**实验3：Graph Embedding 算法——DeepWalk**

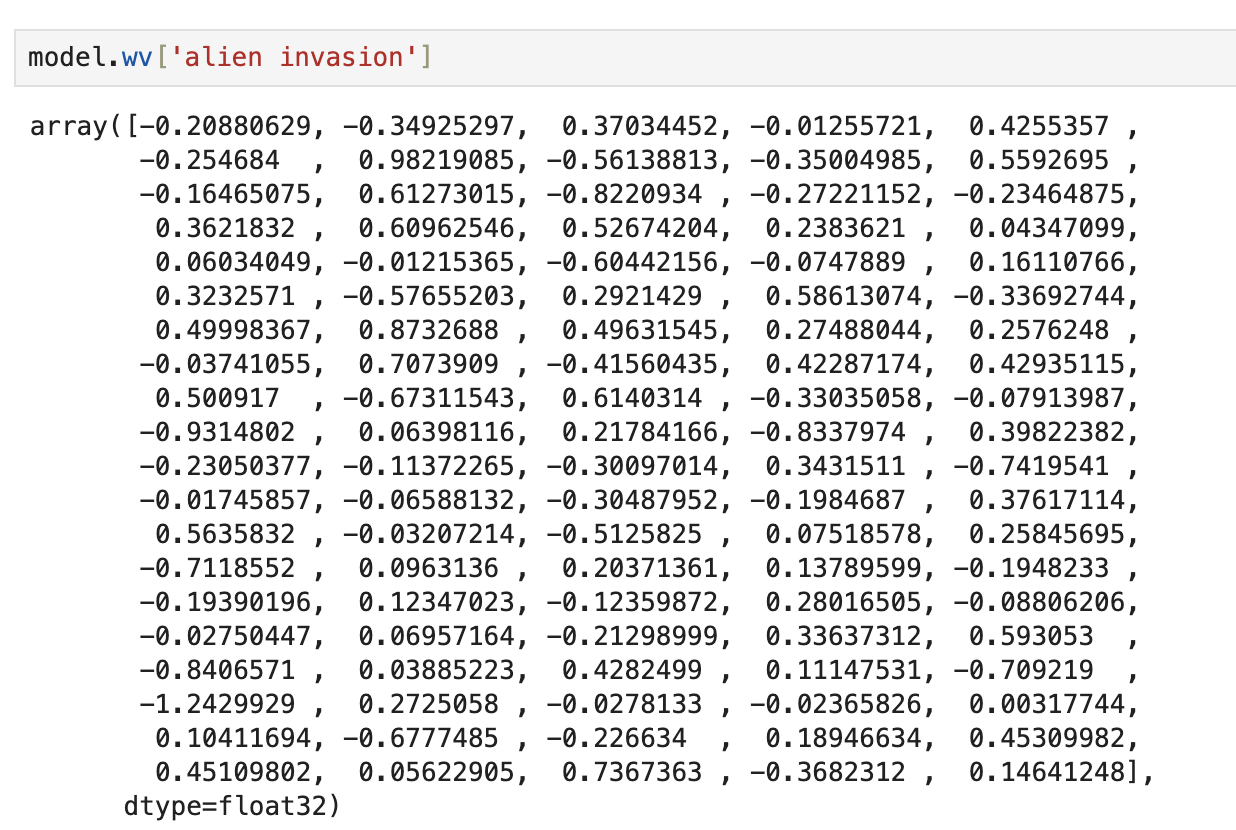
1. 简述DeepWalk算法

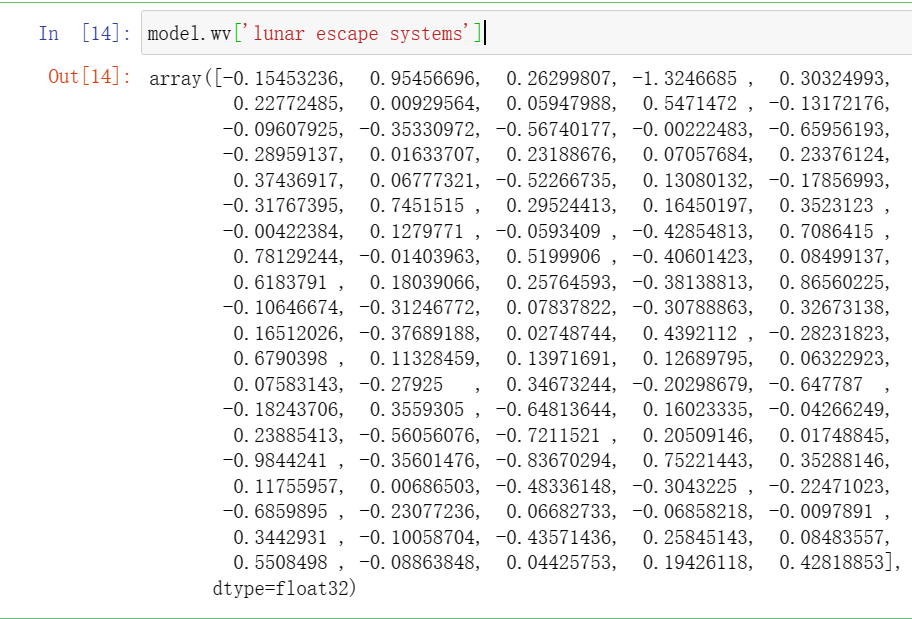
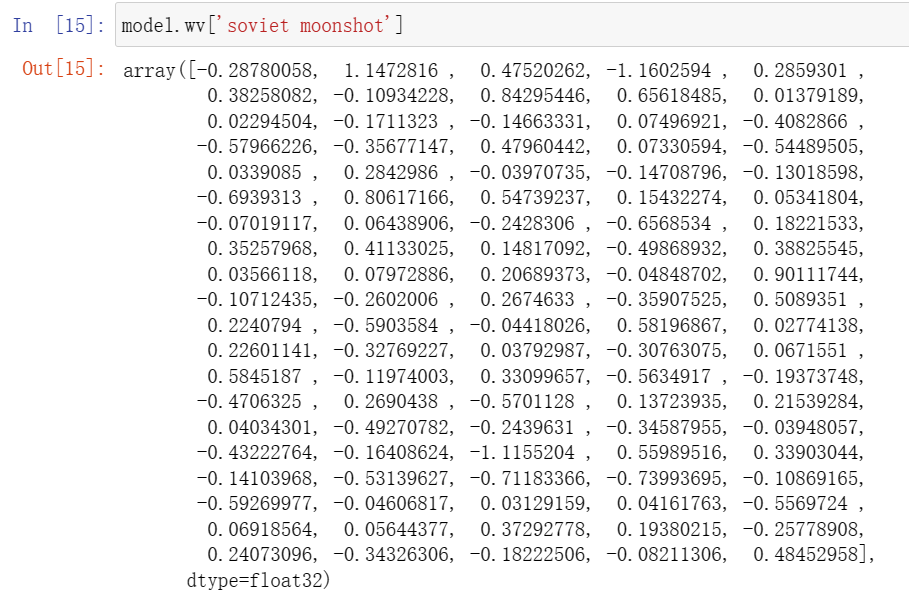
DeepWalk算法是一种学习图（如社交网络、词汇共现网络等）中节点表示的算法，通过将图数据转换为序列数据，使用自然语言处理（NLP）中的技术来学习节点的向量表示。DeepWalk的核心思想是利用图中节点的邻居信息来学习每个节点的低维度特征表示，这些特征可以用于后续的图分析任务，比如节点分类、链接预测或者图聚类。

