1 = np.array(query)
4.     point2 = np.array(node)
5.     return np.linalg.norm(point1 - point2)

# 分别注册为 mode=text 的函数和 mode=embedding 的函数
# def tfidf_similarity(query: str, nodes: List[DocNode], **kwargs)
6. func1 = register_similarity(tfidf_similarity, mode= "text" , batch=True)
7. func2 = rag.register_similarity(euclidean_distance, mode="embedding")

# 进行文本相似度计算
8. ret = func1("hello world.", [DocNode(text="hello lazyllm.")])
9. print(f"ret: {ret}")

# 进行向量相似度计算
10. ret = func2({"key": [1.0, 0.4, 2.1]},
               [DocNode(embedding={"key": [4.2, 2.1, 3.9]})])
11. print(f"ret: {ret}")
```

mode	text		embedding	
参数 输入	str	List[float]	str	List[float]
Text	✓	✗	✗	✗
embedding	✗	✗	✗	✓

实际使用时，只需要保证函数的node参数类型和注册时候的mode参数一致即可，无需关心query的类型。

# 自定义similarity – 参数mode



```
File Edit Selection View Go Run Terminal Help
Untitled (Workspace) [Administrator]

test_param_mode.py 9+, U x Settings
LazyLLM > examples > test_param_mode.py > tfidf_similarity > add_space

1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn.metrics.pairwise import cosine_similarity
3 import numpy as np
4 from scipy.linalg import norm
5 import sys
6 import heapq
7 from typing import List, Tuple
8 from lazyllm.tools.rag import DocNode
9 from typing import List
10 import numpy as np
11 import lazyllm
12
13 def tfidf_similarity(query: str, nodes: List[DocNode], **kwargs) -> List[Tuple[DocNode, float]]:
14     def add_space(s):
15         return ' '.join(list(s))
16     corpus = [add_space(node.get_text()) for node in nodes]
17     query = add_space(query)
18     topk = min(len(nodes), kwargs.get("topk", sys.maxsize))
19     cv = TfidfVectorizer(tokenizer=lambda s: s.split())
20     tfidf_matrix = cv.fit_transform(corpus+[query])
21     query_vec = tfidf_matrix[-1]
22     doc_vecs = tfidf_matrix[:-1]
23     similairyties = cosine_similarity(query_vec, doc_vecs).flatten()
24
25     indexes = heapq.nlargest(topk, range(len(similairyties)), similairyties.__getitem__)
26     results = [(nodes[i], similairyties[i]) for i in indexes]
27     return results
28
29 # 定义一个用于计算embedding相似度的计算方法
30 def euclidean_distance(query: List[float], node: List[float], **kwargs) -> float:
31     point1 = np.array(query)
32     point2 = np.array(node)
```





# 自定义similarity – 参数descend



descend, 布尔值, 指示结果是否按降序排序, 默认值为 True,

需要注意这个参数不是存在在计算相似度输出时生效, 而是在索引的时候才会生效。因为输出时会去重, 所以只是返回了topK个node, 并不一定以按序排列的

## # 定义相似度函数

1. @lazyllm.tools.rag.register\_similarity(mode= "text" , descend=True)

2. def tfidf\_similarity(query: str, nodes: List[DocNode], \*\*kwargs) -> List[Tuple[DocNode, float]]::

.....

