实验一：k-NN手写体识别

姓名：孙铭

学号：1711377

专业：计算机科学与技术

日期：2020/03/01

· 摘要

本次实验内容为使用k-NN模型实现手写体识别算法，根据本次实验任务要求，个人完成情况如下。

- 初级部分，独立完成基于k-NN模型的手写体识别算法，没有使用课程提供的代码。且分别计算出k取1到10之间任意整数时，训练集和测试集在模型上的精度表现结果。同时优化代码复杂度。
- 中级部分，实现k折交叉验证算法以选择合适k值，可以利用`k_fold`参数进行折数的调整。当`k_fold=10`时，算法为十折交叉验证；当`k_fold=1`时，算法为留一法。此外，还使用了python中的`sklearn`机器学习包进行了k-NN算法的实现，并将个人实现的k-NN模型同`sklearn`实现的k-NN模型，在训练集和测试集上的精度表现作对比。
- 高级部分，使用python中的`matplotlib`包进行了实验结果的图像绘制。绘制的图像有：不同k值下模型交叉验证结果对比图、不同k值下两种k-NN分类器精度表现对比图。

接下来将分别从开发环境、数据处理、模型构建、k折交叉验证、sklearn实现k-NN、实验结果分析几个角度详细展开介绍。

· 开发环境

本次作业使用语言版本为`Python 3.6.5 64bit`，处理器为`Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80GHz`，项目工程文件如下。

```
knn.py  # 完整实现的k-NN分类器
sklearn_knn.py  # 使用sklearn包实现的k-NN分类器
drawing.py  # 绘制图像
data/semeion_test.csv   # 测试集
data/semeion_train.csv  # 训练集
picture/Figure_1    # 实验一结果图
picture/Figure_2    # 实验二结果图
```

项目实现过程中用到的包及作用如下。

```
import csv  # 处理csv格式文件
import numpy as np  # 数据处理
from collections import Counter # 统计数据出现的频数
from sklearn import neighbors   # sklearn中的分类器
import matplotlib.pyplot as plt # 绘制图像
```

· 数据处理

本次实验训练集为`semeion_train.csv`，测试集为`semeion_test.csv`。每行数据有256个浮点数和10个整型标签数。256个浮点数代表16*16像素区域内的笔迹情况。1为像素点有笔迹，0为像素点无笔迹。10个整型数字，若将其视为一个数组，则为标为1的数对应的索引值即为该结构化数据的标签值。

读取数据

首先，需要从`.csv`文件中读取数据。`csv`是逗号分隔值文件格式，文件中的内容以纯文本形式存储。即从文件中读取的数据是一个字符串序列，因此，需要将字符串序列转换成浮点型数组并保存，以便于后续运算。实现代码如下。

```
csv_reader = csv.reader(open(filename, encoding='utf-8'))
data = []
for line in csv_reader:
    item = line[0].strip().split(' ')
    # 拼接数据和标签
    data.append(item[:-10] + [item[-10:].index('1')])
data = np.array(data, dtype=float)
```

代码中，使用了csv包进行`.csv`格式文件的读取，利用`strip()`函数移除每行字符串首尾的空格，利用`split(' ')`将空格前后的字符串切割。在拼接数据和标签时，利用列表切片的方式。对于标签的整合，使用`item[-10:].index('1')`的方法，直接返回为1位的数组索引。最后利用`numpy.array`方法将数据以浮点数的形式保存在二维矩阵中。

数据归一化

接下来，需要对数据进行归一化。尽管可以看出，对于由0，1构成的数据集而言，归一化前后的结果应该是一致的。但考虑到数据归一化在整个机器学习领域数据处理过程中都是不可或缺的一环，因此不考虑本次实验数据集的特殊性，我还是将归一化的方法实现。

常用的归一化方法有两种，一种是Min-Max标准化(Min-Max Normalization)，另一种是Z-score标准化。这里我选择的标准化方法是前者，即Min-Max标准化。

Min-Max标准化，也称离差标准化，是对原始数据的线性变换，使结果映射到[0, 1]之间，转换函数为：

$$x^* = \frac{x - min}{max - min}$$

其中，max为样本数据的最大值，min为样本数据的最小值。

实现代码如下。

```
# Min_Max Normalization
_data = data[:, :-1]
_min = _data.min(1).reshape(-1, 1)
_max = _data.max(1).reshape(-1, 1)
_data = (_data - _min) / (_max - _min)
data = np.concatenate((_data, data[:, -1].reshape(-1, 1)),
axis=1) # 按列拼接，默认axis=0时按行拼接
```

