Embedding与向量数据库

>> 今天的学习目标

什么是Embedding

- 什么是Embedding

N-Gram

Word Embedding

余弦相似度计算

- Embedding模型的选择

MTEB榜单

向量维度对模型性能的影响

神奇的“俄罗斯套娃”

如何选择适合的Embedding模型

向量数据库

- 什么是向量数据库

FAISS, Elasticsearch, Milvus, Pinecone的特点

向量数据库与传统数据库的区别

如何将数据导入向量数据库

Embedding与原数据导入Faiss

不同向量数据库的功能与性能比较

什么是Embedding

为酒店建立内容推荐系统

西雅图酒店数据集：

- 下载地址： https://github.com/susanli2016/Machine-Learning-with-Python/blob/master/Seattle_Hotels.csv
- 字段：name, address, desc
- 基于用户选择的酒店，推荐相似度高的Top10个其他酒店
- 方法：计算当前酒店特征向量与整个酒店特征矩阵的余弦相似度，取相似度最大的Top-k个

name	address	desc
Hilton Garden Seattle Downtown	1821 Boren Avenue,	Located on the southern tip of Lake Union, the Hilton Garden The neighborhood is home to numerous major international comp
Sheraton Grand Seattle	1400 6th Avenue, Se	Located in the city's vibrant core, the Sheraton Grand Seattl
Crowne Plaza Seattle Downtown	1113 6th Ave, Seatt	Located in the heart of downtown Seattle, the award-winning Crowne Plaza Hotel Seattle ? Downtown offers an exceptional h
Kimpton Hotel Monaco Seattle	1101 4th Ave, Seatt	What?s near our hotel downtown Seattle location? The better question might be what?s not nearby. In addition to being one
The Westin Seattle	1900 5th Avenue,燬e	Situated amid incredible shopping and iconic attractions, The
The Paramount Hotel Seattle	724 Pine Street, Se	More than just a hotel, The Paramount Hotel Seattle summons t
Hilton Seattle	1301 6th Avenue Sea	Enjoy our central location in the heart of downtown at Hilton
Motif Seattle	1415 Fifth Ave Seat	A downtown Seattle destination hotel just steps from everywhe
Warwick Seattle	401 Lenora Street燬	In a city known for setting trends, Warwick Seattle is leadin
Four Seasons Hotel Seattle	99 Union St, Seattl	Surrounded by snow-capped mountain peaks, deep-blue waters an
W Seattle	1112 4th Ave, Seatt	Soak up the vibrant scene in the Living Room Bar and get in t
Gand Hyatt Seattle	721 Pine St, Seattl	Experience an upscale Pacific Northwest getaway at Grand Hyat
Kimpton Alexis Hotel	1007 1st Ave, Seatt	If you捀e ever had the experience of reading a book that you
Hotel Max	620 Stewart St, Sea	With a world-class art collection and a floor of curated gues
Ace Hotel Seattle	2423 1st Ave, Seatt	We fell in love with a former maritime workers' hotel in Bell
Seattle Marriott Waterfront	2100 Alaskan Way, S	Experience the best of the city when you stay at Seattle Marr
The Edgewater Hotel Seattle	2411 Alaskan Way, S	Welcome to The Edgewater Hotel, The "Best Hotel in Seattle" (
SpringHill Suites Seattle燚owntow	1800 Yale Ave, Seat	Treat yourself to a rewarding stay at the SpringHill Suites S
Fairmont Olympic Hotel	411 University St,	Downtown Seattle捏 premier luxury hotel,燽he Fairmont Olympic
La Quinta Inn & Suites Seattle Do	2224 8th Ave, Seatt	For a comfortable visit to the vibrant hub of the city, our h
Embassy Suites by Hilton Seattle	1255 S King St, Seat	Nestled in Seattle's original neighborhood Pioneer Square, th
Pan Pacific Seattle	2125 Terry Ave, Sea	When you are at Pan Pacific Seattle, you can trust us to make
Kimpton Hotel Vintage Seattle	1100 5th Ave, Seatt	Wondering what捀 around Kimpton Hotel Vintage Seattle? A bett

为酒店建立内容推荐系统

余弦相似度：

- 通过测量两个向量的夹角的余弦值来度量它们之间的相似性。
- 判断两个向量大致方向是否相同，方向相同时，余弦相似度为1；两个向量夹角为90°时，余弦相似度的值为0，方向完全相反时，余弦相似度的值为-1。
- 两个向量之间夹角的余弦值为[-1, 1]

给定属性向量A和B，A和B之间的夹角θ余弦值可以通过点积和向量长度计算得出

$$a \cdot b = |a| \cdot |b| \cos\theta \quad \Rightarrow \quad similarity = \cos(\theta) = \frac{A \cdot B}{|A||B|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

为酒店建立内容推荐系统

计算A和B的余弦相似度：

- 句子A：这个程序代码太乱，那个代码规范
- 句子B：这个程序代码不规范，那个更规范
- Step1，分词

句子A：这个/程序/代码/太乱，那个/代码/规范

句子B：这个/程序/代码/不/规范，那个/更/规范

- Step2，列出所有的词

这个，程序，代码，太乱，那个，规范，不，更

- Step3，计算词频

句子A：这个1，程序1，代码2，太乱1，那个1，规范1，不0，更0

句子B：这个1，程序1，代码1，太乱0，那个1，规范2，不1，更1

为酒店建立内容推荐系统

计算A和B的余弦相似度：

- Step4，计算词频向量的余弦相似度

句子A：（1，1，2，1，1，1，0，0）

句子B：（1，1，1，0，1，2，1，1）

$$\cos(\theta) = \frac{1\times1+1\times1+2\times1+1\times0+1\times1+1\times2+0\times1+0\times1}{\sqrt{1^2+1^2+2^2+1^2+1^2+1^2+0^2+0^2}\times\sqrt{1^2+1^2+1^2+0^2+1^2+2^2+1^2+1^2}}$$
$$= 0.738$$

结果接近1，说明句子A与句子B是相似的

为酒店建立内容推荐系统

什么是N-Gram（N元语法）：

- 基于一个假设：第n个词出现与前n-1个词相关，而与其他任何词不相关.
- N=1时为unigram，N=2为bigram，N=3为trigram
- N-Gram指的是给定一段文本，其中的N个item的序列

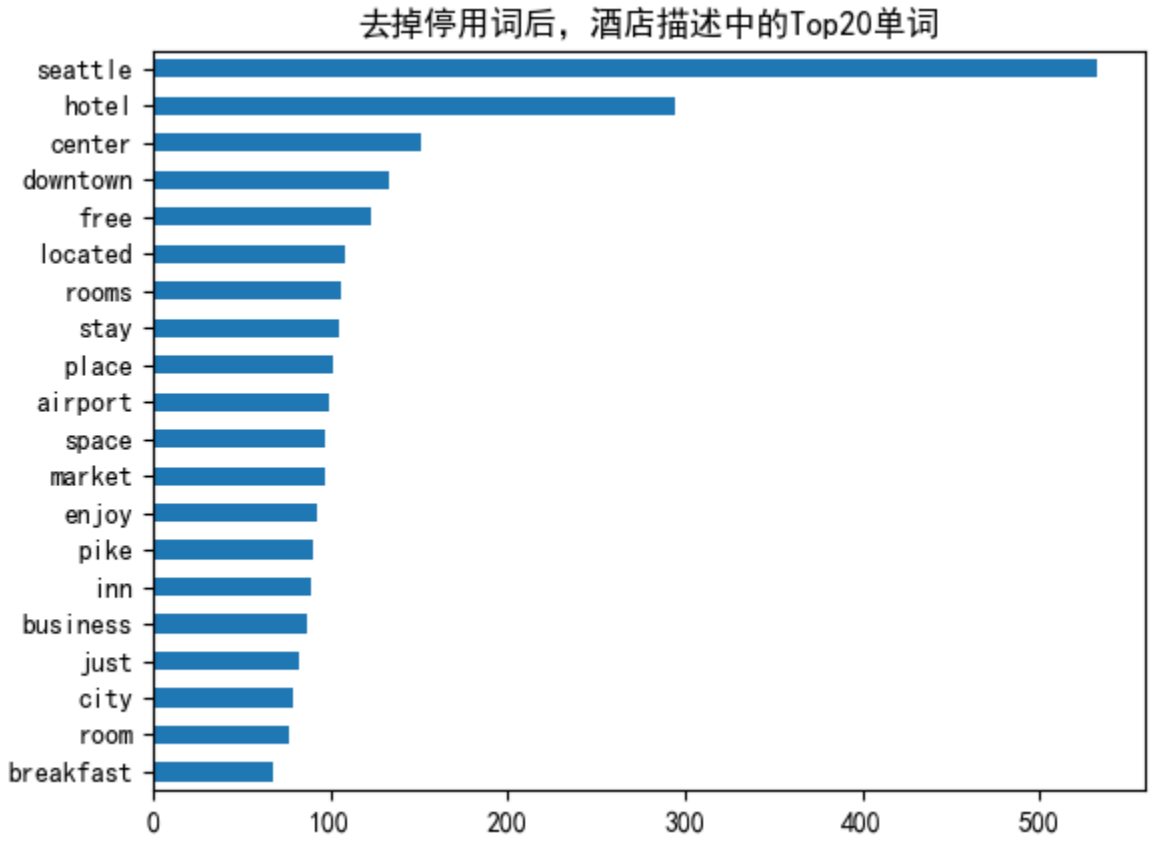
比如文本：A B C D E，对应的Bi-Gram为A B, B C, C D, D E

