# 新建dataframe

新建

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[7,8,9]],columns = ["feature\_1", "feature\_2", "label"])

读取

import pandas as pd

df = pd.read\_csv("datas/hour.csv", sep=",")

# 删除dataframe列

del df["instant"]

df.drop(columns=["instant","dteday"])

# 修改dataframe列名

暴力

a.columns = ['a','b','c']

较好的方法

a.rename(columns={'A':'a', 'B':'b', 'C':'c'}, inplace = True)

# 查看dataframe字段信息

a.info()

# 修改dataframe列类型

df["instant"] = df["instant"].astype("object")

X[['Global\_active\_power',"b"]] = X[['Global\_active\_power',"b"]].astype('float64')

# 查看dataframe统计信息

a.describe()

# 获取dataframe部分列

a.iloc[:,0:3]

df.iloc[:,[-1]]

a[["feature\_1", "feature\_2"]]

# 获取dataframe列名

df.columns返回一个可迭代对象

for i in df.columns:

print(i)

# 获取dataframe的Series

一行

a.iloc[0,:]

一列

a.iloc[:,1]

a["feature\_1"]

# 合并dataframe

横向

pd.concat([a,a],axis=1)

纵向

pd.concat([a,a],axis=0)

# 数据去重

import pandas as pd

df = pd.DataFrame([[1,2,3],[2,3,4],[1,2,3]])

df.drop\_duplicates(inplace=True)

df

# 替换DF中的字符串

#df.int\_rate.replace('%','',inplace = True, regex = True)

a.replace('%','',inplace = True, regex = True)

# Dataframe copy

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[7,8,9]],columns = ["feature\_1", "feature\_2", "label"])

b = a.copy()

b.drop(columns=["feature\_1"],inplace=True)

a

# 分割训练集与测试集

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

X = a.iloc[:,0:-1]

Y = a["label"]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.5,random\_state=0)

Y\_train

# 统计Series值出现次数

a["feature\_1"].value\_counts()

# 异常数据处理

删除

a.replace('?', np.nan).dropna(how = 'any')

# 独热编码

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[1,8,9]],columns = ["feature\_1", "feature\_2", "label"])

from sklearn.preprocessing import OneHotEncoder

hotCoder=OneHotEncoder(sparse = False, handle\_unknown = "ignore")

hot = hotCoder.fit\_transform(a)

pd.DataFrame(hot)

b = pd.DataFrame([[1,2,3],

[4,5,6],

[10,8,9]],columns = ["feature\_1", "feature\_2", "label"])

hotCoder.transform(b)

# 多项式扩展

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[1,8,9]],columns = ["feature\_1", "feature\_2", "label"])

from sklearn.preprocessing import PolynomialFeatures

polyCoder = PolynomialFeatures(degree=2, include\_bias=True, interaction\_only=False)

df = polyCoder.fit\_transform(a)

pd.DataFrame(df, columns=polyCoder.get\_feature\_names())

# 标准化

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[7,8,9]],columns = ["feature\_1", "feature\_2", "label"])

from sklearn.preprocessing import StandardScaler

ssCoder = StandardScaler()

df = ssCoder.fit\_transform(a)

pd.DataFrame(df)

# 规范化，归一化

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[7,8,9]],columns = ["feature\_1", "feature\_2", "label"])

from sklearn.preprocessing import MinMaxScaler

ssCoder = MinMaxScaler(feature\_range=[-1,2])

df = ssCoder.fit\_transform(a)

pd.DataFrame(df)

# LabelEncoder

from sklearn.preprocessing import LabelEncoder

import pandas as pd

a = pd.DataFrame([["b",2,3],

["a",5,6],

["a",8,9]],columns = ["feature\_1", "feature\_2", "label"])

laCoder = LabelEncoder()

b = pd.DataFrame(laCoder.fit\_transform(a["feature\_1"]))

pd.concat([a,b],axis=1)

# dataframe样本采样

df = a.sample(frac=0.66)

df = a.sample(n=3)

pd.concat([a,df])

# LinearRegression

import numpy as np

X = np.mat([[1,1],[2,1],[3,1],[4,1]])

Y = np.mat([[3.2],[4.7],[7.3],[8.5]])

from sklearn.linear\_model import LinearRegression

model = LinearRegression(fit\_intercept=False)

model.fit(X,Y)

model.coef\_

model.score(X,Y)

# Ridge

from sklearn.linear\_model import Ridge

for alpha in [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 2, 3, 5, 10]:

clf = Ridge(alpha=alpha, max\_iter=2000, solver="auto",fit\_intercept=True)

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

print("Ridge:",mse(Y\_test.values, clf.predict(X\_test)))

print(clf.n\_iter\_)

# Lasso

from sklearn.linear\_model import Lasso

for alpha in [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 2, 3]:

clf = Lasso(alpha=alpha, max\_iter=100, fit\_intercept=True)

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

print("Lasso:",mse(Y\_test.values, clf.predict(X\_test)))

print(clf.n\_iter\_)

# 填充缺失值

from sklearn.preprocessing import Imputer

im = Imputer()

im.fit\_transform(df)

# 模型评估

from sklearn.metrics import mean\_squared\_error

print("LinearRegression:",mean\_squared\_error(Y\_test.values, clf.predict(X\_test)))

# 混淆矩阵

pd.crosstab(Y\_test,knn.predict(X\_test),rownames=["label"],colnames=["predict"])

# 保存模型

from sklearn.externals import joblib

joblib.dump(enc,'rf.model')

enc2 = joblib.load('rf.model')

b = enc2.transform(a).toarray()

pd.DataFrame(b)

# 绘制函数图像

import numpy as np

import matplotlib.pyplot as plt

x=np.linspace(-5,5,1000) #这个表示在-5到5之间生成1000个x值

y=[1/(1+np.exp(-i)) for i in x] #对上述生成的1000个数循环用sigmoid公式求对应的y

plt.plot(x,y) #用上述生成的1000个xy值对生成1000个点

plt.show() #绘制图像

# Df拷贝

import pandas as pd

a = pd.DataFrame([[1,2,3],

[4,5,6],

[7,8,9]],columns = ["feature\_1", "feature\_2", "label"])

df = a.copy()

df.drop(columns=["feature\_1"],inplace=True)

print(id(a))

print(id(df))

a

# Python拷贝

import copy

a = [1,2,[1,2]]

b = copy.deepcopy(a)

a[2][0] = -1

b

# CountVectorizer

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer

corpus = [

'我 爱 你',

'我 恨 你'

]

y = [0,1]

vectorizer = CountVectorizer(token\_pattern="[a-zA-Z|\u4e00-\u9fa5]+")

count = vectorizer.fit\_transform(corpus)

print(vectorizer.get\_feature\_names())

print(count.toarray())

transformer = TfidfTransformer()

tfidf\_matrix = transformer.fit\_transform(count)

print(tfidf\_matrix.toarray())

tfidf\_vec = TfidfVectorizer(token\_pattern="[a-zA-Z|\u4e00-\u9fa5]+")

tfidf\_matrix = tfidf\_vec.fit\_transform(corpus)

print(tfidf\_matrix.toarray())

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

model.fit(tfidf\_matrix.toarray(),y)

print(model.predict(tfidf\_matrix.toarray()))

corpus = [

'仇 恨',

'爱 你'

]

tfidf\_matrix = tfidf\_vec.transform(corpus)

model.predict(tfidf\_matrix.toarray())

# TfidfVectorizer

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer

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

df = pd.read\_csv("datas/bayes.txt",header=None)

X = df[1]

Y = df[0]

tfCoder = TfidfVectorizer(token\_pattern="[a-zA-Z|\u4e00-\u9fa5]+")

X = tfCoder.fit\_transform(X).toarray()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0, random\_state=42)

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

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

print(model.predict(X\_train))

print(y\_train.values)

# apply

from sklearn import preprocessing

import pandas as pd

enc = preprocessing.OneHotEncoder(categorical\_features=[0,1])

a = pd.DataFrame([[1,"A","a"],

[0,"B","b"],

[2,"C","c"]],columns = ["ebayno", "p\_sku", "sale"])

def f(x):

i = x.index

v = x.values\*2

print(v)

return pd.Series(v,i)

a.apply(f)

**import** pandas **as** pd  
  
a = pd.DataFrame([[1, **"A"**, **"a"**],  
 [0, **"B"**, **"bB"**],  
 [2, **"C"**, **"cC"**]])  
  
**def** f(x):  
  
 **if** x[1] **not in** x[2]:  
 **return** x  
 **else**:  
 **return** pd.Series()  
  
a = a.apply(f, axis=1)  
print(a.dropna())

# **[Numpy中的矩阵合并](https://www.cnblogs.com/catmelo/p/4292960.html)**

列合并/扩展：**np.column\_stack()**

行合并/扩展：**np.row\_stack()**

# numpy.ravel() 与numpy.flatten()

****numpy.flatten()返回一份拷贝****，对拷贝所做的修改不会影响（reflects）原始矩阵，   
****numpy.ravel()返回的是视图****（view，也颇有几分C/C++引用reference的意味），会影响（reflects）原始矩阵。

# Python pandas数据分析中常用方法

## https://blog.csdn.net/qq\_16234613/article/details/64217337

# 重置索引

import pandas as pd

import numpy as np

path = r"datas\iris.data"

names = ['sepal length', 'sepal width', 'petal length', 'petal width', 'cla']

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

print(df.shape)

df.drop\_duplicates(inplace=True)

print(df.shape)

df.reset\_index(inplace=True)

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['cla2'] = pd.DataFrame(le.fit\_transform(df['cla']))

df

df = df.drop(columns=['cla'])

# df

X = df.iloc[:,:-1]

y = df.iloc[:,-1]

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

X = pd.DataFrame(ss.fit\_transform(X))

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

train\_X,test\_X,train\_y,test\_y = train\_test\_split(X,y,test\_size=0.33,random\_state=0)

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(train\_X,train\_y)

print('acc:',knn.score(train\_X,train\_y))

print('acc:',knn.score(test\_X,test\_y))

# tfidf

corpus=["hi peter",

"hi tom"]

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf2 = TfidfVectorizer(norm=None)

re = tfidf2.fit\_transform(corpus)

print(tfidf2.vocabulary\_)

print(tfidf2.get\_feature\_names())

print(re.todense())

# jupyter notebook 快捷键

#将代码块分割：点到选中的行Ctrl+Shift+-

#将代码块合并：使用Shift选中需要合并的框，Shift+m

#在代码块前增加新代码块，按a；在代码块后增加新代码块，按b；

#删除代码块，按dd

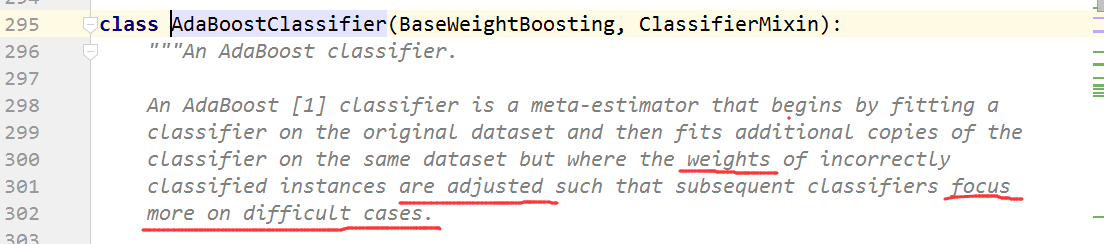
#运行当前代码块，Ctrl+Enter

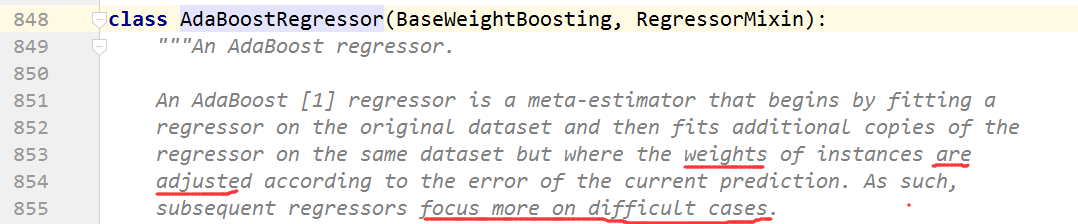
#运行当前代码块并选中下一个代码块（没有就创建），Shift+Enter

清除缓存kernel -> restart

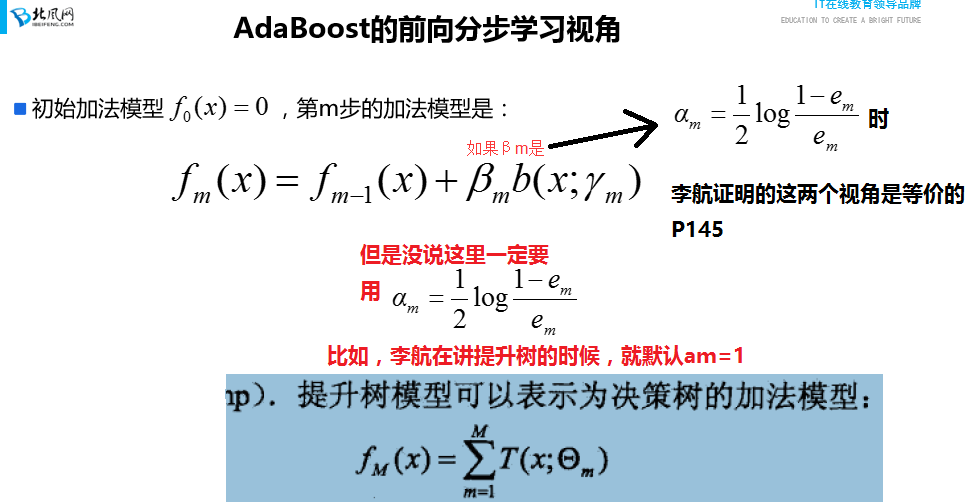
Jupyter的优点是允许将变量放到内存中，可以直接进行类型推断

1.首先sklearn都是从带权学习的视角出发的这是毋庸置疑的了，源码或官网上都写了



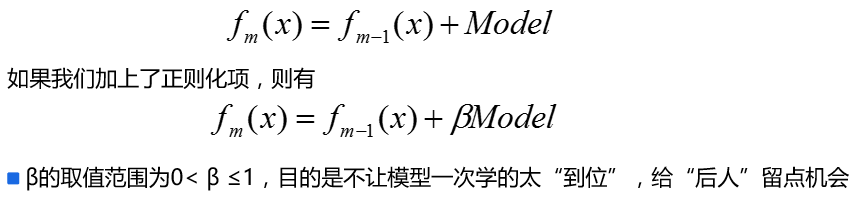


2.



3.如果从前向分布学习视角出发，

的时候，其实就不与带权学习视角下的分类问题是等价的了，这时候我自己玩自己的，比如我想加正则，都是没有关系的，就像这样



（不要一直想着这两个视角等价，其实只是在特定条件下等价）

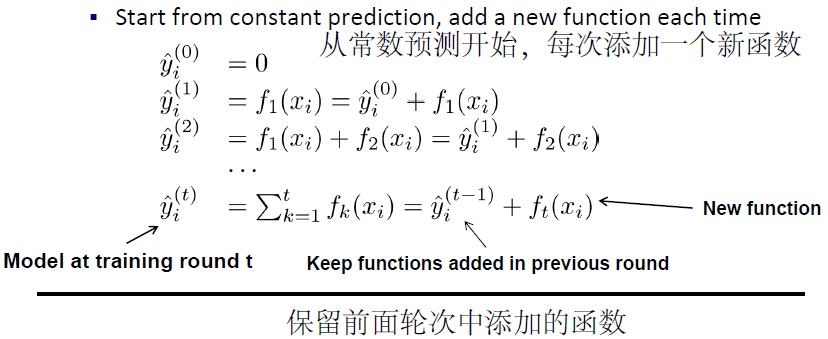
# xgboost小结

我们的目标函数是

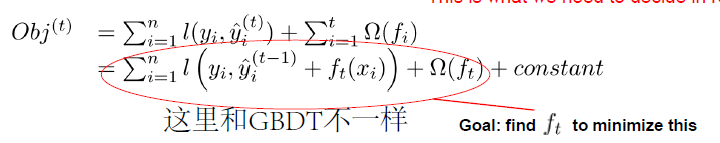


代价函数（每个损失的和）+正则化项（k棵树的复杂度的和）

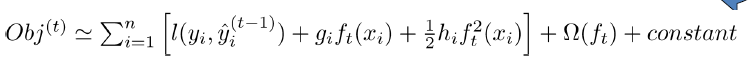
我们想让目标函数最小，但这是一个复杂的优化问题，要使用前向分布学习算法来求解，求解每一棵树不能使用SGD之类的算法，因为我们的模型是树（他不像线性模型），因此要使用加法模型



那么每一步的加法模型的求解就是最小化下图红框里的东西（代价+模型复杂度）

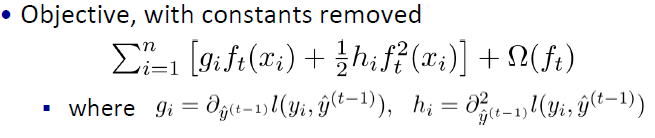


这里作者并没有直接进行最优化，而是先使用泰勒公式对目标函数变形，即：

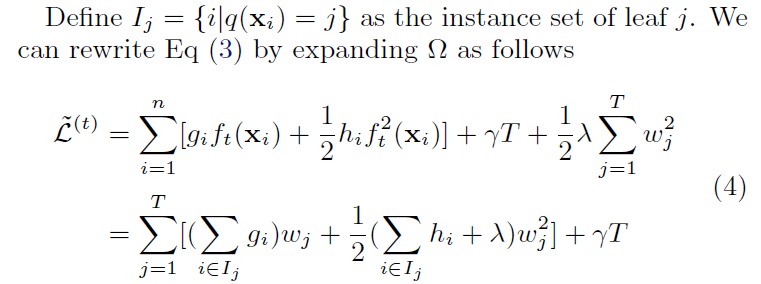


好处是使用二阶近似加快求解，另外也将损失函数抽离出来，以便日后你自定义损失函数，而内部代码无需修改，这是工程上一贯的作风（Second-order approximation can be used to quickly optimize the objective in the **general** setting）

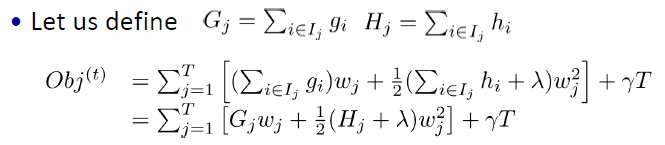
移除目标函数中的常量，变为：



作者将一棵树拆解为权值向量w和树的结构qx，并对新树ft(x)做了替代，同时将目标函数由单个样本的层面转到了叶子节点的层面，即n已经变为T了。



接着，作者这里又做了一些定义【注意：论文中的就是ppt中的】

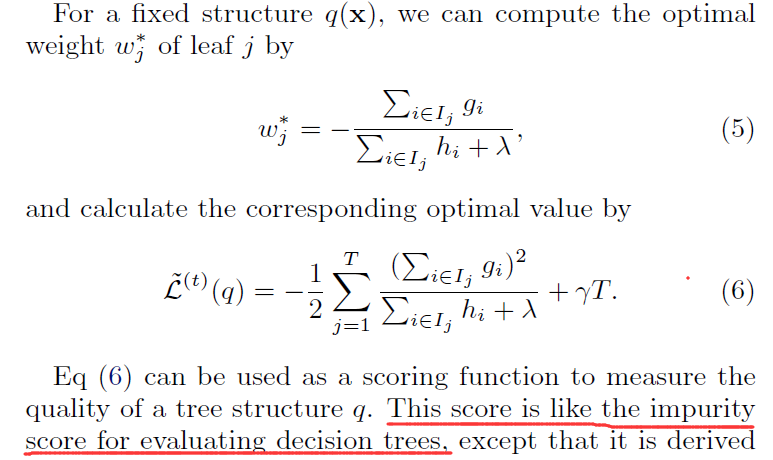


从上图可以看出，只要树的结构被确定，即知道了q(x)【给我一个样本，我可以返回他所在的节点】，那我的和就是常量了，那么Obj中就只剩一个变量w了，根据初中二次函数的知识，就能得到w\*，还能得到Obj的极小值。

那么我们问题的就归结到确定树结构q(x)的问题上了。

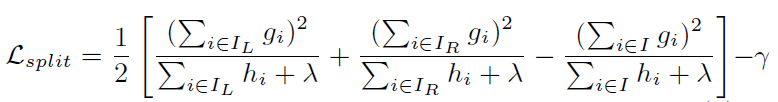
我们的思路是这样的：既然q(x)一经确定就有一个Obj，又希望Obj最小。大家有没有想到决策树构造的过程，根节点不纯度很大，随着节点的划分，都会让不纯度降低。那么在xgboost中，这个Obj就可以看做是不纯度，然后一层层地构建树（即，确定q(x)），从而让Obj变小

其实作者就是这个思路，如下图：



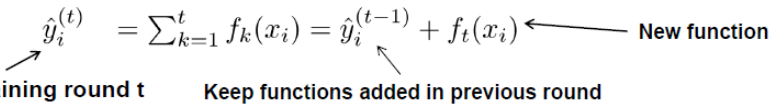
如果这个理解了，下面不就是像决策树一样使用信息增益构建树一样了吗？

贪心策略为：（最后的r表示：每次构建会多一个叶节点，这个是xgboost独有的，把模型复杂度考虑到贪心指标里了）



候选解集合我就不加赘述了

以上过程完成后，也就完成了加法模型中一棵树的构建



加入加法模型，继续加入新的树，重复以上操作，直至达到收敛条件或达到k次循环。

# Python write的问题

import time

start = time.time()

a = open("a",mode='w')

for i in range(1000000):

for j in range(10):

a.write(str(j))

a.close()

end = time.time()

end - start

import time

start = time.time()

a = open("a",mode='w')

s = ""

for i in range(1000000):

for j in range(10):

s += str(j)

a.write(s)

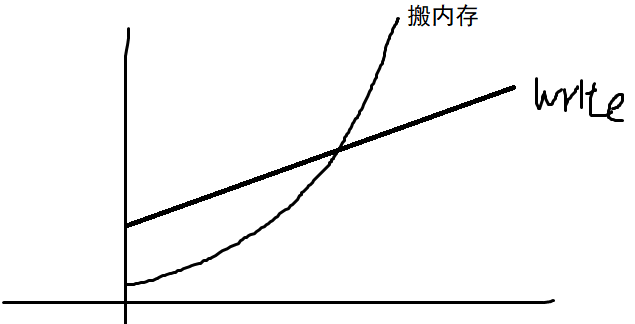
s = ""

a.close()

end = time.time()

end - start

缓冲区较小时使用可以提高效率



# 特征值分解

http://www.mamicode.com/info-detail-2513749.html

import numpy as np

from numpy.linalg import eig

A = np.array([[126, 52, -3, -69],

[ 52, 292, -73, -80],

[ -3, -73, 141, -31],

[-69, -80, -31, 78]])

vals, vecs = eig(A)

Lambda = np.diag(vals)

np.dot(np.dot(vecs, Lambda), vecs.T)

# 奇异值分解

import numpy as np

from numpy.linalg import svd

A = np.array([[126, 52, -3, -69],

[ 52, 292, -73, -80],

[ -3, -73, 141, -31],

[-69, -80, -31, 78],

[-69, -80, -31, 178]])

U, vals, VT = svd(A)

Lambda = np.diag(vals)

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

Lambda = np.r\_[Lambda,h]

np.dot(np.dot(U, Lambda), VT)

