法律声明

- 本课件包括演示文稿、示例、代码、题库、视频和声音等内容，小象学院和主讲老师拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意及内容，我们保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

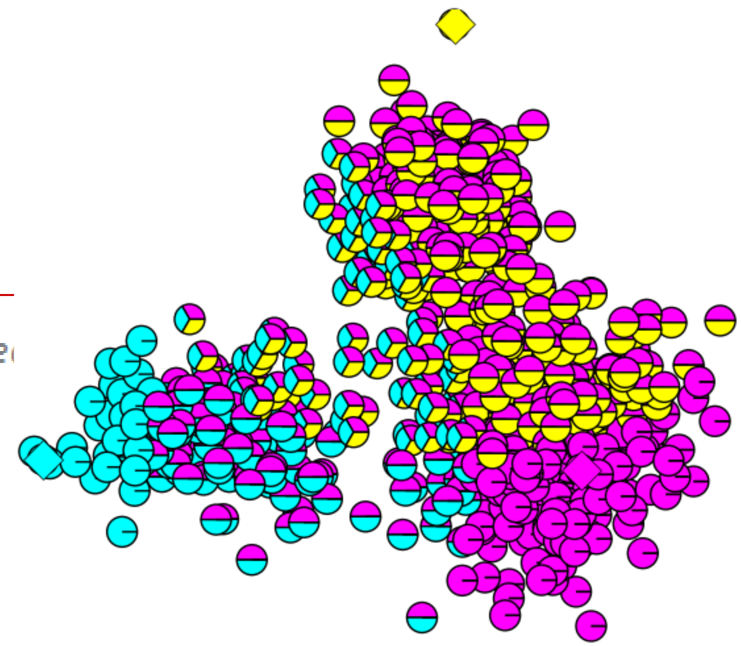
推荐系统

小象学院 ChinaHadoop.cn 邹博

朝花夕拾：Canopy算法

- 虽然Canopy算法可以划归为聚类算法，但更多的可以使用Canopy算法做空间索引，其时空复杂度都很出色，算法描述如下：
- 对于给定样本 $x_1, x_2 \cdots x_m$，给定先验值 r_1, r_2，$(r_1 < r_2)$
  - $x_1, x_2 \cdots x_m$ 形成列表L；构造 $x_j (1 \le j \le m)$ 的空列表 C_j；
  - 随机选择L中的样本c，要求c的列表 C_c 为空：
    - 计算L中样本 x_j 与c的距离 d_j
      - 若 $d_j < r_1$，则在L中删除 x_j，将 C_j 赋值为{c}
      - 否则，若 $d_j < r_2$，则将 C_j 增加{c}
  - 若L中没有不满足条件的样本c，算法结束。

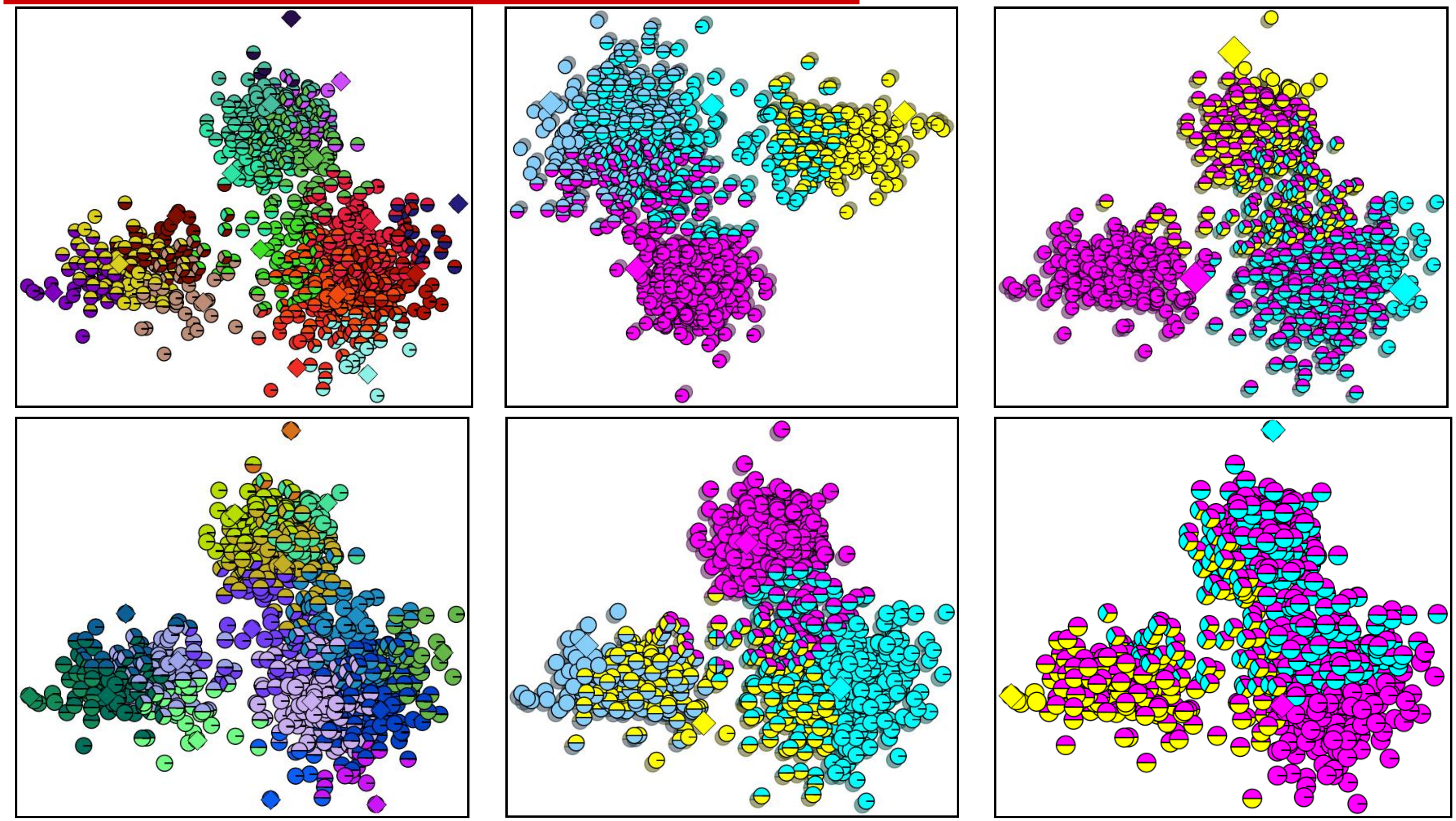
Code



```python
def canopy(data, d_near, d_far):        # d_ne
    m = len(data)        # 样本个数
    cluster = [[] for i in range(m)]
    center = []
    has = m
    while has > 0:
        i = np.random.randint(has)
        i = find_new_canopy(cluster, i)      # 查找cluster中第i个为空的值
        center.append(i)                     # canopy中心
        if i == -1:
            break
        for j in range(m):    # 针对新canopy中心data[i]，计算所有样本与之距离
            d = distance(data[i], data[j])
            if d < d_near:
                cluster[j] = [i]
            elif d < d_far:
                cluster[j].append(i)
        has = calc_candidate(cluster)
    return cluster, center
```

Canopy的调参

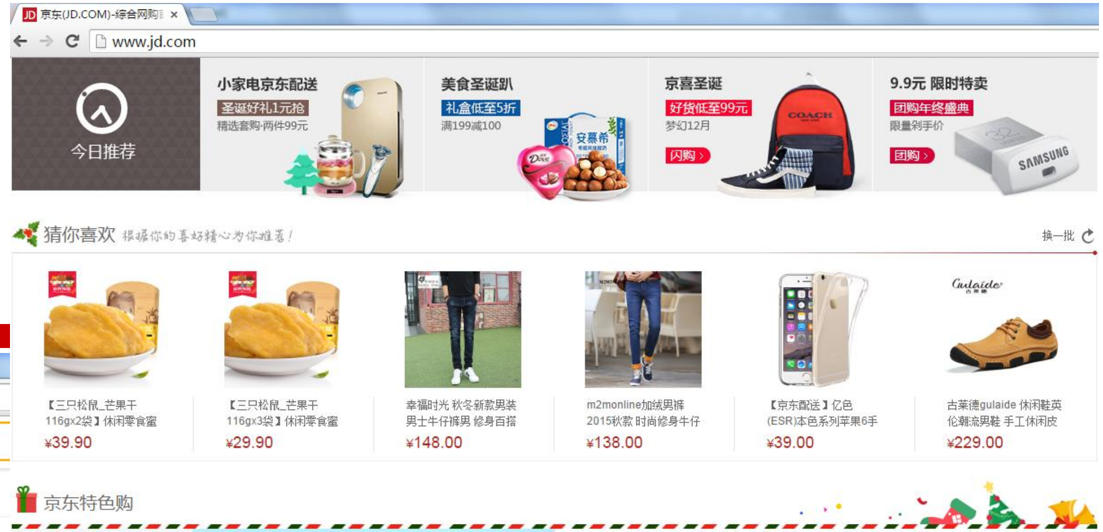
0.5	1	1.5
0.5	1	2

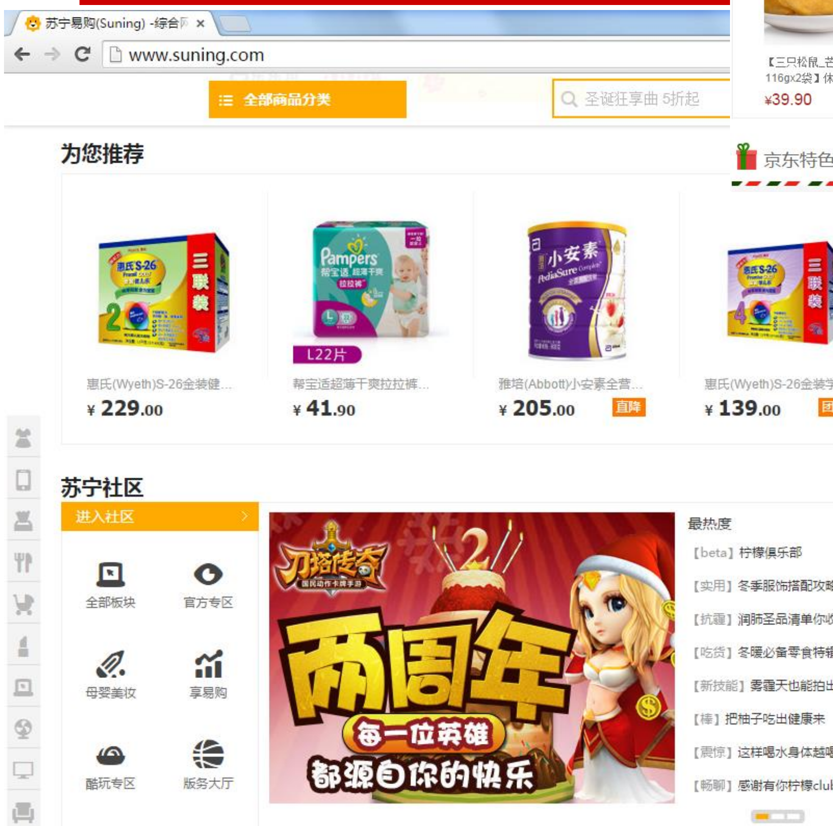


主要内容

- 推荐系统：协同过滤
  - user-based/item-based
- 主成分分析PCA
  - 二阶独立性
  - 思考：与ICA的区别
- 隐特征的个性化推荐
  - LFM：Latent Factor Model
    - 思考：LFM和pLSA的关系是什么？
  - SVD：奇异值分解
    - 思考：与LFM的优势在哪里？

推荐系统





互联网新技术在线教育领航者

陪我去看流星雨

- 某影院收集了N个用户对于M个电影的观影记录。每个用户一行，第i行记录形式为：
  “<用户ID>\t<电影1>;<电影2>;……”
- 已知莫愁的观影记录为：84, 14, 90, 91, 29, 21, 9, 44,24, 89, 8, 42, 41, 40, 25, 37, 30, 16, 97, 52, 62, 56,80, 83, 36, 26, 73, 64, 32, 27, 67, 65, 79, 87, 17。
- 找出与莫愁最匹配的前15名用户。
  - 思考：如何定义“最匹配”？

1347842	44
1347847	30;44
134790	28;30;97
1347912	44;30;78;28;58;68
1347915	8;30
1347916	24;8;58;44;83;32;28;26
1347925	3
1347943	8
1347949	28;58;35;30
1347974	30
1347982	30
1347991	83;77;57
1347999	30
1348027	30;8
1348066	71
1348088	30;97
1348110	83;30
1348118	28;47;30;16;58
1348135	30
1348148	47;48;44;58;33;30;76;18;28
1348153	52
134816	8;30;81
1348186	91
1348187	30
1348191	83;30;97
1348202	25
1348207	28;8
1348221	30;26
1348226	46;28;58
1348230	30
1348252	18
1348253	30;52
1348262	79;52;33

复习：相似度/距离计算方法

- 闵可夫斯基距离Minkowski/欧式距离 $dist(X,Y)=\left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{\frac{1}{p}}$
- 杰卡德相似系数(Jaccard) $J(A,B)=\frac{|A\cap B|}{|A\cup B|}$
- 余弦相似度(cosine similarity) $\cos(\theta)=\frac{a^T b}{|a|\cdot|b|}$
- Pearson相似系数 $\rho_{XY}=\frac{\operatorname{cov}(X,Y)}{\sigma_X\sigma_Y}=\frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X\sigma_Y}=\frac{\sum_{i=1}^{n}(X_i-\mu_X)(Y_i-\mu_Y)}{\sqrt{\sum_{i=1}^{n}(X_i-\mu_X)^2}\sqrt{\sum_{i=1}^{n}(Y_i-\mu_Y)^2}}$
- 相对熵(K-L距离) $D(p\|q)=\sum_x p(x)\log\frac{p(x)}{q(x)}=E_{p(x)}\log\frac{p(x)}{q(x)}$
- Hellinger距离 $D_\alpha(p\|q)=\frac{2}{1-\alpha^2}\left(1-\int p(x)^{\frac{1+\alpha}{2}}q(x)^{\frac{1-\alpha}{2}}dx\right)$

