机器学习与数据挖掘 作业二

吴天 19334019

1 Abstract

本次实验实现了用 Random、Distance-based、Random+Distance 三种初始化方法进行初始化的 Kmeans 聚类算法，并使用 UCI 数据库中的 Dry Bean Dataset, Iris Data Set, Wine Data Set, seeds Data Set, 和 Wholesale customers Data Set 数据集验证了 K-means 聚类在上述数据集上的性能。

2 Assignment

(1). 实现 K-means 算法，采用三种不同初始化方法得到三个不同版本。

(2). 在 UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets.php 找到自己认为合适的 5 个数据集，进行如下分析：

(a) 聚类个数设定为正确类个数情况下，三种不同初始化方法版本的聚类效果（用 Normalized Mutual Information 度量效果）。

(b) 设置不同聚类个数 K 的情况下（如 K=2, 3, 4, ..., n/2），三种不同初始化方法版本的“目标函数 J 随着聚类个数变化的曲线。

3 Method

3.1 Dataset

本次实验采用的数据集是 Dry Bean Dataset, Iris Data Set, Wine Data Set, seeds Data Set, 和 Wholesale customers Data Set.

其中 Dry Bean Dataset 共有 7 个类别、16 个特征，13611 个实例。实验数据可以在https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset获取。

Iris Data Set 共有 3 个类别、4 个特征，每个类别有 50 个实例，共 150 个实例。实验数据可以在http://archive.ics.uci.edu/ml/datasets/Iris获取。

Wine Data Set 该数据集共有 3 个类别、13 个特征，178 个实例。实验数据可以在http://archive.ics.uci.edu/ml/datasets/Wine获取。

seeds Data Set 共有 3 个类别、7 个特征，每个类别有 70 个实例，共 210 个实例。实验数据可以在http://archive.ics.uci.edu/ml/datasets/seeds获取。

Wholesale customers Data Set 取了地点作为类别，共有 3 个类别、8 个特征，440 个实例。实验数据可以在https://archive.ics.uci.edu/ml/datasets/Wholesale+customers获取。

3.2 Data Standardization

采用了 z-score 标准化 (zero-mean normalization)，给予原始数据的均值（mean）记为 μ 和标准差（standard deviation）记为 σ 进行数据的标准化。经过处理的数据符合标准正态分布，即均值为 0，标准差为 1，其转化函数为：

$$X = \frac{X - \mu}{\sigma}$$

3.3 K-means Clustering

聚类是一个将数据集中在某些方面相似的数据成员进行分类组织的过程，聚类就是一种发现这种内在结构的技术，聚类技术经常被称为无监督学习。K-means 是最著名的划分聚类算法，由于简洁和效率使得它成为所有聚类算法中最广泛使用的。

给定一个数据点集合和需要的聚类数目 k，k 由用户指定，k 均值算法根据某个距离函数反复把数据分入 k 个聚类中。K-means 算法步骤如下：

(1). 选取 K 个对象作为初始的聚类中心。

(2). 计算每个对象与各个种子聚类中心之间的距离，把每个对象分配给距离它最近的聚类中心。

(3). 每个聚类的聚类中心会根据聚类中现有的对象被重新计算。

(4). 步骤 2、3 将不断重复直到满足某个终止条件。终止条件可以是以下任何一个：

(a) 没有（或最小数目）对象被重新分配给不同的聚类。

(b) 没有（或最小数目）聚类中心再发生变化。

(c) 误差平方和局部最小。

3.3.1 Initialization

(1). Random method

完全随机地选择 K 个初始中心点 μ_k ， $k = 1, \ldots, K$.

