|  |
| --- |
| **计算机专业类课程** |
| **实验报告** |
| **课程名称：数据挖掘与大数据分析**  **学　　院：计算机科学与工程学院**  **专　　业：互联网加**  **学生姓名：李宏洋**  **学　　号：2020080301012**  **指导教师：曾伟** |
|  |
| **日　　期：　　　2022　年　4　月　9　日** |

第三次实验报告目录

[**一、 实验名称：分类算法** 3](#_Toc100502894)

[**二、 实验学时：4** 3](#_Toc100502895)

[**三、 实验内容和目的：** 3](#_Toc100502896)

[**（一）** **实验内容** 3](#_Toc100502897)

[**（二）** **实验目的** 3](#_Toc100502898)

[**四、 实验步骤：** 3](#_Toc100502899)

[**（一）** **KNN算法** 3](#_Toc100502900)

[**（二）** **ANN算法** 4](#_Toc100502901)

[1、 感知机 4](#_Toc100502902)

[2、 BP神经网络 4](#_Toc100502903)

[**（三）** **决策树算法** 5](#_Toc100502904)

[**五、 实验数据及结果分析：** 5](#_Toc100502905)

[**（一）** **KNN算法** 5](#_Toc100502906)

[1、实验数据 5](#_Toc100502907)

[2、实验结果分析 9](#_Toc100502908)

[**（二）** **ANN算法** 10](#_Toc100502909)

[1、感知机 10](#_Toc100502910)

[2、BP神经网络 14](#_Toc100502911)

[**（三）** **决策树算法** 16](#_Toc100502912)

[1、实验数据 16](#_Toc100502913)

[2、实验结果分析 18](#_Toc100502914)

[**六、 实验结论、心得体会和改进建议：** 18](#_Toc100502915)

[**（一）实验结论** 18](#_Toc100502916)

[**（二）心得与建议** 18](#_Toc100502917)

[**七、 代码展示：** 18](#_Toc100502918)

[**（一）** **KNN算法** 18](#_Toc100502919)

[**（二）** **ANN算法** 21](#_Toc100502920)

[**1、** **感知机** 21](#_Toc100502921)

[**2、** **BP神经网络** 22](#_Toc100502922)

[**（三）** **决策树** 24](#_Toc100502923)

**电 子 科 技 大 学**

**实 验 报 告**

**实验三**

1. **实验名称：分类算法**
2. **实验学时：4**
3. **实验内容和目的：**
4. **实验内容**

理解并自己编程实现KNN算法、ANN算法、决策树算法

1. **实验目的**
   1. 进一步掌握python的使用
   2. 理解并编程实现KNN算法、ANN算法、决策树算法
   3. 运用编写的程序进行分类
2. **实验步骤：**
3. **KNN算法**

数据集我们采用鸢尾花数据集。首先定义一个K近邻类。在其中，首先定义一个方法用于获取并初始化数据。我们将数据用x变量储存，标签用y变量储存，并将字符标签数字化为0、1、2。再根据标签将数据集进一步划分，并取各类前30个数据作为训练集，后20个数据作为测试集。

在可视化过程这方面，运用matplotlib.pyplot库进行可视化，考虑到数据维度较高，故而我们只选择sepal length和petal length两个特征，在二维平面上作散点图。

在预测函数中，我们通过计算输入进来的数据与训练集中各点的距离以形成距离矩阵，通过对距离矩阵的对应行进行排序，则我们可以知道对应输入点的前k个最近的近邻的位置与对应标签，取其中最多的标签作为预测类别。

1. **ANN算法**
2. 感知机

首先读入存储于ann\_Perceptron.txt中以空格隔开的数据，前两列是数据，最后一列存储为标签。

接着计算计算 y\*(w\*x+b)的值，这么计算是因为数据集中除了标签全是正数，而且标签的绝对值均为1，所以乘上标签后只需要判断符号即可判定两类数据，同时还不改变数据大小。

然后用梯度下降算法求解w，b。在f [idx]（idx为y的实际值于预测值之积为负的索引）中的元素个数不为零，即分类不完全正确之前，不中止循环。每一次就将w, b分别调整步长乘以y对对应元素梯度。最后就可以求得优化出来的w, b值。

可视化过程仍然用matplotlib.pyplot库进行实现。

1. BP神经网络

这部分的数据直接由程序中进行赋值，因为如果数值较大的话跑出来数据会非常大，不便于报告中展示。故而我们将输入设置为一个行向量，输出设置为一个行向量，中间隐藏层节点数目手动设定。其中隐藏层节点的数目可以通过经验公式求得，m为输入层节点数，n为输出层节点数，a为调节常数。

只设置一层隐藏层的原因是万能逼近定理的存在。1989年Robert Hecht-Nielsen证明了对于任何闭区间内的一个连续函数都可以用一个隐含层的BP网络来逼近，这就是万能逼近定理。

首先是正向传递的子过程，即数据根据迭代出来的权重值沿网络进行计算直到输出层。在本算法中，激活函数为：s = 1 / (1 + exp(-x))。

接着是反向传递子过程，即反复修正权值和阀值，使得误差函数值达到最小。Widrow-Hoff学习规则是通过沿着相对误差平方和的最速下降方向（即梯度方向），连续调整网络的权值和阀值，根据梯度下降法，权值矢量的修正正比于当前位置上E(w, b)的梯度

最后达到精度要求后中止训练。

1. **决策树算法**

首先初始化数据，这里数据在程序中手动设置，分为数据与标签两个部分。接着计算数据集的总信息熵。然后编写通过计算信息增益挑选最优特征的函数，方便建立决策树时调用。接着建立决策树。这里为了方便通过字典类型的嵌套来完成树形结构的实现。每个节点都对剩余属性值进行判定，以此训练，形成决策树。在形成决策树后，只需要将后续测试值输入即可进行判定。

