****

**计算机学院**

**数据仓库与数据挖掘课程项目报告**

**设计题目**： 科赛竞赛银行二分类解题报告

**学生姓名**： 於文卓 牟宇 张明浩

**学 号**：17061833 17272224 17051734

**指导教师**： 魏丹

**目 录**

目录

[1.项目背景 3](#_Toc28129720)

[1.1 项目简介 3](#_Toc28129721)

[1.2 网格搜索 3](#_Toc28129722)

[1.3 集成学习 3](#_Toc28129723)

[2.项目分工 3](#_Toc28129724)

[3.实验环境 4](#_Toc28129725)

[4.主要设计思想 4](#_Toc28129726)

[4.1 实验工具介绍 4](#_Toc28129727)

[4.2数据预处理方法设计 4](#_Toc28129728)

[4.3数据挖掘算法的选择与设计 5](#_Toc28129729)

[4.4性能评估方法 10](#_Toc28129730)

[5.项目实验过程 10](#_Toc28129731)

[5.1已实现的功能点 10](#_Toc28129732)

[5.2创新点 11](#_Toc28129733)

[5.3项目实现的主要步骤 11](#_Toc28129734)

[（一） 数据分析与可视化（由张明浩同学完成） 11](#_Toc28129735)

[（二） 数据分类与预测（於文卓和牟宇共同完成） 15](#_Toc28129736)

[（三） 特征工程优化（於文卓） 19](#_Toc28129737)

[5.4使用说明 20](#_Toc28129738)

[6.收获和建议 20](#_Toc28129739)

[7.项目源代码（每个部分写明是哪位同学完成） 21](#_Toc28129740)

# 1.项目背景

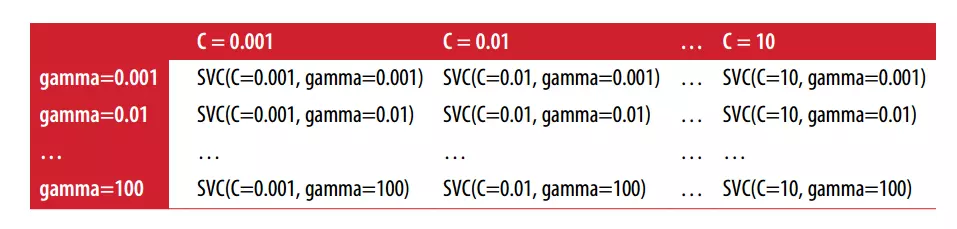
## 项目简介

本项目来自和鲸科赛竞赛平台的一个二分类比赛，比赛提供的数据集是与葡萄牙银行机构的营销活动相关。这些营销活动以电话为基础，银行客服人员需要联系客户来确认客户是否将购买这个银行的产品（定期存款产品）**，**即预测其目标是预测客户是“1否“0“会购买该银行的产品。

## 网格搜索

一种调参手段；**穷举搜索**。所有候选的参数选择中，通过循环遍历，尝试每一种可能性，表现最好的参数就是最终的结果。其原理就像是在数组里找最大值。

以有两个参数的模型为例，参数a有3种可能，参数b有4种可能，把所有可能性列出来，可以表示成一个3\*4的表格，其中每个cell就是一个网格，循环过程就像是在每个网格里遍历、搜索，所以叫网格搜索grid search。



## 集成学习

在机器学习的有监督学习算法中，我们的目标是学习出一个稳定的且在各个方面表现都较好的模型，但实际情况往往不这么理想，有时我们只能得到多个有偏好的模型（弱监督模型，在某些方面表现的比较好）。集成学习就是组合这里的多个弱监督模型以期得到一个更好更全面的强监督模型，集成学习潜在的思想是即便某一个弱分类器得到了错误的预测，其他的弱分类器也可以将错误纠正回来。

# 2.项目分工



# 3.实验环境

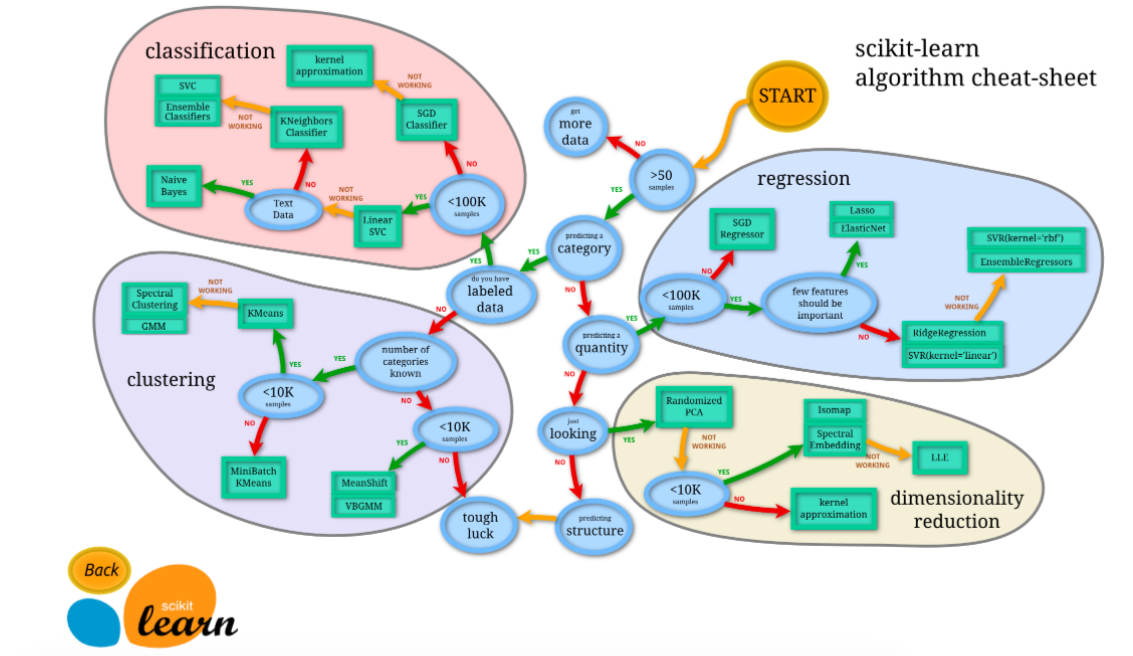
操作系统：Win10

编程语言：Python3.6、Anaconda3

运行平台：Jupyter Notebook、

# 4.主要设计思想

## 4.1 实验工具介绍



1. Jupyter Notebook

Jupyter Notebook 是一个在浏览器中使用的交互式的笔记本，可以实现将代码、文字完美结合起来，它的受众群体大多数是一些从事数据科学领域相关（机器学习、数据分析等）的人员。

1. Scikit-Learn

Scikit-learn项目最早由数据科学家David Cournapeau在2007年发起，需要NumPy和SciPy等其他包的支持，是Python语言中专门针对机器学习应用而发展起来的一款开源框架。

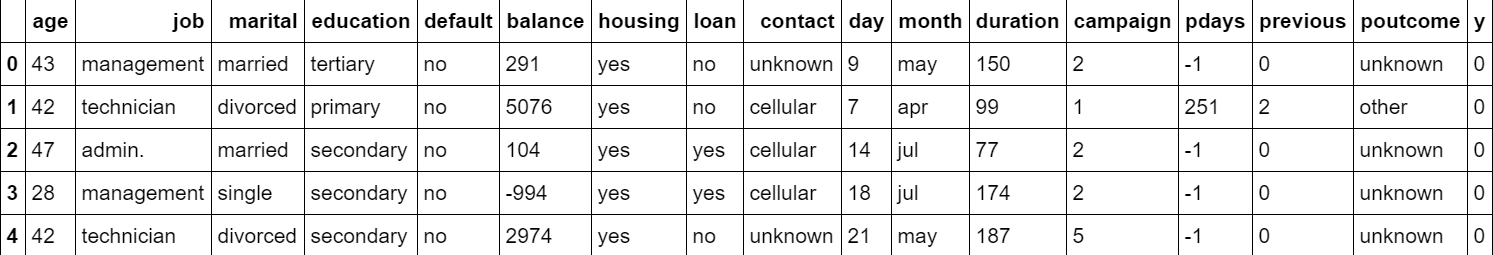
作为专门面向机器学习的Python开源框架，Scikit-learn可以在一定范围内为开发者提供非常好的帮助。它内部实现了各种各样成熟的算法，容易安装和使用，样例丰富，而且教程和文档也非常详细。

## 4.2数据预处理方法设计

**1）数据读入**

使用pandas中的read\_csv函数读入csv文件

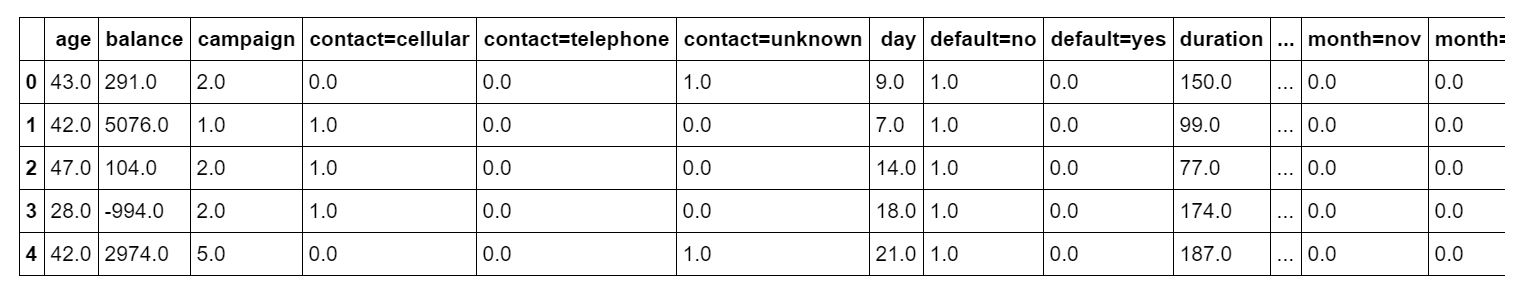
|  |
| --- |
| train\_set = pd.read\_csv("./train\_set.csv")  test\_set = pd.read\_csv("./test\_set.csv")  train\_set = pd.DataFrame(train\_set)  test\_set = pd.DataFrame(test\_set)  train\_set = train\_set.drop(columns="ID")  train\_set.head()  test\_set.head() |



