哈尔滨工业大学

<<大数据分析>>

实验报告之二

(2019 年度春季学期)

姓名:	王丙昊
学号:	1160300302
学院:	计算机学院
教师:	杨东华

实验二 聚类与分类

一、实验目的

掌握对数据使用聚类分析和分类，并理解其在大数据环境下的实现方式。

二、实验环境

Linux 操作系统、Hadoop 伪分布式环境、Java 语言

三、实验过程及结果

1. KMeans

- **Kmeans 算法原理**

“k-means 算法”根据聚类所得簇划分 $C = \{C_1, C_2, \cdots, C_k\}$最小化平方误差，误差越小则簇内样本相似度越高，但是这样需要考察样本集所有可能的簇划分，这是一个 NP 难问题。因此 k-means 算法采用了**贪心策略**，通过**迭代**近似来近似求解平方误差。

- **KMeans 的 MapReduce 实现**

在 KMeans 中实现了两个 MapReduce，第一个 MapReduce 用来初始化时随机选取中心点，第二个 MapReduce 用来进行 KMeans 的迭代。这里主要说明第二个 MapReduce，也就是 **KMeans 的 MapReduce 的具体实现**方法：

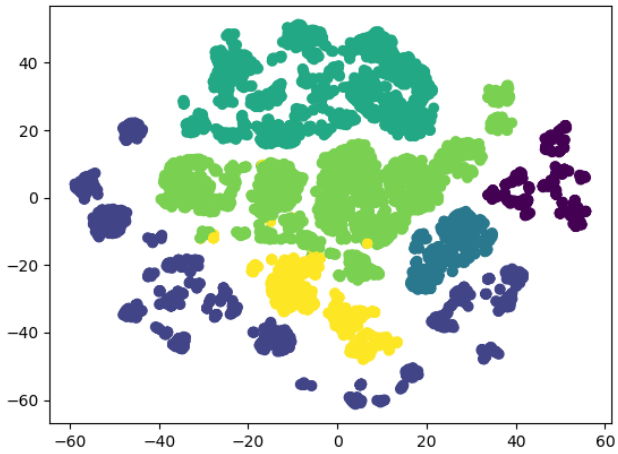
Map 阶段：在对每个样本计算前，需要从 DFS 中读取上一轮 MapReduce 计算的中心点，求每个样本到当前所有中心点的最短距离，记录距离最短的中心点的编号，然后将编号作为 key，原始的样本记录作为 value 输出。

Reduce 阶段：重新计算一个簇的中心点，通过计算被分到该类中的所有记录各维的均值，来获取新的中心点各维的取值。

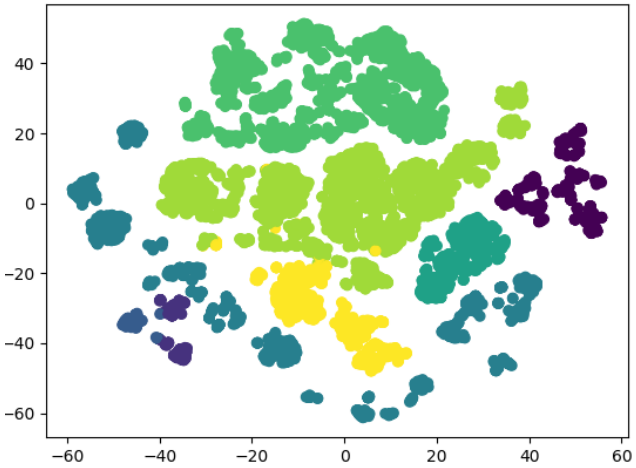
迭代终止条件：当一轮迭代前后计算的中心点不发生变化，或者迭代轮数达到了规定的阈值，就停止迭代。迭代停止后，根据当前的中心点，仅再执行一轮 Map，将每个样本及其所划分的类别输出。

- **KMeans 结果展示**

K = 6：



K = 8：



2. GMM

- **GMM 算法原理**

GMM 采用概率模型来表达聚类原型，为明确显示高斯分布与参数的依赖关系，定义高斯混合分布

$$p_{\mathcal{M}}(\boldsymbol{x}) = \sum_{i=1}^{k} \alpha_i \cdot p(\boldsymbol{x} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

