'''

加载调用数据库

'''

import pandas as pd

import numpy as np

import re

import sklearn

import xgboost as xgb

import seaborn as sns

import matplotlib.pyplot as plt

import plotly.offline as py

py.init\_notebook\_mode(connected=True)

import plotly.graph\_objs as go

import plotly.tools as tls

import warnings

warnings.filterwarnings('ignore')

'''

#得到5个基本模型作为stacking进行预测

'''

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier

from sklearn.svm import SVC

from sklearn.cross\_validation import KFold

train = pd.read\_csv("D:/kaggle/Titanic/data/train.csv")

test = pd.read\_csv("D:/kaggle/Titanic/data/test.csv")

full\_data = [train, test]

#（测试数据集里没有Survived类标记项，存在gender\_submission里）

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

train.info()

test.info()

print(train.isnull().sum())

print(test.isnull().sum())

'''

存储乘客的ID号

'''

PassengerId = test['PassengerId']

print(train.head(3)) #查看前三行数据

'''

(1)加入新特征"Name\_length"，并给出名字的长度

'''

train['Name\_length'] = train['Name'].apply(len) #方法apply的返回值就是()内函数对.apply前数据的返回值

test['Name\_length'] = test['Name'].apply(len)

'''

（2）加入新特征是否存在Cabin

'''

train['Has\_Cabin'] = train["Cabin"].apply(lambda x: 0 if type(x) == float else 1) #lambda为匿名函数，":"前为输入变量，":"后为返回值

test['Has\_Cabin'] = test["Cabin"].apply(lambda x: 0 if type(x) == float else 1) #NaN在python3里是float类型

'''

（3）创造新的家庭成员特征FamilySize作为SibSp和Parch的组合

'''

for dataset in full\_data:

dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

'''

（4）根据新的特征FamilySize,创造新特征IsAlone表示是否一个人

'''

for dataset in full\_data:

dataset['IsAlone'] = 0

dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1 # 方法data.loc[y, 'x'] = a：表示把data数据中，y条件下的'x'的值赋予新值a

'''

（5）消除登船地点缺失的数据,并用频率高的'S'代替

'''

for dataset in full\_data:

dataset['Embarked'] = dataset['Embarked'].fillna('S') # 或者dataset.loc[dataset['Embarked'].isnull(), 'Embarked'] = 'S'

#fillna()会填充NaN数据，返回()内的填充后结果。如果希望在原DataFrame中修改，则在()内添加inplace=True

'''

（6）去除费用特征缺失数据,并以他们中位数代替,创建新的特征CategoricalFare费用类别

'''

for dataset in full\_data:

dataset['Fare'] = dataset['Fare'].fillna(train['Fare'].median()) #.median()代表.之前数据的中位数

train['CategoricalFare'] = pd.qcut(train['Fare'], 4) #.qcut()方法，第一个参数为要分的数据，第二个数为被分成的份数

#cut均分值的分布间隔来选择桶的均匀间隔，qcut是均分值的频数来选择桶的均匀间隔。

print(train['CategoricalFare']) #找划分点

'''

（6）补齐年龄缺失值，创建新的年龄分类特征CategoricalAge

'''

'''

for dataset in full\_data:

age\_avg = dataset['Age'].mean()

age\_std = dataset['Age'].std()

age\_null\_count = dataset['Age'].isnull().sum()

age\_null\_random\_list = np.random.randint(age\_avg - age\_std, age\_avg + age\_std, size=age\_null\_count)

dataset['Age'][np.isnan(dataset['Age'])] = age\_null\_random\_list

dataset['Age'] = dataset['Age'].astype(int)

'''

from sklearn.ensemble import RandomForestRegressor # 使用 RandomForestClassifier 填补缺失的年龄属性

def set\_missing\_ages(df):

age\_df = df[['Age','Fare', 'Parch', 'SibSp', 'Pclass']] # 把已有的数值型特征取出来丢进Random Forest Regressor中

known\_age = age\_df[age\_df.Age.notnull()].as\_matrix() # 乘客分成已知年龄和未知年龄两部分

unknown\_age = age\_df[age\_df.Age.isnull()].as\_matrix()

y = known\_age[:, 0] # y即目标年龄，Age在第1列

X = known\_age[:, 1:] # X即特征属性值

'''

