Embedding与向量数据库

>> 今天的学习目标

什么是Embedding

- CASE：基于内容的推荐

什么是N-Gram

余弦相似度计算

为酒店建立内容推荐系统

- Word Embedding

什么是Embedding

Word2Vec进行词向量训练

向量数据库

- 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

什么是Embedding

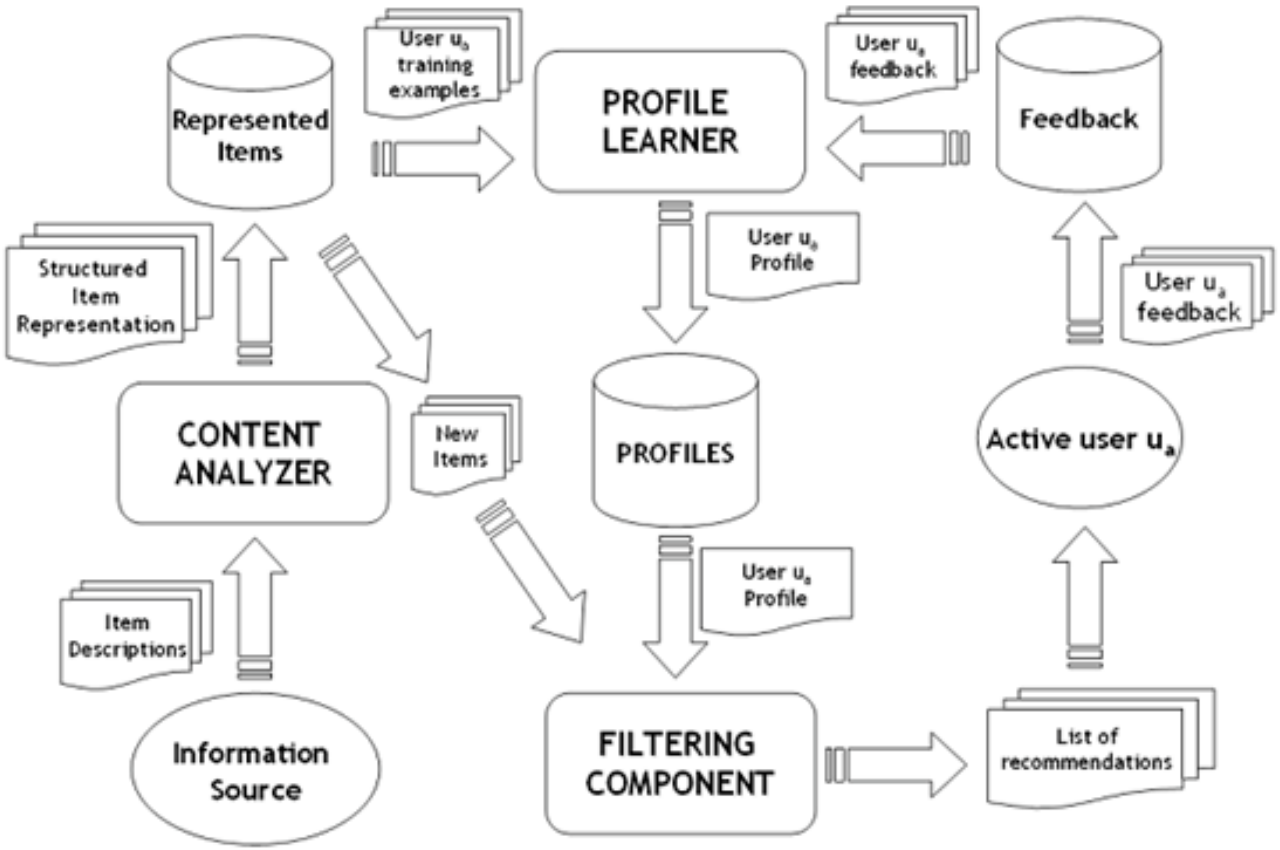
基于内容的推荐

基于内容的推荐：

- 依赖性低，不需要动态的用户行为，只要有内容就可以进行推荐
- 系统不同阶段都可以应用
  - ✓系统冷启动，内容是任何系统天生的属性，可以从中挖掘到特征，实现推荐系统的冷启动。一个复杂的推荐系统是从基于内容的推荐成长起来的
  - ✓商品冷启动，不论什么阶段，总会有新的物品加入，这时只要有内容信息，就可以帮它进行推荐