假设样本由高斯混合分布给出：首先，以概率 α_i 选择第 i 个混合成分（α_i 相当于先验概率）；再从所选的高斯分布中采样。假设样本集 $D=\{x_1,x_2,\cdots,x_m\}$ 由上述过程生成，令 z_j 表示生成样本 x_j 的高斯混合成分，则由贝叶斯定理，z_j 的后验概率为

$$\begin{aligned} p_{\mathcal{M}}(z_j = i \mid \boldsymbol{x}_j) &= \frac{P(z_j = i) \cdot p_{\mathcal{M}}(\boldsymbol{x}_j \mid z_j = i)}{p_{\mathcal{M}}(\boldsymbol{x}_j)} \\ &= \frac{\alpha_i \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} . \end{aligned}$$

记为 γ_{ji}.GMM 算法将 D 分为 k 个簇 $C=\{C_1, C_2, \cdots, C_k\}$，每个样本的簇标记为

$$\lambda_j = \underset{i\in\{1,2,\ldots,k\}}{\arg\max}\ \gamma_{ji}\ .$$

因此，**只要确定模型参数{（αi，μi，Σi）,i=1，2，…,k}**，便可以得到每个样本的簇划分，可以对样本集 D 采用极大似然估计，GMM 的目标便是最大化似然函数

$$\begin{aligned} LL(D) &= \ln\left(\prod_{j=1}^{m} p_{\mathcal{M}}(\boldsymbol{x}_j)\right) \\ &= \sum_{j=1}^{m} \ln\left(\sum_{i=1}^{k} \alpha_i \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\right) \end{aligned}$$

常用 **EM 算法**进行迭代优化求解。

- **GMM 的 MapReduce 实现**

在 GMM 中，需要计算的参数有：

1. 每个样本属于每个类的概率 γ_{ji}（样本 j 被划分为 i 类的概率），最终也是通过这个后验概率来确定每个样本被划分到哪一类；

2. 每个高斯分布的均值、协方差阵及混合系数：

计算新均值向量: $\boldsymbol{\mu}_i' = \frac{\sum_{j=1}^{m}\gamma_{ji}\boldsymbol{x}_j}{\sum_{j=1}^{m}\gamma_{ji}}$;

计算新协方差矩阵: $\boldsymbol{\Sigma}_i' = \frac{\sum_{j=1}^{m}\gamma_{ji}(\boldsymbol{x}_j-\boldsymbol{\mu}_i')(\boldsymbol{x}_j-\boldsymbol{\mu}_i')^{\mathrm{T}}}{\sum_{j=1}^{m}\gamma_{ji}}$;

计算新混合系数: $\alpha_i' = \frac{\sum_{j=1}^{m}\gamma_{ji}}{m}$;

在 GMM 的 MapReduce 实现中，每一轮迭代需要经历两轮 MapReduce，Map 阶段都是计算每个样本被分为每一类的后验概率 γ_{ji}，所以这两轮 MapReduce 可以使用同一个 Map 类；第一轮 MapReduce 的 Reduce 阶段计算**新均值向量和新混合系数（αi，μi)**；第二轮 MapReduce 的 Reduce 阶段计算**新协方差矩阵（Σi)。**

这里使用两轮 MapReduce 分开计算（αi，μi）和（Σi）是有原因的：如果仅使用一轮 MapReduce，在 Reduce 阶段需要计算（αi，μi，Σi)，但（Σi）依赖于（μi）。这样不能通过遍历一次 Iterable<Text> value 来解决，而需要将 value 的数据存在 Reduce 节点上，当计算完（μi）后，才能计算（Σi）。使用两轮 MapReduce 可以避免在 Reduce 节点上存储原数据，在 Reduce 阶段遍历一遍 Iterable<Text> value 便可以得到新参数。

GMM 的 MapReduce 具体实现如下：

Map 阶段：用于计算每个样本属于每一类的后验概率 γ_{ji}. 在对每一个样本计算之前，首先从 DFS 中读取上一轮的迭代结果（αi，μi，Σi)。然后根据算法原理中的公式计算后验概率，对每一个样本 x_j, 在该阶段输出 k 条记录，**key 为所分的类别**，value 为分为该类别的后验概率以及原始数据。

