实验一：K-NN 算法的实现

杨扬　计算机科学与技术　1611132

摘　要

本次实验是一个典型的机器学习分类问题，本文档针对于此问题，介绍了 K-NN 算法的原理，结合于实验数据集本身设计了算法，并基于源程序对实验的设计进行了解析，最后对实验的结果进行了分析。简单来说，本文档在对实验所有要求均做出实现的同时，也成功地进行了多方面的拓展，较好地完成了此次实验。

关键词：机器学习，K-NN, 分类

目录

1 问题描述 **1**

2 解决方法 **3**

3 实验及源程序分析 **4**

3.1 实验环境 4

3.2 源程序分析 4

3.3 实验结果分析 8

4 总结 **9**

参考文献 **9**

1 问题描述

K-近邻算法是机器学习领域中一种用于分类和回归的非参数统计方法。简单来说，在分类问题中，K-近邻算法的思想可概括为如下：给定一个训练数据集，其中的实例类别已定。分类是对于新的类别，根据其 k 个最近邻的训练实例的类别，通过多数表决法等方式进行预测。由以上可以看出，K-近邻算法不具有显性的学习过程，K-近邻算法实际上利用训练数据集对特征空间进行划分，并作为其分类的模型。

本次实验的数据集是一个二进制手写数字图像的分类问题，训练集为 1115 张大小为 16×16 的二进制图像，测试集为 478 张。数据保存在 semeion_train.csv 和 semeion_test.csv 两个文件中，其保存格式如图 1 所示，csv 文件中是一个 1115×266 的矩阵，其中前 256 列是手写数字图像的一维存储形式，1 代表该点有着色，0 代表无，后十列是每张图片对应的 one-hot 形式存储的标签。