```
3750421: 2025-04-23 20:52:58 lazyllm INFO: (lazyllm.tools.rag.default index:78) similarities: [(<Node id=f92f3afe-d0d0-490a-a750-7136c6518e3d>, 0.17713612658396552), (<Node id=4d1dede  
1-8d25-44c9-b46b-0a7ff5548100>, 0.14237098653670469), (<Node id=028a238a-2d4e-4bff-b651-dd88fd789914>, 0.13922906626180442), (<Node id=a4cad15a-a960-4204-a16c-1d3b48e2aaf2>, 0.1306178  
826379467), (<Node id=3237de15-3b3a-437d-908d-9d8b314d4c9d>, 0.1201900259305126), (<Node id=4af98c87-eb81-4881-9eb7-c503172549d5>, 0.11238925646220256), (<Node id=ad884237-19aa-49df-8  
17d-5bade0cd3fc8>, 0.11234754264502872), (<Node id=5997e769-8b93-449a-ab98-2be3634f54a7>, 0.1086605653219957), (<Node id=c0031617-0f10-4c76-ac4e-16db266b6fd0>, 0.10712413118687245), (<  
<Node id=ef531d5a-1c11-47b3-8072-2e4b1e474e4b>, 0.10424504729144217), (<Node id=3f10d25c-af6a-445a-aead-cd306c632a64>, 0.10192814980817448), (<Node id=1fea49e8-fdb7-42ba-992d-03e08356  
e71d>, 0.10111675871070003), (<Node id=2dc70992-a0f0-4054-9324-50ab3d8a973d>, 0.0996930317579351), (<Node id=616219d3-034f-49a1-b37f-4a43800f3ce2>, 0.09734583521946966), (<Node id=eed  
49016-b9c7-469c-b005-66f5be4f7a4a>, 0.08982379688462366), (<Node id=945e4c5a-53c5-4fbc-a4d5-5804b32ee0b5>, 0.08874528667670496), (<Node id=fab2797a-9d4e-4797-a606-4df2b91c0abc>, 0.086  
15517011201235), (<Node id=53ff55e0-a82e-4fd8-8d62-ed3c04c6dbdd>, 0.08572916142979266), (<Node id=9522e2cf-fe6f-4cc7-882a-c1a38ee3902f>, 0.08552962986602412), (<Node id=f492520c-7f2b-  
4422-a45a-9d91aafcca13>, 0.08243643801910652), (<Node id=5891d2bd-613e-4f09-a107-1745d48f1691>, 0.08233583745302925), (<Node id=616834b1-2bad-40b6-a799-9b469c106eb0>, 0.07967803574501  
027), (<Node id=06e49fde-458d-486c-af27-68f9d045bc79>, 0.0796283606914447), (<Node id=38b7814d-017e-4d8f-9a62-896a2d1c8193>, 0.07736697882632115), (<Node id=be7ac28f-ea45-4901-b42b-fc  
fc174eafab>, 0.07683501022804688), (<Node id=7e5da7bf-281d-4269-ac31-ef6ba2a0bbc0>, 0.07540367882754201), (<Node id=bbfc2eb2-684f-43b5-8a73-2b684cb955ef>, 0.07462729365133622), (<Node  
id=96fe6b7e-b3dd-4039-8d55-f0663b105d59>, 0.0726009494110424), (<Node id=c482cd6b-d8f4-4978-accf-fea6e968e357>, 0.07029171227021017), (<Node id=fbe992dd-dd3d-429d-9f40-5c23cdeda6ad>  
, 0.06991471065233446), (<Node id=a2fafbfb-04c4-4f36-9241-f1f9fca1e5f5>, 0.06951876691240758), (<Node id=2bba7a0c-eadd-439a-81e1-c3352794f2ec>, 0.06759124472665365), (<Node id=8af1fa1e  
-089d-48ef-9545-84ab1bb93b6e>, 0.06651583882658682), (<Node id=3e5fe53c-5f24-4c77-8066-d1a35934b6c6>, 0.06311871060427025), (<Node id=5a1d4855-727e-4aeb-aaf9-64b67abfe7d5>, 0.05835147  
047857443), (<Node id=8cf1e81d-e4a9-4db0-ab2d-bb6af6b053cd>, 0.05827593719008275), (<Node id=edaa90d4-47a9-43ba-9997-18239f678bf1>, 0.05822128969007686), (<Node id=8996750a-620a-4500-  
