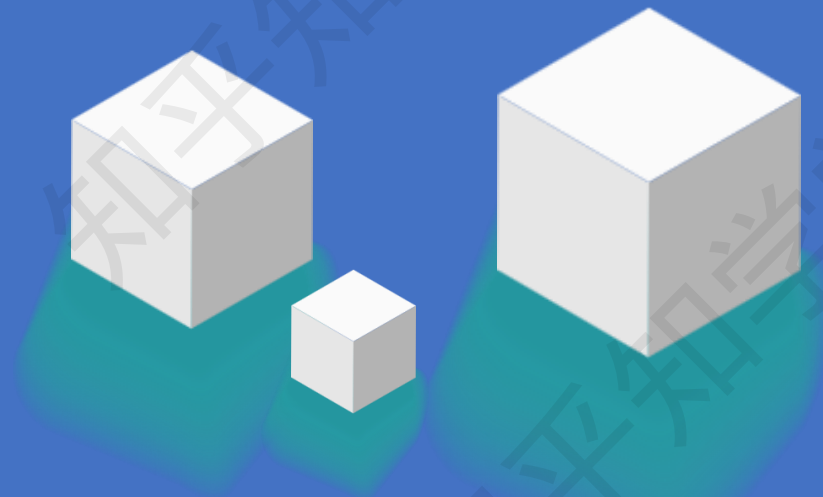
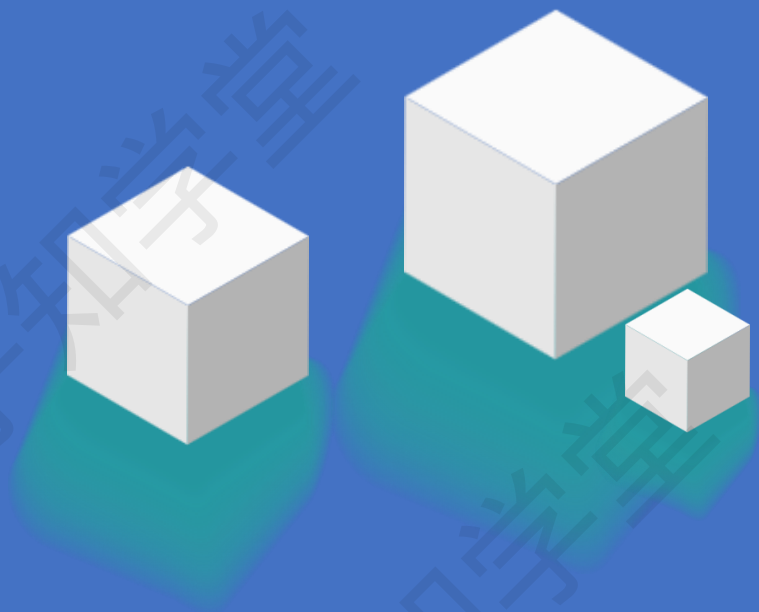


# 向量数据库使用



# >> 今天的学习目标

---

## 向量数据库的使用

- 什么是向量数据库
- FAISS, Milvus, Pinecone的特点
- 向量数据库与传统数据库的对比
- Faiss工具使用
- Case: 文本抄袭自动检测分析
- 使用DeepSeek + Faiss搭建本地知识库检索

# 什么是向量数据库

## Thinking: 什么是向量数据库?

一种专门用于存储和检索高维向量数据的数据库。它将数据（如文本、图像、音频等）通过嵌入模型转换为向量形式，并通过高效的索引和搜索算法实现快速检索。

向量数据库的核心作用是实现相似性搜索，即通过计算向量之间的距离（如欧几里得距离、余弦相似度等）来找到与目标向量最相似的其他向量。它特别适合处理非结构化数据，支持语义搜索、内容推荐等场景。

## Thinking: 如何存储和检索嵌入向量?

存储：向量数据库将嵌入向量存储为高维空间中的点，并为每个向量分配唯一标识符（ID），同时支持存储元数据。

检索：通过近似最近邻（ANN）算法（如PQ等）对向量进行索引和快速搜索。比如，FAISS和Milvus等数据库通过高效的索引结构加速检索。

# 常见的向量数据库

---

## 1. FAISS

特点：由Facebook开发，专注于高性能的相似性搜索，适合大规模静态数据集。

优势：检索速度快，支持多种索引类型。

局限性：主要用于静态数据，更新和删除操作较复杂。

## 2. Milvus

特点：开源，支持分布式架构和动态数据更新。

优势：具备强大的扩展性和灵活的数据管理功能。

## 3. Pinecone

特点：托管的云原生向量数据库，支持高性能的向量搜索。

优势：完全托管，易于部署，适合大规模生产环境。

# 向量数据库与传统数据库的对比

## 1. 数据类型

传统数据库：存储结构化数据（如表格、行、列）。

向量数据库：存储高维向量数据，适合非结构化数据。

## 2. 查询方式

传统数据库：依赖精确匹配（如=、<、>）。

向量数据库：基于相似度或距离度量（如欧几里得距离、余弦相似度）。

## 3. 应用场景

传统数据库：适合事务记录和结构化信息管理。

向量数据库：适合语义搜索、内容推荐等需要相似性计算的场景。

# Faiss工具使用

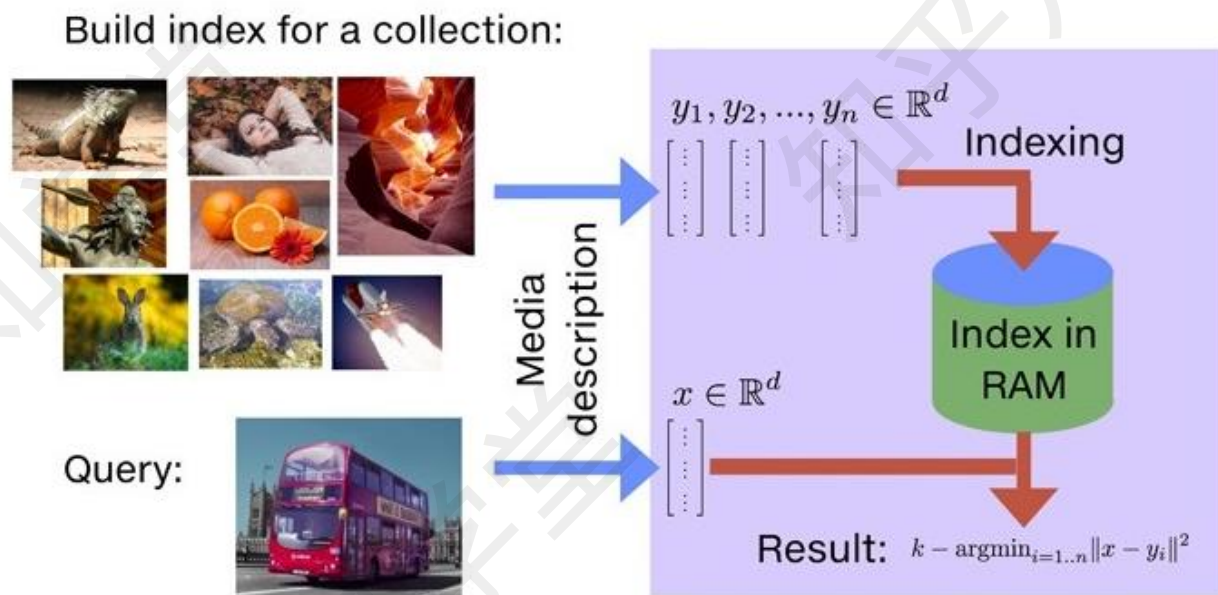
# Faiss工具

## Faiss工具

- FAIR（Facebook AI Research）团队开发的AI相似性搜索工具处理大规模d维向量近邻检索的问题
- 使用Faiss，Facebook 在十亿级数据集上创建的最邻近搜索（nearest neighbor search），速度提升了 8.5 倍
- Faiss 只支持在 RAM 上搜索
- Faiss 用 C++ 实现，支持 Python

pip install faiss-cpu

pip install faiss-gpu



# Faiss工具

---

## Faiss使用

- 常用的功能包括：索引Index，PCA降维、PQ乘积量化
- 有两个基础索引类Index、IndexBinary

## 索引选择：

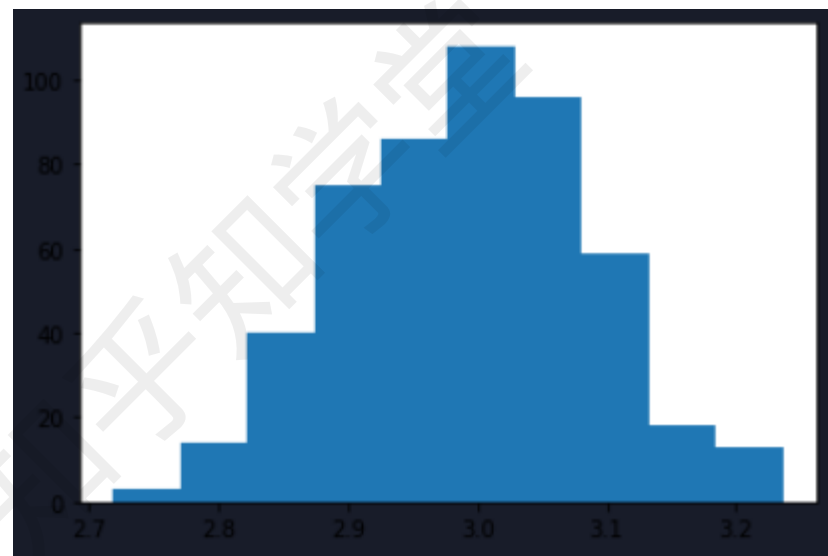
- 精度高，使用IndexFlatL2，能返回精确结果
- 速度快，使用IndexIVFFlat，首先将数据库向量通过聚类方法分割成若干子类，每个子类用类中心表示，当查询向量来临时，选择距离最近的类中心，然后在子类中应用精确查询方法，通过增加相邻的子类个数提高索引的精确度
- 内存小，使用IndexIVFPQ，可以在聚类的基础上使用PQ乘积量化进行处理



