

机器学习第一次作业 机器学习平台python和matlab的熟悉

1 问题描述

1 用python或者matlab编写一个KNN分类器

- 训练集为semeion_train.csv
- 测试集为semeion_test.csv
- 计算在测试集上的错误率 ($k = 1\ 3\ 5\ 7$)

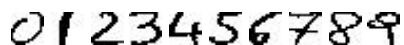
2 选做

- 在训练集上划分一个交叉验证集（可以是训练集数据的20%左右），利用交叉验证选择k值
- 画一个以k值为x轴，交叉验证集错误率为y的曲线

3 本次实验的简要介绍

- 实验内容

本次实验使用kNN算法实现手写数字的识别。数据有256个特征值，代表了一个16*16的位图的像素值，0为无像素，1为存在像素。利用python PIL做出其中各个数字的典型图像如下所示：



- kNN算法简介

kNN算法是一种监督学习算法。假设给定一个训练数据集，其中的实例类别已经确定。分类是对于新的类别，根据其最相近的k个邻居的类别，通过多数表决的方式进行预测。利用训练集对特征空间进行分类划分，并作为其分类的模型。

2. 解决方法

1 解决思路

- 计算待分类点与已知类别的点之间的距离
- 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
- 出现次数相同且最高的类别，按照距离远近进行预测分类

2 基本理论

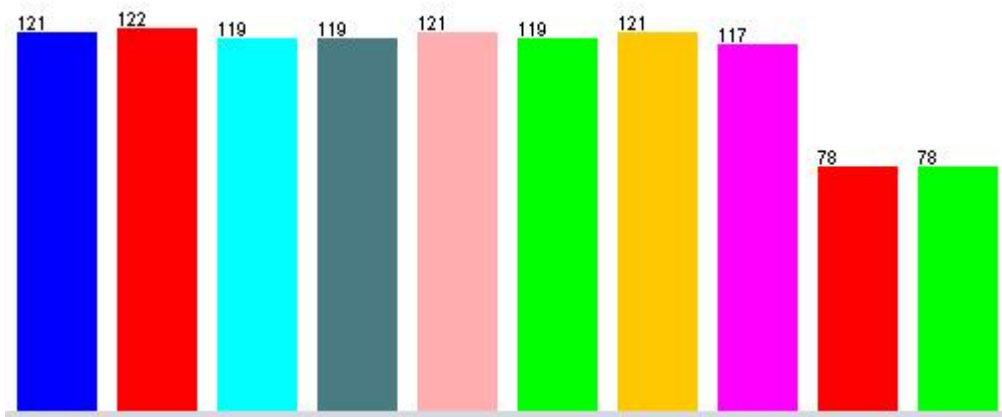
- 距离的度量
 - 本次实验采用欧式距离
- 分类决策标准
 - 多数表决

- 加权统计
- 交叉验证
 - 按照题目进阶要求，采用交叉验证的方式选取最好的k值

3 实验及分析

1 对数据的简单分析

在完成实验之前，利用weka对数据进行了简单的分析，如图所示：



观察图像可以看出，数字8和数字9在训练集中出现的次数与其他数字出现的比例大概为2:3。所以在算法中，我增加了数字8和数字9的权重，以减弱因其数量较少造成的偏差。

2 代码展示

```
import numpy as np
import matplotlib as mpl

mpl.use('Agg')
import random as rd
import matplotlib.pyplot as plt
from PIL import Image

# 导入数据
def load_file(filename):
    x = open(filename)
    array_of_lines = x.readlines()
    number_of_lines = len(array_of_lines)
    return_mat = np.zeros((number_of_lines, 266))
    class_label_vector = []
    index = 0
    for line in array_of_lines:
        line = line.strip()
        list_from_line = line.split(' ')
        return_mat[index, :] = list_from_line[0:266]

        class_label_vector.append(int(list_from_line[-1]))
        index += 1
```

```
print(type(return_mat))
return return_mat

# 返回未排序的结果
def get_distance(data1, data2):
    distance_list = []
    for index in range(data2.shape[0]):
        vec = data2[index, :]
        distance_list.append(euclidean(data1, vec))
    return distance_list

# 计算向量之间的距离
def euclidean(v1, v2):
    sum = 0.0
    for i in range(len(v2)):
        if v1[i] != v2[i]:
            sum += 1
    return sum

# 取距离最近的前k个值，并记录其对应的下标
def knn(distance_list, k):
    # 使用numpy中的函数argsort()实现（猜想该函数的复杂度较高，所以自己写排序函数）
    # temp = np.argsort(distance_list)
    # temp_res = []
    # for i in range(k + 1):
    #     temp_res.append(temp[i])

    temp_res = list(map(distance_list.index, heapq.nsmallest(k+1, distance_list)))
    # ----- 原方案 -----
    # temp_res = np.array(range(0, k+1)).reshape(1,k+1)
    #
    # for i in range(k+1):
    #     temp_res[0][i] = 256
    #
    # for i in range(len(distance_list)):
    #     for j in range(k+1):
    #         if distance_list[i] < temp_res[0][j]:
    #
    #             for l in range(k - j):
    #                 temp_res[0][k - l] = temp_res[0][k - l - 1]
    #                 temp_res[0][j] = i
    #                 break

    return temp_res

# 分析得到的k个值，确定包含最多的类别，对于相同的类别，按照距离进行判断
def get_result(data_result, temp_res):
    res = []
    real = -1
```

```
for i in range(len(temp_res)):
    temp = data_result[temp_res[i], :]
    for j in range(len(temp)):
        if temp[j] == 1:
            if i != 0:
                res.append(j)
            else:
                real = j
            break

res_t = []
for i in range(len(res)):
    res_t.append(res[i])
    res_t.append(res[i])
    if res[i] == 8 or res[i] == 9:
        res_t.append(res[i])

count = np.bincount(res_t)
result = np.argmax(count)
return result, real

# 原始数据数字图形的绘制, 在/im/origin文件夹下
def draw_pic(train_set, result_set):
    pic = np.array(range(0, 256)).reshape(16, 16)
    res_e = 0
    for j in range(train_set.shape[0]):
        temp_train = train_set[j, :]
        temp_res = result_set[j, :]
        for i in range(len(temp_train)):
            pic[int(i / 16)][i % 16] = 255 - temp_train[i] * 255
        for k in range(len(temp_res)):
            if temp_res[k] == 1:
                res_e = k

        new_im = Image.fromarray(pic)
        path = r'E:\工作学习\大三下\机器学习\project\kNN_hmw\im\origin\_origin_' +
str(res_e) + '_' + str(j) + '.jpg'
        new_im = new_im.convert('RGB')
        new_im.save(path)

# 识别结果与原始数据的对比, 在/im/recognize文件夹下
def pic_rec_ori(raw_index, train_set, res_index, real, res):
    pic = np.array(range(0, 16 * 34)).reshape(16, 34)
    temp_ori = train_set[raw_index, :]
    temp_res = train_set[res_index, :]

    for i in range(len(temp_ori)):
        pic[int(i / 16)][i % 16] = 255 - temp_ori[i] * 255
        pic[int(i / 16)][16] = 255
        pic[int(i / 16)][17] = 255
        pic[int(i / 16)][i % 16 + 18] = 255 - temp_res[i] * 255
```

```

new_im = Image.fromarray(pic)
path = r'E:\工作学习\大三下\机器学习\project\kNN_hmw\im\recognize\_ori_' \
      + str(real) + '_rec' + str(res) + '_' + str(raw_index) + '.jpg'
new_im = new_im.convert('RGB')
new_im.save(path)

# 柱形图绘制函数
def plain_rect(data_num, k):
    fig = plt.figure()
    fig.tight_layout()
    font_size = 10
    fig_size = (8, 6)

    names = (u'Accurate', u'Inaccurate')
    subjects = (u'0', u'1', u'2', u'3', u'4', u'5', u'6', u'7', u'8', u'9')
    scores = ((data_num[0][0], data_num[1][0], data_num[2][0], data_num[3][0],
data_num[4][0],
                data_num[5][0], data_num[6][0], data_num[7][0], data_num[8][0],
data_num[9][0]),
              (data_num[0][1], data_num[1][1], data_num[2][1], data_num[3][1],
data_num[4][1],
                data_num[5][1], data_num[6][1], data_num[7][1], data_num[8][1],
data_num[9][1]))

    mpl.rcParams['font.size'] = font_size
    mpl.rcParams['figure.figsize'] = fig_size
    bar_with = 0.30

    index_u = np.arange(len(scores[0]))
    rects1 = plt.bar(index_u, scores[0], bar_with, color='#0072BC',
label=names[0])
    rects2 = plt.bar(index_u, scores[1], bar_with, color='#ED1C24',
label=names[1])

    plt.xticks(index_u + bar_with, subjects)
    plt.ylim(ymax=30, ymin=0)

    plt.title(u'Comparisons of Accurate and Inaccurate Numbers of Number' +
str(k+1))
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.03), fancybox=True,
ncol=5)

# 添加数据标签
def add_labels(rects):
    for rect in rects:
        height = rect.get_height()
        plt.text(rect.get_x() + rect.get_width() / 2, height, height,
ha='center', va='bottom')
        # 柱形图边缘用白色填充, 纯粹为了美观
        rect.set_edgecolor('white')

add_labels(rects1)

```

```
add_labels(rects2)

filename_com = 'compare_' + str(k) + '.png'
plt.savefig(filename_com)
plt.show()

if __name__ == '__main__':
    rate = 0.2 # 通过更改rate的值, 可以改变验证集的大小
    N = 30 # 这是k值取值范围, 1 - N

    # 初始数据读入
    data = load_file('semeion_train.data')
    data_train = data[:, 0:255]
    data_result = data[:, 256:266]

    # 绘制数据的图示
    # draw_pic(data_train, data_result)
    # 随机选取一部分作为验证集
    test_set = data_train
    item_to_del = []
    for index_o in range(data_train.shape[0]):
        if rd.random() > rate:
            item_to_del.append(index_o)
    test_set = np.delete(test_set, item_to_del, axis=0)
    print("验证集大小为: ", test_set.shape[0])

    # 计算并输出准确率
    results = []
    for k in range(N):
        # 用于统计正确率
        count = 0
        count_num = np.array(range(0, 20)).reshape(10, 2)
        for i in range(10):
            for j in range(2):
                count_num[i][j] = 0 # 初始化矩阵

        for index in range(test_set.shape[0]):
            row = test_set[index, :]
            temp = data_train
            dis_list = get_distance(row, temp)
            k_res = knn(dis_list, k + 1)
            res_end = get_result(data_result, k_res)
            if res_end[0] == res_end[1]:
                count = count + 1
            # print(res_end[0], "-", res_end[1]) # 每一次的预测结果和准确值
            if index % 30 == 0:
                print("=====", index, "=====")
            # 绘制对比图像
            # if k == 0:
            #     pic_rec_ori(index, data_train, k_res[1], res_end[1], res_end[0])

        # 绘制柱形图的数据统计
        if res_end[0] == res_end[1]:
```

```

        count_num[res_end[0]][0] += 1 # 预测准确
    else:
        count_num[res_end[1]][1] += 1 # 预测不准确

    print(count / test_set.shape[0]) # 打印准确率
    # 用于绘制k的取值与准确率关系的图像
    results.append(count / test_set.shape[0])

    # 绘制柱形图
    plain_rect(count_num, k)
# 绘制折线图
fig = plt.figure()
fig.tight_layout()
X = np.linspace(0, N, N, endpoint=True)
plt.plot(X, results)
plt.savefig('line_plot.png')
plt.show()

# 找到合适的k值，并验证测试集semeion_test.data
a = max(results)
k_val = 0
count_test = 0
for i in range(len(results)):
    if results[i] == a:
        k_val = i

# 测试
test = load_file("semeion_test.data")
for index in range(test.shape[0]):
    row = test[index, :]
    temp = data_train
    dis_list = get_distance(row, temp)
    k_res = knn(dis_list, k_val + 1)
    res_end = get_result(data_result, k_res)
    if res_end[0] == res_end[1]:
        count_test = count_test + 1
    print("raw:", res_end[1], ",pre:", res_end[0])

print("k值: ", k_val+1)
print("准确率: ", count_test/test.shape[0])
print("错误率: ", 1 - count_test/test.shape[0])

```

3 优化

- 欧氏距离的计算
 - 由于数据集中特征值均为0, 1. 计算欧式距离的时候， $(v1 - v2)^2$ 等价于 当 $v1 \neq v2$ 时，距离加一。从而减少了乘法的执行次数
- 排序算法的优化
 - 最初采用全排序找到前k个。优化成为使用堆排序，返回前k个值。代码如下：

```

# 取距离最近的前k个值，并记录其对应的下标
def knn(distance_list, k):
    # 方案一 使用numpy中的函数argsort()实现（猜想该函数的复杂度较高，所以自己写排序函数）
    temp = np.argsort(distance_list)
    temp_res = []
    for i in range(k + 1):
        temp_res.append(temp[i])

    # 方案二 堆排序
    temp_res = list(map(distance_list.index, heapq.nsmallest(k+1, distance_list)))

    # ----- 原方案 -----
    temp_res = np.array(range(0, k+1)).reshape(1,k+1)
    for i in range(k+1):
        temp_res[0][i] = 256

    for i in range(len(distance_list)):
        for j in range(k+1):
            if distance_list[i] < temp_res[0][j]:

                for l in range(k - j):
                    temp_res[0][k - l] = temp_res[0][k - l - 1]
                temp_res[0][j] = i
                break

    return temp_res

```

- 特征权重

- 通过观察训练集的特征，发现其中数字8, 9的个数与其他数字的个数比为2: 3, 所以为了消除个数对其的影响，给数字加上权重，代码中刚开始的处理方式是加权，即乘以1.5，但是由于这样子做的话需要统计每个数字的个数并且再乘以权值之后再次进行比较，实现比较麻烦。所以此处采用等效的方式，如果结果不是8或9，就在有序链表尾添加两个，是8或9就添加3次。然后直接调用pythont提供的优化过性能的api，实现这一功能。代码如下：

```

# 分析得到的k个值，确定包含最多的类别，对于相同的类别，按照距离进行判断
def get_result(data_result, temp_res):
    res = []
    real = -1

    for i in range(len(temp_res)):
        temp = data_result[temp_res[i], :]
        for j in range(len(temp)):
            if temp[j] == 1:
                if i != 0:
                    res.append(j)
            else:

```



```

        real = j
        break

    res_t = []
    for i in range(len(res)):
        res_t.append(res[i])
        res_t.append(res[i])
        if res[i] == 8 or res[i] == 9:
            res_t.append(res[i])

    # 这里data_result中的结果是按照距离排序的，所以在相同的时候，优先返回距离
    小的类别
    count = np.bincount(res_t)
    result = np.argmax(count)
    return result, real

```

4 结果展示

- k 为 1 3 5 7 时的正确率和错误率

| | |
|--------------------------|--------------------------|
| k值: 1 | k值: 3 |
| 准确率: 0.8284518828451883 | 准确率: 0.8138075313807531 |
| 错误率: 0.17154811715481166 | 错误率: 0.18619246861924688 |
| | |
| k值: 5 | k值: 10 |
| 准确率: 0.8305439330543933 | 准确率: 0.7866108786610879 |
| 错误率: 0.16945606694560666 | 错误率: 0.21338912133891208 |

- 进阶要求
 - 交叉验证选取k值

```

rate = 0.2 # 通过更改rate的值，可以改变验证集的大小
N = 30 # 这是k值取值范围，1 - N

# 初始数据读入
data = load_file('semeion_train.data')
data_train = data[:, 0:255]
data_result = data[:, 256:266]

# 绘制数据的图示
# draw_pic(data_train, data_result)
# 随机选取一部分作为验证集
test_set = data_train
item_to_del = []
for index_o in range(data_train.shape[0]):
    if rd.random() > rate:
        item_to_del.append(index_o)

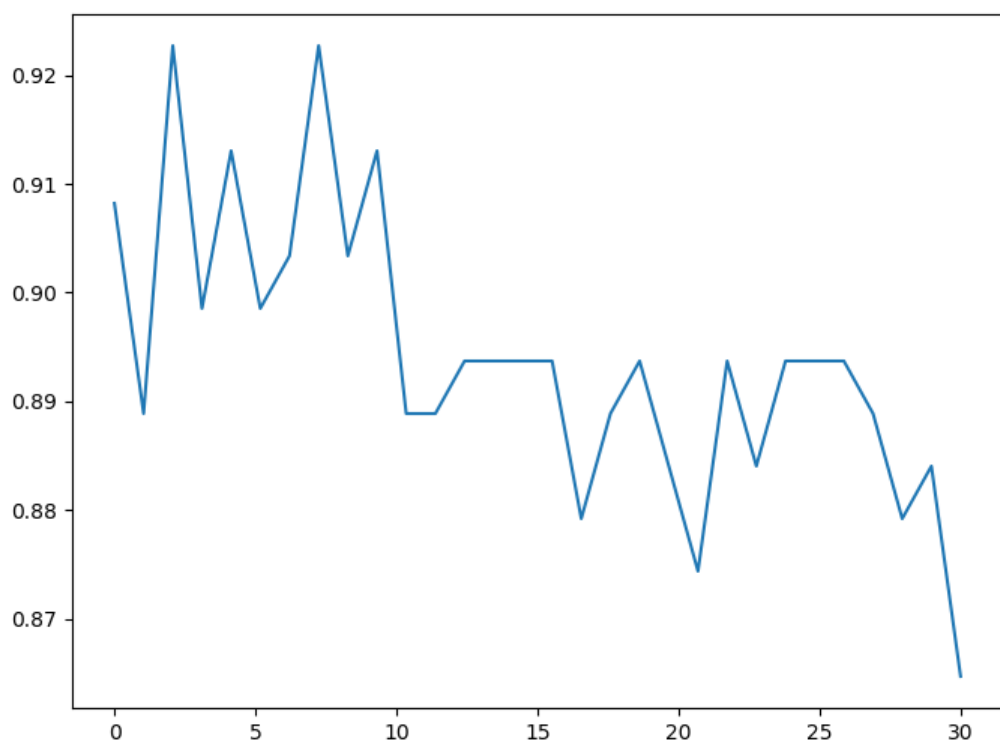
```

```
test_set = np.delete(test_set, item_to_del, axis=0)
print("验证集大小为: ", test_set.shape[0])
```

如上所示，通过更改a的值，可以选取不同大小的验证集。此处选出训练集的百分之二十左右。选取的k值如下：

```
k值: 8
准确率: 0.8326359832635983
错误率: 0.16736401673640167
```

- 不同k下的正确率曲线



注：由于验证集随机选取的，所以可能有不同的结果。