Code

```
if __name__ == "__main__":
    fp = file("D:\Python\\film2.txt")
    first_user = [1, 0, 0.0, 0, [84, 14, 90, 91, 29, 21, 9, 44,
        24, 89, 8, 42, 41, 40, 25, 37, 30, 16, 97, 52, 62, 56,
        80, 83, 36, 26, 73, 64, 32, 27, 67, 65, 79, 87, 17]]
    first_user[4].sort()
    for line in fp:
        d = line.split('\t')
        user = int(d[0])
        films = map(int, d[1].split(';'))
        films.sort()
        test_user = []
        test_user.append(user)  # 用户ID
        test_user.append(0)     # first和user观看的相同电影数目
        test_user.append(0.0)   # first和user的相似度
        test_user.append(len(films))     # user所看的电影数目
        test_user.append(films) # user的观看电影列表
        test_user[2] = calc_similarity(first_user[4], test_user[4])
        test_user[1] = calc_same(first_user[4], test_user[4])
        add_user(test_user)
    # 打印结果
    for u in similar_users:
        print u
```

```
1347842 44
1347847 30;44
134790  28;30;97
1347912 44;30;78;28;58;68
1347915 8;30
1347916 24;8;58;44;83;32;28;26
1347925 3
1347943 8
1347949 28;58;35;30
1347974 30
1347982 30
1347991 83;77;57
1347999 30
1348027 30;8
1348066 71
1348088 30;97
1348110 83;30
1348118 28;47;30;16;58
1348135 30
1348148 47;48;44;58;33;30;76;18;28
1348153 52
134816  8;30;81
1348186 91
1348187 30
1348191 83;30;97
1348202 25
1348207 28;8
1348221 30;26
1348226 46;28;58
1348230 30
1348252 18
1348253 30;52
1348262 79;52;33
```

Code

```python
def cos_similar(a1, a2):
    return float(calc_same(a1, a2)) / math.sqrt(len(a1)*len(a2))


def jaccard_similar(a1, a2):
    n = calc_same(a1, a2)
    return float(n) / float(len(a1) + len(a2) - n)


def eular_similar(a1, a2):
    return float(calc_same(a1, a2))


def calc_similarity(a1, a2):
    if similar == 1:    #夹角余弦
        return cos_similar(a1, a2)
    if similar == 2:
        return jaccard_similar(a1, a2)
    return eular_similar(a1, a2)
```

```python
def calc_same(a1, a2):
    n1 = len(a1)
    n2 = len(a2)
    i = 0
    j = 0
    count = 0
    while (i < n1) and (j < n2):
        if a1[i] > a2[j]:
            j += 1
        elif a1[i] < a2[j]:
            i += 1
        else:
            count += 1
            i += 1
            j += 1
    return count
```

余弦相似度

```
[1001129, 35, 1.0, 35, [8, 9, 14, 16, 17, 21, 24, 25, 26, 27, 29, 30, 32, 36, 37, 40, 41, 42, 44
[305344, 35, 0.606976978666884, 95, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18
[387418, 34, 0.5865557254410011, 96, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 1
[2118461, 30, 0.581675050747111, 76, [1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 16, 17, 18, 19, 20,
[1664010, 31, 0.5617822916268811, 87, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 18,
[2439493, 32, 0.5608858475195935, 93, [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 14, 15, 16, 17, 18,
[1114324, 22, 0.5606119105813882, 44, [5, 7, 8, 9, 12, 16, 17, 18, 21, 24, 26, 28, 30, 32, 3
[1403217, 16, 0.5204800389058843, 27, [2, 3, 5, 8, 9, 10, 16, 17, 23, 24, 26, 27, 28, 29, 30,
[825819, 11, 0.5156879540323449, 13, [8, 16, 17, 24, 26, 30, 32, 41, 44, 58, 73, 77, 83]]
[1932594, 21, 0.5019960159204453, 50, [3, 5, 6, 7, 9, 10, 15, 16, 17, 18, 19, 20, 24, 25, 26,
[1314869, 19, 0.4897622991151437, 43, [3, 5, 6, 8, 9, 13, 16, 17, 18, 23, 24, 26, 27, 28, 30,
[525356, 15, 0.4791574237499549, 28, [1, 6, 8, 9, 16, 17, 25, 26, 30, 33, 36, 41, 43, 48, 49,
[2606799, 17, 0.4661472225410195, 38, [2, 3, 5, 8, 16, 17, 23, 25, 26, 27, 28, 29, 30, 32, 3
[2056022, 19, 0.4635525346505533, 48, [1, 3, 8, 14, 17, 18, 19, 20, 21, 23, 24, 26, 27, 28, 3
[397001, 9, 0.458682472293863, 11, [8, 18, 24, 25, 29, 30, 44, 57, 79, 83, 97]]
[322009, 13, 0.4581897949526569, 23, [1, 8, 16, 17, 18, 19, 26, 28, 30, 32, 44, 45, 52, 55, 5
[2238060, 15, 0.4553825555391872, 31, [12, 16, 17, 18, 23, 24, 25, 26, 27, 28, 29, 30, 32, 3
[952063, 10, 0.45175395145262565, 14, [8, 16, 17, 24, 25, 26, 28, 30, 58, 76, 79, 83, 91, 92
[786312, 11, 0.4509560339299333, 17, [1, 3, 6, 18, 21, 24, 25, 30, 37, 44, 52, 56, 57, 62, 71
[1473980, 13, 0.4485426135725303, 24, [5, 8, 17, 18, 21, 24, 26, 28, 30, 33, 35, 36, 45, 55,
```

Jaccard相似度

```
[1001129, 35, 1.0, 35, [8, 9, 14, 16, 17, 21, 24, 25, 26, 27, 29, 30, 32, 36, 37, 40, 41, 42, 44
[1114324, 22, 0.38596491228070173, 44, [5, 7, 8, 9, 12, 16, 17, 18, 21, 24, 26, 28, 30, 32, 3
[2118461, 30, 0.37037037037037035, 76, [1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 16, 17, 18, 19, 20
[305344, 35, 0.3684210526315789, 95, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 1
[387418, 34, 0.35051546391752575, 96, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17,
[1403217, 16, 0.34782608695652173, 27, [2, 3, 5, 8, 9, 10, 16, 17, 23, 24, 26, 27, 28, 29, 30
[1664010, 31, 0.34065934065934067, 87, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 18,
[2439493, 32, 0.3333333333333333, 93, [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 14, 15, 16, 17, 18,
[1932594, 21, 0.328125, 50, [3, 5, 6, 7, 9, 10, 15, 16, 17, 18, 19, 20, 24, 25, 26, 28, 30, 37,
[1314869, 19, 0.3220338983050847, 43, [3, 5, 6, 8, 9, 13, 16, 17, 18, 23, 24, 26, 27, 28, 30,
[525356, 15, 0.3125, 28, [1, 6, 8, 9, 16, 17, 25, 26, 30, 33, 36, 41, 43, 48, 49, 52, 58, 62, 71
[2606799, 17, 0.30357142857142855, 38, [2, 3, 5, 8, 16, 17, 23, 25, 26, 27, 28, 29, 30, 32, 3
[825819, 11, 0.2972972972972973, 13, [8, 16, 17, 24, 26, 30, 32, 41, 44, 58, 73, 77, 83]]
[2056022, 19, 0.296875, 48, [1, 3, 8, 14, 17, 18, 19, 20, 21, 23, 24, 26, 27, 28, 30, 32, 33, 3
[2238060, 15, 0.29411764705882354, 31, [12, 16, 17, 18, 23, 24, 25, 26, 27, 28, 29, 30, 32,
[322009, 13, 0.28888888888888886, 23, [1, 8, 16, 17, 18, 19, 26, 28, 30, 32, 44, 45, 52, 55,
[1663888, 16, 0.2857142857142857, 37, [3, 6, 8, 10, 12, 16, 18, 24, 29, 33, 37, 44, 46, 47, 4
[1473980, 13, 0.2826086956521739, 24, [5, 8, 17, 18, 21, 24, 26, 28, 30, 33, 35, 36, 45, 55,
[1784150, 13, 0.2826086956521739, 24, [2, 5, 8, 16, 17, 18, 26, 28, 30, 32, 37, 40, 48, 52, 5
[786312, 11, 0.2682926829268293, 17, [1, 3, 6, 18, 21, 24, 25, 30, 37, 44, 52, 56, 57, 62, 71
```

欧拉相似度

```
[1001129, 35, 35.0, 35, [8, 9, 14, 16, 17, 21, 24, 25, 26, 27, 29, 30, 32, 36, 37, 40, 41, 42, 4
[305344, 35, 35.0, 95, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22
[387418, 34, 34.0, 96, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22,
[2439493, 32, 32.0, 93, [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2
[1664010, 31, 31.0, 87, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 18, 19, 20, 22, 23, 24
[2118461, 30, 30.0, 76, [1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
[1114324, 22, 22.0, 44, [5, 7, 8, 9, 12, 16, 17, 18, 21, 24, 26, 28, 30, 32, 33, 35, 40, 41, 42,
[1932594, 21, 21.0, 50, [3, 5, 6, 7, 9, 10, 15, 16, 17, 18, 19, 20, 24, 25, 26, 28, 30, 37, 38, 3
[1314869, 19, 19.0, 43, [3, 5, 6, 8, 9, 13, 16, 17, 18, 23, 24, 26, 27, 28, 30, 32, 33, 35, 37, 4
[1461435, 19, 19.0, 56, [2, 3, 4, 5, 6, 7, 8, 10, 15, 17, 19, 23, 24, 27, 28, 29, 31, 33, 34, 35,
[2056022, 19, 19.0, 48, [1, 3, 8, 14, 17, 18, 19, 20, 21, 23, 24, 26, 27, 28, 30, 32, 33, 35, 36
[2606799, 17, 17.0, 38, [2, 3, 5, 8, 16, 17, 23, 25, 26, 27, 28, 29, 30, 32, 35, 36, 40, 43, 44,
[1403217, 16, 16.0, 27, [2, 3, 5, 8, 9, 10, 16, 17, 23, 24, 26, 27, 28, 29, 30, 32, 33, 36, 47, 5
[1639792, 16, 16.0, 51, [3, 7, 10, 11, 12, 15, 16, 19, 23, 24, 29, 30, 31, 35, 36, 39, 41, 43, 4
[1663888, 16, 16.0, 37, [3, 6, 8, 10, 12, 16, 18, 24, 29, 33, 37, 44, 46, 47, 48, 50, 52, 54, 56
[2238060, 15, 15.0, 31, [12, 16, 17, 18, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 35, 44, 45, 47
[525356, 15, 15.0, 28, [1, 6, 8, 9, 16, 17, 25, 26, 30, 33, 36, 41, 43, 48, 49, 52, 58, 62, 71, 7
[1473980, 13, 13.0, 24, [5, 8, 17, 18, 21, 24, 26, 28, 30, 33, 35, 36, 45, 55, 57, 58, 64, 67, 7
[1784150, 13, 13.0, 24, [2, 5, 8, 16, 17, 18, 26, 28, 30, 32, 37, 40, 48, 52, 58, 68, 74, 77, 83
[2237185, 13, 13.0, 27, [3, 6, 8, 9, 16, 18, 19, 24, 26, 29, 30, 37, 44, 46, 47, 49, 50, 55, 57,
```

Jaccard相似度的由来

- 记：R(u)是给用户u作出的推荐列表，而T(u)是用户在测试集上真正的行为列表。
- 准确率/召回率：

$$\text{Precision}(\mathrm{u}) = \frac{\mathrm{R}(\mathrm{u}) \cap T(u)}{R(u)} \qquad \text{Recall}(\mathrm{u}) = \frac{\mathrm{R}(\mathrm{u}) \cap T(u)}{T(u)}$$

- Jaccard系数：

$$Jaccard(\mathrm{u}) = \frac{\mathrm{R}(\mathrm{u}) \cap T(u)}{R(u) \cup T(u)}$$

- Jaccard系数特点和用途：
  - 各个特征间是均一无权重的
  - 网页去重/考试防作弊系统/论文抄袭检查

评价推荐系统的首要离线指标

- 通过将单个用户的准确率(或召回率)做累加，即得到整个推荐系统的准确率(或召回率)，该离线指标常常用于比较各个推荐系统之间的优劣。

$$\mathrm{Precision}(\mathrm{u})=\frac{\mathrm{R}(\mathrm{u})\cap T(u)}{R(u)} \qquad \mathrm{Precision}=\frac{\sum_{\mathrm{u}\in \mathrm{U}}\mathrm{R}(\mathrm{u})\cap T(u)}{\sum_{\mathrm{u}\in \mathrm{U}}\mathrm{R}(\mathrm{u})}$$

$$\mathrm{Recall}(\mathrm{u})=\frac{\mathrm{R}(\mathrm{u})\cap T(u)}{T(u)} \qquad \mathrm{Recall}=\frac{\sum_{\mathrm{u}\in \mathrm{U}}\mathrm{R}(\mathrm{u})\cap T(u)}{\sum_{\mathrm{u}\in \mathrm{U}}T(u)}$$

$$\mathrm{F}_{\beta}=\frac{(1+\beta)^{2}\,\mathrm{Recall}\cdot\mathrm{Precision}}{\beta^{2}\cdot\mathrm{Recall}+\mathrm{Precision}}$$

评价推荐系统的其他指标

- 覆盖率：$Coverage = \left| \bigcup_{u \in U} R(u) \right| / |I|$
  - 考虑不同商品出现的次数(概率)，则可用信息熵或基尼系数。$H = -\sum_{i=1}^{n} p_i \ln p_i$　　$Gini = \sum_{i=1}^{n} p_i (1 - p_i)$
- 多样性：

$$Diversity(u) = 1 - \frac{\sum_{i,j \in R(u)} s(i,j)}{\frac{1}{2} |R(u)| (|R(u)| - 1)}$$

$$Diversity = \frac{1}{|U|} \sum_{u \in U} Diversity(u)$$

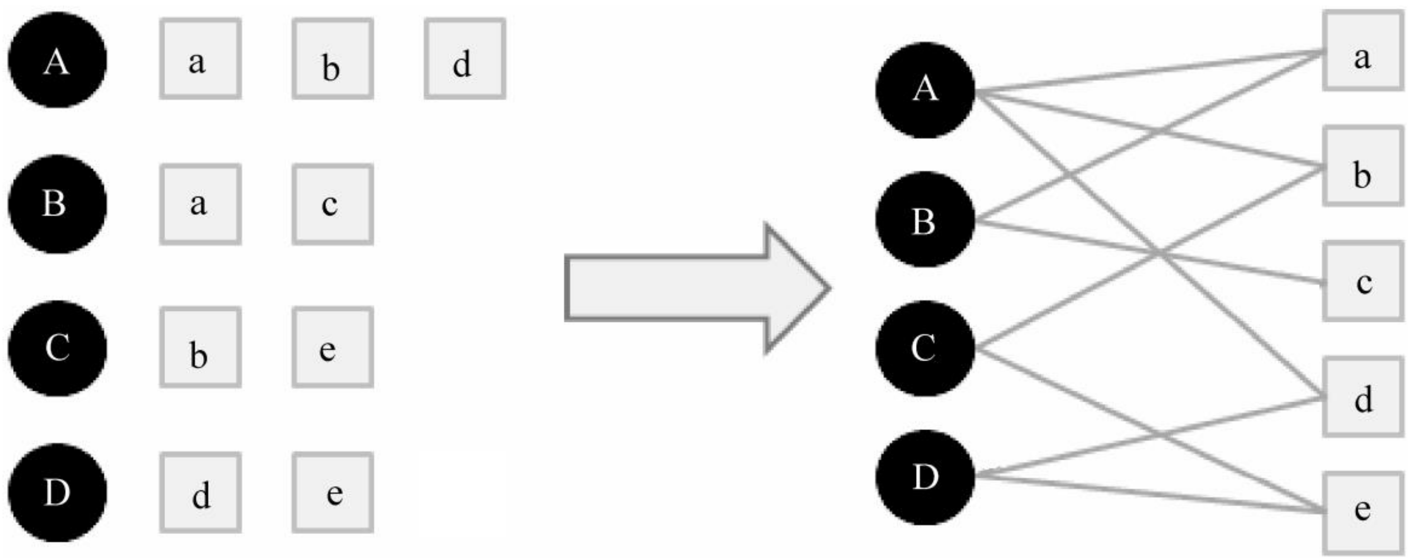
- 惊喜度(serendipity)：满意度/相似度
  - 用户惊喜来自于和用户喜欢的物品不相似，但用户却觉得满意的推荐。