# Faiss工具

```
import numpy as np
import matplotlib.pyplot as plt
# 512维，data包含2000个向量，每个向量符合正态分布
d = 512
n_data = 2000
np.random.seed(0)
data = []
mu = 3
sigma = 0.1
for i in range(n_data):
    data.append(np.random.normal(mu, sigma, d))
data = np.array(data).astype('float32')
# 查看第6个向量是不是符合正态分布
import matplotlib.pyplot as plt
plt.hist(data[5])
plt.show()
```

```
# 创建查询向量
query = []
n_query = 10
.....
for i in range(n_query):
    query.append(np.random.normal(mu, sigma, d))
query = np.array(query).astype('float32')
```



# Faiss工具

```
import faiss
```

```
index = faiss.IndexFlatL2(d) # 构建 IndexFlatL2
```

```
print(index.is_trained) # False时需要train
```

```
index.add(data) #添加数据
```

```
print(index.ntotal) #index中向量的个数
```

```
#精确索引无需训练便可直接查询
```

```
k = 10 # 返回结果个数
```

```
query_self = data[:5] # 查询自身
```

```
dis, ind = index.search(query_self, k)
```

```
print(dis.shape) # 打印张量 (5, 10)
```

```
print(ind.shape) # 打印张量 (5, 10)
```

```
print(dis) # 升序返回每个查询向量的距离
```

```
print(ind) # 升序返回每个查询向量
```

```
True
```

```
2000
```

```
(5, 10)
```

```
(5, 10)
```

```
[[0.      8.007045 8.313328 8.53525  8.560175 8.561642 8.624167  
8.628234 8.709978 8.77004 ]
```

```
.....
```

```
[0.      8.346273 8.407202 8.462828 8.49723  8.520801 8.597084  
8.600386 8.605133 8.630594]]
```

```
[[ 0 798 879 223 981 1401 1458 1174 919 26]
```

```
[ 1 981 1524 1639 1949 1472 1162 923 840 300]
```

```
[ 2 1886 375 1351 518 1735 1551 1958 390 1695]
```

```
[ 3 1459 331 389 655 1943 1483 1723 1672 1859]
```

```
[ 4 13 715 1470 608 459 888 850 1080 1654]]
```

# Faiss工具

---

## IndexIVFFlat:

- IndexFlatL2为暴力搜索，速度慢
- IndexIVFFlat的目的是提供更快的搜索，首先将数据库向量通过聚类方法分割成若干子类，每个子类用类中心表示
- IndexIVFFlat需要一个训练的阶段，与另外一个索引quantizer有关，通过quantizer来判断属于哪个cell
- 当进行查询向量计算时，选择距离最近的类中心，然后在子类中应用精确查询方法，通过增加相邻的子类个数提高索引的精确度

- nlist，将数据库向量分割为多少了维诺空间
- quantizer = faiss.IndexFlatL2(d) # 量化器
- index.nprobe，选择n个维诺空间进行索引
- 通过改变nprobe的值，调节速度与精度

nprobe较小时，查询可能会出错，但时间开销很小

nprobe较大时，精度逐渐增大，但时间开销也增加

nprobe=nlist时，等效于IndexFlatL2索引类型。

# Faiss工具

# IndexIVFFlat快速索引

nlist = 50 # 将数据库向量分割为多少了维诺空间

k = 10

quantizer = faiss.IndexFlatL2(d) # 量化器

# METRIC\_L2计算L2距离, 或faiss.METRIC\_INNER\_PRODUCT计算内积

index = faiss.IndexIVFFlat(quantizer, d, nlist, faiss.METRIC\_L2)

# 倒排表索引类型需要训练, 训练数据集与数据库数据同分布

print(index.is\_trained)

index.train(data)

print(index.is\_trained)

index.add(data)

index.nprobe = 50 # 选择n个维诺空间进行索引

dis, ind = index.search(query, k)

print(dis)

print(ind)

False

True

[[0. 8.007045 8.313328 8.53525 8.560175 8.561642 8.624167  
8.628234 8.709978 8.77004 ]

.....