(2). Distance-based method

首先随机选择一个初始中心点，接着对于每一个点，计算它到已选中心点的最小距离, 所有点中最小距离最大的点 μ 作为中心点，直至选完 $k-1$ 个中心点:

$$\mu = x^{(i)}, if\ i = argmax\ min \parallel x^{(n)} - \mu_j \parallel^2$$

(3). Random + Distance method 首先随机选择一个初始中心点，接着对于每一个点，计算它到已选中心点的最小距离, 所有点中最小距离前 1/4 大的点中随机选取 μ 作为中心点，直至选完 $k-1$ 个中心点:

3.3.2 Update The Assignment And Center

评估数据点 $x^{(n)}$ 到每一个中心点 μ_k 的距离，数据点 $x^{(n)}$ 会被注释到有最小距离的类 k

$$r_{nk} = \begin{cases} 1, & if\ k = argmin_j \parallel x^{(n)} - \mu_j \parallel^2 \\ 0, & otherwise \end{cases}$$

用每个类的数据的平均值更新中心点

$$\boldsymbol{\mu}_k \leftarrow \frac{\sum_{n=1}^{N} r_{nk}\boldsymbol{x}_n}{\sum_{n=1}^{N} r_{nk}}$$

最终目的是最小化目标函数:

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} \parallel x^{(n)} - \mu_k \parallel^2$$

即最小化每个簇中所有样本点到其中心代表点的距离之和。

可以看到上述更新的两个步骤均可以使得 J 单调不增，故 J 可以单调下降且收敛。(然而有可能是局部最优解)

3.4 Evalution Metric

我们使用 Normalized Mutual Information(NMI) 即归一化互信息来评估每种方法的分类性能，在整个评估过程中，原始数据集提供的细胞类型注释作为基本事实。互信息指的是两个随机变量之间的关联程度如下公式计算:

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

标准互信息是将互信息归一化到 0 1:

$$U(X,Y) = 2\frac{I(X;Y)}{H(X)+H(Y)}$$

其中:

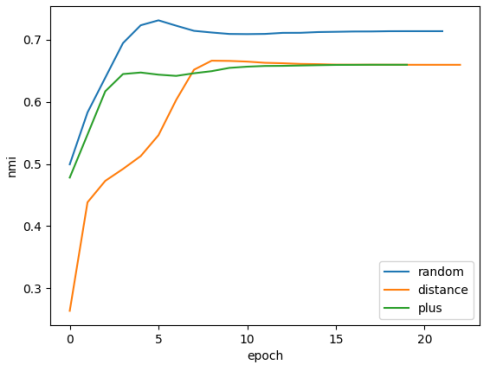
$$H(X) = \sum_{i=1}^{n} p(x_i)I(x_i) = \sum_{i=1}^{n} p(x_i)log_b\frac{1}{p(x_i)} = -\sum_{i=1}^{n} p(x_i)log_b p(x_i)$$

NMI 值越大，代表聚类的效果越好。

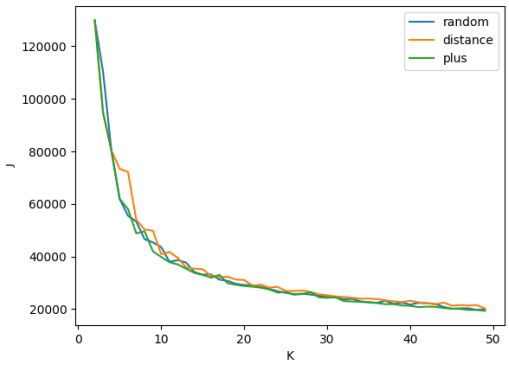
4 Result

4.1 Nonlinear Classfier and Regularize

本次实验采用的数据集是 Dry Bean Dataset, Iris Data Set, Wine Data Set, seeds Data Set, 和 Wholesale customers Data Set 五个数据集，对于每个数据集，绘制了两个图：分别是：聚类个数设定为正确类个数情况下，三种不同初始化方法版本的聚类效果（用 NMI 度量），以及设置不同聚类个数 K 的情况下，三种不同初始化方法版本的“目标函数 J 随着聚类个数变化的曲线，如图 1 ∼ 5 所示:



(a) 三种初始化下，NMI 随迭代次数的变化曲线

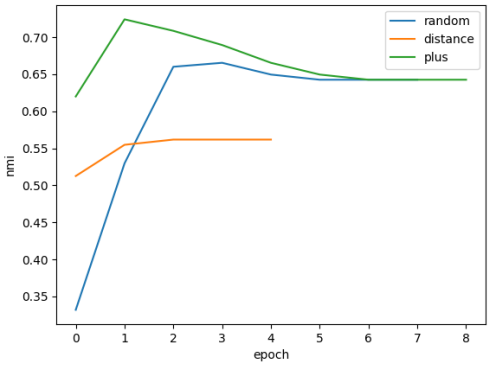


(b) 三种初始化下，目标函数 J 随聚类个数 k 变化曲线

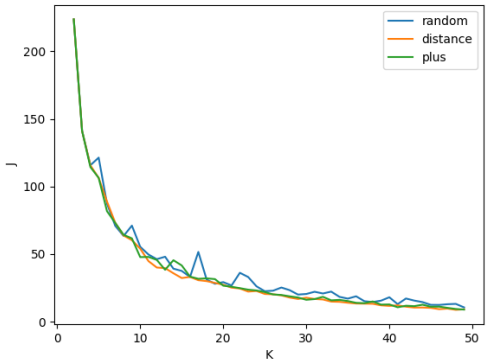
图 1: Dry Bean 数据集下 K-means 聚类结果

从实验结果中，我们可以看到，由于 K-means 算法采用的是迭代方法，得到的结果只是局部最优，且由于随机初始点的原因，我们不能保证三种初始化方法哪种一定最好，distance 的方法也可能受离群值影响较大，因此不同数据集上三个方法的表现不一致，(图 1(a)、图 2(a)、图 3(a)、图 4(a)、图 5(a))。

