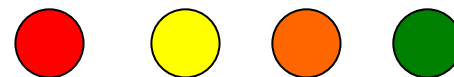
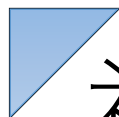


# 网络社区发现 C12

信息科学与技术学院

胡俊峰



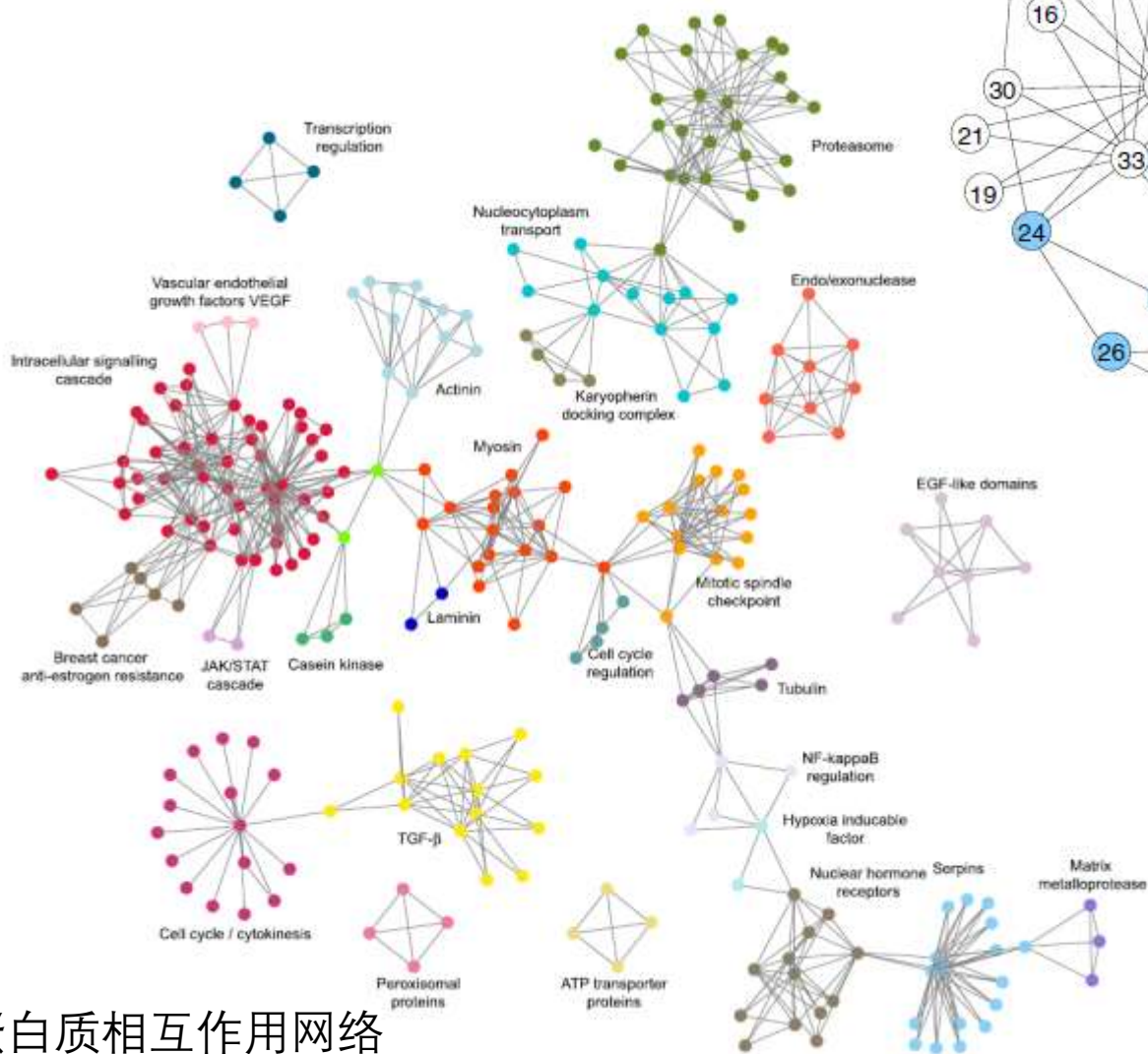


# 社区发现与谱聚类算法

- 基于分割的聚类
  - 谱聚类算法
- 标签传播与网路社区发现
- 词义上下文嵌入-word embedding
- 图嵌入算法
- 观影数据分析

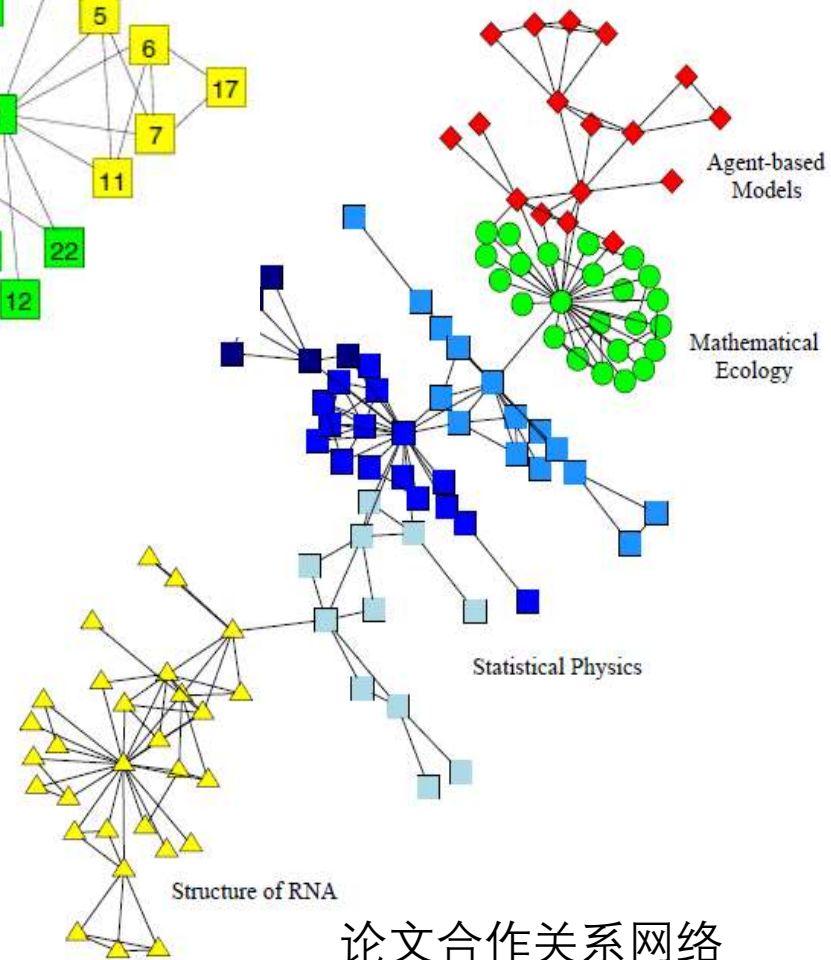
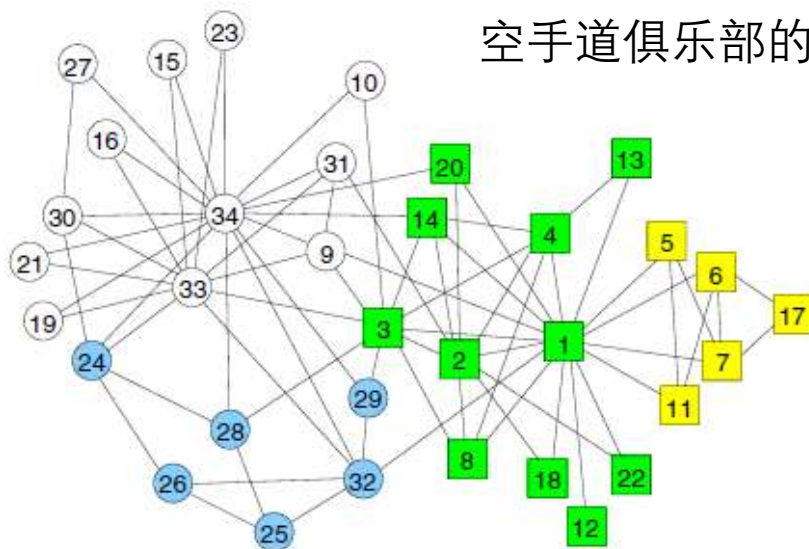


# 网络社区结构



蛋白质相互作用网络

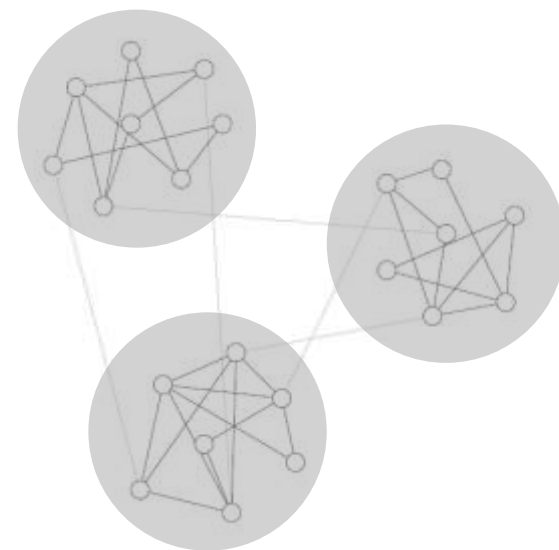
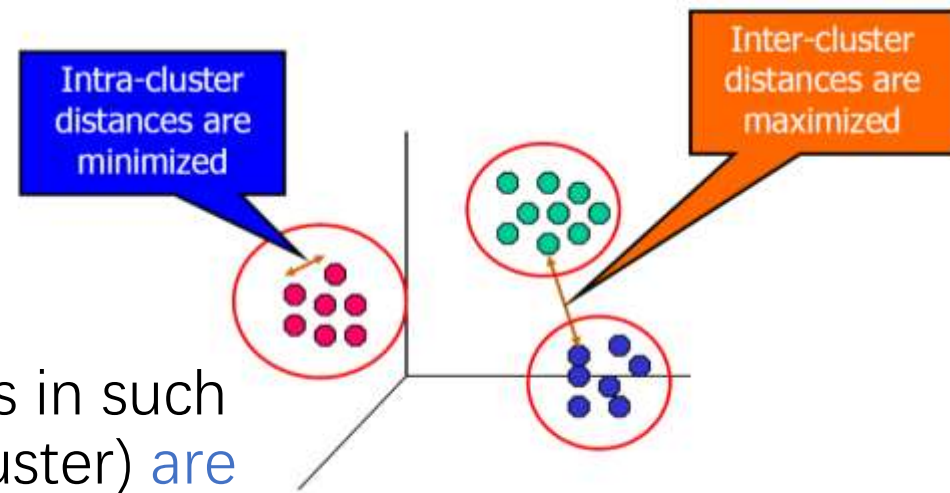
空手道俱乐部的人际关系网络



