这个星期的作业是：

1：完成贝叶斯里面代码实战，每人敲一次，并自己写注释

# -\*- coding: utf-8 -\*-

from sklearn.naive\_bayes import GaussianNB

from sklearn.datasets import load\_digits

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

#导入数据

digits = load\_digits()

X, y = digits.data, digits.target

Xtrain,Xtest,Ytrain,Ytest = train\_test\_split(X,y,test\_size=0.3,random\_state=420)

#建模

gnb = GaussianNB().fit(Xtrain,Ytrain)

#查看分数,calculate the accuracy score

acc\_score = gnb.score(Xtest,Ytest)

print(acc\_score)

#查看预测结果

Y\_pred = gnb.predict(Xtest)

print(Y\_pred)

#查看预测的概率结果

prob = gnb.predict\_proba(Xtest)

print(prob)

print(prob.shape)

#每一列对应一个标签下的概率

print(prob[1,:].sum())

#每一行的和都是一

print(prob.sum(axis=1))

#使用混淆矩阵来查看贝叶斯结果

from sklearn.metrics import confusion\_matrix as CM

print(CM(Ytest, Y\_pred))

from sklearn.preprocessing import MinMaxScaler

from sklearn.naive\_bayes import MultinomialNB, BernoulliNB

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

from sklearn.datasets import make\_blobs

from sklearn.metrics import brier\_score\_loss

import numpy as np

class\_1 = 500

class\_2 = 500 #两个类别分别设定500个样本

centers = [[0.0, 0.0], [2.0, 2.0]] #设定两个类别的中心

clusters\_std = [0.5, 0.5] #设定两个类别的方差

#生成数据集

X, y = make\_blobs(n\_samples=[class\_1, class\_2],centers=centers,

cluster\_std=clusters\_std,

random\_state=0, shuffle=False)

#分割数据集

Xtrain, Xtest, Ytrain, Ytest = train\_test\_split(X,y,test\_size=0.3,random\_state=420)

#先归一化，保证输入多项式朴素贝叶斯的特征矩阵中不带有负数（次数，频率等都为正数）

mms = MinMaxScaler().fit(Xtrain)

Xtrain\_ = mms.transform(Xtrain)

Xtest\_ = mms.transform(Xtest)

#建立一个多项式朴素贝叶斯分类器

mnb = MultinomialNB().fit(Xtrain\_, Ytrain)

#重要属性：调用根据数据获取的，每个标签类的对数先验概率log(P(Y))

#由于概率永远是在[0,1]之间，因此对数先验概率返回的永远是负值

print("对数先验概率mnb.class\_log\_prior\_=",mnb.class\_log\_prior\_)

print("mnb.class\_log\_prior\_.shape=",mnb.class\_log\_prior\_.shape)

#可以使用np.exp来查看真正的概率值

print("使用np.exp来查看真正的概率值 np.exp(mnb.class\_log\_prior\_)=",np.exp(mnb.class\_log\_prior\_))

# print("np.unique(Ytrain)=",np.unique(Ytrain)) #找出 Ytrain中不同的值

# print("Ytrain.shape = ",Ytrain.shape)

print("根据实际数目计算概率(Ytrain == 0).sum()/Ytrain.shape[0]=",(Ytrain == 0).sum()/Ytrain.shape[0])

print("根据实际数目计算概率 (Ytrain == 1).sum()/Ytrain.shape[0]=",(Ytrain == 1).sum()/Ytrain.shape[0])

#3) 伯努利朴素贝叶斯

from sklearn.naive\_bayes import BernoulliNB

#普通来说我们应该使用二值化的类sklearn.preprocessing.Binarizer来将特征一个个二值化 然而这样效率过低，因此我们选择归一化之后直接设置一个阈值

mms = MinMaxScaler().fit(Xtrain)

Xtrain\_ = mms.transform(Xtrain)

Xtest\_ = mms.transform(Xtest)

#A)不设置二值化

bnl\_ = BernoulliNB().fit(Xtrain\_, Ytrain) #模型拟合

print("准确分数： bnl.score(Xtest\_,Ytest) = ", bnl\_.score(Xtest\_,Ytest))

#B)设置二值化阈值为0.5

bnl = BernoulliNB(binarize=0.5).fit(Xtrain\_, Ytrain) #模型拟合

print("准确分数： bnl.score(Xtest\_,Ytest) = ", bnl.score(Xtest\_,Ytest))

2：完成XGBOOST 所以代码实战，每人敲一次，并自己写注释

# -\*- coding: utf-8 -\*-

import xgboost as xgb

from xgboost import XGBRegressor as XGBR

from sklearn.ensemble import RandomForestRegressor as RFR

from sklearn.linear\_model import LinearRegression as LinearR

from sklearn.datasets import load\_boston

from sklearn.model\_selection import KFold,cross\_val\_score as CVS,train\_test\_split as TTS

from sklearn.metrics import mean\_squared\_error as MSE

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from time import time

import datetime

data = load\_boston()

X = data.data

y = data.target

Xtrain,Xtest,Ytrain,Ytest = TTS(X,y,test\_size=0.3,random\_state=420)

#reg.predict(Xtest)

#reg.score(Xtest,Ytest)

#

#MSE(Ytest,reg.predict(Xtest))

#

#reg.feature\_importances\_

reg = XGBR(n\_estimators=100)

#CVS(reg,Xtrain,Ytrain,CV=5).mean()

#CVS(reg,Xtrain,Ytrain,cv=5,scoring='neg\_mean\_squared\_error').mean()

import sklearn

rfr = RFR(n\_estimators=100)

lr = LinearR()

reg = XGBR(n\_estimators=10,silent=False)

