你也可以构建的高级但却很简单的购物推荐系统--item2vec

潘振福 AI菜鸟成长日志 2019-06-09

简介

本文可以算是一遍item2vector的工程化教程，其基本理论源自于论文《Item2Vec: Neural Item Embedding for Collaborative Filtering 》(https://arxiv.org/abs/1603.04259)将引导你了解word2vec背后的思想，以及它在电商推荐系统领域中的一个扩展（item2vec）。具体来说，利用word2vec的概念和gensim软件包提供的模块，将构建一个轻量级的电影推荐系统。

item2vector简介： 计算items的相似性是现代推荐系统中的一个关键组成部分。过去，人们只是通过判断"有无"的方式来向量化系统中的各个items，这样的方式并不能很好地表示出items之间的关系，而且泛化能力很差。因此需要一种特殊的方法来将items映射到能够满足我们需求的向量上去。

后来研究人员研究出了使用SVD（奇异值分解，矩阵分解的一种）的方式来计算表示items的向量，这种方法从某种程度上已经满足了人们在计算items的相似性以及模型泛化能力的需求了。但是随着自然语言领域不断发展，研究人员开发出了一种叫做word2vec的神经语言模型，通过神经网络模型来隐式地计算出词汇表中的每一个单词的向量化表示，这种表示方式能够很好地描述单词间的语义和语法关系，从而使得模型具有相当的泛化能力，因此该方法及其变形已经被广泛地应用到NLP领域的各个方面。

所以，论文指出，可以使用相似的方法来计算协助过滤中所有items的向量化表示，从而隐式地得到items间的相互关系，使得我们能够更好地计算出items间的相似性以及提升模型的泛化能力。

它是一种基于item的协同过滤算法(Collaborative Filtering algorithm)，利用在潜在空间(latent space)中item embedding 表征。与传统的奇异值分解（SVD）方法甚至是深度学习算法相比，该算法具有很强的竞争力。

简单介绍一下word2vec 和item2vec的原理

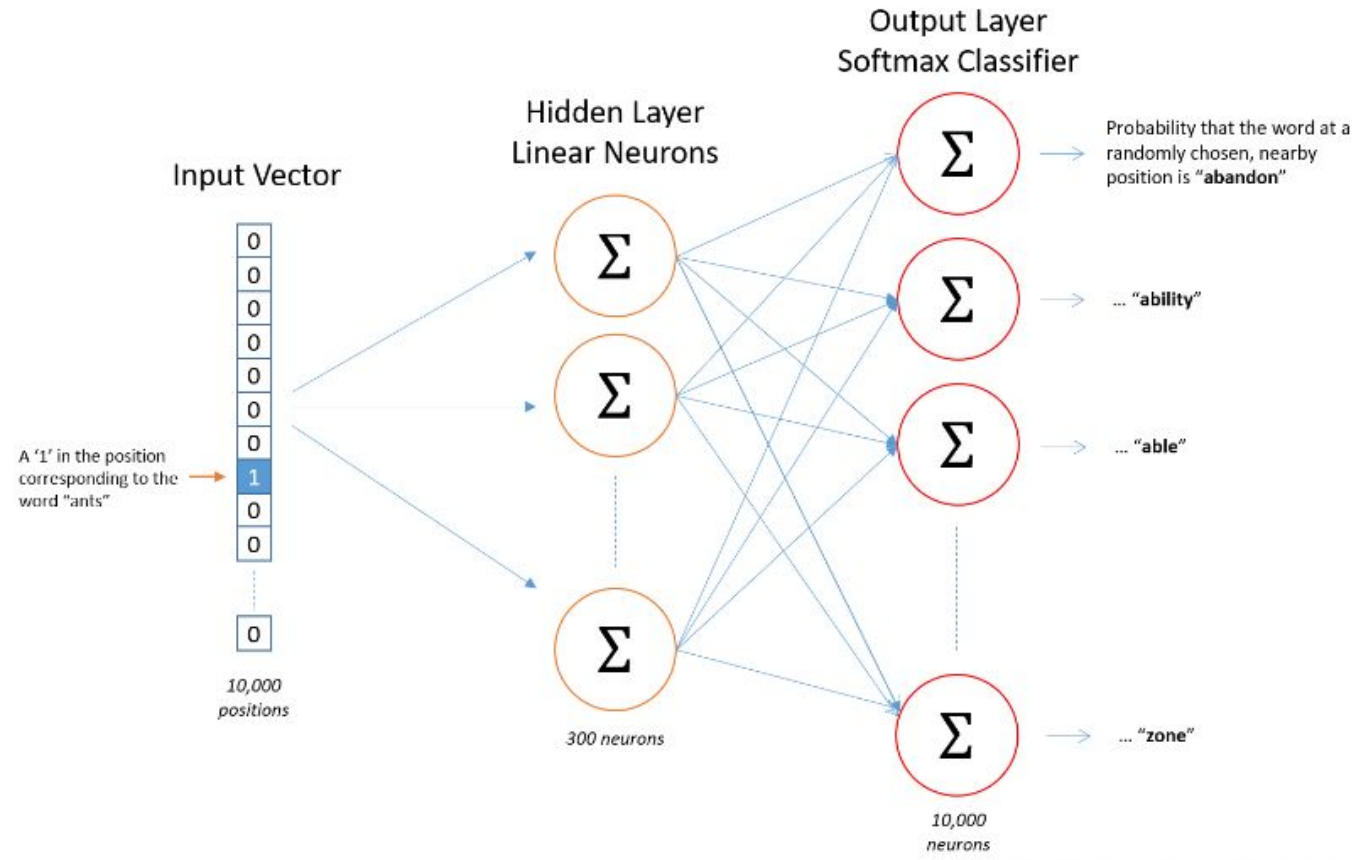
文章将引用 McCormick, C.的一篇关于 Word2Vec的教程，想了解更多关于word2vec的内容和原理，极力推荐阅读原文。