1. **One-Hot**

这个数据集大部分数据都是离散数据，很多特征的值都是文本的形式，这样的特征无法训练，需要将其编码，具体方式为讲特征的所有可能出现的值转成新的特征，用0或1来表示某一个样本是否具有这个特征，这种方式叫做One-Hot编码。以下是经过One-Hot编码后的数据结果,最终数据集有原来的变为(25317, 18)扩展为(25317, 53)

|  |
| --- |
| from sklearn.feature\_extraction import DictVectorizer  vec = DictVectorizer(sparse=False)  train\_set = vec.fit\_transform(train\_set.to\_dict(orient="record"))  train\_set = pd.DataFrame(train\_set)  train\_set.columns = vec.feature\_names\_  train\_set.head()  train\_set.shape |



## 4.3数据挖掘算法的选择与设计

（此步骤由牟宇同学与於文卓同学共同完成）

1）使用十折交叉验证，从众多模型中筛选出最优的模型

|  |
| --- |
| #定义好不同的模型  models = {}  models['LR'] = LogisticRegression()  models['LDA'] = LinearDiscriminantAnalysis()  models['KNN'] = KNeighborsClassifier()  models['DTC'] = DecisionTreeClassifier()  models['NB'] = GaussianNB()  models["NN"] = MLPClassifier()  models["RF"] = RandomForestClassifier()  models["AdaBoost"] = AdaBoostClassifier(DecisionTreeClassifier())  models["Bagging"] = BaggingClassifier(DecisionTreeClassifier())  models["Xgboost"] = XGBClassifier()  models['SVM'] = SVC()  results = []  num\_folds = 10  seed = 7  #使用准确度进行评估  kfold = KFold(n\_splits=num\_folds, random\_state=seed)  for key in tqdm(models):  #交叉验证每次运行的评分数组  cv\_results = cross\_val\_score(  models[key],  x\_train,  y\_train,  cv=kfold,  )  results.append(cv\_results)  print('%s::%f(%f)' % (key, cv\_results.mean(), cv\_results.std())) |

2）网格搜索

具体步骤为：

1. **给定learning rate，搜索合适的n\_estimators**

|  |
| --- |
| def modelfit(alg, X, Y, useTrainCV=True, cv\_folds=10, early\_stopping\_rounds=50):  if useTrainCV:  xgb\_param = alg.get\_xgb\_params()  xgtrain = xgb.DMatrix(X.values, label=Y.values)  cvresult = xgb.cv(xgb\_param,  xgtrain,  num\_boost\_round=alg.get\_params()['n\_estimators'],  nfold=cv\_folds,  metrics='auc',  early\_stopping\_rounds=early\_stopping\_rounds,  verbose\_eval=False)  alg.set\_params(n\_estimators=cvresult.shape[0])  #Fit the algorithm on the data  alg.fit(X, Y, eval\_metric='auc')  #Predict training set:  dtrain\_predictions = alg.predict(X)  dtrain\_predprob = alg.predict\_proba(X)[:, 1]  #Print model report:  print("\nModel Report")  print("Best Num Of Tree:{}".format(cvresult.shape[0]))  print("Accuracy : %.4g" %  metrics.accuracy\_score(Y.values, dtrain\_predictions))  print("AUC Score (Train): %f" % metrics.roc\_auc\_score(Y, dtrain\_predprob))  feat\_imp = pd.Series(  alg.get\_booster().get\_fscore()).sort\_values(ascending=False)  b = plt.figure(figsize=(14,5),dpi=80)  feat\_imp.plot(kind='bar', title='Feature Importances')  plt.ylabel('Feature Importance Score')  xgb1 = XGBClassifier(learning\_rate=0.1,  n\_estimators=1000,  max\_depth=5,  min\_child\_weight=1,  gamma=0,  subsample=0.8,  colsample\_bytree=0.8,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27)  # 得到Best Num Of Tree  modelfit(xgb1, x\_train, y\_train) |

1. **网格搜索调整max\_depth 和 min\_child\_weight**

|  |
| --- |
| param\_test1 = {  'max\_depth': range(3, 10, 2),  'min\_child\_weight': range(1, 6, 2)  }  gsearch1 = GridSearchCV(estimator=XGBClassifier(learning\_rate=0.1,  n\_estimators=186,  max\_depth=5,  min\_child\_weight=2,  gamma=0,  subsample=0.8,  colsample\_bytree=0.8,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27),  param\_grid=param\_test1,  scoring='roc\_auc',  n\_jobs=4,  iid=False,  cv=5)  gsearch1.fit(x\_train, y\_train)  res1 = gsearch1.cv\_results\_  res2 = gsearch1.best\_params\_  res3 = gsearch1.best\_score\_  print(res2)  with open("./max\_depth\_min\_child\_weight.txt", "w") as f:  ans = "grid\_scores\_:" + str(res1) + "\n best\_params\_:" + str(  res2) + "\n best\_score\_:" + str(res3)  f.write(ans) |

1. **网格搜索调整gamma**

|  |
| --- |
| param\_test3 = {'gamma': [i / 10.0 for i in range(0, 5)]}  gsearch3 = GridSearchCV(estimator=XGBClassifier(learning\_rate=0.1,  n\_estimators=179,  max\_depth=5,  min\_child\_weight=3,  gamma=0,  subsample=0.8,  colsample\_bytree=0.8,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27),  param\_grid=param\_test3,  scoring='roc\_auc',  n\_jobs=4,  iid=False,  cv=5)  gsearch3.fit(x\_train, y\_train)  res1 = gsearch3.cv\_results\_  res2 = gsearch3.best\_params\_  res3 = gsearch3.best\_score\_  print(res2)  with open("./gamma.txt", "w") as f:  ans = "grid\_scores\_:" + str(res1) + "\n best\_params\_:" + str(  res2) + "\n best\_score\_:" + str(res3)  f.write(ans) |

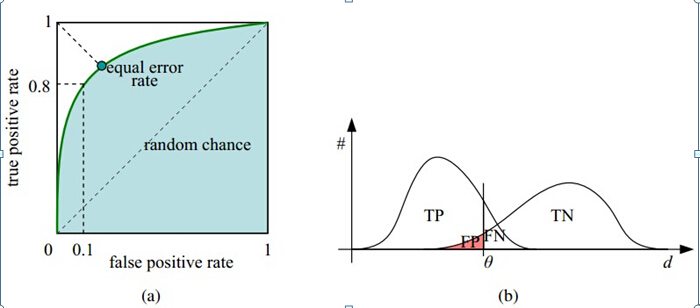
1. **网格搜索调整subsample和colsample\_bytree**

|  |
| --- |
| param\_test4 = {  'subsample': [i / 10.0 for i in range(6, 10)],  'colsample\_bytree': [i / 10.0 for i in range(6, 10)]  }  gsearch4 = GridSearchCV(estimator=XGBClassifier(learning\_rate=0.1,  n\_estimators=186,  max\_depth=6,  min\_child\_weight=6,  gamma=0.3,  subsample=0.8,  colsample\_bytree=0.9,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27),  param\_grid=param\_test4,  scoring='roc\_auc',  n\_jobs=4,  iid=False,  cv=5)  gsearch4.fit(x\_train, y\_train)  res1 = gsearch4.cv\_results\_  res2 = gsearch4.best\_params\_  res3 = gsearch4.best\_score\_  print(res2)  with open("./subsample\_colsample\_bytree.txt", "w") as f:  ans = "grid\_scores\_:" + str(res1) + "\n best\_params\_:" + str(  res2) + "\n best\_score\_:" + str(res3)  f.write(ans) |

1. **降低Learning Rate，再次搜索合适的estimators**

## 4.4性能评估方法

本实验所使用的性能评估指标均为AUC



AUC（Area Under Curve）被定义为ROC曲线下的面积。

# 5.项目实验过程

## 5.1已实现的功能点

1. 实现对数据的可视化分析（由张明浩同学完成）
2. 最优模型的筛选（牟宇）
3. 网格搜索Xgboost调参（牟宇和於文卓）
4. 特征工程提高模型准确率（於文卓）