了解Embedding可以从了解物体的特征表达开始

基于内容的推荐



- 物品表示 Item Representation：

为每个item抽取出features

- 特征学习Profile Learning：

利用一个用户过去喜欢（不喜欢）的item的特征数据，来学习该用户的喜好特征（profile）；

- 生成推荐列表Recommendation Generation：

通过用户profile与候选item的特征，推荐相关性最大的item。

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

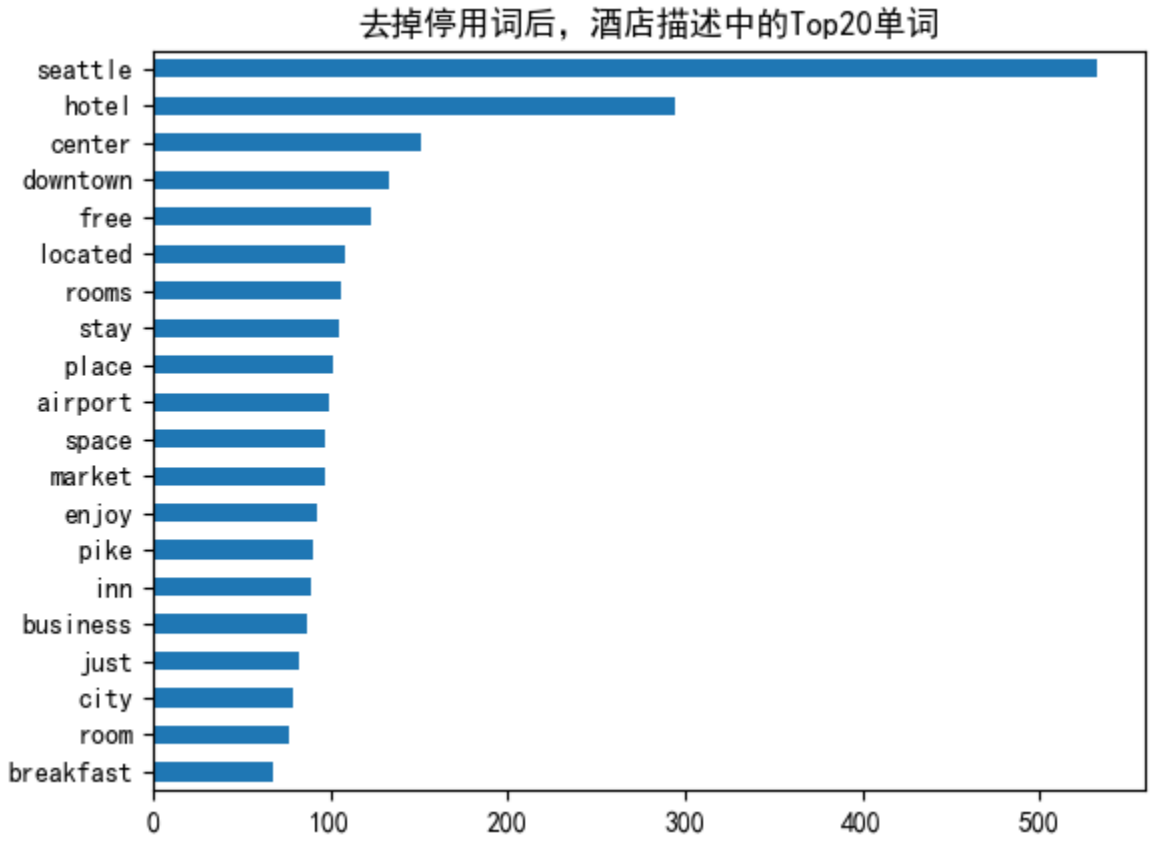
比如文本：A B C D E，对应的Bi-Gram为A B, B C, C D, D E