Reduce1 阶段：第一轮 MapReduce 的 Reduce 阶段，计算的是新均值向量和新混合系数（αi，μi）。求解过程中遍历一遍该 reduce 节点上的数据即可，代入公式便可求得

$$\boldsymbol{\mu}_i' = \frac{\sum_{j=1}^{m}\gamma_{ji}\boldsymbol{x}_j}{\sum_{j=1}^{m}\gamma_{ji}};\ \alpha_i' = \frac{\sum_{j=1}^{m}\gamma_{ji}}{m};\ 。$$

Reduce2 阶段：第二轮 MapReduce 的 Reduce 阶段，计算的是新的协方差矩阵（Σi)。首先需要从 DFS 中读取本轮迭代的第一个 MapReduce 计算的均值向量（μi），然后根据样本数据、均值向量以及后验概率来计算新协方差矩阵。

$$\boldsymbol{\Sigma}_i' = \frac{\sum_{j=1}^{m}\gamma_{ji}(\boldsymbol{x}_j-\boldsymbol{\mu}_i')(\boldsymbol{x}_j-\boldsymbol{\mu}_i')^{\mathrm{T}}}{\sum_{j=1}^{m}\gamma_{ji}};$$

- **GMM 实现时注意的问题**

（1）迭代终止条件：如果已经达到了所规定的最大迭代轮数或者似然函数 LL(D)增长很少甚至不再增长，则退出；否则开始新一轮的迭代。

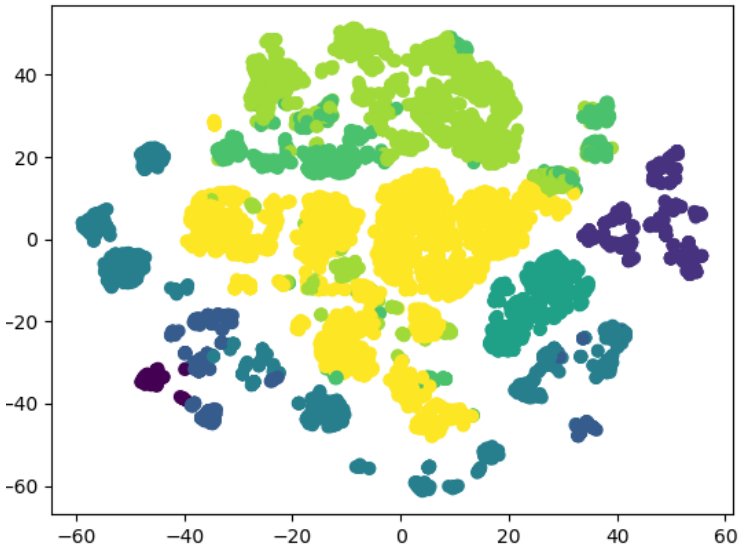
（2）初始均值向量的选取：GMM 的执行速度要比 KMeans 慢得多，尤其是当数据量较大、分类较多时更明显；GMM 同样不能得到全局最优解，当初始参数选择不好时容易陷入**局部极值**，因此可以使用 KMeans 的结果来为 GMM 初始化，再通过 GMM 迭代。

（3）迭代过程中奇异阵的处理：使用 GMM 算法在实验数据集上迭代时，会出现协方差阵是奇异矩阵的情况。每当计算完新的协方差阵之后，判断其是否为奇异阵，如果是则在其对角线上加上一个很小的偏移量。

- **GMM 结果展示**

K = 8

由于 GMM 算法在数据量较大时执行速度非常缓慢，在测试时仅选用了其中一部分数据进行测试，所以结果不如 KMeans 的聚类效果。



3. NaiveBayes

- **NaiveBayes 算法原理**

基于概率和误判损失来选择最优的类别标记。在生成式模型中，计算分类的后验概率需要用到**类别先验概率 P(Y)**以及**类条件概率 P(X|Y)**。其中类别先验概率可以根据各类样本比例进行估计；而类条件概率难以从样本直接获得。