论文合作关系网络

# 聚类

- 维基百科上对于聚类的定义：
- **Clustering** is the task of **grouping** a set of objects in such a way that objects **in the same group** (called a cluster) **are more similar** (in some sense or another) to each other **than** to those **in other groups** (clusters).
- 对于什么是“类”，没有统一的、定量的定义，通常与聚类算法有关。
- 图聚类：类内连接尽可能紧密，类间连接尽可能松散。

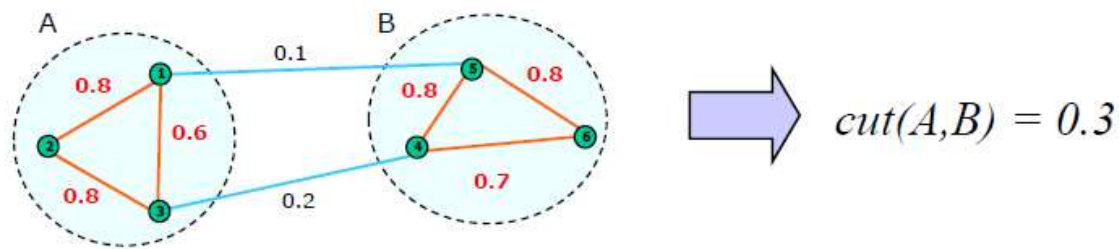


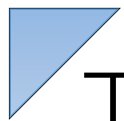
# 基于分割的聚类

- 用一个**最小切分**，将图分割成两个或者多个子图。
- 切分 (cut)：两个子图之间连边的边权和

$$Cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$$

- 倾向于把连接数少的顶点切出来。（极端情况，一个类里一个点也没有，另一个类里有所有的点。Cut = 0）





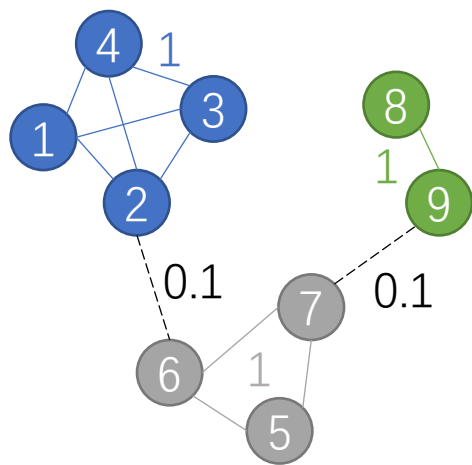
# The Kernighan-Lin Algorithm

- 指定类的个数和类的大小。例如把所有的节点平均分成两个类。
- KL算法：
  1. 首先，随机把图分成相等的两个部分。
  2. 交换不属于同一类的一对顶点，算出交换后cut size减小的值，选择cut size减小最多（或者增长最少）的一对顶点，交换。
  3. 每个顶点在一轮里只能被交换一次。重复第2步，直到某一类中所有的顶点都被交换过了，回溯交换的过程，取cut size最小时的交换状态，作为这一轮的结果。
  4. 重复2-3步，直到cut size不再减小。
- 如果分类个数大于2，可以重复二分。但不能保证解最优。
- 缺点：依赖初始状态的选取。
- 通常以其他算法的结果作为初始状态，利用KL算法优化调整其他算法得到聚类结果。



# 谱聚类

- 谱聚类：利用**特征值/特征向量**方法聚类的技术
  - 拉普拉斯矩阵、邻接矩阵等。
- 将图上的顶点投影到前K个特征向量张成的空间中，在这个空间里做聚类（比如用K-means），可以解决直接用K-means分不出的类。（比如非凸集）



Laplacian 矩阵的特征向量（对应特征值左小右大）：

1	0.3333	0.3482	0.1437	0.0008	0.0146	-0.0226	0.8035	0.145	-0.2822
2	0.3333	0.3366	0.1289	-0.0009	-0.0297	0.0474	0	0	0.8694
3	0.3333	0.3482	0.1437	0.0008	0.0146	-0.0226	-0.2762	-0.7684	-0.2822
4	0.3333	0.3482	0.1437	0.0008	0.0146	-0.0226	-0.5273	0.6234	-0.2822
5	0.3333	-0.1407	-0.4661	-0.0369	0.8049	0.0476	0	0	0.0207
6	0.3333	-0.1251	-0.4462	-0.0335	-0.4582	0.6773	0	0	-0.0658
7	0.3333	-0.1515	-0.438	0.0353	-0.374	-0.7297	0	0	0.0229
8	0.3333	-0.4901	0.4166	-0.6888	-0.0127	-0.0228	0	0	0.0003
9	0.3333	-0.4737	0.3738	0.7224	0.0259	0.0479	0	0	-0.0009



# 谱聚类

- 拉普拉斯矩阵:

$$L = D - W$$

其中,  $W$ 是有向无环图的邻接矩阵。 $D$ 是节点的度数矩阵（对角矩阵）， $D_{ii} = d_i$ .

- 拉普拉斯矩阵有以下性质:

1. 对于任意的  $f \in R^n$

$$f^T L f = \frac{1}{2} \sum_{i,j} w_{ij} (f_i - f_j)^2$$

2.  $L$ 是对称且半正定的。
3.  $L$ 最小的特征值为0, 对应的特征向量为全1向量。
4.  $L$ 有 $n$ 个非负、实数特征值  $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ .





# 谱聚类

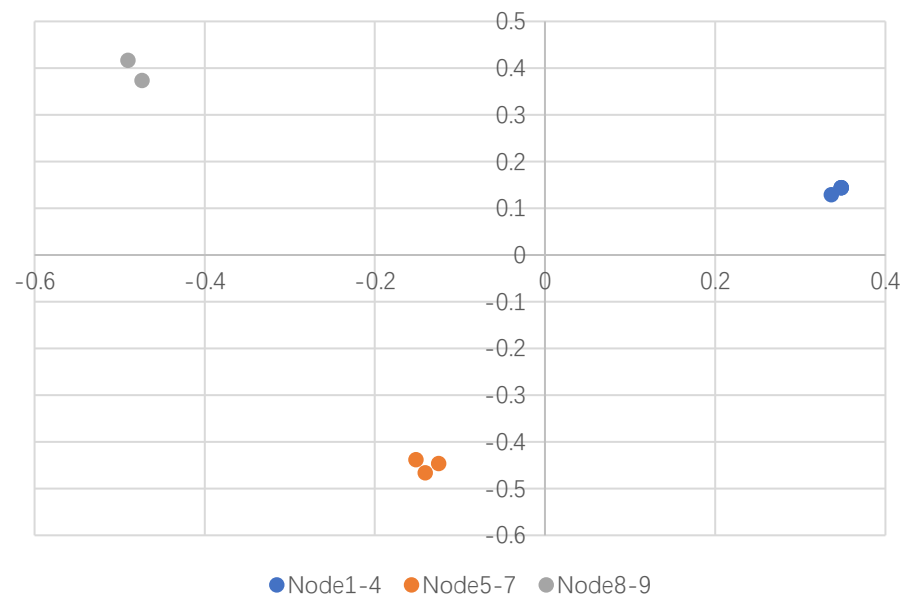
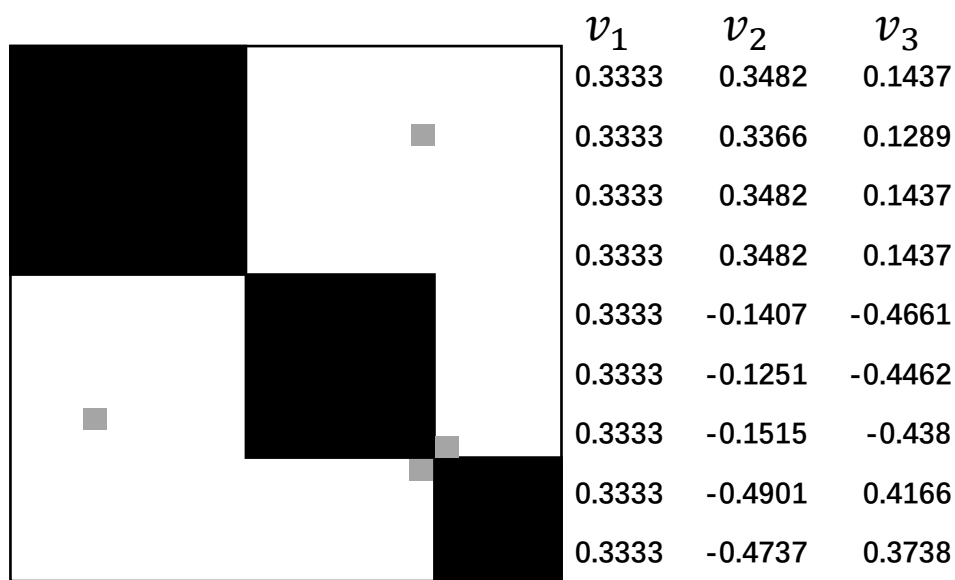
- 如果有图上有 $k$ 个连通分量 ( $L$ 矩阵为块对角形式),  $L$ 有 $k$ 个简并的最小特征向量 (对应特征值=0)。取这 $k$ 个特征向量为 $[11\cdots 11, 00\cdots 00, \cdots, 00\cdots 00; 00\cdots 00, 11\cdots 11, \cdots, 00\cdots 00; 00\cdots 00, \cdots, 11\cdots 11]$
- 将这 $k$ 个特征向量拼成一个 $n \times k$ 的矩阵, 每一行对应一个 $k$ 维的向量。
- 属于同一连通分量的节点向量是一样的。

	$v_1$	$v_2$	$v_3$
	1	0	0
	1	0	0
	1	0	0
	1	0	0
	0	1	0
	0	1	0
	0	1	0
	0	0	1
	0	0	1



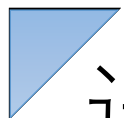
# 谱聚类

- 如果Laplacian矩阵并不是严格块对角的形式。在连通分量之间添加少量边作为扰动。最小特征值对应的k个特征向量解除简并。
- 但是前2到k个特征值接近0，同样将节点投影到前k个特征向量的空间中，属于同一个类的点在空间中接近。



特征值: 0, 0.0334, 0.1027, 2.0487, 3.0339, 3.1, 4, 4, 4.0812





# 谱聚类 and 图分割

- 最小化Ratio cut等价于:

$$\min_{A \subset V} f' L f$$

$$\text{subject to } f \perp \mathbf{1}, \quad f_i \text{ defined as in (2), } \|f\| = \sqrt{n}.$$