[0. 8.346273 8.407202 8.462828 8.49723 8.520801 8.597084  
8.600386 8.605133 8.630594]]

[[ 0 798 879 223 981 1401 1458 1174 919 26]

[ 1 981 1524 1639 1949 1472 1162 923 840 300]

[ 2 1886 375 1351 518 1735 1551 1958 390 1695]

[ 3 1459 331 389 655 1943 1483 1723 1672 1859]

[ 4 13 715 1470 608 459 888 850 1080 1654]]



# Faiss工具

---

## IndexIVFPQ:

- IndexFlatL2和IndexIVFFlat在index中都保存了完整的数据库向量，在数据量非常大的时候会占用太多内存（IndexFlatL2 和 IndexIVFFlat都要存储所有的向量数据）
- 对于超大规模数据集来说，可能会内存溢出，可以使用IndexIVFPQ索引来压缩向量
- 采用乘积量化方法（PQ，Product Quantizer，压缩算法）保存原始向量的有损压缩形式，所以查询结果是近似的

- nlist，将数据库向量分割为多少维诺空间
- quantizer = faiss.IndexFlatL2(d) # 量化器
- index.nprobe，选择n个维诺空间进行索引
- 通过改变nprobe的值，调节速度与精度

nprobe较小时，查询可能会出错，但时间开销很小

nprobe较大时，精度逐渐增大，但时间开销也增加

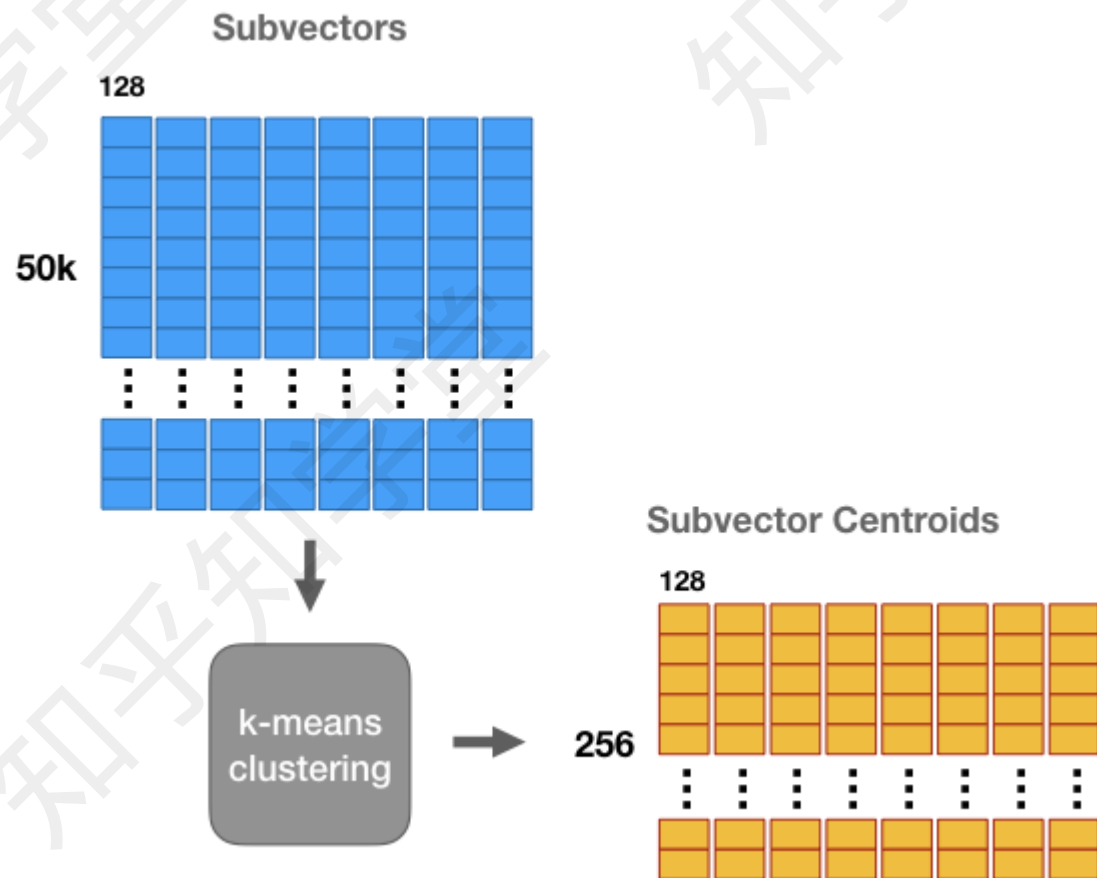
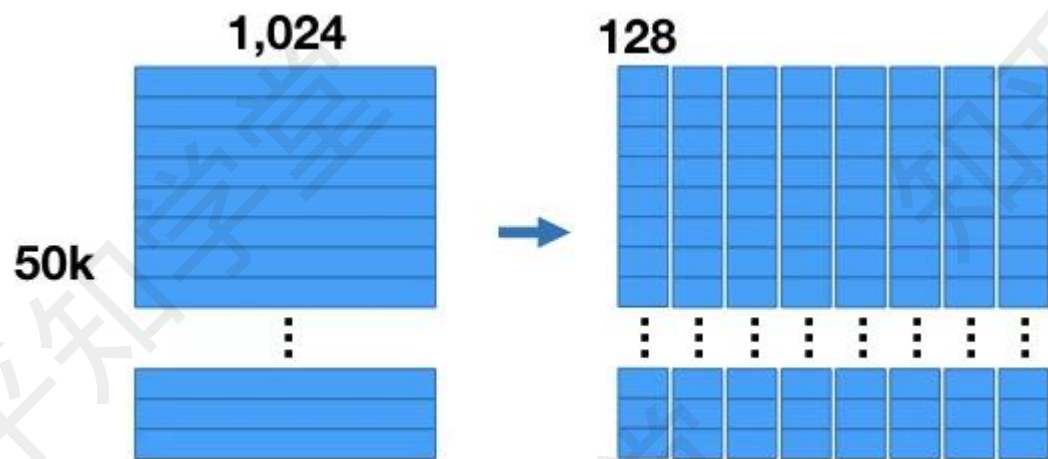
nprobe=nlist时，等效于IndexFlatL2索引类型。

