07+KNN

2019 年 3 月 12 日

1 K 近邻（K nearest neighbor）

1.1 一、概念

K 近邻和 K 均值有异曲同工之妙，主要的计算都集中在计算距离上，分类/聚类也都建立在计算好的距离基础上，因此距离公式是个非常重要的组成部分。

K 近邻是有监督学习算法，模型学习仅仅是储存了所有数据而已。然后将需要预测的数据拿来一一比对，选取训练集中和输入最相近的数据标签进行返回。

而 K 则表示返回的距离最相近的 K 个样本的标签，K 是奇数较为常见，因为当返回到标签有多种时更容易 “投票” 选出哪个/类样本和输入更接近。K 控制了分类边界的 “粗糙” 和 “圆润” 的程度，如果 K 等于 1，边界将会非常粗糙非常容易造成过拟合，如果 K 无穷大为训练集的样本数，那么如果训练集中各类样本数目不严格相等，最后返回的标签将会是训练样本中存在最多的类别标签。所以这样看来，K 的值是调控模型从过拟合到欠拟合的超参数。

同时 K 近邻也可能存在不能确定的情况，比如 K=3 时，前三个最接近的样本来自三类，此时若返回最接近的那个样本标签，则此时相当于 K=1。此时随即返回一个标签的处理更为合适。

1.2 二、应用

先前各种距离已经 Kmeans 中介绍过，这里用 L2 距离进行计算。

```
In [1]: import numpy as np
        import pandas as pd
        from scipy import stats

        iris = pd.read_csv('data_set/Iris.data')
        index = np.random.permutation(iris.shape[0])
        train = iris.iloc[index[0:100]].copy()
        test = iris.iloc[index[100:150]].copy()
        train = np.array(train)
        test = np.array(test)
```

一些实施细节：

1. 不需要把标签设置成 0-1，因为 KNN 不涉及对标签的操作，所以可以直接返回标签
2. 取距离前 K 小的坐标用的是 *np.argpartition(arr, k)* 它利用了快速排序中的思想，只关注第 k 大的数然后比第 k 大的数的坐标在前，小的在后，于是这样就可以实现不用完全排序就找出前 k 小的元素坐标。
3. L2 距离公式中有开根号的部分，但开根号不改变增减性，所以这里并不需要
4. numpy 中没有取众数的函数，有 *np.bincount()* 可以统计各个元素出现的次数，但是这样也计算了很多我们不需要的东西，所以用了 scipy 的库
5. KNN 的实现有些书会将 KD 树，其目的还是为了更快地搜索，我的数据结构基础不好，担心讲不清楚暂时作罢

```
In [162]: def vote(train, index, k):
              if len(np.unique(train[index, 4])) < k:
                  return np.array(stats.mode(train[index, 4]))[0]
              else:
                  return train[np.random.randint(len(index)), 4]

          def KNN(train, test, k=3):
              m = test.shape[0]
              prediction = None
              for i in range(m):
                  diff = train[:, 0:4] - test[i, 0:4]
                  diff = np.power(diff, 2)
                  diff = np.sum(diff, axis=1)
                  index = np.argpartition(diff, k)[:k]
                  pre = vote(train, index, k)
                  prediction = np.append(prediction,pre)
              prediction = np.delete(prediction, 0)
              return prediction

          prediction = KNN(train, test)
          print(prediction)
```

```
['Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor'
```

```
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor'
 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa'
 'Iris-setosa' 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-virginica' 'Iris-setosa' 'Iris-setosa' 'Iris-virginica'
 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica'
 'Iris-virginica']
```

```
In [163]: print(prediction == test[:, 4])
          print(np.sum(prediction == test[:, 4])/test.shape[0])
```

```
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True False  True  True  True  True  True  True  True  True  True  True
  True]
0.9795918367346939
```

```
In [169]: np.argpartition([2,6,5,7,4,8,9,1], 3)
```

```
Out[169]: array([7, 0, 4, 2, 1, 5, 6, 3], dtype=int64)
```

```
In [174]: np.argpartition([2,6,5,7,4,9,8,1], 7)
```

```
Out[174]: array([7, 0, 4, 2, 1, 3, 6, 5], dtype=int64)
```

```
In [ ]:
```