#作出XGBT的学习曲线

def plot\_learning\_curve(estimator,title,X,y,ax=None,ylim=None,cv=None,n\_jobs=None):

from sklearn.model\_selection import learning\_curve

import matplotlib.pyplot as plt

import numpy as np

train\_sizes,train\_scores,test\_scores = learning\_curve(estimator,X,y,shuffle=True,cv=cv,n\_jobs=n\_jobs)

if ax == None:

ax = plt.gca()

else:

ax = plt.figure()

ax.set\_title(title)

if ylim is not None:

ax.set\_ylim(\*ylim)

ax.set\_xlabel("Training examples")

ax.set\_ylabel("Score")

ax.grid()

ax.plot(train\_sizes,np.mean(train\_scores,axis=1),'o-',color='r',label="Trainning score")

ax.plot(train\_sizes,np.mean(test\_scores,axis=1),'o-',color='g',label="Test score")

ax.legend(loc='best')

return ax

cv = KFold(n\_splits=5, shuffle = True, random\_state=42)

plot\_learning\_curve(XGBR(n\_estimators=100,random\_state=420),"XGB",Xtrain,Ytrain,ax=None,cv=cv)

plt.show()

#画出(10,1010,50)范围区间内，每个estimators取值时的预测分数。

axisx = range(10,1010,50)

rs = []

for i in axisx:

reg = XGBR(n\_estimators=i,random\_state=420)

rs.append(CVS(reg,Xtrain,Ytrain,cv=cv).mean())

print(axisx[rs.index(max(rs))],max(rs))

plt.figure(figsize=(20,5))

plt.plot(axisx,rs,c="red",label="XGB")

plt.legend()

plt.show()

进化的学习曲线：方差与泛化误差

axisx = range(10,1010,50)

rs = []

var = []

ge = []

for i in axisx:

reg = XGBR(n\_estimators=i,random\_state=420)

cvresult = CVS(reg,Xtrain,Ytrain,cv=cv)

#记录偏差

rs.append(cvresult.mean())

#记录方差

var.append(cvresult.var())

#计算泛化误差可控部分

ge.append(1-cvresult.mean()\*\*2+cvresult.var())

#打印R2最高所对应的参数取值，并打印这个参数下的方差

print(axisx[rs.index(max(rs))],max(rs),var[rs.index(max(rs))])

print(axisx[var.index(min(var))],rs[var.index(min(var))],min(var))

print(axisx[ge.index(min(ge))],rs[ge.index(min(ge))],var[ge.index(min(ge))],min(ge))

plt.figure(figsize=(20,5))

plt.plot(axisx,rs,c="red",label="XGB")

plt.legend()

plt.show()

#细化学习曲线，找出最佳n\_estimators

axisx = range(10,300,10)

rs = []

var = []

ge = []

for i in axisx:

reg = XGBR(n\_estimators=i,random\_state=420)

cvresult = CVS(reg,Xtrain,Ytrain,cv=cv)

#记录偏差

rs.append(cvresult.mean())

#记录方差

var.append(cvresult.var())

#计算泛化误差可控部分

ge.append(1-cvresult.mean()\*\*2+cvresult.var())

#打印R2最高所对应的参数取值，并打印这个参数下的方差

print(axisx[rs.index(max(rs))],max(rs),var[rs.index(max(rs))])

print(axisx[var.index(min(var))],rs[var.index(min(var))],min(var))

print(axisx[ge.index(min(ge))],rs[ge.index(min(ge))],var[ge.index(min(ge))],min(ge))

rs = np.array(rs)

var = np.array(var)\*0.01

plt.figure(figsize=(20,5))

plt.plot(axisx,rs,c="black",label="XGB")

#添加方差线

plt.plot(axisx,rs+var,c="red",linestyle='-.')

plt.plot(axisx,rs-var,c="red",linestyle='-.')

plt.legend()

plt.show()

plt.figure(figsize=(20,5))

plt.plot(axisx,ge,c="gray",linestyle='-.')

plt.show()

#细化学习曲线，找出最佳subsample

axisx = np.linspace(0,1,20)

rs = []

#var = []

#ge = []

for i in axisx:

reg = XGBR(n\_estimators=180,subsample=i,random\_state=420)

rs.append(CVS(reg,Xtrain,Ytrain,cv=cv).mean())

print(axisx[rs.index(max(rs))],max(rs))

plt.figure(figsize=(20,5))

plt.plot(axisx,rs,c="black",label="XGB")

plt.legend()

plt.show()

#细化学习曲线

axisx = np.linspace(0.05,1,20)

rs = []

var = []

ge = []

for i in axisx:

reg = XGBR(n\_estimators=180,subsample=i,random\_state=420)

cvresult = CVS(reg,Xtrain,Ytrain,cv=cv)

rs.append(cvresult.mean())

#记录方差

var.append(cvresult.var())

#计算泛化误差可控部分

ge.append(1-cvresult.mean()\*\*2+cvresult.var())

#打印R2最高所对应的参数取值，并打印这个参数下的方差

print(axisx[rs.index(max(rs))],max(rs),var[rs.index(max(rs))])

print(axisx[var.index(min(var))],rs[var.index(min(var))],min(var))

print(axisx[ge.index(min(ge))],rs[ge.index(min(ge))],var[ge.index(min(ge))],min(ge))

rs = np.array(rs)

var = np.array(var)

plt.figure(figsize=(20,5))

plt.plot(axisx,rs,c="black",label="XGB")

#添加方差线

plt.plot(axisx,rs+var,c="red",linestyle='-.')

plt.plot(axisx,rs-var,c="red",linestyle='-.')

plt.legend()

plt.show()

axisx = np.linspace(0.75,1,25)

reg = XGBR(n\_estimators=180,subsample=0.7708,random\_state=420).fit(Xtrain,Ytrain)

reg.score(Xtest,Ytest)

MSE(Ytest,reg.predict(Xtest))

#首先我们先来定义一个评分函数，这个评分函数能够帮助我们直接打印Xtrain上的交叉验证结果

def regassess(reg,Xtrain,Ytrain,cv,scoring = ["r2"],show=True):

score = []

for i in range(len(scoring)):

if show:

print("{}:{:.2f}".format(scoring[i],CVS(reg,Xtrain,Ytrain,cv=cv,scoring=scoring[i]).mean()))

score.append(CVS(reg,Xtrain,Ytrain,cv=cv,scoring=scoring[i]).mean())

return score

#运行一下函数来看看效果

regassess(reg,Xtrain,Ytrain,cv,scoring = ["r2","neg\_mean\_squared\_error"])