# 网格搜索与交叉验证

# 交叉验证经常与网格搜索进行结合，作为参数评价的一种方法，这种方法叫做grid search with cross validation。

# sklearn因此设计了一个这样的类GridSearchCV，这个类实现了fit，predict，score等方法，被当做了一个estimator，

# 使用fit方法，该过程中：（1）搜索到最佳参数；（2）实例化了一个最佳参数的estimator

from sklearn.datasets import load\_iris

from sklearn.svm import SVC

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

from sklearn.model\_selection import GridSearchCV

iris = load\_iris()

# X\_train,X\_test,y\_train,y\_test = train\_test\_split(iris.data,iris.target,random\_state=0)

# print("Size of training set:{} size of testing set:{}".format(X\_train.shape[0],X\_test.shape[0]))

#把要调整的参数以及其候选值 列出来；

param\_grid = {"gamma":[0.001,0.01,0.1,1,10,100],

"C":[0.001,0.01,0.1,1,10,100]}

print("Parameters:{}".format(param\_grid))

grid\_search = GridSearchCV(SVC(),param\_grid,cv=5) #实例化一个GridSearchCV类

X\_train,X\_test,y\_train,y\_test = train\_test\_split(iris.data,iris.target,random\_state=10)

grid\_search.fit(X\_train,y\_train) #训练，找到最优的参数，同时使用最优的参数实例化一个新的SVC estimator。

print("Test set score:{:.2f}".format(grid\_search.score(X\_test,y\_test)))

print("Best parameters:{}".format(grid\_search.best\_params\_))

print("Best score on train set:{:.2f}".format(grid\_search.best\_score\_))# best\_estimator的分数 best\_score\_：提供优化过程期间观察到的最好的评分

# 分类评估指标总结

from sklearn.metrics import roc\_curve,confusion\_matrix,roc\_auc\_score,precision\_score,recall\_score,f1\_score,classification\_report

import os

import pandas as pd

from sklearn.tree import DecisionTreeClassifier,export\_graphviz

from sklearn.model\_selection import GridSearchCV

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

import matplotlib.pylab as plt

data = pd.read\_excel(r'datas/Mass.xlsx',na\_values='?')

data.dropna(inplace = True)

X = data.iloc[:,:-1]

y = data.iloc[:,-1]

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

#------------------------------------------

# 1. 建立决策树分类器，使用网格搜索找到最优参数。（50分）

dt = DecisionTreeClassifier()

paramsDict = {'max\_depth':range(2,100),

'min\_samples\_leaf':[1,3,5]}

gridSearch = GridSearchCV(dt,param\_grid=paramsDict)

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

print(gridSearch.best\_params\_)

#-------------------------------

best\_dt = DecisionTreeClassifier(max\_depth=2,min\_samples\_leaf=1)

# 2. 划分数据集，训练模型，导出树状图，并画出ROC曲线。（50分）

best\_dt.fit(X\_train,y\_train)

y\_pred = best\_dt.predict(X\_test)

print(y\_pred)

y\_score = best\_dt.predict\_proba(X\_test)

print(y\_score)#两列 第一列为属于0的概率 第二列为属于1的概率

fpr, tpr, thresholds = roc\_curve(y\_test,y\_score[:,1])#属于1的概率

plt.plot(fpr,tpr)

plt.show()

print('准确率：',best\_dt.score(X\_test,y\_test))

print('精确率：',precision\_score(y\_test,y\_pred))

print('召回率：',recall\_score(y\_test,y\_pred))

print('f1-score:',f1\_score(y\_test,y\_pred))

print('auc:',roc\_auc\_score(y\_test,y\_score[:,1]))

print('混淆矩阵：\n',confusion\_matrix(y\_test,y\_pred))

print('分类报告：\n',classification\_report(y\_test,y\_pred))

#分类报告<https://blog.csdn.net/akadiao/article/details/78788864>

# macro-average VS micro-average

<https://www.cnblogs.com/yuuken/p/8822496.html>

多标签分类的结果评估---macro-average和micro-average介绍

# 带权抽样本

import random

a = ["A","B","C","D"]

res = []

for T in range(10000):

res.append(random.choices(a,weights=[10,20,30,40])[0])

pd.Series(res).value\_counts()

# 回归模型评估

#回归模型评估调库

import pandas as pd

path = "datas/hour.csv"

df = pd.read\_csv(path, sep=",")

## 删除无用的列

del df["instant"]

del df["dteday"]

del df["casual"]

del df["registered"]

## 检查哪些特征需要做独热编码

# for i in df.columns:

# print(df[i].value\_counts())

#

## 独热编码,对需要进行独热编码的列编码

from sklearn.preprocessing import OneHotEncoder

hotCoder = OneHotEncoder(sparse=False,handle\_unknown="ignore")

hot = hotCoder.fit\_transform(df[["season","mnth","hr","weekday","weathersit"]])

hot = pd.DataFrame(hot)

## 删除掉独热编码的列

del df["season"]

del df["mnth"]

del df["hr"]

del df["weekday"]

del df["weathersit"]