- 取消标记向量f的离散条件:

$$\min_{f \in \mathbb{R}^n} f' L f \quad \text{subject to } f \perp \mathbf{1}, \quad \|f\| = \sqrt{n}.$$

- 易得:  $f$ 取 $L$ 的第二小特征值对应的特征向量。对应的分类: 
$$\begin{cases} v_i \in A & \text{if } f_i \geq 0, \\ v_i \in \bar{A} & \text{if } f_i < 0. \end{cases}$$

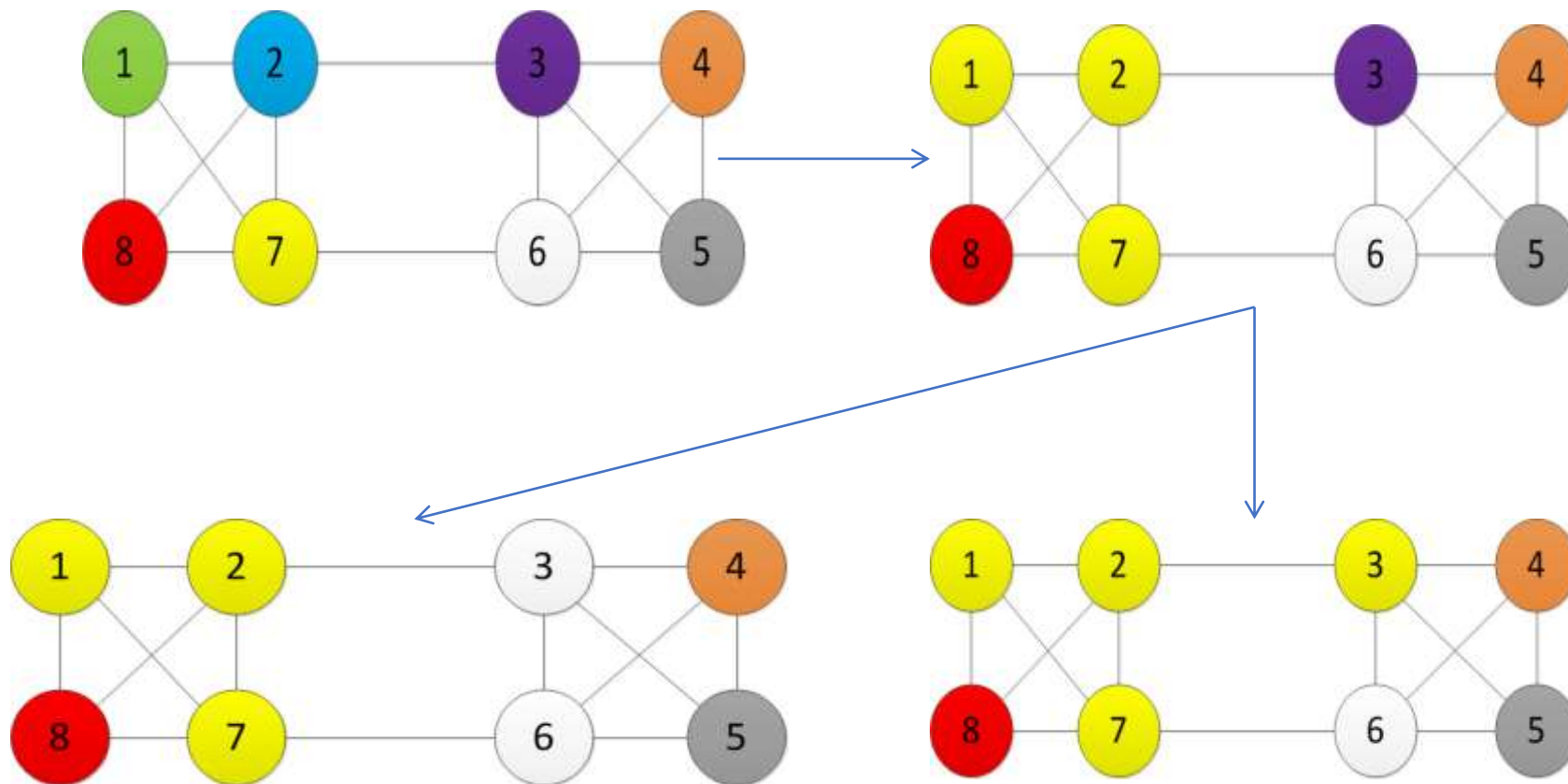


# 标签传播算法与社区发现

- 无监督/半监督的分类方法。
- 连接到同类邻居的边 > 连接到不同类邻居的边
- 每个节点所属的类由邻居节点所属的类决定。
- 无监督的标签传播：不需要指定类的个数和类的大小。
- 半监督的标签传播：可以利用少量已经标注了类标签的数据点。聚类的数目和标注的类的数目保持一致。
- 优点：时间复杂度  $O(knd)$ （ $d$  是平均度数， $k$  是迭代次数）。近乎线性时间。



# 标签传播算法



# 标签传播算法

- LPA<sup>[1]</sup>
  - 1. 给每个节点一个唯一的标签
  - 2. 对于每个节点, 选择它邻居中出现最多的标签
  - 3. 重复第二步直到所有节点的标签不再变化
- 同步传播 & 异步传播
- 强社区(社区内每个节点连向社区内部的点比连向社区外部的点多)

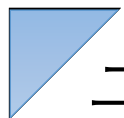


# 异步算法流程

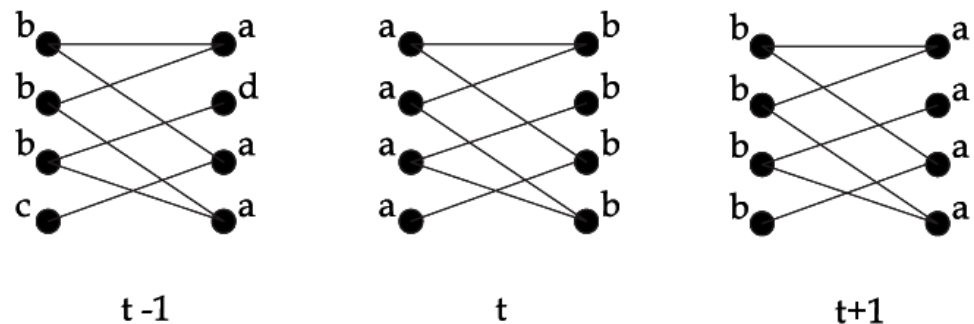
- 1) 初始化网络中所有节点的标签, 对于给定节点 $x$ ,  $C_x(0)=x$
- 2) 设置  $t=1$
- 3) 以随机顺序排列网络中的节点, 并按排列序号将其设置为 $x$
- 4) 对于特定顺序选择的每个 $x \in X$ , 让 $C_x(t)=f(C_{x1}(t), \dots, C_{xim}(t), \dots)$ 。返回相邻标签中出现频率最高的标签。如果有多个最高频率的标签, 就随机选择一个标签返回
- 5) 如果每个节点都有其邻居节点中数量最多的标签, 没有更新, 则停止算法, 否则, 设置 $t=t+1$ 并转到3

(计算过程结果不唯一且不保证收敛)





# 无监督标签传播



- 上述的标签传播算法的两个缺陷：
  1. 在类似二部图的结构中，会发生标签的振荡而无法收敛。
  2. 可能出现某个非常庞大的社区。
- 同步更新(synchronous): 用 $t$ 时刻的标签更新 $t+1$ 时刻的标签。
- 异步更新(asynchronous): 用已经更新的 $t+1$ 时刻的标签和未更新的 $t$ 时刻标签，更新 $t+1$ 时刻当前节点的标签。
- 异步更新收敛更快，且更不容易陷入振荡。同步更新结果更稳定。





# 无监督标签传播

- 带有节点偏好和hop衰减的标签传播：

$$\mathcal{L}'_i = \operatorname{argmax}_{\mathcal{L}} \sum_{i' \in \mathcal{N}_i} s_{i'}(\mathcal{L}_{i'}) f(i')^m w_{i',i},$$

- $s_i(\mathcal{L})$ 是标签 $\mathcal{L}$ 在节点 $i$ 处的hop score,  $f(i)$ 是任意可比的节点 $i$ 的特征（节点偏好）。例如，可以取 $f(i) = \text{Deg}(i)$ 。参数 $m$ 控制特征对于标签传播的影响。
- 标签的hop score会随着标签的传播衰减：

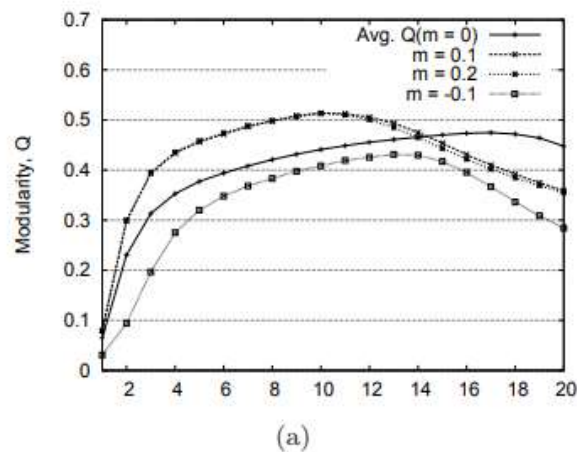
$$s'_i(\mathcal{L}'_i) = \left[ \max_{i' \in \mathcal{N}_i(\mathcal{L}'_i)} s_{i'}(\mathcal{L}_{i'}) \right] - \delta,$$

- $\mathcal{N}_i(\mathcal{L})$ 是 $i$ 的标签为 $\mathcal{L}$ 的邻居。 $\delta$ 控制标签的hop score的衰减速度。当被选择的标签等于现在的标签时,  $\delta = 0$ 。（避免负反馈循环）

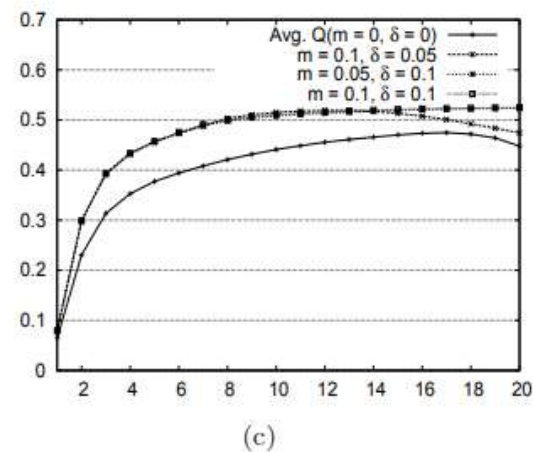
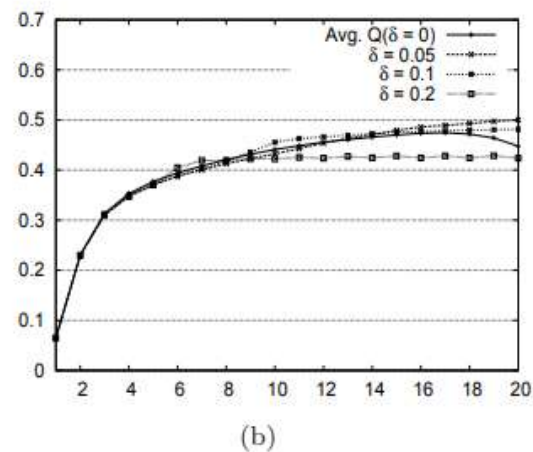