## 5.2创新点

通过研究数据集做了进一步特征工程，对数据集中的离散特征和连续特征进行特征交叉，从而很好的提高了排名

## 5.3项目实现的主要步骤

### 数据分析与可视化（由张明浩同学完成）

1. 数据来源介绍

本数据集由和鲸（Kesci）竞赛平台提供，选自UCI机器学习库中的【银行营销数据集（Bank Marketing Data Set）】

这些数据与葡萄牙银行机构的营销活动相关。这些营销活动以电话为基础，银行客服人员需要联系客户来确认客户是否将购买这个银行的产品（定期存款产品）

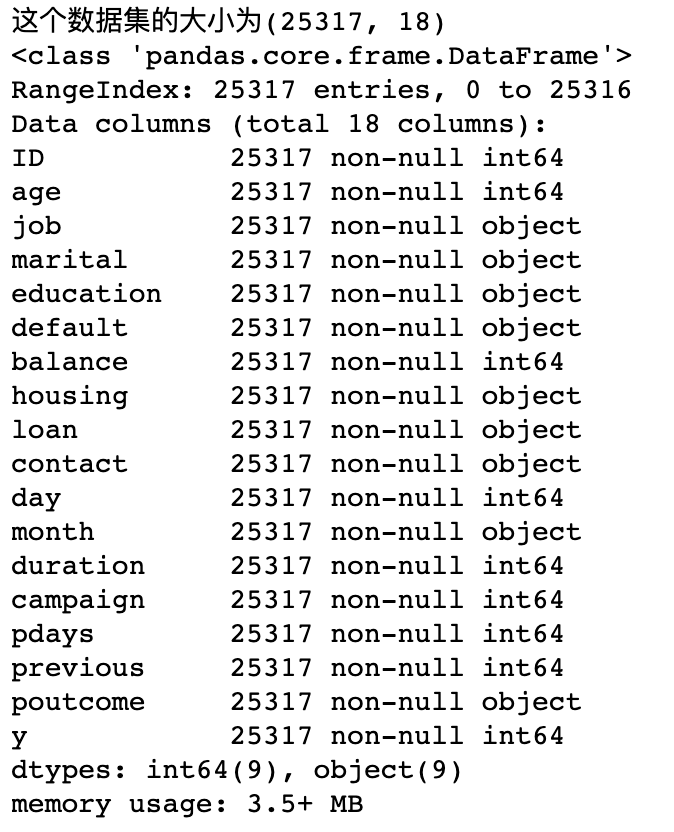


1. 数据总体分析

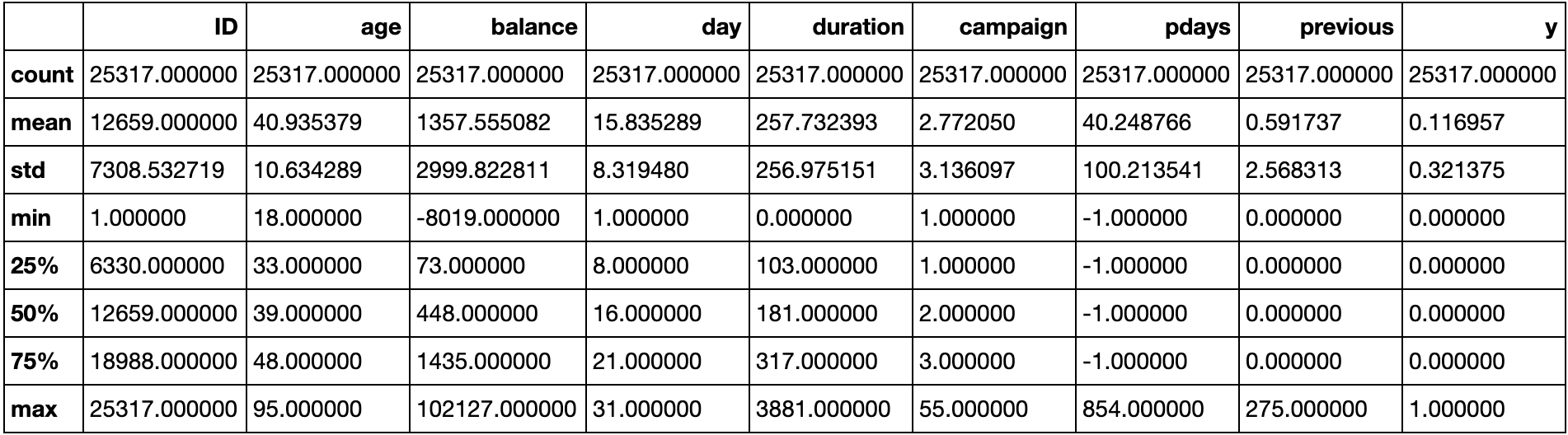
导入相关工具包，使用的工具包有Numpy,pandas,seaborn和matplotlib。Numpy,pandas主要用来读取、筛选和处理数据,seaborn和matplotlib用来完成数据的可视化

首先利用pandas读入数据集，将数据集转换为Dataframe。

通过info() 函数可以简单了解数据集的信息，比如数据集的尺寸，数据集占用的内存大小，每个特征的属性包括名字和数值类型等。

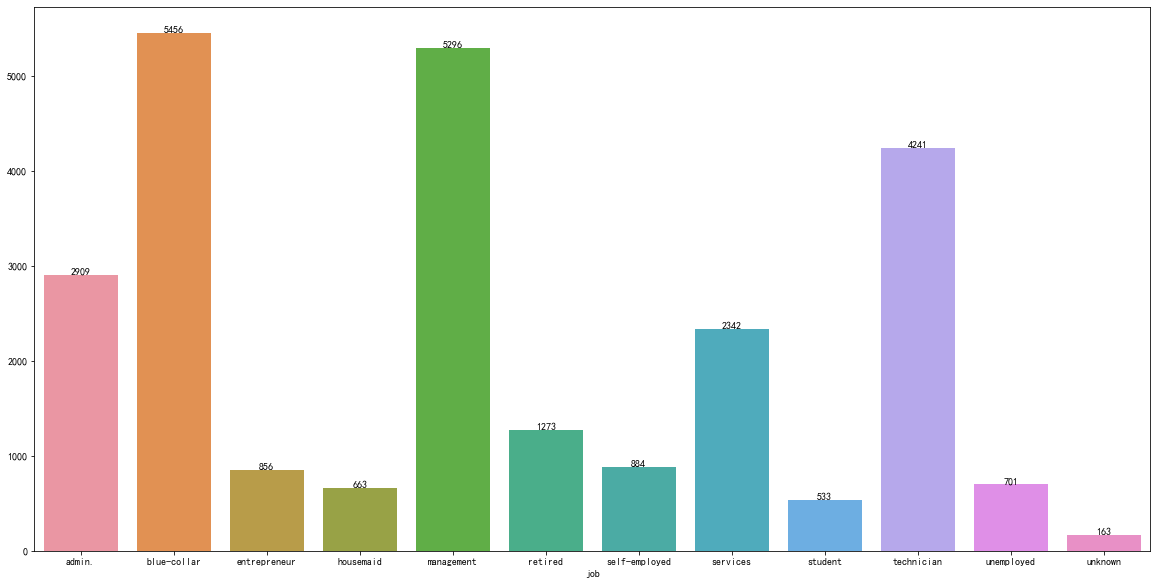


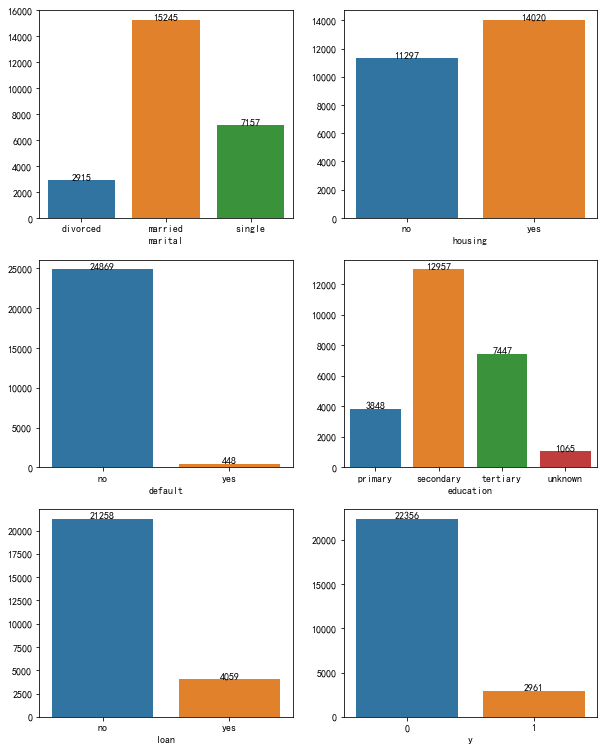
通过describe() 函数发现balance,duration,和pdays的std较大，说明值的分布范围较广，后期可能需要进行处理。年龄从最低18岁到最大95岁，跨度也比较大。



我们可以利用seaborn进行可视化，我们统计了样本中职业、教育情况、婚姻、住房贷款、个人贷款、违约记录六个字段的值的分布情况，以下是我从这个样本中了解到的相关信息：

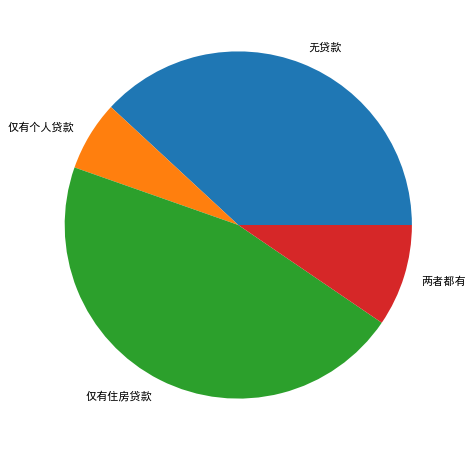
* 大部分人的职业为蓝领、部门经理和技术人员。大部分人都接受了中学及以上的教育。样本中多数人结过婚，有一定的住房贷款和个人贷款，他们大都信用较好，很少有违约的行为。可以了解到这个样本的人群大多是一群教育程度高、有家庭有事业但是受背负一定数量贷款的人群，这群人需要一定的理财产品来管理他们的收入。
* 最后一个图是这个样本的标签，0表示最终不会购买产品，1表示购买。可以看到这是个不均衡样本，之后再数据预处理的时候要对这种情况做一定的处理。
* 数据中有大量unknown的样本，在数据预处理的时候需要特殊处理





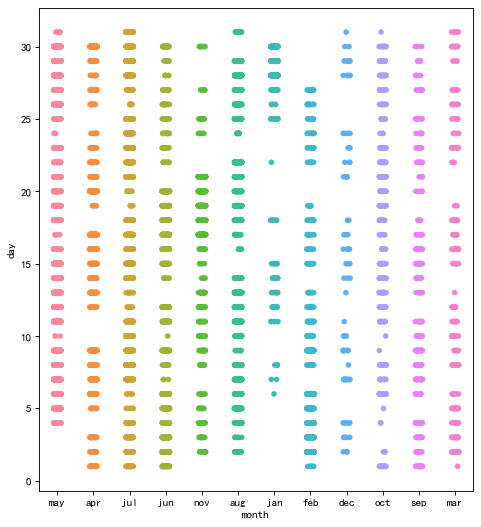
1. 数据特征联系

我们可以把同个特征的不同值联系起来，对多个值同时进行可视化。比如贷款：



可以看出有部分人不仅有住房贷款还有个人贷款的压力，在后续做分类的时候这两个特征可能会对结果有影响。

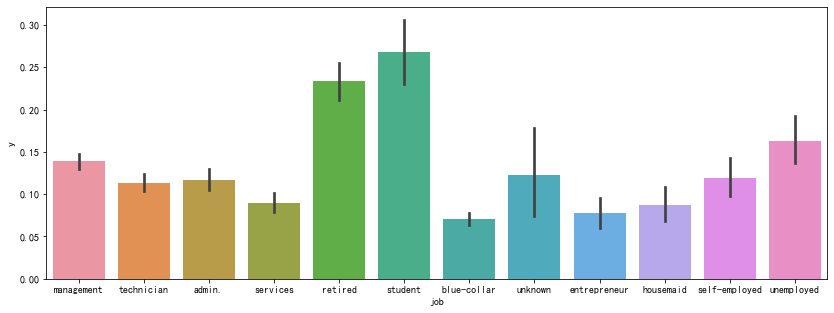
我们再把day和month用分类散点图展示出来，表示银行最后一次联系某用户的日期：