- 当一阶特征不够用时，可以用N-Gram做为新的特征。比如在处理文本特征时，一个关键词是一个特征，但有些情况不够用，需要提取更多的特征，采用N-Gram => 可以理解是相邻两个关键词的特征组合

如何了解事物的特征表达？N-Gram就是最基本的一种方式

为酒店建立内容推荐系统

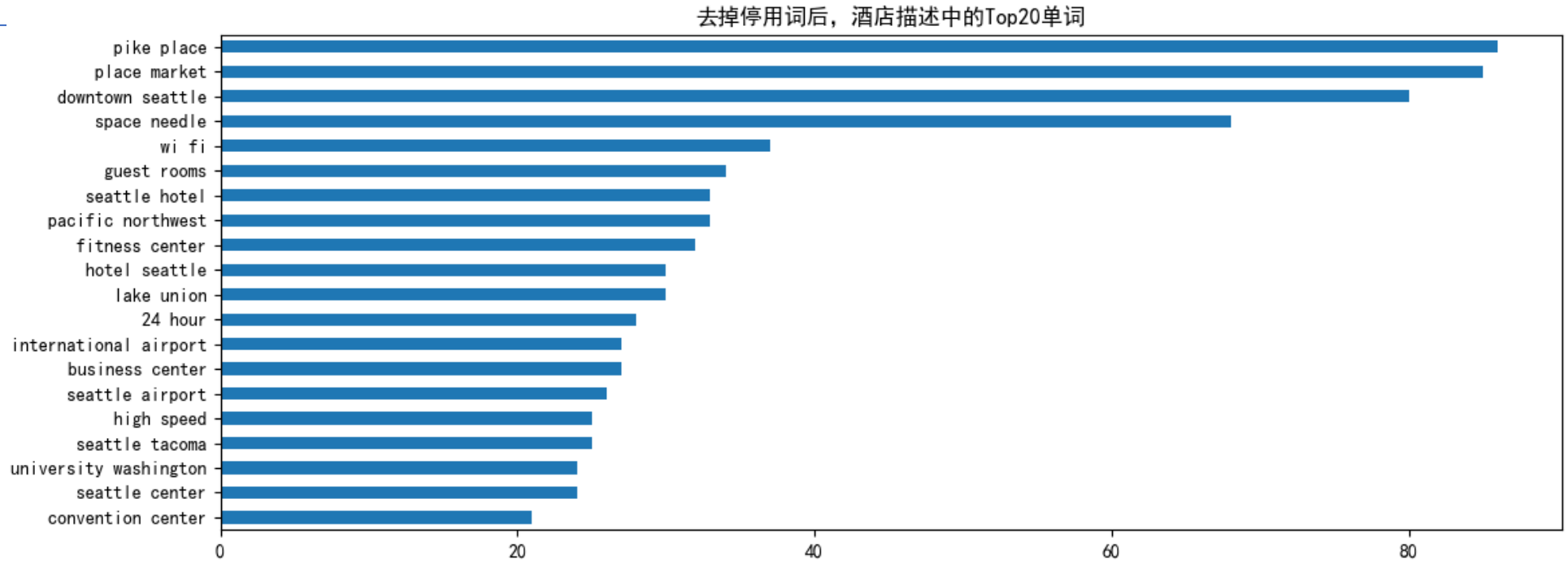
```
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签

df = pd.read_csv('Seattle_Hotels.csv', encoding="latin-1")

# 得到酒店描述中n-gram特征中的TopK个

def get_top_n_words(corpus, n=1, k=None):

    # 统计ngram词频矩阵

    vec = CountVectorizer(ngram_range=(n, n), stop_words='english').fit(corpus)

    bag_of_words = vec.transform(corpus)

    sum_words = bag_of_words.sum(axis=0)

    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]

    # 按照词频从大到小排序

    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)

    return words_freq[:k]

common_words = get_top_n_words(df['desc'], 1, 20)

df1 = pd.DataFrame(common_words, columns = ['desc' , 'count'])

df1.groupby('desc').sum()['count'].sort_values().plot(kind='barh', title='去掉停用词后，酒店描述中的Top20单词')

plt.show()
```



为酒店建立内容推荐系统

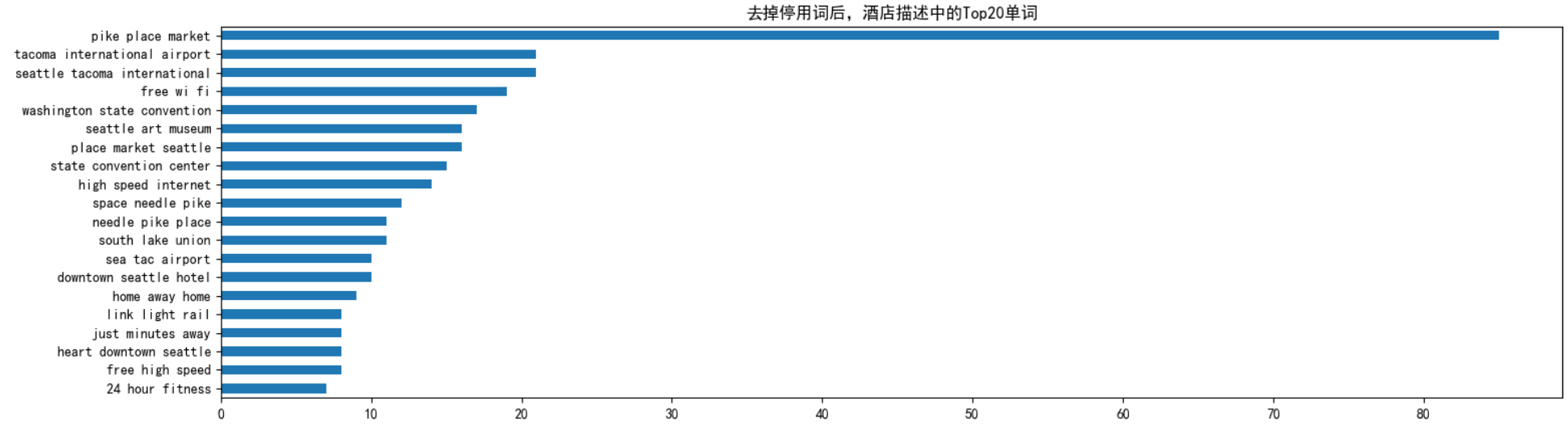
Bi-Gram

```
common_words = get_top_n_words(df['desc'], 2, 20)
```



Tri-Gram

```
common_words = get_top_n_words(df['desc'], 3, 20)
```



为酒店建立内容推荐系统

```
def clean_text(text):
    # 全部小写
    text = text.lower()
    ……
    return text
df['desc_clean'] = df['desc'].apply(clean_text)
# 使用TF-IDF提取文本特征
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 3), min_df=0, stop_words='en
tfidf_matrix = tf.fit_transform(df['desc_clean'])
print(tfidf_matrix)
print(tfidf_matrix.shape)
# 计算酒店之间的余弦相似度（线性核函数）
cosine_similarities = linear_kernel(tfidf_matrix, tfidf_matrix)
print(cosine_similarities)
```

```
  (151, 13732)  0.02288547901274695
  (151, 21971)  0.03682289049851129
  (151, 19896)  0.017052721912118395
  (151, 22212)  0.016660175897670552
  (151, 13482)  0.024448380094538605
  (151, 12132)  0.024170114517665667
  (151, 11079)  0.07169842086767955
  (151, 11324)  0.013227229761895392
(152, 26879)
```

```
[[1.         0.01161917 0.02656894 ... 0.01184587 0.00244782 0.00583589]
 [0.01161917 1.         0.015586   ... 0.01625083 0.00313105 0.00797999]
 [0.02656894 0.015586   1.         ... 0.02071479 0.00748781 0.01028037]
 ...
 [0.01184587 0.01625083 0.02071479 ... 1.         0.01066904 0.0079114 ]
 [0.00244782 0.00313105 0.00748781 ... 0.01066904 1.         0.00257955]
 [0.00583589 0.00797999 0.01028037 ... 0.0079114  0.00257955 1.        ]]
(152, 152)
```

152家酒店，之间的相似度矩阵

(1-Gram, 2-Gram, 3-Gram)