1. **实验数据及结果分析：**

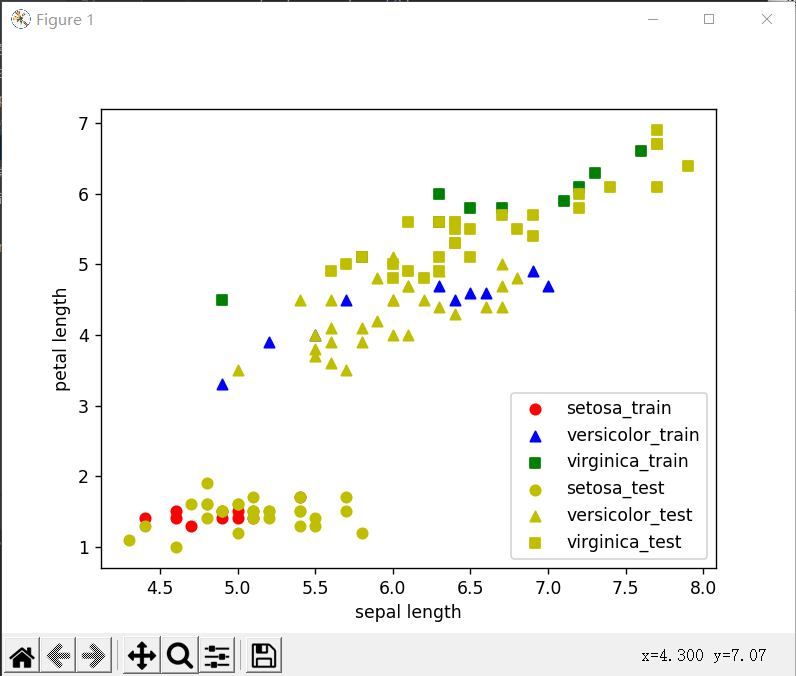
**（一） KNN算法**

1、实验数据

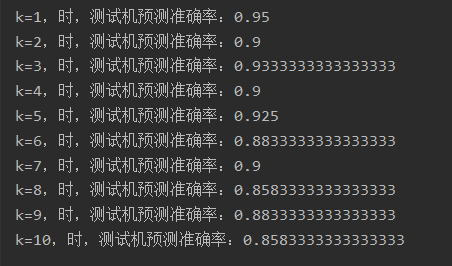
我们分别将训练集与测试集定为（10，40）、（20，30）、（30，20）、（40，10）来观察准确值、绘图以及距离矩阵，结果如下：