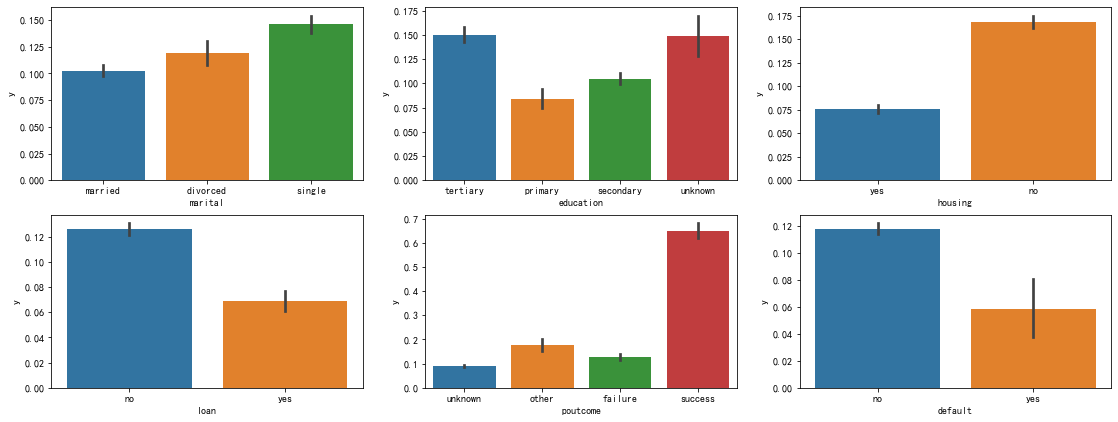


可以看到这个银行几乎全年每天都在联系客户，而节假日像是十二月二十五日圣诞节到一月的假期，几乎没什么数据，可以表明这份数据是真实的。

1. 数据与y标签的联系

以上的可视化大概了解了各个特征在所有样本中的分布情况，接下来我会筛选几个特征去寻找与y标签之间的联系，希望挖掘出与购买产品这个结果相关的信息。





上图筛选了职业、婚姻、教育、有无贷款、上次活动结果和违约记录这几个特征后，分析了最终是否会购买产品之间的关联性，得到如下分析：

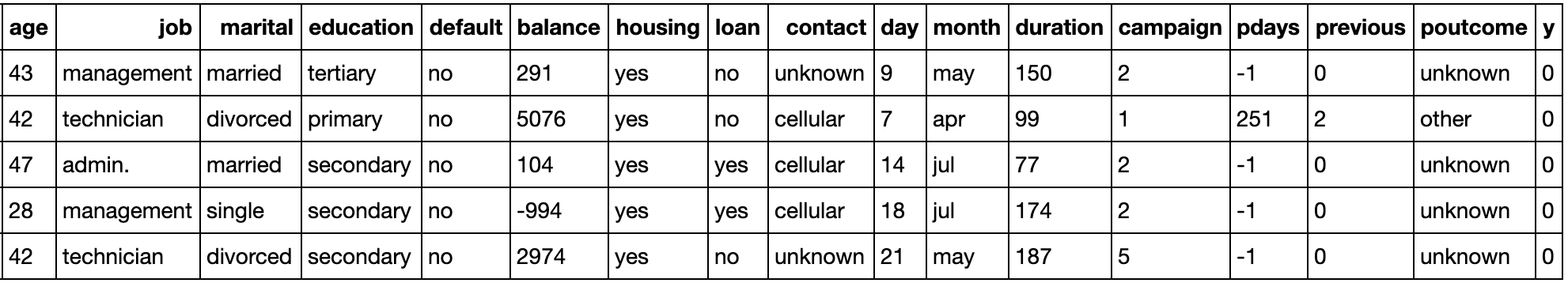
* 虽然样本中蓝领、部门经理、技术人员这三种职业最多，但其大部分人最终不会购买产品。反而是退休人员和学生更加可能购买产品
* 没有负债的人相比有负债的人更加容易购买产品
* 上一次活动成功的人有很大概率会再次购买产品
* 受过高等教育的人以及单身人士有相对较高的概率会购买产品

### 数据分类与预测（於文卓和牟宇共同完成）

这个数据来源为和鲸{Kesci}练习赛赛题，除了分析数据特征外，更重要的是需要完成一个分类任务，其目标是预测客户是(' 1 ')或者否(' 0 ')购买该银行的产品。最终需要预测测试数据中样本为1的概率，将结果打包成csv文件上传到平台，平台以AUC为评判标准评判最终结果。

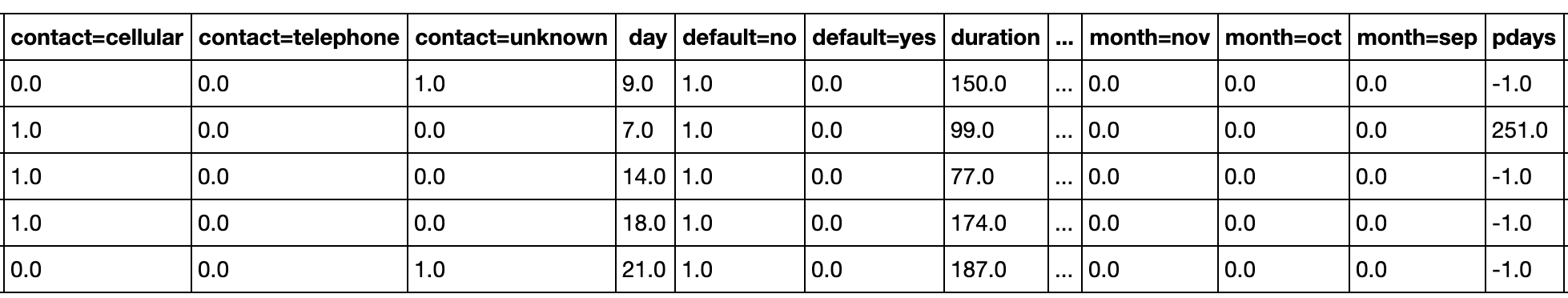
1. 数据读入与简单处理

分别读入训练数据集合测试是数据及。训练数据集带有标签，而测试数据集不带标签，部分数据展示如下：





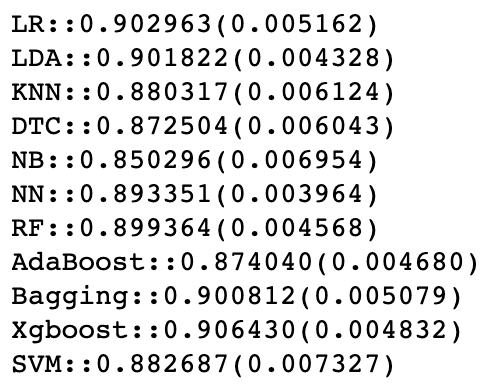
如前一节分析所讲，这个数据集大部分数据都是离散数据，很多特征的值都是文本的形式，这样的特征无法训练，需要将其编码，具体方式为讲特征的所有可能出现的值转成新的特征，用0或1来表示某一个样本是否具有这个特征，这种方式叫做One-Hot编码。以下是经过One-Hot编码后的数据结果,最终数据集有原来的变为(25317, 18)扩展为(25317, 53)



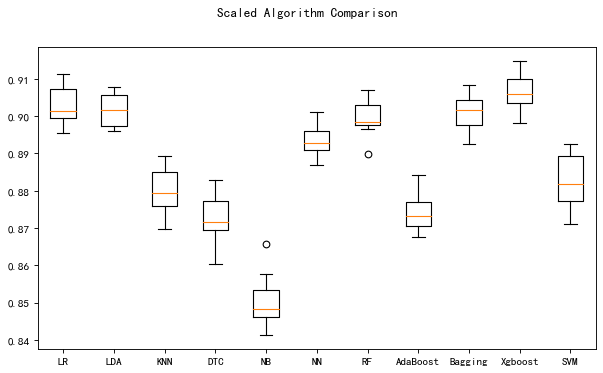
1. 模型选择（牟宇）

常用的分类模型在Sklearn中有很多，如何选择一个合适的模型其实是需要自己不断尝试的，我在前期尝试了很多不同的模型，比如逻辑回归、KNN、决策树、高斯贝叶斯、神经网络、SVM还有集成学习的一些模型比如随机森林、Adaboost等。我将数据集分成训练集和验证集，采用对不同的模型使用十折交叉验证的方式，筛选出一个较合适的模型，为方便起见，各个模型的参数均使用默认值。

各个模型结果如下：



我们使用正态化-箱线图来更直观地观察各个模型的结果：



最后结果显示在这十一种模型中，贝叶斯的效果最不尽如人意，Xgboost的效果最优，通过我网上提交成绩的排名显示也是Xgboost模型要优于其他分类模型。我将这个模型作为我的核心模型，使用一定的方式调整参数，使之达到最优的效果。