总结与思考

- 根据用户观影数据，如何计算电影之间的相似度？
  - 遍历用户观看的电影列表，得到每个电影所对应的用户列表，然后调用前述代码即可。
  - Item-Based/User-Based
- 如果电影M1非常流行，相当数目的人都看过；电影M2流行度偏低，则如果两人都看过M2，则他们的相似度应该更高。
  - 适当提高非流行商品的权值。
- 基于用户行为的数据而设计的推荐算法被称为协同过滤算法(Collaborative Filtering, CF)。
  - ItemCF/UserCF

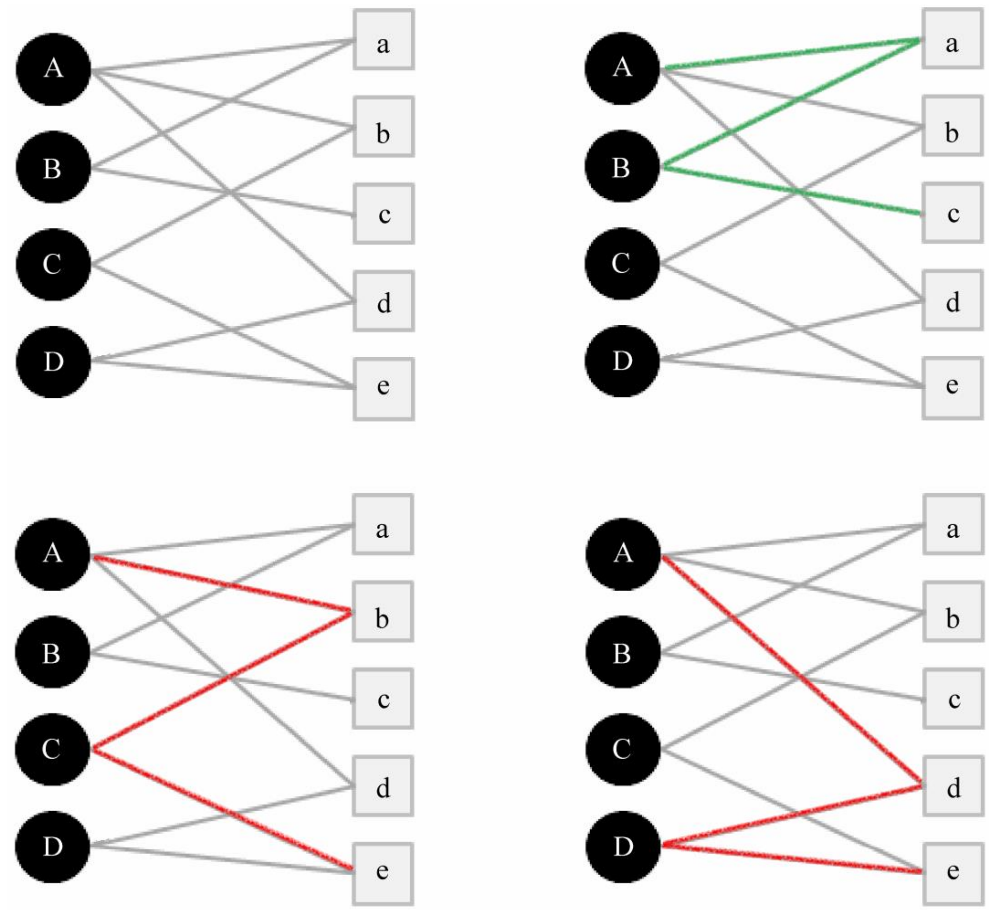
随机游走算法

- 假定只有4个用户，5个商品：
- 整理用户A、B、C、D对于商品a、b、c、d、e的喜爱列表，得到二部图。



分析最短路径

- A-c没有直接路径
- A-e没有直接路径
- A-a-B-c
- A-b-C-e
- A-d-D-e



随机游走：谱聚类、推荐系统

- 图论中的随机游走是一个随机过程，它从一个顶点跳转到另外一个顶点。谱聚类即找到图的一个划分，使得随机游走在相同的簇中停留而几乎不会游走到其他簇。
- 转移矩阵：从顶点vi跳转到顶点vj的概率正比于边的权值wij

$$p_{ij} = w_{ij} / d_i \qquad P = D^{-1}W$$

基于隐变量的推荐

- 模拟场景：假定Ben、Tom、John、Fred对6种商品进行了评价，评分越高代表对该商品越喜欢。0表示未评价。

	Ben	Tom	John	Fred
Season 1	5	5	0	5
Season 2	5	0	3	4
Season 3	3	4	0	3
Season 4	0	0	5	3
Season 5	5	4	4	5
Season 6	5	4	5	5

$$A=\begin{bmatrix}5&5&0&5\\5&0&3&4\\3&4&0&3\\0&0&5&3\\5&4&4&5\\5&4&5&5\end{bmatrix}$$

LFM (Latent Factor Model)

- 对于K个隐变量，得：$A_{m\times n} = U_{m\times k} \cdot V_{m\times k}^T$
- 目标函数：

$$J(U,V;A) = \sum_{i=1}^{m}\sum_{j=1}^{n}\left(a_{ij} - \sum_{r=1}^{k} u_{ir}\cdot v_{jr}\right)^2 + \lambda\left(\sum_{i=1}^{m}\sum_{r=1}^{k} u_{ir}^2 + \sum_{j=1}^{n}\sum_{r=1}^{k} u_{jr}^2\right)$$

- 梯度：

$$\begin{cases} \dfrac{\partial J(U,V;A)}{\partial u_{ir}} = -2\cdot\left(a_{ij} - \sum_{r=1}^{k} u_{ir}\cdot v_{jr}\right)\cdot v_{jr} + 2\lambda u_{ir} \\ \dfrac{\partial J(U,V;A)}{\partial v_{jr}} = -2\cdot\left(a_{ij} - \sum_{r=1}^{k} u_{ir}\cdot v_{jr}\right)\cdot u_{ir} + 2\lambda v_{jr} \end{cases}, 1 \le r \le k$$

Code

```python
def lmf(a, k):
    m = len(a)        # 用户数目
    n = len(a[0])     # 商品数目
    alpha = 0.01      # 学习率
    lamda = 0.01      # 惩罚因子
    u = np.random.rand(m,k)  # u[i][r]: 用户i和隐因子r的相关性
    v = np.random.rand(n,k)  # v[j][r]: 商品j和隐因子r的相关性
    for t in range(1000):
        for i in range(m):
            for j in range(n):
                if math.fabs(a[i][j]) > 1e-4:   # 只预测非零元素
                    err = a[i][j] - np.dot(u[i], v[j])  # 当前误差
                    for r in range(k):
                        gu = err * v[j][r] - lamda * u[i][r]    # 梯度
                        gv = err * u[i][r] - lamda * v[j][r]
                        u[i][r] += alpha * gu
                        v[j][r] += alpha * gv
    return u, v
```

k=2

$$A=\begin{bmatrix}5&5&0&5\\5&0&3&4\\3&4&0&3\\0&0&5&3\\5&4&4&5\\5&4&5&5\end{bmatrix}$$

- 用户-隐变量矩阵：

```
[ 1.91316162  1.48647507]
[ 2.00767948  0.01424435]
[ 0.99638121  1.65954907]
[ 0.92150319  1.69601488]
[ 2.11736323  0.50758455]
[ 2.01620667  0.90330121]
```

商品-隐变量矩阵：

```
[ 2.36060448  0.32842613]
[ 1.43507734  1.49737432]
[ 1.45241856  2.14869133]
[ 2.12189192  0.62295528]
```

- 预测矩阵：

```
[ 5.00441514  4.97134447  5.97268753  4.98552967]
[ 4.74401538  2.90250444  2.94659764  4.26895245]
[ 2.89710123  3.91485026  5.01302126  3.1480381 ]
[ 2.73232016  3.86199747  4.9826208   3.01187159]
[ 5.16496115  3.79862406  4.16594018  4.8090184 ]
[ 5.05613421  4.24599253  4.86929147  4.84088889]
```

k=3

$$A=\begin{bmatrix}5&5&0&5\\5&0&3&4\\3&4&0&3\\0&0&5&3\\5&4&4&5\\5&4&5&5\end{bmatrix}$$

- 用户-隐变量矩阵：

```
[ 1.09352762  1.07340551  1.69107405]
[ 0.52354011  1.84004003  1.30207454]
[ 0.08192958  0.66013988  1.60761524]
[ 1.84620189  0.87925007 -0.34397234]
[ 1.58109439  0.88730524  1.28310674]
[ 1.76468018  1.23593556  0.89748219]
```

商品-隐变量矩阵：

```
[ 1.2462658   1.44967981  1.24049425]
[ 0.2441074   1.51407311  1.82676406]
[ 2.07342683  1.21575006 -0.24481364]
[ 1.57119311  0.69362001  1.48336917]
```

- 预测矩阵：

```
[ 5.01668801  4.98134591  3.15834433  4.97116572]
[ 4.93515498  5.29233811  3.00378528  4.03032841]
[ 3.05333495  3.9562334   0.57887395  2.97130031]
[ 3.14879364  1.15356412  4.98112196  3.00036717]
[ 4.84845889  4.07333512  4.0429029   5.00297826]
[ 5.10429288  3.9415565   4.94180808  4.96122038]
```

隐变量数目对预测的影响

- k=2

```
[ 5.00441514  4.97134447  5.97268753  4.98552967]
[ 4.74401538  2.90250444  2.94659764  4.26895245]
[ 2.89710123  3.91485026  5.01302126  3.1480381 ]
[ 2.73232016  3.86199747  4.9826208   3.01187159]
[ 5.16496115  3.79862406  4.16594018  4.8090184 ]
[ 5.05613421  4.24599253  4.86929147  4.84088889]
```

$$A = \begin{bmatrix} 5 & 5 & 0 & 5 \\ 5 & 0 & 3 & 4 \\ 3 & 4 & 0 & 3 \\ 0 & 0 & 5 & 3 \\ 5 & 4 & 4 & 5 \\ 5 & 4 & 5 & 5 \end{bmatrix}$$

- k=3

```
[ 5.01668801  4.98134591  3.15834433  4.97116572]
[ 4.93515498  5.29233811  3.00378528  4.03032841]
[ 3.05333495  3.9562334   0.57887395  2.97130031]
[ 3.14879364  1.15356412  4.98112196  3.00036717]
[ 4.84845889  4.07333512  4.0429029   5.00297826]
[ 5.10429288  3.9415565   4.94180808  4.96122038]
```

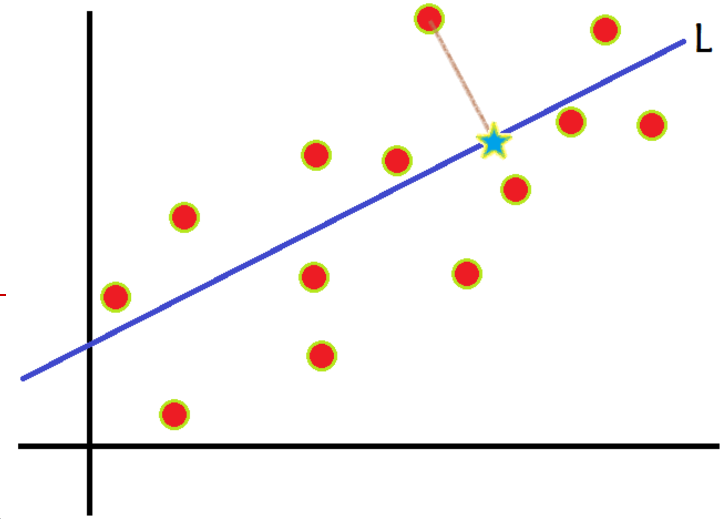
小结

- 使用随机梯度下降算法可以完成矩阵分解，从而获得“用户-隐变量矩阵”、“商品-隐变量矩阵”，对“用户-商品矩阵”的预测良好；
- 不同的隐变量数目对结果会产生一定的影响，需要交叉验证等调参工作；
- 如果分解后的矩阵存在负权值，虽然可解释，但需避免。

降维问题的提出

- 实际问题往往需要研究多个特征，而这些特征存在一定的相关性。
  - 数据量增加了问题的复杂性。
- 将多个特征综合为少数几个代表性特征：
  - 既能够代表原始特征的绝大多数信息，
  - 组合后的特征又互不相关，降低相关性。
  - 主成分
- 即主成分分析。

考察降维后的样本方差



- 对于n个特征的m个样本，将每个样本写成行向量，得到矩阵A

$$A=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}=\begin{pmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_{m-1}^T \\ a_m^T \end{pmatrix}$$

- 思路：寻找样本的主方向**u**：将m个样本值投影到某直线L上，得到m个位于直线L上的点，计算m个投影点的方差。认为方差最大的直线方向是主方向。
  - 假定样本是中心化的；若没有去均值化，则计算m个样本的均值，将样本真实值减去均值。

计算投影样本点的方差

- 取投影直线L的延伸方向u，计算A×u的值

$$A \cdot u = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \cdot u = \begin{pmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_{m-1}^T \\ a_m^T \end{pmatrix} \cdot u = \begin{pmatrix} a_1^T \cdot u \\ a_2^T \cdot u \\ \vdots \\ a_{m-1}^T \cdot u \\ a_m^T \cdot u \end{pmatrix}$$