# Faiss工具

乘积量化PQ (Product quantization) :

- PQ是一种建立索引的方式
- 假设原始向量是1024维，可以把它拆解成8个子向量，每个子向量128维

- 对于这8组子向量的每组子向量，使用 KMeans 方法聚成  $k=256$  类。也就是每个子向量有256个中心点(centroids)



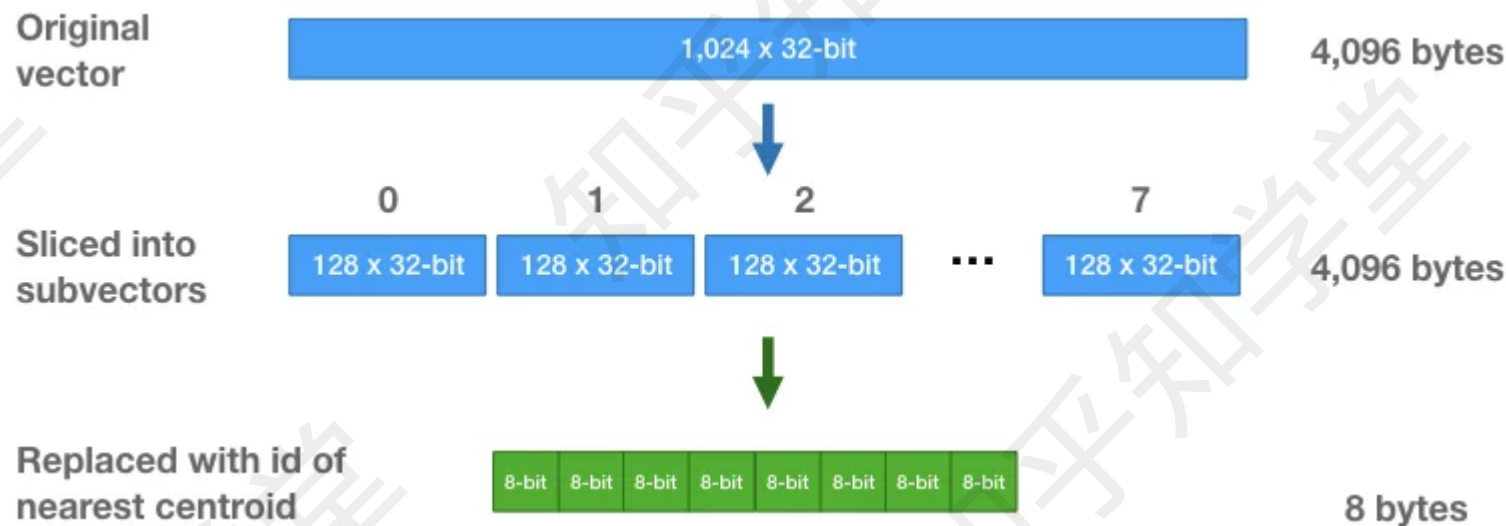
# Faiss工具

Thinking: 对于每组子向量，都有50K个，划分成了多少个桶？

如何用桶的ID来表示，原来一个子向量大小是 $128 \times 32\text{bit}$ ，现在大小是多少？

=> 聚类就是压缩

注意到每组子向量有256个中心点，我们可以中心点的ID来表示每组子向量中的每个向量。中心点的ID只需要 $\log_2 256 = 8$ 位来保存即可。这样，初始一个由32位浮点数组成的1024维向量，可以转化为8个8位整数组成