1. 网格搜索最优参数（牟宇和於文卓）

通过网格搜索，主要调整如下几个参数：

* 学习率（learning rate）<=通过交叉验证得到
* 决策树数量（n\_estimators）
* 决策树相关参数

（max\_depth,min\_child\_weight,gamma,subsample,colsample\_bytree）

* 正则化参数（lambda,alpha）<=正则化参数可以降低模型复杂度

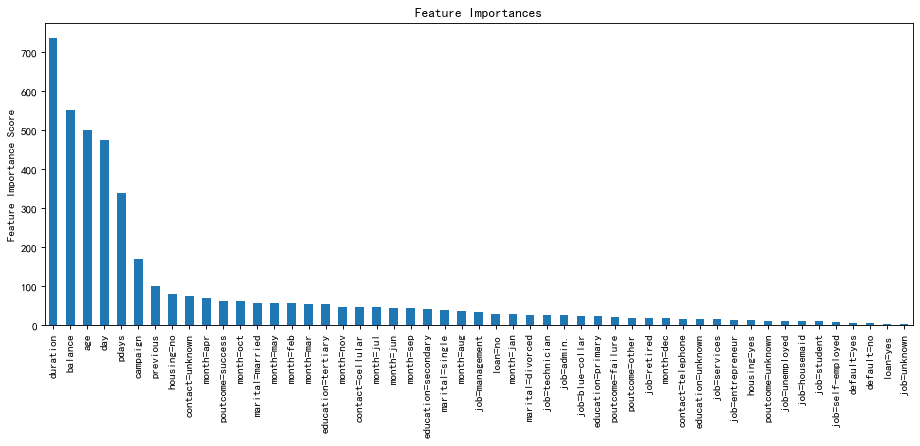
首先给Xgboost中的各个参数调整一个合适的默认值，其次定义一个函数modelfit，这个函数会使用xgb自带的交叉验证库调整决策树的数量，根据最后的预测结果，选择一个最优的数量，并且打印Feature Importances图，其表示的是数据集中各个特征的重要程度。

Model Report

Best Num Of Tree:179

Accuracy : 0.9395

AUC Score (Train): 0.967925



得到在学习率为0.1的情况下，最好的数量为179。通过Xgb分析，我了解到对于最后结果影响最大的几个特征为最后一次联系的交流时长和每年账户的平均余额。

接下来使用网格搜索调整max\_depth和 min\_child\_weight，得到最优的结果为max\_depth: 5，min\_child\_weight: 3。

同理，带入上面已经确定的参数，来确定gamma和colsample\_bytree、subsample的值。得到最优gamma值为0.3，colsample\_bytree为0.8 ，subsample为0.9。

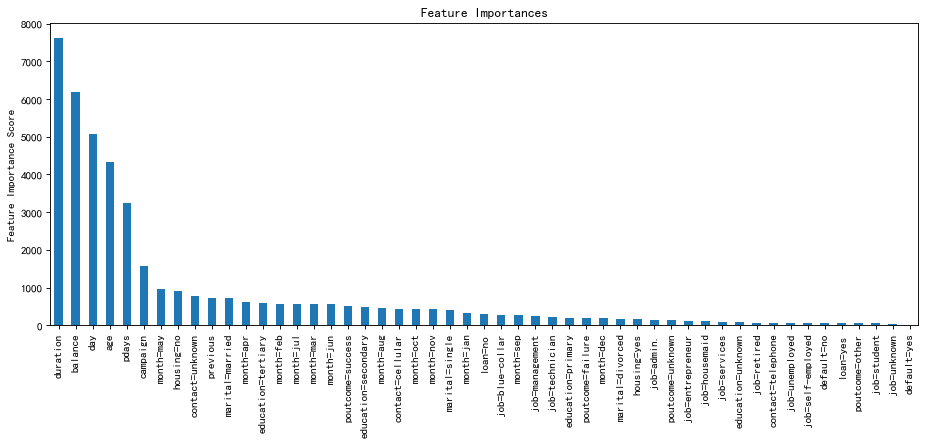
前面的参数基本都调完了就轮到学习率了，将xbg模型的参数换上刚才调好的，将学习率减小，并增加决策树的数量，重新做交叉验证，寻找一个最合适的树的数量。

Model Report

Best Num Of Tree:1605

Accuracy : 0.9381

AUC Score (Train): 0.968239



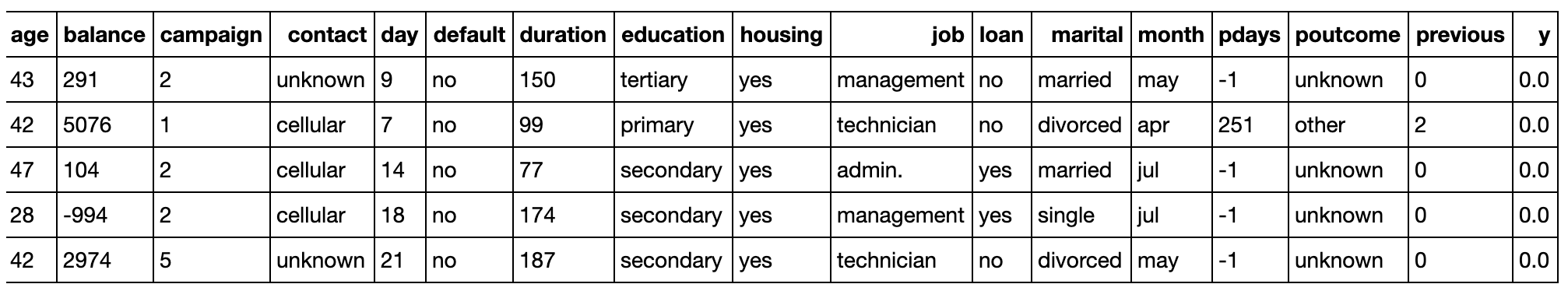
最终得到的树的数量为1650。

至此我的xgboost模型已经调完，在测试集的预测结果为91%左右。直接可以用此模型去预测结果了，最终结果为0.93473540，但排名一般，排在280+/745。

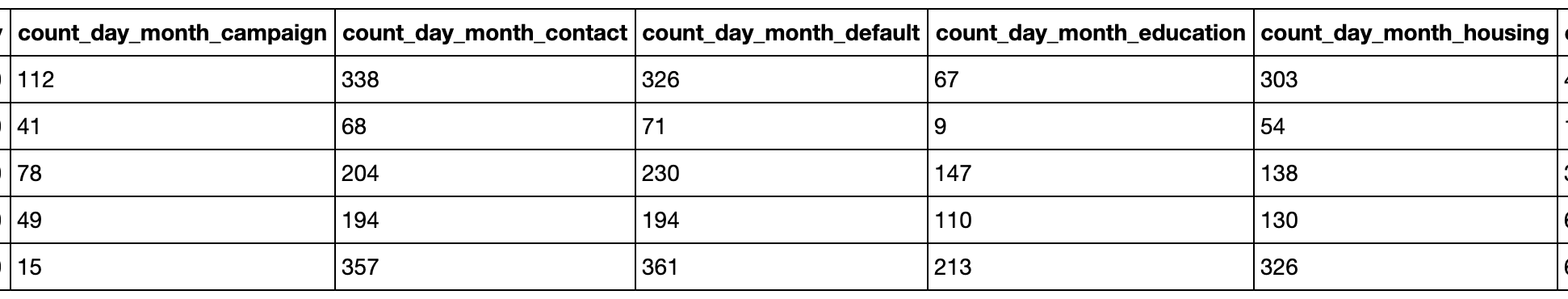
### 特征工程优化（於文卓）

有这么一句话在业界广泛流传，数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已。那特征工程到底是什么呢？顾名思义，其本质是一项工程活动，目的是最大限度地从原始数据中提取特征以供算法和模型使用。

1. 分离连续特征和离散特征

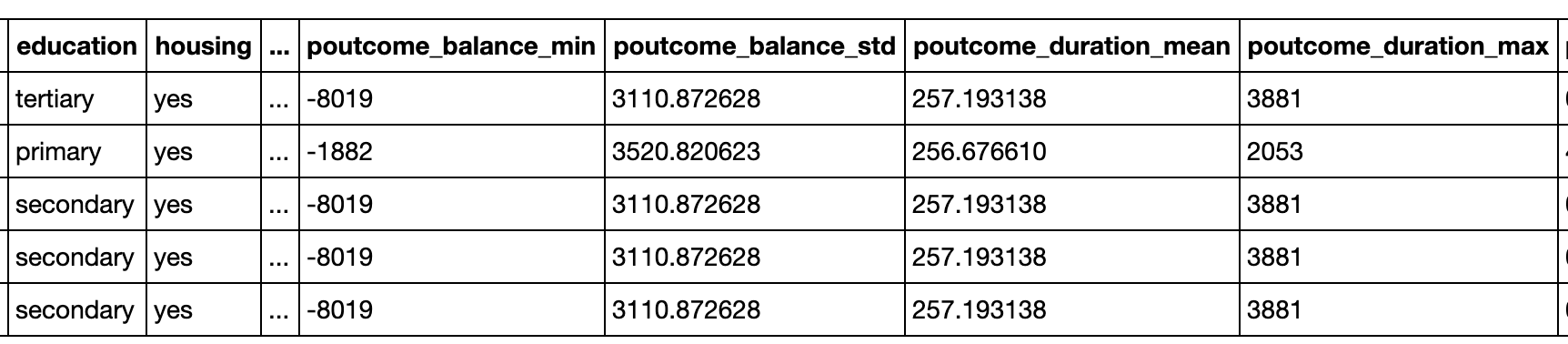


1. 离散特征和时间融合统计



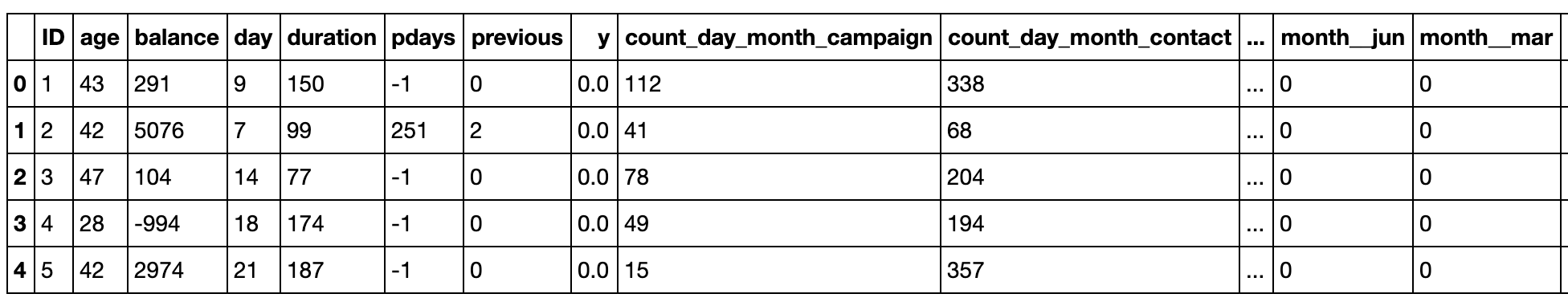
1. 特征交叉

首先按照一个离散特征分组,计算一个离散特征和其他所有特征的关系,如果其他特征是离散特征,计算count和nunique,如果是连续特征,计算mean等。



1. One-Hot编码离散值

至此，原来的数据集已经从(25317, 18)扩展为(25317, 535)。



1. 分离训练集和测试集

**最终选择的主要特征为**

['age','balance','day','duration','pdays','previous','campaign\_contact\_count','campaign\_duration\_max','campaign\_duration\_std','campaign\_age\_max','campaign\_day\_std','campaign\_balance\_min','campaign\_previous\_std','contact\_campaign\_count','contact\_duration\_max','education\_campaign\_count','education\_duration\_mean','housing\_campaign\_count','job\_campaign\_count','job\_duration\_mean','job\_duration\_max','job\_age\_mean',