- 求向量A×u的方差

$$Var(A \cdot u) = (Au - E)^T (Au - E) = (Au)^T (Au) = u^T A^T Au$$

- 目标函数：

$$J(u) = u^T A^T Au$$

目标函数 $J(u)=u^T A^T Au$

- 由于u数乘得到的方向和u相同，因此，增加u是单位向量的约束，即：$\|u\|_2=1$
- 从而：$\|u\|_2=1 \Rightarrow u^T u=1$
- 建立Lagrange方程：

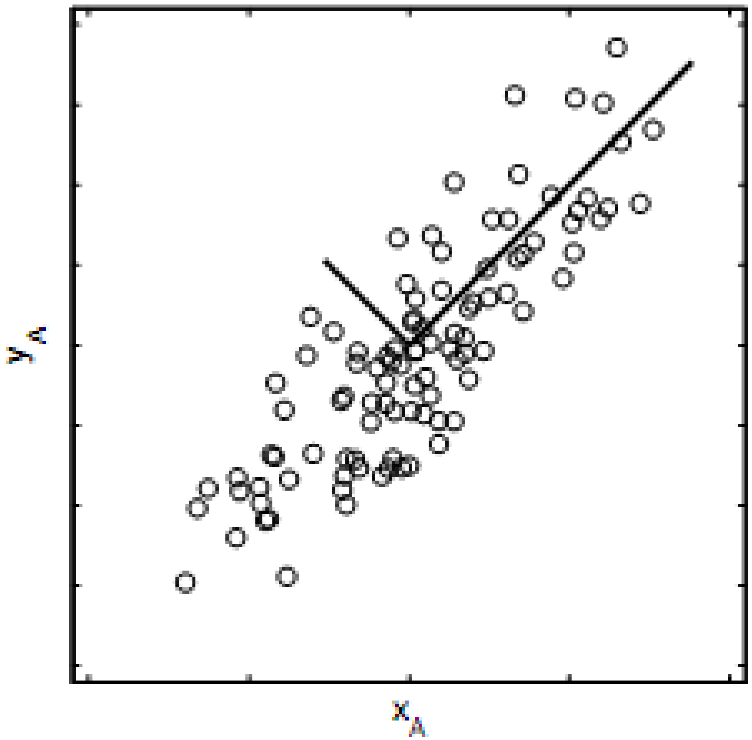
$$L(u)=u^T A^T Au-\lambda\left(u^T u-1\right)$$

$$\frac{\partial L(u)}{\partial u}=2A^T Au-2\lambda u \overset{令}{=} 0 \Rightarrow \left(A^T A\right)u=\lambda u$$

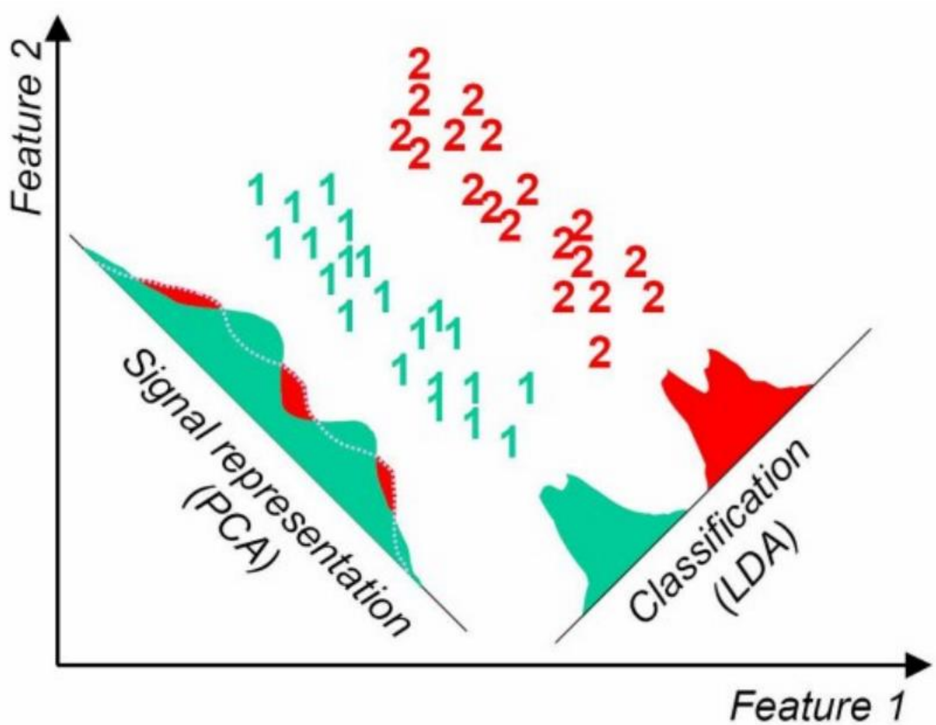
方差和特征值 $A^T Au = \lambda u$

- 若A中的样本都是去均值化的，则A^TA与A的协方差矩阵仅相差系数n-1
  - A^TA常常称为散列矩阵(scatter matrix)
- 根据上式，**u**是A^TA的一个特征向量，λ的值的大小为原始观测数据的特征在向量**u**的方向上投影值的方差。
- 以上即为主成分分析PCA的核心推导过程。

PCA的两个特征向量



y_A
x_A

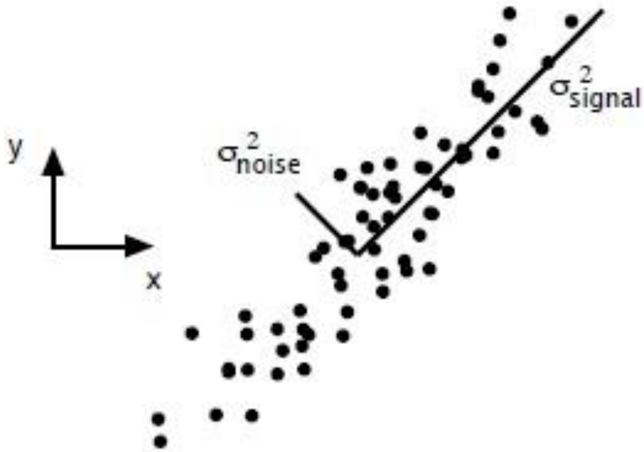


Feature 2
Feature 1
Signal representation
(PCA)
Classification
(LDA)

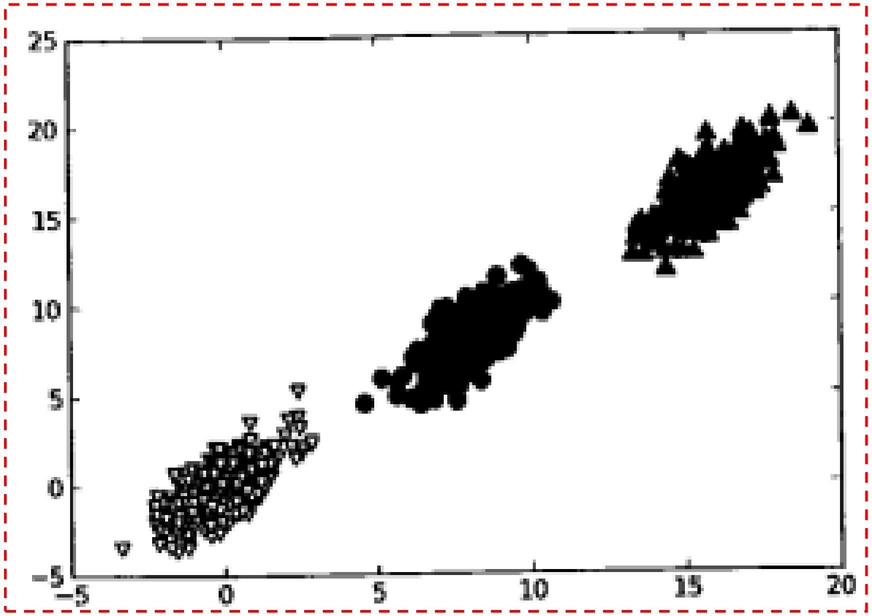
PCA的应用

- OBB树
  - Oriented Bounding Box
  - GIS中的空间索引
- 特征提取
- 数据压缩
  - 降维
  - 对原始观测数据A在λ值前k大的特征向量u上投影后，获得一个A(m×n)Q(n×k) 的序列，再加上特征向量矩阵Q，即将A原来的m×n个数据压缩到m×k+k×n个数据。

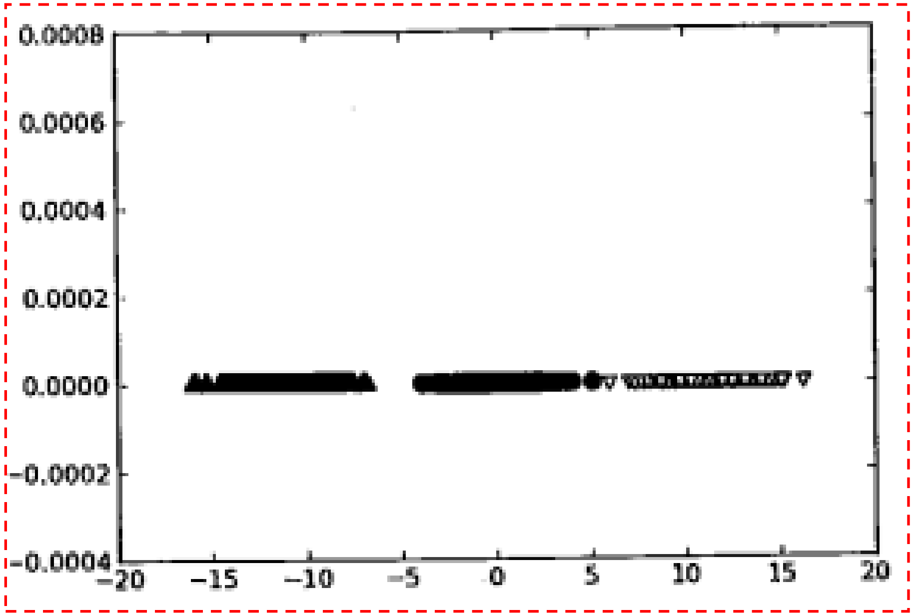
PCA的重要应用——去噪



PCA的重要应用——降维



25
20
15
10
5
0
−5
−5
0
5
10
15
20



0.0008
0.0006
0.0004
0.0002
0.0000
−0.0002
−0.0004
−20
−15
−10
−5
0
5
10
15
20

PCA总结

- 实对称阵的特征值一定是实数，不同特征值对应的特征向量一定正交，重数为r的特征值一定有r个线性无关的特征向量；
- 样本矩阵的协方差矩阵必然一定是对称阵，协方差矩阵的元素即各个特征间相关性的度量；
  - 具体实践中考虑是否去均值化；
- 将协方差矩阵C的特征向量组成矩阵P，可以将C合同为对角矩阵D，对角阵D的对角元素即为A的特征值。
  - $P^TCP=D$
  - 协方差矩阵的特征向量，往往单位化，即特征向量的模为1，从而，P是标准正交阵：$P^TP=I$。
  - 即将特征空间线性加权，使得加权后的特征组合间是不相关的。选择若干最大的特征值对应的特征向量(即新的特征组合)，即完成了PCA的过程。

关于PCA的进一步考察

- 若A是m×n阶矩阵，不妨认为m>n，则A^TA是n×n阶方阵。根据下式计算：

$$\left(A^T \cdot A\right)v_i = \lambda_i v_i \Rightarrow \begin{cases} \sigma_i = \sqrt{\lambda_i} \\ u_i = \dfrac{1}{\sigma_i} A \cdot v_i \end{cases} \Rightarrow A = U\Sigma V^T$$

- 从而，将矩阵A可以写成U、V两个方阵和对角矩阵D的乘积，这一过程，称作奇异值分解SVD。

SVD的推导和证明

- 记矩阵$A_{m\times n}$, $\Lambda = diag(\lambda_1, \lambda_2 \cdots \lambda_n)$, $\Sigma = diag(\sigma_1, \sigma_2 \cdots \sigma_n)$, $V = (v_1, v_i \cdots v_n)$
- 有 $(A^T \cdot A)v_i = \lambda_i v_i \Rightarrow V^T A^T AV = \Lambda$

$$\xrightarrow{\text{令}\sigma_i^2=\lambda_i} V^T A^T AV = \Sigma^2$$

$$\Rightarrow \Sigma^{-1} V^T A^T AV = \Sigma \Rightarrow (AV\Sigma^{-1})^T AV = \Sigma$$

$$\xrightarrow{\text{令}U=AV\Sigma^{-1}} U^T AV = \Sigma$$

$$\Rightarrow A = U\Sigma V$$

- 注：$U^T U = (\Sigma^{-1} V^T A^T) \cdot (AV\Sigma^{-1})$

$$= \Sigma^{-1} V^T \cdot (A^T A) V\Sigma^{-1} = \Sigma^{-1} V^T \cdot (V\Sigma^2 V^T) V\Sigma^{-1} = I$$

SVD的提法

$$\left(A^T \cdot A\right)v_i = \lambda_i v_i \Rightarrow \begin{cases} \sigma_i = \sqrt{\lambda_i} \\ u_i = \dfrac{1}{\sigma_i} A \cdot v_i \end{cases} \Rightarrow A = U\Sigma V^T$$

- 奇异值分解(Singular Value Decomposition)是一种重要的矩阵分解方法，可以看做对称方阵在任意矩阵上的推广。
  - Singular：突出的、奇特的、非凡的
  - 似乎更应该称之为“优值分解”
- 假设A是一个m×n阶实矩阵，则存在一个分解使得：

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V_{n\times n}^T$$

  - 通常将奇异值由大而小排列。这样，Σ便能由A唯一确定了。
- 与特征值、特征向量的概念相对应：
  - Σ对角线上的元素称为矩阵A的奇异值；
  - U的第i列称为A的关于σ_i的左奇异向量；
  - V的第i列称为A的关于σ_i的右奇异向量。

SVD举例 $A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$

- 已知4×5阶实矩阵A，求A的SVD分解：

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad V^T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ \sqrt{0.8} & 0 & 0 & 0 & -\sqrt{0.2} \end{bmatrix}$$