值得指出，代码中最后一行是将归一化后的原始数据同标签拼接起来，以便于将data作为函数的返回值返回。

· k-NN模型构建

首先，简要说明一下k-NN算法的原理。

k-NN(K-Nearest Neighbor, KNN)算法的核心思想是，若一个样本在特征空间中的k个最临近样本中的大多数属于一个类别，则该样本也属于这个类别。显而易见，k-NN算法的三要素为k-means问题（即k值的选取）、距离度量的方式以及分类决策规则。

首先，解决距离度量的方式。我们知道常见距离度量方式有三种：欧氏距离、曼哈顿距离、闵可夫斯基距离。欧式距离和曼哈顿距离分别是闵可夫斯基距离在p=2、p=1时的特例。这里我选择的距离是欧式距离，公式如下。

$$d(x,y)=\sqrt{\sum_{k=1}^{n}(x_k-y_k)^2}$$

实现的代码如下。

```
# Euclidean Distance
def euclidean_dist(data_1, data_2):
    return np.linalg.norm(data_1 - data_2, ord=2, axis=1)
```

对于分类决策规则，我选择的策略即“少数服从多数”，取前k个样本的标签众数。这里存在的问题是当集合中有多个标签众数相同时，该选择哪一个作为最后返回的标签。这里我的筛选方式非常简单，即选择集合中含有最小欧氏距离标签的标签组。

至于k-means问题，我通过n次k折交叉验证以确定最后选择的近邻数k。这一部分将在报告后文分析。

下面给出k-NN模型构建代码。

```
def knn_model(train, test, k):
    acc = 0
    for _test in test:
```

```
        ed = euclidean_dist(train[:, :-1], _test[:-1])
        # 返回升序排列的索引值，并取前k个
        ed_index = np.argsort(ed)[:k]
        # 提取标签
        k_labels = [train[item][-1] for item in ed_index]
        # 统计众数，两元素众数相同时返回排在前面的元素
        label = Counter(k_labels).most_common(1)[0][0]
        if label == _test[-1]:
            acc += 1
    rate = acc / len(test)
    return rate
```

关于代码的解释已在注释中说明，此处不再赘述。

· k折交叉验证

接下来，需要着手解决前文提到的k-means问题。

这里，我利用了k折交叉验证(k-fold cross validation)算法。简单来说，原理如下。

> 1. *将全部训练集S分成k个不相交的子集，假设S中的训练样例个数为m，那么每一个子集有m/k个训练样例，相应的子集称为{s1, s2, ..., sk}*
> 2. *每次从分好的子集中选出一个作为测试集，其它k-1个作为训练集*
> 3. *根据训练集训练出模型或假设函数，并将该模型放置在测试集上，得到分类器正确率*
> 4. *计算k次求得分类率的平均值，作为该模型或者假设函数的真实分类率*

算法实现代码如下。

```
# times:循环次数，k_fold:折数
def k_fold_cross(times, k_fold, train, k):
    rate = []
    for i in range(0, times):        # 进行i次k折交叉验证
        np.random.shuffle(train)     # 随机打乱
        p = len(train) / k_fold      # 子集大小（向下取整）
        # 每个子集大小
        subset_num = [p for j in range(0, k_fold)]
        remainder = len(train) - int(p) * k_fold
        for j in range(0, remainder):
            subset_num[i] += 1
        subset = []
        count = 0
        for j in range(0, k_fold):  # 数据集切片
            subset.append(train[int(count): \
                                int(count+subset_num[j]), :])
            count += subset_num[j]
        # k折交叉验证，对于k组数据，1组test set，k-1组train set
```

```
        for j in range(0, len(subset)):
            _rate = 0
            for h in range(0, len(subset)):
                if (j + h) < len(subset):
                    _rate += knn_model(subset[j+h], \
                             subset[j], k)
                elif (j + h) >= len(subset):
                    _rate += knn_model(subset[j+h-  \
                             len(subset)], subset[j], k)
            rate.append(_rate / k_fold)
    rate = np.array(rate)
    res = np.sum(rate) / len(rate)
    return res
```