此外，在聚类个数正确的情况下，各个数据集上 K-means 表现也不尽相同，比如在

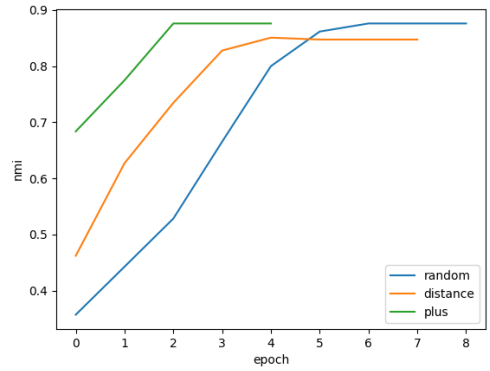


(a) 三种初始化下，NMI 随迭代次数的变化曲线

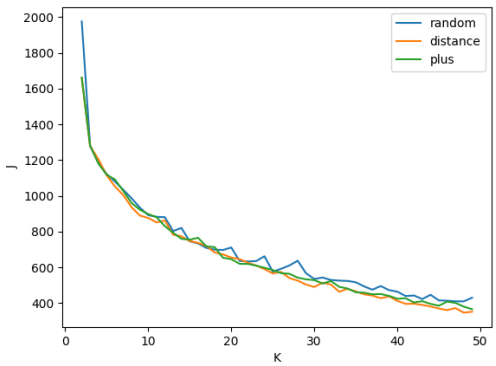


(b) 三种初始化下，目标函数 J 随聚类个数 k 变化曲线

图 2: Iris 数据集下 K-means 聚类结果

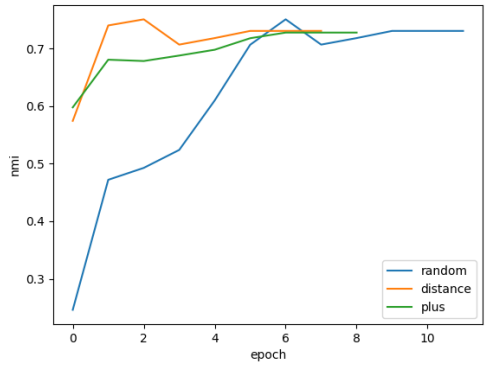


(a) 三种初始化下，NMI 随迭代次数的变化曲线

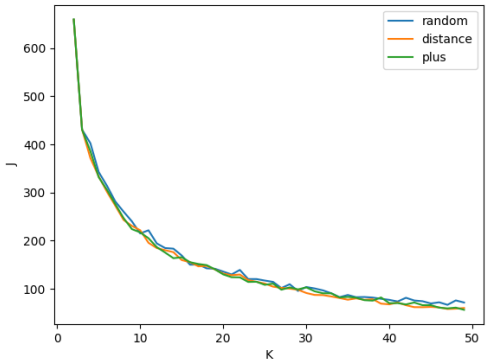


(b) 三种初始化下，目标函数 J 随聚类个数 k 变化曲线

图 3: Wine 数据集下 K-means 聚类结果

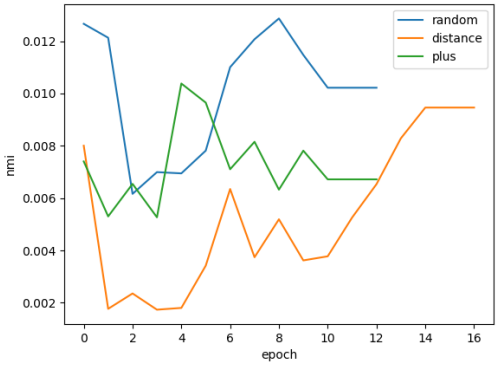


(a) 三种初始化下，NMI 随迭代次数的变化曲线

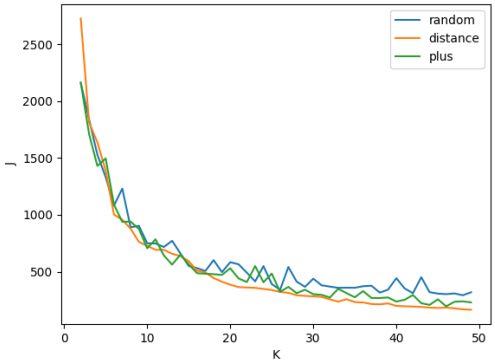


(b) 三种初始化下，目标函数 J 随聚类个数 k 变化曲线

图 4: seeds 数据集下 K-means 聚类结果



(a) 三种初始化下，NMI 随迭代次数的变化曲线



(b) 三种初始化下，目标函数 J 随聚类个数 k 变化曲线

图 5: Wholesale customers 数据集下 K-means 聚类结果

Wine 数据集上表现非常好 NMI 近 0.9(图 3(a))，而在 Wholesale customers 数据集上表现的最好的初始化方式的 NMI 也不过 0.01(图 5(a))，这意味着使用 K-means 聚类的结果基本上都不正确。这可能是由于 K-means 可处理的数据类型有限，采用欧氏距离作为相似性度量的聚类算法，只有数据对象分布较均匀的球形或类球形的簇结构，才能得到良好的聚类效果。

通过绘制目标函数 J 随聚类个数 k 变化曲线 ($K = 2, \ldots, 50$)，可以看到随着聚类个数 K 的不断增大，目标函数 J 的值是不断变小的 (图 1(b)、图 2(b)、图 3(b)、图 4(b)、图 5(b))，显然，中心点变多，所有点到最近中心点的距离和自然会越来越小。由图我们可以发现，一开始 J 的减小幅度是非常大的，后面 J 的减小幅度越来越小，这样，我们可以利用肘部法则猜测肘部的大致范围，这与实际的聚类个数是一致的。

5 code

代码实现中，将 K-means 模型作为一个类，初始的参数有 k 值和初始点的选取方法，如图 6 所示：

```
class kmeans():
    def __init__(self, k, init_func):
        assert init_func == "random" or init_func == "distance" or init_func == "plus"
        self.k = k
        self.init_func = init_func
        self.centers = None
```