## 多项式扩展

from sklearn.preprocessing import PolynomialFeatures

polyCoder = PolynomialFeatures(degree=3,interaction\_only=False)

poly = polyCoder.fit\_transform(df[["temp","atemp","hum","windspeed"]])

poly = pd.DataFrame(poly,columns=polyCoder.get\_feature\_names())

## 标准化

from sklearn.preprocessing import StandardScaler

scaleCoder = StandardScaler()

std = scaleCoder.fit\_transform(poly)

std = pd.DataFrame(std,columns=polyCoder.get\_feature\_names())

## 删除掉标准化的列

del df["temp"]

del df["atemp"]

del df["hum"]

del df["windspeed"]

## 合并

df = pd.concat([df,hot],axis=1)

df = pd.concat([df,std],axis=1)

## 构建X和Y

Y = df[["cnt"]]

del df["cnt"]

X = df

## 划分训练集和测试集

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

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

X, Y, test\_size=0.33, random\_state=42)

## 训练Ridge

from sklearn.linear\_model import Ridge

model = Ridge(alpha=0.7,fit\_intercept=True,max\_iter=200,solver="auto")

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

model.score(X\_test,y\_test) #R2

from sklearn.metrics import mean\_squared\_error,mean\_absolute\_error

mean\_squared\_error(y\_pred=model.predict(X\_test),y\_true=y\_test)

mean\_absolute\_error(y\_pred=model.predict(X\_test),y\_true=y\_test)

# 网格搜索和交叉验证调库

# 现有文件名为aviation的航空客运信息数据集，共包括5000个样本，每个样本有55个属性，其中runoff\_flag代表是否流失，

# 要求通过这些数据构建客户流失预警模型，而且由于营销资源有限，希望结合客户特征进行有针对性的、高效率的开展客户挽留。

# 具体任务如下：

# 1.读入aviation数据集，设置MEMBER\_NO为索引列。（6分）

# 2.剔除重复值、缺失值。（6分）

# 3.随机抽取500样本，切片特征X和标签Y。（6分）

# 4.使用交叉验证方法（10折）比较逻辑回归、决策树算法性能差异，评估指标用F1分数（6分）

# 5.使用网格搜索对上题中F1分数较高的算法进行超参数调优。（5分）

# 6.使用4、5中确定的最优算法和最优参数建立模型。（5分）

# 7.按照6：4划分数据集。（4分）

# 8.使用训练集数据进行模型训练，对测试集数据进行预测，打印混淆矩阵。（4分）

# 9.打印精确率、召回率、F1分数和AUC值。（3分）

# 10.画出ROC曲线。（3分）

#网格搜索和交叉验证调库

import os

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split,cross\_val\_score,GridSearchCV

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics

import matplotlib.pyplot as plt

# 1.读入aviation数据集，设置MEMBER\_NO为索引列。（5分）

df = pd.read\_excel('datas/aviation.xls',index\_col='MEMBER\_NO')

print(df.info())

# 2.剔除重复值、缺失值。（5分）

df.drop\_duplicates(inplace=True) # 去重

df.dropna(inplace=True) # 删除缺失值

print(df.info())

# 3.随机抽取500样本，切片特征X和标签Y。（5分）

data = df.sample(500) # 抽样

# 从样本数据集data中切片x和y

datax = data.iloc[:,0:-1]

datay = data.iloc[:,-1]

# 4.使用交叉验证方法（10折）比较逻辑回归、决策树算法性能差异，评估指标用F1分数（5分）

clf1 = DecisionTreeClassifier() # 决策树

clf2 = LogisticRegression() # 逻辑回归

# 使用F1分值进行10折交叉验证

model1 = cross\_val\_score(clf1,datax,datay,scoring='f1',cv=10)

model2 = cross\_val\_score(clf2,datax,datay,scoring='f1',cv=10)

print('决策树的F1分值为：')

print(model1)

print('均值：{}，标准差：{}'.format(model1.mean(),model1.std()))

print('逻辑回归的F1分值为：')

print(model2)

print('均值：{}，标准差：{}'.format(model2.mean(),model2.std()))

# 5.使用网格搜索对上题中F1分数较高的算法进行超参数调优。（5分）

# 设置参数

param = {'C':[0.1,1,10], #np.aragr(0.1,1,0.3)等价于[0.1,0.4,0.7]

'max\_iter':[80,100,160]}

# 初始化出一个对象

searchmodel = GridSearchCV(estimator=clf2,

param\_grid=param,

scoring='f1',

cv=10)

# 用网格搜索器进行训练

searchmodel.fit(datax,datay)

# 打印网格搜索后的最优分数

print(searchmodel.best\_score\_)

# 打印网格搜索后的最优参数组合

print(searchmodel.best\_params\_)

# 网格搜索后的打印结果

# 0.970963702095 {'C': 10, 'max\_iter': 100}

# 6.使用4、5中确定的最优算法和最优参数建立模型。（5分）

model = LogisticRegression(C=10,max\_iter=100)