# 无监督标签传播

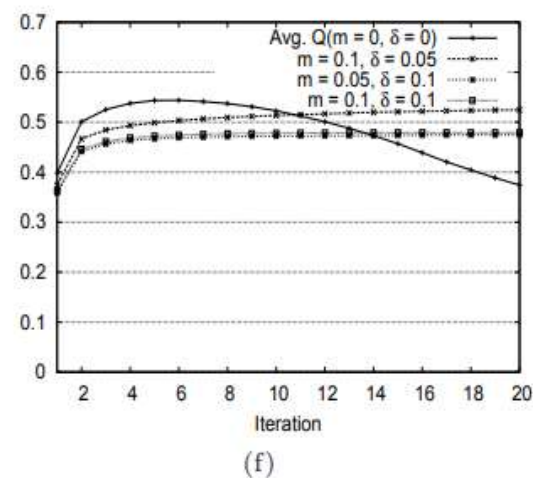
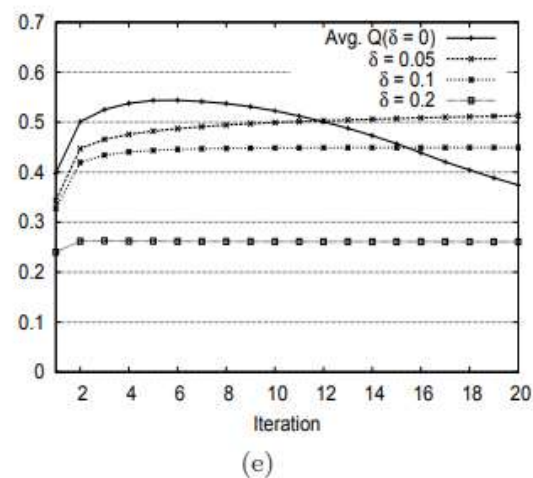
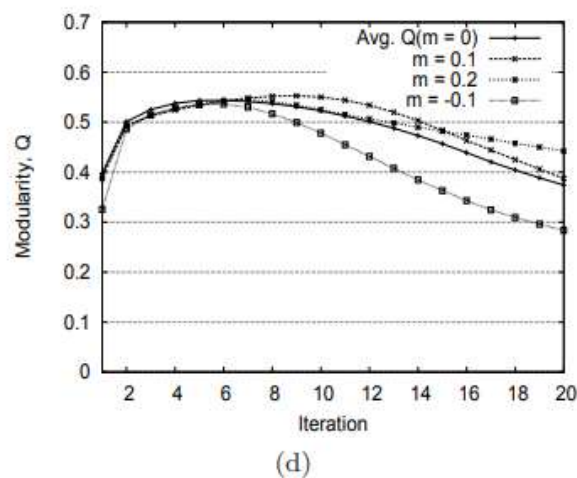
- 节点偏好度的引入能够更快到达模块度的峰值，但下落也更快。
- 选择合适的节点偏好能提高模块度的峰值。
- 跳衰减的引入能够避免出现庞大社区，避免出现传播一定步数后模块度的下降。但是衰减太快社区也无法正常增长，限制模块度的升高。



Synchronous:



Asynchronous:



# 半监督标签传播

- 假设某个点的标签能够以一定概率传播给与它相连的点。
- 概率转移矩阵

$$T_{ij} = P(i \rightarrow j) = \frac{w_{ij}}{\sum_k w_{ik}}$$

- t时刻标签矩阵:  $Y^t \in N^{n \times C}$

$$Y_{ij}^0 = \begin{cases} 1, & x_i \in c_j \\ 0, & otherwise \end{cases} \text{ 对 } L \text{ 个标注数据}$$

对未标注数据,  $Y_{ij}^0$  可以取任意值。

- 标签的传播:  $Y_{ij}^{t+1} = \sum_k T_{ki} Y_{kj}^t$



# 标签传播

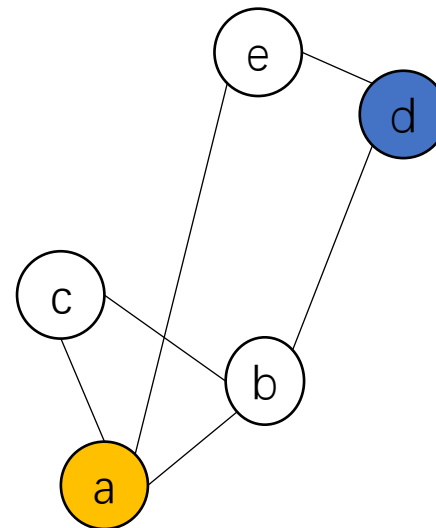
- 1: 按如上所述方式计算概率转移矩阵 $T$ 。t=0时，按如上所述初始化标签矩阵 $Y$ 。
- 2: 标签传播:  $Y^{t+1} = T^T Y^t$
- 3: 对于已标注数据，恢复它们的标注标签。对于未标注数据，对行归一化。
- 4: 重复2-3步，直到 $Y^t$ 收敛。
- 5: 最终节点 $x_i$ 的标签 $c_j = \operatorname{argmax}_j Y_{ij}$



# 标签传播

- 已标数据: [a: 1, d: 0]
- 未标数据: [c: ?, b: ?, e: ?]
- 标签转移矩阵T:

	a	b	c	d	e
a	0.00	0.50	0.40	0.00	0.10
b	0.50	0.00	0.40	0.10	0.00
c	0.50	0.50	0.00	0.00	0.00
d	0.00	0.20	0.00	0.00	0.80
e	0.20	0.00	0.00	0.80	0.00



初始化 $Y^0$

	0	1
a	0	1
b	1	0
c	1	0
d	1	0
e	1	0

# 标签传播

- 最基本的标签传播算法：

$$z_i = \arg \max_z \sum_{j \neq i} W_{ij} \delta_{z_j z},$$

- 标签传播算法的目标函数可以写成：

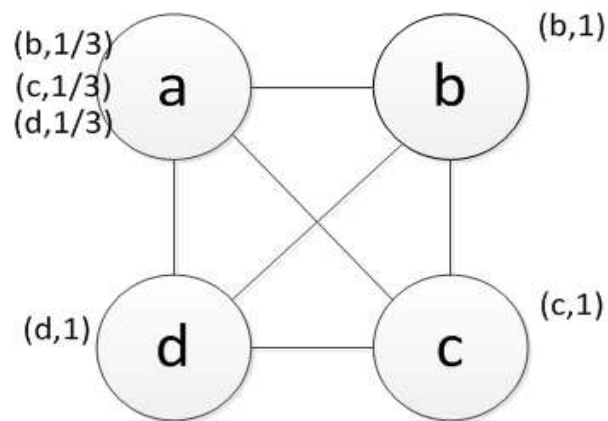
$$Q_{\text{LPA}} = \sum_{i,j} W_{ij} \delta_{z_i z_j} = \sum_z \sum_{i,j} W_{ij} \delta_{z_i z} \delta_{z_j z}$$

- 最大化类内权重和。等价于最小化cut size。



# 多标签传播算法

- COPRA<sup>[2]</sup>
- 节点的标签为一个向量(多标签)
- 标签形式:  $((c_1, b_1), (c_2, b_2), \dots, (c_v, b_v))$



$$b_t(c, x) = \frac{\sum_{y \in N(x)} b_{t-1}(c, y)}{|N(x)|},$$

- 传递更多信息
- 重叠社区发现



# 总结

- 图分割：
  - 最小化 Cut / Ratio Cut / Normalization cut
- 谱聚类：
  - 求矩阵特征值/特征向量的聚类方法
  - 是一种求最小分割的近似方法
- 标签传播
  - 类标签沿着边在图上传播。
  - 目标和求最小割是一致的。








# 文本集中词的向量空间表示 (word embedding)

- 通过低维度（一般几百维 $\ll$ 词汇数），信息密集（基本不出现0）的向量表示词汇的语义信息
- 例如：
- 北京大学：[1.1, 4.5, -0.76, ...] ( $len=300$ )
- 方法：Word2Vec(**CBOW**, Skip-gram)





# 文本词向量模型 (word2vec)

- 本质：上下文词PMI向量的特征空间降维表达 (shifted positive PMI)
- 

## Neural Word Embedding as Implicit Matrix Factorization

**Omer Levy**

Department of Computer Science  
Bar-Ilan University  
omerlevy@gmail.com

**Yoav Goldberg**

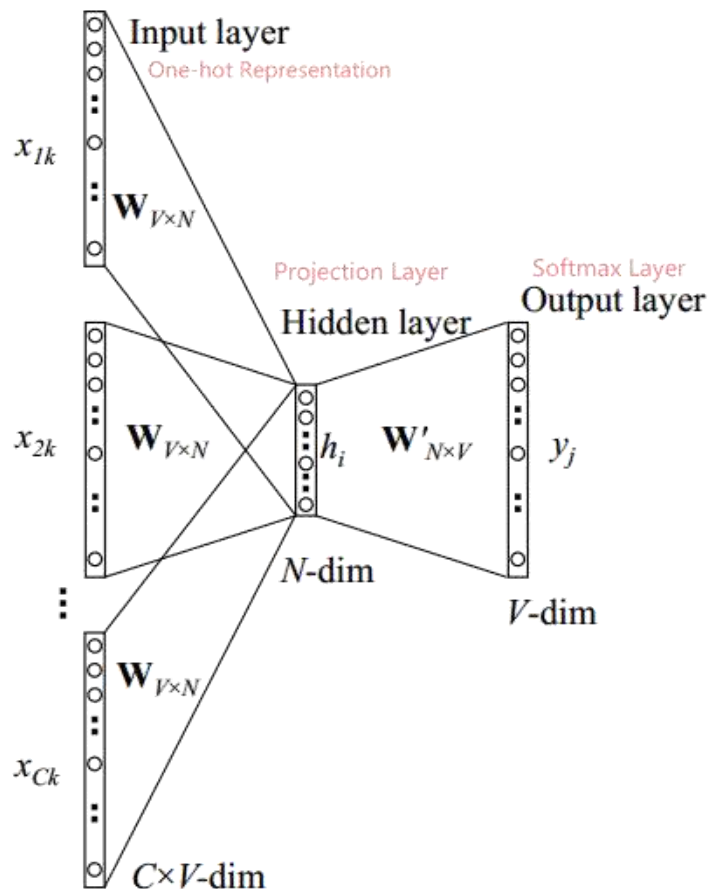
Department of Computer Science  
Bar-Ilan University  
yoav.goldberg@gmail.com

### Abstract

We analyze skip-gram with negative-sampling (SGNS), a word embedding method introduced by Mikolov et al., and show that it is implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global constant. We find that another embedding method, NCE, is implicitly factorizing a similar matrix, where each cell is the (shifted) log conditional probability of a word given its context. We show that using a sparse *Shifted Positive PMI* word-context matrix to represent words improves results on two word similarity tasks and one of two analogy tasks.