b030-a5b1df4104da>, 0.053140860794803214), (<Node id=9e73d111-10cc-4d9d-b982-8bd62a425468>, 0.052312782033853075), (<Node id=73433a7a-3e39-4314-9784-1008996ef979>, 0.0506887786854336)  
, (<Node id=57d2705c-be6e-405f-89f0-6fd63bb5be0c>, 0.05068188688867835), (<Node id=ba0878e4-2e80-44c7-a73f-2c85c57c00fc>, 0.04788520872026006), (<Node id=a3b3df39-dc37-40cc-9e91-a5892  
0f48453>, 0.0477963520476066), (<Node id=51ac5f27-cb42-468f-a81a-092b7c394331>, 0.047079305445043254), (<Node id=84c41a60-a9a2-44ca-8286-efd7b38f5666>, 0.046885548289832574), (<Node i  
d=7eeddeba-3fe8-4f9a-a04c-393a722f9b67>, 0.04630095707319781), (<Node id=30ce4e6d-8bd0-4cb5-953b-a5b5ffce9d09>, 0.04615912278493783), (<Node id=07944ff7-2ed4-476e-a621-85cc55143fba>  
, 0.04411976180963896), (<Node id=02844ad3-8639-4c5f-a36a-ddcc46d2a6ee>, 0.04407105593467625), (<Node id=201c3966-7134-4c9c-9419-e1b0dbb6ca8dd>, 0.04294079635617862), (<Node id=97de2fcc-  
2da5-4bb8-9706-7192c2c40002>, 0.04165291248871655), (<Node id=92643659-be62-48d7-9508-98067b07b990>, 0.04152713611630621), (<Node id=6d355c53-8942-4f57-ae3a-53c12660b723>, 0.040617848  
15943932), (<Node id=9d546a0f-5f47-45ee-b51d-708a4af66393>, 0.03866119254052912), (<Node id=3bad6161-447c-4262-878b-5ad2129e70e9>, 0.03769939774964795), (<Node id=52c23127-6f1c-4a98-9  
609-04e5800063cb>, 0.036252834977278256), (<Node id=d5b8728a-9524-4937-b8ff-7f170601a478>, 0.03217012973290899), (<Node id=ec40d3bc-e647-472a-b147-68c61e7f2a1e>, 0.031172154995678054)  
, (<Node id=3c834da1-12d9-43ab-9ba5-eb33a1690e59>, 0.03092541815300178), (<Node id=c4919d4f-d113-4cc0-99aa-a6ed5615980b>, 0.030184129687514296), (<Node id=3574a207-b306-4ed7-8aea-9622  
1de7fc48>, 0.02879056291873744), (<Node id=eaec3b27-862e-4afb-bd23-d2e5bf5c3391>, 0.027578532797523363), (<Node id=71b0d882-521d-466f-81be-f2669f9547b1>, 0.026235116312896916), (<Node  
id=e1e6971a-6529-47f7-80c1-a7617ceb0a68>, 0.024101278580890576), (<Node id=126210b6-1d39-42bb-a82f-dcf74af68ae8>, 0.0231444923476416), (<Node id=0f84f534-76ca-4e94-9dad-4a50f4997f64>  
, 0.022893735328847135), (<Node id=488d7954-67e3-48d9-b7fc-a4c4a053101d>, 0.02253629061219496), (<Node id=8a188c8b-4314-4df7-a2a4-f2c079e0c40b>, 0.020806348425922272), (<Node id=c5687  
dc9-b354-4a83-ae3b-303f29d35e36>, 0.01931519813640627), (<Node id=7f5692c3-fea6-4138-b435-42a621e7c177>, 0.016122874611967417), (<Node id=af1d3b4b-42c3-4beb-9ae8-6f9d2075ba93>, 0.0148  
10879467822486), (<Node id=15d58451-51dc-46f2-bddd-377f3d92aa28>, 0.01321824964264029), (<Node id=f472da90-b6ef-4828-a2b2-1bae24b7d3f3>, 0.010775263287670173)]
```



# 自定义similarity – 参数descend



descend, 布尔值, 指示结果是否按降序排序, 默认值为 True,

需要注意这个参数不是存在在计算相似度输出时生效, 而是在索引的时候才会生效。因为输出时会去重, 所以只是返回了topK个node, 并不一定以按序排列的

## # 定义相似度函数

1. @lazylm.tools.rag.register\_similarity(mode="embedding", descend=False)
2. def euclidean\_distance(query: List[float], node: List[float], \*\*kwargs) -> float:

.....

```
3750421: 2025-04-23 20:52:59 lazylm INFO: (lazylm.tools.rag.default index:78) similarities: [{"<Node id=eed49016-b9c7-469c-b005-66f5be4f7a4a>, 1.085723992821663}, {"<Node id=a4cad15a-a960-4204-a16c-1d3b48e2aaf2>, 1.1371467758241736}, {"<Node id=f92f3afe-d0d0-490a-a750-7136c6518e3d>, 1.1373163434791596}, {"<Node id=06e49fde-458d-486c-af27-68f9d045bc79>, 1.1540416483148312}, {"<Node id=616219d3-034f-49a1-b37f-4a43800f3ce2>, 1.1566592177393187}, {"<Node id=028a238a-2d4e-4bff-b651-dd88fd789914>, 1.177230677363927}, {"<Node id=1fea49e8-fdb7-42ba-992d-03e08356e71d>, 1.1958199209721865}, {"<Node id=ef531d5a-1c11-47b3-8072-2e4b1e474e4b>, 1.1983408126675064}, {"<Node id=4d1deded-8d25-44c9-b46b-0a7ff5548100>, 1.2236628049142304}, {"<Node id=c0031617-0f10-4c76-ac4e-16db266b6fd0>, 1.2345596957332328}, {"<Node id=3237de15-3b3a-437d-908d-9d8b314d4c9d>, 1.2367388198081208}, {"<Node id=38b7814d-017e-4d8f-9a62-896a2d1c8193>, 1.2398505444550625}, {"<Node id=2bba7a0c-eadd-439a-81e1-c3352794f2ec>, 1.241516147145497}, {"<Node id=2dc70992-a0f0-4054-9324-50ab3d8a973d>, 1.2482578028363545}, {"<Node id=fab2797a-9d4e-4797-a606-4df2b91c0abc>, 1.248802755880549}, {"<Node id=73433a7a-3e39-4314-9784-1008996ef979>, 1.2516938992293603}, {"<Node id=8a188c8b-4314-4df7-a2a4-f2c079e0c40b>, 1.2519682377024868}, {"<Node id=96fe6b7e-b3dd-4039-8d55-f0663b105d59>, 1.253969108151495}, {"<Node id=ba0878e4-2e80-44c7-a73f-2c85c57c00fc>, 1.2617715367263804}, {"<Node id=edaa90d4-47a9-43ba-9997-18239f678bf1>, 1.2627103342260622}, {"<Node id=f492520c-7f2b-4422-a45a-9d91aafcca13>, 1.2667246803760819}, {"<Node id=8996750a-620a-4500-b030-a5b1df4104da>, 1.2696251469461692}, {"<Node id=4af98c87-eb81-4881-9eb7-c503172549d5>, 1.2710658935693946}, {"<Node id=bbfc2eb2-684f-43b5-8a73-2b684cb955ef>, 1.2765418463020337}, {"<Node id=7eeddeba-3fe8-4f9a-a04c-393a722f9b67>, 1.2772607584972409}, {"<Node id=3f10d25c-af6a-445a-aead-cd306c632a64>, 1.2779152728004881}, {"<Node id=945e4c5a-53c5-4fbe-a4d5-5804b32ee0b5>, 1.278721715629854}, {"<Node id=9d546a0f-5f47-45ee-b51d-708a4af66393>, 1.2835319547890802}, {"<Node id=3e5fe53c-5f24-4c77-8066-d1a35934b6c6>, 1.2845771012174692}, {"<Node id=9522e2cf-fe6f-4cc7-882a-c1a38ee3902f>, 1.2868987281143962}, {"<Node id=7f5692c3-fea6-4138-b435-42a621e7c177>, 1.2872061480489152}, {"<Node id=02844ad3-8639-4c5f-a36a-ddcc46d2a6ee>, 1.2887753961887451}, {"<Node id=5997e769-8b93-449a-ab98-2be3634f54a7>, 1.289447295254008}, {"<Node id=8af1fa1e-089d-48ef-9545-84ab1bb93b6e>, 1.2932150152700863}, {"<Node id=57d2705c-be6e-405f-89f0-6fd63bb5be0c>, 1.2943332682567996}, {"<Node id=a2fafbfb-04c4-4f36-9241-f1f9fca1e5f5>, 