**实验一 KNN**

一、实验目的及要求   
 1、掌握 Python 安装、编程环境搭建的法。

2、熟练掌握 numpy、matplotlib 等基本数学运算、图形显示 Python 包的安装与应用

方法。

3、掌握 KNN 算法的基本原理。

4、掌握利用 numpy 进行文本类数据集的读取、训练数据测试数据拆分的方法

5、掌握 k-folder 交叉验证的实验过程

二、预习要求   
 阅读本实验例程部分，实现基本的 KNN 分类算法，以便能够充分利用实验时间编

程调试。

三、实验设备   
 硬件：PC机。

软件：Python 及相关集成开发环境。

四、实验内容   
利用利用 Python 编程实现基于 KNN 分类算法的 iris 数据分类，具体要求下：

（1）将数据集进行随机分割，分别设置 90%-10%，80%-20%训练-测试数据比例，分

别运行 3 次、计算并打印分类准确率。

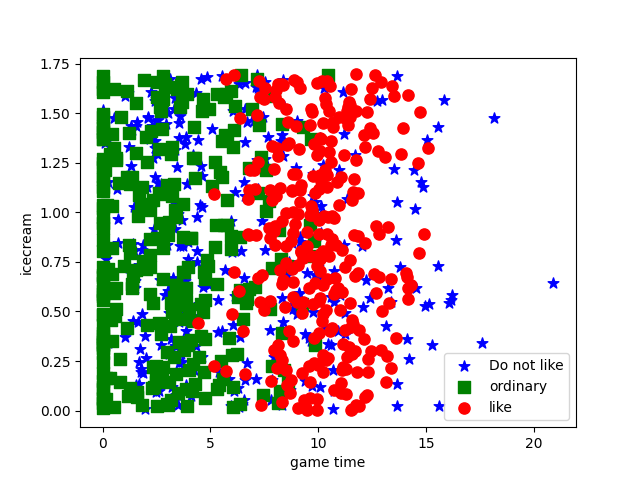
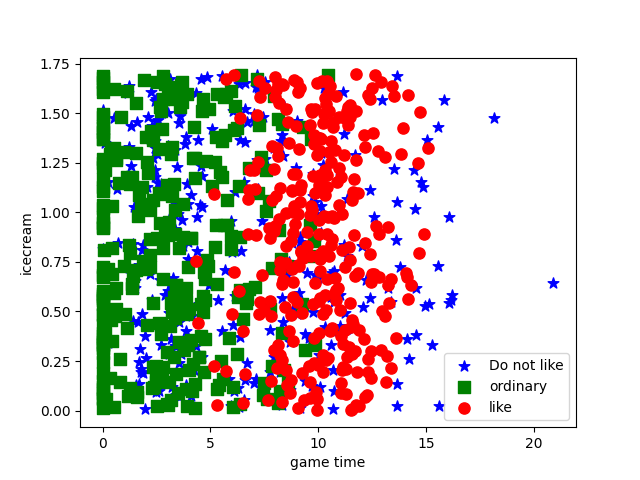
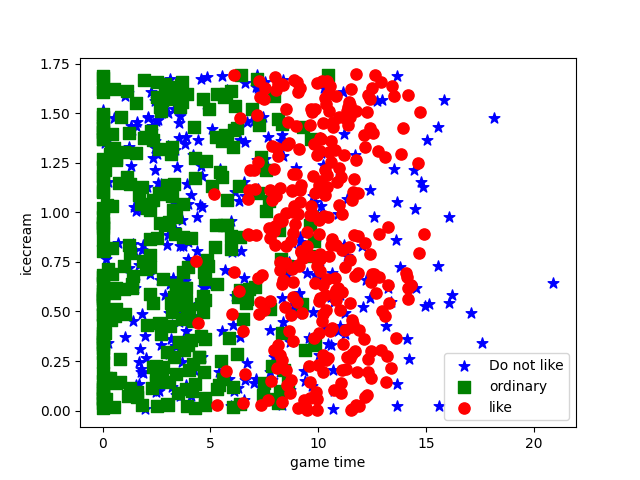
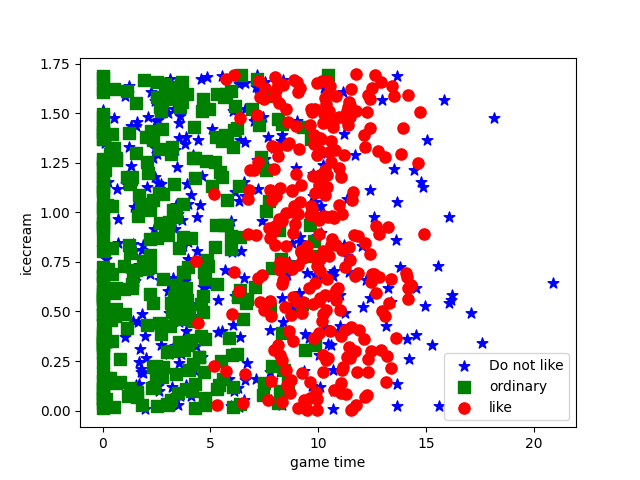
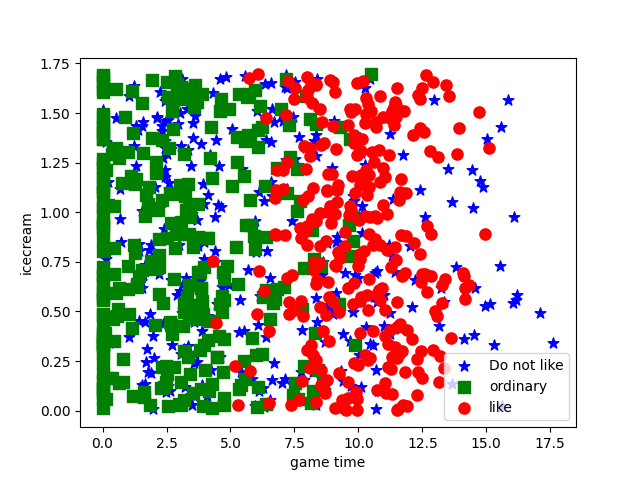
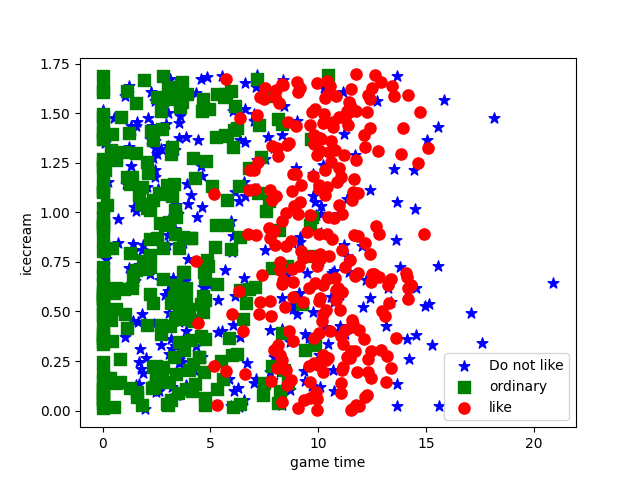
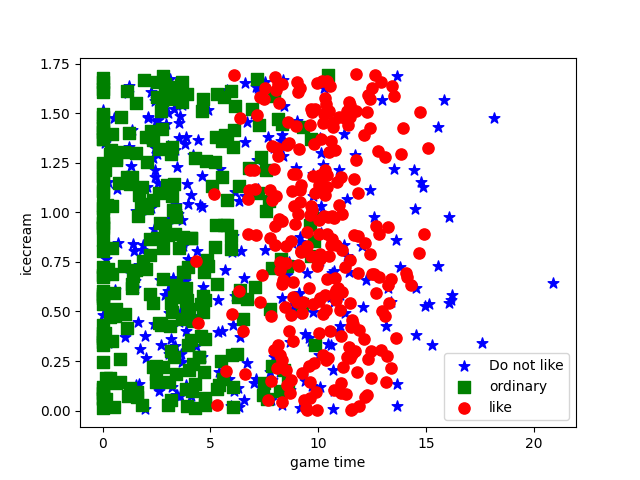
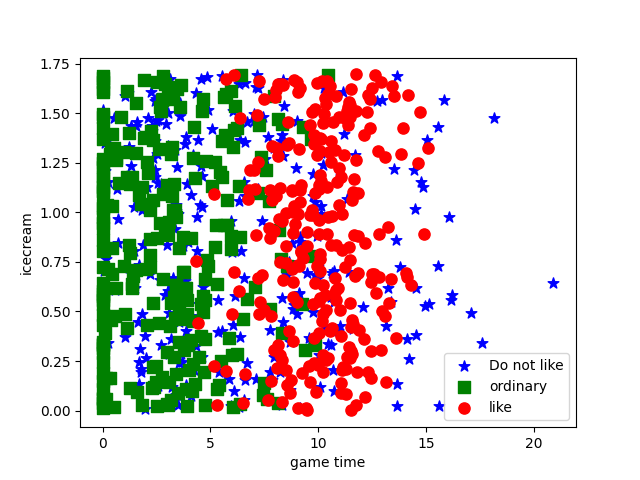
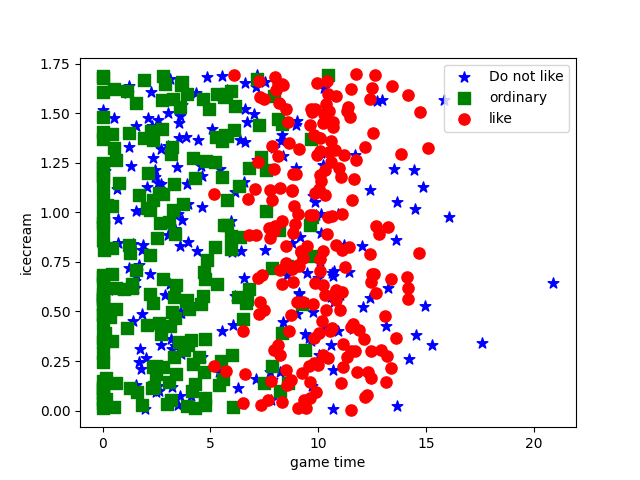
（2）利用 10-folder 和 5-folder 交叉验证的方法进行分类测试，并分别打印准确率

（3）利用 matplotlib 的 scatter 函数，实现分类结果的显示（利用数据的 1、3 维）其

中分类正确的三种类别分别用红、黄、绿、三色显示。分类错误的样本点用黑色显示。

1. 实验报告内容要求   
   1. 列出编写的 python 代码。对主要的语句进行注释

import numpy as np  
import operator  
import matplotlib.pyplot as plt  
**'''  
 trainData - 训练集 N  
 testData - 测试 1  
 labels - 训练集标签  
'''**def knn(trainData, testData, labels, k):  
 *计算训练样本的行数* rowSize = trainData.shape[0]  
 *计算训练样本和测试样本的差值* diff = np.tile(testData, (rowSize, 1)) - trainData  
 *计算差值的平方和* sqrDiff = diff \*\* 2  
 sqrDiffSum = sqrDiff.sum(axis=1)  
 *计算距离* distances = sqrDiffSum \*\* 0.5  
 *对所得的距离从低到高进行排序* sortDistance = distances.argsort()  
   
 count = {}  
   
 for i in range(k):  
 vote = labels[sortDistance[i]]  
 count[vote] = count.get(vote, 0) + 1  
 *对类别出现的频数从高到低进行排序* sortCount = sorted(count.items(), key=operator.itemgetter(1), reverse=True)  
   
 *返回出现频数最高的类别* return sortCount[0][0]  
  
def evaluate\_KNN(train\_datas,train\_labs,test\_datas,test\_labs,K=5):  
 N\_test = test\_datas.shape[0]  
   
 N\_right= 0  
 for i in range(N\_test):  
 det\_lab = knn(train\_datas, test\_datas[i], train\_labs, K)  
   
 if det\_lab == test\_labs[i]:  
 N\_right += 1 *计算准确率* acc = N\_right\*100/N\_test  
 return acc   
  
def evaluate\_KNN\_fold(datas,labs, k\_fold,K=10):  
 N = np.shape(datas)[0]  
 n\_each\_fold = int(N/k\_fold)  
 N\_used = n\_each\_fold\*k\_fold  
  
 index = np.random.permutation(N)[:N\_used]  
  
 index\_fold = np.reshape(index,(k\_fold,n\_each\_fold))  
 acc\_all = []  
 for i in range(k\_fold):  
 index\_test = index\_fold[i,:]  
 sel\_train = [i for i in range(k\_fold)]  
 sel\_train.remove(i)  
 index\_train = index\_fold[sel\_train,:].flatten()  
  
 train\_datas = datas[index\_train]  
 train\_labs = labs[index\_train]  
  
 test\_datas = datas[index\_test]  
 test\_labs = labs[index\_test]  
  
 acc = evaluate\_KNN(train\_datas,train\_labs,test\_datas,test\_labs,K)  
 print(**"acc in fold %d = %.2f %%"**%(i,acc))  
 acc\_all.append(acc)  
   
 return np.average(acc\_all)  
  
def indexSplit(N,train\_ratio):  
 N\_train = int(N\*train\_ratio)  
 index\_random = np.random.permutation(N)  
 index\_train = index\_random[:N\_train]  
 index\_test = index\_random[N\_train:]  
  
 return index\_train,index\_test  
  
  
  
def autoNorm(datas):  
 val\_min = np.min(datas,axis=0,keepdims=True)  
 val\_max = np.max(datas,axis=0,keepdims=True)  
 val\_range = val\_max - val\_min  
  
 norm\_datas = (datas - val\_min)/val\_range  
 return norm\_datas, val\_range, val\_min   
  