* + 1. 训练集10，测试集40

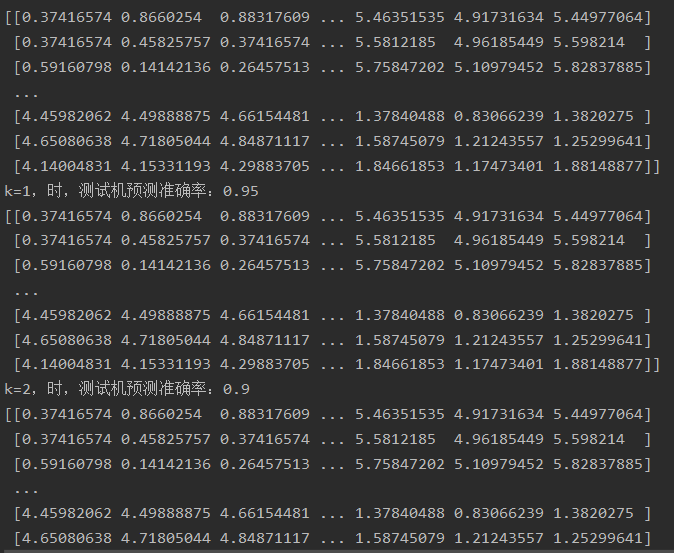
可视化图片：



准确率：

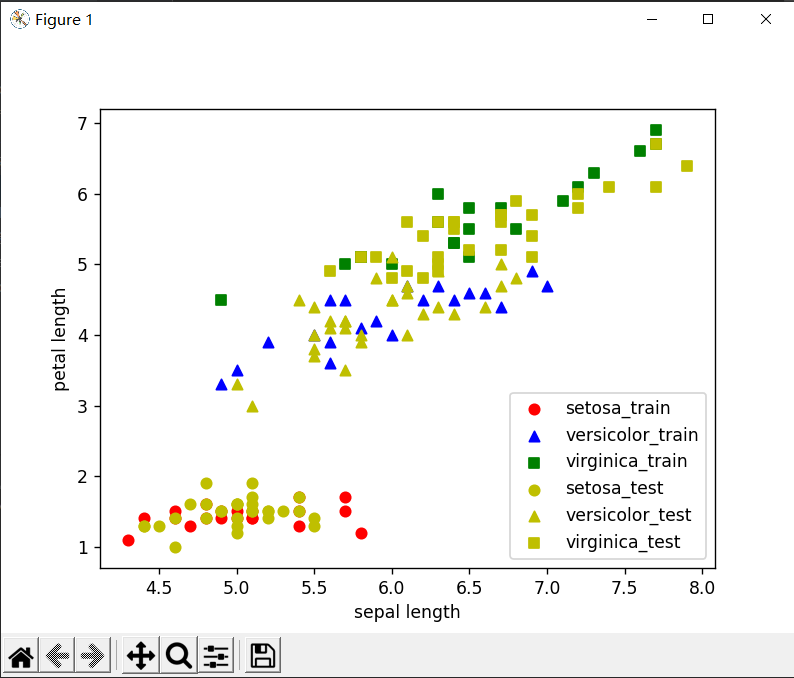


距离矩阵：

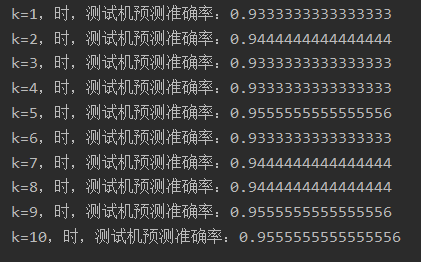


* + 1. 训练集20，测试集30

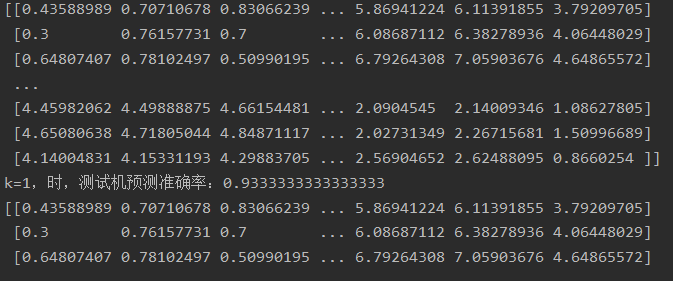
可视化图片：



准确率：

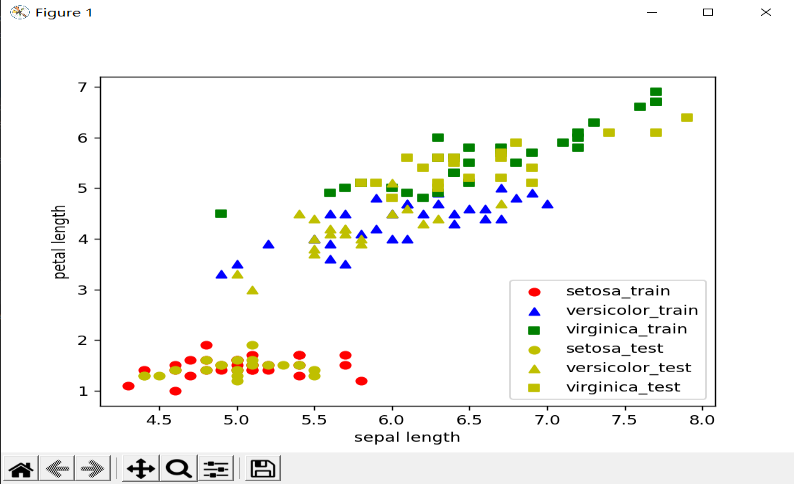


距离矩阵：

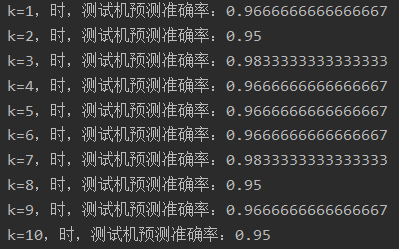


* + 1. 训练集30，测试集20

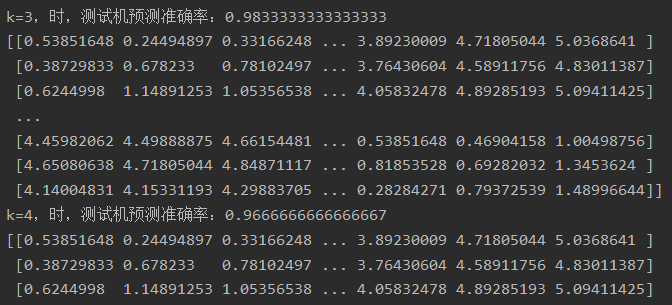
可视化图片：



准确率：

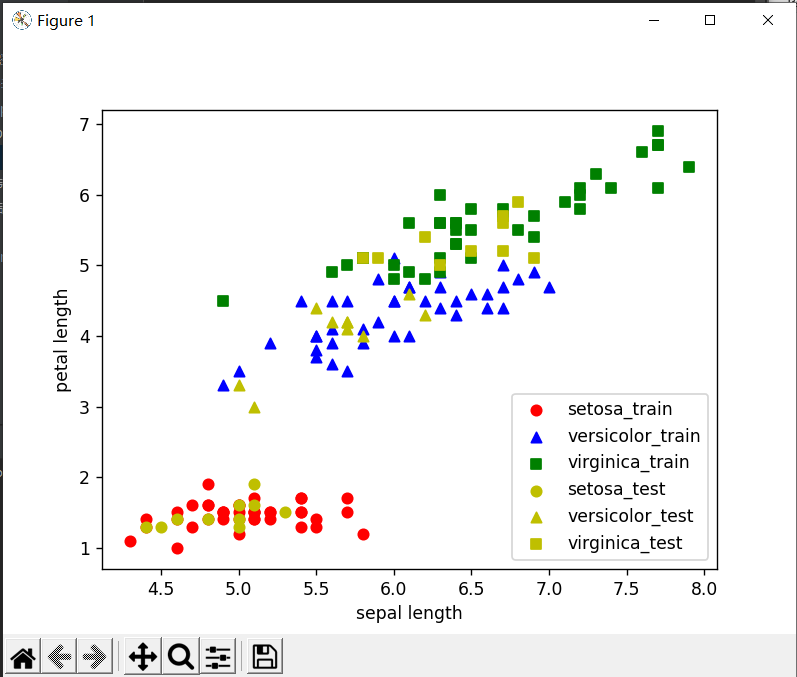


距离矩阵：

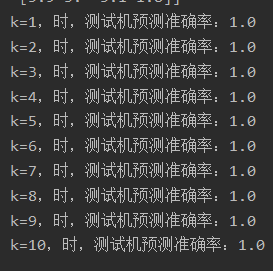


* + 1. 训练集40，测试集10

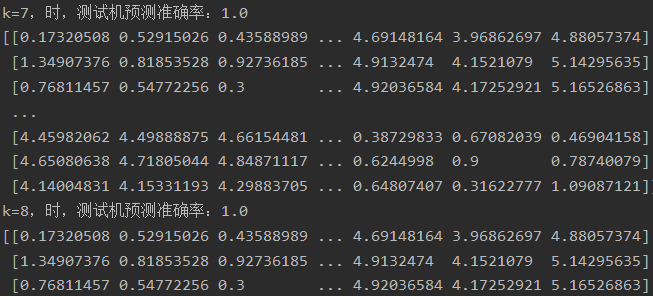
可视化图片：



准确率：



距离矩阵：



2、实验结果分析

首先我们可以明显发现，随着训练集数据的增多，预测准确率逐步提高。其次，由于数据集较为特殊，各类数据之间距离较远，所以即使训练集较少的情况下，测试结果表现仍然较好。而测试的结果随k值的变化成先上升后下降的趋势，这说明我们在运用KNN算法时需要设置合适的k值才能达到最好的效果

