王耳学推荐 | （九）Item2Vec和Deep Walk

原创 王耳 sad tom cat 2020-06-21

收录于话题
#王耳学推荐

14个

> 在看embedding的内容，这周看的是Item2Vec和Graph Embeding里的Deep Walk，属于embedding里较为基础的知识。笔者水平有限，如果文中有什么纰漏或者逻辑错误，还请读者朋友不吝斧正，万分感激。

简易目录

介绍Item2Vec

Item2Vec[1]，顾名思义，就是为item做embedding的方法，将skip-gram使用在协同过滤的问题上。既然是协同过滤，那么首先要确认item2vec的使用场景————适用于item-based CF（基于物品的协同过滤）。

文中提到的业务问题，是在windows 10应用商店的推荐问题：在“用户也喜欢”该怎么做推荐？如下图所示。

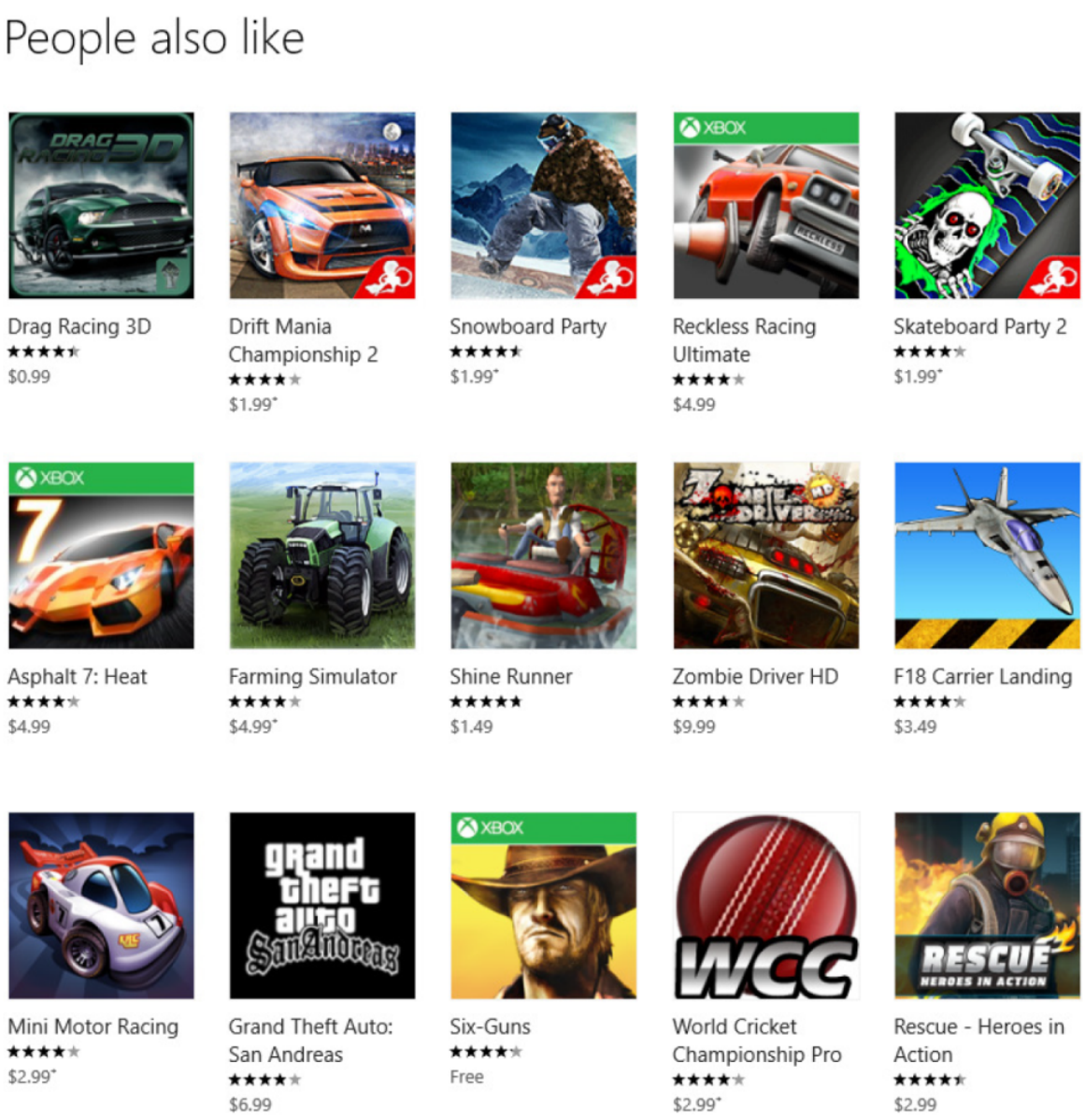


Figure 1: Recommendations in Windows 10 Store based on similar items to Need For Speed.

