随机森林

import pandas as pd  
import numpy as np  
from collections import Counter  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
# 加载数据  
##### 加载训练和测试数据  
#####--------------------------------------------------------------------------------------------------  
#这里读入数据的时候我们没有做任何的处理（像去除空值这些）  
train=pd.read\_csv(r"C:\Users\阿韩想养二哈\Desktop\Titanic数据集\Titanic数据集\train.csv")  
test=pd.read\_csv(r"C:\Users\阿韩想养二哈\Desktop\Titanic数据集\Titanic数据集\test.csv")  
#查看样本数和特征数  
#train\_num,train\_var\_num=np.shape(train)  
#test\_num,test\_var\_num=np.shape(test)  
#print("训练集：有",train\_num,"个样本","每个样本有",train\_var\_num,"个变量.")  
#print("测试集：有",test\_num,"个样本","每个样本有",test\_var\_num,"个变量.")  
print(train.info())  
#print(test.info())  
  
  
#去除离群点  
#####--------------------------------------------------------------------------------------------------  
#离群点检测  
def detect\_outliers(df,n,features):  
 """  
 输入：  
 df：数据框，为需要检测的样本集  
 n：正整数，样本特征超出四分位极差个数的上限，有这么多个特征超出则样本为离群点  
 features:列表，用于检测是否离群的特征  
 输出：  
  
 """  
 outlier\_indices=[]  
 outlier\_list\_col\_index=pd.DataFrame()  
  
 #对每一个变量进行检测  
 for col in features:  
 #计算四分位数相关信息  
 Q1=np.percentile(df[col],25)  
 Q3=np.percentile(df[col],75)  
 IQR=Q3-Q1  
 #计算离群范围  
 outlier\_step=1.5\*IQR  
 #计算四分位数时如果数据上有空值，这些空值也是参与统计的，所以统计出来的Q1、Q3、IQR这些数据有可能是NAN，但是这并不要紧，在判断是否大于或小于的时候跟NAN比较一定是false，因而样本并不会因为空值而被删除掉  
 #空值会在后面特征工程时再做处理  
  
 #找出特征col中显示的离群样本的索引  
 outlier\_list\_col=df[(df[col]<Q1-outlier\_step)|(df[col]>Q3+outlier\_step)].index  
 #额外存储每一个特征在各样本中的离群判断  
 temp=pd.DataFrame((df[col]<Q1-outlier\_step)|(df[col]>Q3+outlier\_step),columns=[col])  
 #将索引添加到一个综合列表中，如果某个样本有多个特征出现离群点，则该样本的索引会多次出现在outlier\_indices里  
 outlier\_indices.extend(outlier\_list\_col)  
 #额外存储每一个特征在各样本中的离群判断，方便查看数据  
 outlier\_list\_col\_index=pd.concat(objs=[outlier\_list\_col\_index,temp],axis=1)  
 #选出有n个以上特征存在离群现象的样本  
 outlier\_indices=Counter(outlier\_indices)  
 multiple\_outliers=list(k for k,v in outlier\_indices.items() if v>n)  
 return multiple\_outliers,outlier\_list\_col\_index  
  
#获取离群点  
outliers\_to\_drop,outlier\_col\_index=detect\_outliers(train,2,["Age","SibSp","Parch","Fare"])  
#这里选取了"Age","SibSp","ParCh","Fare"四个数值型变量；另一个数值型变量舱位等级没选是因为该变量只有1、2、3级不可能有离群点，其他符号型变量诸如性别、登录港口，也只有有限的类型，一般不可能离群，也没有必要分析是否离群。  
#输出离群点信息  
print(train.loc[outliers\_to\_drop])  
print(outlier\_col\_index.loc[outliers\_to\_drop])#查看哪个特征对样本成为离群点有决定作用.  
  
#输出数据集各变量详细信息  
print(train.describe())  
  
#删除离群点  
train = train.drop(outliers\_to\_drop, axis = 0).reset\_index(drop=True)  
#整合训练集和测试集（只是为了后面在有的内容统计和值的处理上更方便，也可以不整合对每个数据集单独处理）  
#整合需要在训练集剔除离群点后再做，因为测试集是不需要剔除离群点  
  
  
#查看缺失值  
print(train.info())  
  
#输出数据集各变量详细信息  
print(train.describe())  
  
#查看数据之间的相关性  
#corr中无参数默认是皮尔逊相关系数，若要改成斯皮尔曼相关系数要在corr中加上method='spearman'  
g=sns.heatmap(train[["Survived","Age","SibSp","Parch","Pclass","Fare"]].corr(),annot=True,fmt = ".2f",cmap = "coolwarm")  
plt.show()  
  