word2vec其实包括两个浅层神经网络结构。即CBOW和Skip-Gram结构。这些体系结构描述了神经网络如何"学习"神经网络中每一层节点的权重。这里将简单地解释一下Skip-Gram的结构，这足以让我们理解item2vec了。

Skip-Gram就是：给一个单词，预测在字典里头常常出现在该单词"邻近"的单词的概率。这里的"邻近"就是定义在一个固定大小的窗口(window)内，在句子中，距离给定单

词小于固定窗口(window)的大小的单词。例如：“在 这里 生活着，我 很 快乐！”，如果取大小为2的窗口，“我”的邻近单词[“很”，“快乐”，“生活着”，“这里”]，[“在”]就不在邻近的集合内部。

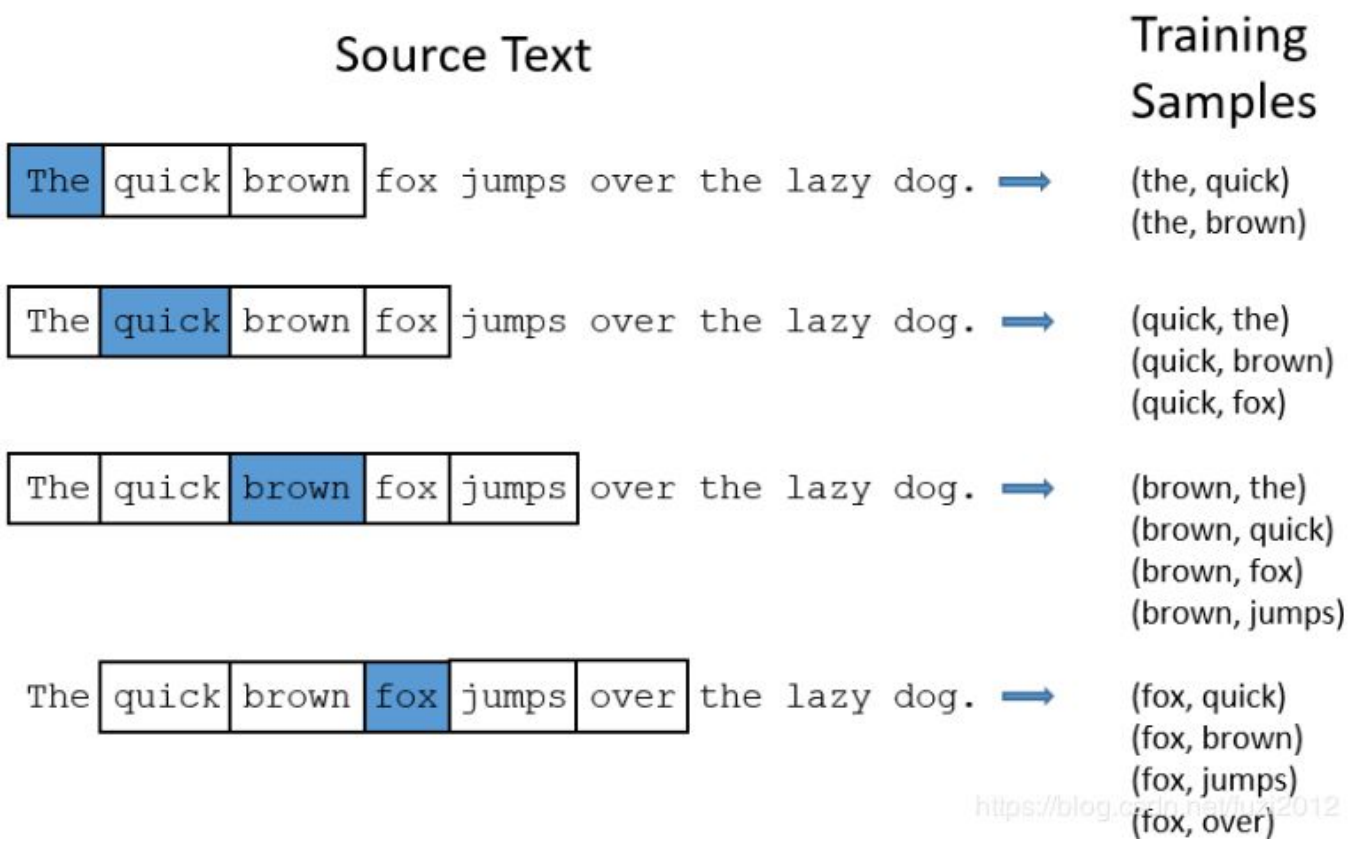
Skip Gram体系结构使用以下布局来解决问题：



(The vocabulary size depends on the data, and the size of the middle layer is a hyperparameter)

在这种结构中，对于一个给定的单词，它的一个独热编码(one-hot)的单词向量被投影到一个嵌入到中间层的低维单词(embedding)中，然后转换成一个指定其周围单词概率的向量。想了解更多，请阅读此文

这里的机器学习优化问题是从数据中学习最优的投影矩阵。Wrod2vec是一种“自监督”算法，从这个意义上说，虽然我们不需要提供标记的数据，但是需要从数据中生成“正确的概率”，以便算法学习权重。下面的图表最能说明生成training sample的过程。根据生成的词对，可以计算出“正确”概率。每个训练示例都将被传递到这个结构中，以调整投影矩阵的权重。



令人兴奋的是，在这个体系结构的中间层学习到的嵌入表征(embedding representation)保留了语义属性和单词之间的关系。而单词的输入编码并不要求有任何的关于单词特征和单词之间关系的属性。例如，从输入层中的一个独热编码(one-hot)创建的词表征(representation)没有这些属性。

word2vec完整的skip-Gram的架构还包括负采样，以降低计算复杂性，提高词向量的质量。以上基本是描述了skip-Gram的简单形式，但这就是理解item2vec所需要的全部内容。**item2vec**架构也是利用了skip-Gram和负采样，所以只要假设每个item是一个“word”，每个项目集合是一个“sentence”，我们的目标是学习item embedding，以找出item之间的关系。想了解更多item2vec，请查看原论文。

