深蓝学院
shenlanxueyuan.com

点云作业第三讲——聚类算法

主讲人 王永浩

算法概述

●K-Means聚类

- N data points are independent, so we can optimize for each n separately.
- Simply assign the n^{th} data point to the closest cluster center, which will minimize $||x_n - \mu_k||^2$
- Formally

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

E-Step

- With r_{nk} fixed, the objective function J is a quadratic function of μ_k
- Compute its first order derivative and make it to 0
- Consider each center μ_k separately

$$2\sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k) = 0$$

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}}$$

M-Step

算法概述

●K-Means聚类

两种初始化方法

◆ Random方法：随机选取k个中心点

◆ K-Means++：

a) 从输入的数据点集合中随机选择一个点作为第一个聚类中心μ1

b) 对于数据集中的每一个点xi，计算它与已选择的聚类中心中最近聚类中心的距离$D(x_i) = argmin||x_i - \mu_r||_2^2, r = 1,2 \dots \dots \mathrm{k}_{\mathrm{selected}}$

c) 选择一个新的数据点作为新的聚类中心，选择的原则是：D(x)较大的点，被选取作为聚类中心的概率较大

d) 重复b和c直到选择出k个聚类质心

e) 利用这k个质心来作为初始化质心去运行标准的K-Means算法

算法概述

●K-Means聚类

```
if 'random' == self.init:
    init_index = np.random.choice(a=data.shape[0],size=self.k_,replace=False)
elif 'k_means++' == self.init:
    init_index = []
    init_index.append(np.random.choice(a=data.shape[0],size=1,replace=False)[0])
    for i in range(self.k_-1):
        init_points = data[init_index,:]
        tmpindex = np.argmax(self._dismatrix(data,init_points))
        x_index = int(tmpindex/data.shape[1])
        init_index.append(x_index)
```

算法概述

●K-Means聚类

```
for iteration in range(self.max_iter_):
    # E-STEP
    dis_matrix = self._dismatrix(data,cluster_center_last)
    label_matrix = np.argmin(dis_matrix,axis=1)
    # M-STEP
    for lb_index in range(self.k_):
        lb_points = data[label_matrix == lb_index]
        cluster_center_now[lb_index,:] = np.mean(lb_points,axis=0)
    # determine whether to stop or not
    if np.linalg.norm(cluster_center_now - cluster_center_last) < self.tolerance_:
        break
    cluster_center_last = cluster_center_now.copy()
    # plt.scatter(cluster_center_now[:,0],cluster_center_now[:,1],s=50,marker='D')
self.cluster_center = cluster_center_now
# return label_matrix
```

算法概述

●GMM模型

1. Initialize the means μ_k, covariances Σ_k and weights π_k

2. E-step. Evaluate the posterior $p(z_{nk} = 1|x_n)$, intuitively this is the probability of x_n being assigned to each of the K clusters.

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

算法概述

●GMM模型

3. M-Step. Estimate the parameters using MLE.

4. Evaluate the log likelihood, if converges, stop. Otherwise go back to E-step

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

$$\begin{aligned}
\boldsymbol{\mu}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \\
\boldsymbol{\Sigma}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \left(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}\right) \left(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}\right)^{\mathrm{T}} \\
\pi_k^{\text{new}} &= \frac{N_k}{N}
\end{aligned}$$

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

算法概述

深蓝学院 shenlanxueyuan.com

●GMM模型

GMM模型需要估计的参数都是多维数组，所以可以用向量化编程的方式加速

```
for iteration in range(self.max_iter):
    # E-step
    N = self.Gaussian(data,self.mean,self.var) # k,n
    nowlikelihood = -np.sum(np.log(np.sum(self.weight*N,axis=0)))
    if lastlikelihood - nowlikelihood < self.tol:
        break
    lastlikelihood = nowlikelihood
    gamma_znk = self.weight*N
    gamma_znk = gamma_znk/np.sum(gamma_znk,axis=0,keepdims=True) # k,n
    # M-step
    N_k = np.sum(gamma_znk,axis=1,keepdims=True) # k,1
    self.mean = (1/N_k * np.matmul(gamma_znk,data))[:,np.newaxis,:] # k,1,2

    tmpdata = (data - self.mean)[:,:,:,np.newaxis] # k,n,2,1
    tmpdata = np.matmul(tmpdata,np.transpose(tmpdata,(0,1,3,2))) # k,n,2,2
    self.var = (1/N_k)[:,:,np.newaxis] * np.sum(gamma_znk[:,:,np.newaxis,np.newaxis] * tmpdata,axis=1) # k,2,2
    self.weight = N_k/data.shape[0]
```

