Machine Learning: Assignment 1

黄绵秋 19307130142@fudan.edu.cn

19 级计算机 — October 19, 2021

Assignment 1

使用 KNN 算法对 iris 数据集进行分类, 采用交叉验证的方式处理训练数据和测试数据

1 任务描述

1.1 使用 KNN 算法对 iris 数据集进行分类

1.2 通过优化和调超参来使准确率进一步提高

2 数据描述

数据直接采用`sklearn.datasets` 的数据，使用`sklearn.datasets.load_iris()` 来进行数据的导入。数据一共有 150 条，每个样本有 4 个特征，基于这 4 个特征进行分类。

3 数据预处理

3.1 线性扫描法

- 简单将`iris` 的数据进行导入并将`data` 和`target` 分别存储，利用了`data` 中的全部 feature 以增加预测的准确性
- 使用简单交叉验证的方式对数据进行分组处理，一部分用于训练，一部分用于测试。分组比例默认为训练:测试=0.8:0.2

```python
def data_split(data, label, split_rate=0.2):
    """split the data according to the split rate

    Args:
        data (list): the raw dataset, it should be list or list-like(like numpy.array or torch.tensor)
        label (type): the raw label dataset, it should be list or list-like(like numpy.array or torch.tensor)
        split_rate (float, optional): Defaults to 0.2.

    Returns:
        [train_data, test_data, train_label, test_label]
    """
    random.seed(2019)
    random.random()
    train_data, test_data, train_label, test_label = [[], [], [], []]
    for i in range(len(data)):
        if random.random() < split_rate:
            test_data.append(data[i])
            test_label.append(label[i])
        else:
            train_data.append(data[i])
            train_label.append(label[i])
    return train_data, train_label, test_data, test_label
```

Notice: 为保证测试结果与随机情况无关，使用`random.seed` 来保证每次运行时随机生成的为相同的值。但使用简单交叉验证会出现一个问题，即数据与`seed` 的选取非常相关，不同的`seed` 会极大程度影响实验结果（后续实验结果展示会有体现）。

4 算法介绍

4.1 算法原理

4.1.1 算法概论

KNN(K Nearest Neighbors) 算法通过距离寻找预测点的最近的 K 个点的类别来预测预测点属于哪个类别。给定一个需要预测的点的坐标，分别计算所有点到预测点的距离，然后求得距离最近的前 k 个数，依据这 k 个数的类别进行投票，获得票数最多的类别被选为预测类别。

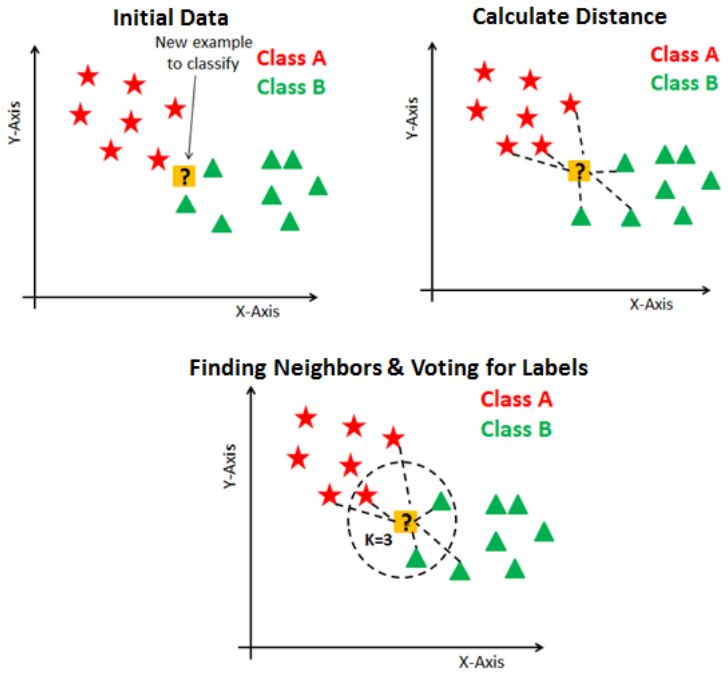


Figure 1: KNN 算法流程图解

4.1.2 算法细节

KNN 算法在实现时需要注意的细节如下

- 有时特征并不是量化的，譬如看两个词之间的关系，两个词均不是数字，此时无法直接利用 KNN 算法，因而需要先进行特征提取 (feature extraction) 和嵌入 (embedding)。本例中，由于`sklearn` 中的数据已经进行了特征提取等，我们便无须再进行此步操作
- 数据之间并不直接拥有“距离”，我们将不同种类的特征映射到线性空间的不同维度上，我们将映射后在线性空间中的点之间的位置的距离认作距离，是抽象出来的距离，常用欧式距离来进行计算，公式如下

点 $x_i(x_i^1, x_i^2, \ldots, x_i^l)$ 和 $x_j(x_j^1, x_j^2, \ldots, x_j^l)$ 之间的距离`Distance` 满足：

$$\texttt{Distance} = \sqrt{\sum_{k=1}^{l}(x_k^i - x_k^j)^2}$$

- 由于 k 值会影响参与投票的点的数量，因而会极大程度地影响投票结果。所以需要调整 k 值找到最合适的 k 值以使预测结果尽可能精确。

4.2 算法实现

4.2.1 模型初始化

```
    def __init__(self, k=4):
        """Initialize the model with the given k value

        Args:
            k (int, optional): Defaults to 4.

        Raises:
            ValueError: k value must be a positive number
        """
        if k <= 0:
            raise ValueError("Invalid k value!It should be a positive number")
        self.k = k
        self.data = None
        self.label = None
```

4.2.2 数据加载

```
    def get_data(self, data, label):
        """This function is used to load the data

        Args:
            data (list): the feature of the dataset
            label (list): the label of the dataset
        """
        self.data = data
        self.label = label
```

4.2.3 距离计算

```
    @staticmethod
    def get_distance(point_1, point_2):
        differ_pow2 = [(point_1[i] - point_2[i]) ** 2 for i in range(len(point_1))]
        return math.sqrt(sum(differ_pow2))
```

4.2.4 标签预测

```
    def predict(self, point):
        """prediction the most probable label

        Args:
            point (list): the point to predict

        Returns:
            int : the label of the prediction
        """
        distance = dict()
        for i in range(len(self.data)):
            distance[i] = self.get_distance(self.data[i], point)      # 获取各点到预测点的距离
        sorted_index = [item[0] for item in sorted(distance.items(), key=lambda x: x[1])]
        top_k_index = sorted_index[:self.k]                           # 进行排序并获取前k个结果
        top_k = dict()
        label_set = set(self.label)
        for i in label_set:                                           # k个最近的邻点进行投票
            top_k[i] = 0
        for p in top_k_index:
            top_k[self.label[p]] += 1
```

```
        top_k = sorted(top_k.items(), key=lambda x: -x[1])          # 找到票数最高的label作为预测值
        return top_k[0][0]
```

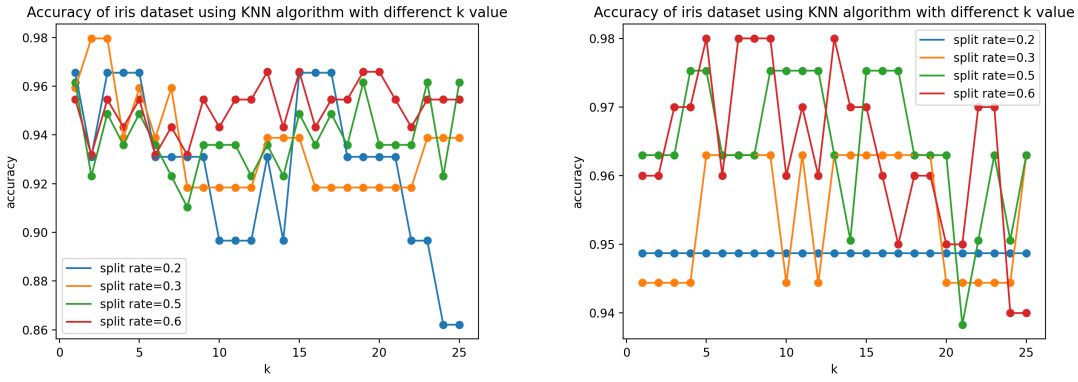
4.2.5 正确率检测

```
    def score(pred, labels):
    """count the accuracy based on the test dataset

    Args:
        pred (list): the prediction of the test dataset label
        labels (list): the true values of the test dataset label

    Returns:
        score (float): the score of the prediction
    """
    count = 0
    for i in range(len(pred)):
        if pred[i] == labels[i]:
            count += 1
    score = count / len(pred)
    return score
```

4.3 实验结果

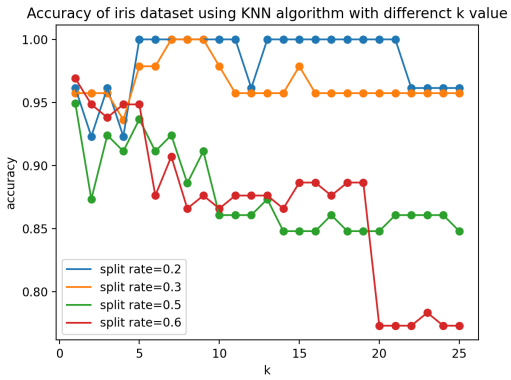
- 由于`iris` 的样本数量较小，受随机化的影响很大，因而不同`random.seed` 结果大相径庭。
- 样本数量小也导致简单交叉验证的比例影响也很大，因为参与训练的样本数量可能不够，准确性受影响。
- k 值的选取会直接影响参与投票的样本数量，因而需要寻找最合适的 k 值

由此依据这几点进行实验，结果如下

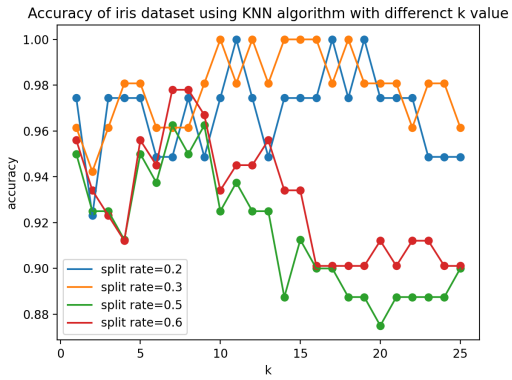


(a) random.seed=2019 (b) random.seed=2021

Figure 2: 实验结果



(c) random.seed=810975



(d) random.seed=114514

Figure 2: 实验结果

从实验结果可以发现由于数据量小，`random.seed` 对准确率分布的影响很大，但数据均有共同趋势，即为当 k 不断增大时，准确率会下降。这是因为 k 值较大时，参与投票的邻居太多，较后的邻居相关性较差，因而导致相关性下降。

4.4 模型优化

4.4.1 优化思路

- 鉴于数据量小，简单交叉验证中`split_rate` 对结果影响较大，为消除此影响，采用 K 折交叉验证，将数据等分成 k 组，然后每次用当中 1 组作为测试样本，其他组作为训练样本，每个组均进行相同操作，最后对准确率取平均。该方法同时可以避免`random_seed` 带来的影响。
- 使用 KD-Tree 进行剪枝处理求 k 个最近邻居，但由于数据量只有 150，因而认为使用 KD-Tree 会造成建树、剪枝时大量的资源浪费，因而放弃该优化思路。
- 每个邻居的权重都相同，会导致可能出现两个标签票数相同的情况。而且，逻辑上而言更靠近预测点的邻居话语权应该更大、权重应该更大，因为投票时采用基于距离确定权重的方式，可以获得更准确的数据。

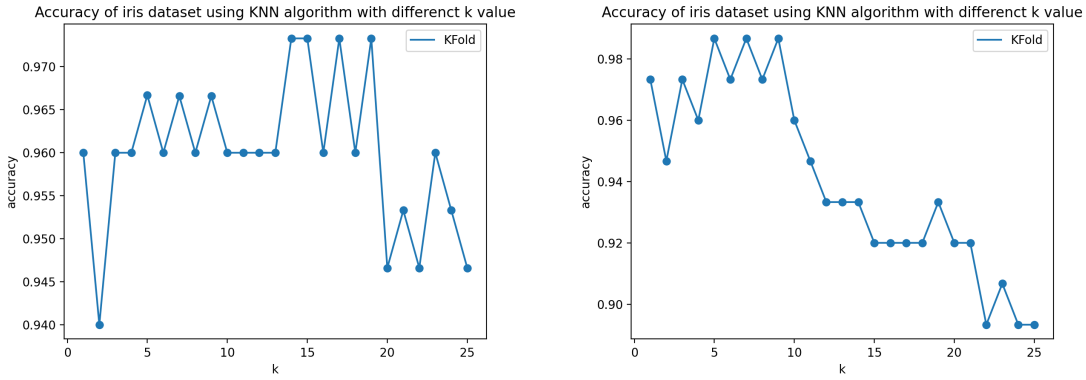
4.4.2 优化结果

自己未基于`numpy` 实现了一份 KFold 算法，使用了`random.shuffle` 以实现数据的乱序处理，代码如下

```python
def _k_fold(data, label, k=10):
    """k fold algorithm to split the data

    Args:
        data (list): the raw dataset, it should be list or numpy.array
        label (type): the raw label dataset, it should be list or numpy.array
        k (int, optional): the total number of the groups. Defaults to 10.

    Returns:
        data_group_list, label_group_list : the list of the groups
    """
    if k <= 0:
        raise ValueError("Invalid k value!It should be a positive number")
    data_group_list = []            # to store the divided data groups
    label_group_list = []           # to store the divided label groups
    group_count = len(data)//k      # the number of data in a single group
    if type(data) == numpy.ndarray:
        data = data.tolist()        # turn the numpy.array into list
        label = label.tolist()      # otherwise the shuffle may go wrong
    data_copy = copy.deepcopy(data)
    label_copy = copy.deepcopy(label)
    random.seed(2021)
    random.shuffle(data_copy)
```

```
    random.seed(2021)
    random.shuffle(label_copy)
    for i in range(k):
        if i != k - 1:
            data_group = data_copy[i*group_count : (i+1)*group_count]
            label_group = label_copy[i*group_count : (i+1)*group_count]
        else:
            data_group = data_copy[i*group_count:]
            label_group = label_copy[i*group_count:]
        data_group_list.append(data_group)
        label_group_list.append(label_group)
    return data_group_list, label_group_list
```

完成KFold 算法后利用该算法进行交叉验证处理，重新进行了一次实验，实验结果如下

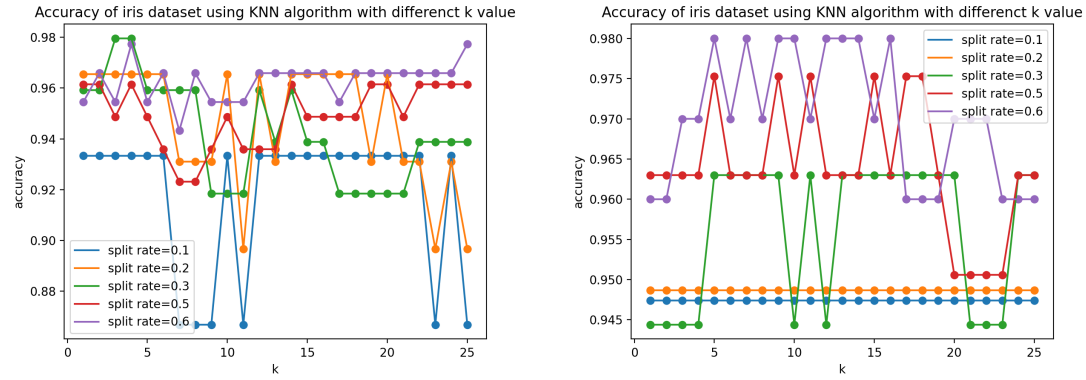


(a) devided into 10 groups (b) devided into 5 groups

Figure 3: 基于 KFold 算法的优化

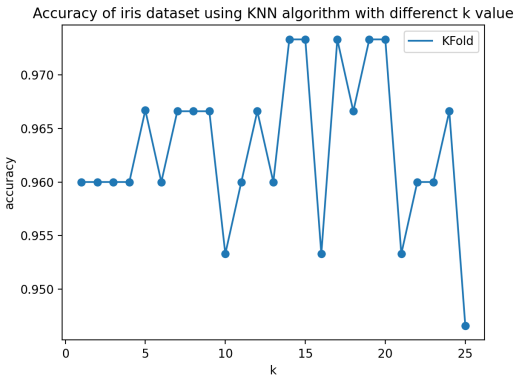
再调整投票权重计算方式，调整方式（逻辑代码）和优化结果如下

```
    for neighbor in k_Nearest_Neighbors:
        _vote[neighbor.label] += constant_number / (distance(neighbor, point) + another_constant)
```

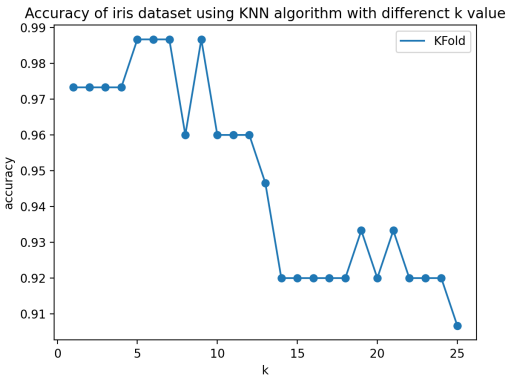


(a) random.seed=2019 (b) random.seed=2021

Figure 4: 基于权重策略调整的优化



(c) devided into 10 groups



(d) devided into 5 groups

Figure 4: 基于权重策略调整的优化

从上述结果中可以得出,基于`KFold`算法进行优化后,分成 5 组的情况比简单交叉验证时取`split_rate=0.2`时更能体现准确率跟随 k 的变化的趋势，因而认为`KFold`的优化是合理的。在对投票权重策略进行调整之后，准确率的变化更加平稳，且维持了较高水平，因而认为投票权重策略的优化也是合理的。。

5 实验总结

本次实验我通过纯 Python 实现了 KNN 算法，并使用简单交叉算法进行数据处理，并将结果进行可视化，得到了较好的结果，在 k 值取 3-7 之间有较好的效果，大概达到 96% 左右。之后通过使用调整交叉验证算法，改用`KFold`交叉验证算法进一步使得数据稳定，尽可能地利用到了各组数据；再通过调整投票权重策略使得结果更可信更有说服力。最终结果较好，但鉴于普通 KNN 已有很好的效果，优化并没有起到非常明显的效果，而且因为数据量较小，偶然因素影响较大，但逻辑上可知优化是合理且可行的。