为酒店建立内容推荐系统

```
# 基于相似度矩阵和指定的酒店name，推荐TOP10酒店

def recommendations(name, cosine_similarities = cosine_similarities):

    recommended_hotels = []

    # 找到想要查询酒店名称的idx

    idx = indices[indices == name].index[0]

    print('idx=', idx)

    # 对于idx酒店的余弦相似度向量按照从大到小进行排序

    score_series = pd.Series(cosine_similarities[idx]).sort_values(ascending = False)

    # 取相似度最大的前10个（除了自己以外）

    top_10_indexes = list(score_series.iloc[1:11].index)

    # 放到推荐列表中

    for i in top_10_indexes:

        recommended_hotels.append(list(df.index)[i])

    return recommended_hotels

print(recommendations('Hilton Seattle Airport & Conference Center'))

print(recommendations('The Bacon Mansion Bed and Breakfast'))
```

```
idx= 49
['Embassy Suites by Hilton Seattle Tacoma International Airport',
'DoubleTree by Hilton Hotel Seattle Airport', 'Seattle Airport
Marriott', 'Motel 6 Seattle Sea-Tac Airport South', 'Econo Lodge
SeaTac Airport North', 'Four Points by Sheraton Downtown Seattle
Center', 'Knights Inn Tukwila', 'Econo Lodge Renton-Bellevue',
'Hampton Inn Seattle/Southcenter', 'Radisson Hotel Seattle
Airport']
idx= 116
['11th Avenue Inn Bed and Breakfast', 'Shafer Baillie Mansion Bed
& Breakfast', 'Chittenden House Bed and Breakfast', 'Gaslight
Inn', 'Bed and Breakfast Inn Seattle', 'Silver Cloud Hotel -
Seattle Broadway', 'Hyatt House Seattle', 'Mozart Guest House',
'Quality Inn & Suites Seattle Center', 'MarQueen Hotel']
```

查找和指定酒店相似度最高的Top10家酒店

为酒店建立内容推荐系统

CountVectorizer：

- 将文本中的词语转换为词频矩阵
- fit_transform：计算各个词语出现的次数
- get_feature_names： 可获得所有文本的关键词
- toarray()：查看词频矩阵的结果。

```
vec = CountVectorizer(ngram_range=(n, n), stop_words='english').fit(corpus)
bag_of_words = vec.transform(corpus)
print('feature names:')
print(vec.get_feature_names())
print('bag of words:')
print(bag_of_words.toarray())
```

```
feature names:
['00 night plus', '000 crystals marble', '000 sq ft', '000 square feet', '000
redesigned venues', '10 unique guestrooms', '100 meters away', '100 non smokin
'10best georgetown inn', '11 km emerald', '11 km seattle', '11 miles downtown'
'120 luxury guestrooms', '120 sqft 11sqm', '1200 people range', '12pm noon dai
minute walk', '15 minutes drive', '15 minutes water', '15 people doesn', '150
property', '178 guest rooms', '18 acre retreat', '18 acres natural', '188th st
landmark building', '1906 located street', '1909 cecil bacon', '1910 landmark
executive hotel', '1928 inn tucked', '1930s art deco', '1960s renamed mason',
```

```
bag of words:
[[0 0 1 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

为酒店建立内容推荐系统

TF-IDF：

- TF：Term Frequency，词频 $$TF = \frac{\text{单词次数}}{\text{文档中总单词数}}$$

一个单词的重要性和它在文档中出现的次数呈正比。

- IDF：Inverse Document Frequency，逆向文档频率

一个单词在文档中的区分度。这个单词出现的文档数越少，区分度越大，IDF越大 $$IDF = \log \frac{\text{文档总数}}{\text{单词出现的文档数}+1}$$

为酒店建立内容推荐系统

基于内容的推荐：

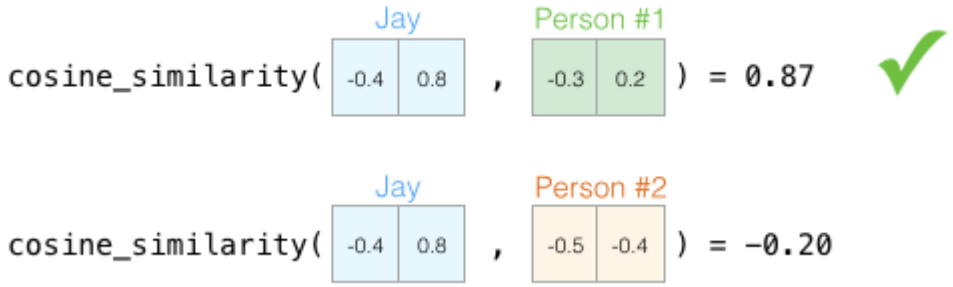
- Step1，对酒店描述（Desc）进行特征提取
  - N-Gram，提取N个连续字的集合，作为特征
  - TF-IDF，按照(min_df, max_df)提取关键词，并生成TFIDF矩阵
- Step2，计算酒店之间的相似度矩阵
  - 余弦相似度
- Step3，对于指定的酒店，选择相似度最大的Top-K个酒店进行输出

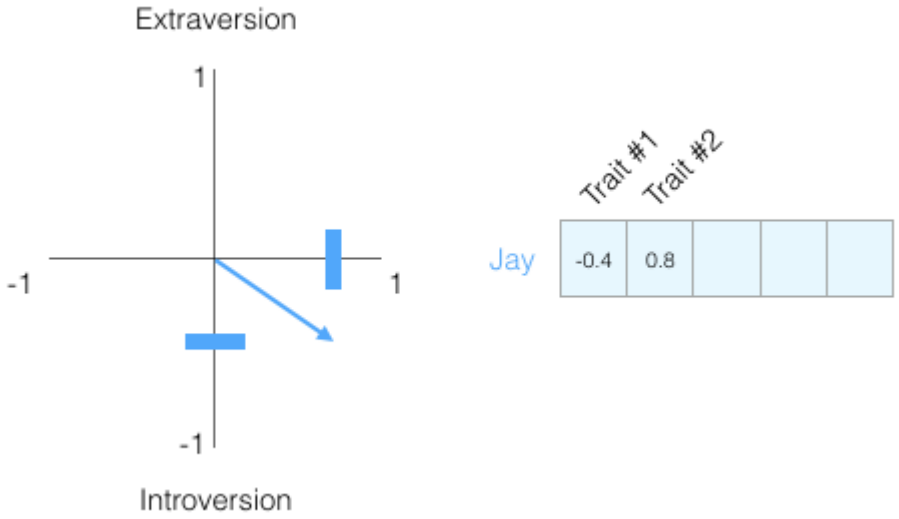
Thinking :N-Gram + TF-IDF的特征表达会让特征矩阵非常系数，计算量大，有没有更适合的方式?

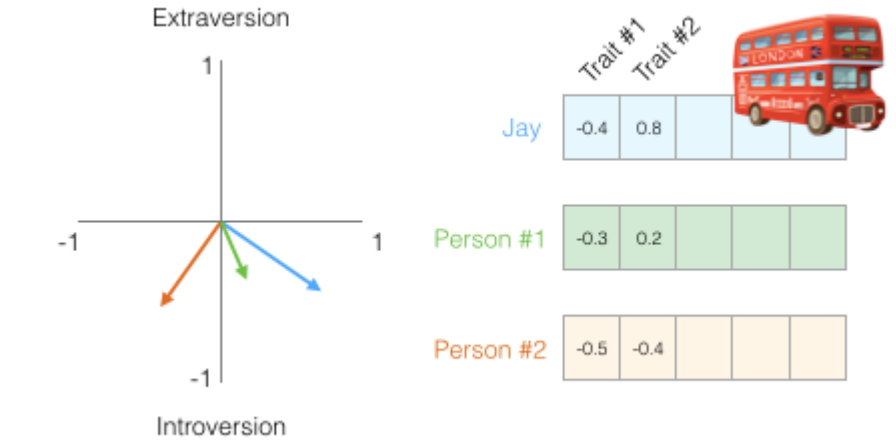
Word Embedding

什么是Embedding：

- 一种降维方式，将不同特征转换为维度相同的向量
- 离线变量转换成one-hot => 维度非常高，可以将它转换为固定size的embedding向量
- 任何物体，都可以将它转换成为向量的形式，从Trait #1到 #N
- 向量之间，可以使用相似度进行计算
- 当我们进行推荐的时候，可以选择相似度最大的







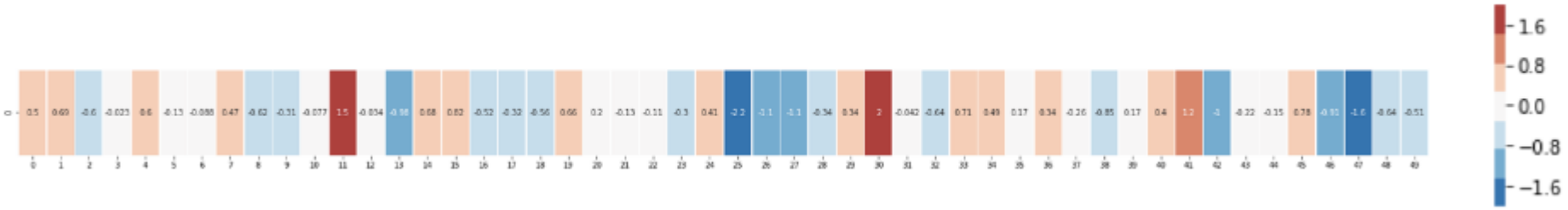
Word Embedding

将Word进行Embedding：

- 如果我们将King这个单词，通过维基百科的学习，进行GloVe向量化，可以表示成

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961
, -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 ,
-0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,
-1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```

- 这50维度的权重大小在[-2,2]，按照颜色的方式来表示

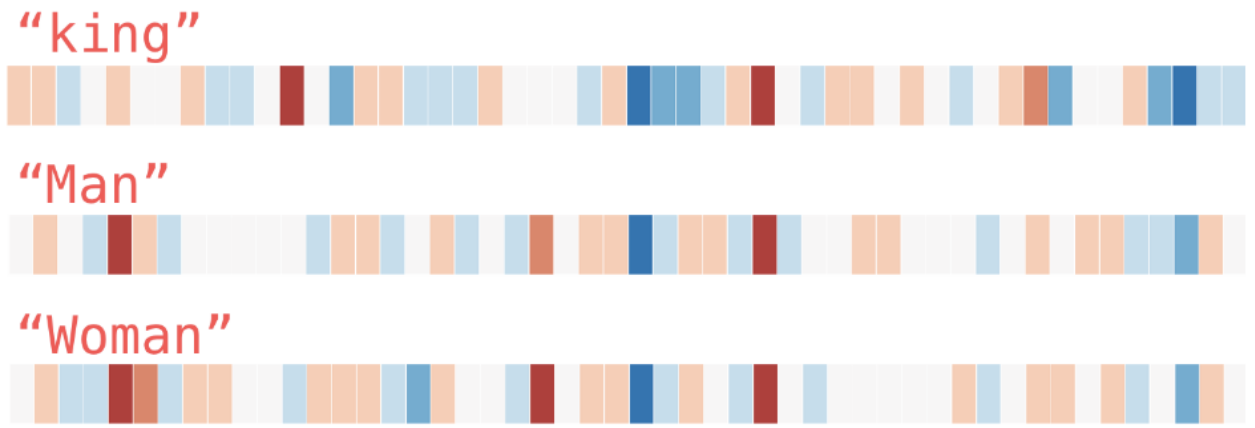


Word Embedding

将Word进行Embedding：

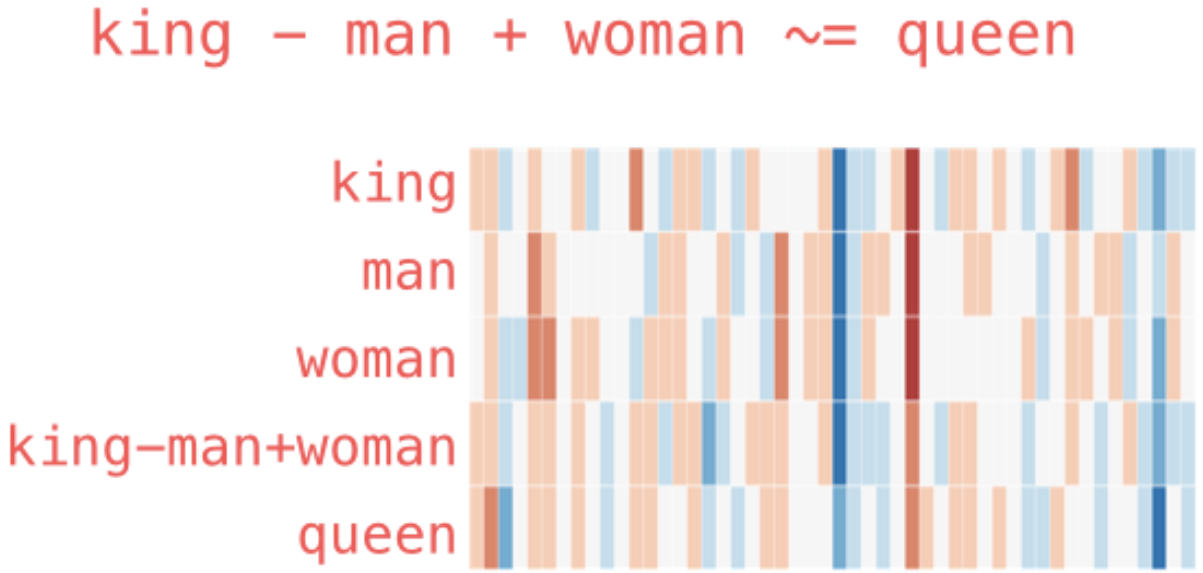
- 我们将King与其他单词进行比较，可以看到Man和Woman更相近
- 同样有了向量，我们还可以进行运算

king-man+woman与queen的相似度最高