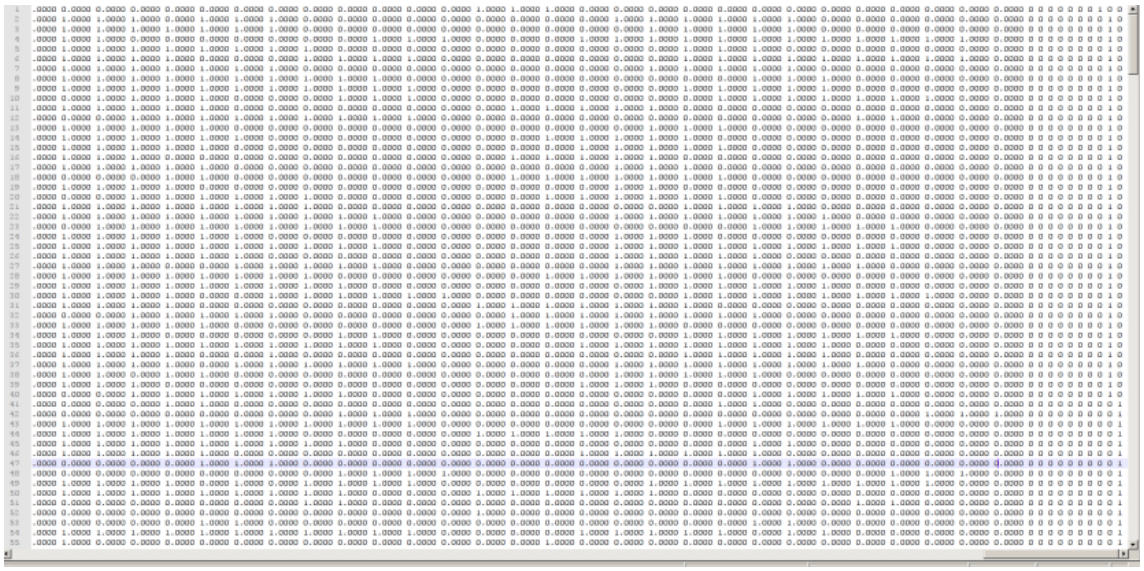


图 1: 数据存储形式

为了对数据有更直观的理解，我们可以将每一条数值形式存储的图片转换为二进制图像来进行查看，如图 2、3 所示，此时我们便可以清晰地看出每一条数据的含义。

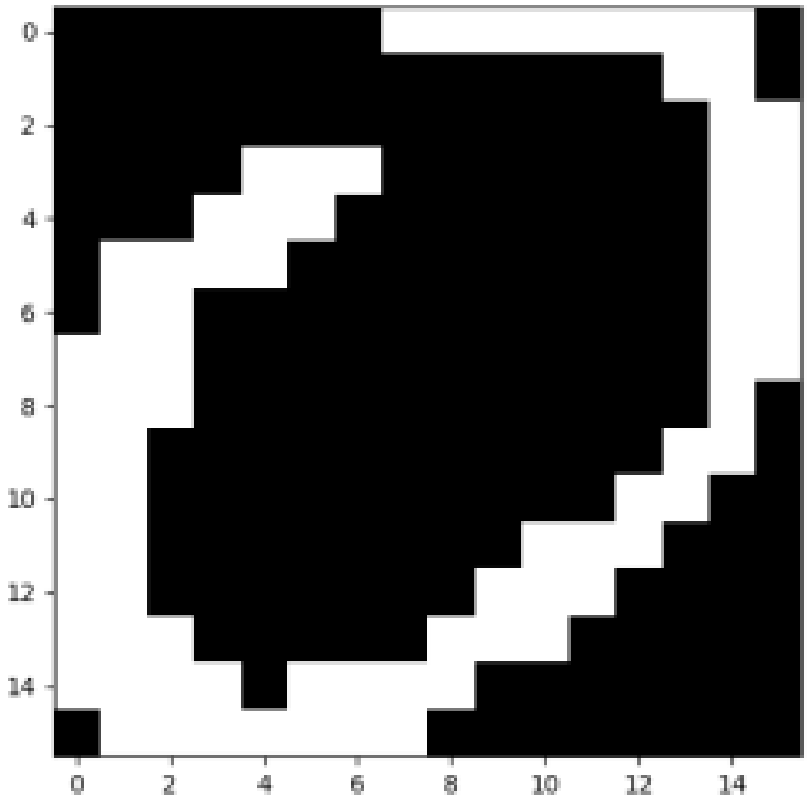


图 2: 第 201 条样本对应灰度图像

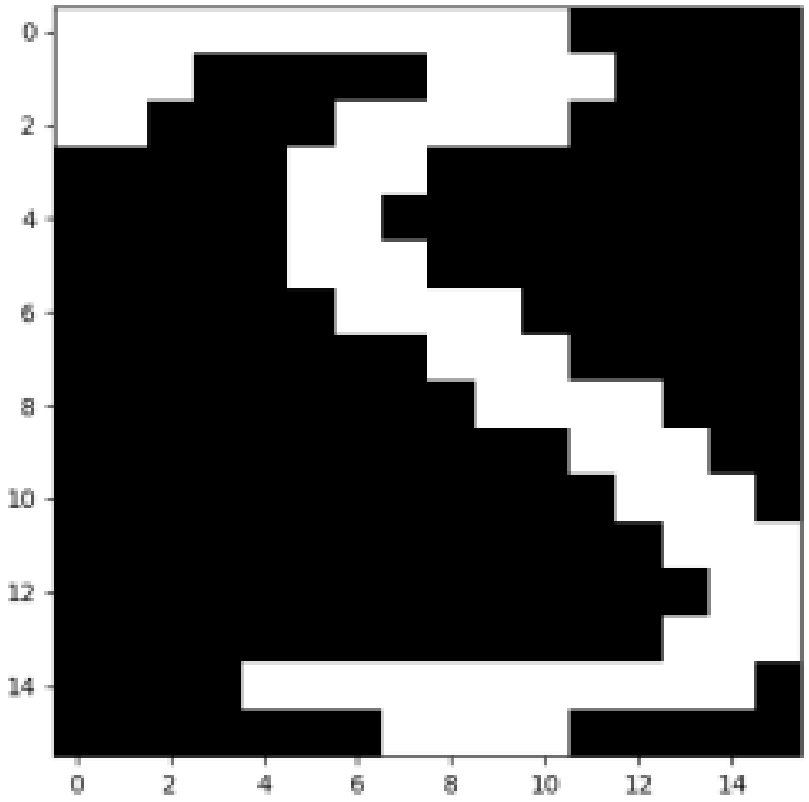


图 3: 第 271 条样本对应灰度图像

2 解决方法

不难看出，此问题是一个典型的分类问题，类别共有十项，分别是 0-9 的每一个数字，每个数字在二维平面上的结构特征具有明显不同，我们运用 K-NN 的思想，将训练数据分为 10 个不同的簇，然后将测试数据逐条输入，通过 K-NN 算法即可判别出其所属的种类。

