机器学习 第一次课程作业

UNIkeEN

2024 年 4 月 7 日

问题解答

1. After initializing the center parameters $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$, the k-mean algorithm is to repeat the following two steps until convergence:

   1. Assign the points to the nearest μ_k;
   2. Update μ_k to be the mean of the data points assigned to it.

   Prove that each of the above two steps will never increase the k-mean objective function,

$$J(\mu_1, \ldots, \mu_K) = \sum_{t=1}^{N} \sum_{k=1}^{K} r_{tk} \left\| x_t - \mu_k \right\|^2,$$

   where

$$r_{tk} = \begin{cases} 1, & \text{if } x_t \text{ is assigned to cluster } k; \\ 0, & \text{otherwise.} \end{cases}$$

解

分配步骤 在分配步骤中，每个数据点 x_t 被分配到最近的中心 μ_k。对于每个 x_t，如果其中心变化，旧、新的中心分别为 $\mu_{k_{\text{old}}}$、$\mu_{k_{\text{new}}}$。$r_{tk_{\text{new}}} = 1$，$r_{tk_{\text{old}}}` = 0$。

$$||x_t - \mu_{k_{\text{new}}}||^2 \leq ||x_t - \mu_{k_{\text{old}}}||^2 \tag{1}$$

由分配操作可知，分配到新中心的距离不会比原来的中心远。

$$J' = J - r_{tk_{\text{old}}}||x_t - \mu_{k_{\text{old}}}||^2 + r_{tk_{\text{new}}}||x_t - \mu_{k_{\text{new}}}||^2 \leq J \tag{2}$$

更新步骤 假设我们有一组数据点 $x_1, x_2, ..., x_m$ 属于 $\mathbb{R}^n$。我们想要找到一个点 μ 来最小化到所有点的平方距离之和，即最小化以下目标函数：

$$\Psi(\mu) = \sum_{i=1}^{m} ||x_i - \mu||^2 \tag{3}$$

$$\frac{\partial}{\partial \mu}\Psi(\mu) = \frac{\partial}{\partial \mu}\sum_{i=1}^{m}||x_i - \mu||^2 = \sum_{i=1}^{m}2(\mu - x_i) \tag{4}$$

使导数等于零，得到：

$$\sum_{i=1}^{m}2(\mu - x_i) = 0 \tag{5}$$

$$\Rightarrow 2m\mu - 2\sum_{i=1}^{m}x_i = 0 \tag{6}$$

$$\Rightarrow \mu = \frac{1}{m}\sum_{i=1}^{m}x_i \tag{7}$$

这表明 μ 是这组点的算术均值。

在更新步骤中，每个簇的中心 μ_k 被更新为分配给该簇的所有数据点的均值 μ'_k。由于 μ'_k 是最小化簇内所有点到中心的平方距离之和的点，对于簇 k，有：

$$\sum_{x_t \in C_k}||x_t - \mu'_k||^2 \leq \sum_{x_t \in C_k}||x_t - \mu_k||^2 \tag{8}$$

因为 μ'_k 是该条件下的最优点。现在，对于所有的簇 k，更新目标函数 J 可得：

$$J' = \sum_{k=1}^{K}\sum_{x_t \in C_k}||x_t - \mu'_k||^2 \leq J \tag{9}$$

结合以上两点，我们证明了 k-means 算法中的两个步骤都不会增加目标函数 J 的值。

2. Give a variant of k-mean algorithm somewhat between the original k-mean and Expectation-Maximization (EM) for Gaussian Mixture Models (GMM). Please specify the computational details of the formulas. Pseudo-codes of the algorithm would be great.

   Discuss the advantages or limitations of your algorithm.

解

算法设计 伪代码见算法 1

- 初始化：随机生成 K 个初始的簇中心 $\mu_1, \mu_2, ..., \mu_K$
- 分配步骤：对于每个数据点 x_i，计算它对于每个簇中心 μ_k 的归属概率。基于该点到簇中心的距离，并使用高斯函数：

$$p(k|x_i) = \frac{\exp(-\frac{1}{2\sigma^2}||x_i - \mu_k||^2)}{\sum_{j=1}^{K}\exp(-\frac{1}{2\sigma^2}||x_i - \mu_j||^2)} \tag{10}$$

  σ^2 为超参数，较小时算法接近于传统的 k-means。

- 更新步骤：使用上一步计算出的归属概率来更新每个簇的中心。新的簇中心是簇中数据点的加权平均值，权重是各点的归属概率：

$$\mu_k = \frac{\sum_{i=1}^{N} p(k|x_i)x_i}{\sum_{i=1}^{N} p(k|x_i)} \tag{11}$$

- 迭代：重复分配步骤与更新步骤

优点与限制 优点：

- 更适合 GMM 的数据分布，把距离改成后验概率、降低了对异常值的敏感度
- 更好地处理数据中的重叠和不清晰的簇边界

限制：

- 与传统的 k-means 相比计算复杂度更高
- 对于不同的数据分布与目标任务，需要设置不同的超参数 σ^2，对初始值敏感、可能需要 K 折交叉验证等

除此之外，另一种方式即在分配时设置阈值。

Algorithm 1 Soft k-means 算法

输入: 数据集 $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, 簇数 K, 方差 σ^2
输出: 簇中心 $\mu_1, \mu_2, \ldots, \mu_K$