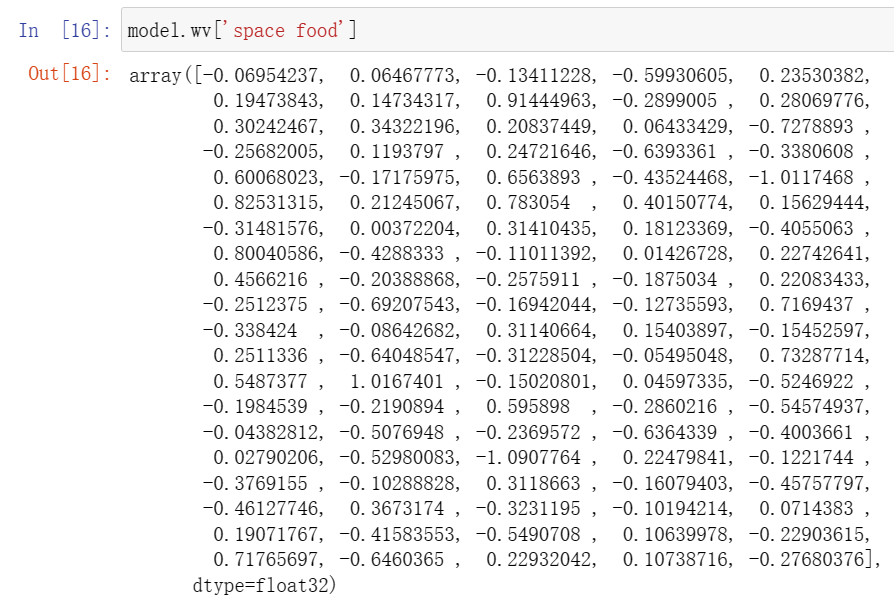
DeepWalk算法主要包括以下几个步骤：

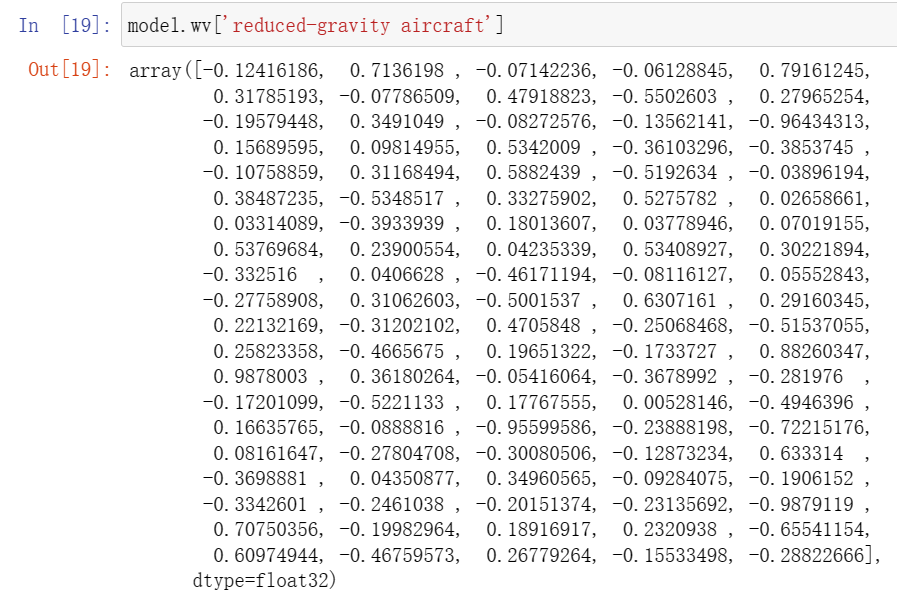
1. 随机游走：对图中的每个节点进行随机游走，生成路径。每一步的转移是随机选择当前节点的一个邻居。通过这种方式，可以为每个节点生成一个节点序列，这个序列捕获了图中节点的邻域信息。
2. 序列处理：将上一步骤生成的节点序列看作是句子，其中的节点类比于句子中的单词。这样，可以使用NLP中的技术来处理这些“句子”。
3. 学习节点表示：采用Word2Vec模型（特别是Skip-Gram模型）来学习节点的向量表示。Skip-Gram模型的目标是通过当前节点的表示来预测其上下文节点的概率。在DeepWalk中，一个节点的“上下文”就是在随机游走过程中与该节点相邻的其他节点。
4. 优化和训练：通过优化Skip-Gram模型的损失函数来训练节点的向量表示，通常使用随机梯度下降（SGD）或其变种进行优化。
5. DeepWalk核心代码
   1. RandomWalk函数
6. **def** get\_randomwalk(node, path\_length):
8. random\_walk = [node]
10. **for** i **in** range(path\_length-1):
11. temp = list(G.neighbors(node))
12. # Add your code here1
13. temp = list(set(temp) - set(random\_walk))
14. **if** len(temp) == 0:
15. **break**
17. random\_node = random.choice(temp)
18. random\_walk.append(random\_node)
19. node = random\_node
21. **return** random\_walk
    1. DeepWalk实现
22. model = Word2Vec(window = 4, sg = 1, hs = 0,
23. negative = 10,
24. alpha=0.03, min\_alpha=0.0007,
25. seed = 14)
27. model.build\_vocab(random\_walks, progress\_per=2)
28. 输出三个以上点的embedding

