矩阵分解与特征降维 C06

2020/10/30 胡俊峰

北京大学信息科学技术学院

主要内容

- HITS算法
- Pandas表的聚集与关联运算
- 特征分析与降维
- 数据可视化

Data Aggregations on Multi-Indices

- Pandas has built-in data aggregation methods,
- such as mean(), sum(), and max().
- For hierarchically indexed data, these can be passed a level parameter that controls which **subset of the data the aggregate is computed on**.

- Group by certain Key

```
data_mean = health_data.mean(level='year')
data_mean
```

subject	Bob		Guido		Sue	
type	HR	Temp	HR	Temp	HR	Temp
year						
2013	37.5	38.2	41.0	35.85	32.0	36.95
2014	38.5	37.6	43.5	37.55	56.0	36.70

```
health_data
```

	subject	Bob		Guido		Sue	
	type	HR	Temp	HR	Temp	HR	Temp
year	visit						
2013	**1**	31.0	38.7	32.0	36.7	35.0	37.2
	2	44.0	37.7	50.0	35.0	29.0	36.7
2014	**1**	30.0	37.4	39.0	37.8	61.0	36.9
	2	47.0	37.8	48.0	37.3	51.0	36.5

Combining Datasets: Merge and Join

- *one-to-one*
- *many-to-one*
- *many-to-many* joins.

Combining Datasets: Concat and Append

```
ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])
ser2 = pd.Series(['D', 'E', 'F'], index=[4, 5, 6])
pd.concat([ser1, ser2])
```

```
1    A
2    B
3    C
4    D
5    E
6    F
dtype: object
```

```
df1 = make_df('AB', [1, 2])
df2 = make_df('AB', [3, 4])
display('df1', 'df2', 'pd.concat([df1, df2])')
```

df1

	A	B
1	A1	B1
2	A2	B2

df2

	A	B
3	A3	B3
4	A4	B4

pd.concat([df1, df2])

	A	B
1	A1	B1
2	A2	B2
3	A3	B3
4	A4	B4

One-to-one joins :

```
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})
display('df1', 'df2')
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df2

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

```
df3 = pd.merge(df1, df2)
df3
```

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

Many-to-one joins：

```
df4 = pd.DataFrame({'group': ['Accounting', 'Engineering', 'HR'],
                    'supervisor': ['Carly', 'Guido', 'Steve']})
display('df3', 'df4', 'pd.merge(df3, df4)')
```

df3

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

df4

	group	supervisor
0	Accounting	Carly
1	Engineering	Guido
2	HR	Steve

pd.merge(df3, df4)

	employee	group	hire_date	supervisor
0	Bob	Accounting	2008	Carly
1	Jake	Engineering	2012	Guido
2	Lisa	Engineering	2004	Guido
3	Sue	HR	2014	Steve

推荐算法与搜索排名

- PCA降维与特征空间
- HITS算法与搜索排名
- SVD与隐含语义挖掘
- 协同过滤算法

协方差矩阵的物理意义

- $C(p;q)=\frac{1}{N}\sum_{k=1}^{N}X_p^{(k)}X_q^{(k)}$

对角线$(p;p)$上的元素：第p维特征的方差

矩阵$(p;q)$元的大小反映了所有样本第p维和第q维数据的相关性（若不相关，则为0）

PCA（主成分分解）

—— 对于实对称矩阵，存在一组正交变换使其对角化

新特征的方差

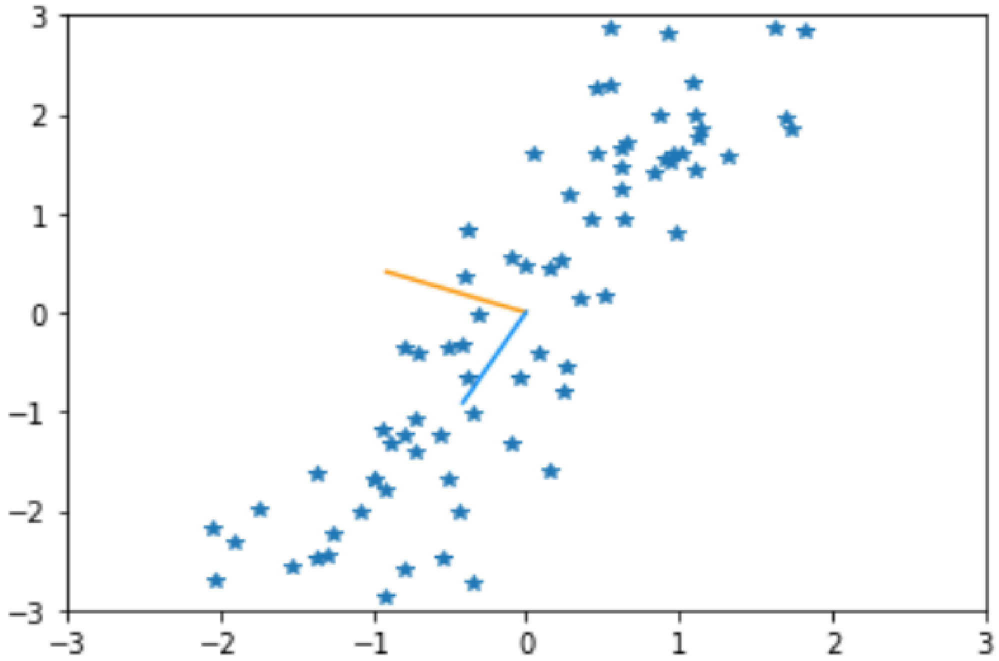
$$A = Q\Sigma Q^T = Q \begin{bmatrix} \lambda_1 & \cdots & \cdots & \cdots \\ \cdots & \lambda_2 & \cdots & \cdots \\ \cdots & \cdots & \ddots & \cdots \\ \cdots & \cdots & \cdots & \lambda_m \end{bmatrix} Q^T$$

新特征空间的基底