# Faiss工具

# 乘积量化索引

nlist = 50

m = 8 # 列方向划分个数，必须能被d整除

k = 10

quantizer = faiss.IndexFlatL2(d)

# 8 表示每个子向量被编码为 8 bits

index = faiss.IndexIVFPQ(quantizer, d, nlist, m, 8)

index.train(data)

index.add(data)

index.nprobe = 50

dis, ind = index.search(query\_self, k) # 查询自身

print(dis)

print(ind)



```
[[4.6366587 5.1525526 5.157734 5.1658154 5.1706343
5.1914454 5.198593 5.225469 5.2277184 5.2284985 ]
[0.18771398 4.718711 4.916692 4.927049 4.9375644
4.9418535 4.944034 4.9649167 4.967818 4.9723606 ]]
[[ 0 363 9 1552 1667 492 919 119 276 1571]
[ 1 704 1955 41 1613 1923 979 1846 1282 321]
[ 2 676 1594 482 1637 316 1917 1814 1683 1903]
[ 3 1337 1761 1144 1672 608 865 1282 1023 1181]
[ 4 1263 578 1144 1545 1400 141 717 493 1381]]
```



# Project: 文本抄袭自动检测分析

---

## To DO:

- 如果你是某新闻单位工作人员（这里假设source=新华社），为了防止其他媒体抄袭你的文章，你打算做一个抄袭自动检测分析的工具：

1) 定义可能抄袭的文章来源

2) 与原文对比定位抄袭的地方

- 原始数据：sqlResult.csv，共计89611篇

从数据库导出的文章，字段包括：id, author, source, content, feature, title, url

- 常用中文停用词：chinese\_stopwords.txt

# Project: 文本抄袭自动检测分析

id	author	source	content	feature	title	url
89617		快科技 @http://www.kkj.cn/	此外，自本周（6月12日）起，除小米手机6等15款机型外，其余机型已暂停更新发布（含开发版/体验版内测，稳定版暂不受影响），以确保工程师可以集中全部精力进行系统优化工作。有人猜测这也是将精力主要用到MIUI 9的研发之中。 MIUI 8去年5月发布，距今已有一年有余，也是时候更新换代了。当然，关于MIUI 9的确切信息，我们还是等待官方消息。	{"type":"科技","site":"cnbeta","commentNum":"37","joinNum":"20007","clickNum":"19920","shareNum":"0",.....}	小米MIUI 9首批机型曝光：共计15款	http://www.cnbeta.com/articles/tech/623597.htm
89616		快科技 @http://www.kkj.cn/	骁龙835作为唯一通过Windows 10桌面平台认证的ARM处理器，高通强调，不会因为只考虑性能而去屏蔽掉小核心。相反，他们正联手微软，找到一种适合桌面平台的、兼顾性能和功耗的完美方案。 报道称，微软已经拿到了一些新的源码，以便Windows 10更好地理解big.little架构。 资料显示，骁龙835作为一款集成了CPU、GPU、基带、蓝牙/Wi-Fi的SoC，比传统的Wintel方案可以节省至少30%的PCB空间。 按计划，今年Q4，华硕、惠普、联想将首发骁龙835 Win10电脑，预计均是二合一形态的产品。 当然，高通骁龙只是个开始，未来也许还能见到三星Exynos、联发科、华为麒麟、小米澎湃等进入Windows 10桌面平台。	{"type":"科技","site":"cnbeta","commentNum":"15","joinNum":"5522","clickNum":"5493","shareNum":"0",.....}	骁龙835在Windows 10上的性能表现有望改善	http://www.cnbeta.com/articles/tech/623599.htm
89615		快科技 @http://www.kkj.cn/	此前的一加3T搭载的是3400mAh电池，DashCharge快充规格为5V/4A。至于电池缩水，可能与刘作虎所说，一加手机5要做市面最轻薄大屏旗舰的设定有关。 按照目前掌握的资料，一加手机5拥有5.5寸1080P三星AMOLED显示屏、6G/8GB RAM，64GB/128GB ROM，双1600万摄像头，备货量“惊喜”。根据京东泄露的信息，一加5起售价是xx99元，应该是在2799/2899/2999中的某个。	{"type":"科技","site":"cnbeta","commentNum":"18","joinNum":"9445","clickNum":"9425","shareNum":"0",.....}	一加手机5细节曝光：3300mAh、充半小时用1天	http://www.cnbeta.com/articles/tech/623601.htm
89614		新华社	这是6月18日在葡萄牙中部大佩德罗冈地区拍摄的被森林大火烧毁的汽车。 新华社记者张立云摄	{"type":"国际新闻","site":"环球","commentNum":"0","joinNum":"0","clickNum":"0","shareNum":"0",.....}	葡森林火灾造成至少62人死亡 政府宣布进入紧急状态（组图）	http://world.huanqiu.com/hot/2017-06/10866126.html