简单来说，我所采用的解决方法是基于 K-NN 算法的核心思想进行设计的，具体流程如下：

- 计算待分类点与已知类别的点之间的欧氏距离
- 按照距离递增次序排序
- 选取与待分类点距离最小的 k 个点
- 确定前 k 个点所在类别的出现次数

- 返回前 k 个点出现次数最高的类别作为待分类点的预测分类

此外实验中我同样尝试了基于距离加权的 K-NN 算法，流程如下：

- 计算待分类点与已知类别的点之间的欧氏距离
- 按照距离递增次序排序
- 选取与待分类点距离最小的 k 个点
- 设置一个大小为 10 的权值数组，将每一项的值置为-1
- 将 k 个点与待分类点的距离加到权值数组对应的项
- 返回权值数组中非负的最小的类别作为待分类点的预测分类

3 实验及源程序分析

3.1 实验环境

本次实验在 Ubuntu 18.04 下进行，实验所使用的编程语言为 Python 3.6，IDE 为 Pycharm 2018，主要使用了 numpy、matplotlib、sklearn 等 Python 包。

3.2 源程序分析

由于此次实验的内容较少，实验数据规模也不算太大，所以我们在一个 py 文件里即可完成本次实验，具体来说，程序的步骤如下所示：

首先我们在 Python 中导入所需要用到的 Python 包：

```
import numpy as np
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
```

然后我们需要导入相应的实验数据，这里需要注意的是，虽然实验数据存储的格式是 csv 格式，但是我们打开文件即可发现同行数据间的分隔方式并不是 csv 文件常见的以逗号分隔，而是以空格来分隔，所以这里我们不需要采用 pandas 或者 csv 这样的数据读写包，只需要采用 numpy 中的一个方法就可以了：

```python
def read_data(path):
    raw = np.loadtxt(path)
    data = raw[:, 0:-10]  # 倒数十列之前全为训练样本
    label = raw[:, -10:]  # 倒数十列为标签
    return data, label
```

导入数据之后，下面就进入到了 K-NN 算法的具体步骤，首先我们定义一个计算欧式距离的函数，同样调用 numpy 中的基本数学函数即可，这里换成其他距离函数也是极为简单的：

```python
def cal_distance(vec1, vec2):
    return np.sqrt(np.sum(np.square(vec1 - vec2)))  # 计算欧式距离
```

然后我们需要计算待分类点与训练样本中每一个点的距离，从中选取最小的 k 个点，如果需要实现加权的 K-NN，还需要返回待分类点和 k 个点的距离：

```python
def get_knn(vec, data, k):
    distances = []  # 存储目标点与训练样本中的点的距离
    for i in range(data.shape[0]):  # 与训练样本中的每一个点都需要计算距离
        distance = cal_distance(vec, data[i])
        distances.append(distance)
    distances = np.array(distances)
    idx = np.argsort(distances)[0:k]  # 取出前k个距离最小的点对应的索引
    distances = distances[idx]  # 取出k个点的距离，其余点可以省略
    return distances, idx
```

首先我们实现最简单的 K-NN 算法，在得到最近的 k 个点后，我们进行投票处理，得到对待分类点的预测标签：

```python
def predict_label(test, train, label, k):
    preds = []  # 存储对待分类点预测的标签
    for i in range(test.shape[0]):
        vec = test[i]
        vote = np.zeros(10)  # 投票数组，用于判定k个点中哪类点最多
        pred = list(np.zeros(10))  # 标签为one-hot形式，所以声明一个10×1的全零数组
        distances, idx = get_knn(vec, train, k)  # 计算出k个近邻
        knn_label = label[idx]  # 在训练样本标签中取出k个近邻的标签
        for j in range(knn_label.shape[0]):
            vote[np.nonzero(knn_label[j])] += 1  # 进行投票
        pred[vote.argmax()] = 1
        preds.append(pred)
    return np.array(preds)
```

