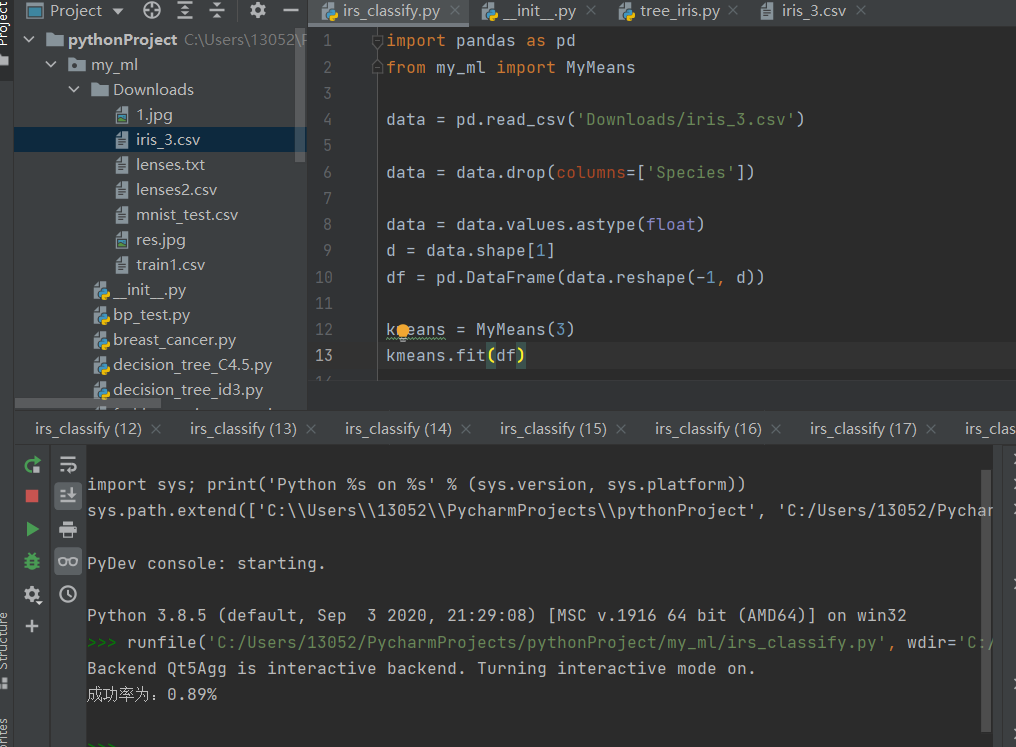
1.K-means——鸢尾花分类

## 1.1iris.csv数据集



import pandas as pd

from my\_ml import MyMeans

data = pd.read\_csv('Downloads/iris\_3.csv')

data = data.drop(columns=['Species'])

data = data.values.astype(float)

d = data.shape[1]

df = pd.DataFrame(data.reshape(-1, d))

kmeans = MyMeans(3)

kmeans.fit(df)

test\_center, labels = kmeans.centers, kmeans.labels

x = []

data\_test1 = pd.read\_csv('Downloads/iris\_3.csv')

data\_test1.loc[data\_test1.Species == 'setosa', 'Species'] = 0

data\_test1.loc[data\_test1.Species == 'versicolor', 'Species'] = 1

data\_test1.loc[data\_test1.Species == 'virginica', 'Species'] = 2

data\_test1 = data\_test1.values.astype(float)

x1, y = 0, 150

for i in range(150):

if (labels[i] == data\_test1[i, -1]):

x1 = x1+1

x.append(x1)

data\_test2 = pd.read\_csv('Downloads/iris\_3.csv')

data\_test2.loc[data\_test2.Species == 'setosa', 'Species'] = 0

data\_test2.loc[data\_test2.Species == 'versicolor', 'Species'] = 2

data\_test2.loc[data\_test2.Species == 'virginica', 'Species'] = 1

data\_test2 = data\_test2.values.astype(float)

x2 = 0

for i in range(150):

if (labels[i] == data\_test2[i, -1]):

x2 = x2+1

x.append(x2)

data\_test3 = pd.read\_csv('Downloads/iris\_3.csv')

data\_test3.loc[data\_test3.Species == 'setosa', 'Species'] = 1

data\_test3.loc[data\_test3.Species == 'versicolor', 'Species'] = 0

data\_test3.loc[data\_test3.Species == 'virginica', 'Species'] = 2

data\_test3 = data\_test3.values.astype(float)

x3 = 0

for i in range(150):

if (labels[i] == data\_test3[i, -1]):

x3 = x3+1

x.append(x3)

data\_test4 = pd.read\_csv('Downloads/iris\_3.csv')

data\_test4.loc[data\_test4.Species == 'setosa', 'Species'] = 1

data\_test4.loc[data\_test4.Species == 'versicolor', 'Species'] = 2

data\_test4.loc[data\_test4.Species == 'virginica', 'Species'] = 0

data\_test4 = data\_test4.values.astype(float)

x4 = 0

for i in range(150):

if (labels[i] == data\_test4[i, -1]):

x4 = x4+1

x.append(x4)

data\_test5 = pd.read\_csv('Downloads/iris\_3.csv')

data\_test5.loc[data\_test5.Species == 'setosa', 'Species'] = 2

data\_test5.loc[data\_test5.Species == 'versicolor', 'Species'] = 0

data\_test5.loc[data\_test5.Species == 'virginica', 'Species'] = 1

data\_test5 = data\_test5.values.astype(float)

x5 = 0

for i in range(150):

if (labels[i] == data\_test5[i, -1]):

x5 = x5+1

x.append(x5)

data\_test6 = pd.read\_csv('Downloads/iris\_3.csv')

data\_test6.loc[data\_test6.Species == 'setosa', 'Species'] = 2

data\_test6.loc[data\_test6.Species == 'versicolor', 'Species'] = 1

data\_test6.loc[data\_test6.Species == 'virginica', 'Species'] = 0

data\_test6 = data\_test6.values.astype(float)

x6 = 0

for i in range(150):

if (labels[i] == data\_test6[i, -1]):

x6 = x6+1

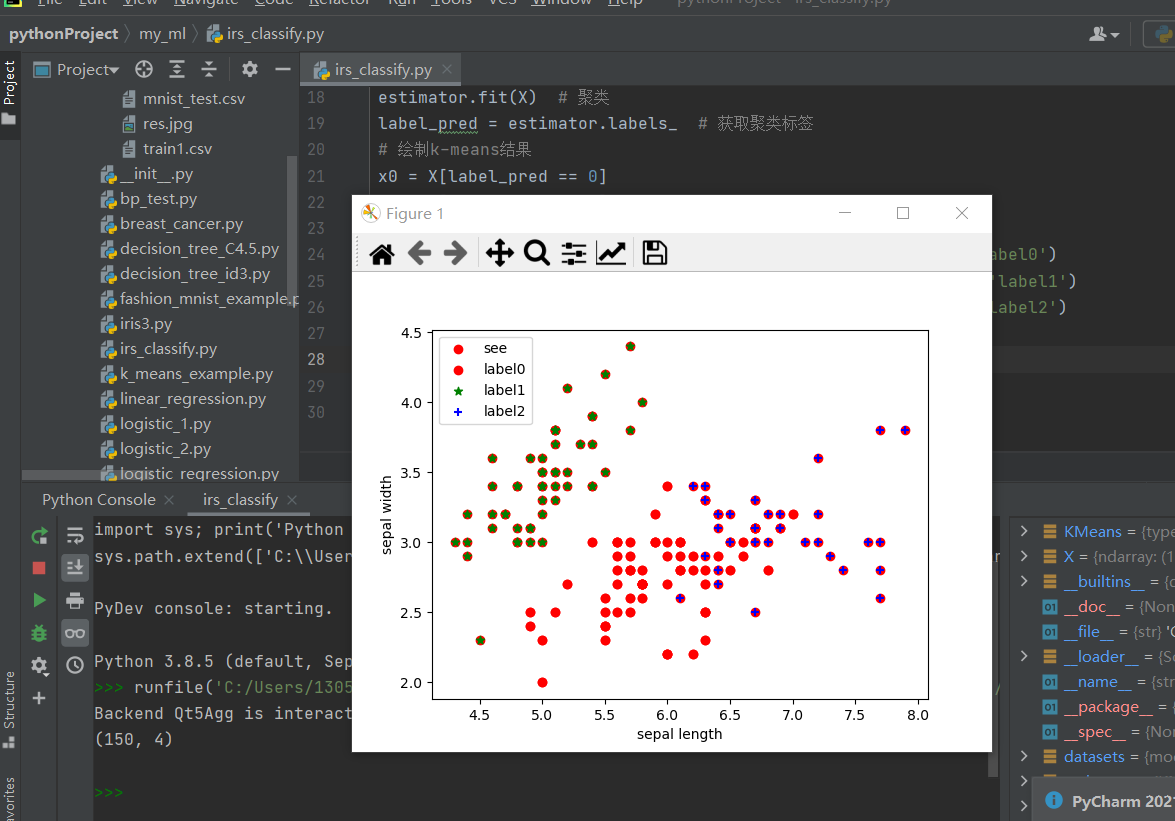
x.append(x6)

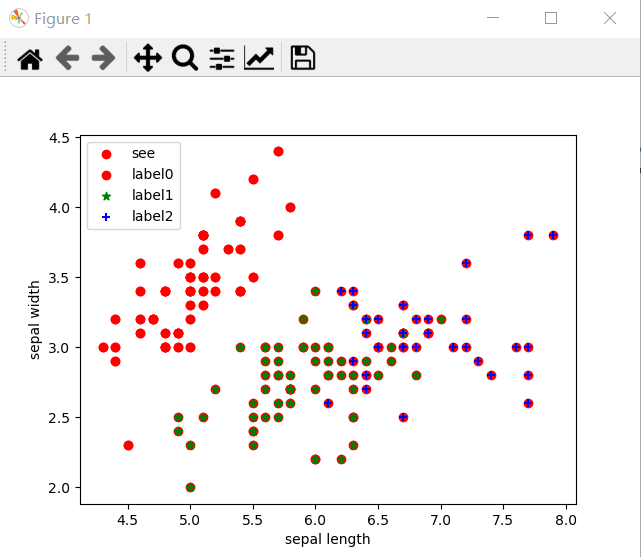
x\_max = max(x)

print("成功率为：{:.2f}%".format(x\_max / y))

#print(data)

## 1.2 kmeans 库数据集

1. 



import numpy as np

import pandas as pd

class MyMeans:

#随机初始化质心

def \_\_init\_\_(self,k,iters=None):

self.k = k

self.iters = iters

def random\_center(self,data, k):

data\_min = data.iloc[:, :].min()

data\_max = data.iloc[:, :].max()

center = np.random.uniform(data\_min, data\_max, (k, data.shape[1]))

return center

def cal\_distance(self,x, center):

distance = np.sum(np.power(x.values - center, 2), axis=1) #平方去掉负数,axis是在1轴上进行求和

distance\_min = distance.min()

distance\_min\_index = distance.argmin()

return distance\_min, distance\_min\_index

def fit(self,data):

m, n = data.shape #n表示样本的维度数

centers = self.random\_center(data, self.k) #生成质心

add\_3 = np.zeros((m, 3)) #添加三列数据,第一列为最小距离与当前距离的差值,第二列为当前(之前)分类，第三列为计算后新的分类

add\_3[:, 0] = np.inf

add\_3[:, 1:] = -1

data\_ext = pd.concat([data, pd.DataFrame(add\_3)], axis=1, ignore\_index=True)

isChange = True

while isChange:

if self.iters != None:

if self.iters <= 0:

break

else:

self.iters -= 1

for i in range(m):

dist\_min, dist\_min\_index = self.cal\_distance(data\_ext.iloc[i, :n], centers)

data\_ext.iloc[i, n] = dist\_min

data\_ext.iloc[i, n+1] = dist\_min\_index

isChange = (data\_ext.iloc[:, -1] != data\_ext.iloc[:, -2]).any() #判断类别算法发生改变

#yield (centers,data\_ext)#等待

if isChange:

cent\_df = data\_ext.groupby(n+1).mean() #groupby.mean()将每个类别分别求均值

centers = cent\_df.iloc[:, :n].values #将原本添加进来的三列去除

data\_ext.iloc[:, -1] = data\_ext.iloc[:, -2]

self.centers=centers

self.labels = data\_ext.iloc[:,-1].values

import matplotlib.pyplot as plt

import numpy as np

from sklearn.cluster import KMeans

from sklearn import datasets

iris = datasets.load\_iris()

X = iris.data[:, :4] # #表示我们取特征空间中的4个维度

print(X.shape)

# 绘制数据分布图

plt.scatter(X[:, 0], X[:, 1], c="red", marker='o', label='see')

plt.xlabel('sepal length')

plt.ylabel('sepal width')

plt.legend(loc=2)

plt.show()

estimator = KMeans(n\_clusters=3) # 构造聚类器

estimator.fit(X) # 聚类

label\_pred = estimator.labels\_ # 获取聚类标签

# 绘制k-means结果

x0 = X[label\_pred == 0]

x1 = X[label\_pred == 1]

x2 = X[label\_pred == 2]

plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o', label='label0')

plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='\*', label='label1')

plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker='+', label='label2')

plt.xlabel('sepal length')

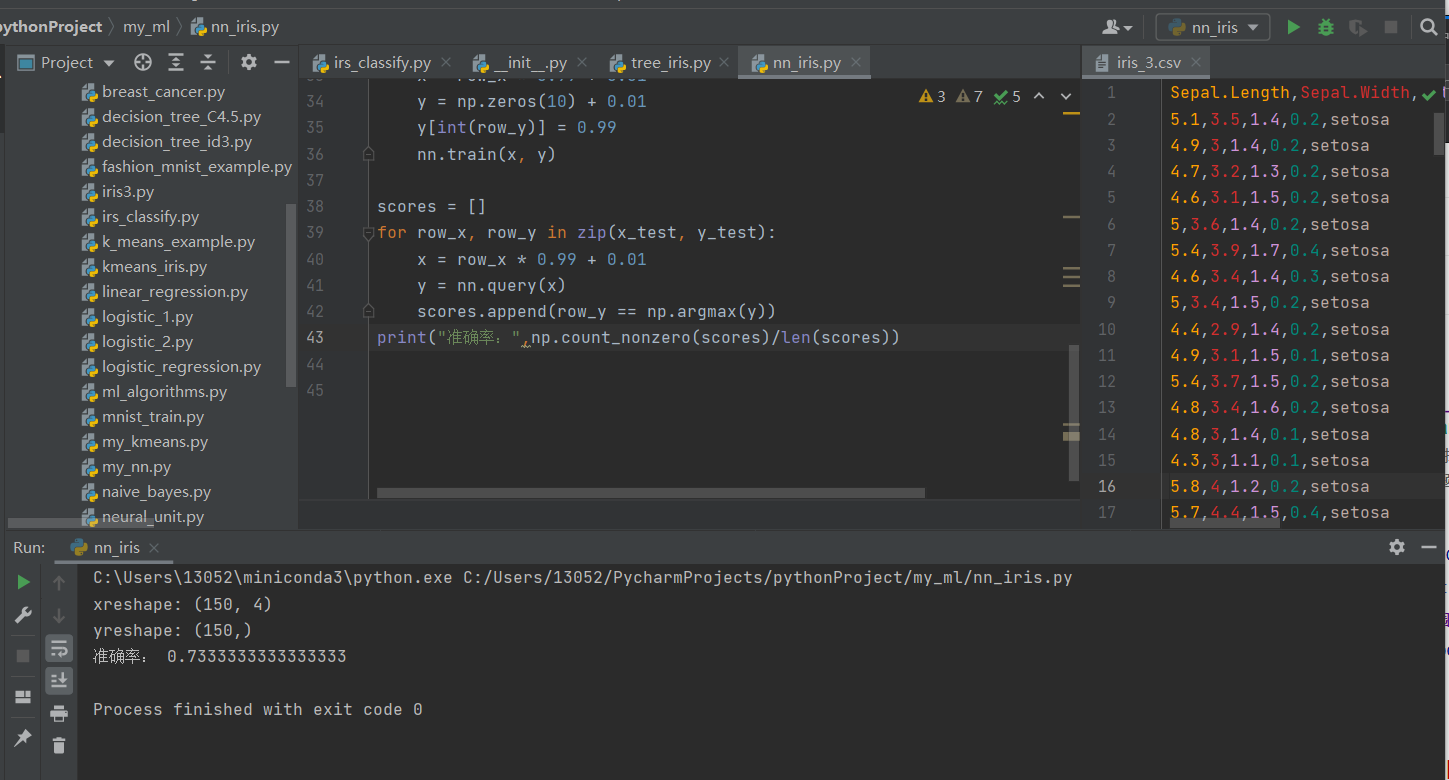
plt.ylabel('sepal width')

plt.legend(loc=2)

plt.show()

2.BP神经网络分类

## iris数据集



import numpy as np

from my\_ml import \*

import pandas as pd

from my\_nn import MyNeuralNetword

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('Downloads/iris\_3.csv')

data.loc[data.Species == 'setosa', 'Species'] = 1

data.loc[data.Species == 'versicolor', 'Species'] = 2

data.loc[data.Species == 'virginica', 'Species'] = 3

data = data.values.astype(float)

x\_data = data[:, :-1]

y\_data = data[:, -1]

x\_data = x\_data.reshape(x\_data.shape[0], -1)

#y\_data = y\_data.reshape(y\_data.shape[0], -1)

print("xreshape:", x\_data.shape)

print("yreshape:", y\_data.shape)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size=0.3)

#定义训练的神经网络模型

OUT\_COUNTS =10 #输出数据个数（输出层的结点个数）

HIDDEN\_COUNTS = [100] #隐层的层数

INPUT\_COUNTS = x\_train.shape[1]

nn = MyNeuralNetword(INPUT\_COUNTS, OUT\_COUNTS, HIDDEN\_COUNTS, lr=0.1)

epoch = 10

for row\_x, row\_y in zip(x\_train, y\_train):

x = row\_x \* 0.99 + 0.01

y = np.zeros(10) + 0.01

y[int(row\_y)] = 0.99

nn.train(x, y)

scores = []

for row\_x, row\_y in zip(x\_test, y\_test):

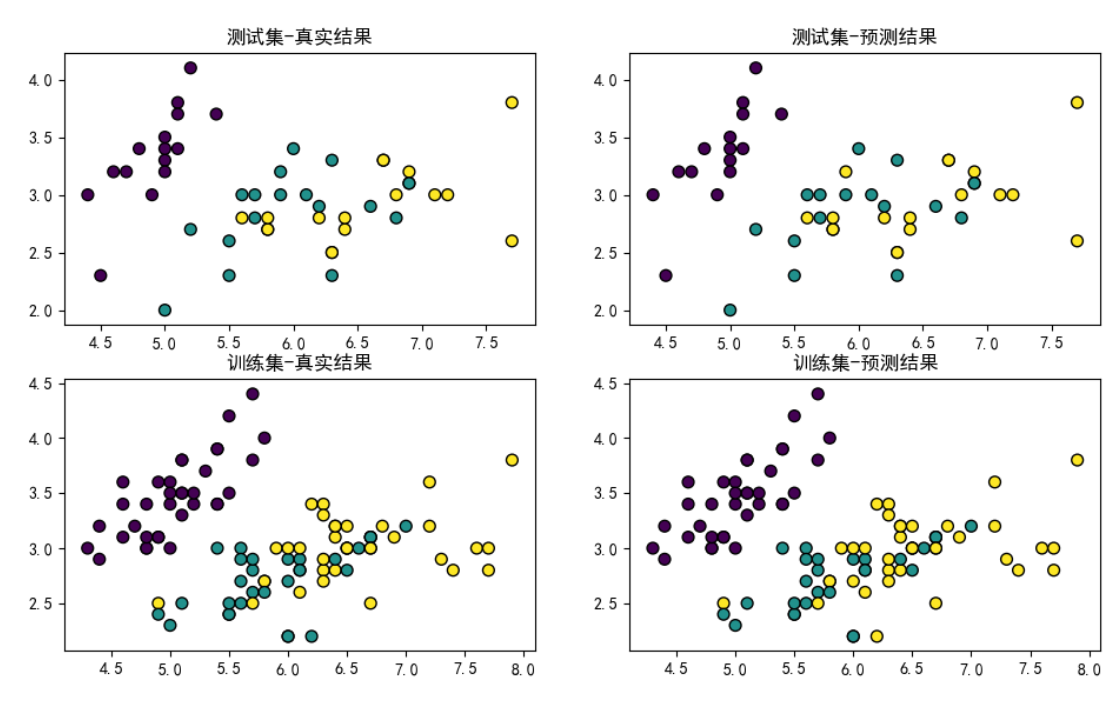
x = row\_x \* 0.99 + 0.01

y = nn.query(x)

scores.append(row\_y == np.argmax(y))

print("准确率：",np.count\_nonzero(scores)/len(scores))

## 2.2 BP库数据集



from my\_ml import \*

import numpy as np

import matplotlib.pyplot as plt

#代价->均方差

def cost(target,pred):

return np.sum(1/2.0\*np.power(target - pred,2))

x = np.array([1.0,0.05,0.10])

w = np.array([0.35,0.15,0.20])

y = 0.1

iters = 1000

alpha = 0.1

costs = []

for i in range(iters):

h = logistic\_hypothesis(w,x)

g = (h-y) \* (1-h) \* h \* x

w -= alpha \* g \* x

costs.append(cost(y,h))

#print(logistic\_hypothesis(w,x))

#plt.plot(range(len(costs)),costs)

#plt.show()

#f(z) z->(x,y) F'(x)y'+F'(y) y'=-F'(y)/F'(x)

def sigmoid(z):

return 1/(1+np.exp(-z))

def log\_hypothesis(w,x):

return sigmoid(w@x)

from sklearn.model\_selection import train\_test\_split

from my\_ml import \*

iris = datasets.load\_iris()

iris\_feature = iris['data']

iris\_target = iris['target']

iris\_target\_name = iris['target\_names']

class NeuralNetwork(object):

def \_\_init\_\_(self, input\_nodes, hidden\_nodes, output\_nodes, learning\_rate):

self.input\_nodes = input\_nodes

self.hidden\_nodes = hidden\_nodes

self.output\_nodes = output\_nodes

self.weights\_input\_to\_hidden = np.random.normal(0.0, self.hidden\_nodes \*\* -0.5,

(self.hidden\_nodes, self.input\_nodes))

self.weights\_hidden\_to\_output = np.random.normal(0.0, self.output\_nodes \*\* -0.5,

(self.output\_nodes, self.hidden\_nodes))

self.lr = learning\_rate # 学习率

self.activation\_function = self.sigmoid

def sigmoid(self, x):

return 1.0 / (1 + np.exp(-x))

# 定义BP神经网络

class Net(torch.nn.Module):

def \_\_init\_\_(self, n\_feature, n\_hidden, n\_output):

super(Net, self).\_\_init\_\_()

self.hidden = torch.nn.Linear(n\_feature, n\_hidden)

self.out = torch.nn.Linear(n\_hidden, n\_output)

def forward(self, x):

x = Fun.relu(self.hidden(x))

x = self.out(x)

return x

for i in range(1000):

out = net(input)

loss = loss\_func(out, label)

# 输出与label对比

optimizer.zero\_grad()

# 初始化

loss.backward()

optimizer.step()

out = net(input)

# out是一个计算矩阵

prediction = torch.max(out, 1)[1]

pred\_y = prediction.numpy()

# 预测y输出数列

target\_y = label.data.numpy()

# 实际y输出数据

def train(self, inputs\_list, targets\_list):

# 正向传播

inputs = np.array(inputs\_list, ndmin=2).T

targets = np.array(targets\_list, ndmin=2).T

hidden\_inputs = np.dot(self.weights\_input\_to\_hidden, inputs)

hidden\_outputs = self.activation\_function(hidden\_inputs)

final\_inputs = np.dot(self.weights\_hidden\_to\_output, hidden\_outputs)

final\_outputs = final\_inputs # 因为的取值为0、1、2，所以这里不再用激活函数了，否则结果会被限制在0到1

delta\_output\_out = final\_outputs - targets

delta\_output\_in = delta\_output\_out

delta\_weight\_ho\_out = np.dot(delta\_output\_in, hidden\_outputs.T)

self.weights\_hidden\_to\_output -= (self.lr \* delta\_weight\_ho\_out)

epochs = 1000 # 训练次数

learning\_rate = 0.001

hidden\_nodes = 10

output\_nodes = 1

batch\_size = 50

input\_nodes = train\_features.shape[1]

network = NeuralNetwork(input\_nodes, hidden\_nodes, output\_nodes, learning\_rate)

for e in range(epochs): # 进行epochs次训练

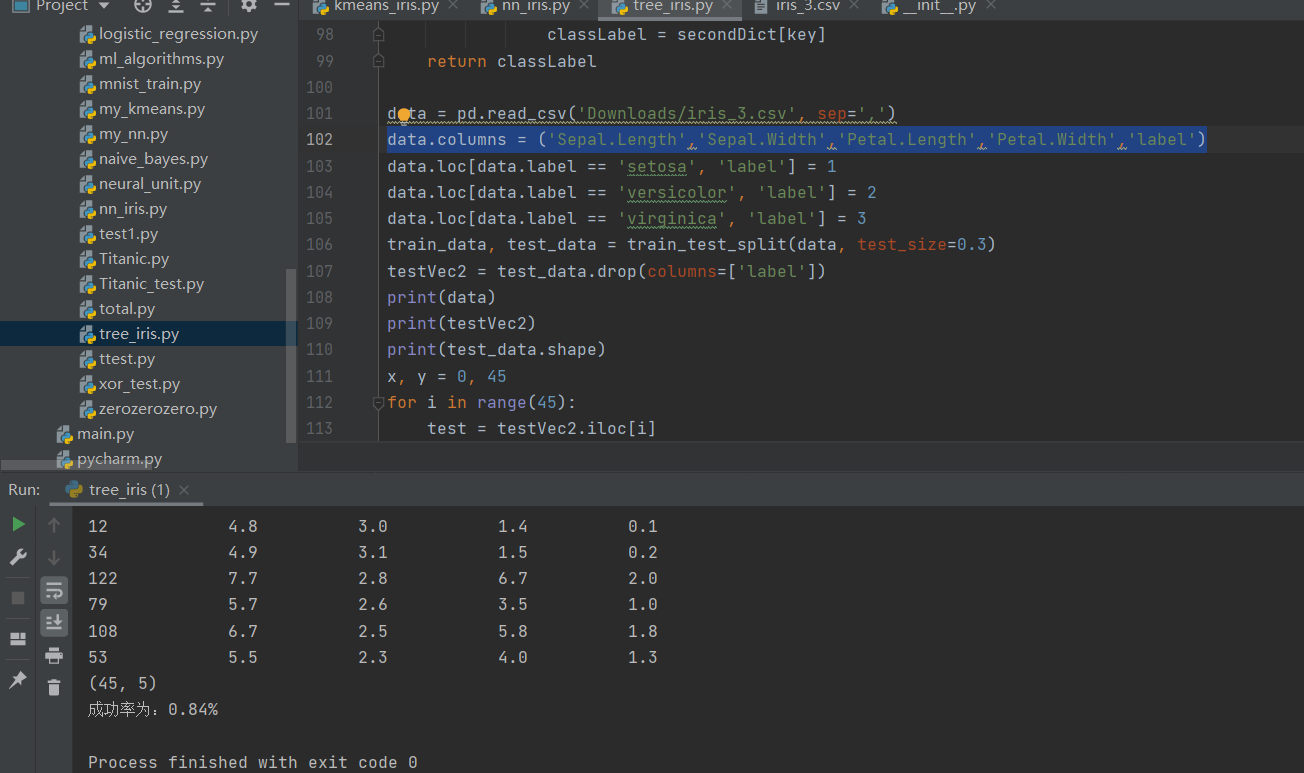
batch = np.random.choice(len(train\_features), size=batch\_size) # 从训练集中随机挑选50个样本进行训练

for record, target in zip(train\_features[batch],train\_targets[batch]):

network.train(record, target)

# 3.决策树模型——鸢尾花分类

## 3.1 决策树 iris.3数据集



from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

import pickle

# 保存和读取训练号的决策树模型

def storeTree(tree, filename):

f = open(filename, tree)

pickle.dump(tree, f)

f.close()

def loadTree(filename):

f = open(filename, 'rb')

t = pickle.load(f)

f.close()

return t

def Entropy(x): # 熵

return (-x) \* (np.log2(x)) - (1 - x) \* (np.log2(1 - x))

def calcEntropy(df, label='label'):

# data's sum counter

m = df.shape[0]

# for label counter

label\_counts = df[label].value\_counts()

prob = label\_counts \* 1.0 / m

return sum(-prob \* np.log2(prob))

def createTestData():

ds = pd.DataFrame([[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']],

columns=['no surfing', 'flippers', 'label'])

return ds

def splitDataFrame(df, feature, value):

columns = df.columns.to\_list()

columns.remove(feature)

return df.loc[df[feature] == value, columns]

def chooseBestFeature(df, label='label'):

features = df.columns.to\_list()

features.remove(label)

# 计算所有样本信息熵

totalEntropy = calcEntropy(df, label)

bestInfoGain = 0

bestFeature = ''

for f in features:

uniqueVals = df[f].unique()

newEntropy = 0

for value in uniqueVals:

subs = splitDataFrame(df, f, value)

prob = len(subs) / float(len(df)) # calculate 子集权重

newEntropy += prob \* calcEntropy(subs) # calculate 条件熵

infoGain = totalEntropy - newEntropy

if infoGain > bestInfoGain: # update 最大增益及最优特征

bestInfoGain = infoGain

bestFeature = f

return bestFeature

def majorityCnt(classList):

classCounts = classList.value\_counts().values

max\_i = classCounts.argmax() # find max index in classlist

return classList.iloc[max\_i]

def createTree(df, label='label'):

classList = df[label] # 统计data中 所有标签

# recursive停止的第一个条件->当前df中所有数据是纯的（一个特征）

if len(classList.unique()) == 1:

return classList.iloc[0]

# recursive停止的第二个条件->当所有标签都划分完了，都纯了，

# 且返回标签中 数量最多的一个

if df.shape[1] == 1:

return majorityCnt(classList)

bestFeature = chooseBestFeature(df, label)

myTree = {bestFeature: {}}

# statistical feature 的属性值

uniqueVals = df[bestFeature].unique()

# 对 每一个特征进行划分，递归建树 function

for value in uniqueVals:

myTree[bestFeature][value] = \

createTree(splitDataFrame(df, bestFeature, value))

return myTree

def classifier\_id3(inputTree, testVec):