在这样一个明确的推荐场景下，基于具体商品的推荐是更有优势的，可根据上图进行说明：

- 已经得知用户的兴趣点是由Need For Speed引导出来的，那么在上下文关系中，由该游戏为基础进行的协同过滤更符合逻辑。此处可以类比于电商场景下商品详情页的推荐。
- 在应用商店中，user的数量远大于item的数量。学习item之间相关关系的计算压力要远小于学习item与user的关系。（user和user就更不可能了，又不是社交场景）

既然问题定位到学习item之间的相关关系，这里使用skip-gram的做法就是Item2Vec的灵魂所在了。Item2vec并没有加入对时间信息的考量，是这么解释的：

> By moving from sequences to sets, the spatial / time information is lost. We choose to discard this information, since in this paper, we assume a static environment where items

> that share the same set are considered similar, no matter in what order / time they were generated by the user.

简而言之，序列数据产生的时间并不重要。继而，序列内item的相对位置也不重要————只要是在一个序列内，所有item组成的pair对都是可以作为正样本的，也就是skip-gram的window size是自适应的，是和序列长相等。

> Since we ignore the spatial information, we treat each pair of items that share the same set as a positive example. This implies a window size that is determined from the set size.

Item2Vec的文章并不长，具体的内容只有5页，感兴趣的读者朋友可以看看原文。

插一句题外话，笔者的毕业论文也是使用这样的skip-gram模型，当时参考的方法是**P2V-MAP**[2]：一个购物篮就是一个序列，对购物篮内的商品做embedding，同样也不考虑商品在购物篮内的相对位置。

介绍Deep Walk

Deep Walk[3]是KDD 2014年的文章，它属于Graph Embedding里比较经典算法。（14年也不算太遥远吧？尽管如此，这方法不算新了。）理清Deep Walk里的工作原理，方便后续更深层次的理解。

一言以蔽之，Graph Embedding是Word2Vec方法在图结构上的大胆尝试，将图上的结点转为低维向量表示。而Deep Walk是**在物品组成的图(结点，邻接矩阵，权重矩阵等)中随机游走获得序列数据，对序列数据做word2vec获取embedding。**（经典到一句话就能总结出它的做法。）

那么技术难点就在于：

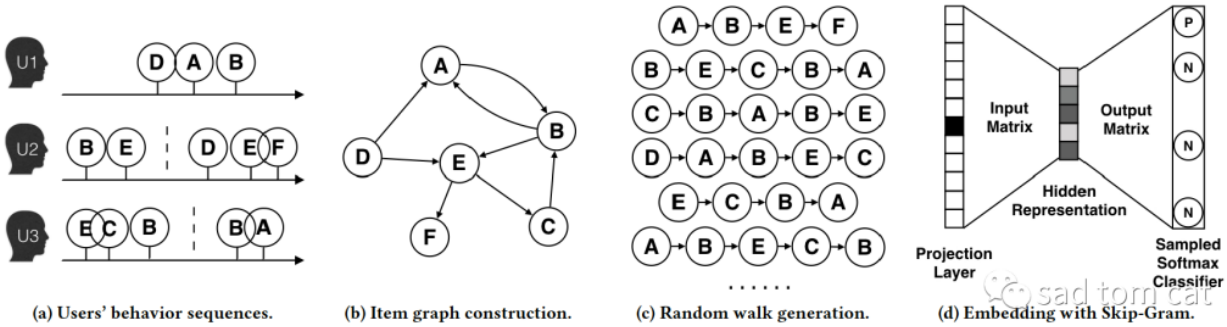
- 如何获取物品组成的图？
- 怎么随机游走，获取序列？
- 怎么做embedding？

逐一解释。

1、获取物品的图

这里参考了阿里的论文[4]和王喆的博客[5]。物品之间的边连接是通过用户行为序列得出的，如下图所示。U2的序列被分成两个子序列，子序列内item的顺序就可以转为图中vertex的指向

关系。总结全部用户的序列，item前后关系出现的次数也可以相对应地转为vertex的边权重。



(a) Users' behavior sequences. (b) Item graph construction. (c) Random walk generation. (d) Embedding with Skip-Gram.