# 词向量表示（之一）：CBOW模型



- CBOW通过上下文预测中心词概率
- 词汇总数 $V$ ，目标词向量维度 $N$ ，上下文参考词个数 $C$
- $x_i$  第 $i$ 个词的one-hot向量，通过词向量矩阵 $W$ 得到 $v_i$ ，计算平均值：

$$h_t = \frac{1}{C} * W^T \sum_{i=1}^C x_i = \frac{1}{C} * \sum_{i=1}^C v_i$$

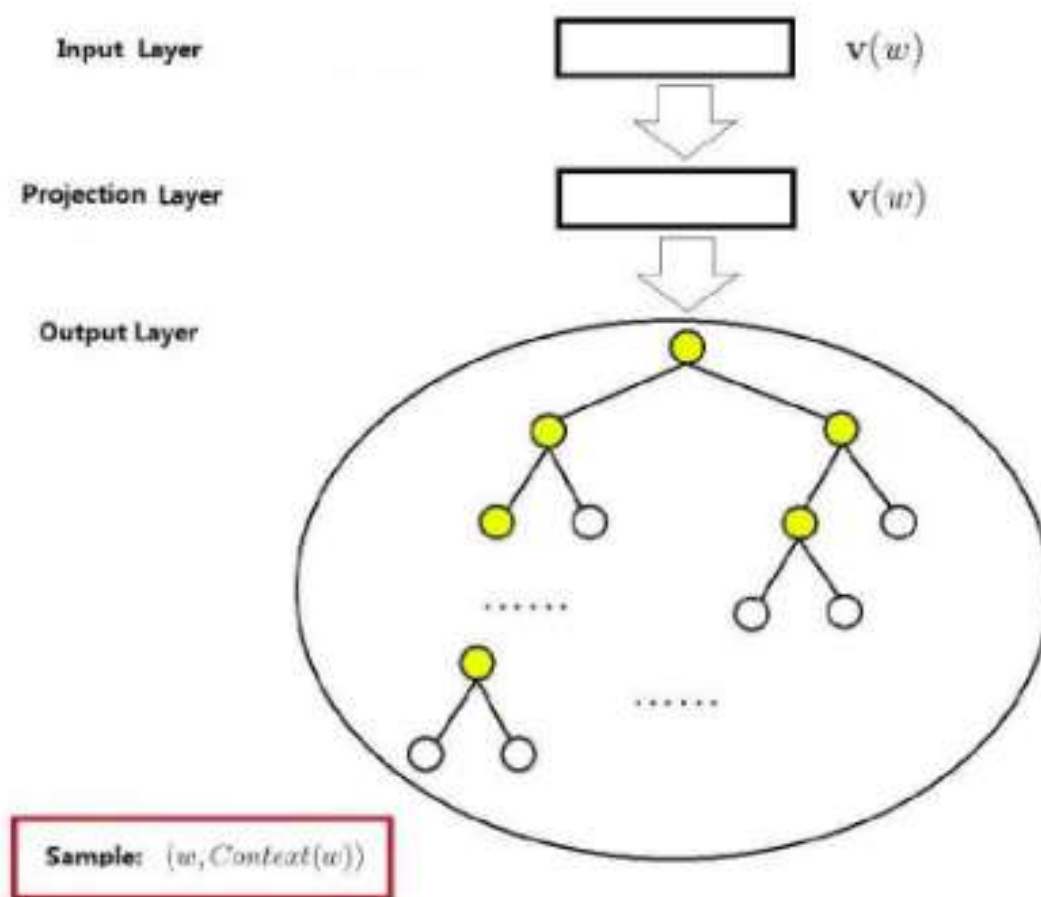
- 最后映射回原本词表维度，softmax计算预测词语概率

Figure 2: Continuous bag-of-words model



# Skip-gram模型

基于 Negative Sampling





# 文本的向量表示

## ——词向量-训练与使用

- Gensim 库:

```
>>> from gensim.models import Word2Vec
>>> sentences = [["cat", "say", "meow"], ["dog", "say", "woof"]]
>>> model = Word2Vec(sentences, min_count=1)
>>> vector = model.wv["cat"]
```

- 参考网站: <https://radimrehurek.com/gensim/models/word2vec.html>



# 使用预训练的wordembedding模型

中文预训练词向量: <https://github.com/Embedding/Chinese-Word-Vectors>

这里用了在微博语料上训练得到的300维词向量 `sgns.weibo.word` , 每行数据代表一个词的词向量:  
第一个是词, 后三百个浮点数表示该词的词向量。

加载预训练词向量:

```
] : 1 word_embeddings = {} # 词条: 向量 (词典)
    2 with open('sgns.weibo.word', encoding='utf-8') as f: # 第一行词表大小 维度。后面 词
    3     info = next(f)
    4     print(info)
    5
    6     for line in f:
    7         values = line.split()
    8         word = values[0]
    9         embedding = np.asarray(values[1:], dtype='float32')
   10         word_embeddings[word] = embedding
   11
   12 word_embeddings['前进']
```

195202 300

195202 300

```
Out[5]: array([-0.063822,  0.184022,  0.100999, -0.019417,  0.095195,  0.206944,
                -0.013827, -0.037561,  0.472569,  0.25951 ,  0.303721, -0.298486,
                -0.016543, -0.470145, -0.181971, -0.225016, -0.081724,  0.403648,
                 0.751034,  0.352738, -0.145962, -0.625916,  0.077062, -0.032005,
                -0.032523,  0.347509, -0.15896 ,  0.285232,  0.255275,  0.04022 ,
                -0.125468,  0.109054, -0.38585 ,  0.037837, -0.021359,  0.30199 ,
                -0.183625,  0.252392, -0.329855, -0.196766,  0.522931, -0.401678,
                 0.378263, -0.336475,  0.133984, -0.433968,  0.1378 , -0.185041,
                 0.310316, -0.189576, -0.135958, -0.153496, -0.00555 ,  0.447206,
                -0.037054,  0.06028 ,  0.486924,  0.0709 , -0.159956,  0.121423,
                0.226428, -0.356725,  0.411512,  0.011405, -0.292778, -0.380954,
                 0.395782,  0.120386,  0.243591,  0.64321 , -0.260389,  0.181542,
                -0.000956,  0.086108, -0.006262, -0.030874,  0.393065,  0.430639,
                 0.206671, -0.491738, -0.33875 ,  0.095566, -0.16227 ,  0.240371,
                -0.415184,  0.156632, -0.20279 , -0.212032, -0.363044, -0.182482,
                -0.145277,  0.539901,  0.259577, -0.631535,  0.254142, -0.269145,
                -0.07179 ,  0.067686,  0.160444, -0.594299, -0.206516,  0.411072,
                -0.105586,  0.058507, -0.116741,  0.210935,  0.155957, -0.271259,
```



上下文分布的相似度经常不能很好的区分态度-立场的表述。可以考虑单独加入情感极性或情感分类维度做进一步的区分

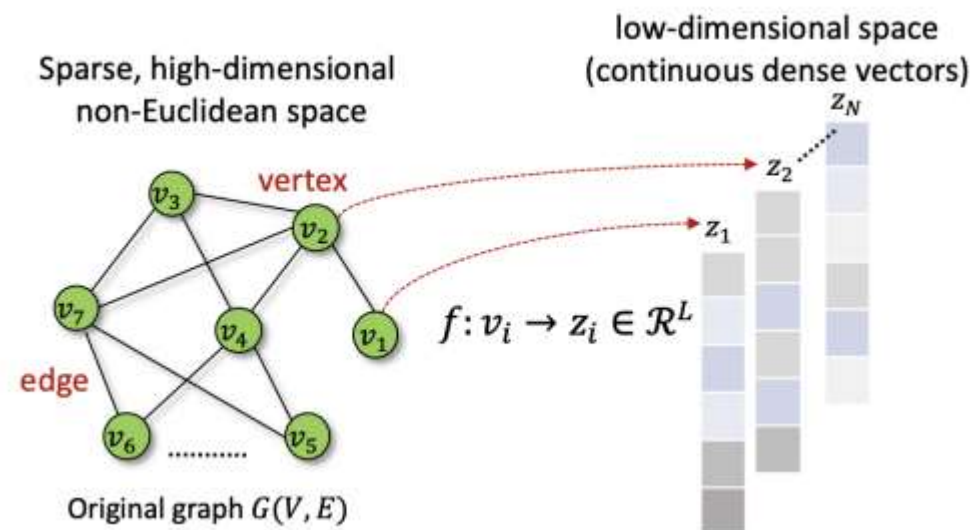
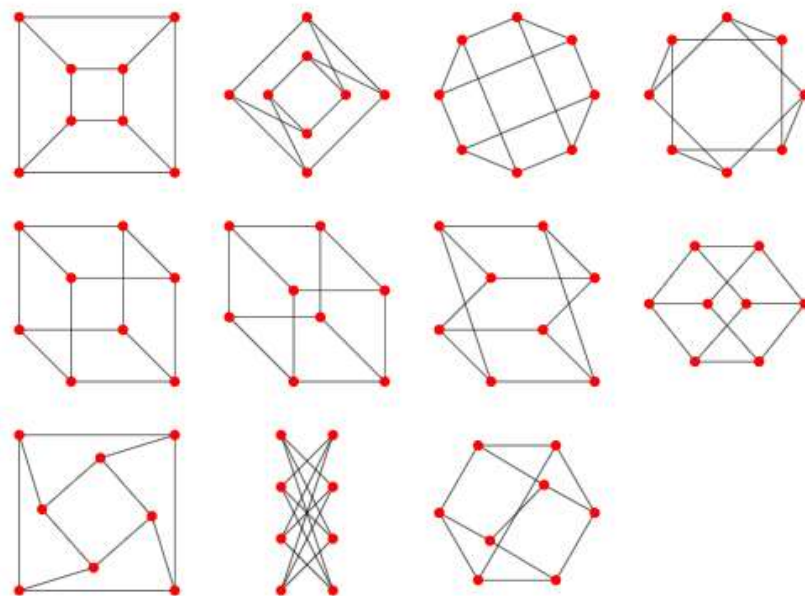
```
In [6]: 1 from sklearn.metrics.pairwise import cosine_similarity # 向量余弦相似度计算
        2
        3 print(cosine_similarity(word_embeddings['赞成'].reshape(1, -1), word_embeddings['支持'].reshape(1, -1)))
        4 print(cosine_similarity(word_embeddings['赞成'].reshape(1, -1), word_embeddings['反对'].reshape(1, -1)))
        5 print(cosine_similarity(word_embeddings['赞成'].reshape(1, -1), word_embeddings['同意'].reshape(1, -1)))
        6 print(cosine_similarity(word_embeddings['赞成'].reshape(1, -1), word_embeddings['数目'].reshape(1, -1)))

[[0.40121433]]
[[0.5830749]]
[[0.59187734]]
[[0.17137685]]
```