基于距离加权的 K-NN 算法 当样本数类别间分布不均时，通常我们需要进行基于距离加权的 K-NN 分类，本实验中作为拓展，同样也进行了实现，具体过程如下:

```python
def predict_label_weighted(test, train, label, k):
    preds = []  # 存储对待分类点预测的标签
    for i in range(test.shape[0]):
        vec = test[i]
        vote = np.zeros(10)  # 投票数组，用于判定k个点中哪类点最多
        pred = list(np.zeros(10))  # 标签为one-hot形式，所以声明一个10×1的全零数组
        distances, idx = get_knn(vec, train, k)  # 计算出k个近邻
        knn_label = label[idx]  # 在训练样本标签中取出k个近邻的标签
        for j in range(knn_label.shape[0]):
            if distances[j] == 0:
                vote[np.nonzero(knn_label[j])] += 0
            else:
                vote[np.nonzero(knn_label[j])] += 1/distances  # 基于距离加权的投票
        pred[vote.argmax()] = 1
        preds.append(pred)
    return np.array(preds)
```

至此，关于 K-NN 分类算法部分的程序就结束了，接下来我们进行交叉验证部分的分析，本实验中以训练样本标号与 10 的模作为分割训练集和测试集的依据，从而满足了 10 折交叉验证所需要的要求:

```python
def train_valid_split(data, label, fold, idx):
    data = list(data)
    label = list(label)
    train_data = []
    train_label = []
    valid_data = []
    valid_label = []
    for i in range(data.__len__()):
        if i%fold == idx:  # 采用取模的方式来进行验证集的划分
            valid_data.append(data[i])
            valid_label.append(label[i])
        else:
            train_data.append(data[i])
            train_label.append(label[i])
    return np.array(train_data), np.array(train_label), \
           np.array(valid_data), np.array(valid_label)
```

然后采用 10 折交叉验证的方式计算出所设 k 值对应的平均误差：

```python
def cross_valid(data, label, fold, k):
    losses = []
    for i in range(fold):  # 进行多折交叉验证
        train_data, train_label, \
        valid_data, valid_label = train_valid_split(data, label, fold, i)
        pred_label = predict_label(valid_data, train_data, train_label, k)
        loss = 1 - metrics.accuracy_score(valid_label, pred_label)
        losses.append(loss)
    return np.mean(losses)  # 返回交叉验证计算出的平均误差
```

然后在多个 k 值中选取出误差最小的 k 值：

```python
def select_k(data, label):
    losses = []
    for i in range(1, 21):  # 所选择的k值范围为1-20
        loss = cross_valid(data, label, 10, i)  # 进行10折交叉验证
        losses.append(loss)
        print(i, "NN loss is ", loss)
    return np.argmin(losses) + 1, losses  # 返回loss值最小的k值
```

为做比较，我在主函数部分调用了 sklearn 中的 K-NN 模型对测试数据进行了分类：

```
pred_label = predict_label(test_data, train_data, train_label, best_k)
loss = 1 - metrics.accuracy_score(test_label, pred_label)
pred_weighted = predict_label_weighted(test_data, train_data, train_label, best_k_weighted)
loss_weighted = 1 - metrics.accuracy_score(test_label, pred_weighted)
print("my simple knn loss:", loss)
print("my distance-weighted knn loss:", loss_weighted)

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(train_data, train_label)
pred_sklearn = knn.predict(test_data)
loss_sklearn = 1 - metrics.accuracy_score(test_label, pred_sklearn)
print("sklearn knn loss: ", loss_sklearn)
```