1.2953936984194667}, {"<Node id=c4919d4f-d113-4cc0-99aa-a6ed5615980b>, 1.2959986803108623}, {"<Node id=97de2fcc-2da5-4bb8-9706-7192c2c40002>, 1.2977231980162973}, {"<Node id=5891d2bd-613e-4f09-a107-1745d48f1691>, 1.2978589066460242}, {"<Node id=ec40d3bc-e647-472a-b147-68c61e7f2a1e>, 1.2979688965602152}, {"<Node id=9e73d111-10cc-4d9d-b982-8bd62a425468>, 1.3014860514179305}, {"<Node id=ad884237-19aa-49df-817d-5bade0cd3fc8>, 1.3016134539508915}, {"<Node id=d5b8728a-9524-4937-b8ff-7f170601a478>, 1.3018939078754936}, {"<Node id=c482cd6b-d8f4-4978-accf-fea6e968e357>, 1.3019908058374114}, {"<Node id=15d58451-51dc-46f2-bddd-377f3d92aa28>, 1.3023876036443816}, {"<Node id=a3b3df39-dc37-40cc-9e91-a58920f48453>, 1.3027105239056704}, {"<Node id=92643659-be62-48d7-9508-98067b07b990>, 1.3047869328600972}, {"<Node id=3c834da1-12d9-43ab-9ba5-eb33a1690e59>, 1.3060539902775608}, {"<Node id=fbe992dd-dd3d-429d-9f40-5c23cdeda6ad>, 1.3064893951520629}, {"<Node id=71b0d882-521d-466f-81be-f2669f9547b1>, 1.3074567575575995}, {"<Node id=30ce4e6d-8bd0-4cb5-953b-a5b5ffce9d09>, 1.3105143409699562}, {"<Node id=7e5da7bf-281d-4269-ac31-ef6ba2a0bbc0>, 1.3125854437979616}, {"<Node id=201c3966-7134-4c9c-9419-e1bdbb6ca8dd>, 1.313043453968928}, {"<Node id=84c41a60-a9a2-44ca-8286-efd7b38f5666>, 1.3133350356871087}, {"<Node id=3bad6161-447c-4262-878b-5ad2129e70e9>, 1.3170654971868168}, {"<Node id=616834b1-2bad-40b6-a799-9b469c106eb0>, 1.3188743666622487}, {"<Node id=eaec3b27-862e-4afb-bd23-d2e5bf5c3391>, 1.3261332893541011}, {"<Node id=53ff55e0-a82e-4fd8-8d62-ed3c04c6dbdd>, 1.327987093260325}, {"<Node id=6d355c53-8942-4f57-ae3a-53c12660b723>, 1.32841968288437}, {"<Node id=8cf1e81d-e4a9-4db0-ab2d-bb6af6b053cd>, 1.3286393314698814}, {"<Node id=c5687dc9-b354-4a83-ae3b-303f29d35e36>, 1.3335999868901705}, {"<Node id=51ac5f27-cb42-468f-a81a-092b7c394331>, 1.335734836392704}, {"<Node id=af1d3b4b-42c3-4beb-9ae8-6f9d2075ba93>, 1.335774970940355}, {"<Node id=52c23127-6f1c-4a98-9609-04e5800063cb>, 1.337038823841012}, {"<Node id=5a1d4855-727e-4aeb-aaf9-64b67abfe7d5>, 1.3397381779461313}, {"<Node id=488d7954-67e3-48d9-b7fc-a4c4a053101d>, 1.3400952546003178}, {"<Node id=f472da90-b6ef-4828-a2b2-1bae24b7d3f3>, 1.34089441711869}, {"<Node id=be7ac28f-ea45-4901-b42b-fcfc174eafab>, 1.3415027657799408}, {"<Node id=0f84f534-76ca-4e94-9dad-4a50f4997f64>, 1.3424956723043207}, {"<Node id=126210b6-1d39-42bb-a82f-dcf74af68ae8>, 1.3538681819574927}, {"<Node id=07944ff7-2ed4-476e-a621-85cc55143fba>, 1.3589298685117588}, {"<Node id=e1e6971a-6529-47f7-80c1-a7617ceb0a68>, 1.3670860215965976}, {"<Node id=...>, ...}]
```





# 自定义similarity – 参数descend