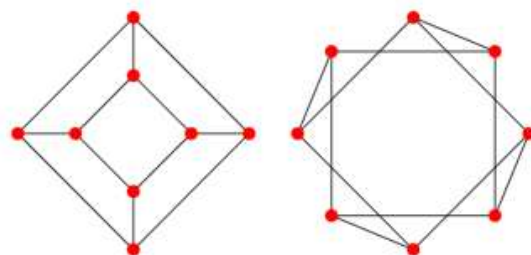


## 图嵌入算法

Download  
Wolfram Notebook



A graph embedding, sometimes also called a graph drawing, is a particular drawing of a [graph](#). Graph embeddings are most commonly drawn in the plane, but may also be constructed in three or more dimensions. The above figure shows several embeddings of the [cubical graph](#). The most commonly encountered graph embeddings are generally [straight line embeddings](#), in which all edges are drawn as straight line segments.





# PMI (jaccard) 关联的PCA近邻嵌入

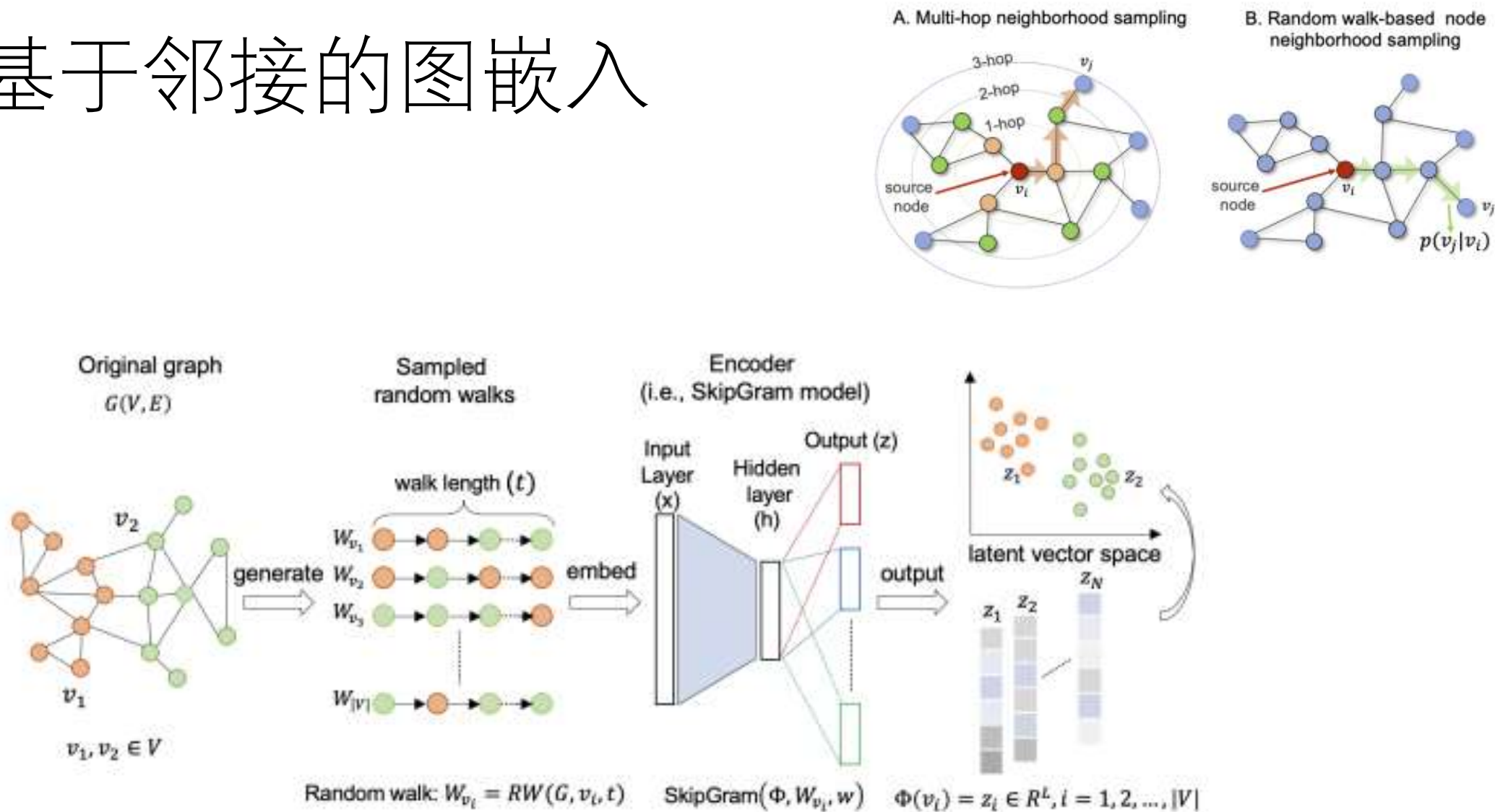
```
7]: 1 #using PMI to calculate the similarity matrix
    2 def similarity_matrix(rating_matrix):
    3     N = len(rating_matrix)
    4     S = np.zeros((N,N))
    5     for i in range(N):
    6         if (i % 100 == 99): # 算前100测试一下, 这个算法太慢
    7             break
    8         for j in range(i,N):
    9             S[i,j] = pmi(i, j, rating_matrix) # i-j pmi
   10             S[j,i] = S[i,j]
   11     return S
   12
   13 # user_similarity = similarity_matrix(train_data)
   14
   15 data_mat_freq = np.where(data_matrix>0, 1, 0) # 评分 ==> 频率
   16 user_similarity_test = similarity_matrix(data_mat_freq)
```

```
1]: 1 a = data_mat_freq @ data_mat_freq.T # 20分钟
    2 np.save("data/movie_coRating_count", a) #保存数组
    3 a[0]
```

```
1]: array([53,  7,  6, ...,  0, 15, 17])
```

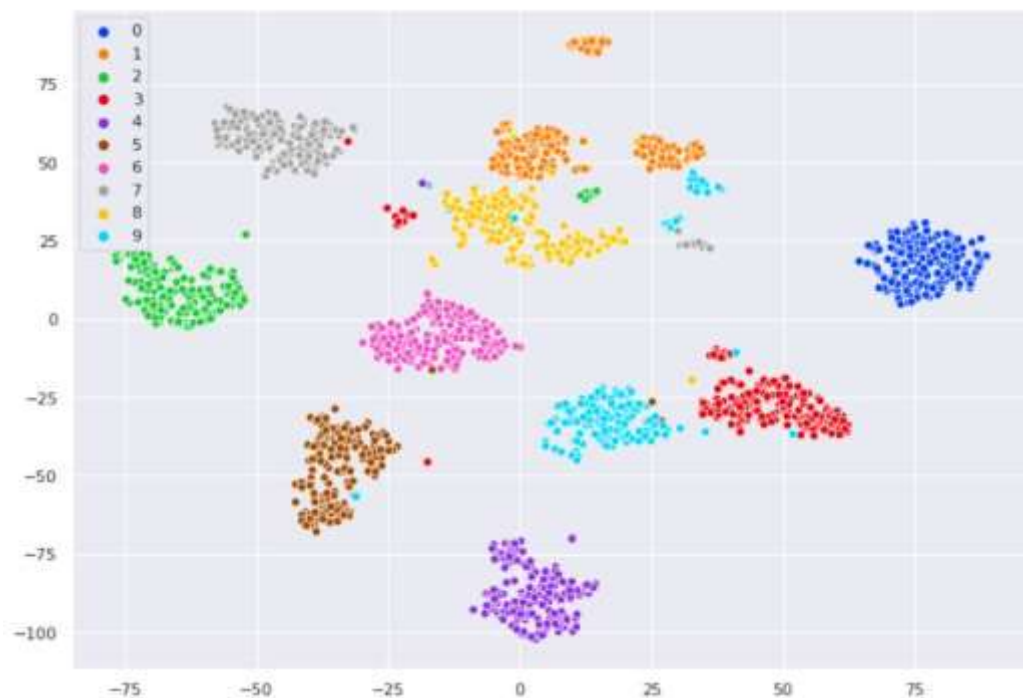


# 基于邻接的图嵌入



# 图的空间嵌入 (embedding) :

- 关联作为特征，目标是空间距离尽量反应关联特征 (t-SNE)



## sklearn.manifold.TSNE

```
class sklearn.manifold. TSNE(n_components=2, *, perplexity=30.0, early_exaggeration=12.0, learning_rate='auto', n_iter=1000, n_iter_without_progress=300, min_grad_norm=1e-07, metric='euclidean', metric_params=None, init='pca', verbose=0, random_state=None, method='barnes_hut', angle=0.5, n_jobs=None)
```

[\[source\]](#)

T-distributed Stochastic Neighbor Embedding.

t-SNE [1] is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.

It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples. For more tips see Laurens van der Maaten's FAQ [2].