2、随机游走的方法

直接上Deep Walk论文里的原图。

Algorithm 1 DEEPWALK(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

1: Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
2: Build a binary Tree T from V
3: **for** $i = 0$ to γ **do**
4: $\mathcal{O} = \text{Shuffle}(V)$
5: **for each** $v_i \in \mathcal{O}$ **do**
6: $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
7: SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
8: **end for**
9: **end for**

事先规定随机游走的序列长度t，外循环的次数γ。所有的结点都进行一次游走(特别注明，第4行的shuffle操作是方便后续的随机梯度下降)。外循环部分是可以并行操作的，这里的代码参考了github.com/shenweichen/GraphEmbedding[6]，使用的是joblib模块里的Parallel和delayed（感兴趣的读者朋友可以试试）。

随机游走的代码：

```
class Deep_Walker:
    def __init__(self, data_cate_path, data_link_path, walk_per_vertex,
            walk_length, n_workers=1, verbose=1):

        self.vertex, self.edge = read_graph_data(data_cate_path, data_link_path)
        self.walk_per_vertex = walk_per_vertex
        self.walk_length = walk_length
        self.vertex_list = sum(list(self.vertex.values()), [])
        self.walks_generated = []

        self.run(walk_length, walk_per_vertex, n_workers, verbose)

    def walk(self, start_vertex, walk_length):
        walk_path = [start_vertex]
        while len(walk_path) < walk_length:
            node_from = walk_path[-1]
            if not self.edge[node_from]:
                break
            walk_path.append(random.choice(self.edge[node_from]))
        return walk_path

    def deep_walk(self, walk_length, walk_per_vertex):
        res = []
        for _ in range(walk_per_vertex):
            random.shuffle(self.vertex_list)
            for v in self.vertex_list:
                one_walk = self.walk(v, walk_length)
                res.append(one_walk)
        return res

    def run(self, walk_length, walk_per_vertex, n_workers=1, verbose=1):
        res = Parallel(n_jobs=n_workers, backend="threading", verbose=verbose)(
            delayed(self.deep_walk)(walk_length, num)
            for num in partition_num(walk_per_vertex, n_workers))
        self.walks_generated = sum(res, [])
        return self.walks_generated
```

其中的两个辅助函数：

```
from collections import defaultdict

def read_graph_data(data_cate_path, data_link_path):
    vertex = defaultdict(list)
```

```
    with open(data_cate_path, 'r+') as f:
        for line in f.readlines():
            node_num, cate_num = line.strip().split(' ')
            vertex[cate_num].append(node_num)

    edge = defaultdict(list)
    with open(data_link_path, 'r+') as f:
        for line in f.readlines():
            node_from, node_to = line.strip().split(' ')
            edge[node_from].append(node_to)
    return vertex, edge

def partition_num(num, workers):
    if num % workers == 0:
        return [num//workers]*workers
    else:
        return [num//workers]*workers + [num % workers]
```