# print("1\n")

firstStr = list(inputTree.keys())[0]

secondDict = inputTree[firstStr]

global classLabel

for key in secondDict.keys():

if (testVec[firstStr] == key):

if type(secondDict[key]) == dict:

classifier\_id3(secondDict[key], testVec)

classLabel = (classifier\_id3(secondDict[key], testVec))

else:

classLabel = secondDict[key]

return classLabel

data = pd.read\_csv('Downloads/iris\_3.csv', sep=',')

data.columns =('Sepal.Length','Sepal.Width','Petal.Length','Petal.Width','label')

data.loc[data.label == 'setosa', 'label'] = 1

data.loc[data.label == 'versicolor', 'label'] = 2

data.loc[data.label == 'virginica', 'label'] = 3

train\_data, test\_data = train\_test\_split(data, test\_size=0.3)

testVec2 = test\_data.drop(columns=['label'])

print(data)

print(testVec2)

print(test\_data.shape)

x, y = 0, 45

for i in range(45):

test = testVec2.iloc[i]

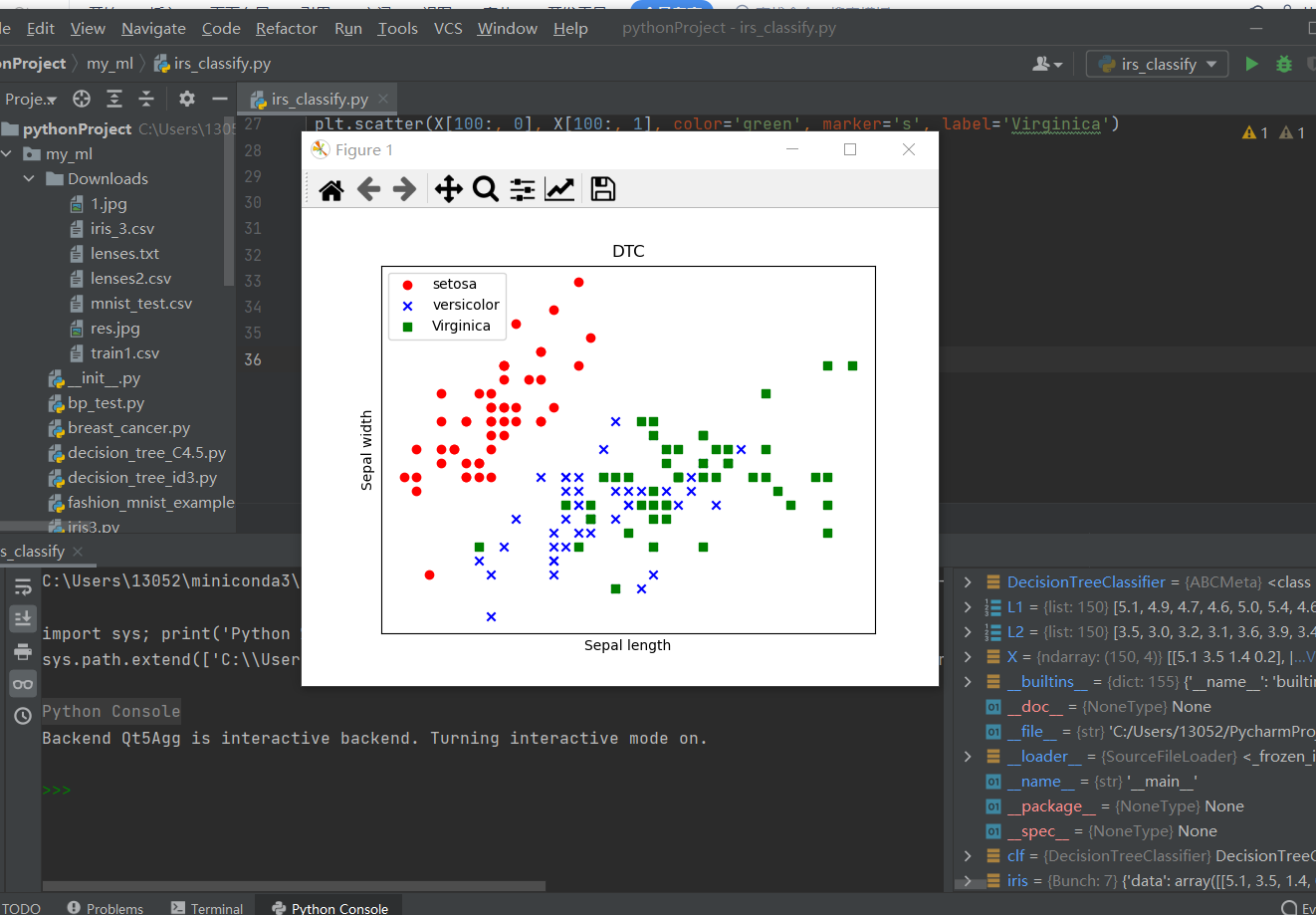
label = classifier\_id3(createTree(train\_data), testVec2.iloc[i])

if(label == test\_data.iloc[i].label):

x=x+1

print("成功率为：{:.2f}%".format(x /y))