```python
eigVals, eigVecs = np.linalg.eig(X_centered.T.dot(X_centered))
eigVecs
```

```
array([[-0.91116273, -0.41204669],
       [ 0.41204669, -0.91116273]])
```

```python
orange = '#FF9A13'
blue = '#1190FF'
plt.plot([0, eigVecs[0][0]], [0, eigVecs[1][0]], orange)
plt.plot([0, eigVecs[0][1]], [0, eigVecs[1][1]], blue)
plt.plot(X_centered[:,0], X_centered[:,1], '*')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.show()
```



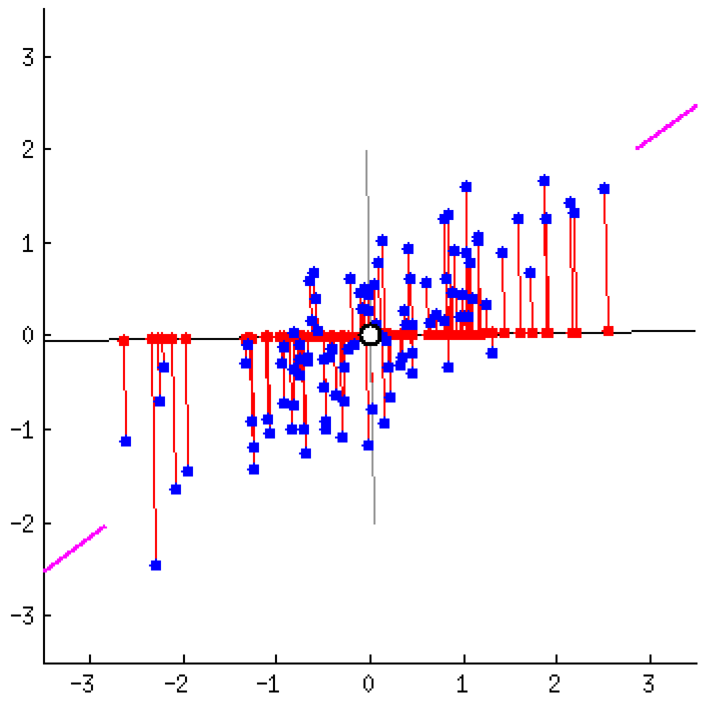
PCA-主成分分析

- 对原始样本进行（线性变换）基变换可以对原始样本给出不同的表示
- 基的维度小于数据的维度可以起到降维的效果
- 对基变换后新的样本求其方差，选取使其方差最大的基
- 新的基之间的协方差要为0

$$X = \begin{pmatrix} a_1 & a_2 & \dots & a_m \\ b_1 & b_2 & \dots & b_m \end{pmatrix}$$

协方差矩阵

$$\frac{1}{m}XX^\top = \begin{pmatrix} \frac{1}{m}\sum_{i=1}^{m} a_i^2 & \frac{1}{m}\sum_{i=1}^{m} a_i b_i \\ \frac{1}{m}\sum_{i=1}^{m} a_i b_i & \frac{1}{m}\sum_{i=1}^{m} b_i^2 \end{pmatrix}$$



目标：基变换后的新样本的协方差矩阵是对角化的

PCA-主成分分析

X：原始数据，每一列为一个数据
P：是一组基按行组成的矩阵
Y：是X对P做基变换的后的新数据
C：原始数据X的协方差矩阵
D：Y的协方差矩阵

$$
\begin{aligned}
D &= \frac{1}{m}YY^\top \\
&= \frac{1}{m}(PX)(PX)^\top \\
&= \frac{1}{m}PXX^\top P^\top \\
&= P(\frac{1}{m}XX^\top)P^\top \\
&= PCP^\top \\
&= P\begin{pmatrix} \frac{1}{m}\sum_{i=1}^m a_i^2 & \frac{1}{m}\sum_{i=1}^m a_i b_i \\ \frac{1}{m}\sum_{i=1}^m a_i b_i & \frac{1}{m}\sum_{i=1}^m b_i^2 \end{pmatrix} P^\top
\end{aligned}
$$

我们找的P，正好是让原协方差矩阵对角化的P

于是，只需对协方差矩阵C进行特征分解，对求得的特征值进行排序，再对特征向量取前K列组成的矩阵乘以原始数据矩阵X，就得到了我们需要的降维后的数据矩阵Y。

PCA数据特征降维

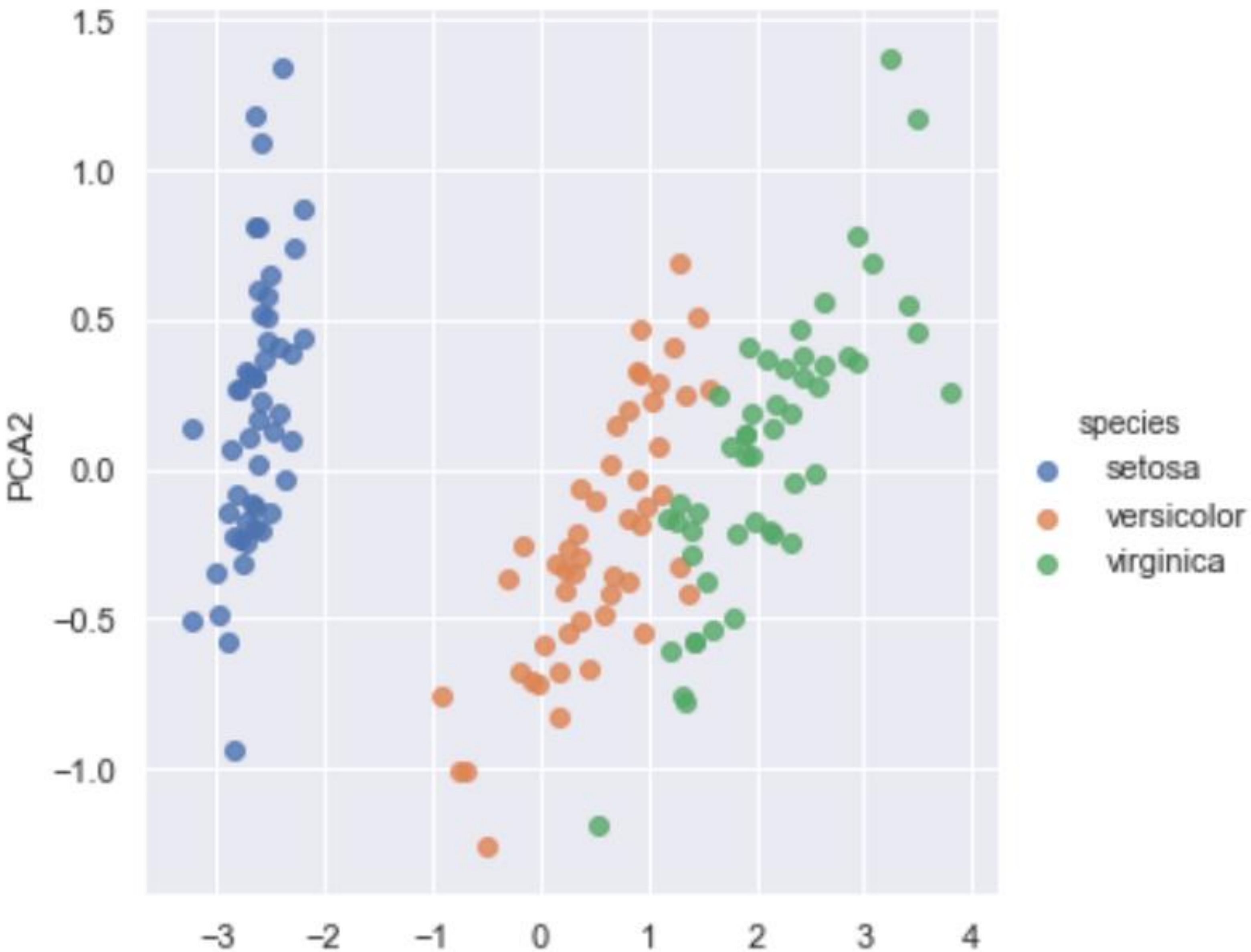
In [78]:

```python
from sklearn.decomposition import PCA  # 1. Choose the model class
model = PCA(n_components=2)            # 2. Instantiate the model with hyperp
model.fit(X_iris)                      # 3. Fit to data. Notice y is not spec
X_2D = model.transform(X_iris)         # 4. Transform the data to two dimensi
X_2D
```

Out[78]:

```
array([[-2.68412563,  0.31939725],
       [-2.71414169, -0.17700123],
       [-2.88899057, -0.14494943],
       [-2.74534286, -0.31829898],
       [-2.72871654,  0.32675451],
       [-2.28085963,  0.74133045],
       [-2.82053775, -0.08946138],
       [-2.62614497,  0.16338496],
       [-2.88638273, -0.57831175],
       [-2.6727558 , -0.11377425],
       [-2.50694709,  0.6450689 ],
       [-2.61275523,  0.01472994],
```

```
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]
sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False)
```



主成分分析与聚类

```
import pandas as pd
```

```
train = pd.read_csv('./python_course/train.csv')
print(train.shape)
```

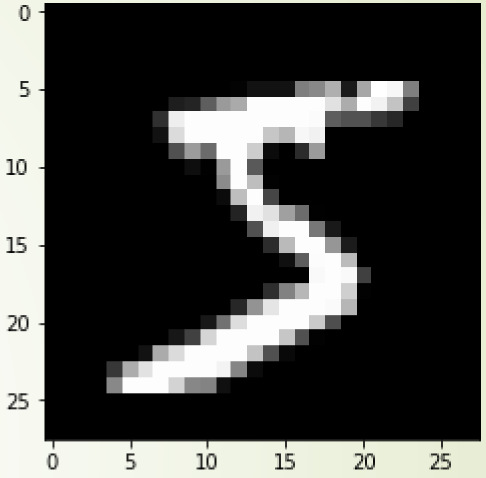
(42000, 785)

```
target = train['label']
train = train.drop("label", axis=1)
X= train[:6000].values
Target = target[:6000]
print(X.shape)
```

提取出data, label

(6000, 784)

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8
0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0



PCA — 手写数字识别

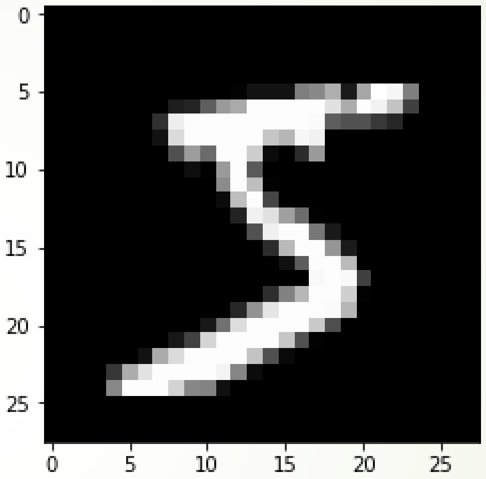
标准化数据，保证每个维度的数据方差是1，均值为0，使得预测结果不会被某些数值大的值主导，保证‘公平’

计算特征值和特征向量

从特征值由大到小排列

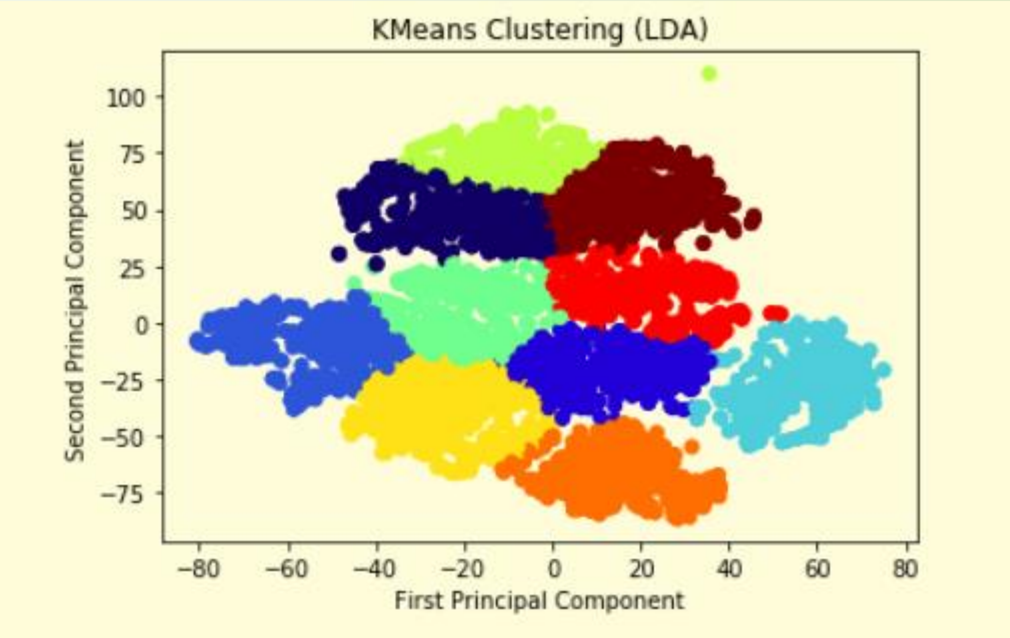
计算每个方差所占能量的比例

聚类与识别



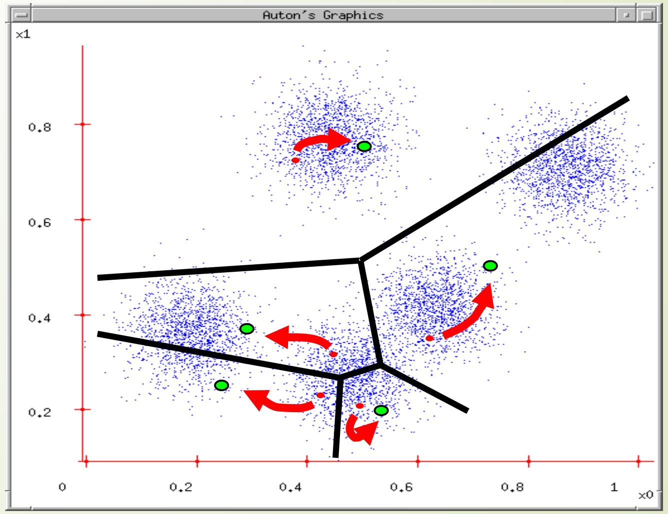
5

```
from sklearn.cluster import KMeans # KMeans cluste

kmeans = KMeans(n_clusters=10)

X_clustered = kmeans.fit_predict(tsne_results)

fig = plt.figure()
ax1 = fig.add_subplot(111)
#设置标题
ax1.set_title('KMeans Clustering (LDA)')
#设置X轴标签
plt.xlabel('First Principal Component')
#设置Y轴标签
plt.ylabel('Second Principal Component')
#画散点图
ax1.scatter(tsne_results[:,0],tsne_results[:,1],c = X_clustered, cmap='jet', marker = 'o')
#显示所画的图
plt.show()
```



K-means 硬聚类分析

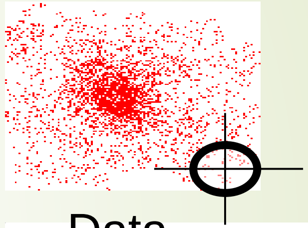
- 类质心按分布密度的贪心求解法
- 局部最优解



距离加权方案与核函数 (kernel)

$$P(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K(\mathbf{x}-\mathbf{x}_i)$$

A function of some finite number of data points $x_1 \ldots x_n$

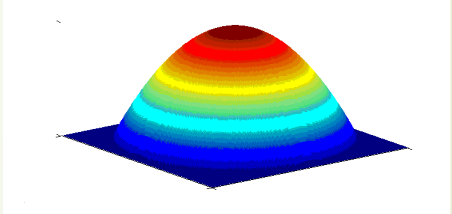


Data

Examples:

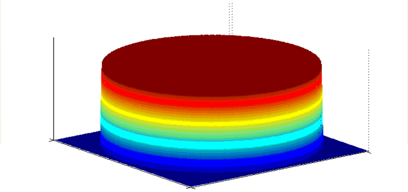
- Epanechnikov Kernel

$$K_E(\mathbf{x}) = \begin{cases} c\left(1-\|\mathbf{x}\|^2\right) & \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$



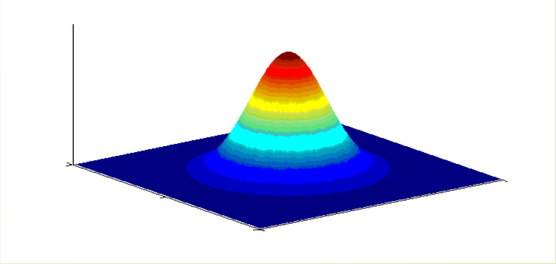
- Uniform Kernel

$$K_U(\mathbf{x}) = \begin{cases} c & \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$



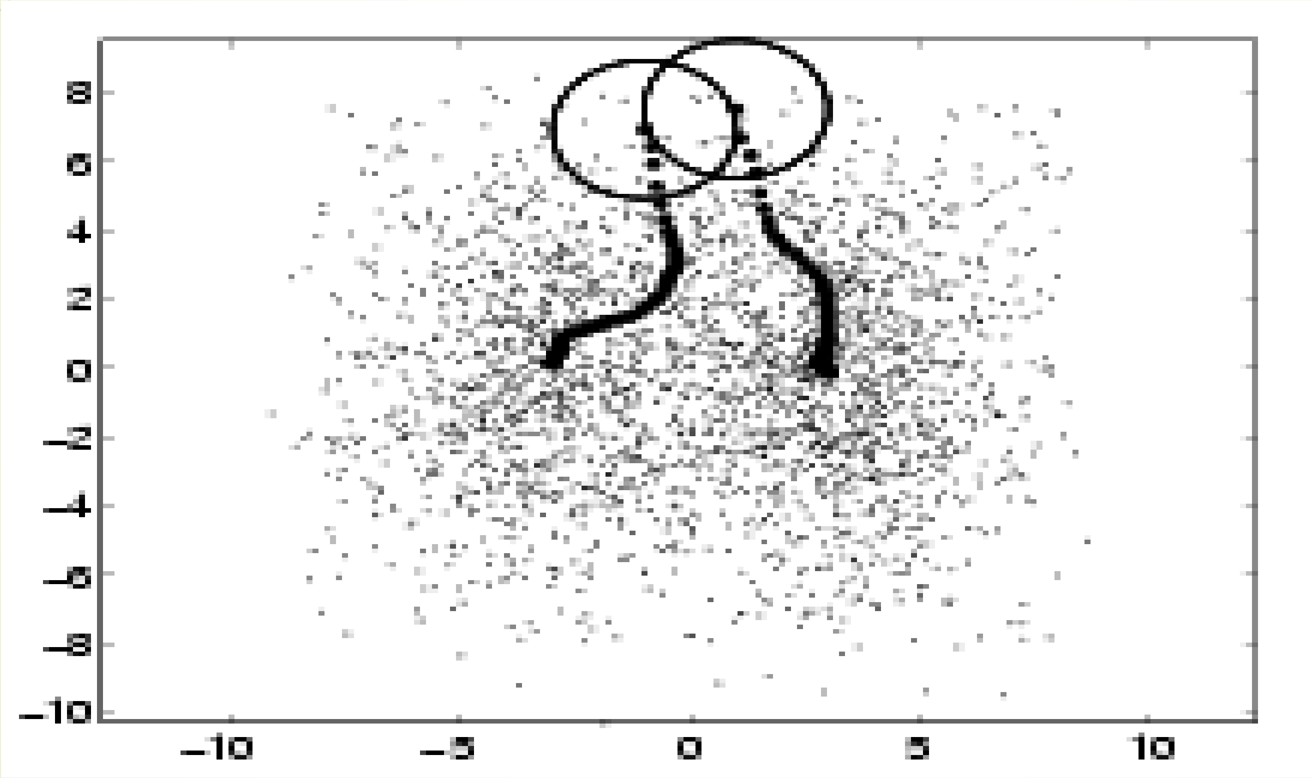
- Normal Kernel

$$K_N(\mathbf{x}) = c \cdot \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right)$$



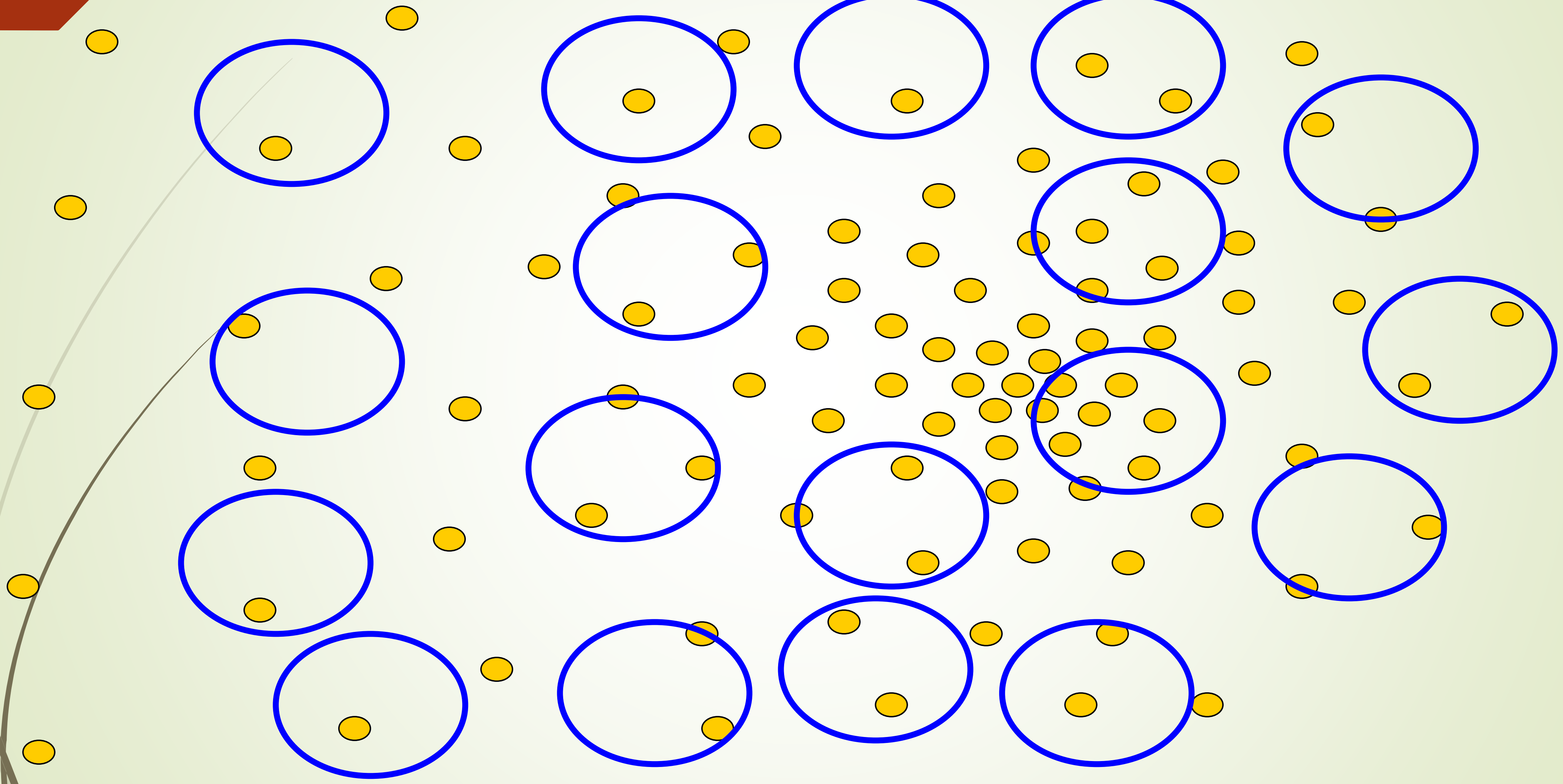
在核函数约束下呈现的按梯度贪心的收敛轨迹

mean shift trajectories



Window tracks signify the steepest ascent directions

多源并发的mean-shift



Tessellate the space with windows

Run the procedure in parallel

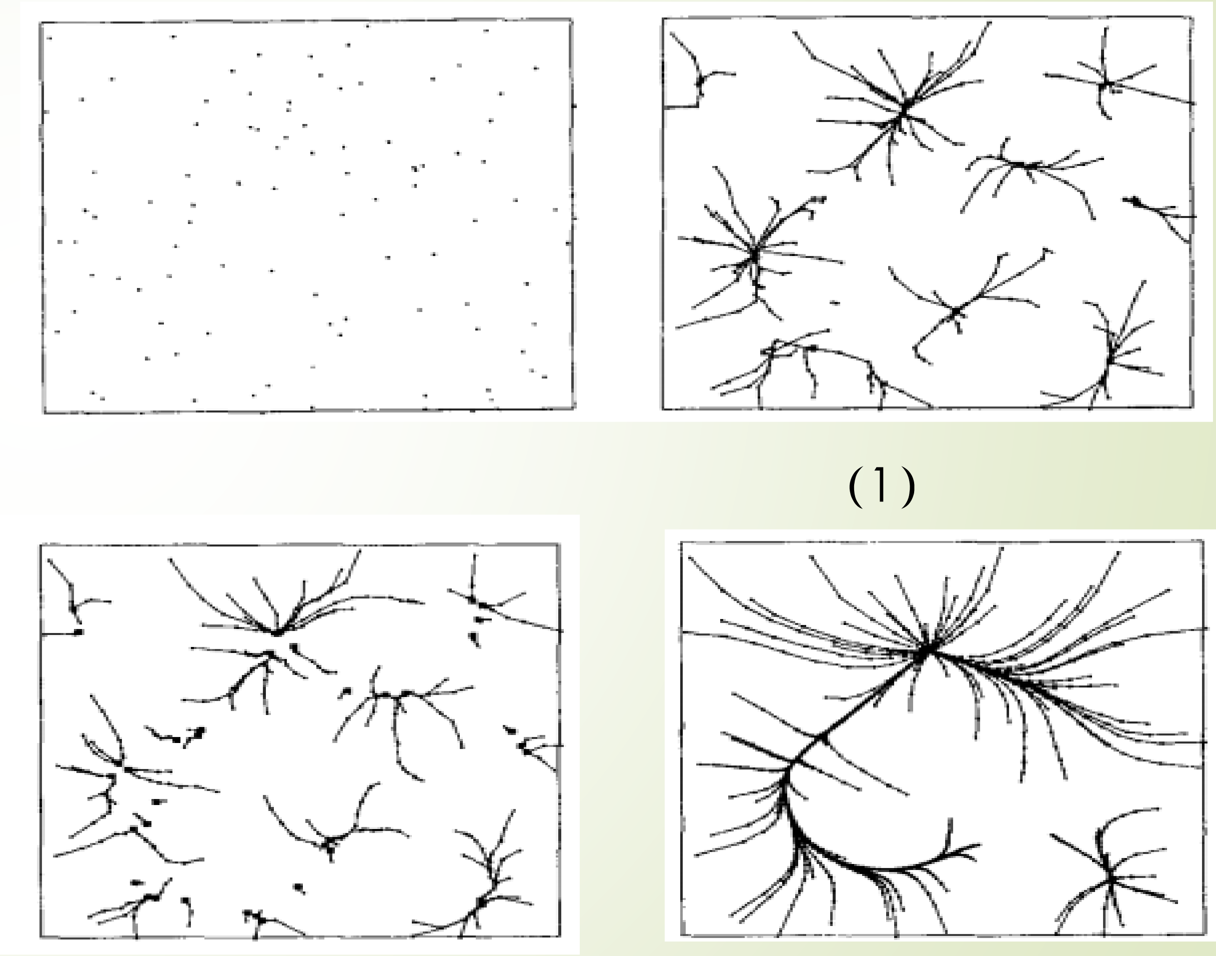
Mean shift trajectories with different kernal

- Same sample space

① Uniform Kernel

② truncated Normal Kernel

③ nontruncated Normal Kernel



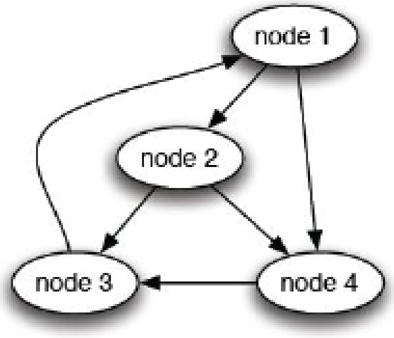
(1)

(2)

(3)

HITS算法(Hyperlink-induced Topic Search)

- 首先，通过关键词在文档中是否出现，得到相关网页集合及其链接关系。
- 每个网页有两个值：a(权威值，入度高）及h（hub，中枢值，出度高）
- 通过反复迭代计算，得出得分最高的网页。



$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 6 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 9 \\ 7 \\ 2 \\ 4 \end{bmatrix}$$

定义n个网页之间链接关系的邻接矩阵为M,即M_{ij}为1$\Longleftrightarrow$从网页i到j有链接，则网页i的中枢值为：

$$a_i \longleftarrow M_{1i}h_1 + M_{2i}h_2 + \ldots M_{ni}h_n \tag{3}$$

$$h_i \longleftarrow M_{i1}a_1 + M_{i2}a_2 + \ldots M_{in}a_n \tag{4}$$

$h^k=(h_1, h_2, \ldots h_n)^T$,$a^k=(a_1, a_2, \ldots a_n)^T$为运行了k次的时候，n个网页的中枢，权威向量，则转换规则为：（先更新a^k)

$$a^k = M^T h^{k-1} \tag{5}$$

$$h^k = M a^k \tag{6}$$

h^0, a^0的元素都为1 迭代可得：$a^1 = M^T h^0, h^1 = M M^T h^0, \ldots$

$$a^k = (M^T M)^{k-1} h^0 \tag{7}$$

$$h^k = (M M^T)^k h^0 \tag{8}$$

定理：n*n的实对称矩阵有n个特征值，且不同特征值的特征向量彼此正交（即特征向量构成线性空间的一组基底）

设$(M^TM)^{k-1}$的特征值为$c_1,c_2,\ldots c_n$,且$c_1>c_2>,\ldots>c_n$,对应的特征向量为$z_1,z_2,\ldots z_n$,而h^0在基底下的表示为

$$h_0 = q_1z_1 + q_2z_2 + \ldots + q_nz_n \tag{9}$$

则

$$\begin{aligned} h^k &= (MM^T)^k h^0 && (10) \\ &= (MM^T)^k (q_1z_1 + q_2z_2 + \ldots q_nz_n) && (11) \\ &= q_1(MM^T)^k z_1 + q_2(MM^T)^k z_2 + \ldots + q_n(MM^T)^k z_n && (12) \\ &= q_1c_1^k z_1 + q_2c_2^k z_2 + \ldots q_nc_n^k z_n && (13) \end{aligned}$$