· sklearn实现k-NN

在利用sklearn包实现k-NN之前，首先需要安装sklearn。

```
pip install --user sklearn
```

对于k-NN分类器的实现，我们需要调用的包为：

```
from sklearn import neighbors
```

需要用到的几个函数及说明如下。

首先是`neighbors.KNeighborsClassifier()`，该函数的声明方式为：

```
neighbors.KNeighborsClassifier(n_neighbors=5,
weights='uniform', algorithm='auto', leaf_size=30,p=2,
metric='minkowski', metric_params=None, n_jobs=1)
```

`n_neighbors` 是用来确定多数投票规则里的k值，也就是在点的周围选取k个值z作为总体范围。

`weights` 这个参数的作用是在进行分类判断的时候给最近邻的点加上权重，默认值`uniform`，即等权重。在此情况下可以使用多数投票规则来判断输入实例的类别预测。还有一个选择是`distance`，按照距离的倒数给定权重。此外，用户还可以自己设定权重。由于本此实验没有涉猎，因此后两种参数意义不再赘述。

`algorithm` 是分类时采取的算法，有 `{'auto','ball_tree','kd_tree', 'brute'}`，一般情况下选择`auto`就可以，它会自动进行选择最合适的算法。

`p` 在机器学习系列中，我们知道`p=1`时，距离方法定义为曼哈顿距离，`p=2`时定义为欧几里得距离。默认值为2。

其次，我需要用到`fix(X, Y)`函数。该函数的作用是对数据和标签进行拟合。X代表数据、Y代表标签，在实际使用过程中需要注意的问题是X和Y的维度特征数量需要一致。

最后，我们用到`score(X, Y)`函数，该函数的作用是对数据集X在Y标签下的预测情况打分，返回结果为分类器正确率。

实现代码如下。

```
def sklearn_knn():
    # 加载数据
    test = knn.load_data('data/semeion_test.csv')
    train = knn.load_data('data/semeion_train.csv')
    # 数据集切片
    train_data, train_labels = train[:, :-1], train[:, -1]
    test_data, test_labels = test[:, :-1], test[:, -1]
    # 初始化
    k_max = 10
    train_accuracy = np.empty(k_max)
    test_accuracy = np.empty(k_max)

    for k in range(0, k_max):
        sknn = neighbors.KNeighborsClassifier(
                                n_neighbors=k+1)
        sknn.fit(train_data, train_labels)
        train_accuracy[k] = sknn.score(train_data,
                                       train_labels)
        test_accuracy[k] = sknn.score(test_data,
                                      test_labels)
    return train_accuracy, test_accuracy
```

· 实验结果分析

实验一

对于n次十折交叉验证，为了进一步探索在不同循环次数下，准确度随近邻值k的变化关系。我设计了相应的实验，记录当循环次数`cycles=1,2,5,10`时，k与准确度之间的关系。实验代码如下。

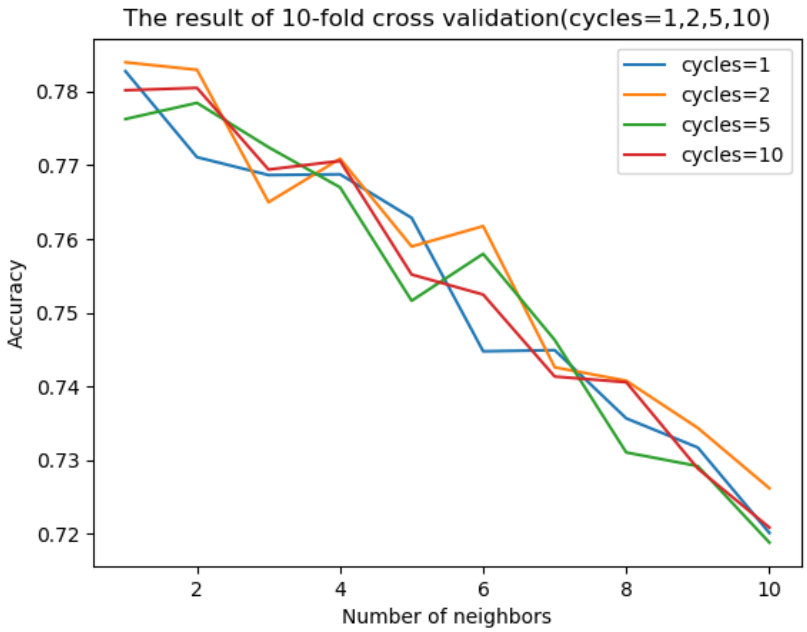
```
# 参数赋值
times = cycles    # k折交叉验证循环次数
k_fold = 10 # k折交叉验证折数
k_max = K  # knn中的k近邻数取值上限
k_res = []    # 最后选取的合适的k值

# 交叉验证选取k值
rate_res = []
for k in range(1, k_max + 1):
    rate = k_fold_cross(times, k_fold, train, k)
```

```
        rate_res.append(rate)
        k_res.append(k)
    # 根据选择的k值，对测试集进行计算
    k_res_max = k_res[rate_res.index(max(rate_res))]
    knn_res = knn_model(train, test, k_res_max)
```