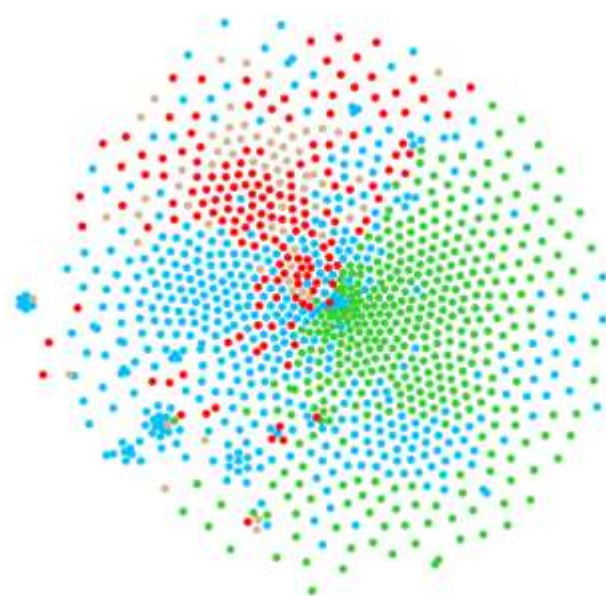
# 基于T-SNE算法实现评论分布可视化

加分：更加合理的可视化方案

```
1 vectorizer = CountVectorizer()
2 X = vectorizer.fit_transform(comments) #获取词袋中所有文本关键词
3 word = vectorizer.get_feature_names()
4
5 transformer = TfidfTransformer()
6 tfidf = transformer.fit_transform(X) #将词频矩阵X统计成TF-IDF值 #查看数据结构 tfidf[i][j]表示i类文本中的tf-idf权重
7 weight = tfidf.toarray()
```

```
1 #散点图数据准备, TSNE降维数, weight为tf-idf矩阵
2 tsne = TSNE(n_components=2)
3 decomposition_data = tsne.fit_transform(weight)
```

```
1 fig = plt.figure(figsize=(10, 10))
2 ax = plt.axes()
3 ax.axis('off')
4 labels = ["hegang", "huxijin", "shower", "war"]
5 colors = ['limegreen', 'deepskyblue', 'red', 'tan']
6 for i in range(decomposition_data.shape[0]):
7     plt.scatter(decomposition_data[i,0], decomposition_data[i,1], c=colors[classes[i]-1])
```



# 电影推荐系统（问题分析）：

目标：

- 推荐给已有用户更多的候选电影
  - 推荐给新用户可选的电影

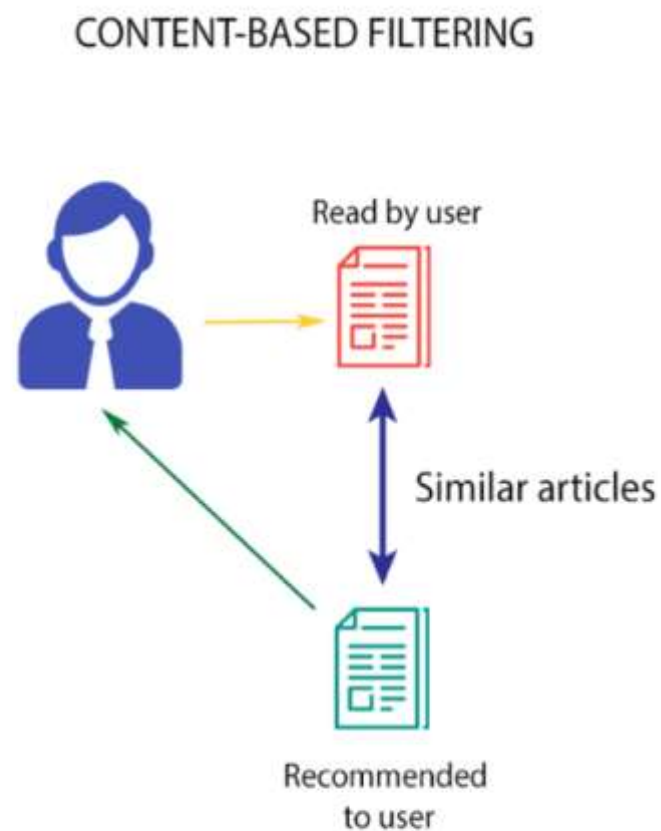
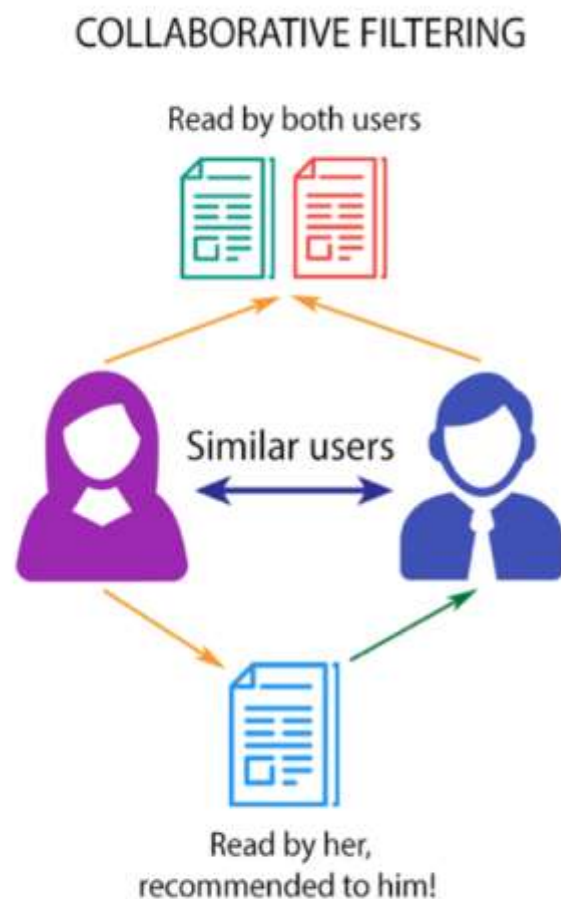
条件：

- 已知所有用户历史观影及评分
- 已知用户部年龄、职业、性别信息
- 已知电影名称、类属信息





# 关联挖掘与推荐模型:



# 机器学习——各种距离度量方法

参考: <https://blog.csdn.net/qs17809259715/article/details/96110056>

- 
- 
- 一、欧氏距离(EuclideanDistance)
  - 二、曼哈顿距离(ManhattanDistance)
  - 三、切比雪夫距离 (Chebyshev Distance)
  - 四、闵可夫斯基距离(Minkowski Distance)
  - 五、标准化欧式距离(Standardized Euclidean Distance)
  - 六、马氏距离(Mahalanobis distance)
  - 七、余弦距离(Cosine Distance)
  - 八、汉明距离(Hamming Distance)
  - 九、杰卡德距离(Jaccard Distance)
  - 十、相关距离(Correlation distance)
  - 十一、信息熵(Information Entropy)





## 九、杰卡德距离(Jaccard Distance)

### 1.定义

- 杰卡德相似系数(Jaccard similarity coefficient): 两个集合A和B的交集元素在A, B的并集中所占的比例, 称为两个集合的杰卡德相似系数, 用符号  $J(A,B)$  表示:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

杰卡德相似系数

- 杰卡德距离(Jaccard Distance): 与杰卡德相似系数相反, 用两个集合中不同元素占所有元素的比例来衡量两个集合的区分度:

$$J_{\delta}(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

杰卡德距离



信息相关度： 互信息-点互信息

$$I(X; Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

$$\text{pmi}(x; y) \equiv \log \frac{p(x, y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)}.$$



# 作业环境中给出的一个实现：

## 生成User-Movies二维rating矩阵

- 2. 生成User-Movies二维rating矩阵

```
1 # 填写数据矩阵
2 train_data_matrix = np.array(train_data.pivot_table('rating', index='user_id', columns='movie_id', aggfunc='mean').fillna(0))
3
4 # 1. todou 填写测试集矩阵
5 test_data_matrix = np.array(test_data.pivot_table('rating', index='user_id', columns='movie_id', aggfunc='mean').fillna(0))
6
7 train_data_matrix[0:3,:] #每行为一个用户的观影评分向量
```

```
array([[5., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```



# 相似度计算:

计算 user-user 相似度矩阵 和 item-item 相似度矩阵

```
: 1 from sklearn.metrics.pairwise import pairwise_distances
  2 user_similarity = pairwise_distances(train_data_matrix, metric='cosine') # 用户
  3
  4 # 2, todo: 生成电影相似度矩阵
  5
  6 user_similarity[0]
```

array([0. , 0.96598031, 0.94814587, ..., 1. , 0.80161937,  
 0.87312904])

向量空间模型的前提?



## 通过相似度矩阵进行预测

```
1 def predict(train_ratings, similarity, type='user'):  
2     if type == 'user':  
3         mean_user_rating = train_ratings.mean(axis=1) # 平均分  
4         ratings_diff = (train_ratings - mean_user_rating[:, np.newaxis]) # 评分向量中心化,  
5         # 通过相似度矩阵生成推荐, 然后再把中心偏置加回来  
6         pred = mean_user_rating[:, np.newaxis] + \  
7             similarity.dot(ratings_diff) / np.array([np.abs(similarity).sum(axis=1)]).T # user_sim  
8  
9     elif type == 'item':  
10        pred = train_ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)]) # item_sim  
11        # 3、todo:这里尝试对电影的属性做一下中心化, 观察结果数据有无改善。能否尝试分析原因  
12  
13    return pred  
14  
15 item_prediction = predict(train_data_matrix, item_similarity, type='item')  
16 user_prediction = predict(train_data_matrix, user_similarity, type='user')  
17  
18 user_prediction.shape
```

(6040, 3660)



$$RMSE = \sqrt{\frac{1}{N} \sum (x_i - \hat{x}_i)^2}$$

I'll use the scikit-learn's **mean squared error** function as my validation metric. Comparing user- and item-based collaborative filtering, it looks collaborative filtering gives a better result.

## 评估预测效果

```

1 from sklearn.metrics import mean_squared_error
2 from math import sqrt
3
4 def rmse(prediction, ground_truth):
5     prediction = prediction[ground_truth.nonzero()].flatten() # 推荐结果 与 ground truth 比较
6     ground_truth = ground_truth[ground_truth.nonzero()].flatten()
7     errorR = sqrt(mean_squared_error(prediction, ground_truth))
8
9     return errorR
10
11 print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
12 print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
13
14 # 4, todo: 目前这种评测方案有没有问题? 能否设计一种更好的评价方案 (简单说明思路 and 道理+实现)。选做题。
15 # 能完成的同学可以在作业压缩包学号后面添加一个#号以提醒助教优先评阅

```

- RMSE对极端值敏感, 更换 MAE (平均绝对误差)
- F1-score评价推荐情况

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

1900012973 王乐安

2000013157 刘知一

User-based CF RMSE: 3.6224523289261183

Item-based CF RMSE: 3.6385367743842383

# Junk-pop movies:

```
1 pop_junk_movies = ratings_movie_stats[(ratings_movie_stats['rating_average'] < 3) &
2 pop_junk_movies
```

movie_id	rating_average	user_count
788	2.995781	948
1377	2.976722	1031
1391	2.900372	1074
2054	2.933014	1045
2369	2.954762	840
2701	2.158537	902
3354	2.595208	793

# Golden-pop movies:

```
1 pop_golden_moives = ratings_movie_statis[(ratings_movie_statis['rating_average'] > 4.45) & (ratings
2 pop_golden_moives
```

**rating\_average** **user\_count**

**movie\_id**

50	4.517106	1783
----	----------	------

260	4.453694	2991
-----	----------	------

318	4.554558	2227
-----	----------	------

527	4.510417	2304
-----	----------	------

858	4.524966	2223
-----	----------	------

904	4.476190	1050
-----	----------	------





# 看一眼是那些电影：Join操作不work

```
1 junk_gener_distri = pop_junk_moives.join(movies, on = 'movie_id', how = 'left' ) # index
2 junk_gener_distri
```

	rating_average	user_count	movie_id	title	genres
movie_id					
788	2.995781	948	798	Daylight (1996)	Action Adventure Thriller
1377	2.976722	1031	1398	In Love and War (1996)	Romance War
1391	2.900372	1074	1414	Mother (1996)	Comedy
2054	2.933014	1045	2123	All Dogs Go to Heaven (1989)	Animation Children's
2369	2.954762	840	2438	Outside Ozona (1998)	Drama Thriller
2701	2.158537	902	2770	Bowfinger (1999)	Comedy
3354	2.595208	793	3423	School Daze (1988)	Drama

```

1 junk_gener_distri = pop_junk_moives.merge(movies, left_on = 'movie_id', right_on = 'movie_id')
2 junk_gener_distri

```

	movie_id	rating_average	user_count	title	genres
0	788	2.995781	948	Nutty Professor, The (1996)	Comedy Fantasy Romance Sci-Fi
1	1377	2.976722	1031	Batman Returns (1992)	Action Adventure Comedy Crime
2	1391	2.900372	1074	Mars Attacks! (1996)	Action Comedy Sci-Fi War
3	2054	2.933014	1045	Honey, I Shrunk the Kids (1989)	Adventure Children's Comedy Fantasy Sci-Fi
4	2369	2.954762	840	Desperately Seeking Susan (1985)	Comedy Romance
5	2701	2.158537	902	Wild Wild West (1999)	Action Sci-Fi Western
6	3354	2.595208	793	Mission to Mars (2000)	Sci-Fi

```

1 golden_gener_distri = pop_golden_moives.merge(movies, left_on = 'movie_id', right_on = 'movie_id')
2 golden_gener_distri

```

	movie_id	rating_average	user_count	title	genres
0	50	4.517106	1783	Usual Suspects, The (1995)	Crime Thriller
1	260	4.453694	2991	Star Wars: Episode IV - A New Hope (1977)	Action Adventure Fantasy Sci-Fi
2	318	4.551558	3337	Shrek, The (2001)	Comedy Fantasy

:	1	users	用户信息表
---	---	-------	-------

:

	user_id	gender	zipcode	age_desc	occ_desc
0	1	F	48067	Under 18	K-12 student
1	2	M	70072	56+	self-employed
2	3	M	55117	25-34	scientist
3	4	M	02460	45-49	executive/managerial
4	5	M	55455	25-34	writer
...	...	...	...	...	...
6035	6036	F	32603	25-34	scientist
6036	6037	F	76006	45-49	academic/educator
6037	6038	F	14706	56+	academic/educator
6038	6039	F	01060	45-49	other or not specified
6039	6040	M	11106	25-34	doctor/health care

6040 rows × 5 columns



```
In [118]: 1 pop_junk_moives2
```

```
Out[118]:
```

	rating_average	user_count
movie_id		
95	2.876176	638
153	2.642214	777
160	2.238938	565
173	2.308511	564
185	2.869947	569
196	2.823636	550
208	2.631336	651
435	2.606004	533
442	2.992188	640
466	2.795812	573

```
(ratings_movie_statis['user_count']>500)
```



```
: 1 junk_moive_users = ratings.merge(pop_junk_moives2, left_on='movie_id', right_on='movie_id',
2 junk_moive_users
```

:

	user_id	movie_id	rating	timestamp	rating_average	user_count
0	2	3257	3	978300073	2.859425	626
1	8	3257	3	978247143	2.859425	626
2	18	3257	2	978153771	2.859425	626
3	22	3257	1	978136958	2.859425	626
4	26	3257	4	978139867	2.859425	626
...	...	...	...	...	...	...
25007	5636	3697	3	959053960	2.867807	643
25008	5682	3697	4	959043421	2.867807	643
25009	5714	3697	3	959223406	2.867807	643
25010	5767	3697	3	959620056	2.867807	643
25011	5831	3697	2	1024075950	2.867807	643

获得movie\_ID





# 链接用户信息

```
1 junk_moive_users_distri = junk_moive_users.merge(users, left_on ='user_id', right_on = 'user_id', how = 'left')
2 junk_moive_users_distri
```

	user_id	movie_id	rating	timestamp	rating_average	user_count	gender	zipcode	age_desc	occ_desc
0	2	3257	3	978300073	2.859425	626	M	70072	56+	self-employed
1	8	3257	3	978247143	2.859425	626	M	11413	25-34	programmer
2	18	3257	2	978153771	2.859425	626	F	95825	18-24	clerical/admin
3	22	3257	1	978136958	2.859425	626	M	53706	18-24	scientist
4	26	3257	4	978139867	2.859425	626	M	23112	25-34	executive/managerial
...	...	...	...	...	...	...	...	...	...	...
25007	5636	3697	3	959053960	2.867807	643	M	98102	25-34	executive/managerial
25008	5682	3697	4	959043421	2.867807	643	M	23455-4959	18-24	other or not specified
25009	5714	3697	3	959223406	2.867807	643	M	96753	35-44	artist
25010	5767	3697	3	959620056	2.867807	643	M	75287	25-34	artist
25011	5831	3697	2	1024075950	2.867807	643	M	92120	25-34	academic/educator

```

1 show_distri_freq = show_distri.merge(user_occ_distri, left_on = 'occ_desc', right_on = 'occ_desc', how = 'inner')
2 show_distri_freq['freq'] = show_distri_freq['movie_id'] / show_distri_freq['user_id']
3 show_distri_freq.sort_values('freq', inplace = True)
4 show_distri_freq

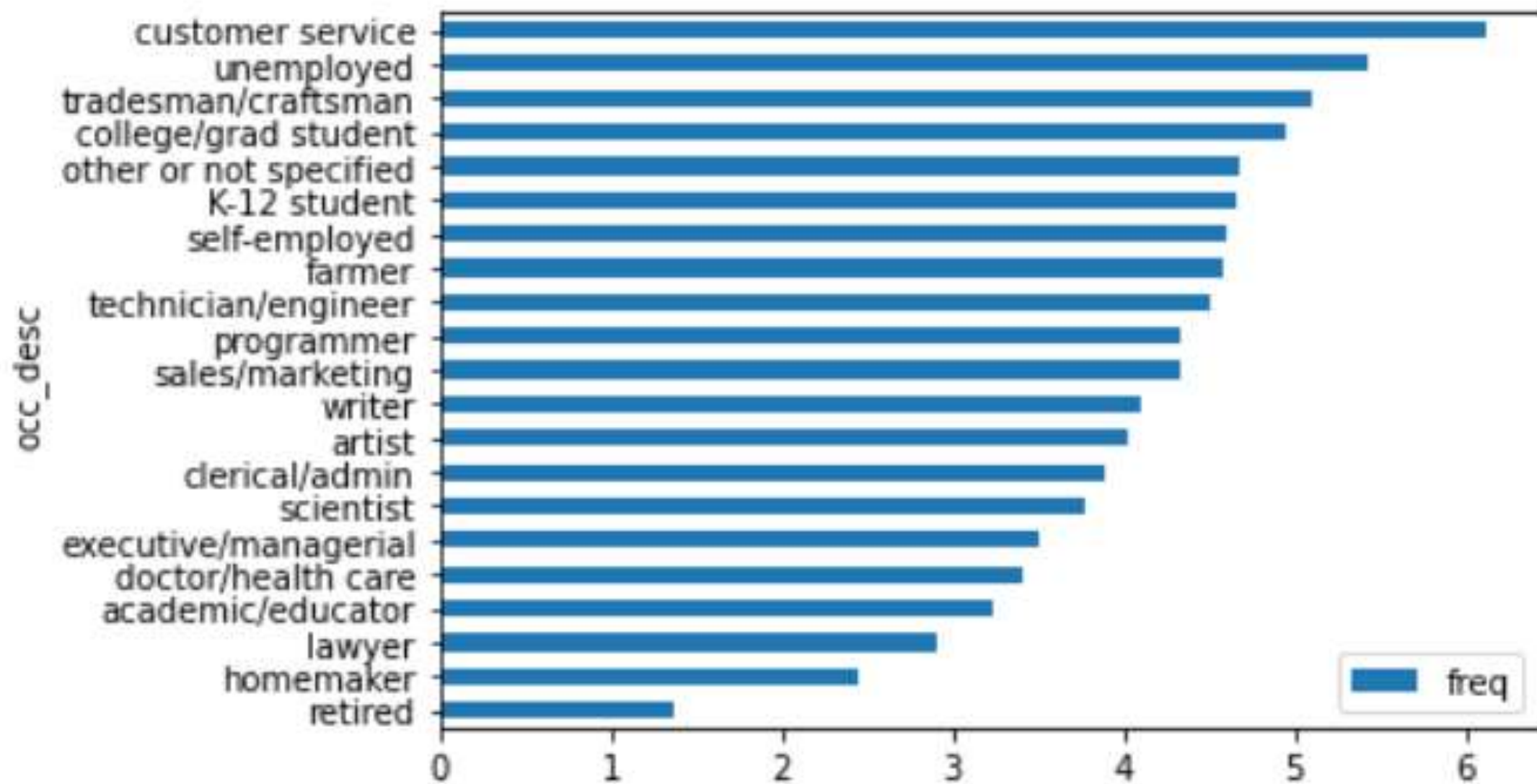
```

	movie_id	user_id	freq
occ_desc			
retired	194	142	1.366197
homemaker	225	92	2.445652
lawyer	375	129	2.906977
academic/educator	1707	528	3.232955
doctor/health care	804	236	3.406780
executive/managerial	2381	679	3.506627
scientist	543	144	3.770833
clerical/admin	675	173	3.901734
artist	1076	267	4.029963
writer	1151	281	4.096085
sales/marketing	1307	302	4.327815
programmer	1684	388	4.340206
technician/engineer	2267	502	4.515936

Where all the junks gone?

```
[167]: 1 show_distri_freq.plot.barh( y ='freq')
```

```
t[167]: <AxesSubplot:ylabel='occ_desc'>
```





# 回顾一下上述数据分析流程：

- 观察数据
- 挖掘有效信息：给出概念化表述、更好的数据认知
- 信息关联与分析
- 进一步给出关联信息分析结果的概念化表述与设想
  - 根据问题优化模型
  - 根据应用场景应用结果