```
File Edit Selection View Go Run Terminal Help
Untitled (Workspace) [Administrator]

test_param_descend.py 9+, U x
Tutorial > rag > courseware > similarity > test_param_descend.py > tfidf_similarity

1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn.metrics.pairwise import cosine_similarity
3 import numpy as np
4 from scipy.linalg import norm
5 import sys
6 import heapq
7 import os
8 from typing import List, Tuple
9 from lazyllm.tools.rag import DocNode, Document
10 from typing import List
11 import numpy as np
12 import lazyllm
13 from lazyllm import OnlineEmbeddingModule, Retriever
14
15 @lazyllm.tools.rag.register_similarity(mode="text", batch=True, descend=True)
16 def tfidf_similarity(query: str, nodes: List[DocNode], **kwargs) -> List[Tuple[DocNode, float]]:
17     def add_space(s):
18         return ' '.join(list(s))
19     corpus = [add_space(node.get_text()) for node in nodes]
20     query = add_space(query)
21     # topk = min(len(nodes), kwargs.get("topk", sys.maxsize))
22     cv = TfidfVectorizer(tokenizer=lambda s: s.split())
23     tfidf_matrix = cv.fit_transform(corpus+[query])
24     query_vec = tfidf_matrix[-1]
25     doc_vecs = tfidf_matrix[:-1]
26     simlairyties = cosine_similarity(query_vec, doc_vecs).flatten()
27
28     # indexes = heapq.nlargest(len(nodes), range(len(simlairyties)), simlairyties.__getitem__)
29     # results = [(nodes[i], simlairyties[i]) for i in indexes]
30     results = [(node, simlarity) for node, simlarity in zip(nodes, simlairyties)]
31     return results
32
```



# 自定义similarity – 参数batch

**batch**, 布尔值, 指示是否将所有的文档 (Node) 一次性给到相似度计算公式, 默认值为 **False**。

```
1. from sklearn.feature_extraction.text import TfidfVectorizer
2. from sklearn.metrics.pairwise import cosine_similarity
3. from lazyllm.tools.rag import register_similarity

4. @register_similarity(tfidf_similarity, mode= "text" , batch=True)
5. def tfidf_similarity(query: str, nodes: List[DocNode]):
6.     def add_space(s):
7.         return ' '.join(list(s))
8.     corpus = [add_space(node.get_text()) for node in nodes]
9.     query = add_space(query)
10.    topk = min(len(nodes), kwargs.get("topk", sys.maxsize))
11.    cv = TfidfVectorizer(tokenizer=lambda s: s.split())
12.    tfidf_matrix = cv.fit_transform(corpus+[query])
13.    query_vec = tfidf_matrix[-1]
14.    doc_vecs = tfidf_matrix[:-1]
15.    similairties = cosine_similarity(query_vec, doc_vecs).flatten()

16.    indexes = heapq.nlargest(topk, range(len(similairties)), similairties.__getitem__)
17.    results = [(nodes[i], similairties[i]) for i in indexes]
18.    return results
```

```
1. from lazyllm.tools.rag import register_similarity
```

# 定义一个用于计算embedding相似度的计算方法

```
1. @register_similarity(mode="embedding")
2. def euclidean_distance(query: List[float], node: List[float]):
3.     point1 = np.array(query)
4.     point2 = np.array(node)
5.     return np.linalg.norm(point1 - point2)
```

当需要计算均值、方差等**统计特征**时,  
Similarity需要看到所有的元素



# 自定义similarity – 参数batch

```
File Edit Selection View Go Run Terminal Help
Untitled (Workspace) [Administrator]

test_param_batch.py 9+, U X
Tutorial > rag > courseware > similarity > test_param_batch.py > tfidf_similarity

1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn.metrics.pairwise import cosine_similarity
3 import numpy as np
4 from scipy.linalg import norm
5 import sys
6 import heapq
7 import os
8 from typing import List, Tuple
9 from lazyllm.tools.rag import DocNode, Document
10 from typing import List
11 import numpy as np
12 import lazyllm
13 from lazyllm import OnlineEmbeddingModule, Retriever
14
15 def tfidf_similarity(query: str, nodes: List[DocNode], **kwargs) -> List[Tuple[DocNode, float]]:
16     def add_space(s):
17         return ' '.join(list(s))
18     corpus = [add_space(node.get_text()) for node in nodes]
19     query = add_space(query)
20     topk = min(len(nodes), kwargs.get("topk", sys.maxsize))
21     cv = TfidfVectorizer(tokenizer=lambda s: s.split())
22     tfidf_matrix = cv.fit_transform(corpus+[query])
23     query_vec = tfidf_matrix[-1]
24     doc_vecs = tfidf_matrix[:-1]
25     simlairyties = cosine_similarity(query_vec, doc_vecs).flatten()
26
27     indexes = heapq.nlargest(len(nodes), range(len(simlairyties)), simlairyties.__getitem__)
28     results = [(nodes[i], simlairyties[i]) for i in indexes]
29     return results
30
31 # 定义一个用于计算embedding相似度的计算方法
32 def euclidean_distance(query: List[float], node: List[float], **kwargs) -> float:
```



**感谢聆听**  
**Thanks for Listening**