# Project: 文本抄袭自动检测分析

Thinking 如何进行文本抄袭自动检测:

- 预测文章风格是否和自己一致 => 分类算法
- 根据模型预测的结果来对全量文本进行比对, 如果数量很大, => 可以先聚类降维, 比如将全部文档自动聚成k=25类
- 文本特征提取 => 计算TF-IDF
- TopN相似 => TF-IDF相似度矩阵中TopN文档
- 编辑距离editdistance => 计算句子或文章之间的编辑距离

原有方法: 分类+聚类

使用Embedding相似查找工具Faiss:

- 文本特征提取 => 计算TF-IDF
- 使用Faiss精确查找IndexFlatL2
- 向index添加数据index.add(data)
- 指定cindex=3352, 查找相似的TopN篇文章

现在方法: Faiss向量相似度

# 使用DeepSeek + Faiss搭建 本地知识库检索

# CASE: DeepSeek + Faiss搭建本地知识库检索

```
from PyPDF2 import PdfReader
from langchain.chains.question_answering import load_qa_chain
from langchain_openai import OpenAI
from langchain_community.callbacks.manager import get_openai_callback
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS
from typing import List, Tuple
```

```
def extract_text_with_page_numbers(pdf) -> Tuple[str, List[int]]:
```

```
    """
```

从PDF中提取文本并记录每行文本对应的页码

参数:

pdf: PDF文件对象

返回:

text: 提取的文本内容

page\_numbers: 每行文本对应的页码列表

```
    """
```

```
    text = ""
```

```
    page_numbers = []
```

```
    for page_number, page in enumerate(pdf.pages, start=1):
```

```
        extracted_text = page.extract_text()
```

```
        if extracted_text:
```

```
            text += extracted_text
```

```
            page_numbers.extend([page_number]
```

```
*)
```

```
            len(extracted_text.split("\n")))
```

```
        else:
```

```
            Logger.warning(f"No text found on page {page_number}.")
```

```
    return text, page_numbers
```

# CASE: DeepSeek + Faiss搭建本地知识库检索

```
def process_text_with_splitter(text: str, page_numbers: List[int]) -> FAISS:
```

```
    """
```

处理文本并创建向量存储

参数:

text: 提取的文本内容

page\_numbers: 每行文本对应的页码列表

返回:

knowledgeBase: 基于FAISS的向量存储对象

```
    """
```

```
# 创建文本分割器，用于将长文本分割成小块
```

```
text_splitter = RecursiveCharacterTextSplitter(
```

```
    separators=["\n\n", "\n", ".", " ", ""],
```

```
    chunk_size=1000,
```

```
    chunk_overlap=200,
```

```
    length_function=len,
```

```
)
```

```
# 分割文本
```

```
chunks = text_splitter.split_text(text)
```

```
#Logger.debug(f"Text split into {len(chunks)} chunks.")
```

```
print(f"文本被分割成 {len(chunks)} 个块。")
```

```
# 创建嵌入模型
```

```
embeddings = OpenAIEmbeddings()
```

```
# 从文本块创建知识库
```

```
knowledgeBase = FAISS.from_texts(chunks, embeddings)
```

```
#Logger.info("Knowledge base created from text chunks.")
```

```
print("已从文本块创建知识库。")
```

```
# 存储每个文本块对应的页码信息
```

```
knowledgeBase.page_info = {chunk: page_numbers[i] for i, chunk in enumerate(chunks)}
```

```
return knowledgeBase
```

# CASE: DeepSeek + Faiss搭建本地知识库检索

```
# 读取PDF文件
pdf_reader = PdfReader('./浦发上海浦东发展银行西安分行个金客户经理考核
办法.pdf')
# 提取文本和页码信息
text, page_numbers = extract_text_with_page_numbers(pdf_reader)
text
```