n\_estimators：基学习器决策树的个数，越多越好，但是性能就会越差，至少100左右可以达到可接受的性能和误差率。

n\_jobs：并行job个数。这个参数告诉引擎有多少处理器是它可以使用。 “-1”意味着没有限制，而“1”值意味着它只能使用一个处理器。

'''

rfr = RandomForestRegressor(random\_state=0, n\_estimators=2000, n\_jobs=-1) # 将X和y用RFR算法fit后存入至rfr

rfr.fit(X, y)

predictedAges = rfr.predict(unknown\_age[:, 1:]) # 用得到的模型rfr进行.predict()方法预测未知年龄结果，predict方法括号内输入的是其余特征属性X'

df.loc[ df.Age.isnull(), 'Age' ] = predictedAges # 用得到的预测结果填补原缺失数据

return df

train = set\_missing\_ages(train)

test = set\_missing\_ages(test)

for dataset in full\_data:

dataset['Age'] = dataset['Age'].astype(int) #把所有Age数据int化，否则是float格式

train['CategoricalAge'] = pd.cut(train['Age'], 5) #cut均分值的分布间隔来选择桶的均匀间隔，均分为5段

print(train['CategoricalAge']) #找划分点

'''

（7）定义消除乘客名字中的特殊字符，并创建新的名字特征Title,包含乘客名字主要信息

'''

def get\_title(name):

title\_search = re.search(' ([A-Za-z]+)', name) #re.search 扫描整个字符串并返回第一个成功的匹配的位置。第一个参数为要匹配的目标，第二个参数为搜索匹配的区域

# If the title exists, extract and return it.

if title\_search:

return title\_search.group(1)

return ""

'''

\*re.search()和group()使用方式：

import re

（1）

m2 = re.search('c', 'abcdefcg') #只匹配第一个c，返回<\_sre.SRE\_Match object; span=(2, 3), match='c'>

（2）

a = re.search('([a-z])','231422sadf')

print(a.group(0)) #返回2

（3）

a = re.search('([a-z]+)','231422sadf')

print(a.group(0)) #返回231422

（4）

a = "123abc456"

print( re.search("([0-9]\*)([a-z]\*)([0-9]\*)",a).group(0) ) #123abc456,返回整体

print( re.search("([0-9]\*)([a-z]\*)([0-9]\*)",a).group(1) ) #123，列出第一个括号匹配内容

print( re.search("([0-9]\*)([a-z]\*)([0-9]\*)",a).group(2) ) #abc，列出第二个括号匹配内容

print( re.search("([0-9]\*)([a-z]\*)([0-9]\*)",a).group(3) ) #456，列出第三个括号匹配内容。若字符串a中没有456，则group(3)就不会返回内容

'''

for dataset in full\_data:

dataset['Title'] = dataset['Name'].apply(get\_title)

'''

将所有非常见的Title分组成一个单独的“稀有”组，并将Title中的非常规写法常规化

'''

for dataset in full\_data:

dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')

dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')

dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')

dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

'''

（8）对特征进行绘制，把所有特征都转化成数值型

map(function, x) 会根据提供的函数对指定序列做映射。

第二个参数序列中的每一个元素调用第一个参数 function函数，返回包含每次 function 函数返回值的新列表。

def square(x) :

return x \*\* 2

print( list(map(square, [1,2,3,4,5])) )

返回：[1, 4, 9, 16, 25]

'''

for dataset in full\_data:

'''

对性别进行绘制

'''

dataset['Sex'] = dataset['Sex'].map( {'female': 0, 'male': 1} ).astype(int) #字典描述映射，类似于函数

'''

对Title进行绘制

'''

title\_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}

dataset['Title'] = dataset['Title'].map(title\_mapping)

dataset['Title'] = dataset['Title'].fillna(0)

'''

对登船地点绘制

'''

dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)

'''

对费用进行绘制

'''

dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0

dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1 #不能用and

dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare'] = 2

dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3

dataset['Fare'] = dataset['Fare'].astype(int)

'''

对年龄进行绘制

'''

dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0

dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1

dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2

dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3

dataset.loc[ dataset['Age'] > 64, 'Age'] = 4

'''