图像绘制代码请参见drawing.py文件，这里不再赘述。

关于不同k值下分类器十折交叉验证结果(cycles=1,2,5,10)的实验结果图如下。



值得指出的一点是，在10折交叉验证过程中，由于对数据集进行10折随机划分，因此每次实验的结果可能存在不同程度浮动。但我的目的是为了观测到一些普遍的规律，而非研究随机划分数据集带来的不可控浮动。

从图像中，可以发现，在相同循环次数下，分类器整体准略率与近邻值k呈负相关；此外，随循环次数的递增，可以观测到图像在不断趋于稳定，与直线越来越接近。因此，对于本模型而言，k近邻值取1到3之间是合理的，在硬件设备允许的情况下，增大交叉验证次数可以提升k值选取的精度。

实验二

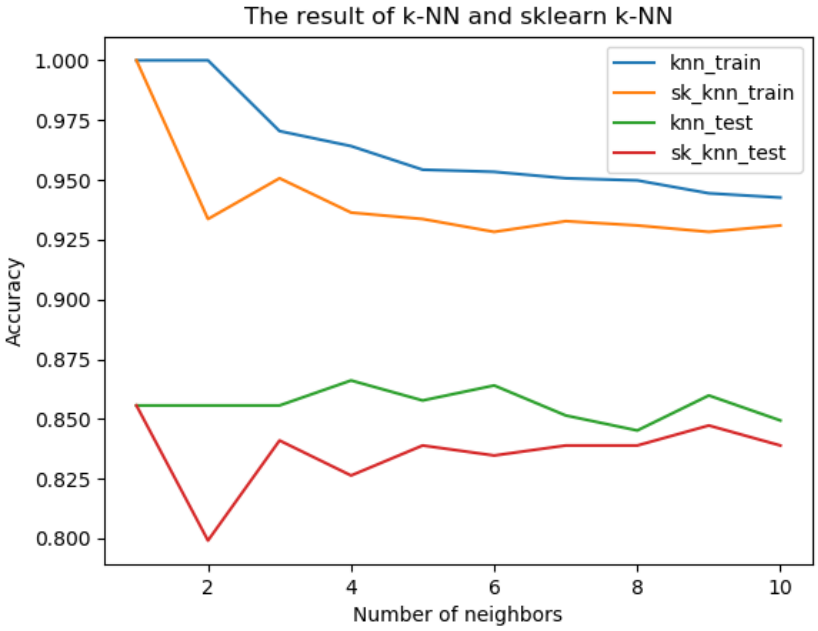
为了比较我自己实现的k-NN分类器同用sklearn实现的k-NN分类器在分类效果上的优劣程度，我设计了一组对比实验，分别从训练集-训练集和训练集-测试集的方式进行精度统计。

sklearn实现的k-NN代码已在前文给出，代码中包含了本对比实验数据的提取。对于自己实现的k-NN分类器的实验数据获取部分，代码如下。

```
# 不通过交叉验证，直接采用grid search对测试集进行准确率分析
def knn_1(K):
    # 加载数据
    test = load_data('data/semeion_test.csv')
    train = load_data('data/semeion_train.csv')
```

```
    # 初始化参数
    k_max = K
    train_acc = np.empty(k_max)
    test_acc = np.empty(k_max)
    # grid search
    for k in range(0, k_max):
        train_acc[k] = knn_model(train, train, k+1)
        test_acc[k] = knn_model(train, test, k+1)
    return train_acc, test_acc
```

关于不同k值下两种分类器准确度结果的比较，如下图所示。



图像中，前缀`knn`代表我自己实现的k-NN分类器，前缀`sk_knn`代表使用机器学习包sklearn实现的分类器。观察图像，不难看出，无论是对于训练集还是测试集，我自己实现的k-NN分类器的分类效果**明显要优于**使用sklearn所实现的k-NN分类器的效果（骄傲.jpg）。

以上是本次实验报告内容。谢谢！