样例：



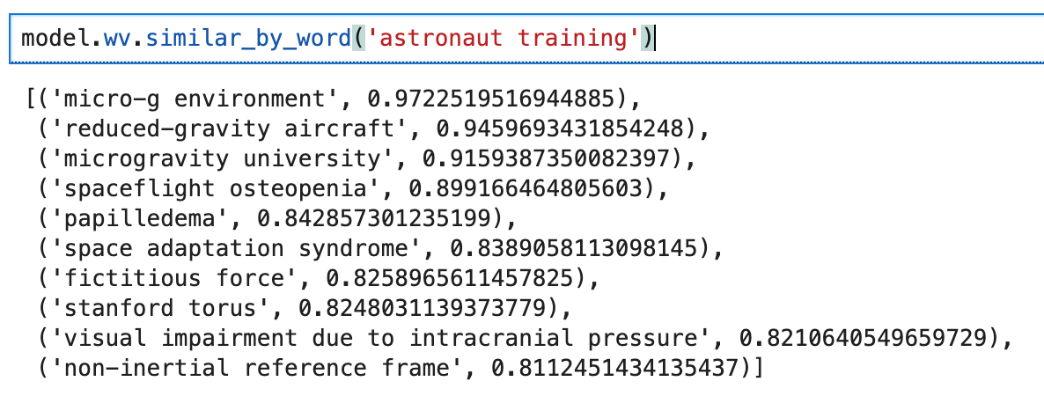
 

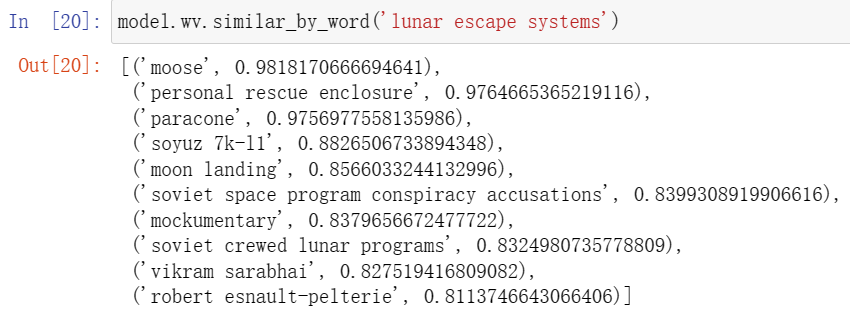


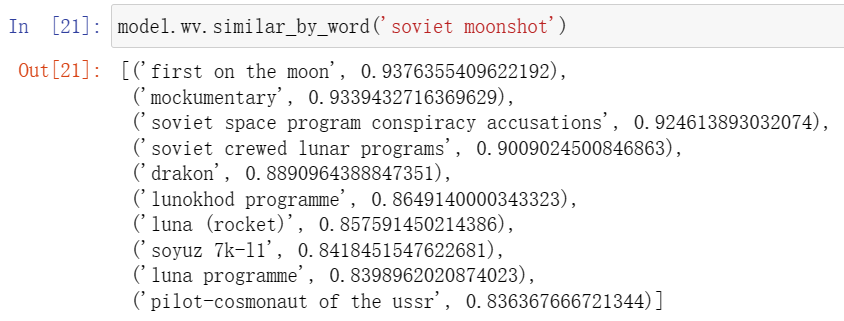


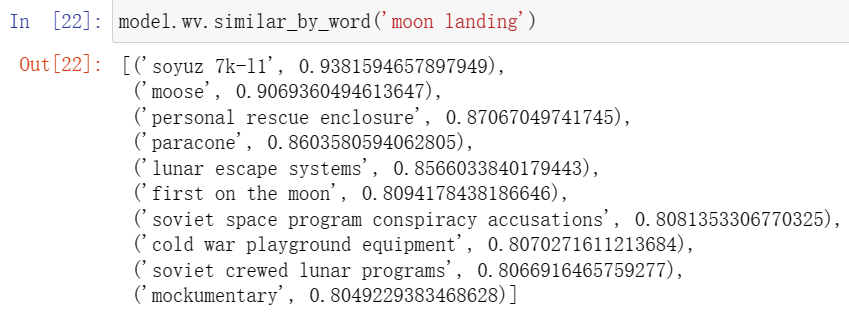
1. 测试三个以上节点的相似节点及相似度

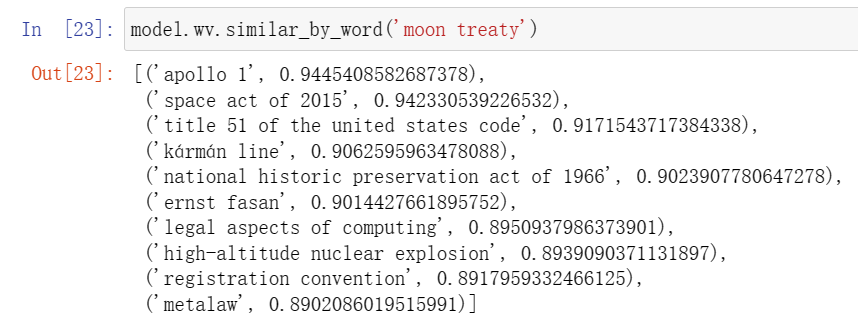
样例：









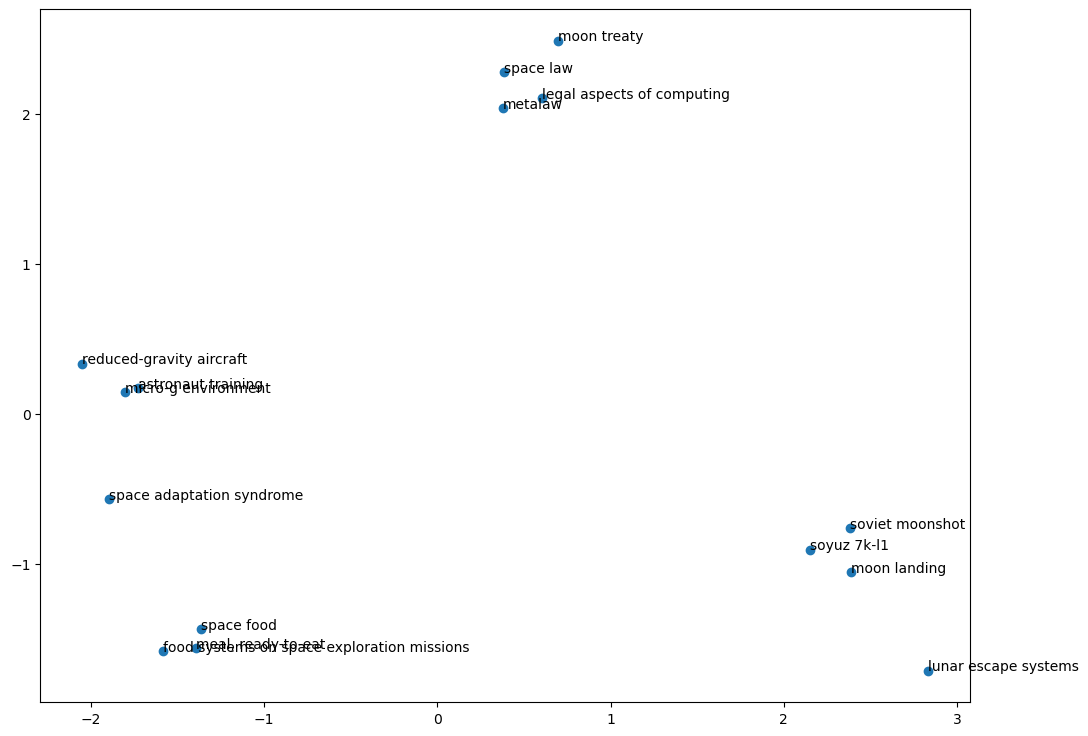


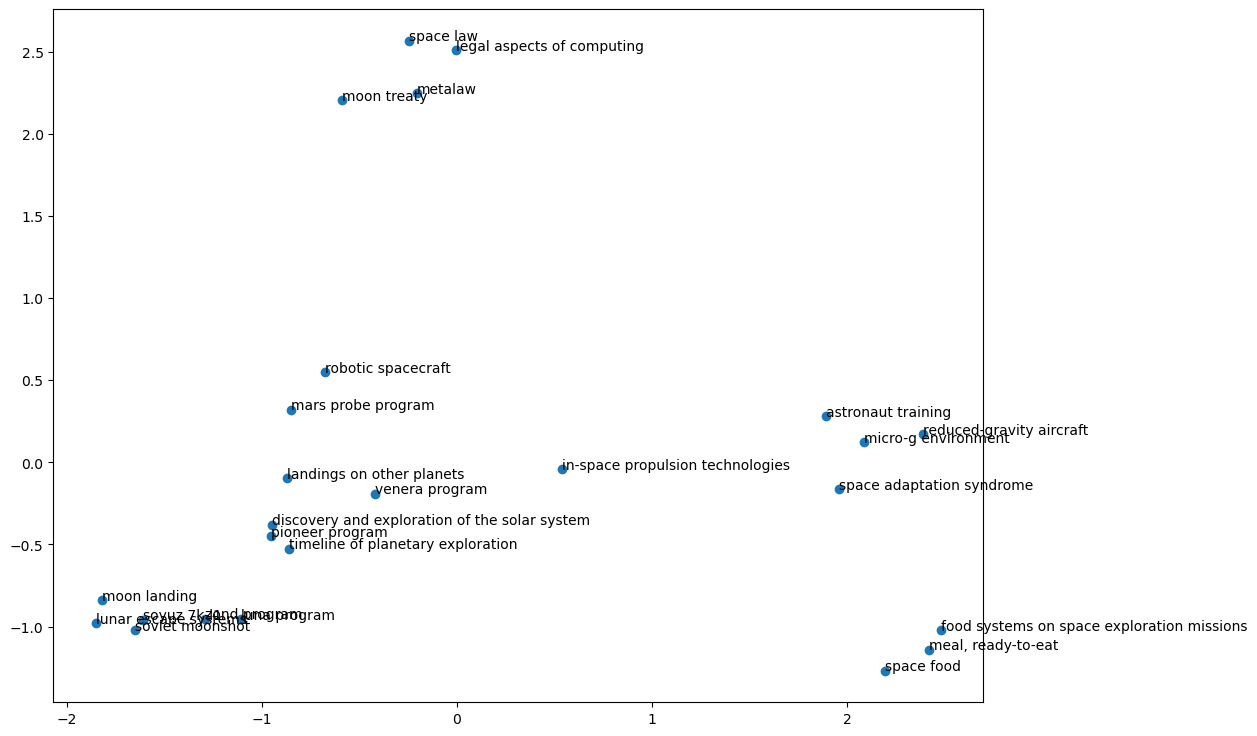
1. 获得二维embedding分布图（节点数>20）

要求：明显能看出相似节点成簇

样例：

/\*源代码词组数量不够，自己往上加了一些\*/





1. (Bonus) 尝试不同参数配置并对比分析结果

Tips: a) 改变RamdomWalk的路径长度获得不同长度的节点序列

b) 改变word2vec的参数设置

c) 分析不同参数对结果的影响

首先，对deepwalk算法中的参数进行分析

RamdomWalk的路径长度决定了每个节点生成的随机游走序列的长度

word2vec的参数分析

1. window:窗口大小,即在随机游走序列中考虑的上下文节点数。