# 7.按照6：4划分数据集。（5分）

# 首先从原数据集df中切片x和y

x = df.iloc[:,0:-1]

y = df.iloc[:,-1]

trainx,testx,trainy,testy = train\_test\_split(x,y,test\_size=0.4,random\_state=12345)

# 8.使用训练集数据进行模型训练，对测试集数据进行预测，打印混淆矩阵。（5分）

model.fit(trainx,trainy) # 对训练集进行训练

# 模型预测

prey = model.predict(testx) # 预测的类标签----0或者1

preproba = model.predict\_proba(testx) # 样本为0的概率和为1的概率

preproba

# 打印混淆矩阵

print(metrics.confusion\_matrix(testy,prey))

# 9.打印精确率、召回率、F1分数和AUC值。（5分）

print('---------精确率---------------')

print(metrics.precision\_score(testy,prey))

print('---------召回率---------------')

print(metrics.recall\_score(testy,prey))

print('---------F1分值---------------')

print(metrics.f1\_score(testy,prey))

print('---------AUC值---------------')

print(metrics.roc\_auc\_score(testy,preproba[:,1]))

# 10.画出ROC曲线。（5分）

fpr,tpr,th = metrics.roc\_curve(testy,preproba[:,1])

plt.xlabel('FPR')

plt.ylabel('TPR')

plt.title('ROC')

plt.plot(fpr,tpr)

plt.show()

# pipeline演示

# pipeline演示

import numpy as np

import pandas as pd

path = "datas/breast-cancer-wisconsin.data"

names = ['id','Clump Thickness','Uniformity of Cell Size','Uniformity of Cell Shape',

'Marginal Adhesion','Single Epithelial Cell Size','Bare Nuclei',

'Bland Chromatin','Normal Nucleoli','Mitoses','Class']

## 加载数据

df = pd.read\_csv(path,sep=",",header=None,names=names)

df.Class = df.Class/2-1

## 删除为空的数据

df = df.replace("?",np.nan)

df = df.dropna(how="any")

## 使用过采样，进行类别平衡

df1 = df[df.Class==1]

df1 = df1.sample(frac=(444-239)/239)

df = pd.concat([df,df1],axis=0)

## 提取X和Y

X = df.iloc[:,0:-1]

y = df.iloc[:,-1]

## 分割数据集

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

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

X, y, test\_size=0.33, random\_state=42)

from sklearn.preprocessing import StandardScaler # 规范化，使各特征的均值为1，方差为0

from sklearn.decomposition import PCA

from sklearn.linear\_model import LogisticRegression

from sklearn.pipeline import Pipeline

pipe = Pipeline([('sc', StandardScaler()),

('pca', PCA(n\_components=10)),

('clf', LogisticRegression()) # 设置随机种子，使测试结果复现

])

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

print(pipe.score(X\_test, y\_test))

from sklearn.externals import joblib

joblib.dump(pipe,"pipe.model")

model2 = joblib.load("pipe.model")

model2.score(X\_test, y\_test)

# pandas打印显式多行多列

<https://blog.csdn.net/saltriver/article/details/78144984>

pd.set\_option('display.max\_colwidth',100)

max\_row，max\_column

# 判断某列的缺失值

a = pd.DataFrame([[1,2,np.nan],[1,2,np.nan],[1,2,3]])

b = a.isnull()

for i in zip(a.columns,b.sum()): #(0, 缺失个数)

print(i)

# KFold

from numpy import array

from sklearn.model\_selection import KFold

# data sample

data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])

# prepare cross validation

kfold = KFold(n\_splits=3, shuffle = True, random\_state= 1)

# enumerate splits

for train, test in kfold.split(data):

print('train: %s, test: %s' % (data[train], data[test]))

from numpy import array

from sklearn.model\_selection import KFold

# data sample

data = pd.DataFrame([[1,5.56],[2,5.7],[3,5.91],[4,6.4],[5,6.8],[6,7.05],[7,8.9],[8,8.7],[9,9],[10,9.05]])

# prepare cross validation

kfold = KFold(n\_splits=3, shuffle = True, random\_state= 1)

# enumerate splits

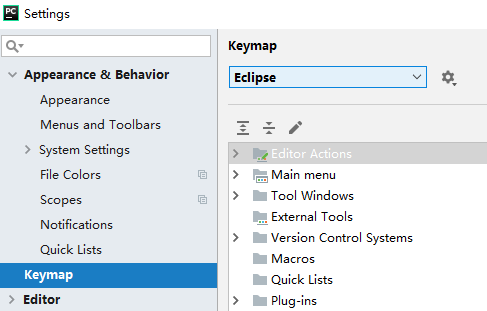
for train, test in kfold.split(data):

print(train)

print(data.iloc[train])

data

# pycharm debug



F5进入函数体

F6在当前代码页执行一行代码

F7退出函数

F8运行到下一个断点

数据规范化

最小-最大规范化、零-均值规范化、小数定标规范化

啤酒和尿布的问题是