'job\_age\_max','job\_age\_std','job\_day\_mean','job\_day\_std','job\_pdays\_max','job\_previous\_mean','loan\_campaign\_count','marital\_campaign\_count','month\_campaign\_count','month\_campaign\_nunique','month\_duration\_mean','month\_age\_mean','month\_age\_max','month\_day\_mean','month\_day\_std','month\_pdays\_mean','month\_pdays\_max','month\_pdays\_std', 'month\_balance\_mean', 'month\_previous\_max', 'month\_previous\_std', 'poutcome\_campaign\_count','count\_day\_month\_campaign','count\_day\_month\_contact', 'count\_day\_month\_default', 'count\_day\_month\_education','count\_day\_month\_housing', 'count\_day\_month\_job','count\_day\_month\_loan','count\_day\_month\_marital','count\_day\_month\_poutcome']

1. 网格搜索调参Xgboost

最终得到参数为：

learning\_rate=0.01

n\_estimators=1489

max\_depth=5

min\_child\_weight=6

gamma=0

subsample=0.8

colsample\_bytree=0.85

## 5.4使用说明

使用Anaconda新建一个虚拟环境，安装好代码所需的函数库Sklearn、Numpy、Pandas等，运行Jupyter Notebook，打开代码文件运行即可

# 6.收获和建议

张明浩：对数据进行可视化分析有助于直观地理解数据特征之间的关系，方便为之后做数据预处理提供依据。最基本的可视化可以统计各特征的数量、最大值、最小值、平均值、方差等，也可以寻找几个特征之间的联系。常用的可视化工具有matplotlib、seaborn等，也可以使用ploty这个库，提供更加全面的可视化功能

牟宇：学会了使用交叉验证的方式评估模型，从而筛选出最优的模型。也学习了使用网格搜索进行调参，对数据挖掘的基本流程有了更深的认识。

於文卓：我们经过前期的数据预处理和训练后得到的模型能达到93%左右的准确率，但提交上去的排名不是很靠前。所以我去了解了更多能提高竞赛排名的一些方法。通过网上搜索，了解到了特征工程，通过特征与特征之间的交叉，可以扩充特征数量。但问题是如何从500多项特征给运算带来了严重的负担，如何从500多项特征中筛选出与结果最相关的特征呢，我直接使用xgboost提供的feature importance图筛选。这样再经过网格搜索调参的模型确实能达到一个更加高的准确度

# 7.项目源代码（每个部分写明是哪位同学完成）

|  |
| --- |
| #!/usr/bin/env python  # coding: utf-8  # In[1]:  from IPython.core.interactiveshell import InteractiveShell  InteractiveShell.ast\_node\_interactivity = 'all' #默认为'last'  import warnings  warnings.filterwarnings('ignore')  get\_ipython().run\_line\_magic('matplotlib', 'inline')  # ## 张明浩 数据分析及可视化  #  # ### 数据来源介绍  # 本数据集由和鲸（Kesci）竞赛平台提供，选自UCI机器学习库中的【银行营销数据集（Bank Marketing Data Set）】  # > [数据来源](https://www.kesci.com/home/competition/5c234c6626ba91002bfdfdd3/content/1)  #  # 这些数据与\*\*葡萄牙\*\*银行机构的营销活动相关。这些营销活动以电话为基础，银行客服人员需要联系客户来确认客户是否将购买这个银行的产品（定期存款产品）  #  # 本文首先将通过可视化的方式简单分析数据的各种情况  #  #  # ### 数据总体分析  #  # 首先导入相关工具包，使用的工具包有`Numpy`,`pandas`,`seaborn`和`matplotlib`。`Numpy`,`pandas`主要用来读取、筛选和处理数据,`seaborn`和`matplotlib`用来完成数据的可视化  # In[2]:  import numpy as np # linear algebra  import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)  import seaborn as sns  import matplotlib.pyplot as plt  plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签  plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号  from collections import Counter  # In[3]:  #读入数据  data = pd.read\_csv("./train\_set.csv")  data = pd.DataFrame(data)  # In[4]:  print(f'这个数据集的大小为{data.shape}')  data.info()  # 首先通过`data.info`可以简单了解数据集的信息，比如数据集的尺寸，数据集占用的内存大小，每个特征的属性包括名字和数值类型等。具体的字段说明见下图  # ![字段说明](./statement.jpg)  # + 可以发现，这个数据集一共有25317条数据，每条数据包含18个特征，每条数据都不存在缺失值  # + 原本为String字段的数值，通过pandas读取后变成了object类型的格式  # In[5]:  data.describe()  # 通过`describe`函数发现  #  # `balance`,`duration`,和`pdays`的std较大，说明值的分布范围较广，后期可能需要进行处理。年龄从最低18岁到最大95岁，跨度也比较大  # In[6]:  plt.figure(figsize=(20,10))  # sns.barplot("job","y",data=data)  pic = data.groupby("job")["ID"].count()  g = sns.barplot(pic.index,pic.values)  for i in range(len(pic)):  a = g.text(i,pic[i],pic[i],color="black",ha="center")  # In[7]:  plt.figure(figsize=(10,13))  name\_list = ["marital","housing","default","education","loan","y"]  for i in range(len(name\_list)):  b = plt.subplot(3,2,i+1)  pic = data.groupby(name\_list[i])["ID"].count()  g = sns.barplot(pic.index,pic.values)  for i in range(len(pic)):  a = g.text(i,pic[i],pic[i],color="black",ha="center")  # 统计了样本中职业、教育情况、婚姻、住房贷款、个人贷款、违约记录六个字段的值的分布情况，以下是我从这个样本中了解到的相关信息  #  # + 大部分人的职业为蓝领、部门经理和技术人员。大部分人都接受了中学及以上的教育。样本中多数人结过婚，有一定的住房贷款和个人贷款，他们大都信用较好，很少有违约的行为。可以了解到这个样本的人群大多是一群教育程度高、有家庭有事业但是受背负一定数量贷款的人群，这群人需要一定的理财产品来管理他们的收入。  #  # + 最后一个图是这个样本的标签，0表示最终不会购买产品，1表示购买。可以看到这是个不均衡样本，之后再数据预处理的时候要对这种情况做一定的处理。  #  # + 数据中有大量unknown的样本，在数据预处理的时候需要特殊处理  # ### 数据特征联系  #  # In[8]:  plt.subplots(figsize=(7,8), dpi= 80)  housing\_loan = data.loc[:,"housing":"loan"]  housing\_loan[housing\_loan=="yes"]= 1  housing\_loan[housing\_loan=="no"]= 0  res = housing\_loan.groupby("housing")["loan"].value\_counts()  total = len(housing\_loan)  neigher = res[0,0]/total  housing = res[1,0]/total  loan = res[0,1]/total  hl = res[1,1]/total  categories=["无贷款","仅有个人贷款","仅有住房贷款","两者都有"]  X = [neigher,loan,housing,hl]  a = plt.pie(X,labels=categories)  # 有部分人不仅有住房贷款还有个人贷款的压力，在后续做分类的时候这两个特征可能会对结果有影响  # In[9]:  day\_month = data.loc[:,"day":"month"]  fig, ax = plt.subplots(figsize=(7,8), dpi= 80)  sns.stripplot(x="month", y="day",size=5,data=day\_month,ax=ax)  # 上图显示的是最后一次联系某用户的日期，可以看到这个银行几乎全年每天都在联系客户，而节假日像是十二月二十五日圣诞节到一月的假期，几乎没什么数据，可以看到这份数据是真实的  # ### 与y标签的联系  # 以上的可视化大概了解了各个特征在所有样本中的分布情况，接下来我会筛选几个特征去寻找与y标签之间的联系，希望挖掘出与购买产品这个结果相关的信息  # In[10]:  plt.figure(figsize=(14,5))  feature = ["job"]  for i in range(len(feature)):  sns.barplot(feature[i],"y",data=data)  # In[11]:  plt.figure(figsize=(19,7))  feature = ["marital","education","housing","loan","poutcome","default"]  for i in range(len(feature)):  a = plt.subplot(2,3,i+1)  c = sns.barplot(feature[i],"y",data=data)  # 上图筛选了职业、婚姻、教育、有无贷款、上次活动结果和违约记录这几个特征后，分析了最终是否会购买产品之间的关联性，得到如下分析：  # + 虽然样本中蓝领、部门经理、技术人员这三种职业最多，但其大部分人最终不会购买产品。反而是退休人员和学生更加可能购买产品  # + 没有负债的人相比有负债的人更加容易购买产品  # + 上一次活动成功的人有很大概率会再次购买产品  # + 受过高等教育的人以及单身人士有相对较高的概率会购买产品  # ## 牟宇、於文卓数据分类与预测  #  # 这个数据来源为和鲸{Kesci}练习赛赛题，除了分析数据特征外，更重要的是需要完成一个分类任务，其目标是预测客户是(' 1 ')或者否(' 0 ')购买该银行的产品。最终需要预测测试数据中样本为1的概率，将结果打包成csv文件上传到平台，平台以AUC为评判标准评判最终结果  #  # ### 数据读入与简单处理  # 分别读入训练数据集合测试是数据及。训练数据集带有标签，而测试数据集不带标签，部分数据展示如下  # In[42]:  train\_set = pd.read\_csv("./train\_set.csv")  test\_set = pd.read\_csv("./test\_set.csv")  train\_set = pd.DataFrame(train\_set)  test\_set = pd.DataFrame(test\_set)  train\_set = train\_set.drop(columns="ID")  train\_set.head()  test\_set.head()  # 如前一节分析所讲，这个数据集大部分数据都是离散数据，很多特征的值都是文本的形式，这样的特征无法训练，需要将其编码，具体方式为讲特征的所有可能出现的值转成新的特征，用0或1来表示某一个样本是否具有这个特征，这种方式叫做One-Hot编码。以下是经过One-Hot编码后的数据结果,最终数据集有原来的变为(25317, 18)扩展为(25317, 53)  # In[43]:  from sklearn.feature\_extraction import DictVectorizer  vec = DictVectorizer(sparse=False)  train\_set = vec.fit\_transform(train\_set.to\_dict(orient="record"))  train\_set = pd.DataFrame(train\_set)  train\_set.columns = vec.feature\_names\_  train\_set.head()  train\_set.shape  # ### 模型选择（主要由牟宇完成）  #  # 常用的分类模型在Sklearn中有很多，如何选择一个合适的模型其实是需要自己不断尝试的，我在前期尝试了很多不同的模型，比如\*\*逻辑回归、KNN、决策树、高斯贝叶斯、神经网络、SVM\*\*还有集成学习的一些模型比如\*\*随机森林、Adaboost\*\*等。我将数据集分成训练集和验证集，采用对不同的模型使用十折交叉验证的方式，筛选出一个较合适的模型，为方便起见，各个模型的参数均使用默认值。  # In[83]:  from sklearn.tree import DecisionTreeClassifier  from sklearn.linear\_model import LogisticRegression  from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  from sklearn.ensemble import RandomForestClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import AdaBoostClassifier  from sklearn.naive\_bayes import GaussianNB  from sklearn.svm import SVC  import xgboost as xgb  from xgboost.sklearn import XGBClassifier  from sklearn.preprocessing import StandardScaler  from sklearn.neural\_network import MLPClassifier  from sklearn.ensemble import BaggingClassifier  from tqdm.notebook import tqdm  from sklearn.feature\_extraction import DictVectorizer  from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score, GridSearchCV  # In[64]:  # 将训练集的一部分拿出来作为测试集,用来最后评判模型在为训练数据上的表现  X = train\_set.iloc[:, :-1]  Y = train\_set.iloc[:, -1]  x\_train, x\_test, y\_train, y\_test = train\_test\_split(X,  Y,  test\_size=0.1,  random\_state=33)  x\_train.head()  # In[67]:  #定义好不同的模型  models = {}  models['LR'] = LogisticRegression()  models['LDA'] = LinearDiscriminantAnalysis()  models['KNN'] = KNeighborsClassifier()  models['DTC'] = DecisionTreeClassifier()  models['NB'] = GaussianNB()  models["NN"] = MLPClassifier()  models["RF"] = RandomForestClassifier()  models["AdaBoost"] = AdaBoostClassifier(DecisionTreeClassifier())  models["Bagging"] = BaggingClassifier(DecisionTreeClassifier())  models["Xgboost"] = XGBClassifier()  models['SVM'] = SVC()  # In[68]:  #使用十折交叉验证  results = []  num\_folds = 10  seed = 7  #使用准确度进行评估  kfold = KFold(n\_splits=num\_folds, random\_state=seed)  for key in tqdm(models):  #交叉验证每次运行的评分数组  cv\_results = cross\_val\_score(  models[key],  x\_train,  y\_train,  cv=kfold,  )  results.append(cv\_results)  print('%s::%f(%f)' % (key, cv\_results.mean(), cv\_results.std()))  # In[74]:  #正态化-箱线图  b = plt.figure(figsize=(9,5),dpi=80)  b = plt.suptitle('Scaled Algorithm Comparison')  b = plt.boxplot(results)  b = plt.xticks(range(1,len(models.keys())+1),models.keys())  b = plt.show()  # 我一共使用了\*\*逻辑回归、LDA、KNN、决策树、贝叶斯、神经网络、随机森林、Adaboost、Bagging、Xgboost和SVM\*\*共十一种模型，在全部使用默认参数（Adaboost和Bagging的使用的弱分类器为决策树）的情况下，使用十折交叉验证。最后结果显示在这十一种模型中，贝叶斯的效果最不尽如人意，\*\*Xgboost的效果最优\*\*，通过我网上提交成绩的排名显示也是Xgboost模型要优于其他分类模型。我将这个模型作为我的核心模型，使用一定的方式调整参数，使之达到最优的效果  # ### 网格搜索（牟宇、於文卓）  #  # 关于Xgboost中的网格搜索，主要参考了这篇文章[XGBoost参数调优完全指南](https://blog.csdn.net/han\_xiaoyang/article/details/52665396)  # > 如果你的预测模型表现得有些不尽如人意，那就用XGBoost吧。XGBoost算法现在已经成为很多数据工程师的重要武器。它是一种十分精致的算法，可以处理各种不规则的数据。  # 构造一个使用XGBoost的模型十分简单。但是，提高这个模型的表现就有些困难(至少我觉得十分纠结)。这个算法使用了好几个参数。所以为了提高模型的表现，参数的调整十分必要。在解决实际问题的时候，有些问题是很难回答的——你需要调整哪些参数？这些参数要调到什么值，才能达到理想的输出？  #  # 我通过网格搜索，主要调整如下几个参数  # + 学习率（learning rate）<=通过交叉验证得到  # + 决策树数量（n\_estimators）  # + 决策树相关参数（max\_depth,min\_child\_weight,gamma,subsample,colsample\_bytree）  # + 正则化参数（lambda,alpha）<=正则化参数可以降低模型复杂度  # In[77]:  from xgboost import plot\_importance  from sklearn import metrics  # #### n\_estimators  # In[84]:  def modelfit(alg, X, Y, useTrainCV=True, cv\_folds=10, early\_stopping\_rounds=50):  if useTrainCV:  xgb\_param = alg.get\_xgb\_params()  xgtrain = xgb.DMatrix(X.values, label=Y.values)  cvresult = xgb.cv(xgb\_param,  xgtrain,  num\_boost\_round=alg.get\_params()['n\_estimators'],  nfold=cv\_folds,  metrics='auc',  early\_stopping\_rounds=early\_stopping\_rounds,  verbose\_eval=False)  alg.set\_params(n\_estimators=cvresult.shape[0])  #Fit the algorithm on the data  alg.fit(X, Y, eval\_metric='auc')  #Predict training set:  dtrain\_predictions = alg.predict(X)  dtrain\_predprob = alg.predict\_proba(X)[:, 1]  #Print model report:  print("\nModel Report")  print("Best Num Of Tree:{}".format(cvresult.shape[0]))  print("Accuracy : %.4g" %  metrics.accuracy\_score(Y.values, dtrain\_predictions))  print("AUC Score (Train): %f" % metrics.roc\_auc\_score(Y, dtrain\_predprob))  feat\_imp = pd.Series(  alg.get\_booster().get\_fscore()).sort\_values(ascending=False)  b = plt.figure(figsize=(14,5),dpi=80)  feat\_imp.plot(kind='bar', title='Feature Importances')  plt.ylabel('Feature Importance Score')  # 首先我给Xgboost中的各个参数调整一个合适的默认值，其次我定义了一个函数`modelfit`，这个函数会使用xgb自带的交叉验证库调整决策树的数量，根据最后的预测结果，选择一个最优的数量，并且打印`Feature Importances`图，其表示的是数据集中各个特征的重要程度  # In[85]:  xgb1 = XGBClassifier(learning\_rate=0.1,  n\_estimators=1000,  max\_depth=5,  min\_child\_weight=1,  gamma=0,  subsample=0.8,  colsample\_bytree=0.8,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27)  # 得到Best Num Of Tree  modelfit(xgb1, x\_train, y\_train)  # 得到在学习率为0.1的情况下，最好的数量为179。