#Fare特征的缺失值进行填充  
train["Fare"]=train["Fare"].fillna(train["Fare"].median())  
#利用柱形图来查看log变换前Fare在整个数据集中的分布  
g=sns.distplot(train["Fare"],color="darkblue",label="Skewness:%.2f"%(train["Fare"].skew()))  
g.legend(loc="best")  
plt.show()  
#下面利用log函数进行数据变换  
train["Fare"]=train["Fare"].map(lambda i:np.log(i) if i>0 else 0)#map()函数具体将元素进行映射的功能  
#查看变换后的数据分布  
g=sns.distplot(train["Fare"],color="darkblue",label="Skewness:%.2f"%(train["Fare"].skew()))  
g.legend(loc="best")  
plt.show()  
  
#Embarked缺失值填充  
#缺失值填充  
train["Embarked"]=train["Embarked"].fillna(train["Embarked"].describe().top)  
print(train["Embarked"].isnull().sum())  
  
#性别Sex的数值化  
train["Sex"]=train["Sex"].map({"male":0,"female":1})  
  
#填充Age缺失值  
#获取Age缺失值索引  
index\_NaN\_age=list(train["Age"][train["Age"].isnull()].index)  
for i in index\_NaN\_age:  
 age\_med=train["Age"].median()#如果通过关联特征找不到匹配的值，则用整个数据的中值填充  
 age\_pred=train["Age"][((train["SibSp"]==train.iloc[i]["SibSp"])&(train["Parch"]==train.iloc[i]["Parch"])&(train["Pclass"]==train.iloc[i]["Pclass"]))].median()  
 if not np.isnan(age\_pred):  
 train["Age"].iloc[i]=age\_pred  
 else:  
 train["Age"].iloc[i]=age\_med  
  
#填充值后再看一次Age在不同Survived下的分布情况  
g = sns.violinplot(data=train, x="Survived", y="Age")  
plt.show()  
  
#对Name进行处理  
#查看Name  
print(train["Name"].head())  
#下面直接提取名字中间部分  
train\_title=[i.split(",")[1].split(".")[0].strip() for i in train["Name"]]  
train["Title"]=pd.Series(train\_title)  
#查看详情  
print(train["Title"].describe())  
print(train["Title"].unique())  
#将title合并为几个组  
train["Title"]=train["Title"].replace(['Mr','Don'],'Mr')  
train["Title"]=train["Title"].replace(['Mrs','Miss','Mme','Ms','Lady','Dona','Mlle'],'Ms')  
train["Title"]=train["Title"].replace(['Sir','Major','Col','Capt'],'Major')  
train["Title"]=train["Title"].replace(['Master','Jonkheer','the Countess'],'Jonkheer')  
train["Title"]=train["Title"].replace(['Rev','Dr'],'Rev')  
#我们查看各组的幸存率情况：  
g=sns.barplot(data=train[:],x="Title",y="Survived")  
g.set\_ylabel("Survival Probability")  
plt.show()  
#下面将姓名数值化  
train["Title"]=train["Title"].map({'Mr':0,'Ms':1,'Major':2,'Jonkheer':3,'Rev':4,'Dr':5})  
train["Title"]=train["Title"].astype(int)  
#将Title哑变量化  
train=pd.get\_dummies(train,columns=["Title"],prefix="TL")  
# 去掉name这一特征  
train.drop(labels = ["Name"], axis = 1, inplace = True)  
print(train.info())  
  
#对Cabin进行处理  
#先查看Cabin的情况  
print(train["Cabin"].describe())  
print(train["Cabin"].isnull().sum())  
#将船舱信息进行替换  
train["Cabin"]=pd.Series([i[0] if not pd.isnull(i) else 'X' for i in train['Cabin']])  
#再来查看一下船舱信息  
print(train["Cabin"].describe())  
print(train["Cabin"].isnull().sum())  
#查看不同船舱的幸存率  
g=sns.barplot(data=train[:],x="Cabin",y="Survived")  
g.set\_ylabel("Survival Probability")  
plt.show()  
#利用哑变量将Cabin信息数值化  
train=pd.get\_dummies(train,columns=["Cabin"],prefix="Cabin")  
#再来查看一下船舱信息  
print(train.info())  
  
#对Ticket进行处理  
Ticket=[]  
for i in list(train["Ticket"]):  
 if not i.isdigit():  
 Ticket.append(i.replace(".","").replace("/","").strip().split(' ')[0])  
 else:  
 Ticket.append("X")  
train["Ticket"]=Ticket  
#查看替换后的情况  
print(train["Ticket"].describe())  
#查看不同船票的生存率  
g=sns.barplot(data=train,x="Ticket",y="Survived")  
g.set\_ylabel("Survival Probability")  
plt.show()  
  