于是朴素贝叶斯做了“**属性条件独立性假设**”：假设所有属性相互独立，则后验概率可以写为：

$$P(c \mid \boldsymbol{x}) = \frac{P(c)\,P(\boldsymbol{x} \mid c)}{P(\boldsymbol{x})} = \frac{P(c)}{P(\boldsymbol{x})} \prod_{i=1}^{d} P(x_i \mid c)$$

在本次实验中，所用到的数据的属性为连续特征，这里假设每个类别中的每个特征均服从**正态分布**。**(Xi | Y=k) ~ N(μ ik, σ ²ik).**对于正态分布的参数估计，可以使用**极大似然估计(MLE)**，估计方法如下：

$$\mu^{*} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma^{*2} = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2$$

当获得类别先验概率以及各类别各维度正态分布的参数之后，可以根据下面的式子对测试集数据进行分类：

$$Y^{new} \leftarrow \arg\max_{y_k} \quad \pi_k \prod_i Normal(X_i^{new}, \mu_{ik}, \sigma_{ik})$$

- **NaiveBayes 的 MapReduce 实现**

经过上面算法原理分析，NaiveBayes 只需要估计类别先验概率 P(Y)，以及各类别各维度高斯分布的均值和方差。

NaiveBayes 的 MapReduce 实现中，需要使用两轮 MapReduce，第一轮 MapReduce 扫描一遍训练数据，计算两个类的先验概率；第二轮 MapReduce 也是扫描一遍训练数据，计算各类别各维度的高斯分布的均值和方差。

TrainPrior_Map 阶段：用于计算先验概率的 Map 阶段。对于每条训练数据，从中提取出类别，以类别为 key，数字 1 为 value 进行输出。

TrainPrior_Reduce 阶段：作用是为每个类别的训练样本进行计数。类别相同的记录会聚集到一个 reduce 节点上，对该节点上的数据进行计数，便可以得到该类的训练样本个数。类别先验的计算方法就是使用某一类的样本数除以样本总数。

TrainGaussian_Map 阶段：以训练样本的类别和维度 i 为 key，样本在维度 i 上的取值 X[i]为 value，进行输出。假设样本共有 n 维，则一条训练数据输出 n 条(key, value)对。

TrainGaussian_Reduce 阶段：每一类的每一维的记录值都会聚集在同一个 Reduce 节点上，通过下面式子可以估计高斯分布的参数：

$$\mu^* = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma^{*2} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

- **NaiveBayes 结果展示**

精确率：

```
<terminated> NaiveBayes [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java
correct rate: 0.7352702906947454
```

4. Logistic 回归

- **Logistic 回归算法原理**

在 Logistic 回归中，做了如下**假设**：

1）X 各维条件独立（贝叶斯假设）；

2）$X_i \mid Y=Y_k \sim N(u_{ik}, \sigma^2)$，即方差与 Y 的类别无关

3）$Y \sim Bernoulli(\Pi)$，即二分类问题

基于如上假设和 Sigmoid 函数，我们可以推出：

$$\ln \frac{p(y=1 \mid \boldsymbol{x})}{p(y=0 \mid \boldsymbol{x})} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b \ .$$

这样可以将后验概率写为：

$$p(y=1 \mid \boldsymbol{x}) = \frac{e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}}{1+e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}}$$

$$p(y=0 \mid \boldsymbol{x}) = \frac{1}{1+e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}}$$

Logistic 回归的目标便是确定上式中的 w 参数矩阵和常量 b，这里可以在 X 的每一组数据前加一项 1，使得 b 可以合并到 w 矩阵中形成**矩阵 W**。接下来使用**极大似然法**来估计 W。

$$\begin{aligned} W_{MLE} &= \arg\max_W P(< X^1, Y^1 > \ldots < X^L, Y^L > |W) \\ &= \arg\max_W \prod_l P(< X^l, Y^l > |W) \end{aligned}$$

但上式中需要 X 与 Y 的联合概率分布，这个是得不到的，于是退而求其次，使用：

$$W_{MCLE} = \arg\max_W \prod_l P(Y^l|W, X^l)$$

这样就找到了需要最大化的目标函数，对其化简可以得到：

$$\begin{aligned} l(W) &\equiv \ln \prod_l P(Y^l|X^l, W) \\ &= \sum_l Y^l(w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