  
  
def evaluate\_KNN\_fold\_norm(datas,labs, k\_fold,K=10):  
 pass  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
  
 *数据处理  
 数据读取  
 分为训练集和测试集和  
 一般测试  
 交叉验证测试  
 数据处理  
 第列、每年飞行的里程数  
 第列、玩游戏所消耗时间的百分比  
 第列、每周消耗的冰淇淋公升量* file\_data = **'datingTestSet2.txt'** dic\_labs ={  
 1:**"Do not like"**,  
 2:**"ordinary"**,  
 3:**"like"** }  
  
 *数据读取* datas = np.loadtxt(file\_data,dtype = float, delimiter = **'**\t**'**,usecols=(0,1,2))  
 labs = np.loadtxt(file\_data,dtype = int, delimiter = **'**\t**'**,usecols=(3))  
 N = np.shape(datas)[0]  
  
 *分割成训练集和测试集* index\_train,index\_test = indexSplit(N,train\_ratio=0.9)  
  
 train\_datas = datas[index\_train,:]  
 train\_labs = labs[index\_train]  
  
  
 test\_datas = datas[index\_test,:]  
 test\_labs = labs[index\_test]  
  
 *对训练数据进行正则化* train\_norm\_datas,val\_range, val\_min= autoNorm(train\_datas)  
   
 *利用正则化参数 对测试数据进行正则化* test\_norm\_datas = (test\_datas-val\_min)/val\_range  
  
 *训练数据显示* fig = plt.figure()  
 unique\_labs = np.unique(train\_labs).tolist()  
 list\_color = [**'b'**,**'g'**,**'r'**,**'c'**,**'m'**,**'y'**,**'k'**,**'w'**]  
 list\_marker = [**'\*'**,**','**,**'o'**,**'^'**,**'1'**,**'+'**,**'x'**,**'D'**]  
 for j,lab in enumerate(unique\_labs):  
 index = np.where(train\_labs == lab)[0]  
 x = train\_datas[index,1]  
 y = train\_datas[index,2]  
 plt.scatter(x,y,s= 65,c=list\_color[j],marker=list\_marker[j],label=dic\_labs[lab])  
 plt.legend(loc=**"best"**)  
 plt.xlabel(**"game time"**)  
 plt.ylabel(**"icecream"**)  
 plt.show()  
  
  
 *一般测试* K=5  
 acc = evaluate\_KNN(train\_datas,train\_labs,test\_datas,test\_labs,K)  
 print(**"acc= %.2f%% without data normalization"**%(acc))  
   
 acc = evaluate\_KNN(train\_norm\_datas,train\_labs,test\_norm\_datas,test\_labs,K)  
 print(**"acc= %.2f%% with data normalization"**%(acc))  
  
 *交叉验证测试* acc = evaluate\_KNN\_fold(datas,labs, k\_fold=5,K=5)  
 print(**"final acc = %.2f%%"**%(acc))  
实验结果：



实验结果数据：

0.8  
acc= 82.00% without data normalization  
acc= 94.50% with data normalization  
acc= 78.50% without data normalization  
acc= 95.50% with data normalization  
acc= 79.00% without data normalization  
acc= 95.50% with data normalization  
0.9  
acc= 79.00% without data normalization  
acc= 98.00% with data normalization  
acc= 76.00% without data normalization  
acc= 96.00% with data normalization  
acc= 82.00% without data normalization  
acc= 95.00% with data normalization  
  
  
利用10-folder和5-folder交叉验证的方法进行分类测试，并分别打印准确率  
k\_fold=10  
acc= 73.00% without data normalization  
acc= 96.00% with data normalization  
acc in fold 0 = 80.00 %  
acc in fold 1 = 82.00 %  
acc in fold 2 = 79.00 %  
acc in fold 3 = 81.00 %  
acc in fold 4 = 77.00 %  
acc in fold 5 = 82.00 %  
acc in fold 6 = 86.00 %  
acc in fold 7 = 79.00 %  
acc in fold 8 = 80.00 %  
acc in fold 9 = 74.00 %  
final acc = 80.00%  
  
k\_fold=5  
acc= 81.00% without data normalization  
acc= 96.00% with data normalization  
acc in fold 0 = 81.00 %  
acc in fold 1 = 82.00 %  
acc in fold 2 = 80.00 %  
acc in fold 3 = 80.00 %  
acc in fold 4 = 76.50 %  
final acc = 79.90%