准备数据

这里将使用由movieens研究团队策划的movieens 20M数据集。它包含由138000名用户完成关于27000部电影的2000万个评价数据和46.5万个标签数据。有关详细信息，您可以访问官方网站。您可以通过此链接下载数据集。

为了构建推荐系统，将使用下载数据集中的“movies.csv”和“ratings.csv”文件。“movies.csv”是一个查找表，用于查找电影的ID及其名称。“ratings.csv”包含所有电影用户的分级。以下代码读取csv文件，检查数据并可视化分级分布。为了方便起见，我还为电影ID和电影名称创建了查找字典。

```
import pandas as pdimport numpy as np

df_movies = pd.read_csv('ml-20m/movies.csv')
```

```
df_ratings = pd.read_csv('ml-20m/ratings.csv')

movieId_to_name = pd.Series(df_movies.title.values, index = df_movies.movieId.
name_to_movieId = pd.Series(df_movies.movieId.values, index = df_movies.title)
    rand_idx = np.random.choice(len(df), 5, replace=False)
    display(df.iloc[rand_idx,:])
    print("Displaying 5 of the total "+str(len(df))+" data points")
```

	movieId	title	genres
5015	5111	Good Son, The (1993)	Drama|Thriller
10003	32906	Ascent, The (Voskhozhdeniye) (1977)	Drama|War
16509	83381	Seven Thieves (1960)	Crime|Drama
846	861	Supercop (Police Story 3: Supercop) (Jing cha ...	Action|Comedy|Crime|Thriller
20155	99258	Who Wants to Kill Jessie? (Kdo chce zabít Jess...	Comedy|Sci-Fi

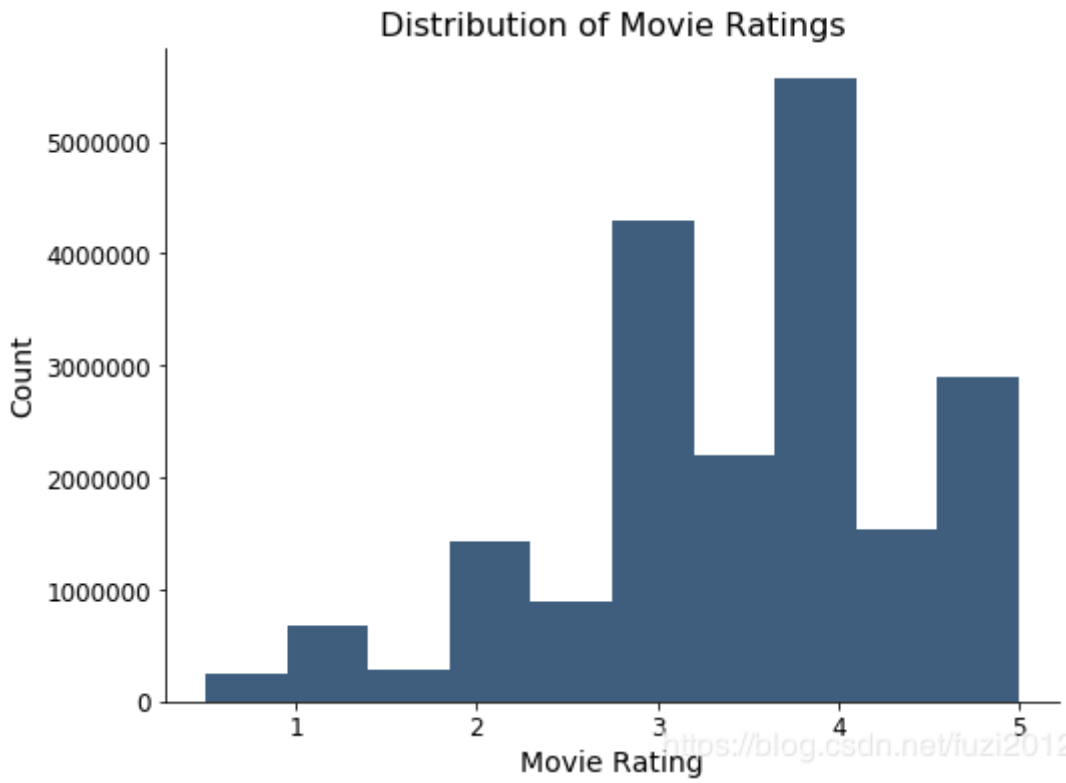
```
Displaying 5 of the total 27278 data points
```