3、skip-gram作embedding

这里的skip-gram没有太多的考究，如下所示。

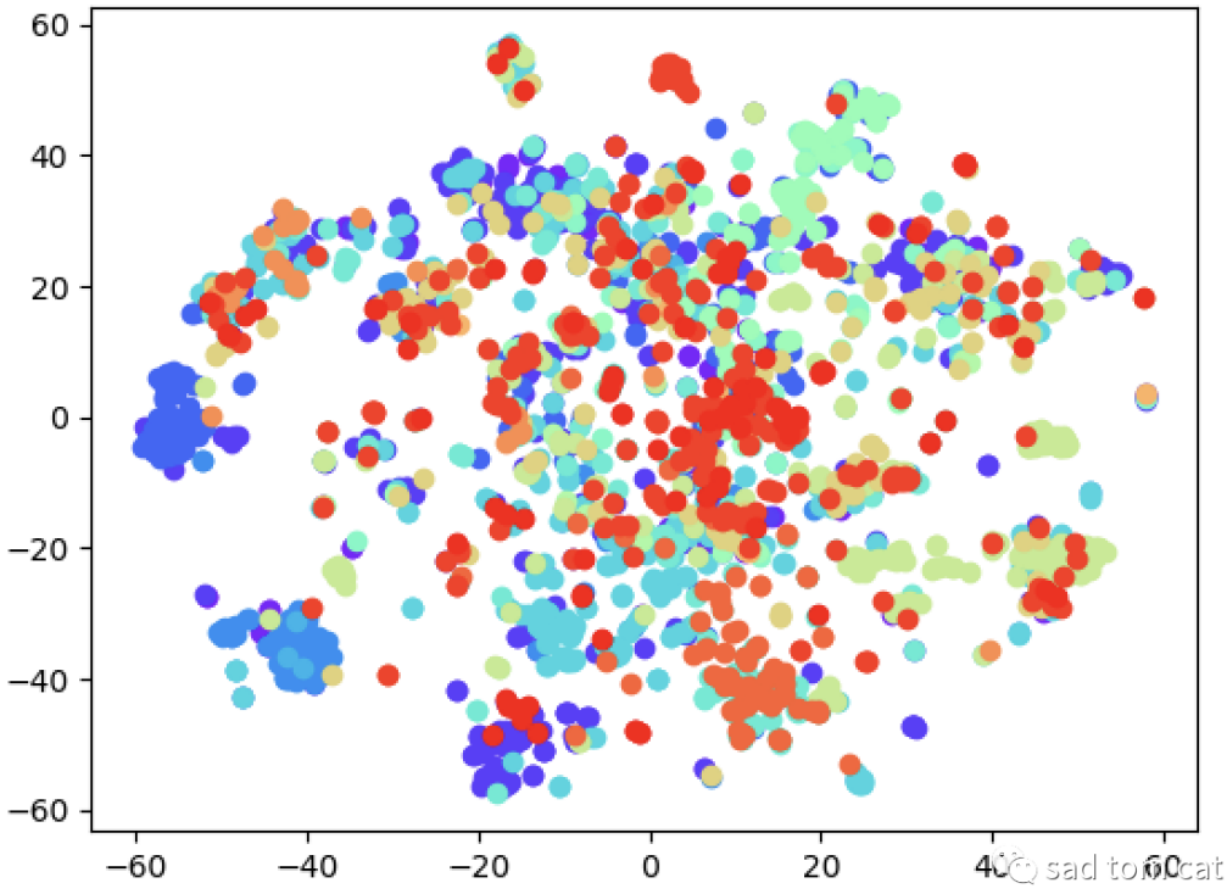
Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

1: **for each** $v_j \in \mathcal{W}_{v_i}$ **do**
2: **for each** $u_k \in \mathcal{W}_{v_i}[j-w : j+w]$ **do**
3: $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
4: $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
5: **end for**
6: **end for**

不过原文中特地强调使用hierarchical softmax进行加速训练。(虽然笔者觉得用negative sampling也可以)

> We can speed up the training process further, by assigning shorter paths to the frequent vertices in the random walks. Huffman coding is used to reduce the access time of frequent elements in the tree.

最后插一张图，是对DeepWalk的embedding结果做t-sne的结果图（颜色相同代表结点属于相同的类）。



小结

- Item2Vec舍弃时间信息，对skip-gram做出一定的改造，是一种描述item间关系的新思路。
- Deep Walk通过随机游走获得序列数据，对图中结点做embedding。但成也萧何败也萧何，随即游走也是它的一个弱点——没有记忆特定的结构，结点的空间关系全部要依托序列数据，这一点在**LINE**[7]的论文也有强调。LINE在损失函数里加入了结点间一级邻接(First-order Proximity)和二级邻接(Second-order Proximity)的考虑。

参考资料

[1] Item2vec: Neural Item Embedding for Collaborative Filtering: *https://arxiv.org/pdf/1603.04259v2.pdf*

[2] P2V-MAP: Mapping Market Structures for Large Retail Assortments: *https://journals.sagepub.com/doi/10.1177/0022243719833631*

[3] DeepWalk: Online Learning of Social Representations: *https://arxiv.org/pdf/1403.6652v2.pdf*

[4] Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba: *https://arxiv.org/pdf/1803.02349.pdf*

[5] 深度学习中不得不学的Graph Embedding方法: *https://zhuanlan.zhihu.com/p/64200072*

[6] github.com/shenweichen/GraphEmbedding: *https://github.com/shenweichen/GraphEmbedding*