```
model.most_similar(positive=["king","woman"], negative=["man"])
```

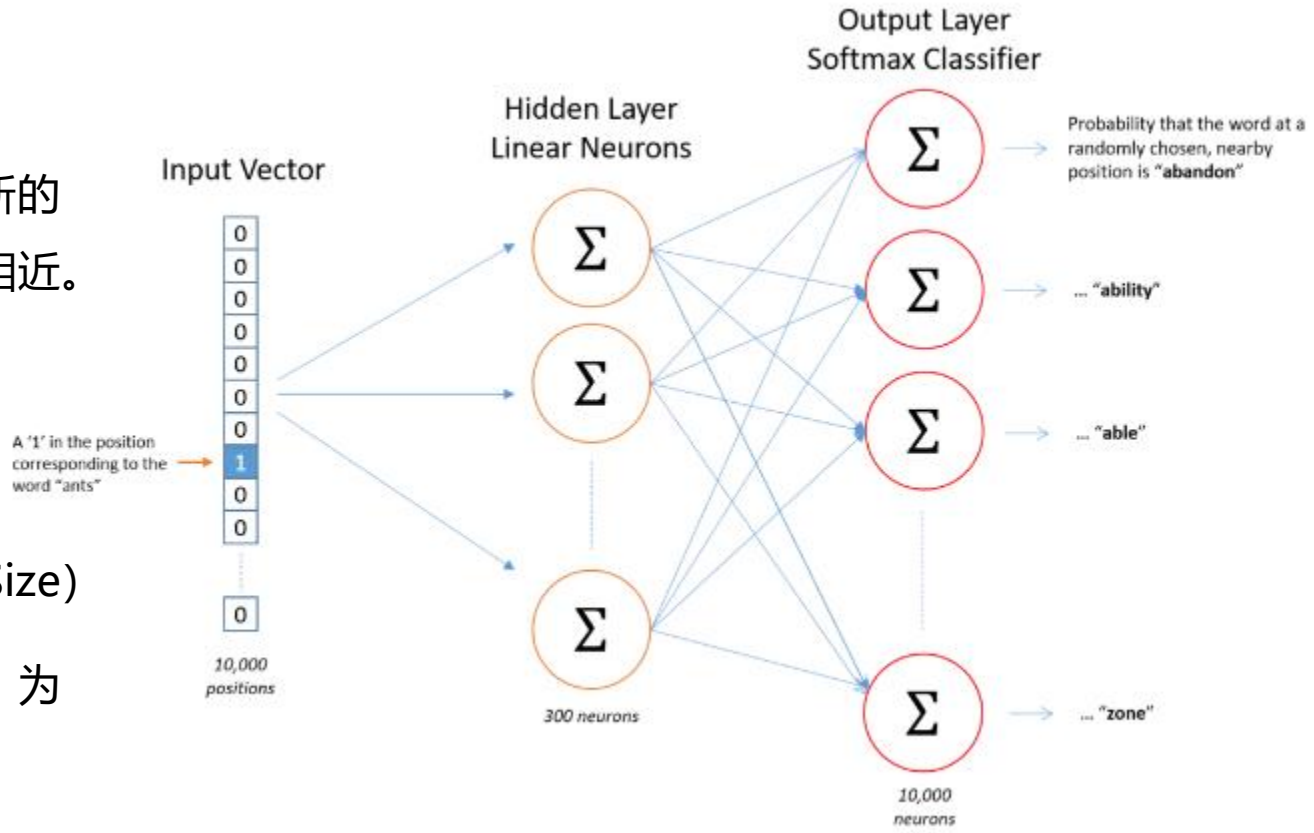
```
[('queen', 0.8523603677749634),
 ('throne', 0.7664333581924438),
 ('prince', 0.7592144012451172),
 ('daughter', 0.7473883032798767),
 ('elizabeth', 0.7460219860076904),
 ('princess', 0.7424570322036743),
 ('kingdom', 0.7337411642074585),
 ('monarch', 0.721449077129364),
 ('eldest', 0.7184862494468689),
 ('widow', 0.7099430561065674)]
```



Word Embedding

Word2Vec：

- 通过Embedding，把原先词所在空间映射到一个新的空间中去，使得语义上相似的单词在该空间内距离相近。
- Word Embedding => 学习隐藏层的权重矩阵
- 输入测是one-hot编码
- 隐藏层的神经元数量为hidden_size（Embedding Size）
- 对于输入层和隐藏层之间的权值矩阵W，大小为[vocab_size, hidden_size]
- 输出层为[vocab_size]大小的向量，每一个值代表着输出一个词的概率

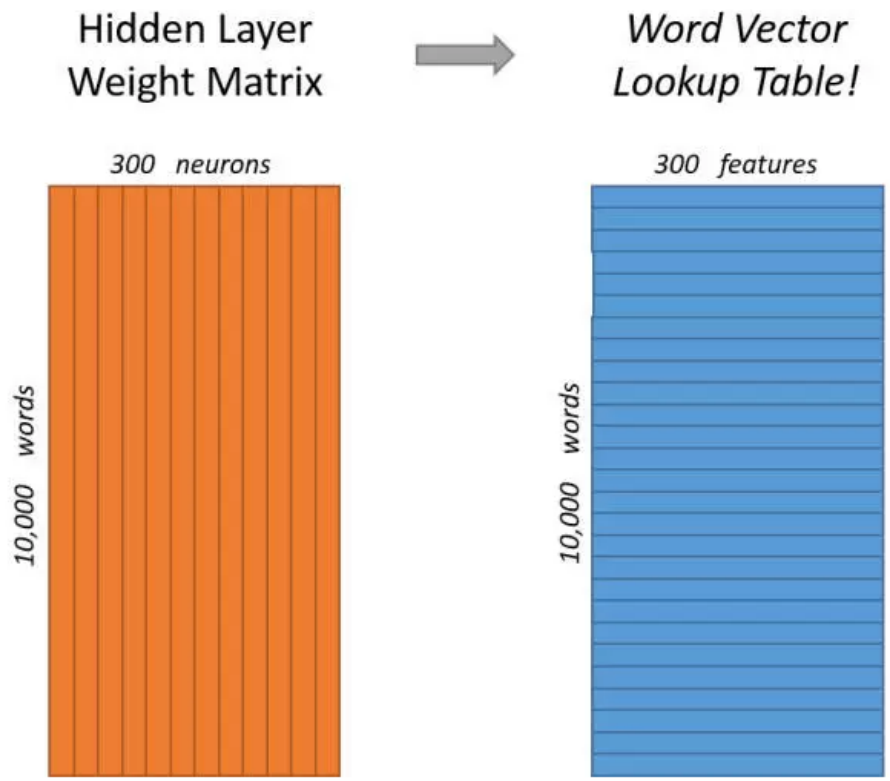


Word Embedding

对于输入的one-hot编码：

- 在矩阵相乘的时候，选取出矩阵中的某一行，而这一行就是输入词语的word2vec表示
- 隐含层的节点个数 = 词向量的维数
- 隐层的输出是每个输入单词的Word Embedding
- word2vec，实际上就是一个查找表

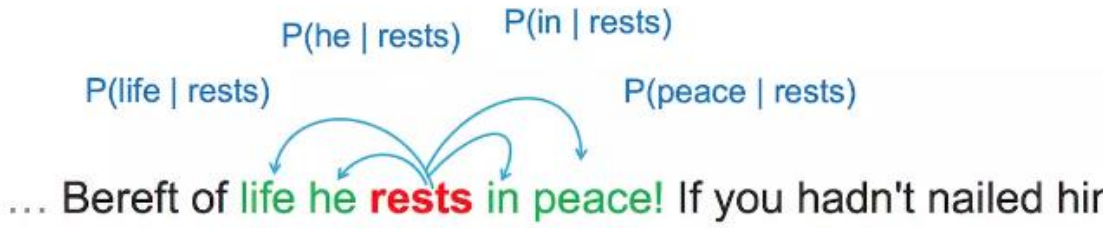
$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$



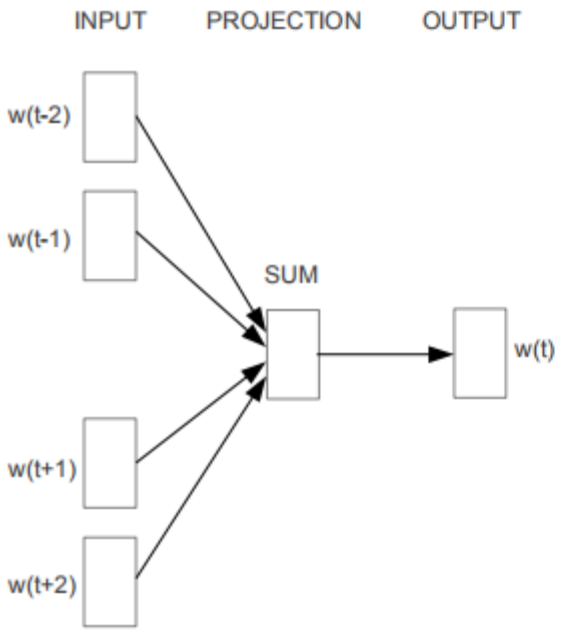
Word Embedding

Word2Vec的两种模式：

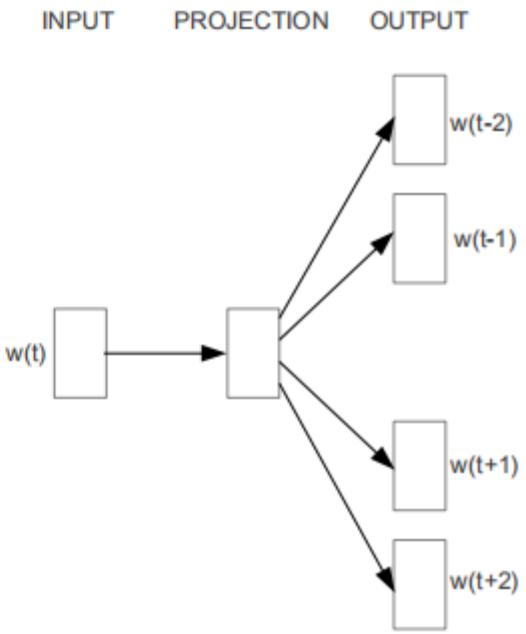
- Skip-Gram，给定input word预测上下文



- CBOW，给定上下文，预测input word（与Skip-Gram相反）



CBOW



Skip-gram

Word2Vec工具

Gensim工具

- pip install gensim
- 开源的Python工具包
- 可 以 从非结构化文本中，无监督地学习到隐层的主题向量表达
- 每一个向量变换的操作都对应着一个主题模型
- 支持TF-IDF，LDA, LSA, word2vec等多种主题模型算法

使用方法：

- 建立词向量模型：word2vec.Word2Vec(sentences)

window,句子中当前单词和被预测单词的最大距离

min_count,需要训练词语的最小出现次数，默认为5

size,向量维度，默认为100

worker,训练使用的线程数，默认为1即不使用多线程

- 模型保存 model.save(fname)
- 模型加载 model.load(fname)

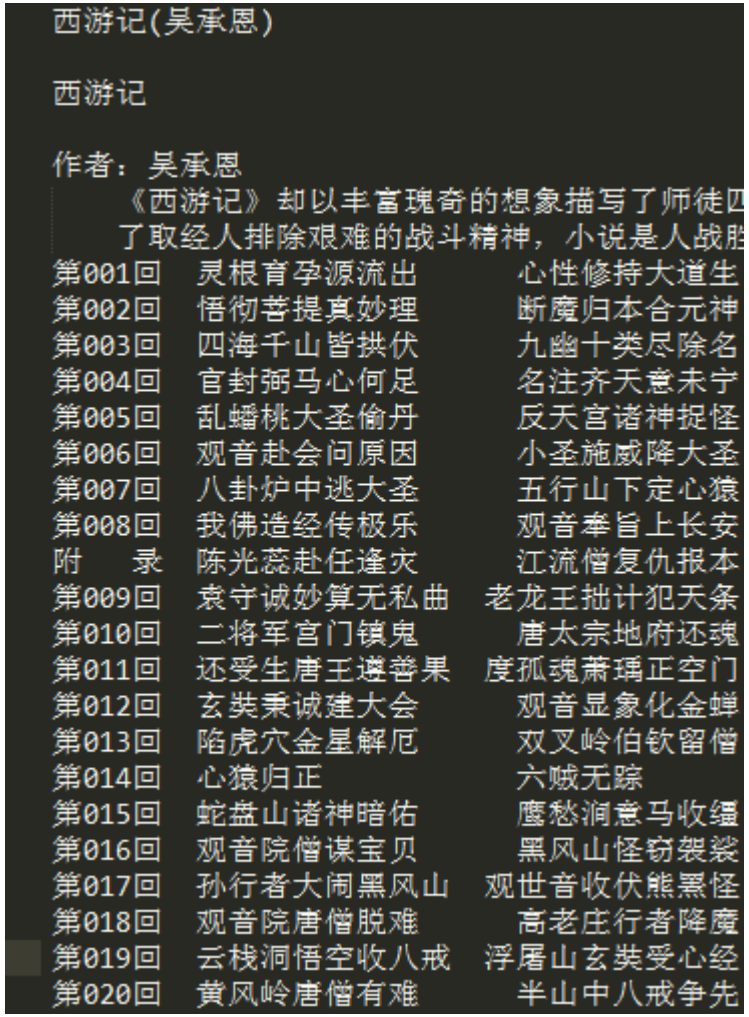
Word2Vec工具

数据集：西游记

- journey_to_the_west.txt
- 计算小说中的人物相似度，比如孙悟空与猪八戒，孙悟空与孙行者

方案步骤：

- Step1，使用分词工具进行分词，比如NLTK,JIEBA
- Step2，将训练语料转化成一个sentence的迭代器
- Step3，使用word2vec进行训练
- Step4，计算两个单词的相似度



Word2Vec工具