图 6: K-means 类

5.1 fit 方法

fit 方法包含了模型训练的全部过程，并输出、返回了一些参数的值，以便后续的画图。输入数据 X 与 y 分别是数据矩阵和实际的类标，最大训练轮数 epoch 以及收敛的限制范围 tol 用于控制模型收敛的次数（若达到最大训练轮数，或相邻两轮迭代的总距离之差绝对值小于 tol, 或中心点位置不改变时终止迭代）。每轮迭代都会计算预测的标签和实际标签的 NMI，采用了 sklearn 中的 normalized_mutual_info_score 得到 NMI，fit 方法实现如图 7 所示：

并且模型收敛中，计算最小距离和目标函数 J 采用的是欧式距离，如图 8 所示：

5.2 初始化方法

Random、Distance、Random + Distance 三种初始化方法的实现，如图 9 所示：

```python
def fit(self, X, y, max_epochs=100, tol=1e-3):
    self.centers = eval("generate_" + self.init_func)(X, self.k)
    last_clusters = None
    last_dist = None
    sum_dists, nmis = [], []

    for epoch in range(max_epochs):
        pred = []
        sum_dist = 0.
        clusters = [[] for _ in range(self.k)]
        #update the label of each point
        for i in range(X.shape[0]):
            idx, dist = nearest_dist(X[i], self.centers)
            clusters[idx].append(i)
            sum_dist += dist
            pred.append(idx)
        #update the label of center points
        for i in range(self.k):
            self.centers[i] = np.mean(X[clusters[i]], axis=0)

        nmi = normalized_mutual_info_score(y, pred)
        sum_dists.append(sum_dist)
        nmis.append(nmi)

        if (last_clusters is not None and last_clusters == clusters) or\
           (last_dist is not None and math.fabs(sum_dist - last_dist) <= tol):
            sum_dists.append(sum_dist)
            nmis.append(nmi)
            print("epoch %d : sum_dist : %f , nmi : %f, k : %d"
              % (epoch, sum_dist, nmi, self.k))
            break
        else:
            last_clusters, last_dist = clusters, sum_dist
    return sum_dists, nmis
```

图 7: fit 方法

```python
def nearest_dist(x, centers):
    minDist, idx = math.inf, -1
    for i, center in enumerate(centers):
        curDist = dist(x, center)
        if curDist < minDist:
            minDist = curDist
            idx = i
    return idx, minDist


def dist(point1, point2):
    return np.sum(np.square(point1 - point2))
```

图 8: 距离计算

```python
def generate_random(X, k):
    idex = list(range(X.shape[0]))
    idxs = np.random.choice(idex,k,replace=False)
    return X[idxs]

def generate_distance(X, k):
    random_idx = np.random.randint(X.shape[0])
    centers = [X[random_idx]]
    d = np.zeros(X.shape[0])

    for pos in range(1, k):
        for i in range(X.shape[0]):
            _, d[i] = nearest_dist(X[i], centers[:pos])
        newcenter = np.argmax(d)
        centers.append(X[newcenter])
    return centers

def generate_plus(X, k):
    random_idx = np.random.randint(X.shape[0])
    centers = [X[random_idx]]
    d = np.zeros(X.shape[0])

    for pos in range(1, k):
        for i in range(X.shape[0]):
            _, d[i] = nearest_dist(X[i], centers[:pos])
        newcenter = np.random.choice(np.where(d>np.quantile(d,0.75,interpolation='higher'))[0])
        centers.append(X[newcenter])
    return centers
```

图 9: 三种初始化方法

5.3 读取数据

数据集均被保存成.csv 或.xlsx 文件格式, 读取数据，返回数据矩阵 X，类标 y，类别数 k 列名为 Class 的作为类标 y, 其余均为数据矩阵 X，并且对 X 进行标准化。

由于不同数据集中，类名可能是字符串的形式，因此对于类名会构造字典，进行一个字符串转数字的转换。读取数据的实现代码见图 10:

```python
# Make types to labels dictionary
def type_to_label_dict(all_type):
    # input -- types
    # output -- type_to_label dictionary
    type_to_label_dict = {}
    for i in range(len(all_type)):
        type_to_label_dict[all_type[i]] = i
    return type_to_label_dict

# Convert types to labels
def convert_type_to_label(types, type_to_label_dict):
    # input -- list of types, and type_to_label dictionary
    # output -- list of labels
    types = types.tolist()
    labels = list()
    for t in types:
        labels.append(type_to_label_dict[t[0]])
    return labels
```