（9）除去冗余的特征属性标签

'''

drop\_elements = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'SibSp']

train = train.drop(drop\_elements, axis = 1) #drop()方法删除，axis=1表示按列删除

train = train.drop(['CategoricalAge', 'CategoricalFare'], axis = 1) #axis=0代表往跨行（down)，而axis=1代表跨列（across)，作为方法动作

test = test.drop(drop\_elements, axis = 1)

'''

axis=0:

df.mean(axis=0)：按列计算均值

df.drop(name, axis=0)：按行删除

（列均行删）

axis=1:

df.mean(axis=1)：按行计算均值

df.drop(name, axis=1)：按列删除

（列删行均）

'''

train.info()

test.info()

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

'''

（1）观察进行过特征清洗,筛选过的新特征数据

'''

print(train.head(3))

'''

（2）Pearson Correlation Heatmap(皮尔森相关热图)：

生成特征的相关图，看看一个特征和另一个特征的相关程度，.corr()方法。

利用Seaborn绘图软件包的.heatmap()方法，非常方便地绘制皮尔森相关热图

'''

colormap = plt.cm.viridis

plt.figure(figsize=(14,12))

plt.title('Pearson Correlation of Features', y=1.05, size=15)

sns.heatmap(train.astype(float).corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linecolor='white', annot=True)

#.heatmap()为画热图方法，.astype(float)把所有数据转换成float类型，.corr()即为求相关性的方法

# linewidths，linecolor划分线的宽度和颜色，annot是否在方格里注释数据，vmin, vmax图例相关度最大值和最小值

plt.show()

'''

（3）生成一些配对图来观察一个特征和另一个特征的数据分布

'''

g = sns.pairplot(train[[u'Survived', u'Pclass', u'Sex', u'Age', u'Parch', u'Fare', u'Embarked', #一行写不下时，直接换行并缩进两个tab（或者使用\）

u'FamilySize', u'Title']], hue='Survived', palette = 'seismic',size=1.2,

diag\_kind = 'kde',diag\_kws=dict(shade=True),plot\_kws=dict(s=10) )

g.set(xticklabels=[])

'''

sns.pairplot()方法：

kind : {‘scatter’, ‘reg’} scatter散点图，reg回归图

diag\_kind : {‘hist’, ‘kde’} hist柱状图，kde密度图

hue : 使用指定变量为分类变量画图。参数类型：string (变量名)。即用一个特征来显示图像上的颜色，类似于打标签。

palette : 调色板颜色

size : 默认 6，图的尺度大小（正方形）参数类型：numeric

markers : 使用不同的形状。参数类型：list

{plot, diag, grid}\_kws : 指定其他参数。参数类型：dicts

'''

plt.show()

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

'''

创建集合和堆叠模型

一、构建一个SklearnHelper类，允许扩展所有Sklearn分类器的共同的内置方法（如训练，预测和拟合）。

二、定义一个get\_oof函数。

此后，如果要调用五个不同的分类器，将减少冗余，不需要多次编写相同的代码。

'''

'''

（1）准备参数

'''

ntrain = train.shape[0] #shape函数查看矩阵维数，直接print(.shape)即可得到行列尺寸。shape[0]为行数，shape[1]为列数。

ntest = test.shape[0]

SEED = 0 # 随机数生成种子，为了可重复性

NFOLDS = 5 # 设置k折交叉验证的k值

kf = KFold(ntrain, n\_folds= NFOLDS, random\_state=SEED) #在训练样本train中使用交叉验证，最终会采用所有NFOLDS行的均值

'''

KFold()方法：

KFold(n, n\_folds=3, shuffle=False, random\_state=None)

第一个参数为数据总量尺寸，第二个参数为k，shuffle表示在划分前要不要打乱数据，

random\_state：随机数生成种子。若设置，保证每次随机生成的数是一致的（即使是随机的）；若不设置，即默认None，则每次生成的随机数都不同。

注意：其实交叉验证的是 0到ntrain-1 的标记，通过标记标定出交叉验证出来的训练子集和测试子集。

例：

from sklearn.cross\_validation import KFold

kf = KFold(12,n\_folds=5,shuffle=False)

for i,(train\_index,test\_index) in enumerate(kf):

print(i,train\_index,test\_index)

输出为：

0 [ 3 4 5 6 7 8 9 10 11] [0 1 2]