如果要收敛，需要对每项正规化。则

$$
\begin{aligned}
h^k &= = \frac{(MM^T)^k h^0}{\|(MM^T)^k h^0\|} && (14)\\
&= \frac{q_1 c_1^k z_1 + q_2 c_2^k z_2 + \dots q_n c_n^k z_n}{\|q_1 c_1^k z_1 + q_2 c_2^k z_2 + \dots q_n c_n^k z_n\|} && (15)\\
&= \frac{q_1 z_1 + q_2(\frac{c_2}{c_1})^k z_2 + \dots q_n(\frac{c_2}{c_1})^k z_n}{\|q_1 z_1 + q_2(\frac{c_2}{c_1})^k z_2 + \dots q_n(\frac{c_2}{c_1})^k z_n\|} && (16)\\
&= \frac{q_1 z_1}{\|q_1 z_1\|} && (17)
\end{aligned}
$$

故h_k收敛，同理可得a_k收敛

奇异值分解

- $N \times M$矩阵X，把每一行当成一个点。设$r = rank(X)$。
- $\vec{v}_1 = argmax_{|\vec{v}|=1} |X\vec{v}|$
- $\vec{v}_2 = argmax_{|\vec{v}|=1, \vec{v}\perp\vec{v}_1} |X\vec{v}|$
- ……
- $\vec{v}_r = argmax_{|\vec{v}|=1, \vec{v}\perp\vec{v}_1, \vec{v}\perp\vec{v}_2 \ldots, \vec{v}\perp\vec{v}_{r-1}} |X\vec{v}|$

- 令$\sigma_i = |X\vec{v}_i|, \overrightarrow{u_i} = \frac{1}{\sigma_i} X\vec{v}_i, 1 \leq i \leq r$
- 则$X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \ldots & \vec{u}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} -\vec{v}_1^T - \\ \ldots \\ -\vec{v}_r^T \end{bmatrix} = UDV^T$

奇异值分解

- $N \times M$矩阵X，把每一行当成一个点。设$r = rank(X)$。
- $X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \dots & \vec{u}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} -\vec{v}_1^T - \\ \dots \\ -\vec{v}_r^T \end{bmatrix} = UDV^T$
- U, D, V分别为$N \times r, r \times r, M \times r$矩阵
- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$称为奇异值

numpy.linalg.svd