- 矩阵U和V都是单位正交方阵：$U^TU=I$， $V^TV=I$

奇异值分解不是唯一的

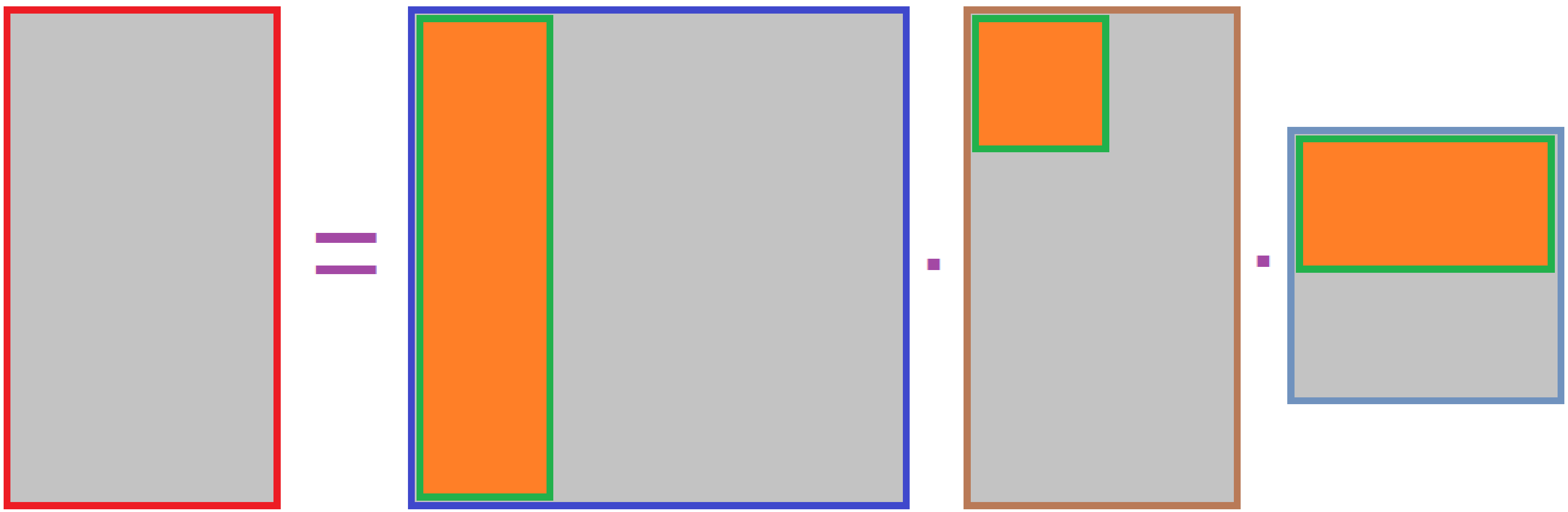
- 由于Σ有一个对角元是零，故这个奇异值分解值不是唯一的。

$$U=\begin{bmatrix}0&0&1&0\\0&1&0&0\\0&0&0&1\\1&0&0&0\end{bmatrix}\quad \Sigma=\begin{bmatrix}4&0&0&0&0\\0&3&0&0&0\\0&0&\sqrt{5}&0&0\\0&0&0&0&0\end{bmatrix}\quad V^T=\begin{bmatrix}0&1&0&0&0\\0&0&1&0&0\\\sqrt{0.2}&0&0&0&\sqrt{0.8}\\0&0&0&1&0\\\sqrt{0.8}&0&0&0&-\sqrt{0.2}\end{bmatrix}$$

$$U=\begin{bmatrix}0&0&1&0\\0&1&0&0\\0&0&0&1\\1&0&0&0\end{bmatrix}\quad \Sigma=\begin{bmatrix}4&0&0&0&0\\0&3&0&0&0\\0&0&\sqrt{5}&0&0\\0&0&0&0&0\end{bmatrix}\quad V^T=\begin{bmatrix}0&1&0&0&0\\0&0&1&0&0\\\sqrt{0.2}&0&0&0&\sqrt{0.8}\\\sqrt{0.4}&0&0&\sqrt{0.5}&-\sqrt{0.1}\\\sqrt{0.4}&0&0&\sqrt{0.5}&-\sqrt{0.1}\end{bmatrix}$$

SVD的四个矩阵 $A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$

- 实际中，往往只保留Σ前k个较大的数

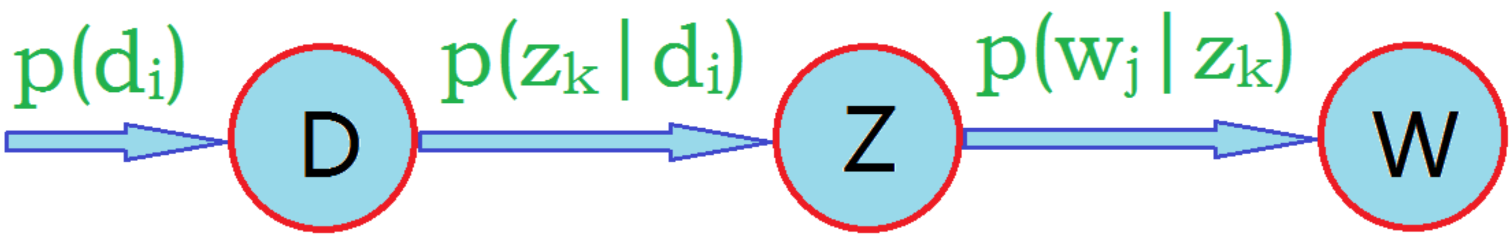


求伪逆

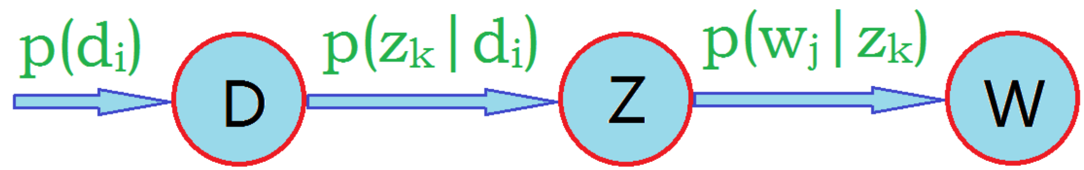
- 奇异值分解可以被用来计算矩阵的伪逆。若矩阵A的奇异值分解为$A=U\Sigma V^T$，那么A的伪逆为$A^+=V\Sigma^+U^T$
  - Σ是对角阵，其伪逆Σ^+由主对角线上每个非零元素求倒数得到。
  - 求伪逆通常可以用来求解**最小二乘法**问题。
- 复习：若A为非奇异矩阵，则线性方程组Ax=b的解为$A^+=\left(A^TA\right)^{-1}A^T$

SVD与pLSA

- 基于概率统计的pLSA模型(probabilistic latent semantic analysis, 概率隐语义分析)，增加了主题模型，形成简单的贝叶斯网络，可以使用EM算法学习模型参数。

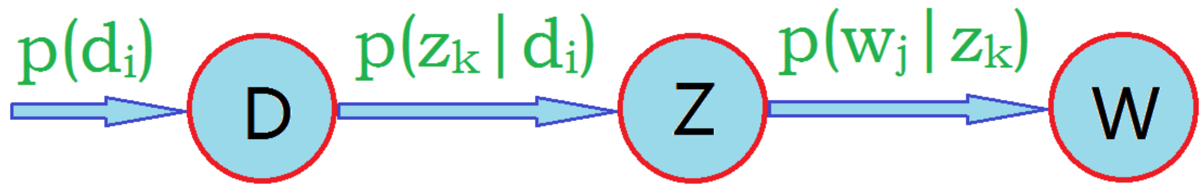


附：参数含义



- D代表文档，Z代表主题(隐含类别)，W代表单词；
  - $P(d_i)$表示文档d_i的出现概率，
  - $P(z_k|d_i)$表示文档d_i中主题z_k的出现概率，
  - $P(w_j|z_k)$表示给定主题z_k出现单词w_j的概率。
- 每个主题在所有词项上服从多项分布，每个文档在所有主题上服从多项分布。
- 整个文档的生成过程是这样的：
  - 以$P(d_i)$的概率选中文档d_i；
  - 以$P(z_k|d_i)$的概率选中主题z_k；
  - 以$P(w_j|z_k)$的概率产生一个单词w_j。

pLSA模型



- 观察数据为(d_i,w_j)对，主题z_k是隐含变量。
- (d_i,w_j)的联合分布为

$$P(d_i,w_j)=P(w_j\mid d_i)P(d_i)$$

$$P(w_j\mid d_i)=\sum_{k=1}^{K}P(w_j\mid z_k)P(z_k\mid d_i)$$

- 而 $P(w_j\mid z_k),P(z_k\mid d_i)$ 对应了两组多项分布，而计算每个文档的主题分布，就是该模型的任务目标。
- 事实上，上式即为矩阵相乘的公式。因此pLSA可以看做是概率化的矩阵分解。$A_{m\times n}=U_{m\times m}\Sigma_{m\times n}V_{n\times n}^{T}$

SVD举例

- 假定Ben、Tom、John、Fred对6种产品进行了评价，评分越高，代表对该产品越喜欢。0表示未评价。

	Ben	Tom	John	Fred
Season 1	5	5	0	5
Season 2	5	0	3	4
Season 3	3	4	0	3
Season 4	0	0	5	3
Season 5	5	4	4	5
Season 6	5	4	5	5

评分矩阵

	Ben	Tom	John	Fred
Season 1	5	5	0	5
Season 2	5	0	3	4
Season 3	3	4	0	3
Season 4	0	0	5	3
Season 5	5	4	4	5
Season 6	5	4	5	5

$$A=\begin{bmatrix}5 & 5 & 0 & 5\\ 5 & 0 & 3 & 4\\ 3 & 4 & 0 & 3\\ 0 & 0 & 5 & 3\\ 5 & 4 & 4 & 5\\ 5 & 4 & 5 & 5\end{bmatrix}$$

SVD分解 $A_{m\times n}=U_{m\times m}\Sigma_{m\times n}V_{n\times n}^{T}$

$$A=\begin{bmatrix}5&5&0&5\\5&0&3&4\\3&4&0&3\\0&0&5&3\\5&4&4&5\\5&4&5&5\end{bmatrix}\qquad U_{6\times 6}=\begin{bmatrix}-0.4472&-0.5373&-0.0064&-0.5037&-0.3857&-0.3298\\-0.3586&0.2461&0.8622&-0.1458&0.0780&0.2002\\-0.2925&-0.4033&-0.2275&-0.1038&0.4360&0.7065\\-0.2078&0.6700&-0.3951&-0.5888&0.0260&0.0667\\-0.5099&0.0597&-0.1097&0.2869&0.5946&-0.5371\\-0.5316&0.1887&-0.1914&0.5341&-0.5485&0.2429\end{bmatrix}$$

$$\Sigma_{6\times 4}=\begin{bmatrix}17.7139&0&0&0\\0&6.3917&0&0\\0&0&3.0980&0\\0&0&0&1.3290\\0&0&0&0\\0&0&0&0\end{bmatrix}\qquad V_{4\times 4}^{T}=\begin{bmatrix}-0.5710&-0.2228&0.6749&0.4109\\-0.4275&-0.5172&-0.6929&0.2637\\-0.3846&0.8246&-0.2532&0.3286\\-0.5859&0.0532&0.0140&-0.8085\end{bmatrix}$$

SVD分解，取k=2 $A_{m\times n}=U_{m\times m}\Sigma_{m\times n}V_{n\times n}^T$

$$A=\begin{bmatrix}5&5&0&5\\5&0&3&4\\3&4&0&3\\0&0&5&3\\5&4&4&5\\5&4&5&5\end{bmatrix}$$

$$U_{6\times 6}=\begin{bmatrix}-0.4472&-0.5373&-0.0064&-0.5037&-0.3857&-0.3298\\-0.3586&0.2461&0.8622&-0.1458&0.0780&0.2002\\-0.2925&-0.4033&-0.2275&-0.1038&0.4360&0.7065\\-0.2078&0.6700&-0.3951&-0.5888&0.0260&0.0667\\-0.5099&0.0597&-0.1097&0.2869&0.5946&-0.5371\\-0.5316&0.1887&-0.1914&0.5341&-0.5485&0.2429\end{bmatrix}$$

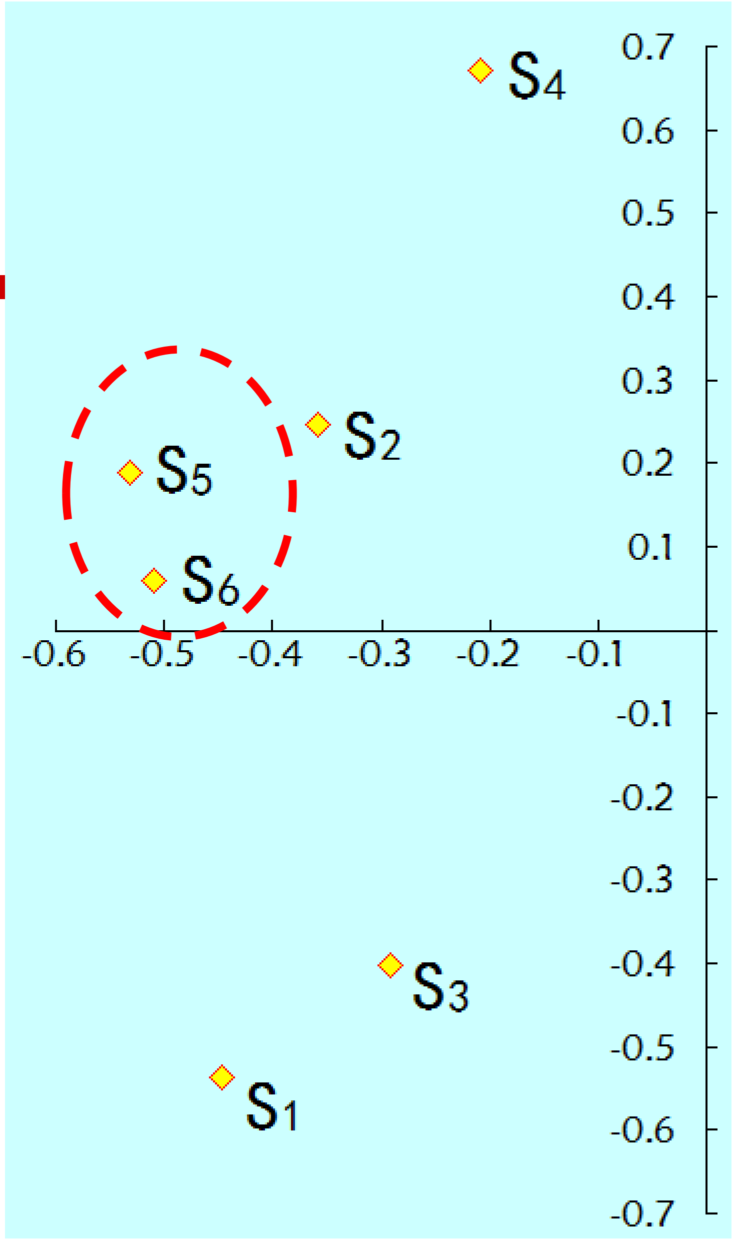
$$\Sigma_{6\times 4}=\begin{bmatrix}17.7139&0&0&0\\0&6.3917&0&0\\0&0&3.0980&0\\0&0&0&1.3290\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

$$V_{4\times 4}^T=\begin{bmatrix}-0.5710&-0.2228&0.6749&0.4109\\-0.4275&-0.5172&-0.6929&0.2637\\-0.3846&0.8246&-0.2532&0.3286\\-0.5859&0.0532&0.0140&-0.8085\end{bmatrix}$$

产品矩阵的压缩

$$U_{6\times 6}=\begin{bmatrix} -0.4472 & -0.5373 & -0.0064 & -0.5037 \\ -0.3586 & 0.2461 & 0.8622 & -0.1458 \\ -0.2925 & -0.4033 & -0.2275 & -0.1038 \\ -0.2078 & 0.6700 & -0.3951 & -0.5888 \\ -0.5099 & 0.0597 & -0.1097 & 0.2869 \\ -0.5316 & 0.1887 & -0.1914 & 0.5341 \end{bmatrix}$$

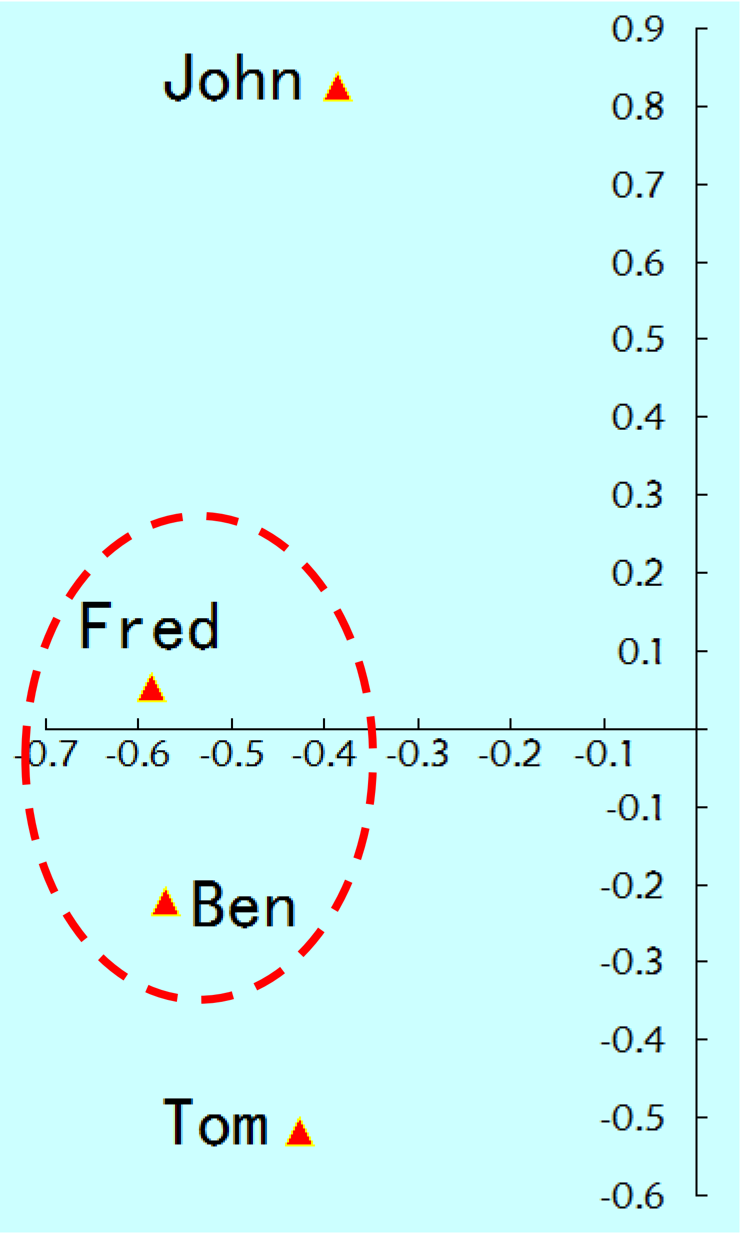
$$A=\begin{bmatrix} 5 & 5 & 0 & 5 \\ 5 & 0 & 3 & 4 \\ 3 & 4 & 0 & 3 \\ 0 & 0 & 5 & 3 \\ 5 & 4 & 4 & 5 \\ 5 & 4 & 5 & 5 \end{bmatrix}$$



用户矩阵的压缩

$$V_{4\times4}^{T}=\begin{bmatrix} -0.5710 & -0.2228 & 0.6749 & 0.4109 \\ -0.4275 & -0.5172 & -0.6929 & 0.2637 \\ -0.3846 & 0.8246 & -0.2532 & 0.3286 \\ -0.5859 & 0.0532 & 0.0140 & -0.8085 \end{bmatrix}$$

$$A=\begin{bmatrix} 5 & 5 & 0 & 5 \\ 5 & 0 & 3 & 4 \\ 3 & 4 & 0 & 3 \\ 0 & 0 & 5 & 3 \\ 5 & 4 & 4 & 5 \\ 5 & 4 & 5 & 5 \end{bmatrix}$$



新用户的个性化推荐

- 对于新用户，如何对其做个性化推荐呢？
  - 将A扩展后重新计算SVD，然后聚类用户？
  - 事实上

$$
\begin{aligned}
& A = U \cdot \Sigma \cdot V^T \\
& \Rightarrow U^T A = U^T U \cdot \Sigma \cdot V^T \\
& \Rightarrow U^T A = \Sigma \cdot V^T \\
& \Rightarrow \Sigma^{-1} U^T A = \Sigma^{-1} \Sigma \cdot V^T \\
& \Rightarrow \Sigma^{-1} U^T A = V^T \\
& \Rightarrow \left(\Sigma^{-1} U^T A \right)^T = V \\
& \Rightarrow A^T U \Sigma^{-1} = V
\end{aligned}
$$

新用户的个性化推荐 $V = A^T \cdot U \cdot \Sigma^{-1}$

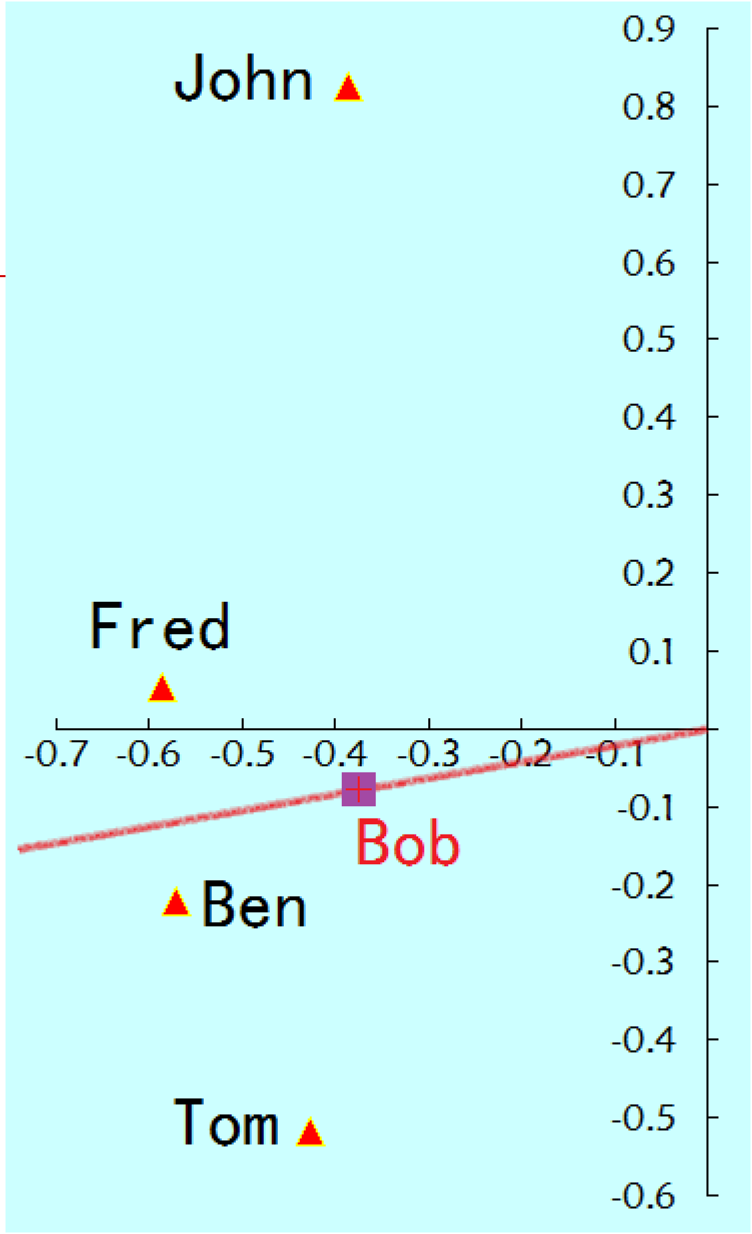
- 假设有个Bob的新用户，对6个产品的评分为 $(5,5,0,0,0,5)^{\mathrm{T}}$，则：

$$V = a^T \cdot U \cdot \Sigma^{-1} = (5,5,0,0,0,5) \cdot \begin{bmatrix} -0.4472 & -0.5373 \\ -0.3586 & 0.2461 \\ -0.2925 & -0.4033 \\ -0.2078 & 0.6700 \\ -0.5099 & 0.0597 \\ -0.5316 & 0.1887 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{17.7139} & 0 \\ 0 & \frac{1}{6.3917} \end{bmatrix}$$

$$= (-0.3775, -0.0802)$$

个性化推荐

- 计算新加的Bob和现有用户的距离：余弦距离(一定意义下即相关系数)，最近的是Ben。
- Ben：5 5 3 0 5 5
- Bob：5 5 0 0 0 5
  - 因此，可顺次推荐S5、S3



PCA和SVD总结

- 矩阵对向量的乘法，对应于对该向量的旋转、伸缩。如果对某向量只发生了伸缩而无旋转变化，则该向量是该矩阵的特征向量，伸缩比即为特征值。
  - PCA用来提取一个场的主要信息(即主成分分量)，而SVD一般用来分析两个场的相关关系。两者在具体的实现方法上也有不同，SVD是通过矩阵奇异值分解的方法分解两个场的协方差矩阵的，而PCA是通过分解一个场的协方差矩阵。
  - PCA可用于特征的压缩、降维；当然也能去噪等；如果将矩阵转置后再用PCA，相当于去除相关度过大的样本数据——但不常见；SVD能够对一般矩阵分解，并可用于个性化推荐等内容。

合理解释该现象

邹博_百度搜索
https://www.baidu.com/s?ie=UTF-8&wd=邹博
邹博
百度一下
网页 新闻 贴吧 知道 音乐 图片 视频 地图 文库 更多»
百度为您找到相关结果约124,000个
搜索工具
今天机器学习班讲了"著名"的SVM,从复习凸...来自邹博_机器学习...
2015年4月18日 - 邹博_机器学习 邹博_机器学习 学习与探索者 +关注 276关注 7398粉丝 422微博 c 今天机器学习班讲了"著名"的SVM,从复习凸优化对偶问题开始,重述了...
weibo.com/2306141363/C... - 百度快照 - 87%好评
Machine Learning在线公开课第 4 期:2.1日晚邹博讲EM、GMM结束...
2015年2月2日 - Machine Learning在线公开课第 4 期:2.1日晚邹博讲EM、GMM结束,PPT帖内下载 - 前三期,我们在线上分别给大家讲了决策树与随机森林、计算广告、logistic回归,详见...
邹博_机器学习的微博_微博
@JENNYZHANG宁儿 //@JENNYZHANG宁儿:机器学习走起,看看自己能否坚持到取得真经[嘻嘻] //@邹博_机器学习:一起机器学习吧:用Python做代码实现,不调用库函数,从...
weibo.com/2306141363/p... - 百度快照 - 87%好评
其他人还搜
展开
陈寅恪
四大哲人
机器学习实战
PeterHarrington著
清通鉴
编年体清代史书
曾国藩
晚清中兴四大名臣
相关学者
展开
易中天
著名学者
章开沅
中国历史学家教育家
霍斐然
中国易经学者
邹广田
中科院院士
相关人物
展开
岳普煜
王金富
杨幂
残雪
相关搜索
邹博 机器学习
邹博简介
邹
邹博 史学家 简介
邹博 机器学习 视频
周伯通
1 2 3 4 5 6 7 8

清通鉴 邹博
相关搜索
清通鉴
清通鉴 中华书局
清通鉴 下载
清通鉴pdf下载
清通鉴pdf
清通鉴 戴逸
邹博 机器学习
邹博 机器学习 视频
邹博简介
1 2 3 4 5 6 7 8 下一页>

思考

- 对于用户观影数据，考虑电影流行度重新设计相似度函数，修正用户相似度公式。
  - 原公式：

$$Jaccard(\mathrm{u})=\frac{\mathrm{R}(\mathrm{u})\cap T(u)}{R(u)\cup T(u)}$$

参考文献

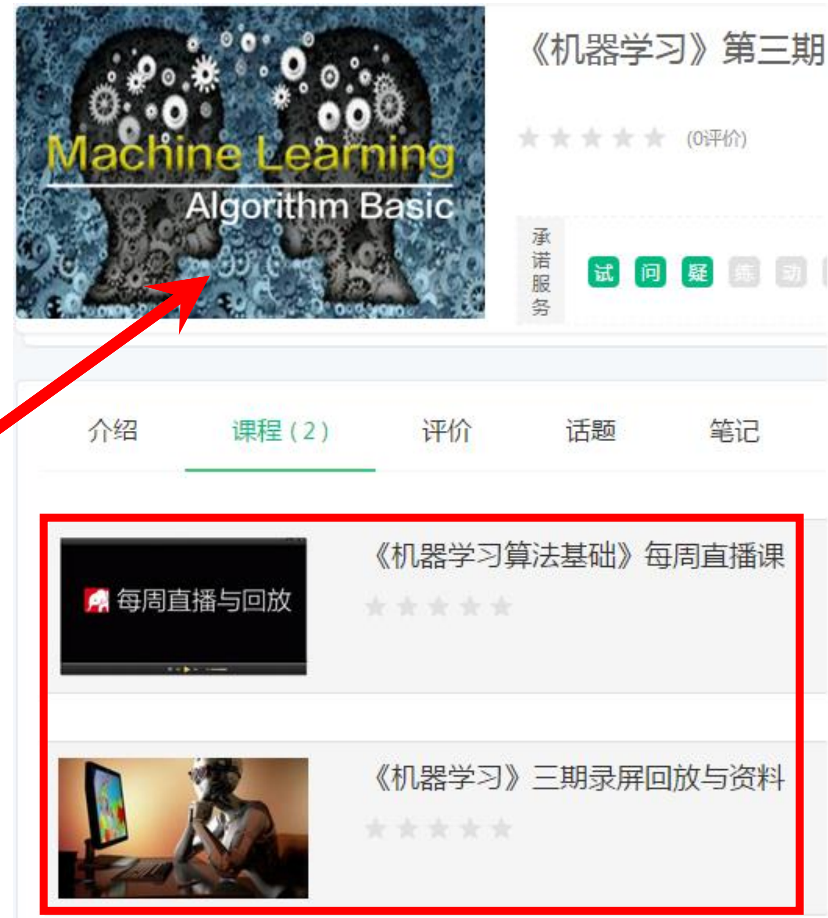
- 项亮，推荐系统实践，人民邮电出版社，2012
- Ulrike von Luxburg. *A tutorial on spectral clustering*, 2007

我们在这里

- http://wenda.ChinaHadoop.cn
  - 视频/课程/社区
- 微博
  - @ChinaHadoop
  - @邹博_机器学习
- 微信公众号
  - 小象
  - 大数据分析挖掘

课程资源

- 直播课的入口
- 录播视频和讲义资料



感谢大家！

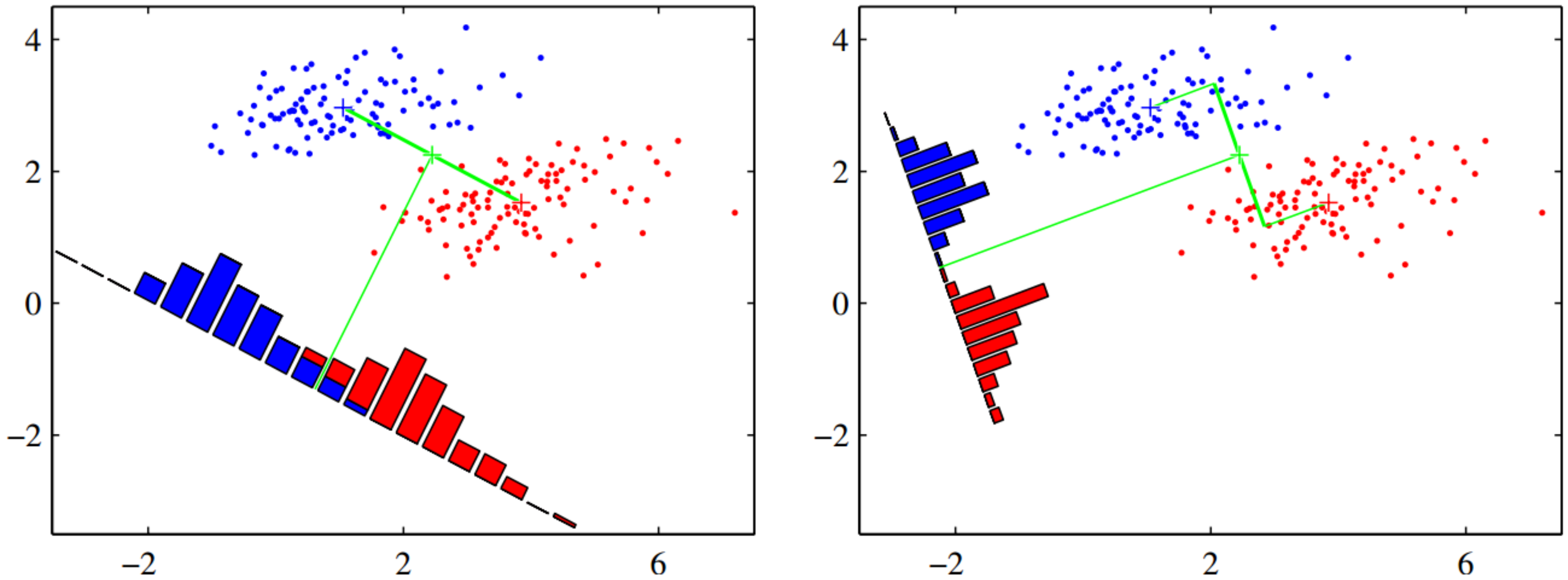
恳请大家批评指正！

附：Linear Discriminant Analysis

- 给定若干样本($\mathbf{x}_i$,c_i)，其中标记只分两类：c_i=0或者c_i=1，设计分类器，将样本分开。
- 方法：
  - Logistic回归/Softmax回归(MaxEnt)
  - SVM
  - 随机森林
  - LDA：Fisher's linear discriminant

LDA的思路

- 假定两类数据线性可分，即：存在一个超平面，将两类数据分开。则：存在某旋转向量，将两类数据投影到1维，并且可分。



LDA的推导

- 假定旋转向量为w，将数据x投影到一维y，得到

$$y = \vec{w}^T \vec{x}$$

- 从而，可以方便的找到阈值w_0，$y \geq w_0$时为C_1类，否则为C_2类。

类内均值和方差

- 令C_1有N_1个点，C_2有N_2个点，投影前的类内均值和投影后的类内均值、松散度为：

$$\begin{cases}\vec{m}_1 = \dfrac{1}{N_1}\sum_{i=1}^{N_1}\vec{x}_i \\ \vec{m}_2 = \dfrac{1}{N_2}\sum_{i=1}^{N_2}\vec{x}_i\end{cases} \qquad \begin{cases}m_1 = w^T\vec{m}_1 \\ m_2 = w^T\vec{m}_2\end{cases} \qquad \begin{cases}s_1^2 = \sum_{i=1}^{N_1}(y_i - m_1)^2 \\ s_2^2 = \sum_{i=1}^{N_2}(y_i - m_2)^2\end{cases}$$

- 松散度(scatter)，一般称为散列值，是样本松散程度的度量，值越大，越分散。
- 严格的说，m_2应该写成：$\vec{m}_2 = \dfrac{1}{N_2}\sum_{i=N_1+1}^{N_1+N_2}\vec{x}_i$

Fisher判别准则

- 目标函数：$J(\vec{w})=\dfrac{(m_2-m_1)^2}{s_1^2+s_2^2} \Rightarrow J(\vec{w})=\dfrac{\vec{w}^T S_B \vec{w}}{\vec{w}^T S_W \vec{w}}$
- 向量表示：$(m_2-m_1)^2=\left(w^T\vec{m}_2-w^T\vec{m}_1\right)^2$

$$
\begin{aligned}
&=\left(w^T(\vec{m}_2-\vec{m}_1)\right)^2=\left((\vec{m}_2-\vec{m}_1)^T w\right)^2\\
&=\left((\vec{m}_2-\vec{m}_1)^T w\right)^T\left((\vec{m}_2-\vec{m}_1)^T w\right)\\
&=\left(w^T(\vec{m}_2-\vec{m}_1)\right)\left((\vec{m}_2-\vec{m}_1)^T w\right)\\
&=w^T\left((\vec{m}_2-\vec{m}_1)(\vec{m}_2-\vec{m}_1)^T\right)w^T\\
&\xleftrightarrow{\text{令}S_B=(\vec{m}_2-\vec{m}_1)(\vec{m}_2-\vec{m}_1)^T} w^T S_B w
\end{aligned}
$$

Fisher判别准则

- 目标函数：$$J(\vec{w})=\frac{(m_2-m_1)^2}{s_1^2+s_2^2} \Rightarrow J(\vec{w})=\frac{\vec{w}^T S_B \vec{w}}{\vec{w}^T S_W \vec{w}}$$
- 其中：$$S_B=(\vec{m}_2-\vec{m}_1)(\vec{m}_2-\vec{m}_1)^T$$

$$S_W=\left(\sum_{i=1}^{N_1}(\vec{x}_i-\vec{m}_1)(\vec{x}_i-\vec{m}_1)^T\right)+\left(\sum_{i=1}^{N_2}(\vec{x}_i-\vec{m}_2)(\vec{x}_i-\vec{m}_2)^T\right)$$

- Within-class scatter matrix
- Between-class scatter
- S_w, S_b可以通过样本计算得到(已知)。

目标函数求极值 $J(\vec{w})=\dfrac{\vec{w}^T S_B \vec{w}}{\vec{w}^T S_W \vec{w}}$

- 求驻点：

$$\frac{\partial J(\vec{w})}{\partial \vec{w}}=\left(\frac{\vec{w}^T S_B \vec{w}}{\vec{w}^T S_W \vec{w}}\right)'$$

$$=\frac{\left(\vec{w}^T S_B \vec{w}\right)'\left(\vec{w}^T S_W \vec{w}\right)-\left(\vec{w}^T S_W \vec{w}\right)'\left(\vec{w}^T S_B \vec{w}\right)}{\left(\vec{w}^T S_W \vec{w}\right)^2}$$

$$=\frac{2S_B\vec{w}\left(\vec{w}^T S_W \vec{w}\right)-2S_W\vec{w}\left(\vec{w}^T S_B \vec{w}\right)}{\left(\vec{w}^T S_W \vec{w}\right)^2}\overset{\text{令}}{=}0$$

$$\Rightarrow S_B\vec{w}\left(\vec{w}^T S_W \vec{w}\right)=S_W\vec{w}\left(\vec{w}^T S_B \vec{w}\right)$$

$$\Rightarrow S_B\vec{w}\propto S_W\vec{w}$$

Fisher判别投影向量公式

- 以上推导得到 $S_B\vec{w} \propto S_W\vec{w}$
- 根据S_B的计算公式 $S_B = (\vec{m}_2 - \vec{m}_1)(\vec{m}_2 - \vec{m}_1)^T$
- 得：

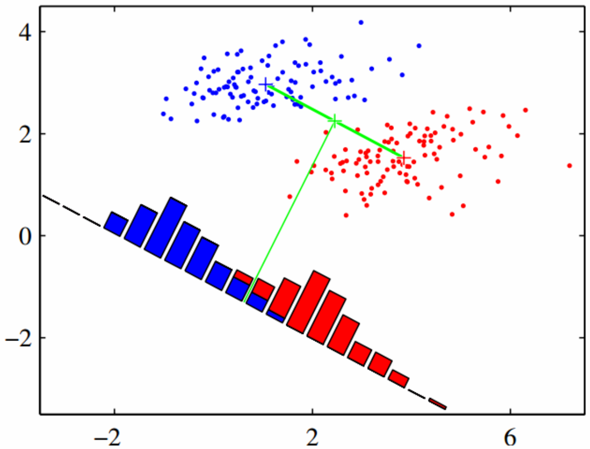
$$
\begin{aligned}
S_B\vec{w} &= (\vec{m}_2 - \vec{m}_1)(\vec{m}_2 - \vec{m}_1)^T \vec{w} \\
&= (\vec{m}_2 - \vec{m}_1)\left((\vec{m}_2 - \vec{m}_1)^T \vec{w}\right) \propto (\vec{m}_2 - \vec{m}_1)
\end{aligned}
$$