1 [ 0 1 2 6 7 8 9 10 11] [3 4 5]

2 [ 0 1 2 3 4 5 8 9 10 11] [6 7]

3 [ 0 1 2 3 4 5 6 7 10 11] [8 9]

4 [0 1 2 3 4 5 6 7 8 9] [10 11]

'''

'''

（2）定义一个辅助类来扩展Sklearn分类器

类的简单方法，它简单地调用sklearn分类器中已经存在的相应方法。

本质上创建了一个包装类来扩展各种Sklearn分类器，这样可以帮助我们在实现到堆栈器时，减少编写相同的代码。

'''

class SklearnHelper(object):

def \_\_init\_\_(self, clf, seed=0, params=None): #clf：输入的sklearn分类器， seed：随机种子， params：分类器的参数

params['random\_state'] = seed

self.clf = clf(\*\*params)

def train(self, x\_train, y\_train): #fit()是调用的通用方法。fit(X)表示用数据X来训练某种模型。函数返回值一般为调用fit方法的对象本身。

self.clf.fit(x\_train, y\_train) #fit(X,y=None)为无监督学习算法，fit(X,y)为监督学习算法

def predict(self, x):

return self.clf.predict(x)

def fit(self,x,y):

return self.clf.fit(x,y)

def feature\_importances(self,x,y):

return self.clf.fit(x,y).feature\_importances\_ #RF包里的写法.feature\_importances\_

'''

（3）定义get\_oof函数

Out-of-Fold Predictions预测函数，用来生成第一级预测结果。

之后堆叠使用基础分类器的预测作为训练到二级模型的输入。

然而，不能简单地对完整的训练数据进行基本模型的训练，在完整的测试集上产生预测，然后输出这些用于二级训练。

因为基本模型预测很有可能已经具有“看到”测试集的风险，并因此在馈送这些预测时过度拟合。

'''

def get\_oof(clf\_obj, x\_train, y\_train, x\_test): # clf\_obj：各个clf分类器根据上一步SklearnHelper()类定义出来的对象

#先申请3个空数组

oof\_train = np.zeros((ntrain,)) #zeros创建零矩阵。()内参数代表尺寸。逗号前代表行数，逗号后代表列数，如果没有列数，则创建的是一维数组

oof\_test = np.zeros((ntest,)) #zeros默认元素为float型，zeros((5,3),int) 则变成整型。注意zeros后两个括号：尺寸的括号可以()也可以[]

oof\_test\_skf = np.empty((NFOLDS, ntest)) #empty创建的数组，元素表示无意义的数值

'''注意：其实交叉验证的是 0到ntrain-1 的标记，通过标记标定出交叉验证出来的训练子集和测试子集'''

for i, (train\_index, test\_index) in enumerate(kf):

x\_tr = x\_train[train\_index] #把交叉验证中第i组获得的训练集在原训练数据集train的输入集x\_train中找出来，定义为x\_tr

y\_tr = y\_train[train\_index] #把交叉验证中第i组获得的训练集在原训练数据集train的输出集y\_train中找出来，定义为y\_tr

x\_te = x\_train[test\_index] #把交叉验证中第i组获得的测试集在原训练数据集train的输入集x\_train中找出来，定义为x\_te

clf\_obj.train(x\_tr, y\_tr) #其实就是 fit(x\_tr, y\_tr)

'''

注意：predict()括号内的是要输入的x，预测出来的结果即为y。

此处y被存进数组oof\_train的对应位置[test\_index]上。

'''

oof\_train[test\_index] = clf\_obj.predict(x\_te) #最终循环结束oof\_train矩阵就是交叉验证中通过各组训练子集fit出来的的模型，分别针对各组测试子集预测出来并连接起来的所有结果。即为对全部训练数据集的预测。

oof\_test\_skf[i, :] = clf\_obj.predict(x\_test) #分别保存 用每一组交叉验证fit出来的模型去预测真正的测试数据集而得到的结果

oof\_test[:] = oof\_test\_skf.mean(axis=0) #axis=0为按列计算均值

return oof\_train.reshape(-1, 1), oof\_test.reshape(-1, 1)

'''

reshape方法:

把矩阵内所有元素全部按新的尺寸依次排布，两个参数分别代表行列。

若其中一个参数是-1，则代表其值依据另一个参数确定而确定。

比如总共有10个元素，一个参数为-1，另一个为5，则为-1的那个参数实际表现值为2

若只有一个参数，且为-1，则数组为一行

reshape方法是暂时性的排布

'''

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

'''

sklear模块中的5个分类模型（作为一级分类）

1. Random Forest classifier

2. Extra Trees classifier 分类树

3. AdaBoost classifer

4. Gradient Boosting classifer 梯度提升

5. Support Vector Machine

'''

'''

（1）列出基学习器的参数Parameters

n\_jobs : 用于训练过程的核心数量。 如果设置为-1，则使用所有内核。

n\_estimators : 学习模型中的分类树数（默认设置为10，default=10）。

max\_depth : 树的最大深度，或者应该扩展多少节点。 如果设置得太高，请注意，如果树太深，则会有过度拟合的风险。

verbose : 控制是否要在学习过程中输出任何文本。 值0将禁止所有文本，而值3在每次迭代时输出树学习过程。

min\_samples\_leaf : 或min\_samples\_split，叶子结点最少的样本数，控制叶节点上的样本数量。

max\_features : 分割节点时要考虑的特征的随机子集的大小。

针对RF： 默认是"None"，即划分时考虑所有的特征数；

"log2" 划分时最多考虑log2N个特征；

"sqrt"或者"auto" 划分时最多考虑√N个特征。

整数，代表考虑的特征绝对数。

浮点数，代表考虑特征百分比，即考虑（百分比xN）取整后的特征数。

其中N为样本总特征数。

learning\_rate : 针对Adaboost，即每个弱学习器的权重缩减系数ν：v\*ak。

warm\_start=False ： 热启动，决定是否使用上次调用该类的结果然后增加新的。

所有参数以字典形式集合写出来，注意字典形式参量名要加''

这些参数可去看sklearn文档！

'''

# Random Forest parameters

rf\_params = {

'n\_jobs': -1,

'n\_estimators': 500,

'warm\_start': True,

#'max\_features': 0.2,

'max\_depth': 6,

'min\_samples\_leaf': 2,

'max\_features' : 'sqrt',

'verbose': 0

}

# Extra Trees Parameters

et\_params = {

'n\_jobs': -1,

'n\_estimators':500,

#'max\_features': 0.5,

'max\_depth': 8,

'min\_samples\_leaf': 2,

'verbose': 0

}

# AdaBoost parameters

ada\_params = {

'n\_estimators': 500,

'learning\_rate' : 0.75

}

# Gradient Boosting parameters

gb\_params = {

'n\_estimators': 500,

#'max\_features': 0.2,

'max\_depth': 5,

'min\_samples\_leaf': 2,

'verbose': 0

}

# Support Vector Classifier parameters

svc\_params = {

'kernel' : 'linear',

'C' : 0.025 #C 即1/lambda，描述正则项系数。

} #还有个参数gamma，和sigama负相关，描述核函数宽度，即回归问题中欠拟合和过拟合平衡性

'''

（2）创建5个SklearnHelper辅助类的对象来表示5个模型

'''

#此处D文件和C文件分开时要 imort C文件里内容（C.）

et = SklearnHelper(clf=ExtraTreesClassifier, seed=SEED, params=et\_params)

rf = SklearnHelper(clf=RandomForestClassifier, seed=SEED, params=rf\_params)

ada = SklearnHelper(clf=AdaBoostClassifier, seed=SEED, params=ada\_params)

gb = SklearnHelper(clf=GradientBoostingClassifier, seed=SEED, params=gb\_params)

svc = SklearnHelper(clf=SVC, seed=SEED, params=svc\_params)

'''

（3）将训练集、测试集和目标集转化为Numpy数组以输入模型

'''

y\_train = train['Survived'].ravel() #ravel函数（临时性操作）：对数组降维（拍平成一行），默认是行序优先（一行行走）

train = train.drop(['Survived'], axis=1)

x\_train = train.values # 创建训练数据集数组

x\_test = test.values # 创建测试数据集数组

'''

（4）Output of the First level Predictions

将训练集和测试集送入模型，然后采用交叉验证方式进行预测，这些预测结果将作为二级模型的新特征。

每个模型的两个预测结果分别为：

针对训练数据集的预测结果 ， 针对测试数据集的预测结果

'''

rf\_oof\_train, rf\_oof\_test = get\_oof(rf,x\_train, y\_train, x\_test) # Random Forest

et\_oof\_train, et\_oof\_test = get\_oof(et, x\_train, y\_train, x\_test) # Extra Trees

ada\_oof\_train, ada\_oof\_test = get\_oof(ada, x\_train, y\_train, x\_test) # AdaBoost

gb\_oof\_train, gb\_oof\_test = get\_oof(gb,x\_train, y\_train, x\_test) # Gradient Boost

svc\_oof\_train, svc\_oof\_test = get\_oof(svc,x\_train, y\_train, x\_test) # Support Vector Classifier

print("Training is complete")

#到此刻训练完成

'''

（5）Feature importances generated from the different classifiers

利用Sklearn模型.featureimportances()功能，得出训练和测试集中各种特征的重要性。

'''

rf\_features = rf.feature\_importances(x\_train,y\_train)

et\_features = et.feature\_importances(x\_train, y\_train)

ada\_features = ada.feature\_importances(x\_train, y\_train)

gb\_features = gb.feature\_importances(x\_train,y\_train)

#注：SVC没有feature\_importances\_

rf\_features = list(rf\_features)

et\_features = list(et\_features)

ada\_features = list(ada\_features)

gb\_features = list(gb\_features)

print(rf\_features)

print(et\_features)

print(ada\_features)

print(gb\_features)

'''

（6）创建DataFrame数据框，为使用Plotly包绘制图像作准备：

DataFrame是pandas模块里的方法，表示建立数据框，为画图做准备

数据框以字典形式

'''

cols = train.columns.values

feature\_dataframe = pd.DataFrame( {u'features': cols,

u'Random Forest feature importances': rf\_features,

u'Extra Trees feature importances': et\_features,

u'AdaBoost feature importances': ada\_features,

u'Gradient Boost feature importances': gb\_features

})

'''

（7）Interactive feature importances via Plotly scatterplots

使用交互式Plotly软件包，通过调用Scatter，生成散点图来显示不同分类器的特征重要性

'''

# Scatter plot 1

trace1 = go.Scatter(

y = feature\_dataframe['Random Forest feature importances'].values,

x = feature\_dataframe['features'].values,

mode = 'markers',

marker = dict(

sizemode = 'diameter', #控制图像整体尺寸问题

sizeref = 1,

size = 25, #单个散点的尺寸

# size = feature\_dataframe['Random Forest feature importances'].values,

# color = np.random.randn(500), #set color equal to a variable

color = feature\_dataframe['Random Forest feature importances'].values, # 设置颜色随y轴上值的大小而变化

colorscale = 'Portland',

# line = dict( #每个散点加上黑线圈

# width = 2,

# color = 'rgb(0, 0, 0)'

# ),

showscale = True # 显示图例

),

text = feature\_dataframe['features'].values

)

data1 = [trace1]

layout1 = go.Layout(

autosize = True,

title = 'Random Forest Feature Importance',

hovermode = 'closest',

# xaxis= dict(

# title = 'Pop',

# ticklen = 5,

# zeroline = False,

# gridwidth = 2,

# ),

yaxis = dict(

title = 'Feature Importance',

ticklen = 5, #y轴上刻度线的长度

gridwidth = 2 #平行于x轴的每条网格线的宽度

),

showlegend = False

)

fig1 = go.Figure(data = data1, layout = layout1)

py.plot(fig1, filename = 'scatter1')

# Scatter plot 2

trace2 = go.Scatter(

y = feature\_dataframe['Extra Trees feature importances'].values,

x = feature\_dataframe['features'].values,

mode = 'markers',

marker = dict(

sizemode = 'diameter',

sizeref = 1,

size = 25,

color = feature\_dataframe['Extra Trees feature importances'].values,

colorscale = 'Portland',

showscale = True

),

text = feature\_dataframe['features'].values

)

data2 = [trace2]

layout2 = go.Layout(

autosize = True,

title = 'Extra Trees Feature Importance',

hovermode = 'closest',

# xaxis= dict(

# title= 'Pop',

# ticklen= 5,

# zeroline= False,

# gridwidth= 2,

# ),

yaxis=dict(

title = 'Feature Importance',

ticklen = 5,

gridwidth = 2

),

showlegend = False

)

fig2 = go.Figure(data = data2, layout = layout2)

py.plot(fig2, filename = 'scatter2')

# Scatter plot 3

trace3 = go.Scatter(

y = feature\_dataframe['AdaBoost feature importances'].values,

x = feature\_dataframe['features'].values,

mode = 'markers',

marker = dict(

sizemode = 'diameter',

sizeref = 1,

size = 25,

color = feature\_dataframe['AdaBoost feature importances'].values,

colorscale = 'Portland',

showscale = True

),

text = feature\_dataframe['features'].values

)

data3 = [trace3]

layout3 = go.Layout(

autosize = True,

title= 'AdaBoost Feature Importance',

hovermode= 'closest',

# xaxis = dict(

# title = 'Pop',

# ticklen = 5,

# zeroline = False,

# gridwidth = 2,

# ),

yaxis = dict(

title = 'Feature Importance',

ticklen = 5,

gridwidth = 2

),

showlegend = False

)

fig3 = go.Figure(data = data3, layout = layout3)

py.plot(fig3, filename = 'scatter3')

# Scatter plot 4

trace4 = go.Scatter(

y = feature\_dataframe['Gradient Boost feature importances'].values,

x = feature\_dataframe['features'].values,

mode = 'markers',

marker=dict(

sizemode = 'diameter',

sizeref = 1,

size = 25,

color = feature\_dataframe['Gradient Boost feature importances'].values,

colorscale = 'Portland',

showscale = True

),

text = feature\_dataframe['features'].values

)

data4 = [trace4]

layout4 = go.Layout(

autosize = True,

title = 'Gradient Boosting Feature Importance',

hovermode = 'closest',

# xaxis= dict(

# title= 'Pop',

# ticklen= 5,

# zeroline= False,

# gridwidth= 2,

# ),

yaxis=dict(

title = 'Feature Importance',

ticklen = 5,

gridwidth = 2

),

showlegend= False

)

fig4 = go.Figure(data=data4, layout=layout4)

py.plot(fig4, filename='scatter4')

'''

（8）计算所有特征重要性的平均值，并将其作为特征重要性数据框中的新列存储

'''

feature\_dataframe['mean'] = feature\_dataframe.mean(axis= 1) #在字典里新添关键词'mean'的项，axis = 1 按行求均值

print(feature\_dataframe.head(3))

'''

（9）绘制平均特征重要性的柱状图

'''

# Bar plot

trace5 = go.Bar(

y = feature\_dataframe['mean'].values,

x = feature\_dataframe['features'].values,

width = 0.5, #柱状宽度

marker = dict(

color = feature\_dataframe['mean'].values,

colorscale = 'Portland',

showscale = True,

reversescale = False #反转图例的表示强度的颜色

),

opacity = 0.6 #不透明度

)

data5 = [trace5]

layout5 = go.Layout(

autosize = True,

title = 'Barplots of Mean Feature Importance',

hovermode = 'closest',

# xaxis= dict(

# title = 'Pop',

# ticklen = 5,

# zeroline = False,

# gridwidth = 2,

# ),

yaxis = dict(

title = 'Mean Feature Importance',

ticklen = 5,

gridwidth = 2

),

showlegend = False

)

fig5 = go.Figure(data = data5, layout = layout5)

py.plot(fig5, filename ='Bar')

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

'''

（1）将一级预测构建出的一组新特征（即针对训练集、测试集的预测结果y），作为训练数据训练二级分类器

'''

base\_predictions\_train = pd.DataFrame( {'RandomForest': rf\_oof\_train.ravel(),

'ExtraTrees': et\_oof\_train.ravel(),

'AdaBoost': ada\_oof\_train.ravel(),

'GradientBoost': gb\_oof\_train.ravel()

})

print(base\_predictions\_train.head())

'''

（2）二级训练集相关热图 — Plotly图

模型之间要 “好而不同”，彼此间相关性越低的模型越好。

'''

data6 = [

go.Heatmap(

z = base\_predictions\_train.astype(float).corr().values , # z即强度颜色

x = base\_predictions\_train.columns.values,

y = base\_predictions\_train.columns.values,

colorscale = 'Viridis',

showscale = True,

reversescale = True

)

]

layout6 = go.Layout(

autosize = True,

title = 'Labelled Heatmap',

)

fig6 = go.Figure(data = data6, layout = layout6)

py.plot(fig6, filename = 'labelled-heatmap')

'''

（3）拟合二级学习模型

把五个一级模型针对训练集的预测结果y结合起来，

把五个一级模型针对测试集的预测结果y结合起来。

以上两个结果分别作为二次模型的训练和测试集的输入x，拟合二级学习模型。

'''

x\_train = np.concatenate(( et\_oof\_train, rf\_oof\_train, ada\_oof\_train, gb\_oof\_train, svc\_oof\_train), axis=1)

x\_test = np.concatenate(( et\_oof\_test, rf\_oof\_test, ada\_oof\_test, gb\_oof\_test, svc\_oof\_test), axis=1)

'''

用XGBoost算法拟合二级学习模型

XGBoost参数：

max\_depth： 想要增长的树的深度。 若设置过高，可能会过拟合。 default = 6

gamma： L1正则化项中叶结点数T前面乘的系数。 越大，算法越保守。 default = 0

min\_child\_weight： L2正则化项中叶结点样本权重和（欧米伽之和）的最小值。 default = 1

lambda： L2正则化项前的系数。 default = 1

eta: 即learning\_rate。在每个增压步骤中使用的步骤尺寸缩小以防止过拟合，即提高鲁棒性。 default = 0.3

subsample： 控制每棵树随机采样的比例。减小该值，算法更保守，避免过拟合。但若该值设置过小，可能会导致欠拟合。典型值：0.5-1。 default = 1

colsample\_bytree： 控制每棵树随机采样的列数的占比(每一列是一个特征)。类似于RF随机选取特征子集。 典型值：0.5-1。 default = 1

scale\_pos\_weight： 在各类别样本十分不平衡时，把这个参数设定为一个正值，可以使算法更快收敛。 default = 1

nthread： 用来进行多线程控制，应当输入系统的核数。如果你希望使用CPU全部的核，就不要输入这个参数，或者设置为-1，算法会自动检测它。类似于n\_jobs

objective： 该参数定义需要被最小化的损失函数。default = 'reg:linear'

最常用的值有：

binary:logistic： 二分类的逻辑回归，返回预测的概率(不是类别)。

multi:softmax： 使用softmax的多分类器，返回预测的类别(不是概率)。在这种情况下，还需多设一个参数：num\_class(类别数目)。

multi:softprob： 和multi:softmax参数一样，但是返回的是每个数据属于各个类别的概率。

'''

gbm = xgb.XGBClassifier(

learning\_rate = 0.02,

n\_estimators= 2000,

max\_depth = 4,

min\_child\_weight = 2,

gamma = 0.9,

subsample = 0.8,

colsample\_bytree = 0.8,

objective = 'binary:logistic',

nthread = -1,

scale\_pos\_weight = 1 ).fit(x\_train, y\_train)

predictions = gbm.predict(x\_test)

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

'''

根据已训练和fit出的所有的一级和二级模型，将预测输出到适用于Titanic比赛的格式，生成提交文件

'''

StackingSubmission = pd.DataFrame({ 'PassengerId': PassengerId,

'Survived': predictions })

StackingSubmission.to\_csv("StackingSubmission.csv", index = False)

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

'''

Summary总结：

以上是一种非常简单的集成堆叠模型的方式。

在高级别的Kaggle比赛中，该方法被广泛应用。

步骤为：

（1）选择多个“好而不同”的一级分类器，

（2）用这些一级分类器分别交叉验证并拟合得到的预测结果y（分训练集上的和测试集上的）作为新的训练集和测试集输入x，为之后堆叠作准备

（3）对一级分类器组合以及超过2级的堆叠。

采取一些额外的步骤提高泛化性能：

（1）在训练模型中实现良好的交叉验证策略，以找到最佳参数值（调参）；

（2）引入更多种基础模型进行学习。 结果越不相关，最终得分越好。“好而不同”

比如引入LR：

from sklearn.linear\_model import LogisticRegression

#LR参数设置

lr\_params = {

'C':1.0,

'penalty':'l1',

'tol':1e-6

}

注：此问题引入LR模型之前准确率为79.9%，引入后却只有77.5%

'''