最后，在 k 值为 1-20 的范围内，我调用 matplotlib 包中的方法绘制了三种 K-NN 算法的 loss 值变化折线：

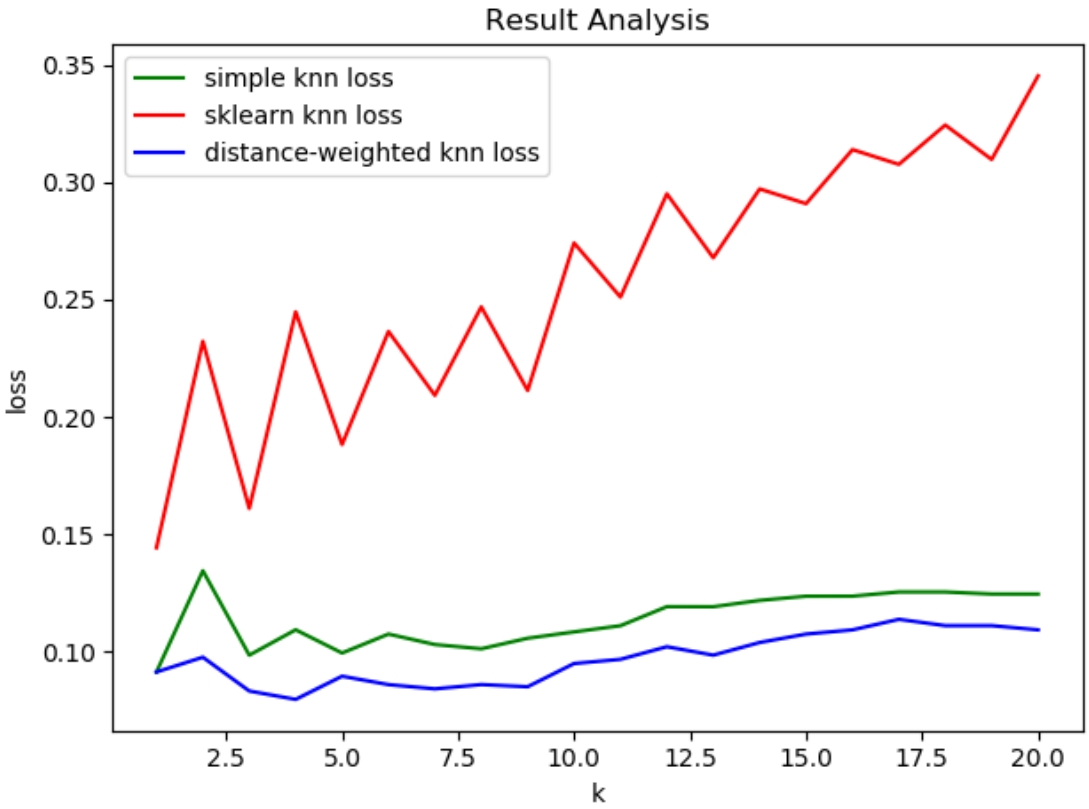
```
k = [x for x in range(1, 21)]

plt.title('Result Analysis')
plt.plot(k, losses, color='green', label='simple knn loss')
plt.plot(k, losses_sklearn, color='red', label='sklearn knn loss')
plt.plot(k, losses_weighted, color='blue', label='distance-weighted knn loss')
plt.legend()  # 显示图例

plt.xlabel('k')
plt.ylabel('loss')
plt.show()
```

3.3 实验结果分析

基于经验，实验中我选择的 k 值范围为 1-20，三种 K-NN 算法在 10 折交叉验证的情况下的平均误差变化折线如下图所示：



从图中可以看出，基于加权的 K-NN 算法在三者中表现最为优异，虽然运行的速度方面与 sklearn 仍有差距，但我所实现的 K-NN 算法都在精确度方面比 sklearn 中的模型更有竞争力。

4 总结

本次实验我从数据的导入开始，完成了一个简单的机器学习项目，初步领略了机器学习的无穷魅力，并在实现各种 K-NN 算法的过程中感受到了理论学习与工程实践之间的差距，同时也收获良多。

参考文献

[1] Thomas M. Mitchell. 1997. Machine Learning (1 ed.). McGraw-Hill, Inc., New York, NY, USA.