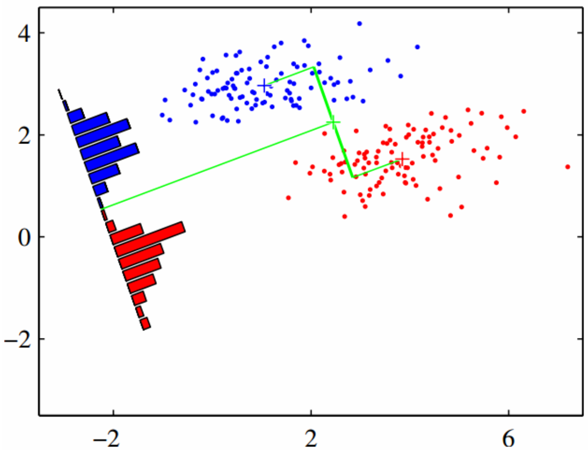
- 从而：

$$S_W\vec{w} \propto S_B\vec{w} \propto \vec{m}_2 - \vec{m}_1$$

- 若S_W可逆，则：$\vec{w} \propto S_W^{-1}(\vec{m}_2 - \vec{m}_1)$

Code

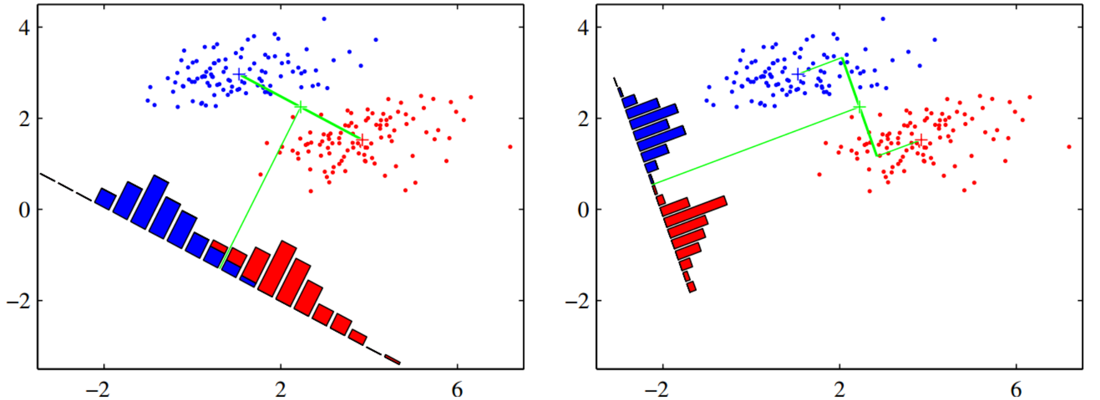




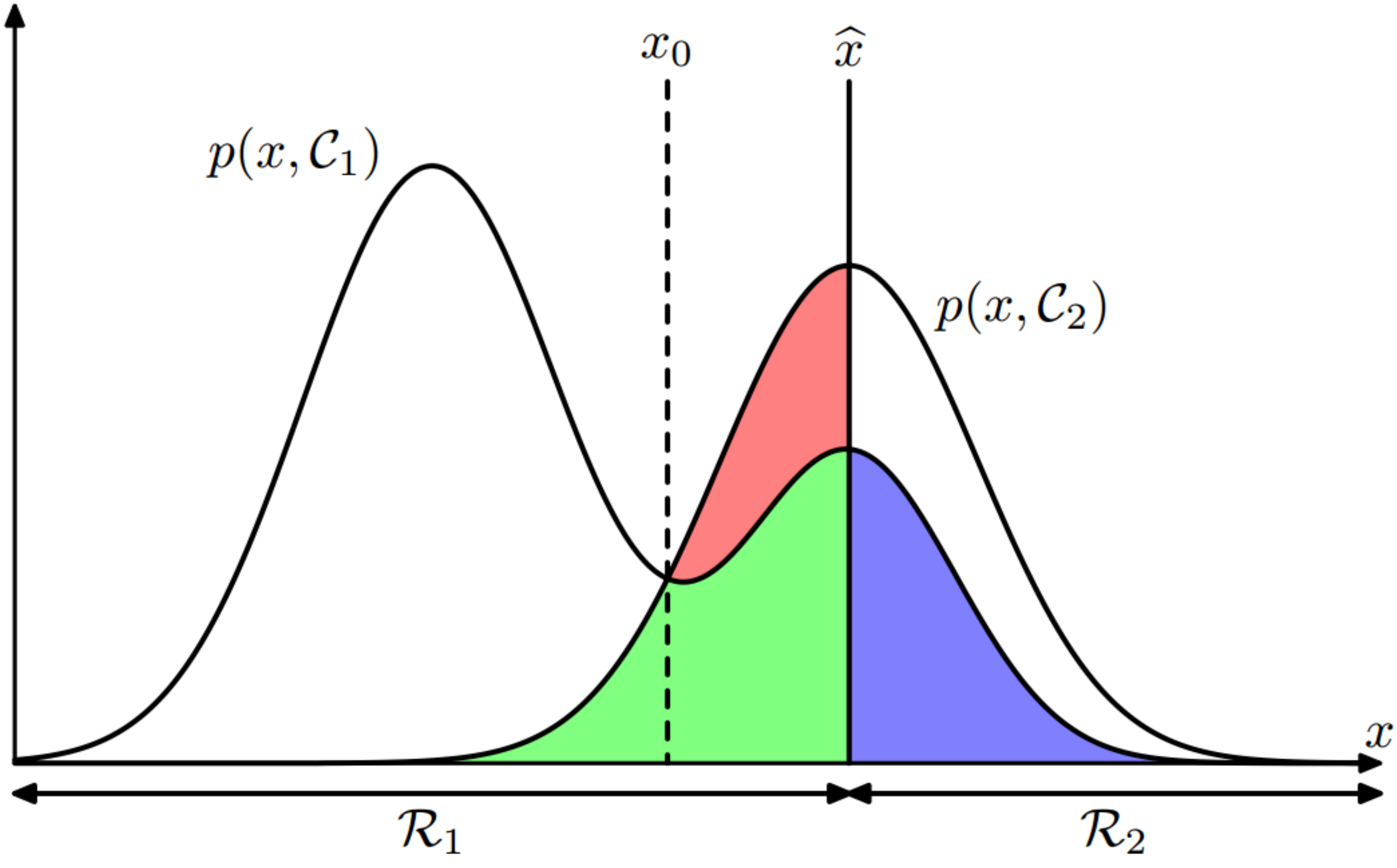

```
def lda(data):
    n = len(data[0]) - 1
    m1 = [0 for x in range(n)]
    m2 = [0 for x in range(n)]
    m = [0 for x in range(n)]
    number1 = 0
    number2 = 0
    for d in data:
        if d[n] == 1:
            add(m1, d)
            number1 += 1
        elif d[n] == 2:
            add(m2, d)
            number2 += 1
    divide(m1, number1)
    divide(m2, number2)
    print m1,m2

    sw = [[] for x in range(n)]
    for i in range(n):
        sw[i] = [0 for x in range(n)]
    calc_sw(data, sw, m1, 1)
    calc_sw(data, sw, m2, 2)
    normal_matrix(sw)
    print "Sw矩阵：", sw
    r = linalg.inv(sw)
    print "逆矩阵：", r
    diff(m1, m2, m)
    normal_vector(m)
    m = multiply(r, m)
    normal_vector(m)
    return m
```

LDA与分类 $\vec{w} \propto S_W^{-1}(\vec{m}_2 - \vec{m}_1)$

- 线性判别分析(Fisher's linear discriminant)
  - 严格的说，它只是给出了数据的特定投影方向
- 投影后，数据可以方便的找到阈值w_0，$y \geq w_0$时为C_1类，否则为C_2类。
  - 思考：一维数据下，可以如何分类？

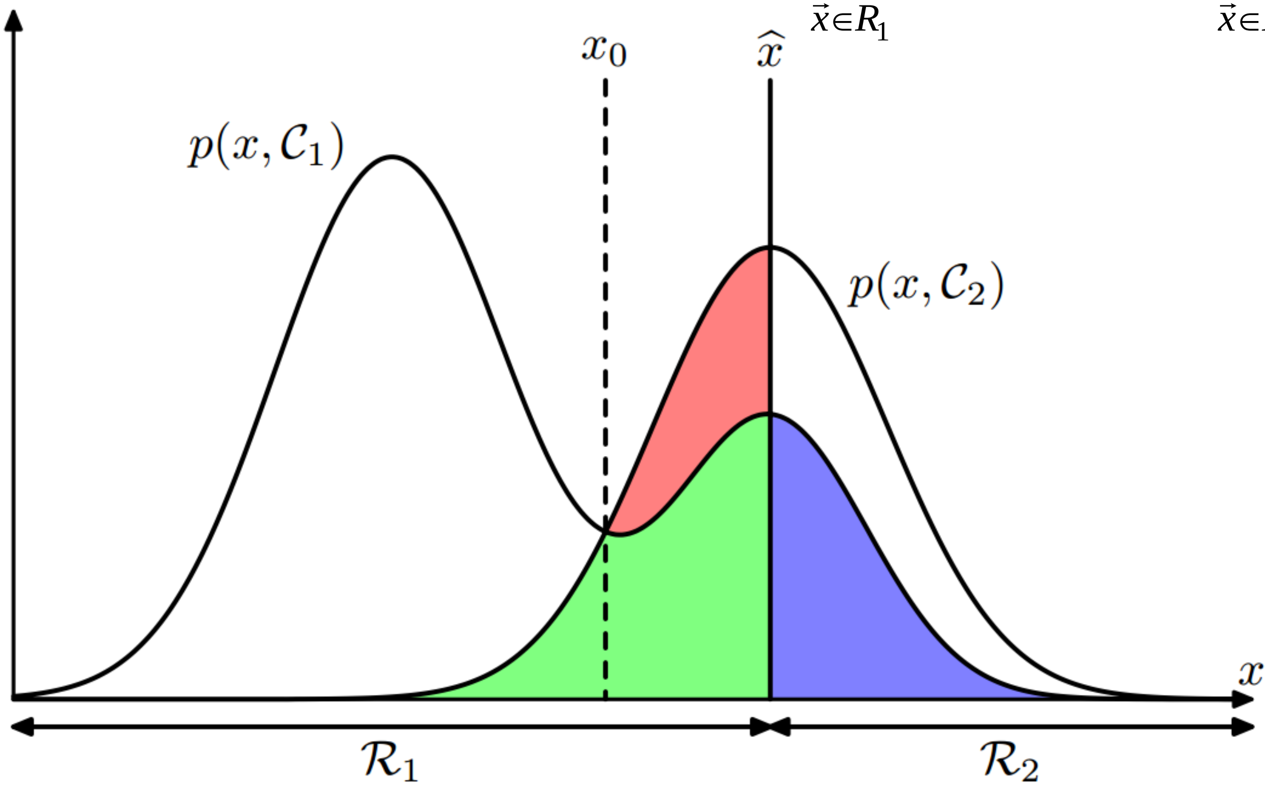


分类step1：极大似然估计

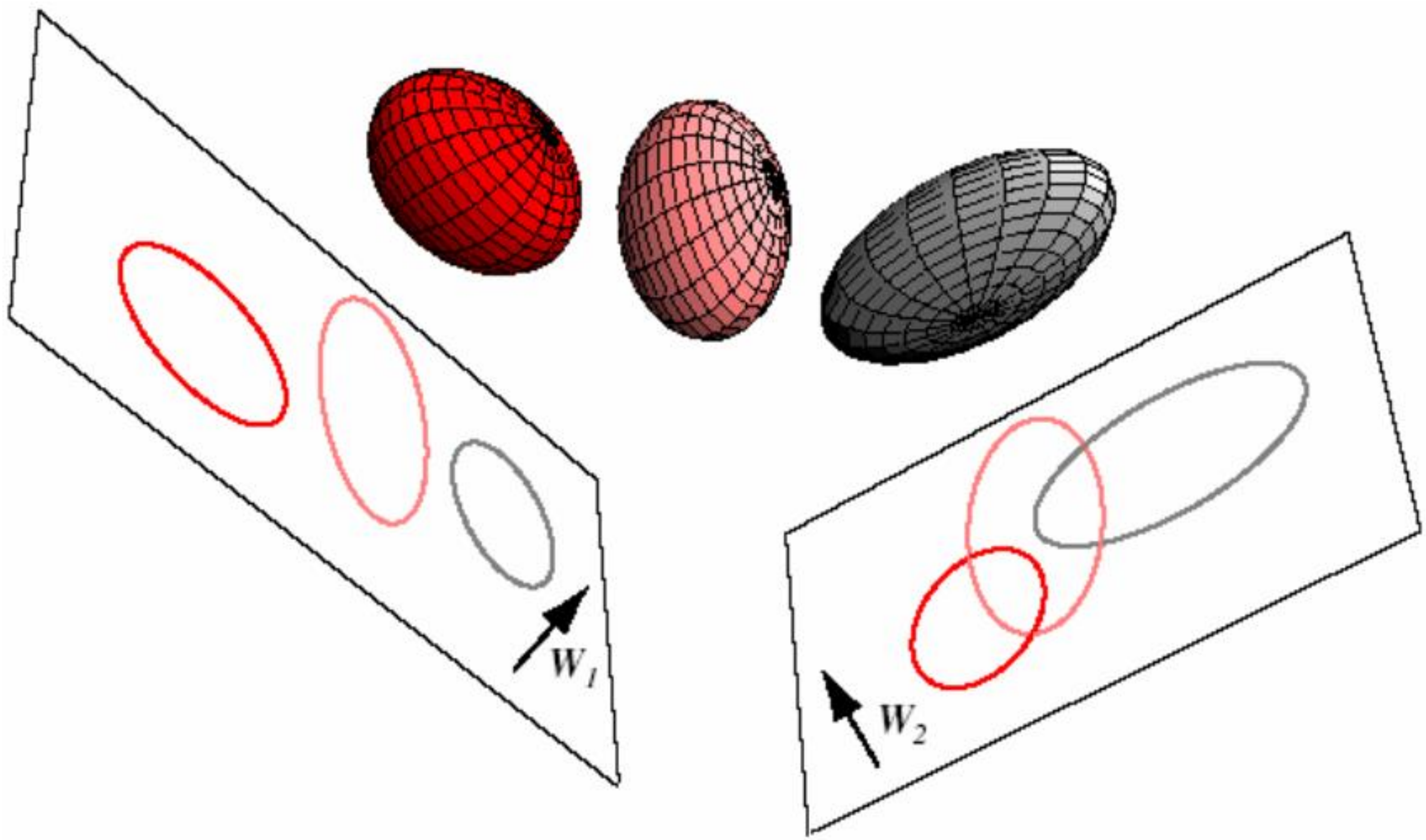


分类step2：误判率准则

$$p(error) = p(\vec{x} \in R_1, C_2) + p(\vec{x} \in R_2, C_1) = \int_{\vec{x} \in R_1} p(\vec{x}, C_2) d\vec{x} + \int_{\vec{x} \in R_2} p(\vec{x}, C_1) d\vec{x}$$

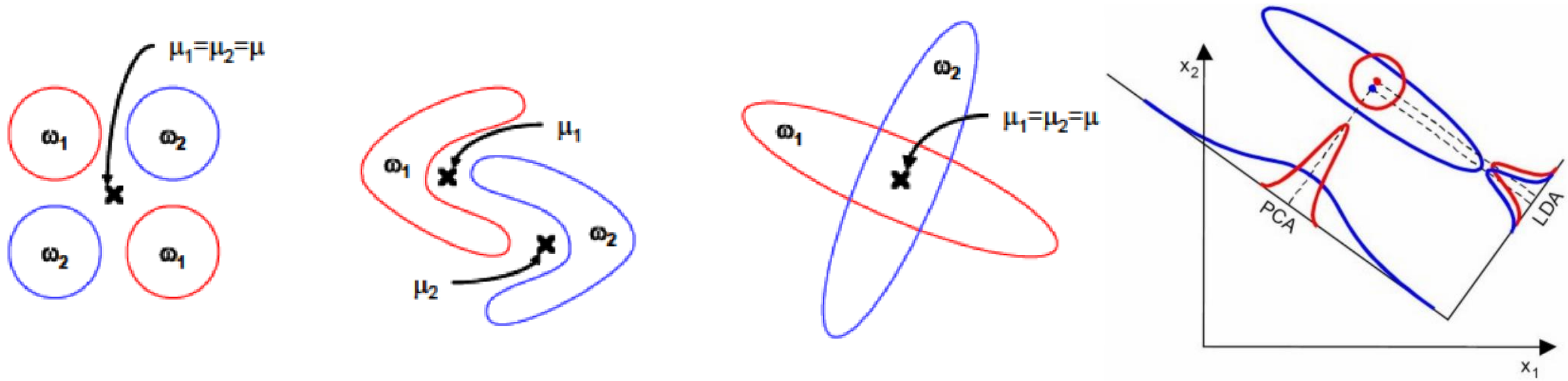


使用LDA将样本投影到平面上



LDA特点

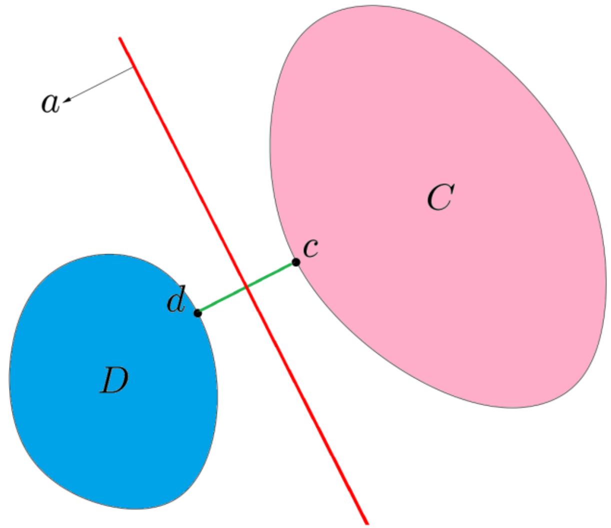
- 由LDA的计算公式看出，LDA是强依赖均值的。如果类别之间的均值相差不大或者需要方差等高阶矩来分类，效果一般。
- 若均值无法有效代表概率分布，LDA效果一般。
  - LDA适用于类别是高斯分布的分类。



LDA与线性回归的关系

- 假定正样本$(\mathbf{x},1)^{(i)}$个数为N_1，负样本$(\mathbf{x},-1)^{(i)}$个数为N_2；将标记加权成正样本$(\mathbf{x},1/N_1)^{(i)}$，负样本$(\mathbf{x},-1/N_2)^{(i)}$，则使用线性回归得到的决策面方向与LDA相同。

$$\begin{cases} (x,1)^{(i)} \Rightarrow \left(x,\dfrac{1}{N_1}\right)^{(i)} \\ (x,-1)^{(i)} \Rightarrow \left(x,-\dfrac{1}{N_2}\right)^{(i)} \end{cases}$$



LDA与PCA

- LDA：
  - 分类性能最好的方向
- PCA：
  - 样本点投影具有最大方差的方向