	userId	movieId	rating	timestamp
7937017	54688	47610	3.5	1370061037
11079211	76564	1101	3.0	946438936
15516997	107317	1270	0.5	1162693457
17771935	122871	5313	3.5	1230660174
16371933	113342	780	3.0	1356738841

```
Displaying 5 of the total 20000263 data points
```



```
import matplotlib.pyplot as pltimport plotly.plotly as py
%matplotlib inline

plt.figure(figsize=(8, 6))
ax = plt.subplot(111)
ax.set_title("Distribution of Movie Ratings", fontsize=16)
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.xlabel("Movie Rating", fontsize=14)
plt.ylabel("Count", fontsize=14)

plt.hist(df_ratings['rating'], color="#3F5D7D")

plt.show()
```



在标准的机器学习开发流程中，首先需要将数据分为训练集和测试集。测试集用于评估模型。评估推荐系统有多种方法，这会影响如何分割数据。将会使用精确度、召回率和topK f-1评分来评估模型性能（在评估性能部分中进行了解释），因此对userId进行分层分割。对于每个用户，以70:30的比率将电影数据分为"训练集和测试集数据"分割开来。通过Scikit-Learn，可以在一行代码中完成这项工作。

```
from sklearn.model_selection import train_test_split

df_ratings_train, df_ratings_test= train_test_split(df_ratings,
                                                    stratify=df_ratings['userI
                                                    random_state = 15688,
                                                    test_size=0.30)
```

```
print("Number of training data: "+str(len(df_ratings_train)))
print("Number of test data: "+str(len(df_ratings_test)))
```

```
Number of training data: 14000184
Number of test data: 6000079
```

为了让模型学习item embedding，需要从数据中获取"单词"和"句子"等价物。在这里，把每个"电影"看做是一个"词"，并且从用户那里获得相似评级的电影都在同一个"句子"中。

具体来说，"句子"是通过以下过程生成的：为每个用户生成2个列表，分别存储用户"喜欢"和"不喜欢"的电影。第一个列表包含所有的电影评级为4分或以上。第二个列表包含其余的电影。这些列表就是训练gensim word2vec模型的输入了。

```
def rating_splitter(df):

    df['liked'] = np.where(df['rating']>=4, 1, 0)
    df['movieId'] = df['movieId'].astype('str')
    gp_user_like = df.groupby(['liked', 'userId'])    return ([gp_user_like.ge
```

```
pd.options.mode.chained_assignment = Nonesplitted_movies = rating_splitter(df_
```

利用Gensim 训练item2vec 的模型

在本节中，我们将把训练数据输入gensim word2vec模块中，调整窗口大小，训练item2vec模型。
对于原来的word2vec，窗口大小会影响我们搜索“上下文”以定义给定单词含义的范围。按照定义，窗口的大小是固定的。但是，在item2vec实现中，电影的“含义”应该由同一列表中的所有邻居捕获。换句话说，我们应该考虑用户“喜欢”的所有电影，以定义这些电影的“含义”。这也适用于用户“不喜欢”的所有电影。然后需要根据每个电影列表的大小更改窗口大小。
为了在不修改gensim模型的底层代码的情况下解决这个问题，首先指定一个非常大的窗口大小，这个窗口大小远远大于训练样本中任何电影列表的长度。然后，在将训练数据输入模型之前对其进行无序处理，因为在使用“邻近”定义电影的“含义”时，电影的顺序没有任何意义。
Gensim模型中的窗口参数实际上是随机动态的。我们指定最大窗口大小，而不是实际使用的窗口大小。尽管上面的解决方法并不理想，但它确实实现了可接受的性能。最好的方法可能是直接修改gensim中的底层代码，但这就超出了我目前的能力范围了，哈哈哈。
在训练模型之前，需要确保gensim是使用C编译器的，运行下面的代码来验证这一点。

```
import warnings
warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim'
```