现在的目标就是最大化 L(W)，由于没有解析解，这里通过梯度上升法来求解。在梯度上升法中，每轮迭代时使用如下式子来更新 W 矩阵：

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

- **Logistic 回归 MapReduce 实现**

经过上面算法原理的分析，Logistic 回归的主要任务就是迭代获得矩阵 W，来最大化似然函数 L(W)。

在 Logistic 回归的 Mapreduce 的实现中，主要使用了两个 MapReduce，第一个用来计算矩阵 W；第二个用来计算似然函数 L(W)。

TrainWeight_Map 阶段：计算矩阵 W 的 Map 阶段。首先要从 DFS 中读取上轮迭代的 W 矩阵结果，用于计算样本二分类的后验概率。执行阶段，对于每个**训练数据**，首先计算它为(Y=1)类的后验概率，也就是算法原理中更新矩阵 W 的式子中的 **$P(Y^l = 1 \mid X^l, W)$**；然后对于样本的每一维数据，以 **key 为维数 i，value 为（Y，X[i]，$P(Y^l = 1 \mid X^l, W)$）**进行输出，其中 Y 为训练样本所属类，X[i]为该样本在维度 i 上的取值。

TrainWeight_Reduce 阶段：计算矩阵 W 的 Reduce 阶段。首先从 DFS 中读取上轮迭代的 W 矩阵结果，用于计算新的矩阵 W。由于 Map 阶段以维数 i 为 key 进行输出，所以更新每一维所需的数据都会聚集到一个 Reduce 节点上。每个 Reduce 节点对数据进行一遍扫描，根据下面的式子进行计算，便可以得到矩阵 W 在每一维上新的取值。

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

LikeHoodFunction_Map 阶段：计算似然函数值的 Map 阶段。对于每条**训练数据**，计算其在迭代（TrainWeight_MapReduce）前后的矩阵 W 下如下式子的取值：

$$Y^l(w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))$$

对于每条记录，分别使用迭代前后的矩阵 W 通过上面的式子进行计算。该阶段的输出，

key 用于区分矩阵 W 是迭代前的还是迭代后的，value 为上面式子的计算结果。

LikeHoodFunction_Reduce 阶段：计算似然函数值的 Reduce 阶段。在 Map 阶段，由相同的权重系数矩阵 W 求得的值都会聚集到一个 Reduce 节点上。在该阶段，只需要对节点上的数据的 value 字段进行累加，便可以得到 L(W)的值。

迭代终止条件：每轮迭代都经过两轮 MapReduce，第一轮 MapReduce 用来计算新的权重系数矩阵 W，第二轮 MapReduce 计算迭代前后权重系数矩阵 W 下的似然函数值 L(W)。当两个似然函数值相差不大或者迭代次数达到规定的最大迭代轮数时，停止迭代。

- **Logistic 回归在测试数据集上进行测试**

在测试数据集上进行测试（精确率），这里使用了一个仅由 Map 阶段组成的 MapReduce。

Test_Map 阶段：在测试集上计算**精确率**的 Map 阶段。首先从 DFS 中获取权重系数矩阵 W。对每一条测试数据，计算其被二分类的后验概率，类别为两个后验概率中较大的一个。将分类结果与它的类别相比较，看是否相同。

- **Logidtic 回归结果展示**

精确率：

```
<terminated> Logistic [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java
correct rate: 0.7780099753136178
```

四、实验心得

**********。

1. 在本次实验中，练习了如何使用 MapReduce 框架来编写一些分类、聚类的算法。大多数算法都涉及到了迭代，学会了如何使用 MapReduce 进行迭代。迭代的基本方法就是每轮迭代完成后，将结果写回 DFS，下一轮迭代开始时，从 DFS 读取上一轮迭代的结果。

2. 使用 MapReduce 框架进行编程时，提前分析好每一个 Map 阶段、Reduce 阶段需要做的工作、数据流，会大大的减少编程的难度，程序本身并不复杂。