```
# 字词分割，对整个文件内容进行字词分割
def segment_lines(file_list,segment_out_dir,stopwords=[]):
    for i,file in enumerate(file_list):
        segment_out_name=os.path.join(segment_out_dir,'segment_{}.txt'.format(i))
        with open(file, 'rb') as f:
            document = f.read()
            document_cut = jieba.cut(document)
            sentence_segment=[]
            for word in document_cut:
                if word not in stopwords:
                    sentence_segment.append(word)
            result = ' '.join(sentence_segment)
            result = result.encode('utf-8')
            with open(segment_out_name, 'wb') as f2:
                f2.write(result)


# 对source中的txt文件进行分词，输出到segment目录中
file_list=files_processing.get_files_list('./source', postfix='*.txt')
segment_lines(file_list, './segment')
```

```
西游记 ( 吴承恩 )

西游记

作者 :  吴承恩
        《 西游记 》 却 以 丰富 瑰奇 的 想象 描写 了 师徒 四众
        的 精怪 生动 地 表现 了 无情 的 山川 的 险阻 ， 并 以
第 001 回    灵根育孕 源流 出          心性 修持 大道 生
第 002 回    悟彻 菩提 真 妙理          断 魔 归本 合元神
第 003 回    四海 千山 皆 拱 伏          九幽 十类 尽 除名
第 004 回    官封弼 马心 何足          名注 齐 天意 未宁
第 005 回    乱 蟠桃 大圣 偷丹          反 天宫 诸神 捉 怪
第 006 回    观音 赴 会 问 原因          小圣 施威 降 大圣
第 007 回    八卦 炉中 逃 大圣          五行 山下 定心 猿
第 008 回    我 佛造 经传 极乐          观音 奉旨 上 长安
附      录    陈光蕊 赴任 逢灾          江流 僧 复仇 报本
第 009 回    袁守诚 妙算 无私 曲    老 龙王 拙计 犯 天条
第 010 回    二 将军 宫门 镇鬼          唐太宗 地府 还魂
第 011 回    还 受生 唐王 遵善果    度 孤魂 萧 瑀 正 空门
第 012 回    玄奘 秉诚建 大会          观音 显象 化 金蝉
第 013 回    陷 虎穴 金星 解厄          双叉岭 伯钦 留僧
第 014 回    心猿 归正                  六贼 无踪
第 015 回    蛇 盘山 诸神 暗佑          鹰愁 涧 意马 收缰
第 016 回    观音院 僧谋 宝贝          黑风山 怪窃 袈裟
第 017 回    孙行者 大闹 黑风山    观世音 收伏 熊 黑 怪
第 018 回    观音院 唐僧脱 难          高老庄 行者 降魔
第 019 回    云栈 洞 悟空 收 八戒    浮屠 山 玄奘 受心经
第 020 回    黄风岭 唐僧 有难          半山 中 八戒 争先
第 021 回    护法 设庄留 大圣          须弥 灵吉定 风魔
第 022 回    八戒 大战 流沙河          木叉 奉法 收悟净
```

Word2Vec工具

```
# 将Word转换成Vec，然后计算相似度
from gensim.models import word2vec
import multiprocessing
# 如果目录中有多个文件，可以使用PathLineSentences
sentences = word2vec.PathLineSentences('./segment')
# 设置模型参数，进行训练
model = word2vec.Word2Vec(sentences, size=100, window=3, min_count=1)
print(model.wv.similarity('孙悟空', '猪八戒'))
print(model.wv.similarity('孙悟空', '孙行者'))
# 设置模型参数，进行训练
model2 = word2vec.Word2Vec(sentences, size=128, window=5, min_count=5,
workers=multiprocessing.cpu_count())
# 保存模型
model2.save('./models/word2Vec.model')
print(model2.wv.similarity('孙悟空', '猪八戒'))
print(model2.wv.similarity('孙悟空', '孙行者'))
print(model2.wv.most_similar(positive=['孙悟空', '唐僧'], negative=['孙行者']))
```