- 较大的window可以捕捉更多的上下文信息,学习到更全面的节点表示。但过大的window可能引入噪声,增加计算开销。

- 较小的window则侧重于节点的局部结构信息,忽略了更远的上下文。

- 需根据具体任务和图的特点选择合适的window大小,一般取值在3-10之间。

2. sg:训练算法,1表示Skip-Gram,0表示CBOW。

- Skip-Gram对罕见词(low-frequency words)的表示质量更好,但训练较慢。适合大规模语料。

- CBOW训练更快,对高频词的表示更准确。小数据集上表现更优。

- 图中节点没有低频高频之分,两种算法差别不大。sg=1更常用。

3. hs:hierarchical softmax,用于优化模型训练。1启用,0禁用。

- hs=1时用Huffman树对输出概率进行估计,加速训练并提升罕见词的表示质量。

- hs=0时使用negative sampling,采样部分负样本进行训练。

- 当词表很大时,hs=0训练更快。DeepWalk中hs一般设为0。

4. negative:negative sampling的采样数。值越大,训练越慢,样本利用率越高。5-20为宜。

5. alpha:初始学习率。值越大,收敛越快,但可能错过最优解。

6. min\_alpha:最小学习率。线性递减learning rate时的下界,低于这个值则训练停止。

7. seed:随机种子,保证每次运行都得到相同的结果,便于复现和调参。

综上,以下为DeepWalk常用的参数设置:

- window=10 捕捉多尺度结构信息

- sg=1 使用Skip-Gram算法

- hs=0 使用negative sampling加速训练

- negative=10 适度的采样数

- alpha=0.025 常用的初始学习率

- min\_alpha=0.0001 训练充分进行

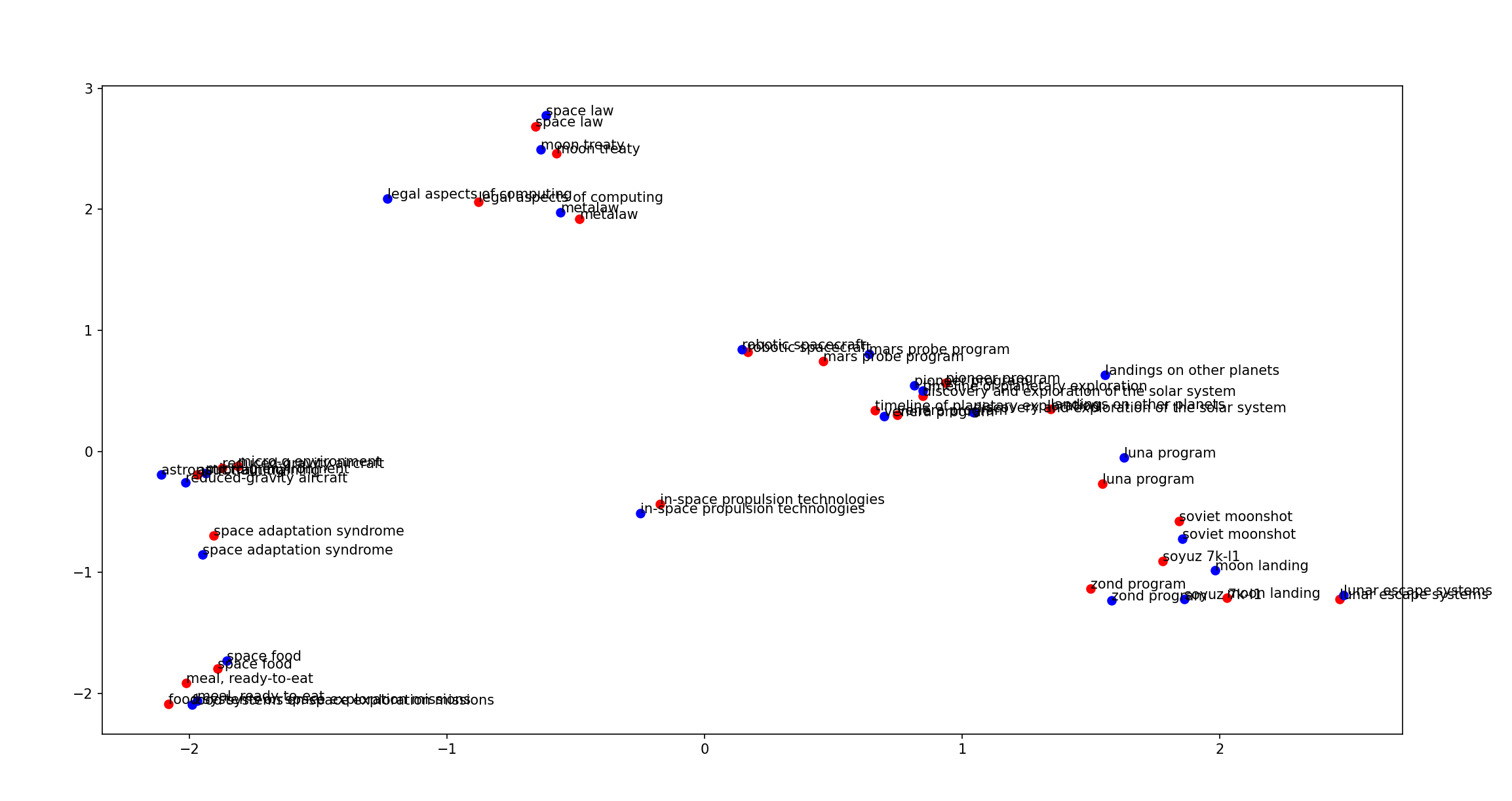