**（二） ANN算法**

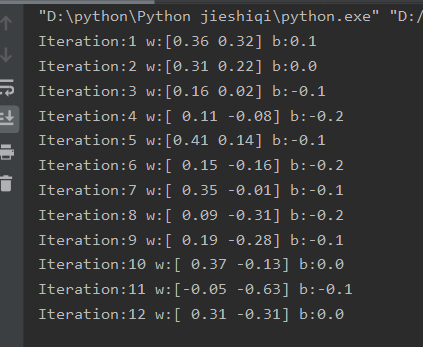
1、感知机

（１）实验数据

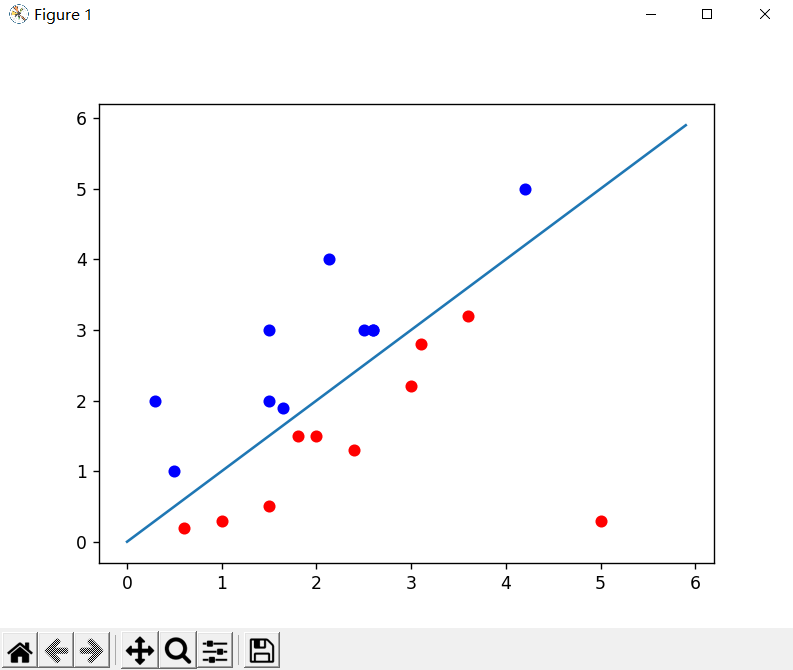
分别调整步长为0.1、0.5、1.0、2.0，结果如下：

步长为0.1时：

迭代数据：

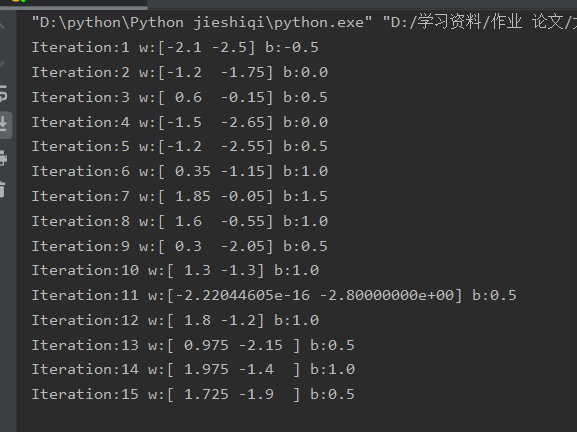


最终函数的可视化：

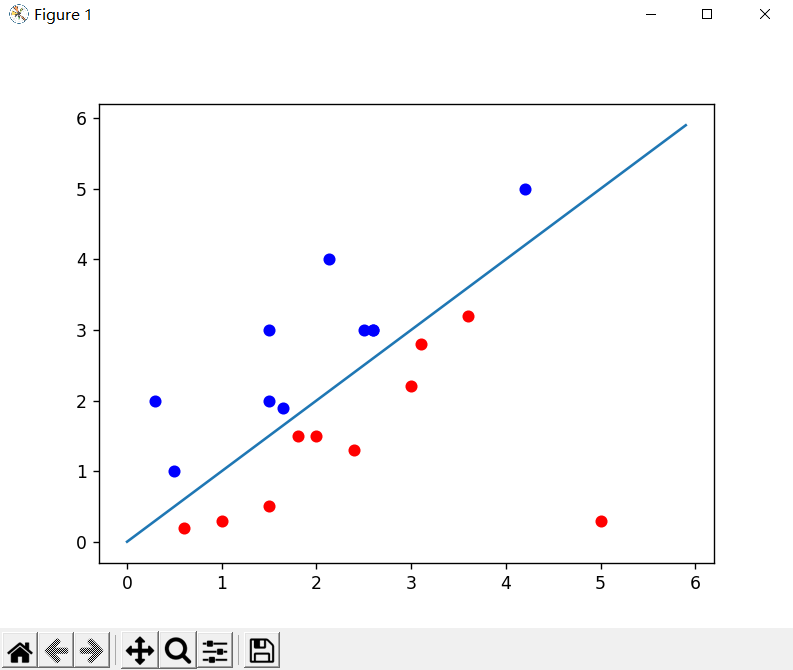


步长为0.5时：

迭代数据：

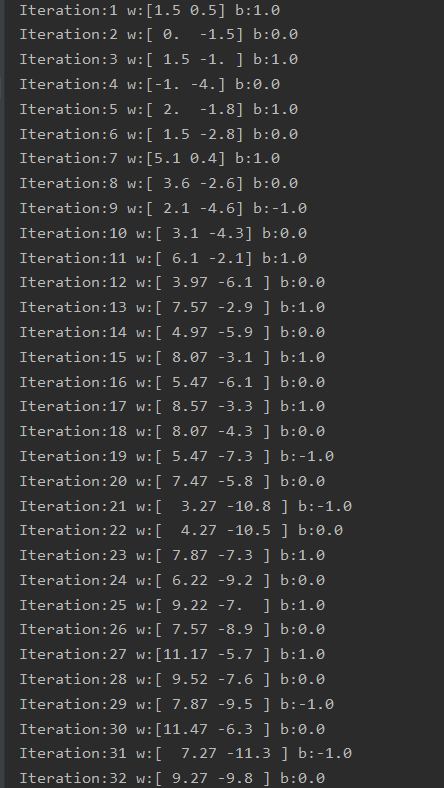


最终函数的可视化：

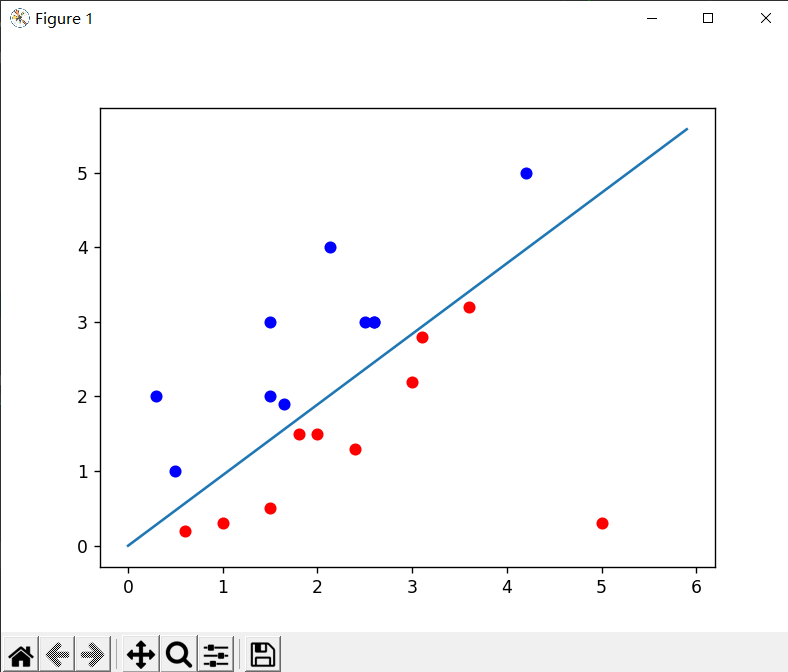


步长为1.0时：

迭代数据：