```
0.96224165
0.98238677
0.9193349
0.9293375
[('菩萨', 0.9546594619750977), ('何干', 0.9491345286369324),
('长老', 0.9437956809997559), ('沙僧见', 0.943665623664856),
('悟净', 0.9424765706062317), ('手指', 0.9390666484832764),
('银角', 0.937991201877594), ('太子', 0.9359638690948486), ('众',
0.9339208006858826), ('禅师', 0.9319193363189697)]
```

Summary

Word2Vec工具的使用：

- Word Embedding就是将Word嵌入到一个数学空间里，Word2vec，就是词嵌入的一种
- 可以将sentence中的word转换为固定大小的向量表达（Vector Respresentations），
- 其中意义相近的词将被映射到向量空间中相近的位置。
- 将待解决的问题转换成为单词word和文章doc的对应关系

大V推荐中，大V => 单词，将每一个用户关注大V的顺序 => 文章

商品推荐中，商品 => 单词，用户对商品的行为顺序 => 文章

打卡：三国演义embedding

使用Gensim中的Word2Vec对三国演义进行Word Embedding

- 分析和曹操最相近的词有哪些
- 曹操+刘备-张飞=?

数据集：three_kingdoms.txt

Embedding模型的选择

Embedding的作用

Thinking：Embedding模型的核心特性是什么？

Embedding模型将文本等离散数据转换为低维、稠密的向量，捕捉其语义信息。

向量空间中的距离（如余弦相似度）可反映文本间的语义相似度。

Thinking：什么是MTEB榜单？

MTEB (Massive Text Embedding Benchmark) 是一个全面的评测基准，它涵盖了分类、聚类、检索、排序等8大类任务和58个数据集。

通过MTEB榜单，可以清晰地看到不同模型（如 BGE系列, GTE, Jina 等）在不同任务类型上的性能表现。

比如例如，某些模型在检索任务上表现优异，而另一些则可能在聚类或分类任务上更具优势。这有助于我们根据具体应用场景，做出初步的模型筛选。

MTEB榜单

Rank (Bo…	Model	Zero-shot	Memory U…	Number of P…	Embedding D…	Max Tokens	Mean (T…	Mean (TaskT…	Bite
1	gemini-embedding-001	99%	Unknown	Unknown	3072	2048	68.37	59.59	79.2
2	Qwen3-Embedding-8B	99%	28866	7B	4096	32768	**70.58**	**61.69**	**80.8**
3	Qwen3-Embedding-4B	99%	15341	4B	2560	32768	69.45	60.86	79.3
4	Qwen3-Embedding-0.6B	99%	2272	595M	1024	32768	64.34	56.01	72.2
5	Linq-Embed-Mistral	99%	13563	7B	4096	32768	61.47	54.14	70.3
6	gte-Qwen2-7B-instruct	⚠ NA	29040	7B	3584	32768	62.51	55.93	73.9
7	multilingual-e5-large-instruct	99%	1068	560M	1024	514	63.22	55.08	80.1
8	SFR-Embedding-Mistral	96%	13563	7B	4096	32768	60.90	53.92	70.0
9	text-multilingual-embedding-002	99%	Unknown	Unknown	768	2048	62.16	54.25	70.7
10	GritLM-7B	99%	13813	7B	4096	4096	60.92	53.74	70.5

https://huggingface.co/spaces/mteb/leaderboard

MTEB榜单（任务类型）

MTEB (Massive Text Embedding Benchmark)：

一个全面的评测基准，它涵盖了分类、聚类、检索、排序等8大类任务和58个数据集。

- 检索 (Retrieval)： 从一个庞大的文档库中，根据用户输入的查询（Query），找出最相关的文档列表。
- 语义文本相似度 (Semantic Textual Similarity, STS)： 判断一对句子的语义相似程度，并给出一个连续的分数（例如1到5分）。
- 重排序 (Reranking)： 对一个已经初步检索出的文档列表进行二次优化排序，使得最相关的文档排在最前面。
- 分类 (Classification)： 将单个文本（如电影评论、新闻文章）划分到预定义的类别中（如“正面/负面”、“体育/科技”）。

通过MTEB榜单，可以看到不同模型（如 BGE系列, GTE, Jina 等）在不同任务类型上的性能表现。

MTEB榜单（任务类型）

- 聚类 (Clustering)： 在没有任何预设标签的情况下，将一组文本自动地分成若干个有意义的群组，使得同一组内的文本语义相似，不同组间的文本语义差异大。
- 对分类 (Pair Classification)： 判断一对文本（句子或段落）是否具有某种特定关系，通常是二分类问题，如“是否是重复问题”、“是否是转述关系”。
- 双语挖掘 (Bitext Mining)： 从两种不同语言的大量句子中，找出互为翻译的句子对。对于机器翻译至关重要。
- 摘要 (Summarization)： 这个任务比较特殊，它不是让模型生成摘要，而是评估一个机器生成的摘要与人工撰写的参考摘要之间的语义相似度。

通过MTEB榜单，可以看到不同模型（如 BGE系列, GTE, Jina 等）在不同任务类型上的性能表现。

向量维度对模型性能的影响

Thinking：向量维度对模型性能的影响是怎样的？

向量维度直接影响模型的表达能力、计算开销和内存占用。

高维度 (如 1024, 4096): 编码更丰富、语义更细致，适用于需要深度语义理解的复杂场景，如大规模、多样化的信息检索，或者细粒度的文本分类。但计算成本更高，所需存储空间更大。

低维度 (如 256, 512): 计算速度快，内存占用小，更适合计算资源有限，或实时性要求高的场景，比如移动端。

Thinking：如果把向量从768维拉长到1024维，检索指标提高不到1%，但内存要多占约35%，是否还要升维？

不需要，性价比低。

反过来，若压缩到768维后，指标下降超过5% => 说明信息损失大，值得使用更高维度。

Jina Embedding

Jina Embedding：

- 由 Jina AI（官网 jina.ai）开发，公司总部位于德国柏林，专注于开源多模态搜索与向量化技术
- jina-embeddings-v4 是一个多模态和多语言检索的通用嵌入模型，特别适合用于复杂的文档检索，包括包含图表、表格和插图的视觉丰富文档。

https://modelscope.cn/models/jinaai/jina-embeddings-v4

Jina Embedding具有灵活的嵌入大小，默认情况下，密集嵌入为2048维，但可以截断到最低128维，性能损失较小。

特性	Jina-Embedding-V4
基础模型	Qwen2.5-VL-3B-Instruct
支持的任务	检索, 文本匹配, 代码
模型数据类型	BFloat 16
最大序列长度	32768
单向量维度	2048
多向量维度	128
套娃维度	128, 256, 512, 1024, 2048
池化策略	平均池化
注意力机制	FlashAttention2

神奇的“俄罗斯套娃”

Jina-embeddings训练时使用了一种特殊技术（Matryoshka Representation Learning, MRL）=> 俄罗斯套娃

生成完整向量： 模型总是先在内部生成一个最完整、维度最高（比如 2048维）的向量。

按需截断： 这个长向量有一个非常神奇的特性，它的前128维、前256维、前512维……本身就是一组高质量的、独立的、可以正常使用的短向量。

用户指定： 当调用模型时，可以通过 embedding_size告诉模型 “我这次只需要前512维就够了” => 模型就会截断向量，只返回前512维

神奇的“俄罗斯套娃”

场景1：社交媒体情感分析

社交媒体的文本短，实时性要求高，计算资源有限 => 可以使用128维

在索引所有社交媒体评论时，都调用模型请求128维的向量。

当用户进行查询时，您也同样将查询文本转换为128维的向量，然后进行比较。

场景2：投资分析报告

投资分析、公司财报包含大量专业术语和细节（如风险提示、前瞻性声明），精准理解至关重要=> 2048维。

在索引所有投资分析报告时，都请求2048维的向量。查询语句也同样生成2048维的向量进行匹配。

动态调整维度是Jina-embedding模型赋予开发者的一个强大选项

单语言与多语言Embedding模型的选择

Thinking：单语言和多语言Embedding模型的特点是怎样的？

单语言模型: 如 BGE-large-zh，专门针对单一语言（如中文）进行训练，

比如：为电商平台开发一个智能客服问答系统。

目标： 系统需要能精准理解用户使用中文提出的问题：

- 我的订单何时能送达？
- 这个商品有保修吗？
- 如何办理退换货？

并从FAQ知识库中匹配最相关的答案。

单语言模型在特定语言任务上，理解更深入、性能更优越。

比如理解“七天无理由退货”

单语言与多语言Embedding模型的选择

多语言模型: 如 m3e-base 或 multilingual-e5-large

能够处理多种语言的文本，并将它们映射到统一的语义空间中。

场景：为一个国际连锁酒店集团，建立全球客户评论分析系统。

将来自世界各地的评论，按主题（如：客房清洁度、员工服务、地理位置）进行自动分类，无论评论是用英文、日文、西班牙文还是中文写的。

总部的经理可以用英文查询“Loud music at night”，系统需要能同时找出写着“夜に音楽がうるさい”的日文评论和“晚上音乐很吵”的中文评论。

多语言Embedding的优势是能将不同语言的文本映射到统一的语义空间。

“clean room”、“部屋が綺麗”和“干净的房间”的向量在空间中会非常接近。

=> 跨语言的聚类分析和检索才能实现。

如何选择适合的Embedding模型

Thinking：如何选择适合的Embedding模型，依靠MTEB就可以？

模型选型是一个系统的过程，不能仅依赖于公开榜单。包括以下关键步骤：

- 明确业务场景与评估指标: 首先定义核心任务是检索、分类还是聚类？并确定衡量业务成功的关键指标，如搜索召回率 (Recall@K)、准确率 (Accuracy) 或 NDCG。
- 构建“黄金”测试集: 准备一套能真实反映您业务场景和数据分布的高质量小规模测试集。

比如，构建一系列“问题-标准答案”对 => 评估模型好坏的“金标准”。

- 小范围对比测试 (Benchmark): 从MTEB榜单中挑选几款排名靠前且符合需求（如语言、维度）的候选模型。使用“黄金”测试集，对这些模型进行评测。

Embedding模型的选择属于综合评估，即结合测试结果、模型的推理速度、部署成本 => 做出最终决策

向量数据库

向量数据库

Thinking：如何让LLM“记住”并利用海量、多样的私有知识？

向量数据库：AI时代的核心记忆体

与传统的关系型数据库不同，向量数据库用于存储和查询由非结构化数据（如文本、图片、音视频）转化而来的高维向量嵌入（Embeddings）。这些向量在多维空间中的距离代表了原始数据的语义相似度。

因此，向量数据库的核心能力是高效的相似性检索。

Thinking：向量数据库的核心价值？

- 为大模型提供长期记忆： 弥补LLM上下文窗口（Context Window）长度限制和知识更新延迟的问题。
- 实现私有知识库的问答与搜索： 将企业内部文档、产品信息等转化为向量，实现基于语义的智能检索。
- 赋能推荐系统、以图搜图等多种应用： 通过计算用户、物品的向量相似度，提供更精准的推荐。

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. Elasticsearch

特点： 强大的分布式搜索和分析引擎，将向量搜索（k-NN）作为其众多功能之一。

优势： 具备业界领先的混合搜索能力，可以无缝结合传统的关键词搜索和向量语义搜索。

3. 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。

4. 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。

数据导入

Thinking：如何将数据导入向量数据库？

Step1，数据清洗与准备

确保原始数据（如文本文档、图片）的质量，进行必要的预处理。

Step2，数据向量化（Embedding）

使用预训练的Embedding Model将原始数据转换成向量。

文本： 可使用 bge-m, Qwen3-Embedding, Jina-Embedding 等模型。

图片： 可使用 CLIP, ResNet 等模型。

选择合适的模型至关重要， 它直接决定了向量的质量和后续检索的效果。

数据导入

```
import os
from openai import OpenAI
client = OpenAI(
    api_key=os.getenv("DASHSCOPE_API_KEY"),
    base_url="https://dashscope.aliyuncs.com/compatible-mode/v1"  # 百炼服务的base_url
)
completion = client.embeddings.create(
    model="text-embedding-v4",
    input='我想知道迪士尼的退票政策',
    dimensions=1024, # 指定向量维度（仅 text-embedding-v3及 text-embedding-v4支持该参数）
    encoding_format="float"
)
print(completion.model_dump_json())
```

{"data":[{"embedding":[0.009544237516820 43,-0.11166470497846603,0.00026108720339834 69,-0.044482450932264 33,0.018730206415057182,-0.013019428588449955,0.01536249555647373 2,-0.0032817272003740072,-0.001339023467153 3108,0.12496358901262283,0.047090634703636 17,-0.036141991615295 41,0.027127988636493683,-0.031470187008380 89,0.07085091620683 67,-0.07320114970207214,-0.038635529577732086,-0.08036649227142334, …],"index":0,"object":"embedding"}],"model":"text-embedding-v4","object":"list","usage":{"prompt_tokens":23,"total_tokens":23},"id":"84faa227-2672-94df-87d3-a1648dc7aea7"}

1-embedding计算.py

数据导入

Step3，数据与元数据一同导入

将生成的向量及与其关联的 元数据（Metadata）一同存入向量数据库。

- 向量（Vector）： 生成的Embedding数字数组。
- 唯一ID（ID）： 用于唯一标识每个数据点，方便后续的更新或删除。
- 元数据（Metadata）： 描述向量的附加信息，是实现高级检索的关键。例如：

文本来源的文件名、章节、URL

商品的类别、品牌、价格

图片的创建日期、作者

数据导入

Thinking：如何将 Embedding 和元数据一起存储在 FAISS？

FAISS 本身只存储和检索向量，不存储元数据。=> 我们需要在 FAISS 之外维护一个元数据的“查找表”，并通过向量在 FAISS 中的唯一ID将两者关联起来。

最直接有效的方法是使用 FAISS 的 IndexIDMap => 允许我们为每个向量指定一个自定义的、唯一的64位整数ID。然后，可以用这个ID作为元数据存储的键。

Embedding与原数据导入Faiss

```
import os
import numpy as np
import faiss
from openai import OpenAI


# Step1. 初始化 API 客户端
try:
    client = OpenAI(
        api_key=os.getenv("DASHSCOPE_API_KEY"),
        base_url="https://dashscope.aliyuncs.com/compatible-mode/v1"
    )
except Exception as e:
    ...
```

```
# Step2. 准备示例文本和元数据
documents = [
    {
        "id": "doc1",
        "text": "迪士尼乐园的门票一经售出，原则上不予退换。但在特殊情况下，如恶劣天气导致园区关闭，可在官方指引下进行改期或退款。",
        "metadata": {"source": "official_faq_v1.pdf", "category": "退票政策", "author": "Admin"}
    },
    {
        "id": "doc2",
        "text": "购买“奇妙年卡”的用户，可以享受一年内多次入... }]
```

2-embedding-faiss-元数据.py

Embedding与原数据导入Faiss

```
# Step3. 创建元数据存储和向量列表
metadata_store = []
vectors_list = []
vector_ids = []
print("正在为文档生成向量...")
for i, doc in enumerate(documents):
    try:
        # 调用API生成向量
        completion = client.embeddings.create(
            model="text-embedding-v4",
            input=doc["text"],
            dimensions=1024,
            encoding_format="float"
        )
        # 获取向量
        vector = completion.data[0].embedding
        vectors_list.append(vector)
        # 存储元数据，并使用列表索引作为唯一ID
        metadata_store.append(doc)
        vector_ids.append(i)
        print(f"  - 已处理文档 {i+1}/{len(documents)}")
    except Exception as e:
        print(f"处理文档 '{doc['id']}' 时出错: {e}")
        continue

# 将向量列表转换为NumPy数组，FAISS需要这种格式
vectors_np = np.array(vectors_list).astype('float32')
vector_ids_np = np.array(vector_ids)
```

2-embedding-faiss-元数据.py

Embedding与原数据导入Faiss

```
# Step4. 构建并填充 FAISS 索引
dimension = 1024  # 向量维度
k = 3          # 查找最近的3个邻居
# 创建一个基础的L2距离索引
index_flat_l2 = faiss.IndexFlatL2(dimension)

# 使用IndexIDMap来包装基础索引，能够映射我们自定义的ID
# 这就是关联向量和元数据的关键！
index = faiss.IndexIDMap(index_flat_l2)

# 将向量和对应的ID添加到索引中
index.add_with_ids(vectors_np, vector_ids_np)
print(f"\nFAISS 索引已成功创建，共包含 {index.ntotal} 个向量。
")
```

```
# Step5. 执行搜索
query_text = "我想了解一下迪士尼门票的退款流程"
print(f"\n正在为查询文本生成向量: '{query_text}'")
try:
    # 为查询文本生成向量
    query_completion = client.embeddings.create(
        model="text-embedding-v4",
        input=query_text,
        dimensions=1024,
        encoding_format="float"
    )
    query_vector =
np.array([query_completion.data[0].embedding]).astype('float32’)
    distances, retrieved_ids = index.search(query_vector, k)
```

2-embedding-faiss-元数据.py

Embedding与原数据导入Faiss

```
# Step6. 展示结果
print("\n--- 搜索结果 ---")
# `retrieved_ids[0]` 包含与查询最相似的k个向量的ID
for i in range(k):
    doc_id = retrieved_ids[0][i]

    # 检查ID是否有效
    if doc_id == -1:
        print(f"\n排名 {i+1}: 未找到更多结果。")
        continue
    # 使用ID从我们的元数据存储中检索信息
    retrieved_doc = metadata_store[doc_id]
```

```
print(f"\n--- 排名 {i+1} (相似度得分/距离: {distances[0][i]:.4f}) ---")
        print(f"ID: {doc_id}")
        print(f"原始文本: {retrieved_doc['text']}")
        print(f"元数据: {retrieved_doc['metadata']}")


except Exception as e:
    print(f"执行搜索时发生错误: {e}")
```

2-embedding-faiss-元数据.py

Embedding与原数据导入Faiss

正在为文档生成向量...
- 已处理文档 1/4
- 已处理文档 2/4
- 已处理文档 3/4
- 已处理文档 4/4

FAISS 索引已成功创建，共包含 4 个向量。
正在为查询文本生成向量: '我想了解一下迪士尼门票的退款流程'

--- 搜索结果 ---
--- 排名 1 (相似度得分/距离: 0.3222) ---
ID: 2
原始文本: 对于在线购买的迪士尼门票，如果需要退票，必须在票面日期前48小时通过原购买渠道提交申请，并可能收取手续费。
元数据: {'source': 'online_policy.html', 'category': '退票政策', 'author': 'E-commerceTeam'}

--- 排名 2 (相似度得分/距离: 0.3312) ---
ID: 0
原始文本: 迪士尼乐园的门票一经售出，原则上不予退换。但在特殊情况下，如恶劣天气导致园区关闭，可在官方指引下进行改期或退款。
元数据: {'source': 'official_faq_v1.pdf', 'category': '退票政策', 'author': 'Admin'}

--- 排名 3 (相似度得分/距离: 1.0135) ---
ID: 1
原始文本: 购买“奇妙年卡”的用户，可以享受一年内多次入园的特权，并且在餐饮和购物时有折扣。
元数据: {'source': 'annual_pass_rules.docx', 'category': '会员权益', 'author': 'MarketingDept'}

2-embedding-faiss-元数据.py

Summary（实现步骤）

Step1, 准备数据：创建示例文本和对应的元数据。

Step2, 生成向量：基于百炼 text-embedding-v4 生成每个文本的向量。

Step3, 创建元数据存储：使用一个简单的 Python 列表来存储元数据。列表的索引将作为每个数据点的唯一ID。

Step4, 构建 FAISS 索引：

- 使用 faiss.IndexFlatL2 创建一个基础的索引 => 这里使用L2距离（欧氏距离）进行精确搜索。
- 用 faiss.IndexIDMap 将基础索引包装起来 => 这样就可以添加带有自定义ID的向量了。

Step5, 添加数据到索引：将生成的向量和对应的ID（即元数据列表的索引）添加到 IndexIDMap 中。

Summary（实现步骤）

Step6, 执行搜索：

- 对一个新的查询文本生成向量。
- 在 FAISS 索引中搜索最相似的向量。
- FAISS 会返回最相似向量的ID。

Step7, 检索元数据：

使用返回的ID，从元数据存储中查找到原始文本和元数据。

Summary

Thinking：如何将metadata元数据管理的更健壮？

在更复杂的生产环境中，可以将 metadata_store 这个简单的列表替换为更健壮的存储方案：

- 键值数据库（如 Redis）：通过ID快速查询，性能极高。
- 关系型数据库（如 PostgreSQL）：可以存储更复杂的结构化元数据。
- 文档数据库（如 MongoDB）：非常适合存储JSON格式的元数据。

FAISS 专注于其最擅长的高速向量检索，而元数据的存储和管理可以交给专业的数据库系统负责，实现了架构上的解耦和高效协同。

向量数据库选择

数据库	核心特点	性能表现	适用场景
FAISS	核心算法库，非数据库。 由Meta AI开发。提供最广泛、最前沿的ANN索引算法。支持CPU和GPU。	纯粹的性能标杆。 在内存中进行裸向量检索时速度极快，尤其是GPU版本，是衡量其他数据库性能的基准。	算法研究者；需要将向量检索能力深度集成到现有系统中的团队；可以自行构建服务层（API、元数据管理）
Milvus	开源领导者，功能全面。 云原生架构，高度可扩展。支持多种ANN索引和丰富的调优参数。	在多个公开的ANN基准测试中，尤其是在大规模数据集上表现优异，吞吐量和延迟控制得很好。	需要处理海量数据、对性能和扩展性有高要求的企业级应用。适合有一定运维能力的团队进行私有化部署。
Pinecone	全托管的商业先驱。 主打“Serverless”和易用性，API设计简洁。性能稳定，专注于提供极致的低延迟检索。	性能非常出色，尤其在低延迟方面有优势。由于是闭源托管服务，其内部优化细节不透明，但用户体验顺滑。	追求快速上线、希望将运维负担降至最低的团队。初创公司和需要快速验证AI应用可行性的场景。
Weaviate	开源，内置数据向量化模块。 能够连接各种Embedding模型，实现数据的自动向量化，简化了开发流程。	性能良好，易用性强。其自动向量化功能在开发效率上是一大亮点。	希望简化ETL流程，快速构建从数据到向量再到检索的全链路应用的开发者。
Qdrant	开源，以性能和内存安全著称 (使用Rust开发)。 过滤查询能力强大且高效。	性能强劲，尤其在过滤条件复杂的混合查询场景下表现突出。内存使用效率高。	对检索性能和资源利用率有极致要求的场景。金融、电商等需要复杂过滤规则的应用。
Elasticsearch	通用搜索巨头，扩展向量检索能力。 将向量检索（KNN）作为功能之一。可以无缝结合其强大的全文检索、聚合分析能力。	作为一个通用数据库，其向量检索性能通常弱于专门的向量数据库。但在混合搜索（关键词+向量）场景下，表现优异	业务需求是“以文本搜索为主，向量搜索为辅”，希望在一个统一的平台中解决所有搜索问题，而非追求极致的向量检索性能。

打卡：向量数据库与元数据管理

你是否有文档需要检索，不放将它Embedding，并放到向量数据库中

- 使用 text-embedding-v4 对原始数据进行向量化
- 使用 Faiss存储和管理你的Embeddings
- 将元数据进行管理，放到 metadata_store中（也可以使用Redis等数据库）
- 进行Query，查看匹配的准确度，并将元数据进行展示


Thank You
Using data to solve problems