模式识别

作业（一） 实现C-Means算法

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2022年2月28日
学　　院	信息学院
课程名称	模式识别

作业（一） 实现C-Means算法

目录

1 实现和运行结果

1.1 选择特征和实现

首先做出Iris数据集两两变量的散点图和单个变量的分布曲线图，如图3（见附录： Iris数据集多变量图），可以发现萼片长款参数单独出现对聚类结果没有较好的贡献，重合部分较多。为了充分利用数据，再计算萼片面积和花瓣面积。发现组合“花瓣面积-萼片面积”、“花瓣面积-萼片长度”等都有较好的区分，但是**考虑到K-Means算法团块状聚类的特点**，我选择做出散点图最符合团状特点的“花瓣面积-萼片面积”作为用于聚类的两个特征，但是该特征选择依然难以避免iris-virginica和iris-versicolor两类有所重叠。实现见附录：核心部分源代码中的代码1。

该实现使用了以下第三方库：

- **Pandas** 用于读取CSV数据集文件、计算平均数和方差等统计量
- **NumPy** 用于加速数值计算

1.2 运行结果

聚类结果如图1，图上除了聚类结果还显示了初始时选择的点、聚类中心。可见C-Means给出了较好的聚类结果。

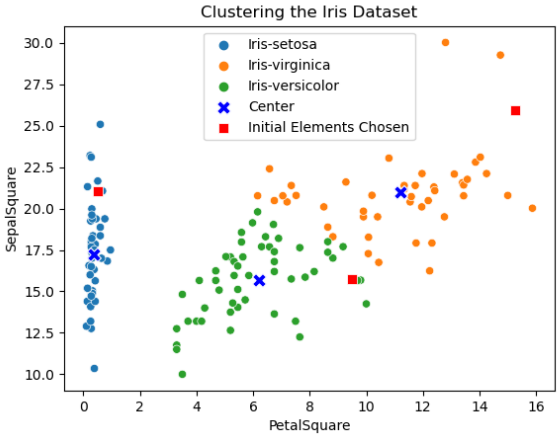

图 1: 聚类结果可视化

2 误差统计和分析

表格 1: 误差统计

物种	数量	聚类数量	错误量	错误率
Iris-virginica	50	46	9	0.2
Iris-versicolor	50	54	13	0.24
Iris-setosa	50	50	0	0

为了进一步判断聚类的效果，将聚类结果和Iris数据集中的标签对照，统计类的大小、错误量、错误率，得到表1和图2。可以发现C-Means较好地区分出了iris-setosa，但是iris-virginica和iris-versicolor中存在一定的错误，这和选择特征时的观察相符合，即iris-virginica和iris-versicolor的分布有重合的部分。

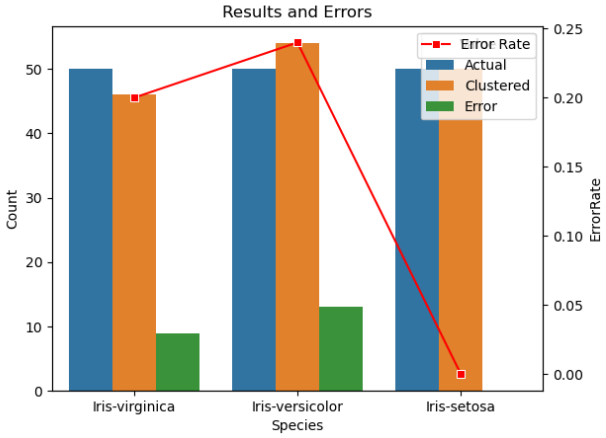

图 2: 结果和错误统计

但是，考虑到错误率相当低，可以认为C-Means算法给出了很好的聚类效果。

A 附录： Iris数据集多变量图

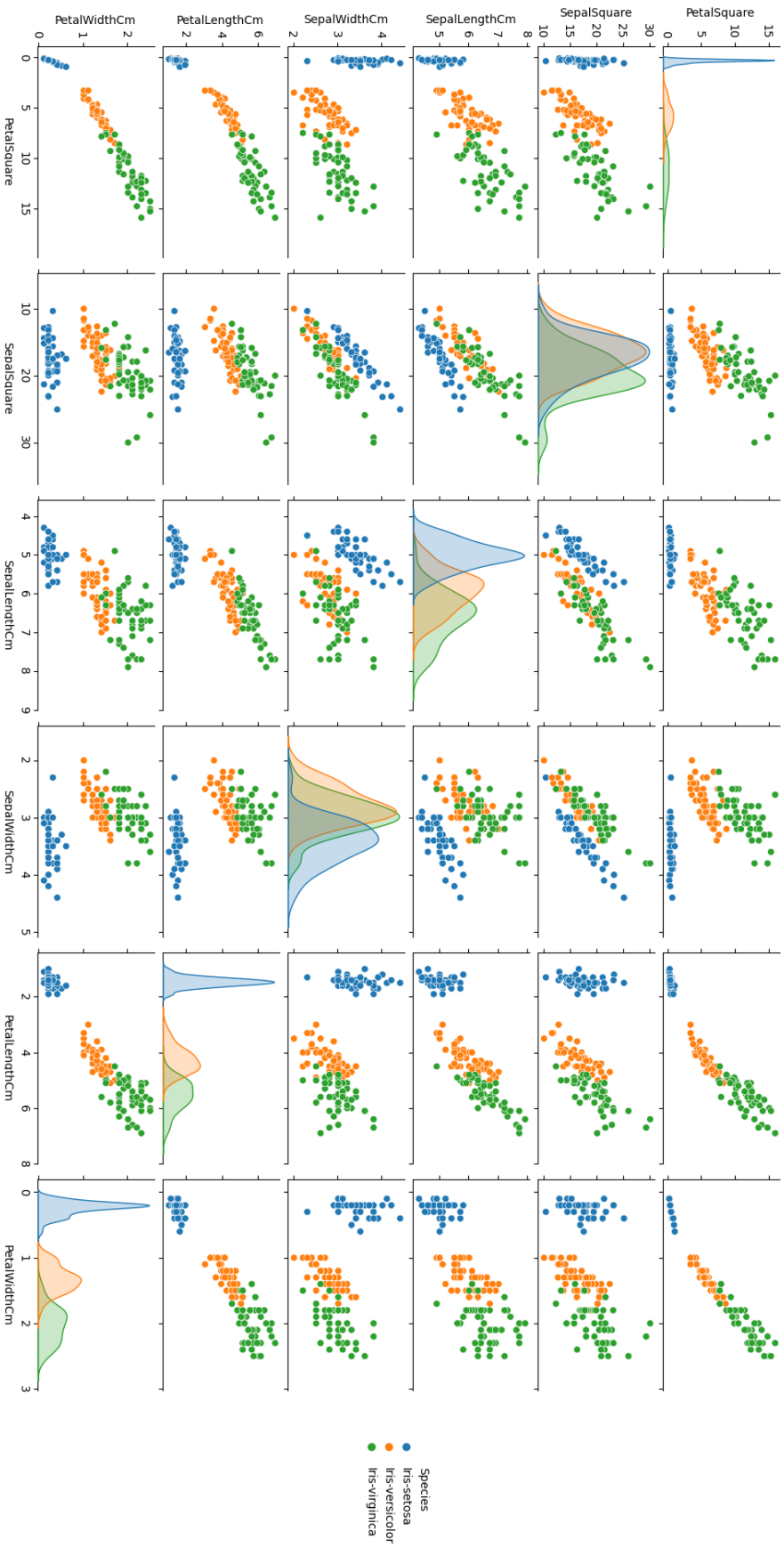

图 3: Iris数据集多变量图(Pair Plot)

B　附录：核心部分源代码

代码 1: 瑞利分布生成

```
class KMeans:
        def __init__(self, feats: pd.DataFrame, k:  int):
                self.tries = 0
                self.feats = feats
                self.k = k

        def _wcss(self, centroids, cluster) ->  float:
                ret = 0.0
                for i, val in  enumerate(self.feats.values):
                ret += np.sqrt(
                (centroids[ int(cluster[i]), 0] - val[0]) ** 2 + (centroids[
                    int(cluster[i]), 1] - val[1]) ** 2)
                return ret

        def cluster(self, max_tries:  int = 32767):
                self.tries = 0

                cluster = np.zeros(self.feats.shape[0])

                centroid_indexes, centroids = self.feats.sample(n=self.k).index, self.feats.sample(n
                    =self.k).values

                while self.tries < max_tries:
                        self.tries += 1

                        for  id, row in  enumerate(self.feats.values):
                                min_dist =  float('inf')
                                for cid, centroid in  enumerate(centroids):
                                        dist = np.sqrt((centroid[0] - row[0]) ** 2 + (centroid[1] -
                                            row[1]) ** 2)
                                        if dist < min_dist:
                                                min_dist, cluster[ id] = dist, cid
                                        clustered_centroids = self.feats.copy().groupby(by=cluster).
                                            mean().values
                                        if np.count_nonzero(centroids - clustered_centroids) == 0:
                                                break
                                        else:
                                                centroids = clustered_centroids
                return centroid_indexes, centroids, cluster, self._wcss(centroids, cluster)

        def get_tries(self) ->  int:
                return self.tries
```