然后打乱数据集：

```
import randomfor movie_list in splitted_movies:
    random.shuffle(movie_list)
```

下面训练两个模型，当然，训练模型需要些时间，不同的机器耗时不一样：

```
from gensim.models import Word2Vecimport datetime
start = datetime.datetime.now()

model = Word2Vec(sentences = splitted_movies, # We will supply the pre-process
                 iter = 5, # epoch
                 min_count = 10, # a movie has to appear more than 10 times to
                 size = 200, # size of the hidden layer
                 workers = 4, # specify the number of threads to be used for t
                 sg = 1, # Defines the training algorithm. We will use skip-gr
                 hs = 0, # Set to 0, as we are applying negative sampling.
                 negative = 5, # If > 0, negative sampling will be used. We wi
                 window = 9999999)

print("Time passed: " + str(datetime.datetime.now()-start))#Word2Vec.save('iter
```

Time passed: 2:12:26.134283

```
from gensim.models import Word2Vecimport datetime
start = datetime.datetime.now()

model_w2v_sg = Word2Vec(sentences = splitted_movies,
                        iter = 10, # epoch
                        min_count = 5, # a movie has to appear more than 5 tim
                        size = 300, # size of the hidden layer
                        workers = 4, # specify the number of threads to be use
                        sg = 1,
                        hs = 0,
                        negative = 5,
                        window = 9999999)

print("Time passed: " + str(datetime.datetime.now()-start))
model_w2v_sg.save('item2vec_word2vecSg_20180328')del model_w2v_sg
```

Time passed: 5:32:50.270232

模型训练完之后，模型可以保存在您的存储中以备将来使用。注意，gensim保存了所有关于模型的信息，包括隐藏的权重、词汇频率和模型的二叉树，因此可以在加载文件后继续训练。然而，这是以运行模型时的内存为代价的，因为它将存储在你的RAM中。如果你只需要隐藏层的权重，它可以从模型中单独提取。下面的代码演示如何保存、加载模型和提取单词向量(embedding)。

```
import warnings
warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim'
model = Word2Vec.load('item2vec_20180327')
word_vectors = model.wv# del model # uncomment this line will delete the model
```

推荐系统来了！！！

一旦模型训练完，就可以使用Gensim中的内置方法来做推荐！真正使用的是`gensim model.wv.most_similar_word()`方法。这些实用程序将接受用户的输入，并将它们输入gensim方法中，从中根据对IMDB的搜索推断最可能的电影名称，将它们转换为电影ID。

```
import requestsimport refrom bs4 import BeautifulSoupdef refine_search(search_t
    """
    Refine the movie name to be recognized by the recommender
    Args:
        search_term (string): Search Term

    Returns:
        refined_term (string): a name that can be search in the dataset
    """
    target_url = "http://www.imdb.com/find?ref_=nv_sr_fn&q="+"+".join(search_t
    html = requests.get(target_url).content
    parsed_html = BeautifulSoup(html, 'html.parser')    for tag in parsed_html
        search_result = re.findall('fn_tt_tt_1">(.*)</a>(.*)</td>', str(tag))
                str_frac = " ".join(search_result[0][0].split()[1:])+", "+sear
                refined_name = str_frac+" "+search_result[0][1].strip()
                refined_name = search_result[0][0]+" "+search_result[0][1].str
    """
    Turn a list of movie name into a list of movie ids. The movie names has to
       Ambiguous movie names can be supplied if useRefineSearch is set to True

    Args:
        list_of_movieName (List): A list of movie names.
        useRefineSearch (boolean): Ambiguous movie names can be supplied if us

    Returns:
        list_of_movie_id (List of strings): A list of movie ids.
    """
    list_of_movie_id = []    for movieName in list_of_movieName:        if use
            movieName = refine_search(movieName)
            print("Refined Name: "+movieName)        if movieName in name_to_m
            list_of_movie_id.append(str(name_to_movieId[movieName]))    return
    recommend_movie_ls = []    if positive_list:
        positive_list = produce_list_of_movieId(positive_list, useRefineSearch
        negative_list = produce_list_of_movieId(negative_list, useRefineSearch
        recommend_movie_ls.append(movieId)    return recommend_movie_ls
```