算法概述

- 谱聚类

1. Build the graph to get adjacency matrix $W \in \mathbb{R}^{n \times n}$
2. Compute unnormalized Laplacian L
3. Compute the first (smallest) k eigenvectors $v_1, \cdots, v_k$ of L
4. Let $V \in \mathbb{R}^{n \times k}$ be the matrix contraining the vectors $v_1, \cdots, v_k$ as columns
5. For $i = 1, \cdots n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of V
6. Cluster the points $\{y_i \in \mathbb{R}^k\}$ with k-means algorithm into clusters $C_1, \cdots, C_k$
7. The final output clusters are $A_1, \cdots, A_k$ where $A_i = \{j | y_j \in C_i\}$

Unnormalized Spectral Clustering

算法概述

- 谱聚类

1. Build the graph to get adjacency matrix $W \in \mathbb{R}^{n \times n}$
2. Compute normalized Laplacian $L' = L_{rw}$
3. Compute the first (smallest) k eigenvectors $v_1, \cdots, v_k$ of L'
4. Let $V \in \mathbb{R}^{n \times k}$ be the matrix contraining the vectors $v_1, \cdots, v_k$ as columns
5. For $i = 1, \cdots n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of V
6. Cluster the points $\{y_i \in \mathbb{R}^k\}$ with k-means algorithm into clusters $C_1, \cdots, C_k$
7. The final output clusters are $A_1, \cdots, A_k$ where $A_i = \{j | y_j \in C_i\}$

Normalized Spectral Clustering

算法概述

- 谱聚类

1. Build the graph to get adjacency matrix $W \in \mathbb{R}^{n \times n}$
2. Compute normalized Laplacian $L' = L_{rw}$
3. Compute the first (smallest) k eigenvectors $v_1, \cdots, v_k$ of L'
4. Let $V \in \mathbb{R}^{n \times k}$ be the matrix contraining the vectors $v_1, \cdots, v_k$ as columns
5. For $i = 1, \cdots n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of V
6. Cluster the points $\{y_i \in \mathbb{R}^k\}$ with k-means algorithm into clusters $C_1, \cdots, C_k$
7. The final output clusters are $A_1, \cdots, A_k$ where $A_i = \{j | y_j \in C_i\}$

Normalized Spectral Clustering

算法概述

- 谱聚类

  有多种可变的参数：
  近邻选择有多种方式：KNN/ Radius/fully connected
  距离函数的选择
  需要试验多种参数组合，以达到最好的效果

算法概述

深蓝学院 shenlanxueyuan.com

- 谱聚类

```
 D = np.diag(W.sum(axis=1))

if self.normalized_:
    L = a = np.matmul(LA.inv(D), L)
    L = b = np.identity(m) - np.matmul(LA.inv(D), W)
    assert(np.allclose(a,b))

eigvals, eigvecs = LA.eig(L)
"""
From numpy.linalg.eig's doc:
The eigenvalues are not necessarily ordered!!
so we need to sort eigen values!!
"""
sorted_idx = np.argsort(eigvals)
# smallest self.k_ eigenvectors
V = eigvecs[:, sorted_idx[:self.k_]]

# for debugging
self.eigvals = eigvals
self.eigvecs = eigvecs
self.V = V

# run kmeans
self.labels_ = KMeans(n_clusters=self.k_).fit_predict(V)
```

By 林超

算法概述

- 谱聚类

```
def fit(self, data):

    N, _ = data.shape
    # @create affinity matrix - knn for conectivity
    A = pairwise_distances(data)
    gamma = np.var(A)/4
    A = np.exp(-A**2/(2*gamma**2))
    # @get laplacian matrix
    L = csgraph.laplacian(A, normed=True)
    # @spetral decomposition
    eigval, eigvec = np.linalg.eig(L)
    # @get features
    idx_k_smallest = np.where(eigval < np.partition(eigval, self.__K)[self.__K])
    features = np.hstack([eigvec[:, i] for i in idx_k_smallest])
    # @cluster using kmeans++
    k_means = KMeans(init='k-means++', n_clusters=self.__K, tol=1e-6)
    k_means.fit(features)
    # @get cluster ids
    self.__labels = k_means.labels_
```

By Abel

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening