

机器学习作业一

- 姓名：周延霖
- 学号：2013921
- 专业：信息安全

实验要求

题目：基于KNN 的手写数字识别 实验条件：给定semeion手写数字数据集，给定kNN分类算法 实验要求：

1. 基本要求：编程实现kNN算法；给出在不同k值（1，3，5）情况下，kNN算法对手写数字的识别精度（要求采用留一法）
2. 中级要求：与weka机器学习包中的kNN分类结果进行对比
3. 提高要求：将实验过程结果等图示展出

截止日期：10月7日

- 以.ipynb形式的文件提交，输出运行结果，并确保自己的代码能够正确运行
- 发送到邮箱：2120220594@mail.nankai.edu.cn

导入需要的包

```
In [1]: import numpy as np
import operator
from collections import Counter
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
# 为新导入的实现留一法的包，其实最后并没有用到
from tqdm import tqdm
import matplotlib.pyplot as plt
import time
```

导入数据集 semeion

```
In [2]: # 导入数据
def Img2Mat(fileName):
    f = open(fileName)
    ss = f.readlines()
    l = len(ss)
    f.close()
    returnMat = np.zeros((l, 256))
    returnClassVector = np.zeros((l, 1))
    for i in range(l):
        s1 = ss[i].split()
        for j in range(256):
            returnMat[i][j] = np.float(s1[j])
        clCount = 0
        for j in range(256, 266):
            if s1[j] != '1':
```

```

        clCount += 1
    else:
        break
    returnClassVector[i] = clCount
return returnMat, returnClassVector

```

```

In [3]: X,y = Img2Mat('semeion.data')
        np.shape(X), np.shape(y)

```

```

/var/folders/3g/0qh1978d1klcfly3cxll683r0000gn/T/ipykernel_81871/2757158993.
py:12: DeprecationWarning: `np.float` is a deprecated alias for the builtin
`float`. To silence this warning, use `float` by itself. Doing this will not
modify any behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/d
evdocs/release/1.20.0-notes.html#deprecations
        returnMat[i][j] = np.float(s1[j])

```

```

Out[3]: ((1593, 256), (1593, 1))

```

基本要求

编程实现kNN算法；给出不同k值（1，3，5）情况下，kNN算法对手写数字的识别精度（模板中采用的是普通方法分割训练集和测试集，作业中需要用留一法）

```

In [4]: ## KNN算法手动实现
        # def MyKnnClassifier(data_X, data_y, neighbors):
        #     # 生成数据集和测试集
        #     X_train, X_test, y_train, y_test = train_test_split(data_X, data_y, te
        #     trainShape = X_train.shape[0] # 获得训练集的大小
        #     testShape = X_test.shape[0] # 获得测试集的大小
        #     testRes = [] # 存放测试结果
        #     acc = 0 # 计算准确率
        #     for i in range(testShape): # 针对测试集中每一个样本进行预测
        #         # 差异矩阵 = 该样本与训练集中所有样本之差构成的矩阵
        #         testDiffMat = np.tile(X_test[i],(trainShape , 1)) - X_train
        #         sqTestDiffMat = testDiffMat ** 2 # 将差异矩阵平方
        #         # 方差距离为方差矩阵的整行求和，是一个一位列向量
        #         sqTestDiffDis = sqTestDiffMat.sum(axis=1)
        #         testDiffDis = sqTestDiffDis ** 0.5 # 开方生成标准差距离
        #         sortIndex = np.argsort(testDiffDis) # 将标准差距离按照下标排序
        #         labelCount = []
        #         for j in range(neighbors): # 考察k近邻属于哪些类
        #             labelCount.append(y_train[sortIndex[j]][0])
        #         classifyRes = Counter(labelCount) # 把k近邻中最多的那个标签作为分类结果
        #         classifyRes = classifyRes.most_common(2)[0][0]
        #         testRes.append(classifyRes)
        #         if classifyRes == y_test[i]: # 分类正确则将accRate+1
        #             acc += 1
        #         accRate = acc / X_test.shape[0]
        #         print('k={0}时，测试个数为{1} 正确个数为: {2} 准确率为: {3}'.format(neighb
        #         return accRate
        # 以上老师为所写的普通方法

        # KNN算法采用留一法实现
        def MyKnnClassifier(data_X, data_y, neighbors):
            # kf = KFold(n_splits = 1)
            # 总共进行data_X.shape[0]次
            # 总正确个数
            acc_sum = 0
            # 总正确率
            # accRate_sum = 0

```

```

# for train_index, test_index in kf.split(data_X):
for test_index in range(data_X.shape[0]):
    # 下面生成训练集的下标
    train_index = []
    for i in range(data_X.shape[0]):
        if i == test_index:
            continue
        else:
            train_index.append(i)
    # 生成数据集和测试集
    X_train = data_X[train_index]
    X_test = []
    X_test.append(data_X[test_index])
    # np.shape(X_test)
    y_train = data_y[train_index]
    y_test = []
    y_test.append(data_y[test_index])
    # np.shape(y_test)
    # 获得训练集的大小
    trainShape = X_train.shape[0]
    # 获得测试集的大小
    # testShape = X_test.shape[0]
    testShape = 1
    # print(testShape)
    # print(trainShape)
    testRes = [] # 存放测试结果
    acc = 0 # 计算准确率
    for i in range(testShape): # 这里testShape = 1, 之所以用循环是因为不想改
        # 差异矩阵 = 该样本与训练集中所有样本之差构成的矩阵
        testDiffMat = np.tile(X_test[i], (trainShape, 1)) - X_train
        sqTestDiffMat = testDiffMat ** 2 # 将差异矩阵平方
        # 方差距离为方差矩阵的整行求和, 是一个一位列向量
        sqTestDiffDis = sqTestDiffMat.sum(axis=1)
        testDiffDis = sqTestDiffDis ** 0.5 # 开方生成标准差距离
        sortIndex = np.argsort(testDiffDis) # 将标准差距离按照下标排序
        labelCount = []
        for j in range(neighbors): # 考察k近邻属于哪些类
            labelCount.append(y_train[sortIndex[j]][0])
        classifyRes = Counter(labelCount) # 把k近邻中最多的那个标签作为分类
        classifyRes = classifyRes.most_common(2)[0][0]
        testRes.append(classifyRes)
        if classifyRes == y_test[i]: # 分类正确则将accRate+1
            acc += 1

    acc_sum = acc_sum + acc
    # accRateTemplate = acc / X_test.shape[0]
    # accRate_sum = accRate_sum + accRateTemplate

acc = acc_sum
accRate = acc_sum / data_X.shape[0]
print('k={0}时, 测试个数为{1} 平均正确个数为: {2} 平均准确率为: {3}'.format(n,
return accRate

```

实验结果:

```

In [5]: MyKnnClassifier(X, y, 1)
MyKnnClassifier(X, y, 3)
MyKnnClassifier(X, y, 5)

```

k=1时, 测试个数为1593	平均正确个数为: 1459	平均准确率为: 0.9158819836785939
k=3时, 测试个数为1593	平均正确个数为: 1464	平均准确率为: 0.9190207156308852
k=5时, 测试个数为1593	平均正确个数为: 1458	平均准确率为: 0.9152542372881356

Out[5]: 0.9152542372881356

中级要求

模板中与sklearn机器学习包中的kNN分类结果进行对比（作业中需要与weka机器学习包中的kNN分类结果进行对比）

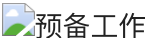
```
In [6]: ## kNN算法sklearn库实现
# def KnnClassifier(data_X, data_y, neighbors, flag=0):
#     X_train, X_test, y_train, y_test = train_test_split(data_X, data_y, te
#     knn = KNeighborsClassifier(n_neighbors=neighbors)
#     knn.fit(X_train, y_train.ravel())
#     print('k={0}时, scikit-learn训练手写体识别的准确率为: {1}'.format(neighbors
#     # 交叉验证
#     if flag == 1:
#         scores = cross_val_score(knn, data_X, data_y.ravel(), cv=10, scoring='
#     return scores.mean()
```

由于本人导入weka包的时候频繁报错，所以转换为直接使用现成的weka工具来进行本次实验，将semeion.data先修改后綴转换为txt文件，再将txt文件通过如下代码转换为csv文件：



```
In [7]: # 将txt转换成csv
import numpy as np
import pandas as pd


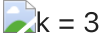
data_txt = np.loadtxt('semeion.txt')
data_txtDF = pd.DataFrame(data_txt)
data_txtDF.to_csv('semeion.csv', index=False)
```


接下来对转换成功的csv文件的后10列进行处理，使其形成10个类别，变成semeion(1).csv文件。

最后用weka工具将semeion(1).csv转换为semeion.arff文件，然后就可以分析了，将文件导入的工具中，初始效果如下图所示：

实验结果：

当 $k = 1$ 时运行工具所得到的结果如下图所示： $k = 1$ 按类别划分的详细精度和混淆矩阵如下图所示：

当 $k = 3$ 时运行工具所得到的结果如下图所示： $k = 3$ 按类别划分的详细精度和混淆矩阵如下图所示：

当 $k = 5$ 时运行工具所得到的结果如下图所示： $k = 5$ 按类别划分的详细精度和混淆矩阵如下图所示：

可以看出不论当 k 为1, 3, 5时，采用留一法的准确率都要高于weka工具所测得的准确率

高级要求

将实验过程结果等图示展出

```
In [8]: scores1 = []
# scores2 = []

# 由于weka包的缘故, 所以weka的图已经在上文中展示, 这里只展示手写的留一法的knn部分

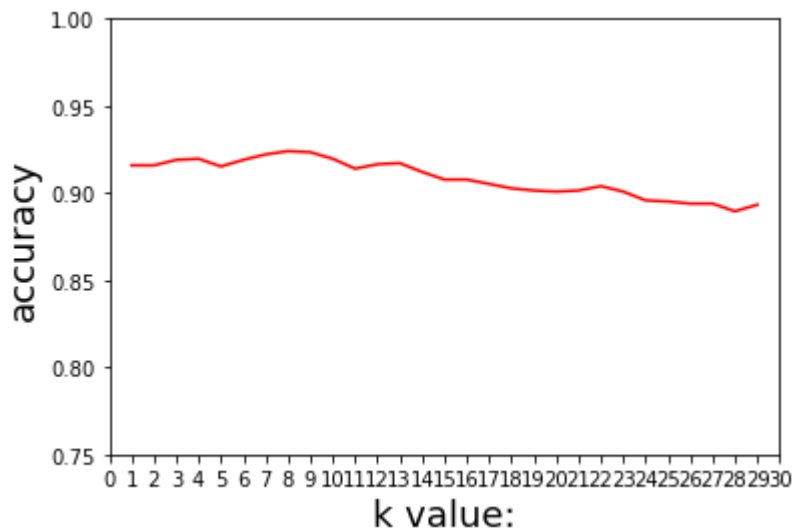
for k in range(1,30):
    score1 = MyKnnClassifier(X, y, k)
    scores1.append(score1)

# for k in range(1,30):
#     score2 = KnnClassifier(X, y, k, 1)
#     scores2.append(score2)
```

k=1时, 测试个数为1593	平均正确个数为: 1459	平均准确率为: 0.9158819836785939
k=2时, 测试个数为1593	平均正确个数为: 1459	平均准确率为: 0.9158819836785939
k=3时, 测试个数为1593	平均正确个数为: 1464	平均准确率为: 0.9190207156308852
k=4时, 测试个数为1593	平均正确个数为: 1465	平均准确率为: 0.9196484620213434
k=5时, 测试个数为1593	平均正确个数为: 1458	平均准确率为: 0.9152542372881356
k=6时, 测试个数为1593	平均正确个数为: 1464	平均准确率为: 0.9190207156308852
k=7时, 测试个数为1593	平均正确个数为: 1469	平均准确率为: 0.9221594475831764
k=8时, 测试个数为1593	平均正确个数为: 1472	平均准确率为: 0.9240426867545511
k=9时, 测试个数为1593	平均正确个数为: 1471	平均准确率为: 0.9234149403640929
k=10时, 测试个数为1593	平均正确个数为: 1465	平均准确率为: 0.9196484620213434
k=11时, 测试个数为1593	平均正确个数为: 1456	平均准确率为: 0.9139987445072191
k=12时, 测试个数为1593	平均正确个数为: 1460	平均准确率为: 0.9165097300690521
k=13时, 测试个数为1593	平均正确个数为: 1461	平均准确率为: 0.9171374764595104
k=14时, 测试个数为1593	平均正确个数为: 1453	平均准确率为: 0.9121155053358443
k=15时, 测试个数为1593	平均正确个数为: 1446	平均准确率为: 0.9077212806026366
k=16时, 测试个数为1593	平均正确个数为: 1446	平均准确率为: 0.9077212806026366
k=17时, 测试个数为1593	平均正确个数为: 1442	平均准确率为: 0.9052102950408035
k=18时, 测试个数为1593	平均正确个数为: 1438	平均准确率为: 0.9026993094789705
k=19时, 测试个数为1593	平均正确个数为: 1436	平均准确率为: 0.901443816698054
k=20时, 测试个数为1593	平均正确个数为: 1435	平均准确率为: 0.9008160703075957
k=21时, 测试个数为1593	平均正确个数为: 1436	平均准确率为: 0.901443816698054
k=22时, 测试个数为1593	平均正确个数为: 1440	平均准确率为: 0.903954802259887
k=23时, 测试个数为1593	平均正确个数为: 1435	平均准确率为: 0.9008160703075957
k=24时, 测试个数为1593	平均正确个数为: 1427	平均准确率为: 0.8957940991839297
k=25时, 测试个数为1593	平均正确个数为: 1426	平均准确率为: 0.8951663527934715
k=26时, 测试个数为1593	平均正确个数为: 1424	平均准确率为: 0.8939108600125549
k=27时, 测试个数为1593	平均正确个数为: 1424	平均准确率为: 0.8939108600125549
k=28时, 测试个数为1593	平均正确个数为: 1417	平均准确率为: 0.8895166352793471
k=29时, 测试个数为1593	平均正确个数为: 1423	平均准确率为: 0.8932831136220967

将手写体识别的留一法的上述所得到的结果用图片进行展示

```
In [9]: plt.xlabel('k value:', fontsize=18)
plt.ylabel('accuracy', fontsize=18)
x_major_locator = plt.MultipleLocator(1)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
plt.xlim(0, 30)
plt.ylim(0.75, 1)
# 普通kNN分类精度
plt.plot(range(1,30), scores1, 'r')
# 这里只展示手写留一法的knn, 原因已在上文中说明
# plt.plot(range(1,30), scores2, 'b')
plt.show()
```



本次实验也到此结束🎉

总结与展望

总结

- 本次是机器学习的第一次实验，在做实验的过程中感受到了一些算法的强大，也通过手写留一法对交叉验证等课堂上所讲述的概念更加的熟悉
- 然后再通过自己对weka工具的探索，虽然最后也没有找到和实现用java代码写，但是对之前上一学年所学习的java课也算有了一定的回顾
- 最后通过使用weka工具成功对knn中k为1，3，5进行了实验，中间在文件格式转换方面也卡了一定的时间，但最后通过助教学长以及同学的帮助也顺利的解决了问题

展望

通过第一次实验，发现自己对机器学习有了更近一步的认识，希望自己能在本学期的课程中学到更多，也希望自己未来能有更好的发展👍