(a) 类标转换

```python
#normalize
def normalize(data):
    mean = np.mean(data, axis=0)
    std = np.std(data, axis=0)
    data = (data - mean)/std
    return data

def readData(file):
    if file[-4:] == "xlsx":
        df=pd.read_excel(file)
    elif file[-3:] == "csv":
        df=pd.read_csv(file)
    types = list(set(df["Class"]))
    types.sort()
    dict = type_to_label_dict(types)
    data_x = df.drop('Class', axis=1).values
    data_y = df["Class"].values.reshape(-1,1)
    data_y = convert_type_to_label(data_y, dict)
    data_x = normalize(data_x)
    return data_x, data_y, len(types)
```

(b) 读取并返回

图 10: xlsx 或 csv 数据读取

5.4 main 函数

main 函数的主要作用是对数据集进行训练，并画图：聚类个数设定为正确类个数情况下，三种不同初始化方法的图，以及设置不同聚类个数 K 的情况下，三种不同初始化方法版本的目标函数 J 随着聚类个数变化的曲线，main 函数实现代码见图 11：

```
################ Fig1 ######################
data_x,data_y,Klength = readData(conf.data)

model = kmeans(Klength, "random")
J1, nmis1 = model.fit(data_x, data_y, conf.epochs)
model = kmeans(Klength, "distance")
J2, nmis2 = model.fit(data_x, data_y, conf.epochs)
model = kmeans(Klength, "plus")
J3, nmis3 = model.fit(data_x, data_y, conf.epochs)

plt.plot(range(len(nmis1)), nmis1, label="random")
plt.plot(range(len(nmis2)), nmis2, label="distance")
plt.plot(range(len(nmis3)), nmis3, label="plus")
plt.xlabel("epoch")
plt.ylabel("nmi")
plt.legend()
plt.savefig(conf.save+"nmis.png", format='png')
plt.close()
```

(a) 固定正确 K

```
################ Fig2 ######################
J1s, J2s, J3s = [],[],[]
shangxian = int(50)
for i in range(2, shangxian):
    model = kmeans(i, "random")
    J1, _ = model.fit(data_x, data_y, conf.epochs)
    J1s.append(J1[len(J1)-1])
    model = kmeans(i, "distance")
    J2, _ = model.fit(data_x, data_y, conf.epochs)
    J2s.append(J2[len(J2)-1])
    model = kmeans(i, "plus")
    J3, _ = model.fit(data_x, data_y, conf.epochs)
    J3s.append(J3[len(J3)-1])

plt.plot(range(2, shangxian), J1s, label="random")
plt.plot(range(2, shangxian), J2s, label="distance")
plt.plot(range(2, shangxian), J3s, label="plus")
plt.xlabel("K")
plt.ylabel("J")
plt.legend()
plt.savefig(conf.save+"Js.png", format='png')
plt.close()
```

(b) K 值变化

图 11: main 函数聚类并画图

6 Summary

通过这次实验，我对老师课堂上讲到的 K-means 无监督聚类理论知识的掌握更加深入了，并且动手将算法实现，这提高了我把理论知识实现为可行代码的能力，感觉收获很大。

同时通过对不同数据集运行了自己的代码，我发现我的代码实现的效率比较低，一旦数据量变大，并且聚类个数 k 增大，由于要对每个中心、每个数据点进行计算，程序的运行时间非常漫长，需要进一步优化我的实现。此外，K-means 也有其自身的局限性，在某些数据集上的表现并不是很好，这是由于数据的特点决定的，也提醒了我在以后的任务建模的时候，要慎重考虑选择的算法。

7 Supplementary material

本实验的代码已上传至 GitHub，可以在我的 Github 获得。
GitHub 仓库是https://github.com/PointerA/ML_DM_course