numpy.linalg.svd(*a*, *full_matrices=True*, *compute_uv=True*) [source]

Singular Value Decomposition.

When *a* is a 2D array, it is factorized as `u @ np.diag(s) @ vh = (u * s) @ vh`, where *u* and *vh* are 2D unitary arrays and *s* is a 1D array of *a*'s singular values. When *a* is higher-dimensional, SVD is applied in stacked mode as explained below.

Parameters:

a : *(..., M, N) array_like*

A real or complex array with `a.ndim >= 2`.

full_matrices : *bool, optional*

If True (default), *u* and *vh* have the shapes `(..., M, M)` and `(..., N, N)`, respectively. Otherwise, the shapes are `(..., M, K)` and `(..., K, N)`, respectively, where `K = min(M, N)`.

compute_uv : *bool, optional*

Whether or not to compute *u* and *vh* in addition to *s*. True by default.

Returns:

u : *{ (..., M, M), (..., M, K) } array*

Unitary array(s). The first `a.ndim - 2` dimensions have the same size as those of the input *a*. The size of the last two dimensions depends on the value of *full_matrices*. Only returned when *compute_uv* is True.

s : *(..., K) array*

Vector(s) with the singular values, within each vector sorted in descending order. The first `a.ndim - 2` dimensions have the same size as those of the input *a*.

奇异值分解与主成分分析

- $X = UDV^T$
- 考虑X的变换$Y = XV = UD$(维度重组)

- 则$C_Y = \frac{1}{N}Y^TY = \frac{1}{N} = \frac{1}{N}D^TU^TUD = \frac{1}{N}D^2$是个对角矩阵
- 各个维度之间不相关
- 对角元的大小——这一维自身的方差
- 按方差大小排列，得到第1,2,…, r个主成分
- 方差小的几个主成分可以认为是噪声，将其丢弃——降维