- 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

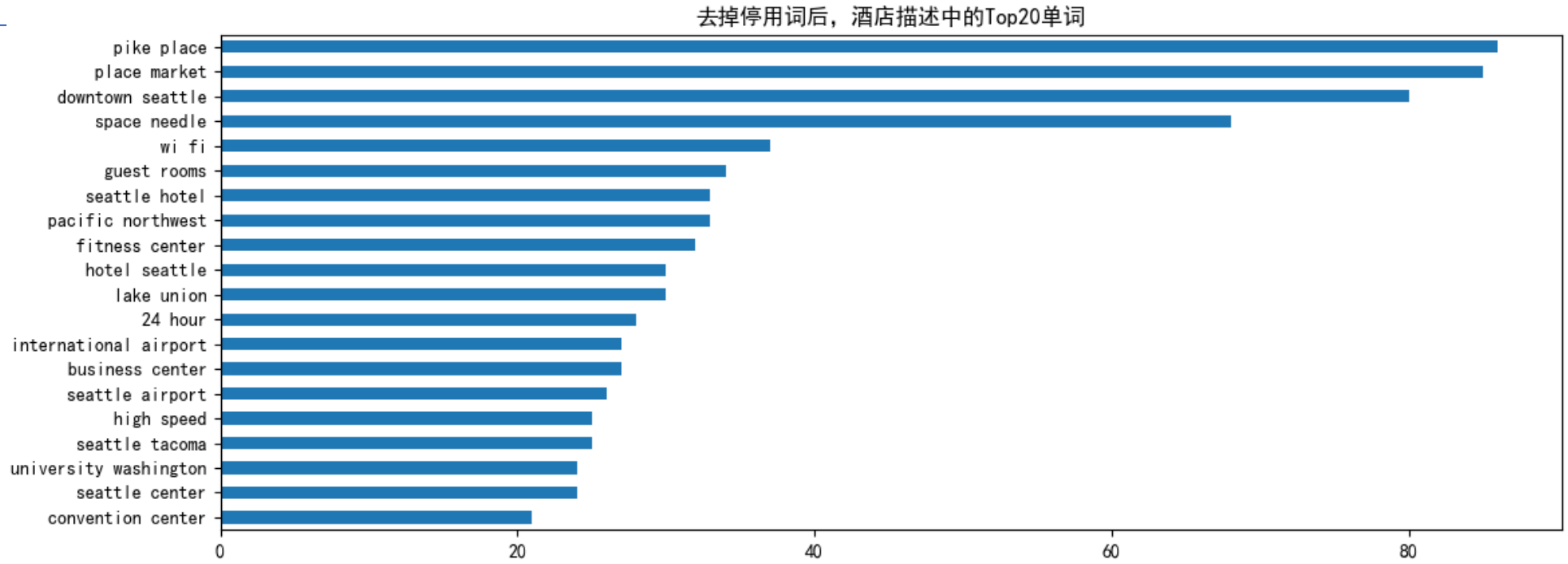
```
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签

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

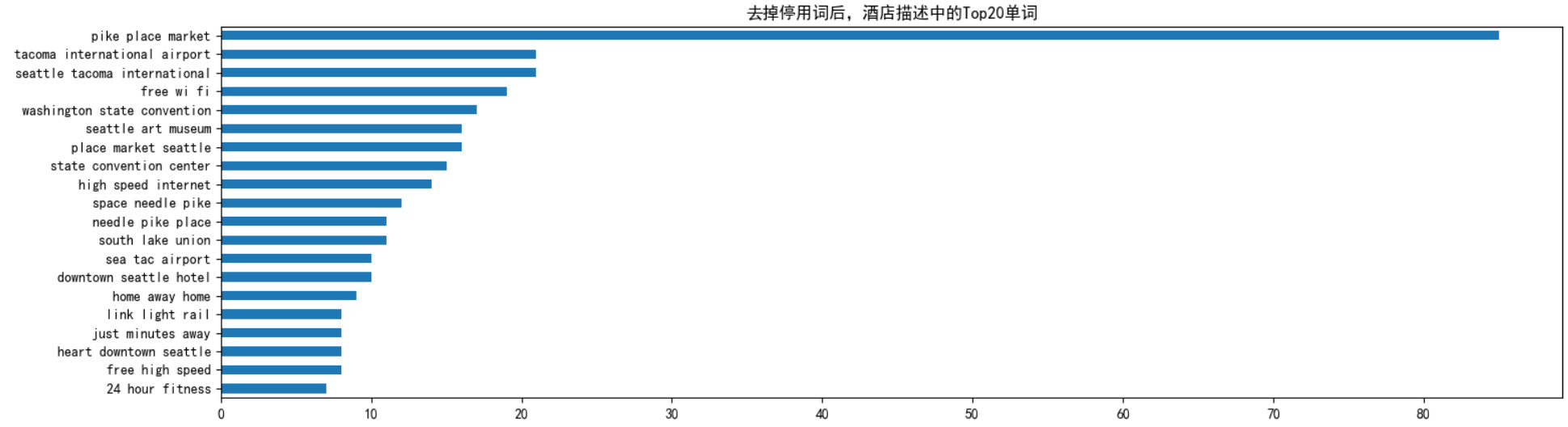
Bi-Gram

```
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

TfidfVectorizer:

- 将文档集合转化为tf-idf特征值的矩阵

构造函数

- analyzer：word或者char，即定义特征为词（word）或n-gram字符
- ngram_range: 参数为二元组(min_n, max_n)，即要提取的n-gram的下限和上限范围
- max_df：最大词频，数值为小数[0.0, 1.0],或者是整数，默认为1.0
- min_df：最小词频，数值为小数[0.0, 1.0],或者是整数，默认为1.0
- stop_words：停用词，数据类型为列表

功能函数:

- fit_transform：进行tf-idf训练，学习到一个字典，并返回Document-term的矩阵，也就是词典中的词在该文档中出现的频次

```
# 使用TF-IDF提取文本特征
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 3), min_df=0,
stop_words='english')
tfidf_matrix = tf.fit_transform(df['desc_clean'])
print(tfidf_matrix)
print(tfidf_matrix.shape)
```

```
  (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)
```

为酒店建立内容推荐系统

基于内容的推荐：

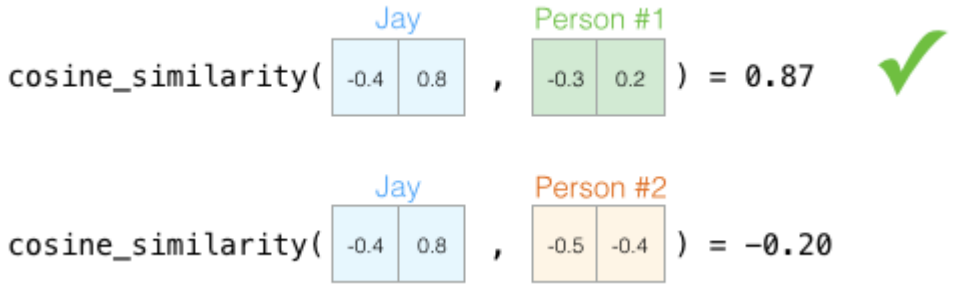
- Step1，对酒店描述（Desc）进行特征提取
  - N-Gram，提取N个连续字的集合，作为特征
  - TF-IDF，按照(min_df, max_df)提取关键词，并生成TFIDF矩阵
- Step2，计算酒店之间的相似度矩阵
  - 余弦相似度
- Step3，对于指定的酒店，选择相似度最大的Top-K个酒店进行输出