#关闭打印功能试试看？

regassess(reg,Xtrain,Ytrain,cv,scoring = ["r2","neg\_mean\_squared\_error"],show=False)

#观察一下eta如何影响我们的模型：

from time import time

import datetime

for i in [0,0.2,0.5,1]:

time0=time()

reg = XGBR(n\_estimators=180,random\_state=420,learning\_rate=i)

print("learning\_rate = {}".format(i))

regassess(reg,Xtrain,Ytrain,cv,scoring = ["r2","neg\_mean\_squared\_error"])

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

print("\t")

axisx = np.arange(0.05,1,0.05)

rs = []

te = []

for i in axisx:

reg = XGBR(n\_estimators=180,random\_state=420,learning\_rate=i)

score = regassess(reg,Xtrain,Ytrain,cv,scoring = ["r2","neg\_mean\_squared\_error"],show=False)

test = reg.fit(Xtrain,Ytrain).score(Xtest,Ytest)

rs.append(score[0])

te.append(test)

print(axisx[rs.index(max(rs))],max(rs))

plt.figure(figsize=(20,5))

plt.plot(axisx,te,c="gray",label="XGB")

plt.plot(axisx,rs,c="green",label="XGB")

plt.legend()

plt.show()

for booster in ["gbtree","gblinear","dart"]:

reg = XGBR(n\_estimators=180,learning\_rate=0.1,random\_state=420,booster=booster).fit(Xtrain,Ytrain)

print(booster)

print(reg.score(Xtest,Ytest)) #自己找线性数据试试看"gblinear"的效果吧~

#默认reg:linear

reg = XGBR(n\_estimators=180,random\_state=420).fit(Xtrain,Ytrain)

reg.score(Xtest,Ytest)

MSE(Ytest,reg.predict(Xtest))

#xgb实现法

import xgboost as xgb

#使用类Dmatrix读取数据

dtrain = xgb.DMatrix(Xtrain,Ytrain)

dtest = xgb.DMatrix(Xtest,Ytest)

#非常遗憾无法打开来查看，所以通常都是先读到pandas里面查看之后再放到DMatrix中

dtrain

#写明参数，silent默认为False，通常需要手动将它关闭

param = {'silent':False,'objective':'reg:linear',"eta":0.1}

num\_round = 180

#类train，可以直接导入的参数是训练数据，树的数量，其他参数都需要通过params来导入

bst = xgb.train(param, dtrain, num\_round)

#接口predict

from sklearn.metrics import r2\_score

r2\_score(Ytest,bst.predict(dtest))

MSE(Ytest,bst.predict(dtest))

#使用网格搜索来查找最佳的参数组合

from sklearn.model\_selection import GridSearchCV

param = {"reg\_alpha":np.arange(0,5,0.05),"reg\_lambda":np.arange(0,2,0.05)}

gscv = GridSearchCV(reg,param\_grid = param,scoring = "neg\_mean\_squared\_error",cv=cv)

axisx = np.arange(0,5,0.05)

rs = []

var = []

ge = []

for i in axisx:

reg = XGBR(n\_estimators=180,random\_state=420,gamma=i)

result = CVS(reg,Xtrain,Ytrain,cv=cv)

rs.append(result.mean())

var.append(result.var())

ge.append((1 - result.mean())\*\*2+result.var())

print(axisx[rs.index(max(rs))],max(rs),var[rs.index(max(rs))])

print(axisx[var.index(min(var))],rs[var.index(min(var))],min(var))

print(axisx[ge.index(min(ge))],rs[ge.index(min(ge))],var[ge.index(min(ge))],min(ge))

rs = np.array(rs)

var = np.array(var)\*0.1

plt.figure(figsize=(20,5))

plt.plot(axisx,rs,c="black",label="XGB")

plt.plot(axisx,rs+var,c="red",linestyle='-.')

plt.plot(axisx,rs-var,c="red",linestyle='-.')

plt.legend()

plt.show()

import xgboost as xgb

#为了便捷，使用全数据

dfull = xgb.DMatrix(X,y)

#设定参数

param1 = {'silent':True,'obj':'reg:linear',"gamma":0}

num\_round = 180

n\_fold=5

#使用类xgb.cv

time0 = time()

cvresult1 = xgb.cv(param1, dfull, num\_round,n\_fold)

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

#看看类xgb.cv生成了什么结果？

cvresult1

plt.figure(figsize=(20,5))

plt.grid()

plt.plot(range(1,181),cvresult1.iloc[:,0],c="red",label="train,gamma=0")

plt.plot(range(1,181),cvresult1.iloc[:,2],c="orange",label="test,gamma=0")

plt.legend()

plt.show()

param1 = {'silent':True,'obj':'reg:linear',"gamma":0,"eval\_metric":"mae"}

cvresult1 = xgb.cv(param1, dfull, num\_round,n\_fold)

plt.figure(figsize=(20,5))

plt.grid()

plt.plot(range(1,181),cvresult1.iloc[:,0],c="red",label="train,gamma=0")

plt.plot(range(1,181),cvresult1.iloc[:,2],c="orange",label="test,gamma=0")

plt.legend()

plt.show()

param1 = {'silent':True,'obj':'reg:linear',"gamma":0}

param2 = {'silent':True,'obj':'reg:linear',"gamma":20}

num\_round = 180

n\_fold=5

time0 = time()

cvresult1 = xgb.cv(param1, dfull, num\_round,n\_fold)

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

time0 = time()

cvresult2 = xgb.cv(param2, dfull, num\_round,n\_fold)

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

plt.figure(figsize=(20,5))

plt.grid()

plt.plot(range(1,181),cvresult1.iloc[:,0],c="red",label="train,gamma=0")

plt.plot(range(1,181),cvresult1.iloc[:,2],c="orange",label="test,gamma=0")

plt.plot(range(1,181),cvresult2.iloc[:,0],c="green",label="train,gamma=20")

plt.plot(range(1,181),cvresult2.iloc[:,2],c="blue",label="test,gamma=20")

plt.legend()

plt.show()

from sklearn.datasets import load\_breast\_cancer

data2 = load\_breast\_cancer()

x2 = data2.data

y2 = data2.target

dfull2 = xgb.DMatrix(x2,y2)

param1 = {'silent':True,'obj':'binary:logistic',"gamma":0,"nfold":5}

param2 = {'silent':True,'obj':'binary:logistic',"gamma":2,"nfold":5}

num\_round = 100

time0 = time()

cvresult1 = xgb.cv(param1, dfull2, num\_round,metrics=("error"))

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

time0 = time()

cvresult2 = xgb.cv(param2, dfull2, num\_round,metrics=("error"))

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

plt.figure(figsize=(20,5))

plt.grid()

plt.plot(range(1,101),cvresult1.iloc[:,0],c="red",label="train,gamma=0")

plt.plot(range(1,101),cvresult1.iloc[:,2],c="orange",label="test,gamma=0")

plt.plot(range(1,101),cvresult2.iloc[:,0],c="green",label="train,gamma=2")

plt.plot(range(1,101),cvresult2.iloc[:,2],c="blue",label="test,gamma=2")

plt.legend()

plt.show()

dfull = xgb.DMatrix(X,y)

param1 = {'silent':True #并非默认

,'obj':'reg:linear' #并非默认

,"subsample":1

,"max\_depth":6

,"eta":0.3

,"gamma":0

,"lambda":1

,"alpha":0

,"colsample\_bytree":1

,"colsample\_bylevel":1

,"colsample\_bynode":1

,"nfold":5}

num\_round = 200

time0 = time()

cvresult1 = xgb.cv(param1, dfull, num\_round)

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

fig,ax = plt.subplots(1,figsize=(15,10))

#ax.set\_ylim(top=5)

ax.grid()

ax.plot(range(1,201),cvresult1.iloc[:,0],c="red",label="train,original")

ax.plot(range(1,201),cvresult1.iloc[:,2],c="orange",label="test,original")

ax.legend(fontsize="xx-large")

plt.show()

param1 = {'silent':True

,'obj':'reg:linear'

,"subsample":1

,"max\_depth":6

,"eta":0.3

,"gamma":0

,"lambda":1

,"alpha":0

,"colsample\_bytree":1

,"colsample\_bylevel":1

,"colsample\_bynode":1

,"nfold":5}

num\_round = 200

time0 = time()

cvresult1 = xgb.cv(param1, dfull, num\_round)

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

fig,ax = plt.subplots(1,figsize=(15,8))

ax.set\_ylim(top=5)

ax.grid()

ax.plot(range(1,201),cvresult1.iloc[:,0],c="red",label="train,original")

ax.plot(range(1,201),cvresult1.iloc[:,2],c="orange",label="test,original")

param2 = {'silent':True

,'obj':'reg:linear'

,"nfold":5}

param3 = {'silent':True

,'obj':'reg:linear'

,"nfold":5}

time0 = time()

cvresult2 = xgb.cv(param2, dfull, num\_round)

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

time0 = time()

cvresult3 = xgb.cv(param3, dfull, num\_round)

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

ax.plot(range(1,201),cvresult2.iloc[:,0],c="green",label="train,last")

ax.plot(range(1,201),cvresult2.iloc[:,2],c="blue",label="test,last")

ax.plot(range(1,201),cvresult3.iloc[:,0],c="gray",label="train,this")

ax.plot(range(1,201),cvresult3.iloc[:,2],c="pink",label="test,this")

ax.legend(fontsize="xx-large")

plt.show()

#默认设置

param1 = {'silent':True

,'obj':'reg:linear'

,"subsample":1

,"max\_depth":6

,"eta":0.3

,"gamma":0

,"lambda":1

,"alpha":0

,"colsample\_bytree":1

,"colsample\_bylevel":1

,"colsample\_bynode":1

,"nfold":5}

num\_round = 200

time0 = time()

cvresult1 = xgb.cv(param1, dfull, num\_round)

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

fig,ax = plt.subplots(1,figsize=(15,8))

ax.set\_ylim(top=5)

ax.grid()

ax.plot(range(1,201),cvresult1.iloc[:,0],c="red",label="train,original")

ax.plot(range(1,201),cvresult1.iloc[:,2],c="orange",label="test,original")

#调参结果1

param2 = {'silent':True

,'obj':'reg:linear'

,"subsample":1

,"eta":0.05

,"gamma":20

,"lambda":3.5

,"alpha":0.2

,"max\_depth":4

,"colsample\_bytree":0.4

,"colsample\_bylevel":0.6

,"colsample\_bynode":1

,"nfold":5}

param3 = {'silent':True

,'obj':'reg:linear'

,"max\_depth":2

,"eta":0.05

,"gamma":0

,"lambda":1

,"alpha":0

,"colsample\_bytree":1

,"colsample\_bylevel":0.4

,"colsample\_bynode":1

,"nfold":5}

time0 = time()

cvresult2 = xgb.cv(param2, dfull, num\_round)

print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))

ax.plot(range(1,201),cvresult2.iloc[:,0],c="green",label="train,final")

ax.plot(range(1,201),cvresult2.iloc[:,2],c="blue",label="test,final")

ax.legend(fontsize="xx-large")

plt.show()

import pickle

dtrain = xgb.DMatrix(Xtrain,Ytrain)

#设定参数，对模型进行训练

param = {'silent':True

,'obj':'reg:linear'

,"subsample":1

,"eta":0.05

,"gamma":20

,"lambda":3.5

,"alpha":0.2

,"max\_depth":4

,"colsample\_bytree":0.4

,"colsample\_bylevel":0.6

,"colsample\_bynode":1}

num\_round = 180

bst = xgb.train(param, dtrain, num\_round)

#保存模型

pickle.dump(bst, open("xgboostonboston.dat","wb"))

#注意，open中我们往往使用w或者r作为读取的模式，但其实w与r只能用于文本文件，当我们希望导入的不是文本文件，而是模型本身的时候，我们使用"wb"和"rb"作为读取的模式。其中wb表示以二进制写入，rb表示以二进制读入

#看看模型被保存到了哪里？

import sys

sys.path

#重新打开jupyter lab

from sklearn.datasets import load\_boston

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

from sklearn.metrics import mean\_squared\_error as MSE

import pickle

import xgboost as xgb

data = load\_boston()

X = data.data

y = data.target

Xtrain,Xtest,Ytrain,Ytest = TTS(X,y,test\_size=0.3,random\_state=420)

#注意，如果我们保存的模型是xgboost库中建立的模型，则导入的数据类型也必须是xgboost库中的数据类型

dtest = xgb.DMatrix(Xtest,Ytest)

#导入模型

loaded\_model = pickle.load(open("xgboostonboston.dat", "rb"))

print("Loaded model from: xgboostonboston.dat")

#做预测

ypreds = loaded\_model.predict(dtest)

from sklearn.metrics import mean\_squared\_error as MSE, r2\_score

MSE(Ytest,ypreds)

r2\_score(Ytest,ypreds)

bst = xgb.train(param, dtrain, num\_round)

import joblib

#同样可以看看模型被保存到了哪里

joblib.dump(bst,"xgboost-boston.dat")

loaded\_model = joblib.load("xgboost-boston.dat")

ypreds = loaded\_model.predict(dtest)

MSE(Ytest, ypreds)

r2\_score(Ytest,ypreds)

#使用sklearn中的模型

from xgboost import XGBRegressor as XGBR

bst = XGBR(n\_estimators=200

,eta=0.05,gamma=20

,reg\_lambda=3.5

,reg\_alpha=0.2

,max\_depth=4

,colsample\_bytree=0.4

,colsample\_bylevel=0.6).fit(Xtrain,Ytrain)

joblib.dump(bst,"xgboost-boston.dat")

loaded\_model = joblib.load("xgboost-boston.dat")

#则这里可以直接导入Xtest

ypreds = loaded\_model.predict(Xtest)

MSE(Ytest, ypreds)

import numpy as np

import xgboost as xgb

import matplotlib.pyplot as plt

from xgboost import XGBClassifier as XGBC

from sklearn.datasets import make\_blobs

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

from sklearn.metrics import confusion\_matrix as cm

from sklearn.metrics import recall\_score as recall

from sklearn.metrics import roc\_auc\_score as auc

class\_1 = 500 #类别1有500个样本

class\_2 = 50 #类别2只有50个

centers = [[0.0, 0.0], [2.0, 2.0]] #设定两个类别的中心

clusters\_std = [1.5, 0.5] #设定两个类别的方差，通常来说，样本量比较大的类别会更加松散

X, y = make\_blobs(n\_samples=[class\_1, class\_2],

centers=centers,

cluster\_std=clusters\_std,

random\_state=0, shuffle=False)

Xtrain, Xtest, Ytrain, Ytest = TTS(X,y,test\_size=0.3,random\_state=420)

(y == 1).sum() / y.shape[0]

#在sklearn下建模#

clf = XGBC().fit(Xtrain,Ytrain)

ypred = clf.predict(Xtest)

clf.score(Xtest,Ytest)

cm(Ytest,ypred,labels=[1,0])

recall(Ytest,ypred)

auc(Ytest,clf.predict\_proba(Xtest)[:,1])

#负/正样本比例

clf\_ = XGBC(scale\_pos\_weight=10).fit(Xtrain,Ytrain)

ypred\_ = clf\_.predict(Xtest)

clf\_.score(Xtest,Ytest)

cm(Ytest,ypred\_,labels=[1,0])

recall(Ytest,ypred\_)

auc(Ytest,clf\_.predict\_proba(Xtest)[:,1])

#随着样本权重逐渐增加，模型的recall,auc和准确率如何变化？

for i in [1,5,10,20,30]:

clf\_ = XGBC(scale\_pos\_weight=i).fit(Xtrain,Ytrain)

ypred\_ = clf\_.predict(Xtest)

print(i)

print("\tAccuracy:{}".format(clf\_.score(Xtest,Ytest)))

print("\tRecall:{}".format(recall(Ytest,ypred\_)))

print("\tAUC:{}".format(auc(Ytest,clf\_.predict\_proba(Xtest)[:,1])))

dtrain = xgb.DMatrix(Xtrain,Ytrain)

dtest = xgb.DMatrix(Xtest,Ytest)

#看看xgboost库自带的predict接口

param= {'silent':True,'objective':'binary:logistic',"eta":0.1,"scale\_pos\_weight":1}

num\_round = 100

bst = xgb.train(param, dtrain, num\_round)

preds = bst.predict(dtest)

#看看preds返回了什么？

preds

#自己设定阈值

ypred = preds.copy()

ypred[preds > 0.5] = 1

ypred[ypred != 1] = 0

#写明参数

scale\_pos\_weight = [1,5,10]

names = ["negative vs positive: 1"

,"negative vs positive: 5"

,"negative vs positive: 10"]

#导入模型评估指标

from sklearn.metrics import accuracy\_score as accuracy, recall\_score as recall, roc\_auc\_score as auc

for name,i in zip(names,scale\_pos\_weight):

param= {'silent':True,'objective':'binary:logistic',"eta":0.1,"scale\_pos\_weight":i}

clf = xgb.train(param, dtrain, num\_round)

preds = clf.predict(dtest)

ypred = preds.copy()

ypred[preds > 0.5] = 1

ypred[ypred != 1] = 0

print(name)

print("\tAccuracy:{}".format(accuracy(Ytest,ypred)))

print("\tRecall:{}".format(recall(Ytest,ypred)))

print("\tAUC:{}".format(auc(Ytest,preds)))

#当然我们也可以尝试不同的阈值

for name,i in zip(names,scale\_pos\_weight):

for thres in [0.3,0.5,0.7,0.9]:

param= {'silent':True,'objective':'binary:logistic',"eta":0.1,"scale\_pos\_weight":i}

clf = xgb.train(param, dtrain, num\_round)

preds = clf.predict(dtest)

ypred = preds.copy()

ypred[preds > thres] = 1

ypred[ypred != 1] = 0

print("{},thresholds:{}".format(name,thres))

print("\tAccuracy:{}".format(accuracy(Ytest,ypred)))

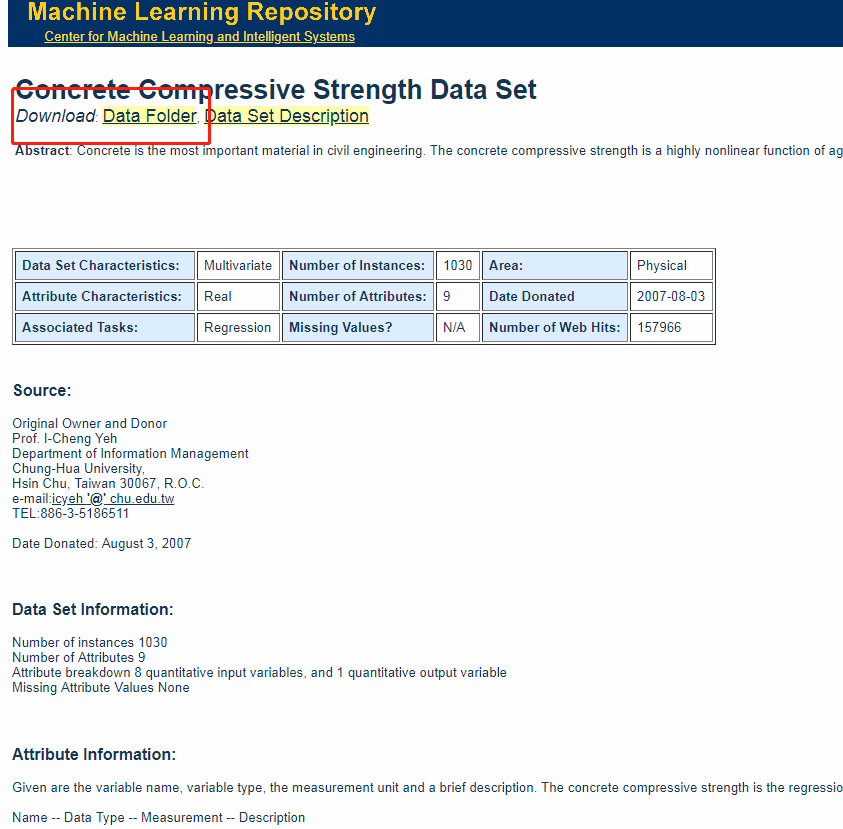
print("\tRecall:{}".format(recall(Ytest,ypred)))

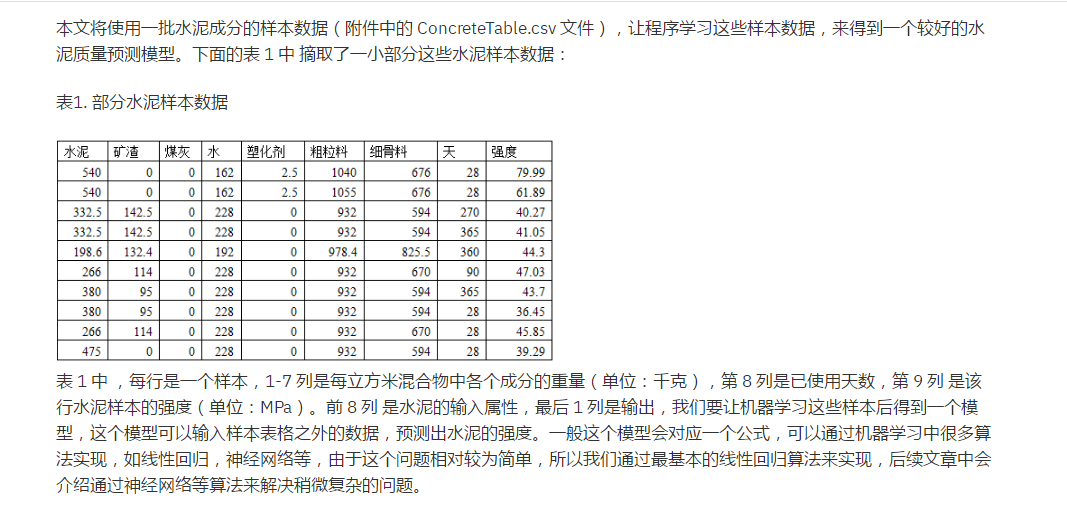
print("\tAUC:{}".format(auc(Ytest,preds)))

3：课外代码实战（待定，）

完成时间：2019年4月7号，23点30分！

本周的实战作业是：（算法模型自由选择）<http://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength>





# -\*- coding: utf-8 -\*-

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from pandas import DataFrame,Series

#from sklearn.cross\_validation import train\_test\_split

from sklearn.model\_selection import KFold, cross\_val\_score as CVS, train\_test\_split as TTS

#from sklearn.linear\_model import LinearRegression as LR

#from sklearn.logistic\_model import LogisticRegression as LR

from sklearn.linear\_model import LogisticRegression as LR

from sklearn.linear\_model import RandomizedLogisticRegression as RLR

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

#读取文件

datafile = u'D:\Machine Learning\9 Homework\Concrete\_Data.xls'#文件所在位置，u为防止路径中有中文名称，此处没有，可以省略

data = pd.read\_excel(datafile)#datafile是excel文件，所以用read\_excel,如果是csv文件则用read\_csv

examDf = DataFrame(data)

examDf.head()

exam\_X = examDf.iloc[: , :-1].values

exam\_Y = examDf.iloc[:,-1].values

X\_train,X\_test,Y\_train,Y\_test = TTS(exam\_X,exam\_Y,train\_size=.3)#X\_train为训练数据标签,X\_test为测试数据标签,exam\_X为样本特征,exam\_y为样本标签，train\_size 训练数据占比

clf = DecisionTreeRegressor(random\_state=0)

rfr = RandomForestRegressor(random\_state=0)

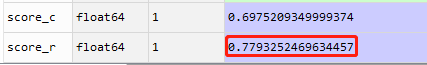
clf = clf.fit(X\_train,Y\_train)

rfr = rfr.fit(X\_train,Y\_train)

score\_c = clf.score(X\_test,Y\_test)

score\_r = rfr.score(X\_test,Y\_test)

Result:



############################调参######################

# -\*- coding: utf-8 -\*-

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from pandas import DataFrame,Series

#from sklearn.cross\_validation import train\_test\_split

from sklearn.model\_selection import KFold, cross\_val\_score as CVS, train\_test\_split as TTS

#from sklearn.linear\_model import LinearRegression as LR

#from sklearn.logistic\_model import LogisticRegression as LR

from sklearn.linear\_model import LogisticRegression as LR

from sklearn.linear\_model import RandomizedLogisticRegression as RLR

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

#读取文件

datafile = u'D:\Machine Learning\9 Homework\Concrete\_Data.xls'#文件所在位置，u为防止路径中有中文名称，此处没有，可以省略

data = pd.read\_excel(datafile)#datafile是excel文件，所以用read\_excel,如果是csv文件则用read\_csv

examDf = DataFrame(data)

examDf.head()

exam\_X = examDf.iloc[: , :-1].values

exam\_Y = examDf.iloc[:,-1].values

X\_train,X\_test,Y\_train,Y\_test = TTS(exam\_X,exam\_Y,train\_size=.3)#X\_train为训练数据标签,X\_test为测试数据标签,exam\_X为样本特征,exam\_y为样本标签，train\_size 训练数据占比

#clf = DecisionTreeRegressor(random\_state=0)

#rfr = RandomForestRegressor(n\_estimators=50,random\_state=0)

#clf = clf.fit(X\_train,Y\_train)

#rfr = rfr.fit(X\_train,Y\_train)

#score\_c = clf.score(X\_test,Y\_test)

#score\_r = rfr.score(X\_test,Y\_test)

from sklearn.model\_selection import cross\_val\_score

scorel = []

for i in range(10,200):

rfr = RandomForestRegressor(n\_estimators=i,n\_jobs=-1,random\_state=90)

# clf = clf.fit(X\_train,Y\_train)

rfr = rfr.fit(X\_train,Y\_train)

# score\_c = clf.score(X\_test,Y\_test)

score = rfr.score(X\_test,Y\_test)

# score = cross\_val\_score(rfr,data.data,data.target,cv=10).mean()

scorel.append(score)

print(max(scorel),([\*range(10,200)][scorel.index(max(scorel))]))

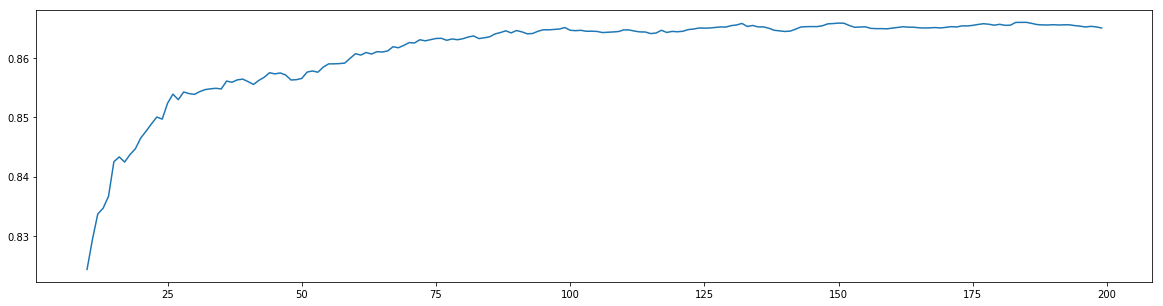
plt.figure(figsize=[20,5])

plt.plot(range(10,200),scorel)

plt.show()

n\_estimators调参结果如下：

**0.866053489354115 185**



继续调参：

# -\*- coding: utf-8 -\*-

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from pandas import DataFrame,Series

#from sklearn.cross\_validation import train\_test\_split

from sklearn.model\_selection import KFold, cross\_val\_score as CVS, train\_test\_split as TTS

#from sklearn.linear\_model import LinearRegression as LR

#from sklearn.logistic\_model import LogisticRegression as LR

from sklearn.linear\_model import LogisticRegression as LR

from sklearn.linear\_model import RandomizedLogisticRegression as RLR

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

#读取文件

datafile = u'D:\Machine Learning\9 Homework\Concrete\_Data.xls'#文件所在位置，u为防止路径中有中文名称，此处没有，可以省略

data = pd.read\_excel(datafile)#datafile是excel文件，所以用read\_excel,如果是csv文件则用read\_csv

examDf = DataFrame(data)

examDf.head()

exam\_X = examDf.iloc[: , :-1].values

exam\_Y = examDf.iloc[:,-1].values

X\_train,X\_test,Y\_train,Y\_test = TTS(exam\_X,exam\_Y,train\_size=.3)#X\_train为训练数据标签,X\_test为测试数据标签,exam\_X为样本特征,exam\_y为样本标签，train\_size 训练数据占比

#clf = DecisionTreeRegressor(random\_state=0)

#rfr = RandomForestRegressor(n\_estimators=50,random\_state=0)

#clf = clf.fit(X\_train,Y\_train)

#rfr = rfr.fit(X\_train,Y\_train)

#score\_c = clf.score(X\_test,Y\_test)

#score\_r = rfr.score(X\_test,Y\_test)

from sklearn.model\_selection import cross\_val\_score

scorel = []

for i in range(10,300):

rfr = RandomForestRegressor(n\_estimators=i,n\_jobs=-1,random\_state=430)

# clf = clf.fit(X\_train,Y\_train)

rfr = rfr.fit(X\_train,Y\_train)

# score\_c = clf.score(X\_test,Y\_test)

score = rfr.score(X\_test,Y\_test)

# score = cross\_val\_score(rfr,data.data,data.target,cv=10).mean()

scorel.append(score)

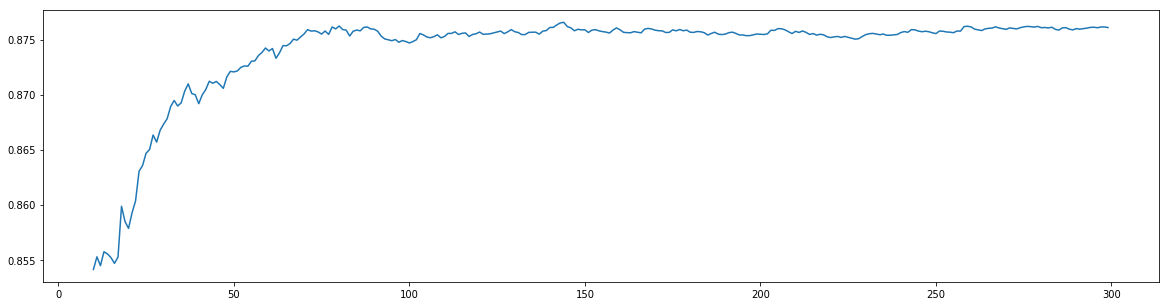
print(max(scorel),([\*range(10,300)][scorel.index(max(scorel))]))

plt.figure(figsize=[20,5])

plt.plot(range(10,300),scorel)