给出他/她喜欢迪斯尼电影“Up（2009）”，下面的代码显示了对该用户的最喜欢的前五的推荐结果：

```
ls = recommender(positive_list=["UP"], useRefineSearch=True, topn=5)
print('Recommendation Result based on "Up (2009)":')
display(df_movies[df_movies['movieId'].isin(ls)])
```

```
Refined Name: Up (2009)
Recommendation Result based on "Up (2009)":
```

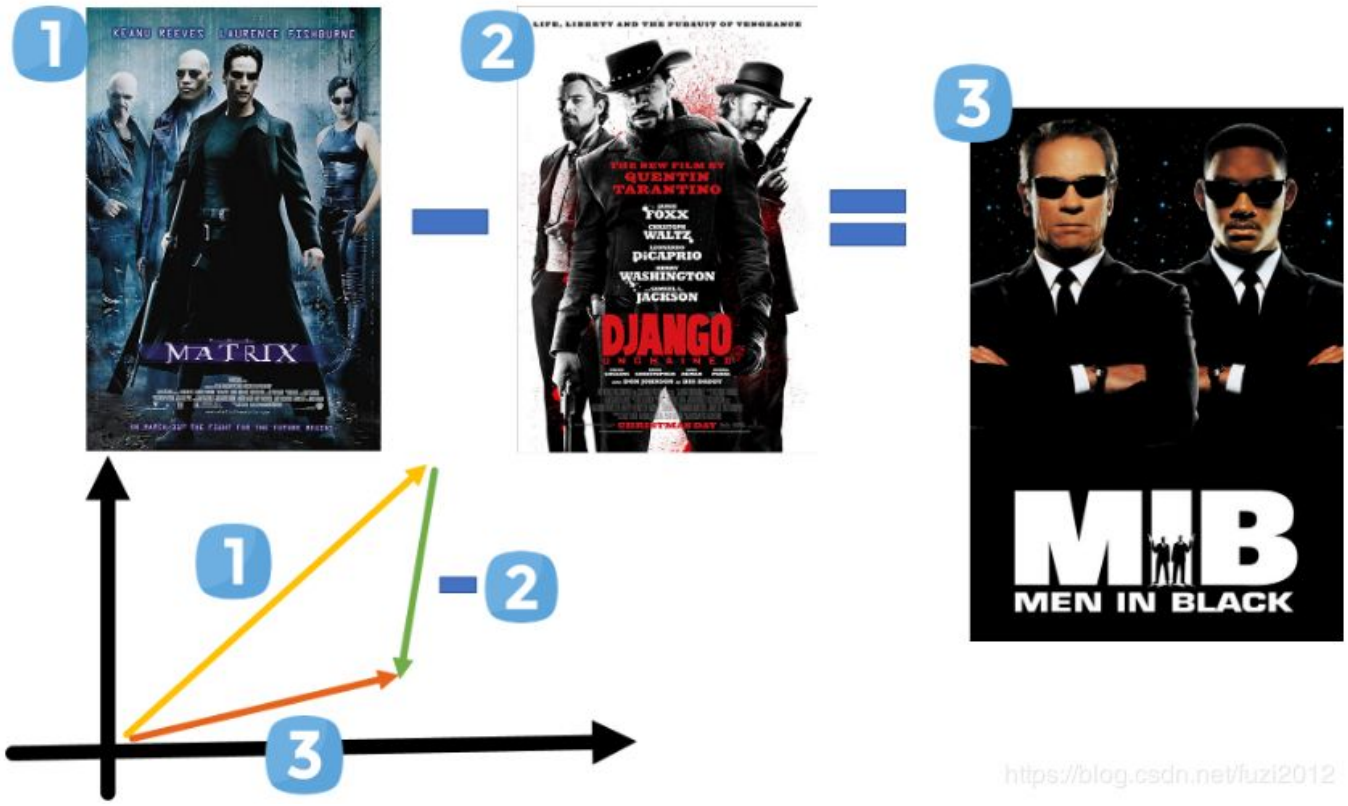
	movieId	title	genres
11614	50872	Ratatouille (2007)	Animation|Children|Drama
12746	60069	WALL·E (2008)	Adventure|Animation|Children|Romance|Sci-Fi
14592	72998	Avatar (2009)	Action|Adventure|Sci-Fi|IMAX
15031	76093	How to Train Your Dragon (2010)	Adventure|Animation|Children|Fantasy|IMAX
15401	78499	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX

再看看更有趣的的，我和我的朋友都喜欢科幻经典小说《“The Matrix (1999)”》。但当谈到昆汀·塔伦蒂诺的标志性作品《Django Unchained (2012)》时，我们有不同的看法。虽然我喜欢荒诞和幽默的讽刺混合，但我的朋友厌恶鲜血和暴力。根据我们的口味，这个模型会推荐什么？把这些数据输入我们的模型，《Men in Black (1997)》和《Ghostbusters (a.k.a Ghost Busters)》都在我朋友的推荐名单上。我有《”Inglourious Basterds (2009)”》《“Inception (2010)”》和《The Dark Knight Rises (2006)》。

```
ls = recommender(positive_list=["The Matrix"], negative_list=["Django Unchaine
print('Recommendation Result based on "The Matrix (1999)" minus "Django Unchai
display(df_movies[df_movies['movieId'].isin(ls)])
```

```
Refined Name: Matrix, The (1999)
Refined Name: Django Unchained (2012)
Recommendation Result based on "The Matrix (1999)" minus "Django Unchained (2012)":
```

	movieId	title	genres
1172	1197	Princess Bride, The (1987)	Action|Adventure|Comedy|Fantasy|Romance
1528	1580	Men in Black (a.k.a. MIB) (1997)	Action|Comedy|Sci-Fi
1557	1610	Hunt for Red October, The (1990)	Action|Adventure|Thriller
1707	1777	Wedding Singer, The (1998)	Comedy|Romance
1839	1923	There's Something About Mary (1998)	Comedy|Romance
2630	2716	Ghostbusters (a.k.a. Ghost Busters) (1984)	Action|Comedy|Sci-Fi
2832	2918	Ferris Bueller's Day Off (1986)	Comedy




```
ls = recommender(positive_list=["The Matrix", "Django Unchained"], useRefineSe
print('Recommendation Result based on "The Matrix (1999)" + ""Django Unchained
display(df_movies[df_movies['movieId'].isin(ls)])
```

```
Refined Name: Matrix, The (1999)
Refined Name: Django Unchained (2012)
Recommendation Result based on "The Matrix (1999)" + ""Django Unchained (2012)":
```

	movieId	title	genres
11401	48780	Prestige, The (2006)	Drama|Mystery|Sci-Fi|Thriller
12525	58559	Dark Knight, The (2008)	Action|Crime|Drama|IMAX
13102	63082	Slumdog Millionaire (2008)	Crime|Drama|Romance
13647	68157	Inglourious Basterds (2009)	Action|Drama|War
14009	70286	District 9 (2009)	Mystery|Sci-Fi|Thriller
15534	79132	Inception (2010)	Action|Crime|Drama|Mystery|Sci-Fi|Thriller|IMAX
18312	91529	Dark Knight Rises, The (2012)	Action|Adventure|Crime|IMAX

这不是一篇刷分博文，就不针对模型的表现做评估了，各位心动的小伙伴可以试试哦。

声明：本文参考一个同性交友平台的某一篇博客。