通过Xgb分析，我了解到对于最后结果影响最大的几个特征为最后一次联系的交流时长和每年账户的平均余额  # #### max\_depth 和 min\_child\_weight  #  # 接下来使用网格搜索调整`max\_depth`和 `min\_child\_weight`，并打印最佳的参数。为方便展示，网格搜索的其余信息我存在了txt文件中  # In[87]:  param\_test1 = {  'max\_depth': range(3, 10, 2),  'min\_child\_weight': range(1, 6, 2)  }  gsearch1 = GridSearchCV(estimator=XGBClassifier(learning\_rate=0.1,  n\_estimators=186,  max\_depth=5,  min\_child\_weight=2,  gamma=0,  subsample=0.8,  colsample\_bytree=0.8,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27),  param\_grid=param\_test1,  scoring='roc\_auc',  n\_jobs=4,  iid=False,  cv=5)  gsearch1.fit(x\_train, y\_train)  res1 = gsearch1.cv\_results\_  res2 = gsearch1.best\_params\_  res3 = gsearch1.best\_score\_  print(res2)  with open("./max\_depth\_min\_child\_weight.txt", "w") as f:  ans = "grid\_scores\_:" + str(res1) + "\n best\_params\_:" + str(  res2) + "\n best\_score\_:" + str(res3)  f.write(ans)  # 得到最优的结果为`max\_depth`: 5，`min\_child\_weight`: 3。进一步在5和3的附近寻找最优的参数  # In[88]:  param\_test2 = {'max\_depth': [5, 6, 7, 8], 'min\_child\_weight': [2, 3, 4, 5]}  gsearch2 = GridSearchCV(estimator=XGBClassifier(learning\_rate=0.1,  n\_estimators=179,  max\_depth=5,  min\_child\_weight=3,  gamma=0,  subsample=0.8,  colsample\_bytree=0.8,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27),  param\_grid=param\_test2,  scoring='roc\_auc',  n\_jobs=4,  iid=False,  cv=5)  gsearch2.fit(x\_train, y\_train)  res1 = gsearch2.cv\_results\_  res2 = gsearch2.best\_params\_  res3 = gsearch2.best\_score\_  print(res2)  with open("./max\_depth\_min\_child\_weight.txt", "w") as f:  ans = "grid\_scores\_:" + str(res1) + "\n best\_params\_:" + str(  res2) + "\n best\_score\_:" + str(res3)  f.write(ans)  # 网格搜索结果仍然为5和3，那么就确定`max\_depth`的值为5,`min\_child\_weight`为3  # #### gamma  #  # 同理，带入上面已经确定的参数，来确定gamma的值  # In[93]:  param\_test3 = {'gamma': [i / 10.0 for i in range(0, 5)]}  gsearch3 = GridSearchCV(estimator=XGBClassifier(learning\_rate=0.1,  n\_estimators=179,  max\_depth=5,  min\_child\_weight=3,  gamma=0,  subsample=0.8,  colsample\_bytree=0.8,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27),  param\_grid=param\_test3,  scoring='roc\_auc',  n\_jobs=4,  iid=False,  cv=5)  gsearch3.fit(x\_train, y\_train)  res1 = gsearch3.cv\_results\_  res2 = gsearch3.best\_params\_  res3 = gsearch3.best\_score\_  print(res2)  with open("./gamma.txt", "w") as f:  ans = "grid\_scores\_:" + str(res1) + "\n best\_params\_:" + str(  res2) + "\n best\_score\_:" + str(res3)  f.write(ans)  # 确认`gamma`值为0.3  # #### subsample和colsample\_bytree  # In[94]:  param\_test4 = {  'subsample': [i / 10.0 for i in range(6, 10)],  'colsample\_bytree': [i / 10.0 for i in range(6, 10)]  }  gsearch4 = GridSearchCV(estimator=XGBClassifier(learning\_rate=0.1,  n\_estimators=186,  max\_depth=6,  min\_child\_weight=6,  gamma=0.3,  subsample=0.8,  colsample\_bytree=0.9,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27),  param\_grid=param\_test4,  scoring='roc\_auc',  n\_jobs=4,  iid=False,  cv=5)  gsearch4.fit(x\_train, y\_train)  res1 = gsearch4.cv\_results\_  res2 = gsearch4.best\_params\_  res3 = gsearch4.best\_score\_  print(res2)  with open("./subsample\_colsample\_bytree.txt", "w") as f:  ans = "grid\_scores\_:" + str(res1) + "\n best\_params\_:" + str(  res2) + "\n best\_score\_:" + str(res3)  f.write(ans)  # 最终确定为：`colsample\_bytree`为0.8 `subsample`为0.9  # #### Learning Rate  # 前面的参数基本都调完了就轮到学习率了，将xbg模型的参数换上刚才调好的，将学习率减小，并增加决策树的数量，重新做交叉验证，寻找一个最合适的树的数量  # In[95]:  xgb1 = XGBClassifier(learning\_rate=0.01,  n\_estimators=2000,  max\_depth=6,  min\_child\_weight=6,  gamma=0.3,  subsample=0.9,  colsample\_bytree=0.8,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27)  # 得到Best Num Of Tree  modelfit(xgb1, x\_train, y\_train)  # 最终得到的树的数量为1650  # In[96]:  xgb1 = XGBClassifier(learning\_rate=0.01,  n\_estimators=1605,  max\_depth=6,  min\_child\_weight=6,  gamma=0.3,  subsample=0.9,  colsample\_bytree=0.8,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27)  xgb1.fit(x\_train, y\_train)  xgb1.score(x\_test,y\_test)  # 至此我的xgboost模型已经调完，在测试集的预测结果为91%左右。直接可以用此模型去预测结果了，最终结果为0.93473540，但排名一般，排在280+/745  # ![排名](res\_xgboost.jpg)  # ## 於文卓 特征工程提分  #  # 300多的排名总是不让人满意，查资料的时候了解到特征工程相关的知识  # > 有这么一句话在业界广泛流传，数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已。那特征工程到底是什么呢？顾名思义，其本质是一项工程活动，目的是最大限度地从原始数据中提取特征以供算法和模型使用。  #  # > [手把手教你用sklearn做特征工程](https://blog.csdn.net/fuqiuai/article/details/79496005)  # ### 分离连续特征和离散特征  # In[117]:  train\_set = pd.read\_csv("./train\_set.csv")  train\_set = pd.DataFrame(train\_set)  test\_set = pd.read\_csv("./test\_set.csv")  test\_set = pd.DataFrame(test\_set)  data = pd.concat([train\_set,test\_set])  data.head()  # In[119]:  feature=data.columns.tolist()  feature.remove('ID')  feature.remove('y')  #离散值和连续值分开处理  sparse\_feature= ['campaign','contact','default','education','housing','job','loan','marital','month','poutcome']  dense\_feature=list(set(feature)-set(sparse\_feature))  sparse\_feature  dense\_feature  # ### 离散特征和时间一起统计  # In[120]:  def feature\_count(data, features):  feature\_name = 'count'  for i in features:  feature\_name += '\_' + i  temp = data.groupby(features).size().reset\_index().rename(columns={0: feature\_name})  data = data.merge(temp, 'left', on=features)  return data,feature\_name  # In[121]:  ll=[]  # 离散特征和时间一起统计  for f in['campaign', 'contact','default','education','housing','job','loan','marital','poutcome']:  data,\_=feature\_count(data,['day','month',f])  ll.append(\_)  data.head()  # ### 特征交叉  #  # 首先按照一个离散特征分组,计算一个离散特征和其他所有特征的关系,如果其他特征是离散特征,计算count和nunique,如果是连续特征,计算mean等  # In[122]:  def get\_new\_columns(name,aggs):  l=[]  for k in aggs.keys():  for agg in aggs[k]:  if str(type(agg))=="<class 'function'>":  l.append(name + '\_' + k + '\_' + 'other')  else:  l.append(name + '\_' + k + '\_' + agg)  return l  # In[123]:  for d in tqdm(sparse\_feature):  aggs={}  for s in sparse\_feature:  aggs[s]=['count','nunique']  for den in dense\_feature:  aggs[den]=['mean','max','min','std']  t = aggs.pop(d)  temp=data.groupby(d).agg(aggs).reset\_index()  temp.columns=[d]+get\_new\_columns(d,aggs)  data=pd.merge(data,temp,on=d,how='left')  data.head()  # ### One-Hot编码离散值  #  # In[124]:  test = data.copy()  for s in ['campaign','contact','default','education','housing','job','loan','marital','month','poutcome']:  test=pd.concat([test,pd.get\_dummies(data[s],prefix=s+'\_')],axis=1)  test.drop(s,axis=1,inplace=True)  test.head()  # 至此，原来的数据集已经从(25317, 18)扩展为(25317, 535)！  # ### 分离训练集和测试集  # In[125]:  train\_set = test[test['y'].notnull()]  test\_set = test[test['y'].isnull()]  y\_train = train\_set['y']  chosen\_columns=['age',  'balance',  'day',  'duration',  'pdays',  'previous',  'campaign\_contact\_count',  'campaign\_duration\_max',  'campaign\_duration\_std',  'campaign\_age\_max',  'campaign\_day\_std',  'campaign\_balance\_min',  'campaign\_previous\_std',  'contact\_campaign\_count',  'contact\_duration\_max',  'education\_campaign\_count',  'education\_duration\_mean',  'housing\_campaign\_count',  'job\_campaign\_count',  'job\_duration\_mean',  'job\_duration\_max',  'job\_age\_mean',  'job\_age\_max',  'job\_age\_std',  'job\_day\_mean',  'job\_day\_std',  'job\_pdays\_max',  'job\_previous\_mean',  'loan\_campaign\_count',  'marital\_campaign\_count',  'month\_campaign\_count',  'month\_campaign\_nunique',  'month\_duration\_mean',  'month\_age\_mean',  'month\_age\_max',  'month\_day\_mean',  'month\_day\_std',  'month\_pdays\_mean',  'month\_pdays\_max',  'month\_pdays\_std',  'month\_balance\_mean',  'month\_previous\_max',  'month\_previous\_std',  'poutcome\_campaign\_count',  'count\_day\_month\_campaign',  'count\_day\_month\_contact',  'count\_day\_month\_default',  'count\_day\_month\_education',  'count\_day\_month\_housing',  'count\_day\_month\_job',  'count\_day\_month\_loan',  'count\_day\_month\_marital',  'count\_day\_month\_poutcome']  x\_train = train\_set[chosen\_columns]  # ### 网格搜索调参Xgboost  #  # 最终调参结果下代码所示  # In[ ]:  xgb1 = XGBClassifier(learning\_rate=0.01,  n\_estimators=1489,  max\_depth=5,  min\_child\_weight=6,  gamma=0,  subsample=0.8,  colsample\_bytree=0.85,  objective='binary:logistic',  nthread=4,  scale\_pos\_weight=1,  seed=27)  xgb1.fit(train\_set[chosen\_columns], y\_train)  # ### 提交结果  # ![最终结果](final\_result.jpg)  # ## 总结  #  # 花了两周时间准备了这个比赛，主要的收获有：  # + 学习使用Sklearn中各个模型  # + 数据的可视化分析  # + 特征工程  # + 集成学习Xgboost  # + 网格搜索调参  #  # 比赛期间共提交比赛结果22次，下图为比赛分数走势图  # In[98]:  rank = [  0.70387718, 0.71620665, 0.91942956, 0.91160398, 0.90830749, 0.88349596,  0.92724778, 0.92724778, 0.93098124, 0.93059653, 0.93274964, 0.93473540,  0.93473540, 0.93507051, 0.88280224, 0.89704597, 0.93496533, 0.93507051,  0.93878203, 0.92839708, 0.93927500, 0.93945472  ]  rank = pd.DataFrame(rank)  sns.lineplot(data=rank) |