- seed=随机整数 便于复现结果

还有一个参数是训练的epochs

通过分析PCA后的散点图来分析修改参数对实验结果的影响

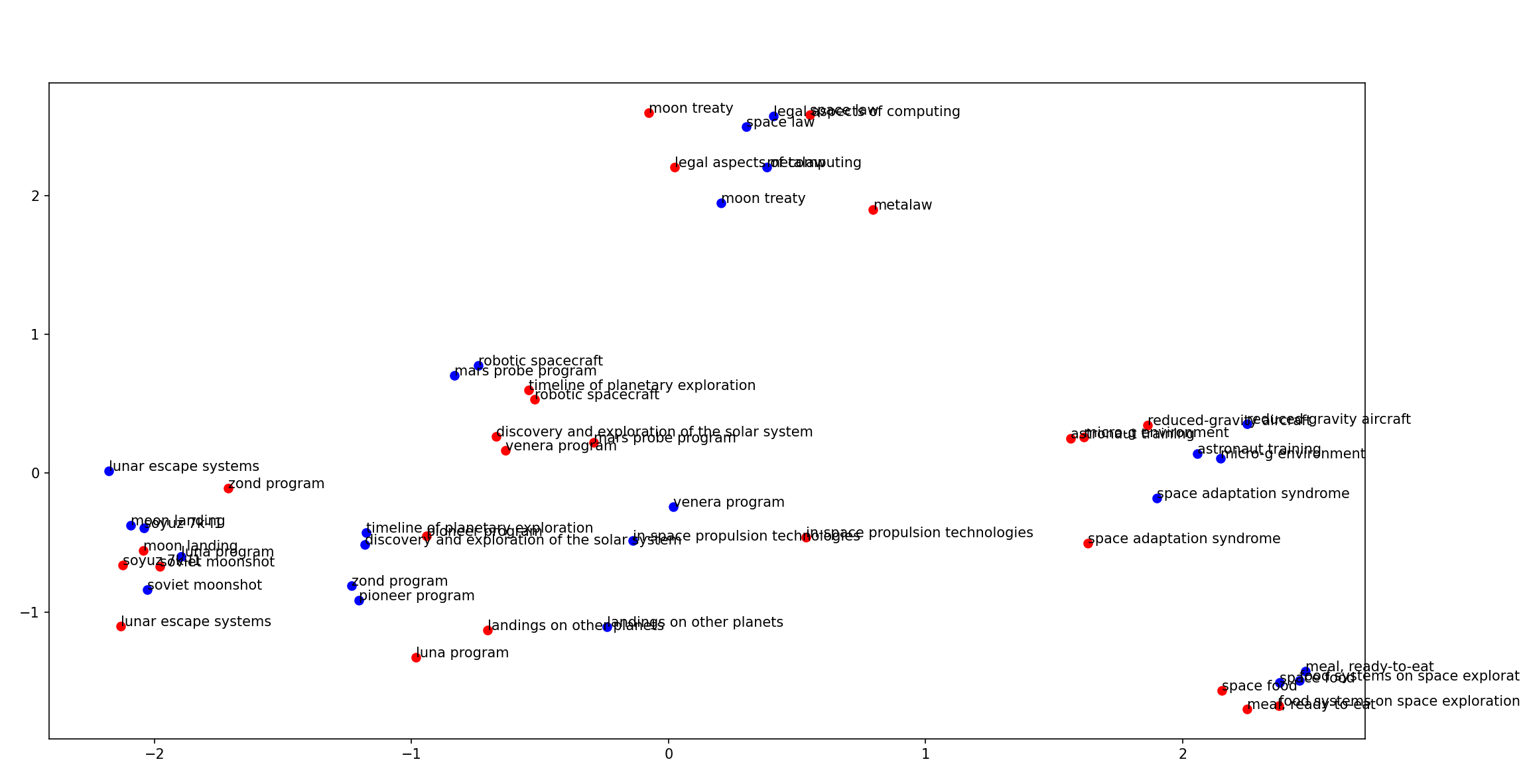
单独对epoch进行修改

红色为epochs = 20，蓝色为epochs = 30