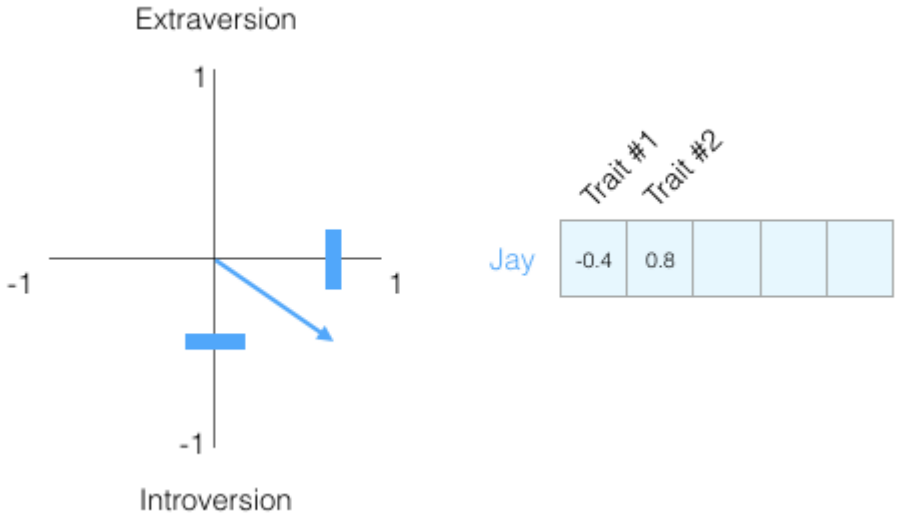
Thinking :N-Gram + TF-IDF的特征表达会让特征矩阵非常系数，计算量大，有没有更适合的方式?