随机初始化簇中心 $\mu_1, \mu_2, \ldots, \mu_K$
repeat
  for 每个数据点 $\mathbf{x}_i$ **do**
    for 每个簇 k **do**
      计算属于簇 k 的概率 $p_{ik} = \frac{\exp(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mu_k\|^2)}{\sum_{j=1}^{K}\exp(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mu_j\|^2)}$
    end for
  end for
  for 每个簇 k **do**
    更新簇中心 $\mu_k = \frac{\sum_{i=1}^{N} p_{ik}\mathbf{x}_i}{\sum_{i=1}^{N} p_{ik}}$
  end for
until 达到收敛条件，簇中心不再显著变化

3. Compare the k-mean algorithm with competitive learning (CL) algorithm. Could you apply the idea of Rival Penalized Competitive Learning (RPCL) to k-mean so that the number of clusters is automatically determined? If so, give the details of your algorithm and then implement it on a three-cluster dataset generated by yourself. If not, state the reasons.

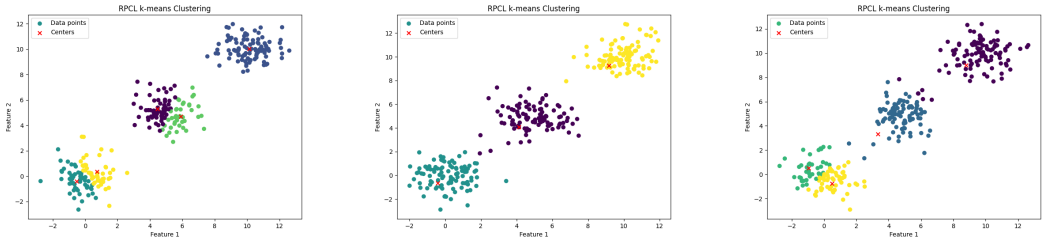
解

关于两种算法的比较如下：

- 两者都通过迭代更新簇类中心，均基于距离进行划分
- 两者的结果都对参数的初始化敏感
- K-means 需要初始化 K，更侧重数据的全局影响与批量处理；而竞争学习不需要参数 K，需要更新率 η，侧重于局部调整、对处理顺序敏感。

实验 将 RPCL 的思想引入 K-means 算法，我的想法如算法 2。在更新阶段，除了用簇内点集均值更新簇中心外，也使用以该簇中心为第二近点的点集均值进行惩罚。更新结束后，如果某簇包含的数据点数小于指定比例（数据总数 $N\times$ `remove_ratio`/当前簇的数量），删除此簇并重新执行一次分配阶段。

详细代码见附录。随机生成三**聚类**数据集并执行上述算法，迭代次数固定、初始簇数 $K=6$，设置不同惩罚率 α 与移除比例 `remove_ratio` 时可视化结果如下：



(a) $\alpha=0$（**不惩罚**），`remove_ratio`= 0.4，最终 $K'=6$ (b) $\alpha=0.15$，`remove_ratio`= 0.4，最终 $K'=3$ (c) $\alpha=0.15$，`remove_ratio`= 0.2，最终 $K'=4$

图 1: RPCL K-means 实验结果

由上图可知，在合适的 α 和 `remove_ratio` 下，算法可以找到合适的聚类数（最终划分为 3 类）。

Algorithm 2 RPCL K-means Clustering

输入: 数据集 $X=\{x_1,x_2,\ldots,x_N\}$, 初始簇数 K, 惩罚率 α, 簇移除比例 remove_ratio
输出: 更新后的簇中心 *centers*, 每个数据点的簇分配 *labels*

从数据集 X 中随机选择 K 个数据点作为初始簇中心 *centers*
for $it=1$ **to** *max_iters* **do**
    计算每个数据点到每个簇中心的距离
    对每个数据点，找到最近的簇中心和第二近的簇中心
    根据最近的簇分配更新簇中心 *centers*
    对每个簇中心，如果被分配的数据点少于 $N\times$remove_ratio/当前簇的数量，则移除该簇中心
    更新每个数据点的簇分配 *labels*
    if 簇中心 *centers* 不再显著变化 **or** 所有簇中心都被移除 **then**
        break
    end if
end for

4. Write a report on experimental comparisons on model selection performance between BIC, AIC

and VBEM.

Specifically, you need to randomly generate datasets based on GMM, by varying some factors, e.g., sample sizes, dimensionality, number of clusters, and so on.

- BIC, AIC: First, run EM algorithm on each dataset X for $k = 1, \ldots, K$, and calculate the log-likelihood value $\ln(p(X|\Theta_k))$, where Θ_k is the maximum likelihood estimate for parameters; Second, select the optimal k^* by

$$k^* = \arg \max_{k=1,\ldots,K} J(k), \tag{12}$$

  where

$$J_{AIC}(k) = \ln(p(X|\Theta_k)) - d_m, \tag{13}$$

$$J_{BIC}(k) = \ln(p(X|\Theta_k)) - \ln(N)\frac{d_m}{2}, \tag{14}$$

  where N is the number of data points in the dataset, d_m is the number of free parameters in the model, and K is a positive integer specified by the user.
- Use VBEM algorithm for GMM to select the optimal k^* automatically or via evaluating the lower bound.

解 对于 VBEM（变分贝叶斯期望最大化）算法，这是一种用于参数估计和模型选择的统计方法，它是 EM 算法的一个扩展。核心思想是利用变分推断来近似贝叶斯模型的后验分布。在高斯混合模型（GMM）的背景下，这意味着算法旨在估计混合组件的参数（如均值、协方差）以及每个组件的混合比例，同时自动确定组件的数量 k，但需要在开始时手动设置组件数量的上限。

实验 分别修改聚类数、每类样本数和特征维度数，随机生成 100 次数据并分别运行 AIC、BIC 和 VBEM 方法，使用准确度（预测得到的最佳聚类数和数据集的实际聚类数）评估其性能，得到结果如下：

聚类数	每类样本数	维度数	AIC	BIC	VBEM
2	300	2	0.94	1.00	0.99
2	1000	2	0.99	1.00	0.93
6	1000	2	0.56	0.65	0.88
6	2000	2	0.63	0.77	0.95
6	2000	5	0.98	0.98	1.00
6	2000	10	0.99	0.97	1.00

表 1: BIC、AIC 与 VBEM 实验结果——准确度评估

另外，选择其中最后一组参数进行运行时间测量（测试平台为 `AMD Ryzen 7 5800H` CPU，运行时间测量受系统调度影响，仅供参考），结果如下：

	AIC	BIC	VBEM
时间 (s)	8.89	8.92	1.47

表 2: BIC、AIC 与 VBEM 实验结果——时间性能

由以上数据观察可得：当特征维度、聚类数较少时，三者效果均较佳；此时增加聚类数，AIC 与 BIC 的准确度显著下降、VBEM 有所下降但相对仍然较好；而特征维度上升后，三者准确度再次达到较高水平。在时间性能方面，VBEM 的效率远高于另外两种方法、AIC 和 BIC 的运行时间基本相同。推测：

- VBEM 在大部分情况下准确度、时间效率均优于另外两种方法，且仅需在初始时设置 K 上限（另两种方法需要手动设置不同 K），在实际使用中倾向于使用 VBEM。
- BIC 的惩罚项大于 AIC，因其考虑了样本数量，在样本数量较多时准确度高，避免选用过于复杂的模型。
- 在聚类数量较多时，VBEM 的性能最佳。

附录 A 相关代码

第三题代码如下：

```
import numpy as np
import matplotlib.pyplot as plt

def initialize_centers(X, K):
    indices = np.random.choice(X.shape[0], K, replace=False)
    return X[indices]

def assign_clusters(X, centers):
    distances = np.sqrt(((X - centers[:, np.newaxis])**2).sum(axis=2))
    closest = np.argmin(distances, axis=0)
    second_closest = np.argpartition(distances, 2, axis=0)[1,:]
    return closest, second_closest

def update_centers(X, centers, closest, second_closest, alpha=0.2):
    new_centers = centers.copy()
    for k in range(centers.shape[0]):
        assigned_idx = (closest == k)
        if np.any(assigned_idx):
            new_centers[k] = X[assigned_idx].mean(axis=0)
        penalty_idx = (second_closest == k)
        if np.any(penalty_idx):
            new_centers[k] -= alpha * X[penalty_idx].mean(axis=0)
    return new_centers

def remove_small_clusters(X, centers, closest, min_size_ratio):
    cluster_sizes = np.array([np.sum(closest == k) for k in
        range(centers.shape[0])])
    keep_centers_mask = cluster_sizes > min_size_ratio * X.shape[0]
```

```
    return centers[keep_centers_mask], keep_centers_mask

def rpcl_kmeans(X, K, max_iters=1000, alpha=0.15, remove_ratio = 0.4):
    centers = initialize_centers(X, K)
    for it in range(max_iters):
        closest, second_closest = assign_clusters(X, centers)
        new_centers = update_centers(X, centers, closest, second_closest, alpha)
        min_size_ratio = remove_ratio / new_centers.shape[0]
        new_centers, keep_centers_mask = remove_small_clusters(X, new_centers,
            closest, min_size_ratio)
        closest = closest[keep_centers_mask[closest]]
        # if (len(centers) == len(new_centers) and
            np.allclose(centers[keep_centers_mask], new_centers, atol=1e-4)) or
            len(centers) == 0:
        #     print(f"Stopped at iteration {it+1}")
        #     break
        centers = new_centers
    return centers, closest

# 生成数据集
X = np.concatenate([np.random.randn(100, 2) + np.array([i*5, i*5]) for i in
    range(3)])

centers, labels = rpcl_kmeans(X, K=6)

# 可视化结果
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', label='Data
    points')
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', label='Centers')
plt.title('RPCL k-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```