# $STFCV$：第一次大作业

# $kNN$ 算法实践与分析

*August 10, 2023*

**NAME: 陈帅行**

## 1. 代码实现

代码已上传至：Github Url

主要基于 $numpy$ 仓库实现，根据要求没有使用 $sklearn$ 等算法库。

最终包含了 3 个文件：

1. $dataloader.py$ 实现了数据集的读取，返回需要的标签等参数。

2. $kNN-Cifar10.py$ 和 $kNN-MINST.py$ 分别基于 $Cifar10$ 和 $MINST$ 数据集实现了 $kNN$ 算法，并计算出最佳的 $k$ 值。

## 2. Distance Metrics

代码中分别计算了了常用的曼哈顿距离与欧氏距离，函数实现如下：

```
def compute_distances(self, X):
    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))

    # 欧氏距离计算
    o_dists = np.sqrt(np.sum(np.square(self.X_train), axis=1) + np.sum(np.square
    (X), axis=1)[:, np.newaxis] - 2 * np.dot(X, self.X_train.T))

    # 曼哈顿距离计算
    m_dists = np.sum(np.abs(self.X_train - X[:, np.newaxis]), axis=2)

    return o_dists, m_dists
```

. Distance Metrics

最终通过 $k$ 折交叉验证，对两者的表现进行了比较。

欧氏距离 (*Euclideandistance*)，度量的是两点之间的直线距离，它考虑了各个特征之间的差异，适用于特征空间的连续性较强的情况。它能够捕捉到特征之间的绝对差异，适用于数值差距交大的情况，但也容易受到受到异常值的影响。

曼哈顿距离 (*Manhattandistance*)，度量的是两点之间的曼哈顿距离，即沿着坐标轴的距离之和。它的优点是对离群点较为鲁棒，不受异常值的干扰。但是并不适用于绝对值差距较大的情况，而适用于具有离散特征或者具有较强方向性的问题。

**3. 选择最优 $k$ 的方法**

在 $kNN\_*.py$ 文件中，我通过循环试验不同的 $k$ 值的准确率差异，来检测出最终的最优 k 值。

最终发现，对于 $Cifar10$ 数据集，$k = 10$ 时效果最好，正确率为 57%。

而对于 $MINST$ 数据集，$k = 3$ 时效果最好，正确率高达 97%。

同时，我也生成了不同 $k$ 值的正确率变化图表，列在 $repo$ 的 $README.md$ 文件之中。

**4.$kNN$ 算法优劣分析**

我认为 $kNN$ 算法最大的优点在于其思想简单，同时也较容易实现，相关的实现代码 100 行以内就可以完成。同时，$kNN$ 算法区别于神经网络，并不需要进行参数预训练，可以实时计算。$kNN$ 算法在使用曼哈顿距离的情况下，可以忽略异常值的影响，对于数据有较高的容纳程度。

但是，$kNN$ 算法的缺点也很明显，因为它需要计算测试样本和训练样本之间的距离，因此数据集规模越大，开销越大，效率很低。同时，$kNN$ 算法对于 k 值的需求过高，不同 k 值之间差别较大，需要不断调试。如果数据集存在不均衡的情况，或者不同类别数据数目差别较大，最后分类效果也会较差。同时，对于一些数据边界和临界，$kNN$ 算法会出现重叠模糊的情况。