#利用哑变量将Ticket数值化  
train=pd.get\_dummies(train,columns=["Ticket"],prefix="T")  
#将Embarked哑变量化  
train = pd.get\_dummies(train, columns = ["Embarked"], prefix="Em")  
#将Pclass哑变量化  
train["Pclass"] = train["Pclass"].astype("category")  
train = pd.get\_dummies(train, columns = ["Pclass"],prefix="Pc")  
train.drop(labels = ["PassengerId"], axis = 1, inplace = True)  
#查看最终数据  
#print(train.head())  
print(train.info())  
  
#重新获取训练数据和测试数据  
train=train[:]  
train["Survived"]=train["Survived"].astype(int)  
Y\_train=train["Survived"]  
X\_train=train.drop(labels=["Survived"],axis=1)  
test=train[:]  
test.drop(labels=["Survived"],axis=1,inplace=True)  
  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import GridSearchCV  
from sklearn.model\_selection import KFold   
# 搜索随机森林的最佳参数  
RFC = RandomForestClassifier()  
kf = KFold(n\_splits=5)  
## 设置参数网络  
rf\_param\_grid = {"max\_depth": [None],  
 "max\_features": [1, 3, 10],  
 "min\_samples\_split": [2, 3, 10],  
 "min\_samples\_leaf": [1, 3, 10],  
 "bootstrap": [False],  
 "n\_estimators" :[100,300],  
 "criterion": ["gini"]}  
gsRFC = GridSearchCV(RFC,param\_grid = rf\_param\_grid, cv=kf, scoring="accuracy", n\_jobs= 1, verbose = 1)  
gsRFC.fit(X\_train,Y\_train)  
RFC\_best = gsRFC.best\_estimator\_  
print(RFC\_best)  
# 打印最佳得分  
print(gsRFC.best\_score\_)  
  
from sklearn.model\_selection import learning\_curve  
# 效果评估  
#####--------------------------------------------------------------------------------------------------  
### 效果评估之学习曲线  
def plot\_learning\_curve(estimator, title, X, y, ylim=None, cv=None,  
 n\_jobs=1, train\_sizes=np.linspace(.1, 1.0, 5)):  
 """Generate a simple plot of the test and training learning curve"""  
 plt.figure()  
 plt.title(title)  
 if ylim is not None:  
 plt.ylim(\*ylim)  
 plt.xlabel("Training examples")  
 plt.ylabel("Score")  
 train\_sizes, train\_scores, test\_scores = learning\_curve(  
 estimator, X, y, cv=cv, n\_jobs=n\_jobs, train\_sizes=train\_sizes)  
 train\_scores\_mean = np.mean(train\_scores, axis=1)  
 train\_scores\_std = np.std(train\_scores, axis=1)  
 test\_scores\_mean = np.mean(test\_scores, axis=1)  
 test\_scores\_std = np.std(test\_scores, axis=1)  
 plt.grid()  
  
 plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,  
 train\_scores\_mean + train\_scores\_std, alpha=0.1,  
 color="r")  
 plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,  
 test\_scores\_mean + test\_scores\_std, alpha=0.1, color="g")  
 plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r",  
 label="Training score")  
 plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g",  
 label="Cross-validation score")  
  
 plt.legend(loc="best")  
 return plt  
g = plot\_learning\_curve(gsRFC.best\_estimator\_,"RF mearning curves",X\_train,Y\_train,cv=kf)  
plt.show()  
  
# 特征变量权重分析  
#####--------------------------------------------------------------------------------------------------  
nrows = ncols = 1  
fig, axes = plt.subplots(nrows = nrows, ncols = ncols, sharex="all", figsize=(15,15))  
#names\_classifiers = [("AdaBoosting", ada\_best),("ExtraTrees",ExtC\_best),("RandomForest",RFC\_best),("GradientBoosting",GBC\_best)]  
names\_classifiers = [("RandomForest",RFC\_best)]  
nclassifier = 0  
for row in range(nrows):  
 for col in range(ncols):  
 name = names\_classifiers[nclassifier][0]  
 classifier = names\_classifiers[nclassifier][1]  
 indices = np.argsort(classifier.feature\_importances\_)[::-1][:40]  
 g = sns.barplot(y=X\_train.columns[indices][:40],x = classifier.feature\_importances\_[indices][:40] , orient='h')  
 g.set\_xlabel("Relative importance",fontsize=12)  
 g.set\_ylabel("Features",fontsize=12)  
 g.tick\_params(labelsize=9)  
 g.set\_title(name + " feature importance")  
 nclassifier += 1  
plt.show()