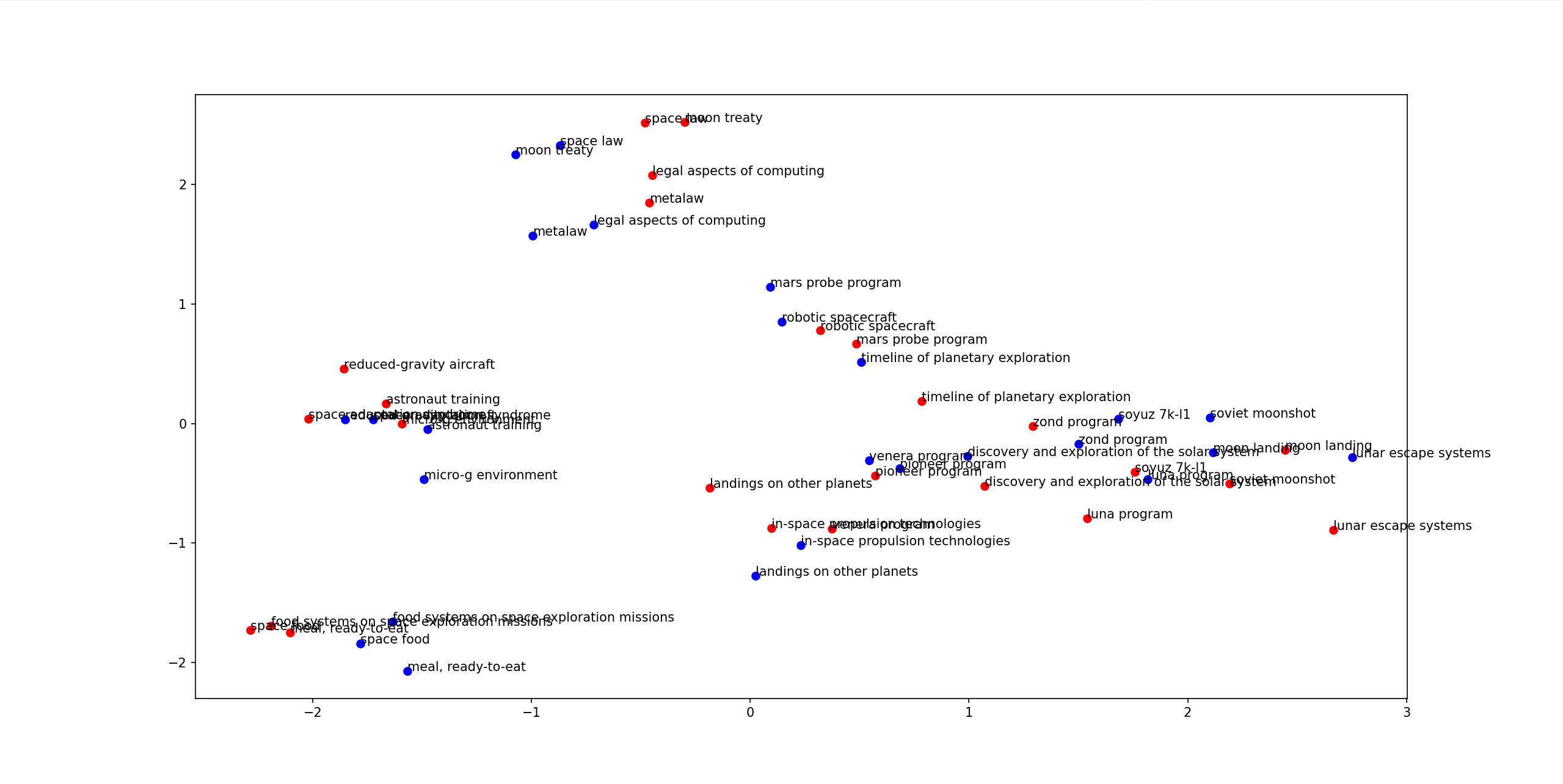


单独对path\_length进行修改

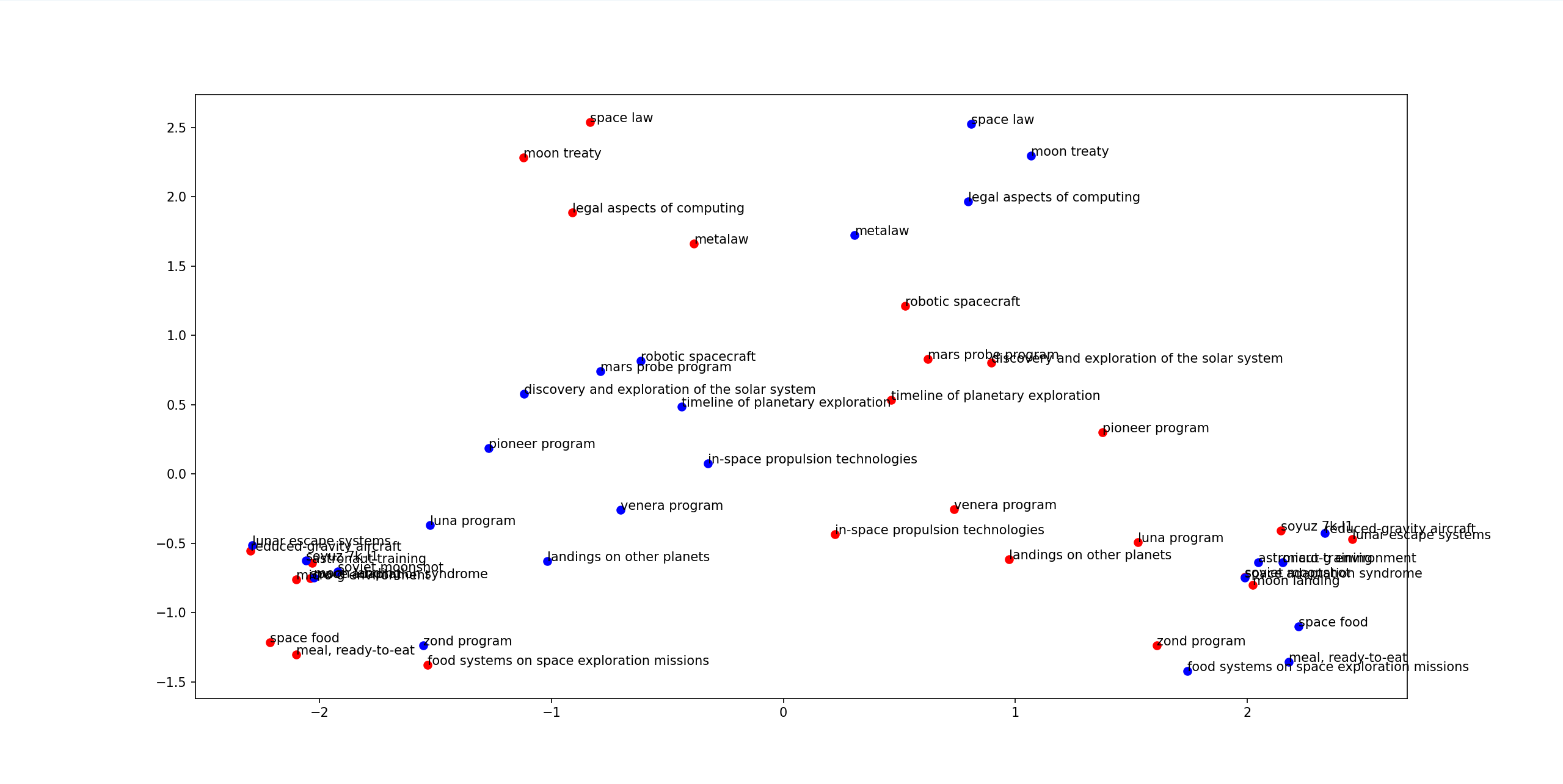
红色path\_length = 10,蓝色path\_length = 20



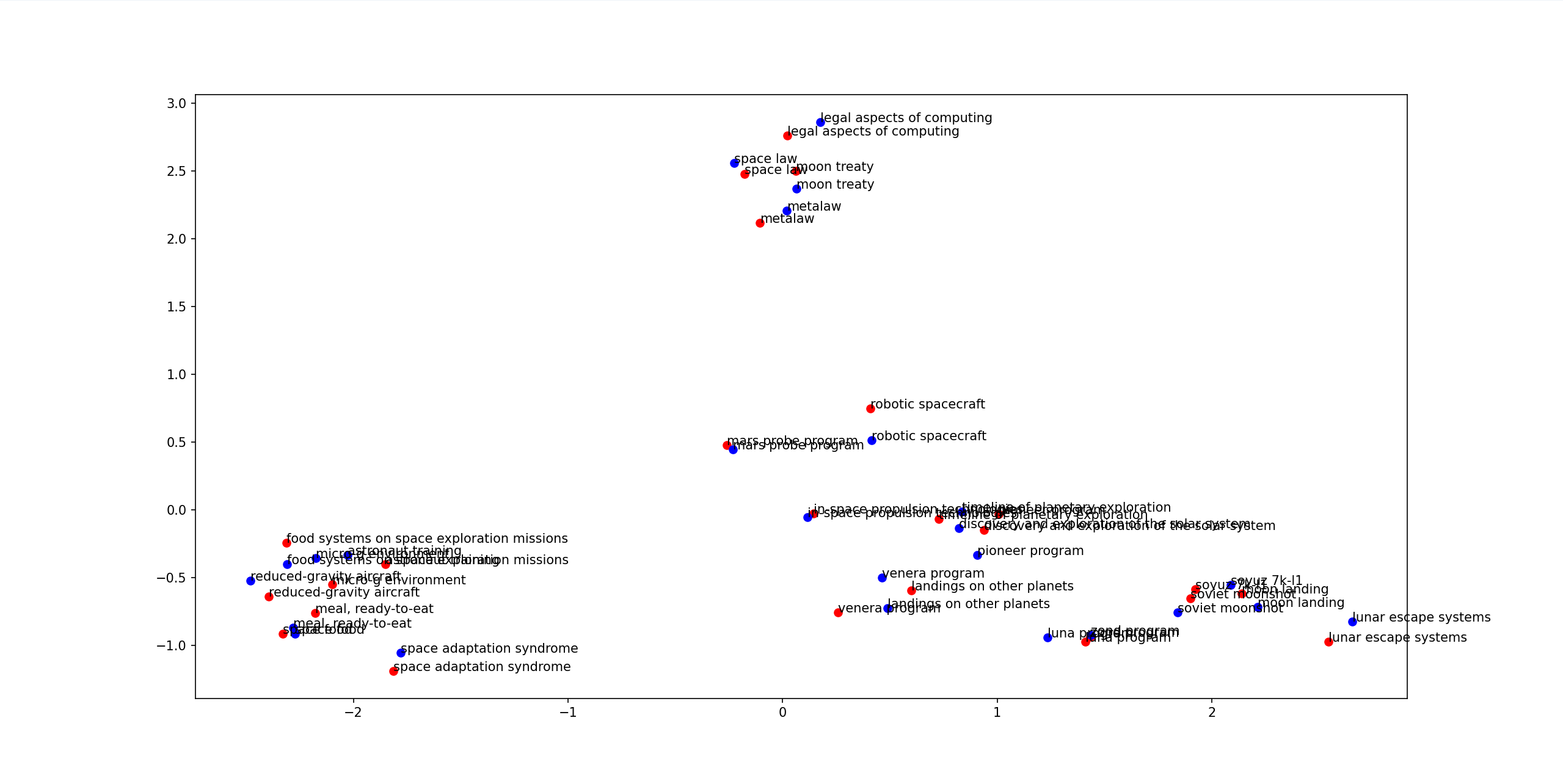
单独对Windows进行修改，红色Windows = 4，蓝色Windows = 8



单独对learning\_rate进行修改，红色learning\_rate = 0.0007，蓝色为0.001



单独对negative修改，红色为10，蓝色为18



由此可以看出，算法中的参数的微小调整对实验结果影响较大。