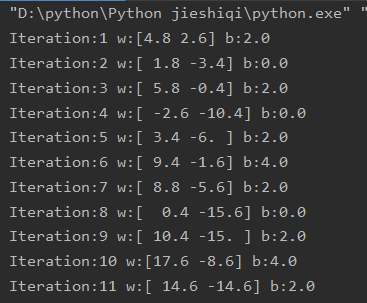


最终函数的可视化：

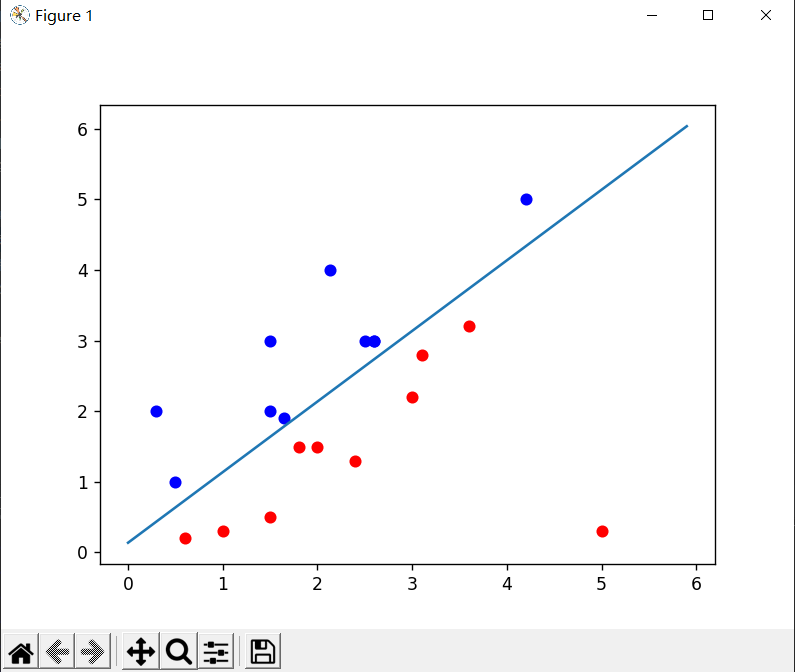


步长为2.0时：

迭代数据：



最终函数的可视化：



（２）实验结果分析

随着步长的增加，迭代次数先减少后增加，由此可见一个合适的步长可以大幅提升训练的效率与结果。从结果来看，最后两个系数之比都接近于-1，符合预期。而不论什么步长，训练结果都较为相似，也可以见的这个算法有一定的鲁棒性。

2、BP神经网络

（１）实验数据

这里输入均取（2，5）不变，输出均取（3，6，4）不变。将隐藏层节点数作为变量进行分析。根据经验公式，在此时隐藏层最佳节点数为3至 12。故分别取3、6、9、12进行观察。观察变量有：输入层权重矩阵V，输出层权重矩阵W，进行100次训练后得出网络的平均训练次数。