'百度文库 - 好好学习, 天天向上 \n-1 上海浦东发展银行西安分行 \n个金客户经理管理考核暂行办法 \n \n \n第一章 总 则 \n第一条 为保证我分行个金客户经理制的顺利实施, 有效调动个\n金客户经理的积极性, 促进个金业务快速、稳定地发展, 根据总行《上\n海浦东发展银行个人金融营销体系建设方案(试行)》要求, 特制定\n《上海浦东发展银行西安分行个金客户经理管理考核暂行办法(试\n行)》(以下简称本办法)。 \n第二条 个金客户经理系指各支行(营业部)从事个人金融产品\n营销与市场开拓, 为我行个人客户提供综合银行服务的我行市场人\n员。 \n第三条 考核内容分为两大类, 即个人业绩考核、 工作质量考核。 \n个人业绩包括个人资产业务、负债业务、卡业务。工作质量指个人业\n务的资产质量。 \n第四条 为规范激励规则, 客户经理的技术职务和薪资实行每年\n考核浮动。客户经理的奖金实行每季度考核浮动, 即客户经理按其考\n核内容得分与行员等级结合, 享受对应的行员等级待遇。 \n 百度文库 - 好好学习, 天天向上 \n-2 第二章 职位设置与职责 \n第五条 个金客户经理职位设置为: 客户经理助理、客户经理、 \n高级客户经理、资深客户经理。 \n第六条 个金客户经理的基本职责: \n (一) 客户开发。研究客户信息、联系与选择客户、与客户建.....

# CASE: DeepSeek + Faiss搭建本地知识库检索

```
print(f"提取的文本长度: {len(text)} 个字符。")
```

```
# 处理文本并创建知识库
```

```
knowledgeBase = process_text_with_splitter(text, page_numbers)
```

```
knowledgeBase
```

提取的文本长度: 3881 个字符。

文本被分割成 5 个块。

已从文本块创建知识库。

```
<langchain_community.vectorstores.faiss.FAISS at 0x170ab59f7d0>
```



# CASE: DeepSeek + Faiss搭建本地知识库检索

# 设置查询问题

```
query = "客户经理被投诉了, 投诉一次扣多少分"
```

```
query = "客户经理每年评聘申报时间是怎样的? "
```

```
if query:
```

```
    # 执行相似度搜索, 找到与查询相关的文档
```

```
    docs = knowledgeBase.similarity_search(query)
```

```
    # 初始化OpenAI语言模型
```

```
    llm = OpenAI(
```

```
        api_key = 'sk-
```

```
LhQP1mEVWmxMPOS17LJ2T3BlbkFJaqSx1pPfBAHlKcxh8ly5',
```

```
        base_url = 'https://openai.itit.cc/v1'
```

```
    )
```

```
    # 加载问答链
```

```
    chain = load_qa_chain(llm, chain_type="stuff")
```

```
    # 准备输入数据
```

```
    input_data = {"input_documents": docs, "question": query}
```

```
# 使用回调函数跟踪API调用成本
```

```
with get_openai_callback() as cost:
```

```
    # 执行问答链
```

```
    response = chain.invoke(input=input_data)
```

```
    print(f"查询已处理。成本: {cost}")
```

```
    print(response["output_text"])
```

```
    print("来源:")
```

```
# 记录唯一的页码
```

```
unique_pages = set()
```

```
# 显示每个文档块的来源页码
```

```
for doc in docs:
```

```
    text_content = getattr(doc, "page_content", "")
```

```
    source_page = knowledgeBase.page_info.get(
```

```
        text_content.strip(), "未知"
```

```
    )
```

```
    if source_page not in unique_pages:
```

```
        unique_pages.add(source_page)
```

```
    print(f"文本块页码: {source_page}")
```

# 打卡：创建本地Faiss知识检索



结合你的业务场景，创建本地Faiss知识检索

Step1, 收集整理知识库


Step2, 从PDF中提取文本并记录每行文本对应的页码

Step3, 处理文本并创建向量存储

Step4, 执行相似度搜索，找到与查询相关的文档

Step5, 使用问到链对用户问题进行回答

Step6, 显示每个文档块的来源页码

The background is a solid blue color. Scattered across the scene are several white 3D cubes of varying sizes. Each cube is rendered with a slight gray shadow on its bottom face and is positioned on a soft, teal-colored shadow that gives it a sense of floating or being placed on a surface. The cubes are arranged in a non-uniform pattern, with some appearing in small groups and others in isolation. The text "Thank You" is centered in the upper half, and "Using data to solve problems" is centered below it, both in a clean, white, sans-serif font. Faint, diagonal watermarks of the Chinese characters "知乎" (Zhihu) are visible across the entire image.

Thank You

Using data to solve problems