## 3.2 决策树 库数据集



Algorithms = {

'LINEAR':[linear\_hypothesis, linear\_cost],

'LOGISTIC':[logistic\_hypothesis,logistic\_cost]}

import pickle

#保存和读取训练号的决策树模型

def storeTree(tree,filename):

f = open(filename,tree)

pickle.dump(tree,f)

f.close()

def loadTree(filename):

f = open(filename,'rb')

t = pickle.load(f)

f.close()

return t

def Entropy(x):#熵

return (-x)\*(np.log2(x))-(1-x)\*(np.log2(1-x))

def calcEntropy(df,label = 'label'):

# data's sum counter

m = df.shape[0]

#for label counter

label\_counts = df[label].value\_counts()

prob = label\_counts \* 1.0 / m

return sum( -prob\*np.log2(prob) )

def createTestData():

ds =pd.DataFrame( [[1,1,'yes'],[1,1,'yes'],[1,0,'no'],[0,1,'no'],[0,1,'no']],columns = ['no surfing','flippers','label'])

return ds

def splitDataFrame(df,feature,value):

columns = df.columns.to\_list()

columns.remove(feature)

return df.loc[ df[feature ] == value,columns]

def chooseBestFeature(df, label = 'label'):

features = df.columns.to\_list()

features.remove(label)

#计算所有样本集上的信息熵

totalEntropy = calcEntropy(df, label)

bestInfoGain = 0.0

bestFeature = ''

for f in features: #遍历每一个特征

uniqueVals = df[f].unique()

newEntropy = 0

splitinfo = 0

for value in uniqueVals:

subs = splitDataFrame(df, f, value)

weight = len(subs) / float(len(df)) #计算子集权重

newEntropy += weight \* calcEntropy(subs) #计算条件熵

splitinfo -= weight \* np.log2(weight) #计算增益率

#infoGain = totalEntropy - newEntropy #计算信息增益

infoGain = (totalEntropy - newEntropy) / splitinfo #计算信息增益率

if infoGain > bestInfoGain: #更新最大增益及最优特征

bestInfoGain = infoGain

bestFeature = f

return bestFeature

def majorityCnt(classList):

classCounts = classList.value\_counts().values

max\_i = classCounts.argmax() #find max index in classlist

return classList.iloc[max\_i]

def createTree(df,label = 'label'):

classList = df[label]#统计data中 所有标签

#recursive停止的第一个条件->当前df中所有数据是纯的（一个特征）

if len( classList.unique() ) == 1:

return classList.iloc[0]

#recursive停止的第二个条件->当所有标签都划分完了，都纯了，

#且返回标签中 数量最多的一个

if df.shape[1] == 1:

return majorityCnt(classList)

bestFeature = chooseBestFeature(df,label)

myTree = {bestFeature: {} }

#statistical feature 的属性值

uniqueVals = df[bestFeature].unique()

#对 每一个特征进行划分，递归建树 function

for value in uniqueVals:

myTree[bestFeature][value] = \

createTree( splitDataFrame(df,bestFeature,value) )

return myTree

def classifier(inputTree, testVec):

firstStr = list(inputTree.keys())[0]

secondDict = inputTree[firstStr]

for key in secondDict.keys():

if(testVec[firstStr] == key):

if type(secondDict[key]) == dict:

classLabel = classifier(secondDict[key], testVec)

else:

classLabel = secondDict[key]

return classLabel

def mean\_normalization(x):

return (x - x.mean(axis=0))/(x.max(axis=0) - x.min(axis=0))

def add\_bias(x):

m = x.shape[0]

ones = np.ones((m, 1))

return np.hstack((ones, x))

def gradient\_descend(iters, alpha, x, y, algo='LINEAR'):

hypothesis, cost = Algorithms[algo]

costs = []

T = np.zeros((x.shape[1]))

m = x.shape[0]

for i in range(iters):

costs.append(cost(T, x, y))

T = T - alpha\* np.sum((hypothesis(T, x) - y).reshape(-1,1)\*x)/m

# print(T)

return T, costs,hypothesis

def normal\_eq(x, y):

return np.linalg.inv(x.T@x)@x.T@y

#特征值的标准归一化处理

class standard\_scalar:

def \_\_int\_\_(self):

self.mean = 0

self.std = 0

#用于计算训练集均值和方差，并返回训练集标准归一化结果

def fit\_transform(self,data):

self.mean = data.mean(axis=0)

self.std = data.std(axis=0)

data[::] = (data - self.mean) / self.std

#对测试集进行标准归一化

def transform(self,data):

data[::] = (data -self.mean) / self.std

# 引入数据集

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier # 导入决策树DTC包

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris # 导入方法类

iris = load\_iris() # 导入数据集iris

iris\_feature = iris.data # 特征数据

iris\_target = iris.target # 分类数据

clf = DecisionTreeClassifier() # 所以参数均置为默认状态

clf.fit(iris.data, iris.target) # 使用训练集训练模型

# print(clf)

predicted = clf.predict(iris.data)

# print(predicted)

# 获取花卉两列数据集

X = iris.data

L1 = [x[0] for x in X]

# print(L1)

L2 = [x[1] for x in X]

# print (L2)

# 绘图

plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o', label='setosa')

plt.scatter(X[50:100, 0], X[50:100, 1], color='blue', marker='x', label='versicolor')

plt.scatter(X[100:, 0], X[100:, 1], color='green', marker='s', label='Virginica')

plt.title("DTC")

plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.xticks(())

plt.yticks(())

plt.legend(loc=2)

plt.show()