进行100次训练后得出网络的平均训练次数：

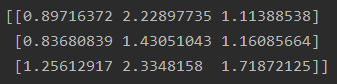


隐藏层节点数为3时：

输入层权重矩阵V：



输出层权重矩阵W：



输出为：

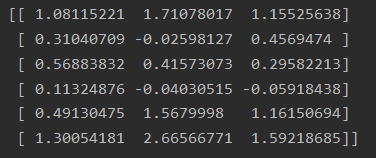


隐藏层节点数为6时：

输入层权重矩阵V：



输出层权重矩阵W：

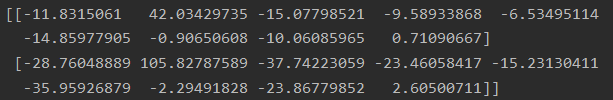


输出为：

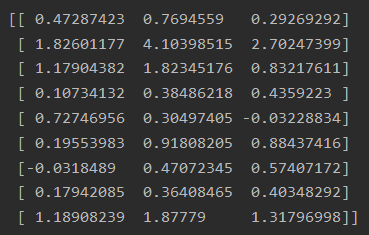


隐藏层节点数为9时：

输入层权重矩阵V：



输出层权重矩阵W：

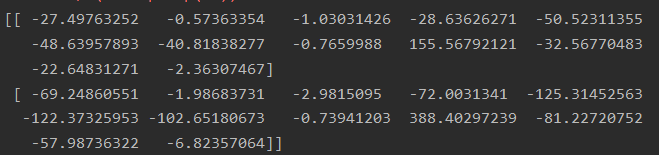


输出为：

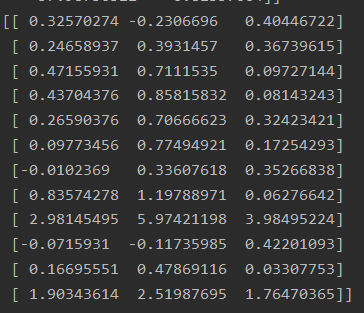


隐藏层节点数为12时：

输入层权重矩阵V：



输出层权重矩阵W：



输出为：



（２）实验结果分析

根据进行100次训练后得出网络的平均训练次数来看，随着隐藏层节点数的增加，训练次数先增加后减少，最后稳定于2000次左右。可见经验公式给出的值较为合理。并且不论采取多少节点数，最后得出的值都极为逼近真实值，这也从侧面论证了万能逼近定理的正确性，以及BP神经网络的性能。但是光看权重矩阵并不能获知太多信息，这也可以看出神经网络这个方法缺少可解释性。

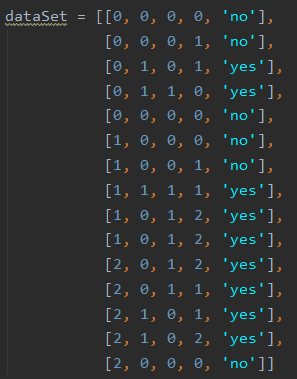
**（三） 决策树算法**

1、实验数据

调整训练数据两次后用同一组数据[0, 1, 0, 1]进去测试的结果如下：

第一次：

训练集：



决策树：

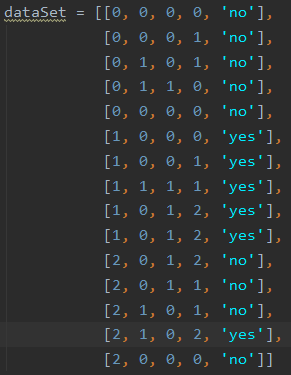


结果：



第二次：

训练集：



决策树：



结果：



2、实验结果分析

根据数据集的变化，生成的决策树发生了相应变化，最终决策也随之变化。我们可以看见，这些变化于数据集所导向的趋势一致，由此可以见得决策树算法的准确性和其对于训练数据的敏感性。

1. **实验结论、心得体会和改进建议：**

**（一）实验结论**

根据实验数据于分析，总结出如下实验结论：首先，本次实验用Python完整实现了要求的算法；并且通过查阅资料等形式进一步加深了对算法的理解；在老师给出的指导的基础上，创新性地调整了各个算法的关键参数进行观察，进一步理解了参数的意义并拓宽了视野。

**（二）心得与建议**

本次实验体验很好，教程详实，老师教学认真仔细，助教讲解耐心。保持现状即可，没有需要进一步调整的东西。

1. **代码展示：**
2. **KNN算法**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

class KNearNeighbor(object):

def \_\_init\_\_(self):

pass

# 定义一个内部类loadData，用于获取并初始化数据

def loadData(self, path):

data = pd.read\_csv(path, header=None)

# 特征及类别名称

data.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'species']

x = data.iloc[0:150, 0:4].values

y = data.iloc[0:150, 4].values

num\_of\_train = 40

# Iris-setosa 输出label用0表示

y[y == 'Iris-setosa'] = 0

# Iris-versicolor 输出label用1来表示

y[y == 'Iris-versicolor'] = 1

# Iris-virginica 输出label用1来表示

y[y == 'Iris-virginica'] = 2

# Iris-setosa 4个特征

self.x\_setosa, self.y\_setosa = x[0:50], y[0:50]

# Iris-versicolor 4个特征

self.x\_versicolor, self.y\_versicolor = x[50:100], y[50:100]

# Iris-virginica 4个特征

self.x\_virginica, self.y\_virginica = x[100:150], y[100:150]

# training set

self.x\_setosa\_train = self.x\_setosa[: num\_of\_train, :]

self.y\_setosa\_train = self.y\_setosa[: num\_of\_train]

self.x\_versicolor\_train = self.x\_versicolor[: num\_of\_train, :]

self.y\_versicolor\_train = self.y\_versicolor[: num\_of\_train]

self.x\_virginica\_train = self.x\_virginica[: num\_of\_train, :]

self.y\_virginica\_train = self.y\_virginica[: num\_of\_train]

self.x\_train = np.vstack([self.x\_setosa\_train, self.x\_versicolor\_train, self.x\_virginica\_train])

self.y\_train = np.hstack([self.y\_setosa\_train, self.y\_versicolor\_train, self.y\_virginica\_train])

# test set

self.x\_setosa\_test = self.x\_setosa[num\_of\_train: 50, :]

self.y\_setosa\_test = self.y\_setosa[num\_of\_train: 50]

self.x\_versicolor\_test = self.x\_versicolor[num\_of\_train: 50, :]

self.y\_versicolor\_test = self.y\_versicolor[num\_of\_train: 50]

self.x\_virginica\_test = self.x\_virginica[num\_of\_train: 50, :]

self.y\_virginica\_test = self.y\_virginica[num\_of\_train: 50]

self.x\_test =np.vstack([self.x\_setosa\_test, self.x\_versicolor\_test, self.x\_virginica\_test])

self.y\_test = np.hstack([self.y\_setosa\_test, self.y\_versicolor\_test, self.y\_virginica\_test])

# 利用matplotlib展示数据

def showData(self):

# 只选择sepal length和petal length两个特征，在二维平面上作图

# 训练集

plt.scatter(self.x\_setosa\_train[:, 0], self.x\_setosa\_train[:, 2], color='red', marker='o', label='setosa\_train')

plt.scatter(self.x\_versicolor\_train[:, 0], self.x\_versicolor\_train[:, 2], color='blue', marker='^', label='versicolor\_train')

plt.scatter(self.x\_virginica\_train[:, 0], self.x\_virginica\_train[:, 2], color='green', marker='s', label='virginica\_train')

# 测试集

plt.scatter(self.x\_setosa\_test[:, 0], self.x\_setosa\_test[:, 2], color='y', marker='o', label='setosa\_test')

plt.scatter(self.x\_versicolor\_test[:, 0], self.x\_versicolor\_test[:, 2], color='y', marker='^', label='versicolor\_test')

plt.scatter(self.x\_virginica\_test[:, 0], self.x\_virginica\_test[:, 2], color='y', marker='s', label='virginica\_test')

plt.xlabel('sepal length')

plt.ylabel('petal length')

plt.legend(loc = 4)

plt.show()

# 预测函数

def predict(self, x, k=1):

# 计算欧氏距离

num\_test = x.shape[0]

# 方差公式

d1 = -2 \* np.dot(x, self.x\_train.T)

# shape(num\_test, 1)

d2 = np.sum(np.square(x), axis=1, keepdims=True)

d3 = np.sum(np.square(self.x\_train), axis=1)

dist = np.sqrt(d1 + d2 + d3)

# print(dist)

# 根据K值，选择最可能属于的类别

y\_pred = np.zeros(num\_test)

for i in range(num\_test):

dist\_k\_min = np.argsort(dist[i])[:k] # 最近邻K个实例位置

y\_kclose = self.y\_train[dist\_k\_min] # 最近邻K个实例对应标签

# 找出K个标签中从属类别最多的作为预测类别

y\_pred[i] = np.argmax(np.bincount(y\_kclose.tolist()))

return y\_pred

if \_\_name\_\_ == "\_\_main\_\_":

path = "./iris.data"

knn = KNearNeighbor()

knn.loadData(path)

knn.showData()

print('训练集：')

print(knn.x\_train)

print('测试集：')

print(knn.x\_test)

for k in range(1, 11):

y\_pred = knn.predict(x=knn.x\_test, k=k)

accuracy = np.mean(y\_pred == knn.y\_test)

print(f'k={k}，时，测试机预测准确率：{accuracy}')

1. **ANN算法**
2. **感知机**

import numpy as np

import matplotlib.pyplot as plt

# 感知机

# load data from txt

data\_set = []

data\_label = []

file = open('ann\_Perceptron.txt')

for line in file:

line = line.split(' ')

for i in range(len(line)):

line[i] = float(line[i])

data\_set.append(line[0:2])

data\_label.append(int(line[-1]))

file.close()

data = np.array(data\_set)

label = np.array(data\_label)

# 初始化w, b, alpha

w = np.array([0, 0])

b = 0

alpha = 2.0

# 计算 y\*(w\*x+b)

f = (np.dot(data, w.T) + b) \* label

idx = np.where(f <= 0)

# 使用随机梯度下降算法求解w, b

iteration = 1

while f[idx].size != 0:

point = np.random.randint((f[idx].shape[0]))

x = data[idx[0][point], :]

y = label[idx[0][point]]

w = w + alpha \* y \* x

b = b + alpha \* y

print('Iteration:%d w:%s b:%s' % (iteration, w, b))

f = (np.dot(data, w.T) + b) \* label

idx = np.where(f <= 0)

iteration = iteration + 1

# 绘图演示

x1 = np.arange(0, 6, 0.1)

x2 = (w[0] \* x1 + b) / (-w[1])

idx\_p = np.where(label == 1)

idx\_n = np.where(label != 1)

data\_p = data[idx\_p]

data\_n = data[idx\_n]

plt.scatter(data\_p[:, 0], data\_p[:, 1], color='red')

plt.scatter(data\_n[:, 0], data\_n[:, 1], color='blue')

plt.plot(x1, x2)

plt.show()

print('\nPerception leanring algorithm is over')

1. **BP神经网络**

import numpy as np

import matplotlib.pyplot as plt

# BP网络

def BP(x, y, num):

epochs = 20000

learning\_rate = 0.3

times = 0

I\_num = x.shape[0]

H\_num = num

O\_num = y.shape[0]

V = np.random.rand(I\_num, H\_num) - 0.5

dV = np.random.rand(I\_num, H\_num)

W = np.random.rand(H\_num, O\_num) - 0.5

dW = np.random.rand(H\_num, O\_num)

mse\_record = np.ones((1, epochs))

for step in range(epochs):

# Forward

H\_in = np.dot(x, V) # 1 x H\_num

H\_out = sigmoid(H\_in) # 1 x H\_num

O\_in = np.dot(H\_out, W) # 1 x O\_num

O\_out = O\_in # 1 x O\_num

error = (O\_out - y) # 1 x O\_num

mse = np.average(np.square(error))

mse\_record[0, step] = mse

if mse < 4e-4:

break

# Backward

# 更新W

for h in range(H\_num):

for j in range(O\_num):

# W 偏导

dW[h, j] = (y[j]- O\_out[j]) \* H\_out[h]

# 更新V

for i in range(I\_num):

for h in range(H\_num):

sum = 0

for j in range(O\_num):

sum = sum + (y[j] - O\_out[j]) \* O\_out[j] \* (H\_out[h]) \* W[h, j]

dV[i, h] = sum \* H\_out[h] \* x[i]

W = W + learning\_rate \* dW

V = V + learning\_rate \* dV

times = step

print(V)

print(W)

print(30\*'-')

print(step, O\_out)

return times

def sigmoid(x):

s = 1 / (1 + np.exp(-x))

return s

def sigmoid\_dervative(x):

s = 1 / (1 + np.exp(-x))

ds = s \* (1 - s)

return ds

if \_\_name\_\_ == "\_\_main\_\_":

x = np.array([2, 5], dtype=float)

y\_true = np.array([3, 6, 4], dtype=float)

BP(x, y\_true, 12)

# traintime = {}

# for j in range(3, 15, 3):

# totaltimes = []

# for i in range(100):

# totaltimes.append(BP(x, y\_true, j))

# average = np.mean(totaltimes)

# traintime[j] = average

# print(traintime)

1. **决策树**

from math import log

import operator

# 初始化数据

def createDataSet():

'''

年龄：0 青年；1 中年；2 老年

有无工作：0 否；1 是

有无自己的房子：0 否；1 是

信贷情况： 0 一般；1 好；2 非常好

类别（是否给贷款）：no 否；yes 是

'''

# 数据集

dataSet = [[0, 0, 0, 0, 'no'],

[0, 0, 0, 1, 'no'],

[0, 1, 0, 1, 'yes'],

[0, 1, 1, 0, 'yes'],

[0, 0, 0, 0, 'no'],

[1, 0, 0, 0, 'no'],

[1, 0, 0, 1, 'no'],

[1, 1, 1, 1, 'yes'],

[1, 0, 1, 2, 'yes'],

[1, 0, 1, 2, 'yes'],

[2, 0, 1, 2, 'yes'],

[2, 0, 1, 1, 'yes'],

[2, 1, 0, 1, 'yes'],

[2, 1, 0, 2, 'yes'],

[2, 0, 0, 0, 'no']]

labels = ['年龄', '有工作', '有自己的房子', '信贷情况'] # 特征标签

return dataSet, labels

# 信息熵计算

def calcShannonEnt(dataSet):

numEntires = len(dataSet)

labelCounts = {}

for featVec in dataSet:

currentLabel = featVec[-1]

if currentLabel not in labelCounts.keys():

labelCounts[currentLabel] = 0

labelCounts[currentLabel] += 1

shannonEnt = 0.0

for key in labelCounts:

prob = float(labelCounts[key]) / numEntires

shannonEnt -= prob \* log(prob, 2)

return shannonEnt

def chooseBestFeatureToSplit(dataSet):

numFeatures = len(dataSet[0]) - 1

baseEntropy = calcShannonEnt(dataSet)

bestInfoGain = 0.0

bestFeature = -1

for i in range(numFeatures):

featList = [example[i] for example in dataSet]

uniqueVals = set(featList)

newEntropy = 0.0

for value in uniqueVals:

subDataSet = splitDataSet(dataSet, i, value)

prob = len(subDataSet) / float(len(dataSet))

newEntropy += prob \* calcShannonEnt(subDataSet)

infoGain = baseEntropy - newEntropy

if (infoGain > bestInfoGain):

bestInfoGain = infoGain

bestFeature = i

return bestFeature

# 代码功能：划分数据集

def splitDataSet(dataSet, axis, value): # 传入三个参数第一个参数是我们的数据集，是一个链表形式的数据集；第二个参数是我们的要依据某个特征来划分数据集

retDataSet = [] # 由于参数的链表dataSet我们拿到的是它的地址，也就是引用，直接在链表上操作会改变它的数值，所以我们新建一格链表来做操作

for featVec in dataSet:

if featVec[axis] == value: # 如果某个特征和我们指定的特征值相等

# 除去这个特征然后创建一个子特征

reduceFeatVec = featVec[:axis]

reduceFeatVec.extend(featVec[axis + 1:])

# 将满足条件的样本并且经过切割后的样本都加入到我们新建立的样本中

retDataSet.append(reduceFeatVec)

return retDataSet

# 统计在classList中每个元素出现的次数，并返回出现最多的元素

def majoritCnt(classList):

classCount = {}

for vote in classList:

if vote not in classCount.keys() : classCount[vote] = 0

classCount[vote] += 1

sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)

return sortedClassCount[0][0]

# 创建决策树

def createTree(dataSet, labels, featLabels):

classList = [example[-1] for example in dataSet]

if classList.count(classList[0]) == len(classList):

return classList[0]

if len(dataSet[0]) == 1:

return majoritCnt(classList)

bestFeat = chooseBestFeatureToSplit(dataSet)

bestFeatLabel = labels[bestFeat]

featLabels.append(bestFeatLabel)

myTree = {bestFeatLabel:{}}

del(labels[bestFeat])

featValues = [example[bestFeat] for example in dataSet]

uniqueVals = set(featValues)

for value in uniqueVals:

myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), labels, featLabels)

print(myTree)

return myTree

def classify(inputTree, featLabels, testVec):

firstStr = next(iter(inputTree))

secondDict = inputTree[firstStr]

featIndex = featLabels.index(firstStr)

for key in secondDict.keys():

if testVec[featIndex] == key:

if type(secondDict[key]).\_\_name\_\_ == 'dict':

classLabel = classify(secondDict[key], featLabels, testVec)

else:classLabel = secondDict[key]

return classLabel

if \_\_name\_\_ == '\_\_main\_\_':

dataSet, labels = createDataSet()

featLabels = []

myTree = createTree(dataSet, labels, featLabels)

testVec = [0, 1, 0, 1]

result = classify(myTree, featLabels, testVec)

if result == 'yes':

print('放贷')

if result == 'no':

print('不放贷')