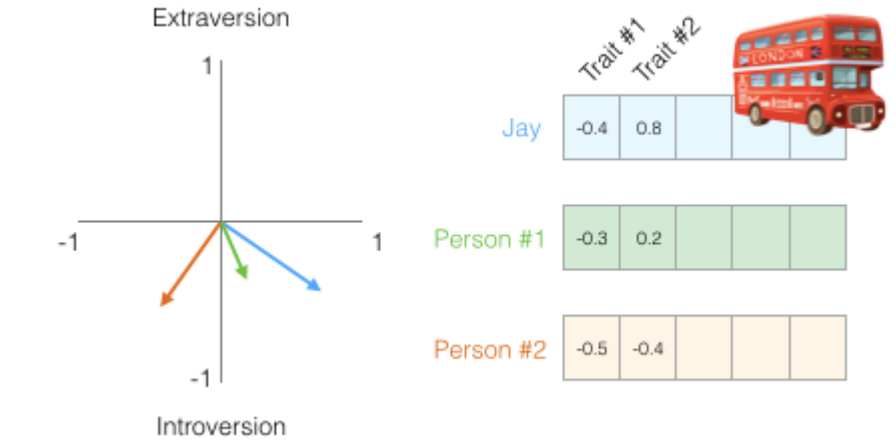
Word Embedding

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







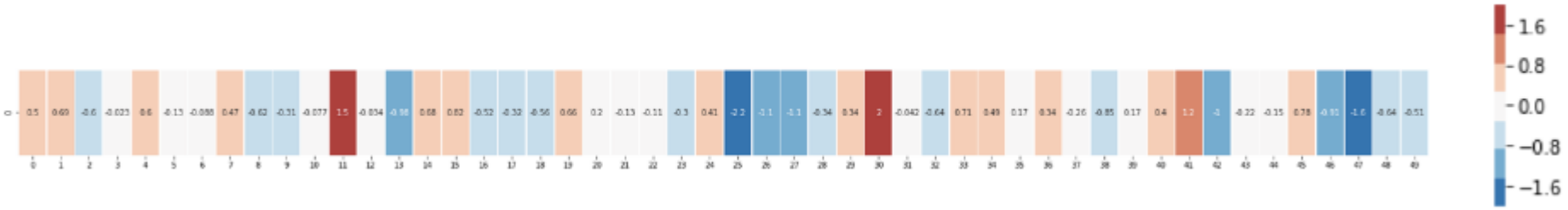
Word Embedding

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

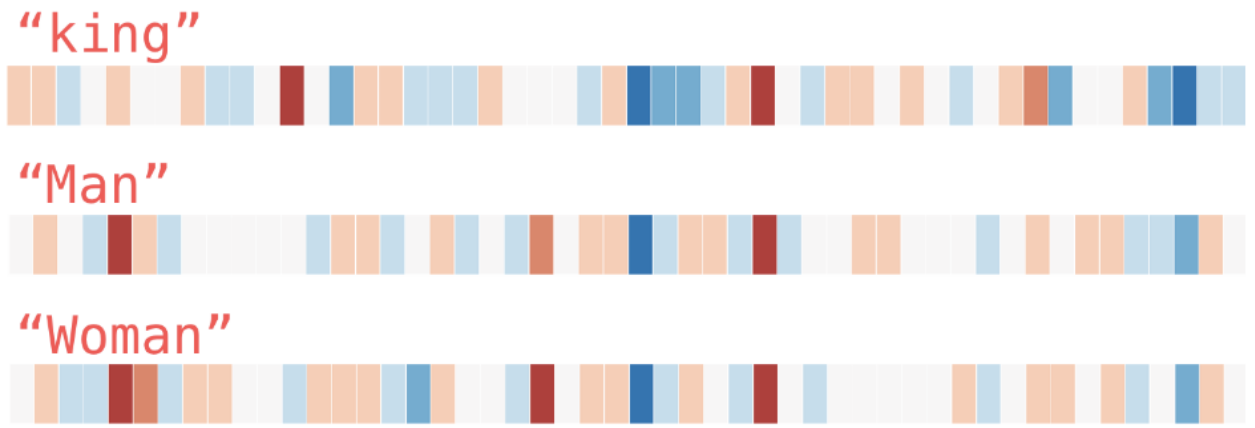


Word Embedding

将Word进行Embedding：

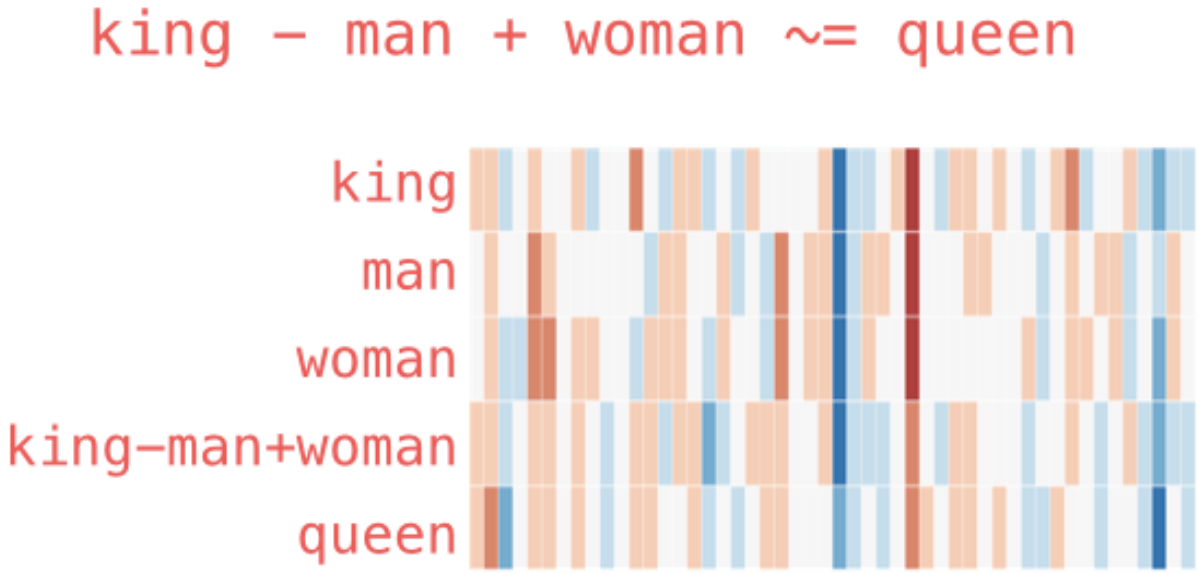
- 我们将King与其他单词进行比较，可以看到Man和Woman更相近
- 同样有了向量，我们还可以进行运算

king-man+woman与queen的相似度最高




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

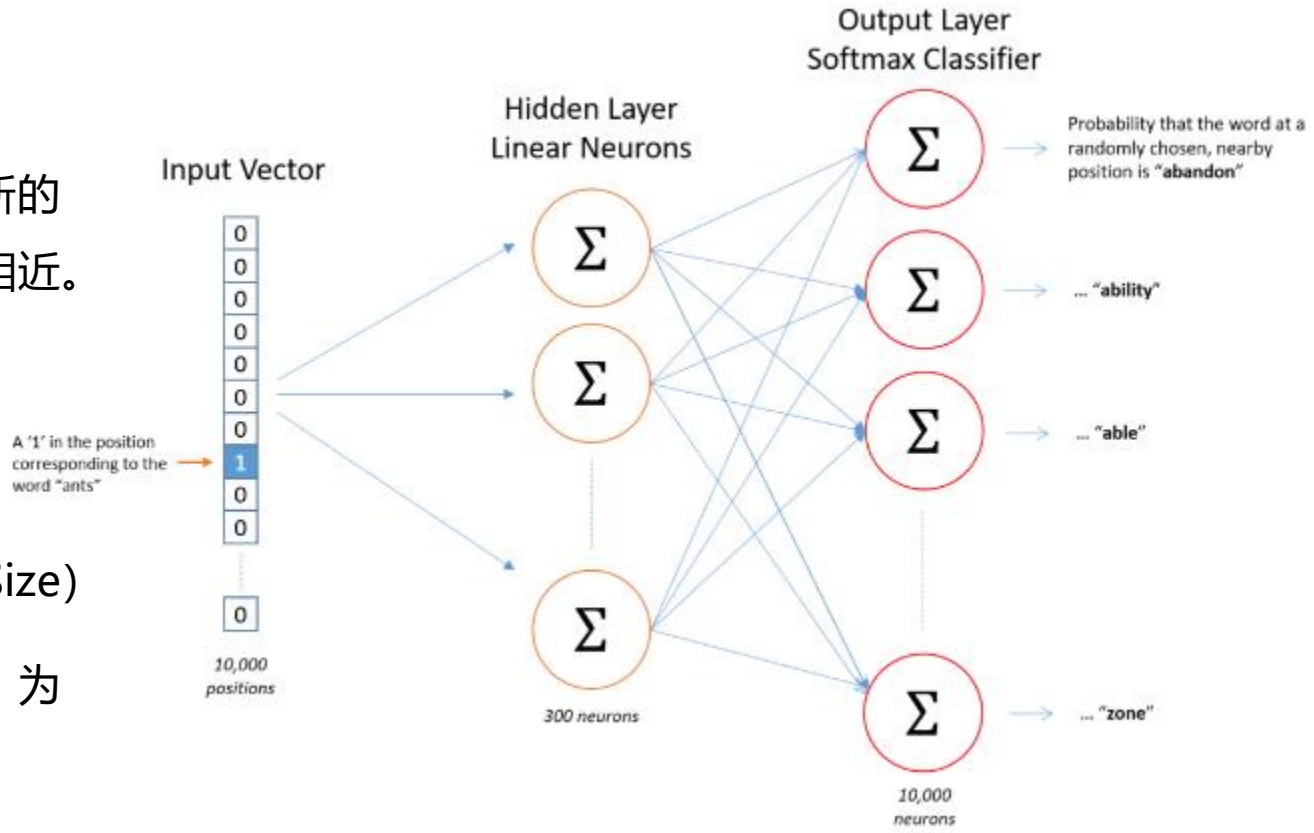
```
[('queen', 0.8523603677749634),
 ('throne', 0.7664333581924438),
 ('prince', 0.7592144012451172),
 ('daughter', 0.7473883032798767),
 ('elizabeth', 0.7460219860076904),
 ('princess', 0.7424570322036743),
 ('kingdom', 0.7337411642074585),
 ('monarch', 0.721449077129364),
 ('eldest', 0.7184862494468689),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

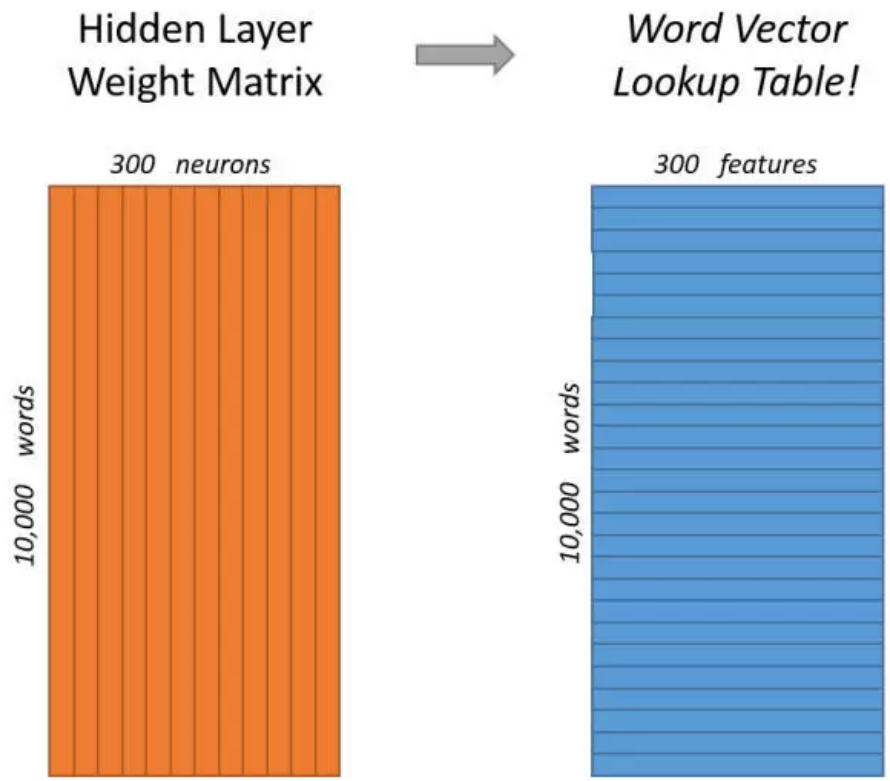


Word Embedding

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

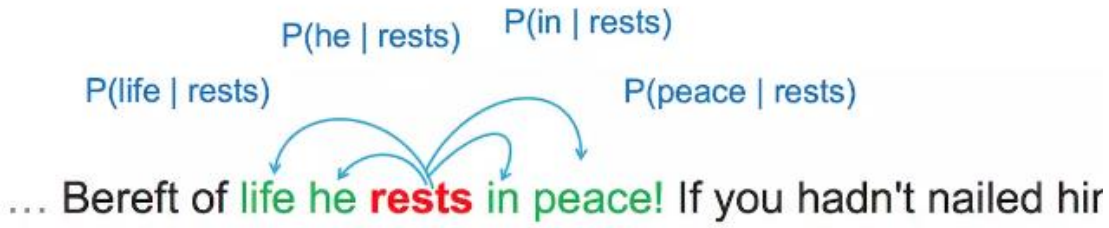
$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$



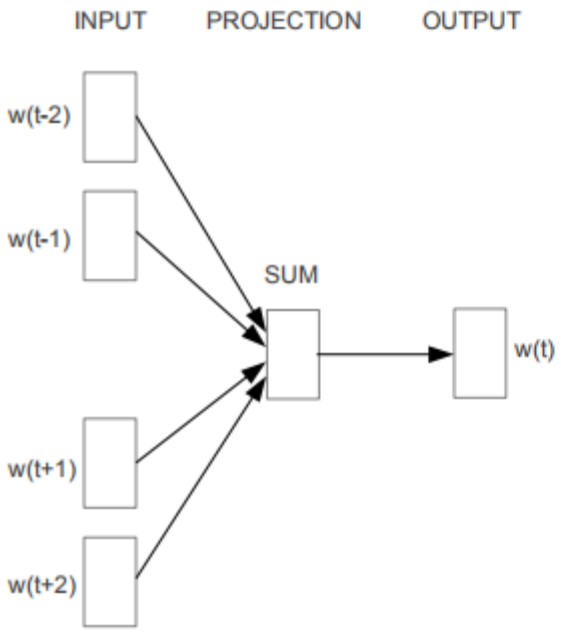
Word Embedding

Word2Vec的两种模式：

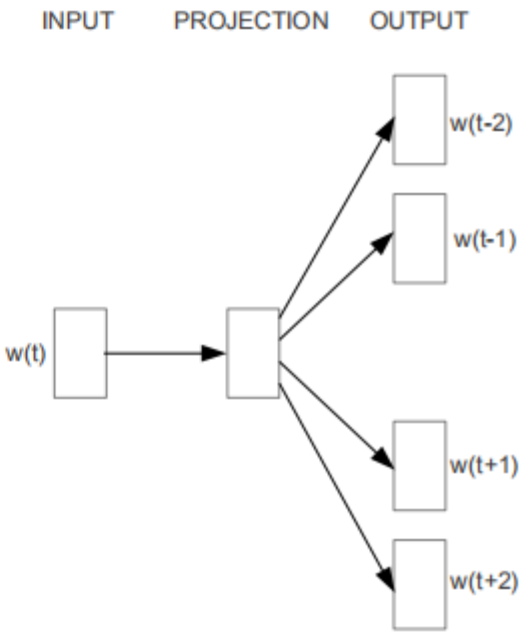
- Skip-Gram，给定input word预测上下文



- 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)

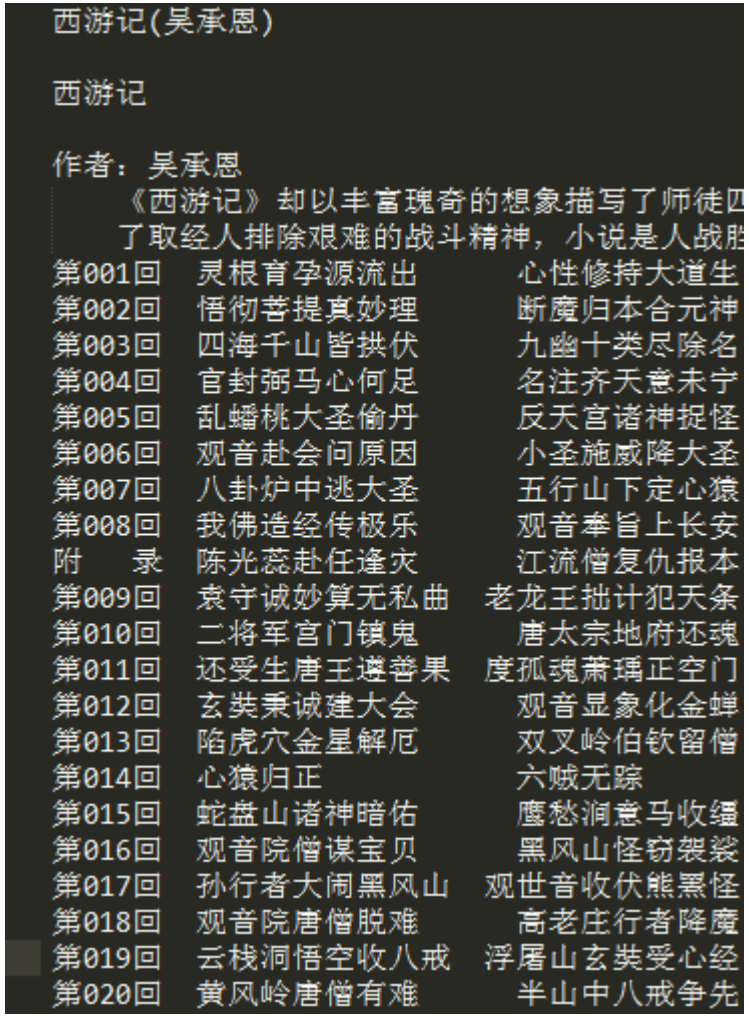
Word2Vec工具

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

Thinking：得到了特征表达（item representation），有什么用?

基于内容的推荐：

- 将你看的item，相似的item推荐给你
- 通过物品表示Item Representation => 抽取特征

TF-IDF => 返回给某个文本的“关键词-TFIDF值”的词数对

TF-IDF可以帮我们抽取文本的重要特征，做成item embedding

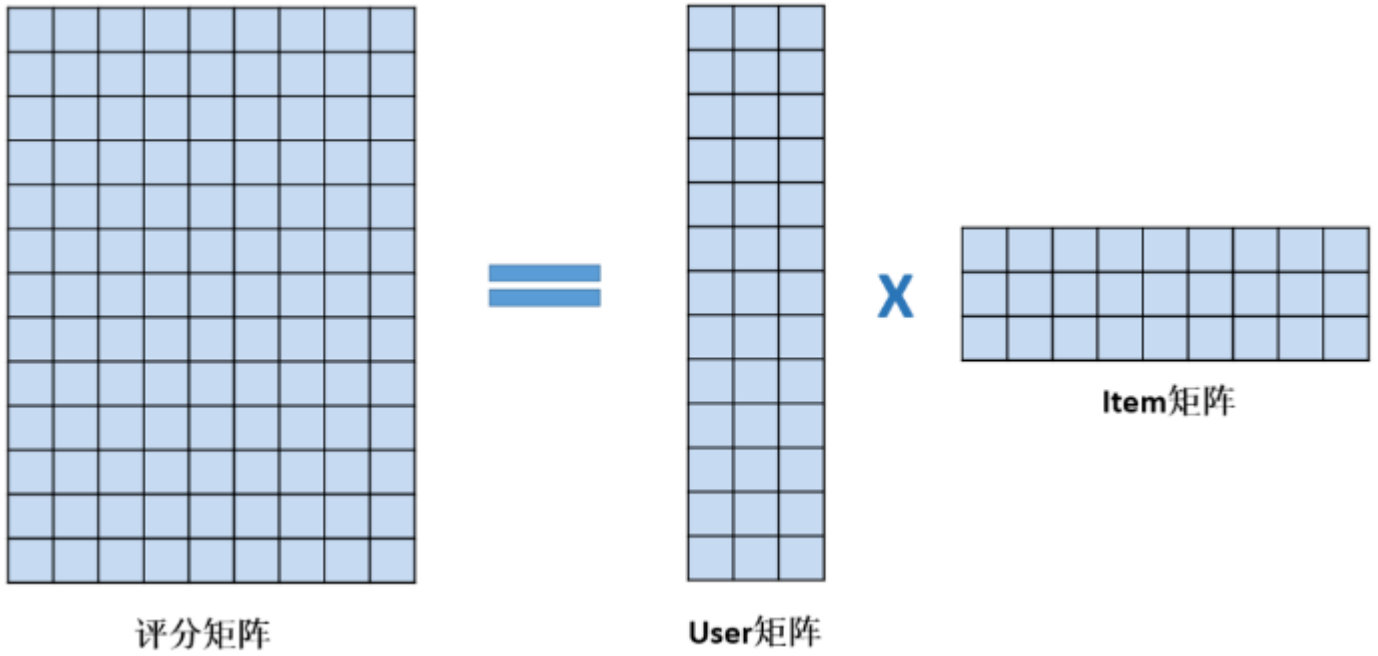
Thinking：如何使用事物的特征表达，进行相似推荐?

- 计算item之间的相似度矩阵
- 对于指定的item，选择相似度最大的Top-K个进行输出

Summary

Embedding的理解：

- Embedding指某个对象 X 被嵌入到另外一个对象Y中，映射 f : X → Y
- 一种降维方式，转换为维度相同的向量
- 矩阵分解中的User矩阵，第i行可以看成是第i个user的Embedding。Item矩阵中的第 j 列 可 以 看 成 是 对 第 j 个 Item 的 Embedding



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

向量数据库

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。

讨论与呈现

结合你的业务场景，你认为事物的特征提取重要么？有什么应用场景？

非结构化数据的理解，pdf, word等

结构化数据的理解，excel，数据表等

如何对这些特征进行理解？

Thinking&Action

Thinking1：假设一个小说网站，有N部小说，每部小说都有摘要描述。如何针对该网站制定基于内容的推荐系统，即用户看了某部小说后，推荐其他相关的小说。原理和步骤是怎样的

Thinking2：Word2Vec的应用场景有哪些

Thinking&Action

Action1：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


Thank You
Using data to solve problems