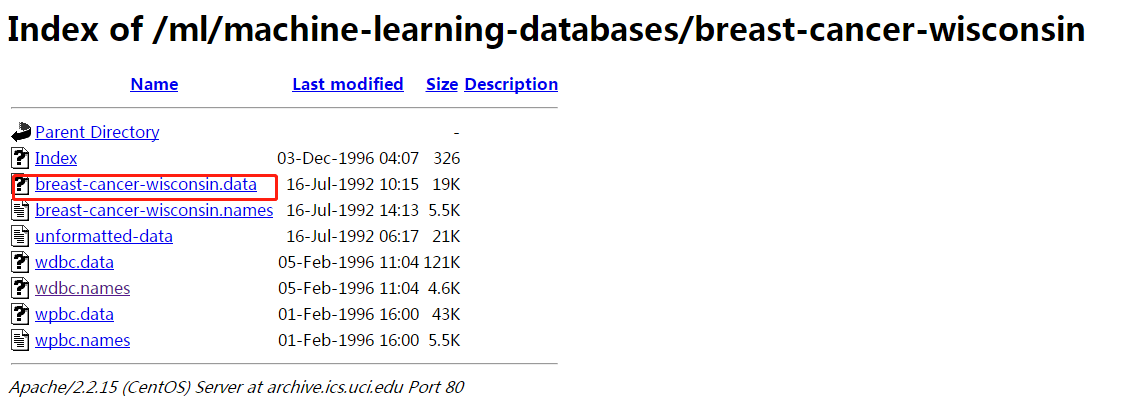
plt.show()

**0.8765644095350895 144**



######################################################

4、 <http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>



方法一：决策树与随机森林分类算法对比

# -\*- coding: utf-8 -\*-

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from pandas import DataFrame,Series

#from sklearn.cross\_validation import train\_test\_split

from sklearn.model\_selection import KFold, cross\_val\_score as CVS, train\_test\_split as TTS

from sklearn.linear\_model import LinearRegression as LR

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_wine

from sklearn.impute import SimpleImputer

#from sklearn.datasets import load\_breast\_cancer\_wisconsin

wine = load\_wine()

#读取文件

#datafile = u'D:/Machine Learning/9 Homework/breast-cancer-wisconsin.cvs'#文件所在位置，u为防止路径中有中文名称，此处没有，可以省略

data = pd.read\_csv('D:\\Machine Learning\\9 Homework\\breast-cancer-wisconsin-2.csv') # 正样本数据

data = data.replace(to\_replace = "?", value = np.nan)

# then drop the missing value

data = data.dropna(how = 'any')

exam\_X = data.iloc[:, :-1].values

exam\_Y = data.iloc[:,-1].values

Xtrain, Xtest, Ytrain, Ytest = TTS(exam\_X,exam\_Y,test\_size=0.3)

clf = DecisionTreeClassifier(random\_state=0)

rfc = RandomForestClassifier(random\_state=0)

clf = clf.fit(Xtrain,Ytrain)

rfc = rfc.fit(Xtrain,Ytrain)

score\_c = clf.score(Xtest,Ytest)

score\_r = rfc.score(Xtest,Ytest)

result:



方法二：逻辑回归与SVM对比

# import the packets

import numpy as np

import pandas as pd

DATA\_PATH = "D:\\Machine Learning\\9 Homework\\breast-cancer-wisconsin.data"

#DATA\_PATH = "D:\\Machine Learning\\9 Homework\\breast-cancer-wisconsin-2.data"

# create the column names

columnNames = [

'Sample code number',

'Clump Thickness',

'Uniformity of Cell Size',

'Uniformity of Cell Shape',

'Marginal Adhesion',

'Single Epithelial Cell Size',

'Bare Nuclei',

'Bland Chromatin',

'Normal Nucleoli',

'Mitoses',

'Class'

]

data = pd.read\_csv(DATA\_PATH, names = columnNames)

# show the shape of data

# print data.shape

# use standard missing value to replace "?"

data = data.replace(to\_replace = "?", value = np.nan)

# then drop the missing value

data = data.dropna(how = 'any')

# print data.shape

# then we split this dataset in to 2 parts:

# - train dataset

# - test dataset

# here we use `train\_test\_split` to split data

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data[ columnNames[1:10] ], # features

data[ columnNames[10] ], # labels

test\_size = 0.25,

random\_state = 43

)

# let's have a look at the distribution of the train data

# print y\_train.value\_counts()

# and the test's dataset

# print y\_test.value\_counts()

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

X\_train = ss.fit\_transform(X\_train)

X\_test = ss.transform(X\_test)

# use logestic-regression

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train, y\_train)

lr\_y = lr.predict(X\_test)

# use svm

from sklearn.svm import LinearSVC

lsvc = LinearSVC()

lsvc.fit(X\_train, y\_train)

svm\_y = lsvc.predict(X\_test)

# now we will check the performance of the classifier

from sklearn.metrics import classification\_report

# use the classification\_report to present result

# `.score` method can be used to test the accuracy

print ('Accuracy of the LogesticRegression: ',lr.score(X\_test, y\_test))

print (classification\_report(y\_test, lr\_y, target\_names = ['Benign', 'Malignant']))

# print 'Accuracy on the train dataset: ', lr.score(X\_train, y\_train)

# print 'Accuracy on the predict result (should be 1.0): ', lr.score(X\_test, lr\_y)

print ('Accuracy of the SVM: ' , lsvc.score(X\_test, y\_test))

print (classification\_report(y\_test, svm\_y, target\_names = ['Benign', 'Malignant']))

result:

