机器学习实验

天池大数据竞赛——重复购买预测

班级：硬件一班 姓名：于华宇 学号：171491120

## Abstract

本篇论文是关于对天池大数据竞赛中的长期赛——Repeated Buyers Prediction比赛的记录。包含赛前知识准备，竞赛具体过程及竞赛代码详解。

经过本次竞赛，我们学习到了许多有关机器学习的知识，以及宝贵的实践经验。从一开始面对数据无从下手，到逐步对数据进行处理预测的过程使我们这一学期宝贵的经验财富。尽管我们最终的结果并不是完美无缺，某些环节可能还有不求甚解的导致的纰漏，但过程中我们所学习得到的，将成为我们未来学习机器学习的基础，这不会是我们学习机器学习的终点，这只是我们的开始。

本篇论文将详细介绍大数据竞赛我们得到的宝贵的实践经验，纸上谈兵的时候很简单的事情，在实践中可能都会困难重重，我们将从安装最基础的scikit-learn包开始揭开此次竞赛的神秘面纱。内容涉及到scikit-learn包的基础知识，及一些pandas中函数的使用方法；数据预处理知识；特征工程知识；决策树，随机森林等算法的基本知识，模型评估标准。论文中还将附上我们实验的代码，仅供参考。

## Introduction

比赛简介：

商家有时会在特定日期（例如“Boxing-day”，“黑色星期五”或“双11”）进行大促销（例如折扣或现金券），以吸引大量新买家。许多吸引的买家都是一次性交易猎人，这些促销可能对销售产生很小的长期影响。为了缓解这个问题，商家必须确定谁可以转换为重复买家。通过瞄准这些潜力忠诚的客户，商家可以大大降低促销成本，提高投资回报率（ROI）。

众所周知，在线广告领域，客户定位极具挑战性，特别是对于新买家而言。但是通过Tmall.com长期积累的用户行为日志，我们或许可以解决这个问题。在这个挑战中，我们提供了一套商家及其在“双11”日促销期间获得的相应新买家。你的任务是预测对于指定商家的新买家将来是否会成为忠实客户。换句话说，您需要预测这些新买家在6个月内再次从同一商家购买商品的概率。一个包含大约20万用户的数据集用于训练，还有一个类似大小的数据集用于测试。与其他比赛类似，您可以提取任何特征，然后使用其他工具进行训练。您只需提交预测结果进行评估。

数据描述：

|  |  |
| --- | --- |
| 数据字段 | 定义 |
| 用户id | 购买者的唯一id |
| 年龄范围 | 用户的年龄范围: 小于18为1 ; [18,24]为2 ; [25,29]为3 ; [30,34]为4 ; [35,39]为5 ; [40,49]为6 ; 大于50为7和8 ; 0或空为未知。 |
| 性别 | 用户的性别: 0位女性, 1为男性, 2或空为未知。 |
| 商户id | 商人的唯一id |
| 标签 | 标签的值有 {0, 1, -1, NULL}. “1”表示“用户ID”是“商家ID”的重复购买者，而“0”则相反。' -1'表示“用户ID”不是给定商家的新客户，因此超出了我们的预测。然而，这样的记录可以提供额外的信息。NULL只出现在测试数据中，表示它是预测值的一部分。 |
| 活动日志 | 这是用户与商户之间的一组交互记录，其中每个记录都是表示为“商品id；品类id；品牌id；时间戳；动作类型”的动作。#用于分离两个相邻的元素。记录不按任何特定顺序排序。 |
| 动作类型 | 它是枚举类型{0,1,2,3}，其中0用于单击，1用于添加到购物车，2用于购买，3用于添加到收藏夹。 |

## Previous Work

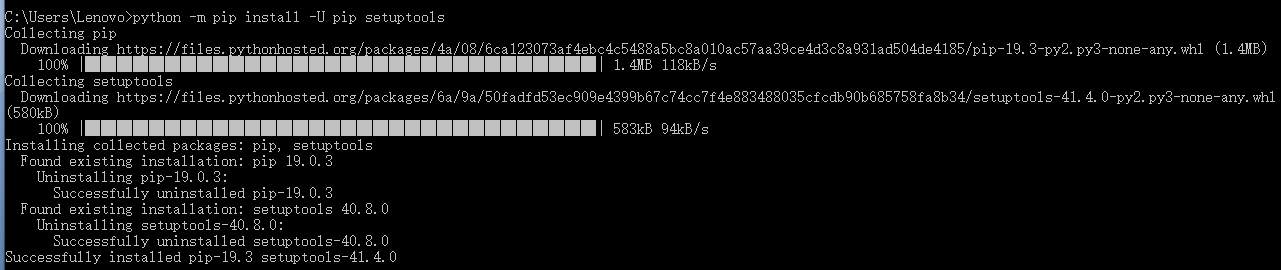
预备知识：

1.1、Scikit-learn包的安装

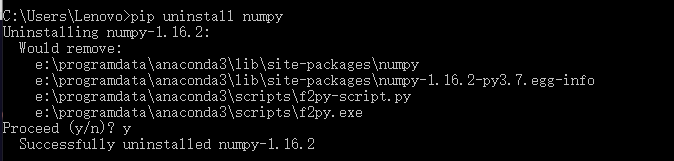
* 1. 首先，在Windows系统中查询已安装Python的版本



* 1. 升级pip为以后安装包做准备



* 1. 卸载已安装的所需要的包，避免不必要的麻烦



* 1. 下载numpy，scikit-learn,scipy,matplotlib









* 1. 将下载好的包放在E:\ProgramData\Anaconda3\Scripts。去到Python安装的scripts目录下，使用pip install xxx.whl安装。先装Numpy，Scipy，Matlotlib包，再装scikit-learn

命令如下：

e:

cd ProgramData\Anaconda3\Scripts

pip install xxx.whl

1.2、Scikit-learn的主要功能

（1）分类：识别给定对象所属类别，属于监督学习范畴，常见应用场景包括垃圾邮件检测和图像识别等。目前scikit-learn已实现算法包括：支持向量机，最近邻算法，逻辑回归，随机森林，决策树以及多层感知机，神经网络等。（由于scikit-learn本身不支持深度学习，也不支持GPU加速，因此对于多层感知机的实现不适合于处理大规模问题，若有相关需求可以查看Keras和Theano等框架）

（2）回归：预测给定对象相关联的连续值属性，常见应用场景包括预测药物反应和预测股票价格等。目前Scikit-learn已经实现的算法包括：支持向量回归，脊回归，Lasso回归，弹性网络，最小角回归，贝叶斯回归等。

（3）聚类：自动识别具有相似属性的给定对象，并将其分组为集合。属于无监督学习范畴，常见应用场景包括顾客细分和实验结果分组。目前Scikit-learn已经实现的算法包括：K-均值聚类，谱聚类，均值偏移，分层聚类，DBSCAN聚类等。

（4）数据降维：使用主成分分析、非负矩阵分解或特征选择等降维技术来减少要考虑的随机变量的个数，应用场景包括可视化处理和效率提升。

（5）模型选择：对于给定参数和模型的比较、验证和选择，其主要目的是通过调整参数来提升精度。目前Scikit-learn实现的模块包括：格点搜索，交叉验证和各种针对预测误差评估的度量函数。

（6）数据预处理：数据的特征提取和归一化，是机器学习过程中第一个也是最重要的一个环节。归一化是指将输入数据转换为具有零均值和单位权方差的新变量，但因为大多数时候都做不到精确等于零，因此会设置一个可接受的范围，一般都要求落在0-1之间。而特征提取是指将文本或图像数据转换为可用于机器学习的数字变量。

1.3、Scikit-learn使用流程示例

（1）导入所需要模型的包

例如：

from sklearn.linear\_model import LogisticRegression

（2）实例化

例如：

model = LogisticRegression()

model.fit(X, y)

（3）提取信息

例如：

predicted = model.predict(X)

score = clf.score(Xtest, Ytest)

2、数据预处理相关知识

（1）数据清洗

数据清洗主要处理的就是缺失值、异常值和重复值。清洗，就是对数据集通过丢弃、填充、替换、去重等操作，达到去除异常、纠正错误、补足缺失的目的。

数据缺失通常有两种情况：一种是数据行记录的缺失，这通常是数据记录丢失，很难找回；另一种是数据列值的缺失，可能由于某些原因，导致的数据不完整。

处理方法：

a. 丢弃

这种方法最简单明了，直接删除整行或者整列带有缺失值的记录。但是这种丢弃意味着会减少数据特征，弊端也十分明显。如下情况，不宜直接使用丢弃的方法：

数据集总体存在大量数据记录不完整情况而且比例较大，比如超过10%，此时删除这些缺失值，就意味着会损失过多的有用信息。带有缺失值的数据记录大量存在着明显的数据分布规律或特征，比如带有缺失值的数据记录是我们的目标标签主要集中在一类或者几类中，如果此时删除这些数据记录，将会使得对应类别的数据样本丢失大量数据，导致模型过拟合或者分类不准确。

b. 补全

相对于丢弃而言，补全则是一种更加合理的处理方法。通过一定的方法将缺失的数据补上，从而形成完整的数据记录。

常用的补全方法有：

统计法：对于数值型数据，可以使用均值、加权平均值、中位数等方法补全；而对于分类型数据，一般会是用类别众数最多的数值补足。

模型法：更多的时候，我们会基于其他字段，将缺失字段作为目标变量进行预测，从而达到最为可能的补全值。

专家补全：对于一些专业性非常强的数据，可以通过自学业界专家，来补全缺失数据，这种方法在很多情况下也是一种非常重要的补全途径。

其他方法：例如随机法、特殊值法，多重填补法等。

c. 真值转换法

在某些情况下，我们可能无法得知缺失值的分布规律，而且也无法应用上面的方法进行补全；或者我们认为当前的缺失记录也是数据的一种规律，不应该轻易的对缺失数据进行处理，那么此时就可以使用真值转换法来处理。

这种方法的观点是，我们承认缺失值的存在，并且把数据缺失也作为数据分布规律的一部分看待，将变量的实际值和缺失值都作为输入维度参与后续的数据处理和模型计算中。但是真实值是可以直接参与计算的，缺失值通常无法参与运算，因此需要对缺失值进行真值转换。

例如对于性别一列，现在存在男、女和未知（空值）三个状态，我们就可以把该列转换为性别\_男（0），性别\_女（1）和性别\_未知（2）。

d.不处理

对于缺失值，不做任何处理，也是一种处理缺失值的思路。这种方式主要取决于后续的模型运算，很多模型对于缺失值有容忍度或灵活的处理方法，因此在数据预处理阶段可以不作处理。

（2） 标准化数据

数据标准化是一个常用的数据预处理操作，目的是处理不同规模和量纲的数据，使其缩放到相同的数据区间和范围，以减少规模、特征、分布差异等对模型的影响。

a. Z-Score 标准化

Z-Score 标准化是基于原始数据的均值和标准差进行的标准化。这种方法适合大多数数据类型的数据，也是很多工具的默认标准化方法。标准化之后的数据是以0为均值，方差为1的正态分布。

b. Max-Min 标准化

Max-Min 标准化方法是对原始数据进行线性变换，这种标准化的方法应用广泛，得到的数据会完全落入[0，1]区间内。

（3） 数据离散化

数据离散化就是把无限空间中有限的个体映射到有限的空间中。数据离散化操作大多是针对连续数据进行的，处理之后的数据值域分布将从连续属性变为离散属性，这种属性一般包含2个或2个以上的值域。

a. 针对时间数据

针对时间数据的离散化主要用于以时间为主要特征的数据集中粒度转换，离散化处理后将分散的时间特征转换为更高层次的时间特征。

b. 针对多值离散数据

针对多值离散数据的离散化指的是要进行离散化处理的数据本身不是数值型数据，而是分类或顺序数据。

例如可以将用户的收入划分为10个区间等。

c. 针对连续数据

针对连续数据的离散化是主要的离散化应用，在分类或关联分析中应用尤其广泛。主要分为两种，一类是将连续数据划分为特定区间的集合，一类是将连续数据划分为特定类。

3、特征工程

（1）特征选择：从给定的特征集合中选出相关特征子集的过程称为特征选择

属性集中某些属性可能对于学习来说很关键，但有些属性意义就不大。对当前学习任务有用的属性或者特征，称为相关特征(relevant feature)；对当前学习任务没用的属性或者特征，称为无关特征(irrelevant feature)。

特征选择可能会降低模型的预测能力，因为被剔除的特征中可能包含了有效的信息，抛弃这部分信息一定程度上会降低模型的性能。但这也是计算复杂度和模型性能之间的取舍：如果保留尽可能多的特征，模型的性能会提升，但同时模型就变复杂，计算复杂度也同样提升；如果剔除尽可能多的特征，模型的性能会有所下降，但模型就变简单，也就降低计算复杂度。常见的特征选择分为三类方法：过滤式(filter)包裹式(wrapper)嵌入式(embedding)

（2）特征提取：特征提取一般是在特征选择之前，它提取的对象是原始数据，目的就是自动地构建新的特征，将原始数据转换为一组具有明显物理意义或者统计意义的特征。一般常用的方法包括降维（PCA、ICA、LDA等）、图像方面的SIFT、Gabor、HOG等、文本方面的词袋模型、词嵌入模型等，这里简单介绍这几种方法的一些基本概念。

（3）特征构建是指从原始数据中人工的构建新的特征。需要花时间去观察原始数据，思考问题的潜在形式和数据结构，对数据敏感性和机器学习实战经验能帮助特征构建。特征构建需要很强的洞察力和分析能力，要求我们能够从原始数据中找出一些具有物理意义的特征。假设原始数据是表格数据，一般你可以使用混合属性或者组合属性来创建新的特征，或是分解或切分原有的特征来创建新的特征。

4、模型原理

**（1）、决策树**

**决策树是一种简单高效并且具有强解释性的模型，广泛应用于数据分析领域。其本质是一颗由多个判断节点组成的树，在使用模型进行预测时，根据输入参数依次在各个判断节点进行判断游走，最后到叶子节点即为预测结果。**

1. **树模型**

 　　首先，在了解树模型之前，自然想到树模型和线性模型有什么区别呢？其中最重要的是，树形模型是一个一个特征进行处理，之前线性模型是所有特征给予权重相加得到一个新的值。决策树与逻辑回归的分类区别也在于此，逻辑回归是将所有特征变换为概率后，通过大于某一概率阈值的划分为一类，小于某一概率阈值的为另一类；而决策树是对每一个特征做一个划分。另外逻辑回归只能找到线性分割（输入特征x与logit之间是线性的，除非对x进行多维映射），而决策树可以找到非线性分割。而树形模型更加接近人的思维方式，可以产生可视化的分类规则，产生的模型具有可解释性（可以抽取规则）。树模型拟合出来的函数其实是分区间的阶梯函数。

　　其次，需要了解几个重要的基本概念：根节点（最重要的特征）；父节点与子节点是一对，先有父节点，才会有子节点；叶节点（最终标签）。

* 决策树：从根节点开始一步步走到叶子节点(决策)
* 所有的数据最终都会落到叶子节点,既可以做分类也可以做回归

1. **决策树的训练与测试**

训练阶段:从给定的训练集构造出来一棵树(从跟节点开始选择特征, 如何进行特征切分)  
测试阶段:根据构造出来的树模型从上到下去走一遍就好了

　　一旦构造好了决策树,那么分类或者预测任务就很简单了,只需要走一遍就可以了,那么难点就在于如何构造出来一颗树,这就没那么容易了,需要考虑的问题还有很多的!

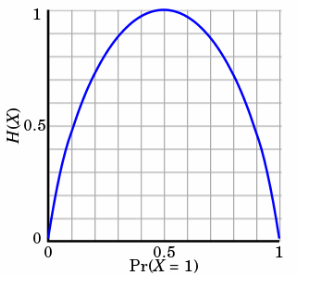
1. **如何切分特征(选择节点)**

* 问题:根节点的选择该用哪个特征呢?接下来呢?如何切分呢?
* 想象一下:我们的目标应该是根节点就像一个老大似的能更好的切分数据(分类的效果更好),根节点下面的节点自然就是二当家了。
* 目标:通过一种衡量标准,来计算通过不同特征进行分支选择后的分类情况,找出来最好的那个当成根节点,以此类推。

1. **衡量标准-熵**

　　熵:熵是表示随机变量不确定性的度量(解释:即物体内部的混乱程度)  
　　公式:H(X)=- ∑ pi \* logpi, i=1,2, ... , n  
　　一个例子:  
　　　　A集合[1,1,1,1,1,1,1,1,2,2]  
　　　　B集合[1,2,3,4,5,6,7,8,9,1]  
　　显然A集合的熵值要低,因为A里面只有两种类别,相对稳定一些而B中类别太多了,熵值就会大很多。

　　熵:不确定性越大,得到的熵值也就越大  
　　　　当p=0或p=1时,H(p)=0,随机变量完全没有不确定性  
　　　　当p=0.5时,H(p)=1,此时随机变量的不确定性最大



如何决策一个节点的选择呢?  
　　**信息增益：**表示特征X使得类Y的不确定性减少的程度。(分类后的专一性,希望分类后的结果是同类在一起)

1. **基尼不纯度**

　　基尼不纯度是指将来自集合中的某种结果随机应用于集合中某一数据项的预期误差率。如何集合中的每一个数据项都属于同一分类，那么推测的结果总会是正确的，因此误差率是 0；如果有 4 种可能的结果均匀分布在集合内，出错可能性是75%，基尼不纯度为 0.75。该值越高，说明拆分的越不理想，如果该值为 0，说明完美拆分。

1. **信息增益**

　　假设集合 U，一次拆分后变为了两个集合 u1 和 u2 ，则有：

信息增益 = E(U) - (Pu1 x E(u1) + Pu2 x E(u2))

　　E 可以是基尼不纯度或熵。  
　　使用 Pu1 和 Pu2 是为了得到拆分后两个集合基尼不纯度或熵的加权平均，其中 :

* Pu1 = Size(u1) / Size(U)
* Pu2 = Size(u2) / Size(U)

　　信息增益越大，说明整个集合从无序到有序的速度越快，本次拆分越有效。

1. **决策树的分类**

　　在建立一棵决策树的过程中，一个很重要的问题就是：怎么样将树干分叉？   
　　由此问题，便引申出了三种基本的决策树：

* ID3：利用数据集的信息增益来划分，在介绍信息增益之前先来了解一下熵的概念。   
  对于一个数据集，其熵定义如下：

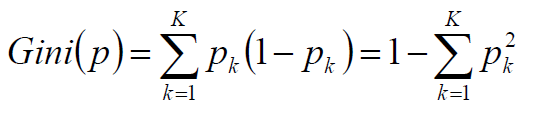
https://images2018.cnblogs.com/blog/1180120/201807/1180120-20180713140539923-859946366.png

其中p(xi)为某一类别xi的概率。   
根据香农信息理论，信息熵表示了信息的不确定度，当数据集呈均匀分布时，熵最大。   
那么，在分类之前数据集的信息熵是可以计算得到的，它和划分后的数据集的信息熵的差值即为信息增益。这里有一个划分好坏的判断标准，对于一个特定的数据集，它的信息熵是一定的，若根据某一规则将数据集划分后，数据集的信息熵越小，说明数据集的不确定度越小，即数据间的差异越小，此时信息增益越大。从而得出，信息增益越大，划分方式越好。

* C4.5: 利用数据集的信息增益比来划分数据集，从ID3进化而来，避免了分类决策时偏向于取值较多的特征的问题。
* CART树：采用基尼系数作为分类标准。基尼系数具有以下特点：

　　（1）基尼系数是一种不等性度量；

　　（2）值介于0-1之间，0表示完全相同，1表示完全不同，基尼系数越大，数据集越混乱。



1. **决策树剪枝**

* 决策树为什么(WHY)要剪枝？

　　原因是避免决策树过拟合(Overfitting)样本。前面的算法生成的决策树非常详细并且庞大，每个属性都被详细地加以考虑，决策树的树叶节点所覆盖的训练样本都是“纯”的。因此用这个决策树来对训练样本进行分类的话，你会发现对于训练样本而言，这个树表现完好，误差率极低且能够正确得对训练样本集中的样本进行分类。训练样本中的错误数据也会被决策树学习，成为决策树的部分，但是对于测试数据的表现就没有想象的那么好，或者极差，这就是所谓的过拟合(Overfitting)问题。Quinlan教授试验，在数据集中，过拟合的决策树的错误率比经过简化的决策树的错误率要高。

* 怎么剪枝

　　现在问题就在于，如何(HOW)在原生的过拟合决策树的基础上，生成简化版的决策树？可以通过剪枝的方法来简化过拟合的决策树。

　　剪枝可以分为两种：

* 预剪枝(Pre-Pruning)和后剪枝(Post-Pruning),下面我们来详细学习下这两种方法：  
  　　PrePrune：预剪枝，边建立决策树边进行剪枝的操作(更实用)，及早的停止树增长，可以通过限制深度,叶子节点个数叶子节点样本数,信息增益量等
* PostPrune：后剪枝，在已生成过拟合决策树上进行剪枝，可以得到简化版的剪枝决策树。  
  　　其实剪枝的准则是如何确定决策树的规模，可以参考的剪枝思路有以下几个：  
  　　1：使用训练集合(Training Set）和验证集合(Validation Set)，来评估剪枝方法在修剪结点上的效用  
  　　2：使用所有的训练集合进行训练，但是用统计测试来估计修剪特定结点是否会改善训练集合外的数据的评估性能，如使用Chi-Square（Quinlan，1986）测试来进一步扩展结点是否能改善整个分类数据的性能，还是仅仅改善了当前训练集合数据上的性能。  
  　　3：使用明确的标准来衡量训练样例和决策树的复杂度，当编码长度最小时，停止树增长，如MDL(Minimum Description Length)准则。

**（2）随机森林**

我们知道bagging算法是在原始的数据集上采用有放回的随机取样的方式来抽取m个子样本，从而利用这m个子样本训练m个基学习器，从而降低了模型的方差。随机森林是在bagging算法的基础之上加了一点小小的改动演化过来的。

随机森林的改动有两处，

第一：不仅随机的从原始数据集中随机的抽取m个子样本，而且在训练每个基学习器的时候，不是从所有特征中选择最优特征来进行节点的切分，而是随机的选取k个特征，从这k个特征中选择最优特征来切分节点，从而更进一步的降低了模型的方差；

第二：随机森林使用的基学习器是CART决策树。

随机森林随机选择的样本子集大小m越小模型的方差就会越小，但是偏差会越大，所以在实际应用中，我们一般会通过交叉验证的方式来调参，从而获取一个合适的样本子集的大小。

**（3）xgboost**

构建过程：

xgboost从顶到底构建树，再从底到顶反向进行剪枝。xgboost不是tree粒度的，而是在特征粒度上。决策树中最耗时的步骤是对特征的值进行排序，xgboost在训练之前，预先对数据进行了排序，保存为block结构，迭代中反复使用这个结构。在进行节点分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，各个特征的增益计算可以并行计算。

xgboost支持自定义目标函数和评估函数，只要目标函数二阶可导。xgboost借鉴了随机森林的做法，支持特征采样。

原理：

XGBoost的基础是梯度提升算法，因此我们必须先从了解梯度提升算法开始。梯度提升（Gradient boosting）是构建预测模型的强大技术之一，它是集成算法中提升法（Boosting）的代表算法。集成算法通过在数据上构建多个弱评估器，汇总所有弱评估器的建模结果，以获取比单个模型更好的回归或分类表现。弱评估器被定义为是表现至少比随机猜测更好的模型，即预测准确率不低于50%的任意模型。

xgboost是利用提升法的继承模型。提升法会逐一构建弱评估器，经过多次迭代逐渐累积多个弱评估器的方法。提升法的中著名的算法包括Adaboost和梯度提升树，XGBoost就是由梯度提升树发展而来的。梯度提升树中可以有回归树也可以有分类树，两者都以CART树算法作为主流，XGBoost背后也是CART树，这意味着XGBoost中所有的树都是二叉的。

5、模型评估标准

（1）基础知识

假正例（FalsePositive，FP）：真实类别为负例，预测类别为正例。

假负例（FalseNegative，FN）：真实类别为正例，预测类别为负例。

真正例（TruePositive，TP）：真实类别为正例，预测类别为正例。

真负例（TrueNegative，TN）：真实类别为负例，预测类别为负例。

（2）评估标准

* Accuracy=(TP+TN)/(TP+TN+FP+FN)×100%

精确度，反映了预测正确的样本占总样本数。scikit-learn中的score接口得出结果就对应着Accuracy。

* P(Precision准确率) =TP/(TP+FP)

反映预测正确的正例数占预测为正例数的多少。

* R(Recall召回率)=TP/(TP+FN)

反映预测正确的正例数占实际真实正例数的多少。

* F1=2×P×R/P+R

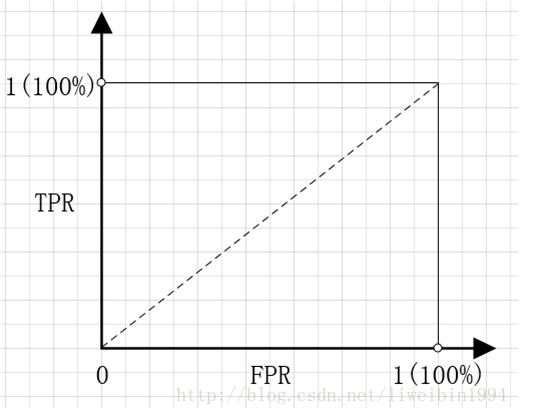
P和R的调和平均数

precision 和 recall 都是越高越好。精确度和召回率都高时，F1 值也会高． F1 值在1时达到最佳值（完美的精确度和召回率），最差为0．在二元分类中，F1 是测试准确度的量度。

* AUC

点(0,1)，即FPR=0，TPR=1。FPR=0说明FP=0，也就是说，没有假正例。所有情况都预测正确了。

点(1,0)，即FPR=1，TPR=0。这个点与上面那个点形成对比，刚好相反。所以这是最糟糕的情况。所有的预测都预测错了。

点(0,0)，即FPR=0，TPR=0。也就是FP=0，TP=0。所有样例都是负例。

点(1,1)，即FPR=1，TPR=1。所有样例都是正例。

考察完这四个点，我们可以知道，如果一个点越接近左上角，那么说明模型的预测效果越好。如果能达到左上角(点(0,1))，那就是最完美的结果了。

AUC的好处：

正负样本分布发生变化了，ROC曲线可以保持不变。

## Ourwork

1. 数据预处理

（1）缺失值处理

我们使用的是data1中的数据（方便处理特征），导入数据后，将user\_info（用户信息）表与user\_log（用户活动日志）表以user\_id属性连接，形成user\_info\_log表即处理特征的数据集。首先我们对数据中的缺失值进行了处理。由于年龄和性别中有特定的值代表缺失值，所以，我们直接用代表缺失的值填充年龄和性别中的缺失部分。brand\_id缺失的项，由于数目极小，所以直接删除了数据，没有做额外填充。

（2）one-hot编码处理

采用data1进行数据预处理时，需要将user\_info\_log中的特征属性处理好在与train和test表合并。所以先将user\_info中的年龄和性别做独热编码，在与user\_log、train、test表合并。其他可能需要处理成独热编码的特征在特种处理中进行。

* 结果：

经过数据预处理后，数据中没有缺失项，且age\_range和gender处理为one-hot编码形式。

1. 特征处理

* 用户特征

1. 特征一：用户每个月产生动作的次数

从记录中time\_stamp中提取月份，按照user\_id和month分组，求行数和，即用户每个月操作的次数总和，将月份属性采用独热编码写出，再用每月用户操作次数总和与月份相乘，即可得到用户每个月操作总数。将此特征与train和test表合并。

这项特征可以反映用户使用购物平台的次数和频率，假如该用户是这个购物平台的忠实用户，那么他会复购某家店铺商品的几率会较大。

1. 特征二：用户对相应商家进行操作的次数

以user\_id和merchant\_id进行分组，求行数和，即为用户对相应商家的总操作数。

该特征可以反映，用户对商家点击，加入购物车，收藏，购买次数和，越大则成为复购用户的可能性越高。

1. 特征三：用户查询物品总数

以user\_id分组，查看列item\_id中不重复元素的总个数，即用户查看的物品总数。

1. 特征四：用户查询种类总数

以user\_id分组，查看列cat\_id中不重复元素的总个数，即用户查看的种类总数。

1. 特征五：用户查询商家总数

以user\_id分组，查看列merchant\_id中不重复元素的总个数，即用户查看的商家总数。

1. 特征六：用户查询品牌总数

以user\_id分组，查看列brand\_id中不重复元素的总个数，即用户查看的品牌总数。

3~6反映了用户感兴趣商品，品牌，种类，商家的范围，范围越小越易复购。

1. 特征七：用户点击总数

以user\_id，action\_type分组，得出总数。并将action\_type离散化处理，再用每个action\_type分出的列与这一行对应的操作总数相乘，即可得对应动作被用户操作的总数。将action\_type=0的那一列重命名成点击次数

1. 特征八：用户加入购物车总数

以user\_id，action\_type分组，得出总数。并将action\_type离散化处理，再用每个action\_type分出的列与这一行对应的操作总数相乘，即可得对应动作被用户操作的总数。将action\_type=1的那一列重命名成加入购物车次数

1. 特征九：用户购买总数

以user\_id，action\_type分组，得出总数。并将action\_type离散化处理，再用每个action\_type分出的列与这一行对应的操作总数相乘，即可得对应动作被用户操作的总数。将action\_type=2的那一列重命名成购买次数

1. 特征十：用户加入收藏夹总数

以user\_id，action\_type分组，得出总数。并将action\_type离散化处理，再用每个action\_type分出的列与这一行对应的操作总数相乘，即可得对应动作被用户操作的总数。将action\_type=3的那一列重命名成加入收藏夹次数

7~10反映了用户进行动作数的总和，一定程度上可以反映用户进行哪些操作有可能复购。

* 商家特征

1. 特征一：商家被操作的总次数

以merchant\_id分组，查看每个商家总行数，即商家被操作次数。

该特征可以反映，商家热门程度，粉丝数多（热门）复购率也会高

1. 特征二：商家被加入购物车次数

以merchant\_id，action\_type分组，得出总数。并将action\_type离散化处理，再用每个action\_type分出的列与这一行对应的操作总数相乘，即可得对应动作商家被操作的总数。将action\_type=1的那一列重命名成商家被加入购物车次数

1. 特征三：商家被点击次数

接着上个特征，将action\_type=0的那一列重命名成商家被点击次数

1. 特征四：商家被收藏次数

接着上个特征，将action\_type=3的那一列重命名成商家被收藏次数

1. 特征六：商家被购买次数

接着上个特征，将action\_type=2的那一列重命名成商家被购买次数

2~5是商家被操作种类数目的总和，一定程度上可以预测那种动作发生后，可能会被复购

1. 特征七：商家商品总数

以merchant\_id分组，计算不重复的item\_id有多少个，即每个商家有多少商品

1. 特征八：商家商品品牌数

以merchant\_id分组，计算不重复的brand\_id有多少个，即每个商家有多少品牌

1. 特征九：商家商品种类数

以merchant\_id分组，计算不重复的cat\_id有多少个，即每个商家有多少商品种类

6~8是商家自己的属性，这些属性可能也会影响用户复购率

1. 特征十：商家每个月被操作数

以merchant\_id和month分组，求行数和，即每个商家有多少商品将月份属性采用独热编码写出，再用每月商家操作次数总和与月份相乘，即可得到商家每个月操作总数。

此项特征可以看出商家哪个月最热门，判断商家售卖商品是否是季节性的。

* 用户——商家特征

1. 特征一：用户对相应商家购买总数

以user\_id,merchant\_id,action\_type分组，每组求行数，即可得到每个用户对相应商家进行的各个操作的总数。将action\_type处理成独热编码，再用操作总数乘相应操作列可得每个用户对相应商家的各个操作总数。即可得1~4的特征。

1. 特征二：用户对相应商家点击总数
2. 特征三：用户对相应商家收藏总数
3. 特征四：用户对相应商家加入购物车总数
4. 特征五：用户对相应商家每个月操作次数

以user\_id,merchant\_id,month分组，每组求行数，即可得到每个用户对相应商家5~11月进行操作的总数。将month处理成独热编码，再用操作总数乘相应操作列，即可得结果，将结果重命名即可。

1. 特征六：用户对相应商家操作的商品数

以user\_id,merchant\_id分组，计算不重复item\_id的个数，即为所求

1. 特征七：用户对相应商家操作的品牌数

以user\_id,merchant\_id分组，计算不重复brand\_id的个数，即为所求

1. 特征八：用户对相应商家操作商品类别数

以user\_id,merchant\_id分组，计算不重复cat\_id的个数，即为所求

1. 特征九：用户对相应商家总操作次数

以user\_id和merchant\_id分组，求每组行数，即为用户对相应商家总操作次数。

* 结果：特征处理后，训练集train、测试集test中都会增加上述介绍过的特征。

1. 模型调用

* 决策树

决策树模型，会对每个特征进行判断，一直到叶节点，得出类别。在本题中，得出是否是重复购买的买家。我们调用scikit-learn中的DecisionTreeClassifier决策树中的分类树来实例化并构建模型。得出模型后，用scikit-learn中的score接口检测模型拟合程度，同时用auc来判断模型拟合效果好不好。利用剪枝参数，将决策树优化。但得出的分数总是不太好。过拟合比较严重。我认为决策树对这个模型并不十分适用。

* 随机森林

随机森林是bagging的集成模型，会生成多棵决策树，每棵决策树随机选取特征进行提问，从而分类。首先分离训练集和测试集，调用scikit-learn中的RandomForestClassifier()实例化随机森林模型，再训练训练集，用score接口返回accuracy分数。再调参，调n\_estimators观察学习曲线变化。之后画出auc图来判断模型拟合效果。分数大概为0.643左右。相较决策树，有了一定的提升。但是提升效果不是非常好，但是也不错。随机森林与决策树有一些内在联系，但随机森林的表现却比较好，集成模型的优势就凸显出来了。

* xgboost

xgboost也是一类集成模型，是boosting集成模型的代表。我们也试了一下xgboost预测模型的效果。首先要列出xgboost所需的参数，分离训练集，测试集，并多次迭代测试模型所得auc分数。得出结果来看，xgboost的效果要比随机森林好，auc可以达到0.7左右。再对模型进行交叉验证，得出结果也较为不错。用xgboost进行预测，结果分数也普遍在0.6以上。可见xgboost是比随机森林效果还要好的继承模型算法。

## Experiments

详细代码如下：

#数据预处理部分

import numpy as np

import pandas as pd

train=pd.read\_csv(r"E:\学习资料\人工智能\大数据竞赛\data\_format1\data\_format1\train\_format1.csv").drop\_duplicates()

test=pd.read\_csv(r"E:\学习资料\人工智能\大数据竞赛\data\_format1\data\_format1\test\_format1.csv")

user\_info=pd.read\_csv(r"E:\学习资料\人工智能\大数据竞赛\data\_format1\data\_format1\user\_info\_format1.csv")

user\_info.isnull().sum()

age\_range=user\_info.loc[:,"age\_range"].values.reshape(-1,1)

from sklearn.impute import SimpleImputer

imp\_0=SimpleImputer(strategy="constant",fill\_value=0)

imp\_0=imp\_0.fit\_transform(age\_range)

user\_info.loc[:,"age\_range"]=imp\_0

gender=user\_info.loc[:,"gender"].values.reshape(-1,1)

imp\_2=SimpleImputer(strategy="constant",fill\_value=2)

imp\_2=imp\_2.fit\_transform(gender)

user\_info.loc[:,"gender"]=imp\_2

user\_info=pd.get\_dummies(user\_info,columns=['age\_range'])

user\_info=pd.get\_dummies(user\_info,columns=['gender'])

user\_log=pd.read\_csv(r"E:\学习资料\人工智能\大数据竞赛\data\_format1\data\_format1\user\_log\_format1.csv").rename(columns={"seller\_id":"merchant\_id"})

#合并训练集，测试集和用户信息

train=train.merge(user\_info,on=["user\_id"])

test=test.merge(user\_info,on=["user\_id"])

user\_info\_log=user\_log.merge(user\_info,on=["user\_id"])

user\_info\_log=user\_info\_log.dropna(subset=["brand\_id"])

user\_info\_log.isnull().sum(axis=0)

###################特征处理#########################

###################用户角度#########################

#特征一：用户每个月产生动作的次数

#从记录中提取月份

user\_info\_log['month']=user\_info\_log['time\_stamp']//100

#size() 返回每个分组的元素个数

t=user\_info\_log.groupby(['user\_id','month']).size().reset\_index().rename(columns={0:'user\_month\_cnt'})

#get\_dummies类别变量向量化,pandas提供对one-hot编码的函数

t=pd.get\_dummies(t,columns=['month'])

#计算每个用户每个月操作次数

for i in range(5,12,1):

t['month\_'+str(i)]\*=t['user\_month\_cnt']

t=t.groupby(['user\_id']).sum().reset\_index()

t=t.rename(columns={'user\_month\_cnt':'user\_sum','month\_5':'user\_month\_5','month\_6':'user\_month\_6','month\_7':'user\_month\_7','month\_8':'user\_month\_8','month\_9':'user\_month\_9','month\_10':'user\_month\_10','month\_11':'user\_month\_11'})

#将特征放入训练集和测试集中

train=train.merge(t,on=['user\_id'])

test=test.merge(t,on=['user\_id'])

#用户其他特征

#1、用户对相应商家进行操作的次数

t=user\_info\_log.groupby(['user\_id','merchant\_id']).size()

t=t.reset\_index().rename(columns={0:'用户对相应商家操作总数'})

train=train.merge(t,on=["user\_id","merchant\_id"])

test=test.merge(t,on=["user\_id","merchant\_id"])

#2、用户查询物品，种类，商家，品牌总数

#agg函数，它提供基于列的聚合操作

#set() 函数创建一个无序不重复元素集，可进行关系测试，删除重复数据，还可以计算交集、差集、并集等

#lamba函数冒号之前是参数，式子本身是值

columns = ['item\_id', 'cat\_id', 'merchant\_id', 'brand\_id']

for col in columns:

t=user\_info\_log.groupby(['user\_id']).agg({col:lambda x:len(set(x))}).reset\_index().rename(columns={col:'用户查询的'+col+'个数'})

train=train.merge(t,on=['user\_id'])

test=test.merge(t,on=['user\_id'])

#3、用户点击、购买、加入购物车、收藏总数

t=user\_info\_log.groupby(['user\_id','action\_type']).size().reset\_index()

t=pd.get\_dummies(t,columns=['action\_type'])

for i in range(0,4,1):

t['action\_type\_'+str(i)]=t[0]\*t['action\_type\_'+str(i)]

t=t.groupby(['user\_id']).sum().reset\_index()

del t[0]

t=t.rename(columns={'action\_type\_0':'点击总数','action\_type\_1':'加入购物车数','action\_type\_2':'购买总数','action\_type\_3':'收藏总数'})

train=train.merge(t,on=['user\_id'])

test=test.merge(t,on=['user\_id'])

y=train.iloc[:,-3]

le=LabelEncoder()

le=le.fit(y)

le.classes\_

#数据显示用户加入购物车数只有0和1，对数据存疑

########################商家角度################################

#4、对于商家（点击总数、加入购物车数、收藏、购买数）

t=user\_info\_log.groupby(['merchant\_id','action\_type']).size().reset\_index()

t=pd.get\_dummies(t,columns=['action\_type'])

for i in range(0,4,1):

t['action\_type\_'+str(i)]=t[0]\*t['action\_type\_'+str(i)]

del t[0]

t=t.rename(columns={'action\_type\_0':'mer被点击总数','action\_type\_1':'mer被加入购物车数','action\_type\_2':'mer被购买总数','action\_type\_3':'mer被收藏总数'})

t=t.groupby(['merchant\_id']).sum().reset\_index()

train=train.merge(t,on=['merchant\_id'])

test=test.merge(t,on=['merchant\_id'])

#商家被操作的总次数

t=user\_info\_log.groupby(['merchant\_id']).size().reset\_index().rename(columns={0:'商家被操作总数'})

train=train.merge(t,on=['merchant\_id'])

test=test.merge(t,on=['merchant\_id'])

#商家商品总数

#agg函数，它提供基于列的聚合操作

#set() 函数创建一个无序不重复元素集，可进行关系测试，删除重复数据，还可以计算交集、差集、并集等

#lamba函数冒号之前是参数，式子本身是值

t=user\_info\_log.groupby(['merchant\_id']).agg({'item\_id':lambda x :len(set(x))}).reset\_index()

t=t.rename(columns={'item\_id':'商家商品总数'})

train=train.merge(t,on=['merchant\_id'])

test=test.merge(t,on=['merchant\_id'])

#商家商品品牌数

t=user\_info\_log.groupby(['merchant\_id']).agg({'brand\_id':lambda x:len(set(x))}).reset\_index().rename(columns={'brand\_id':'商家品牌总数'})

train=train.merge(t,on='merchant\_id')

test=test.merge(t,on='merchant\_id')

#商家商品种类数

t=user\_info\_log.groupby(['merchant\_id']).agg({'cat\_id':lambda x:len(set(x))}).reset\_index().rename(columns={'cat\_id':'商家商品种类数'})

train=train.merge(t,on=['merchant\_id'])

test=test.merge(t,on=['merchant\_id'])

#商家每个月被操作数

t=user\_info\_log.groupby(['merchant\_id','month']).size().reset\_index().rename(columns={0:'商家每个月被操作数'})

t=pd.get\_dummies(t,columns=['month'])

for i in range(5,12,1):

t['month\_'+str(i)]\*=t['商家每个月被操作数']

t=t.groupby(['merchant\_id']).sum().reset\_index()

del t['商家每个月被操作数']

t=t.rename(columns={'month\_5':'商家5月被操作数','month\_6':'商家6月被操作数','month\_7':'商家7月被操作数','month\_8':'商家8月被操作数','month\_9':'商家9月被操作数','month\_10':'商家10月被操作数','month\_11':'商家11月被操作数'})

train=train.merge(t,on=['merchant\_id'])

test=test.merge(t,on=['merchant\_id'])

#用户对相应商家购买,收藏，加入购物车，点击总数

t=user\_info\_log.groupby(['user\_id','merchant\_id','action\_type']).size().reset\_index()

t=pd.get\_dummies(t,columns=['action\_type'])

for i in range(0,4,1):

t['action\_type\_'+str(i)]\*=t[0]

del t[0]

t=t.groupby(['user\_id','merchant\_id']).sum().reset\_index()

t=t.rename(columns={'action\_type\_0':'用户对相应商家点击总数','action\_type\_1':'用户对相应商家加入购物车总数','action\_type\_2':'用户对相应商家购买总数','action\_type\_3':'用户对相应商家收藏总数'})

train=train.merge(t,on=['user\_id','merchant\_id'])

test=test.merge(t,on=['user\_id','merchant\_id'])

#用户对相应商家每个月操作次数

t=user\_info\_log.groupby(['user\_id','merchant\_id','month']).size().reset\_index()

t=pd.get\_dummies(t,columns=['month'])

for i in range(5,12,1):

t['month\_'+str(i)]\*=t[0]

del t[0]

t=t.groupby(['user\_id','merchant\_id']).sum().reset\_index().rename(columns={'month\_5':'用户对相应商家5月操作次数','month\_6':'用户对相应商家6月操作次数','month\_7':'用户对相应商家7月操作次数','month\_8':'用户对相应商家8月操作次数','month\_9':'用户对相应商家9月操作次数','month\_10':'用户对相应商家10月操作次数','month\_11':'用户对相应商家11月操作次数'})

train=train.merge(t,on=['user\_id','merchant\_id'])

test=test.merge(t,on=['user\_id','merchant\_id'])

#用户对相应商家商品，品牌，类别数

t=user\_info\_log.groupby(['user\_id','merchant\_id']).agg({'item\_id':lambda x:len(set(x))}).reset\_index().rename(columns={'item\_id':'user\_merchant\_item\_cnt'})

train=train.merge(t,on=['user\_id','merchant\_id'])

test=test.merge(t,on=['user\_id','merchant\_id'])

t=user\_info\_log.groupby(['user\_id','merchant\_id']).agg({'brand\_id':lambda x:len(set(x))}).reset\_index().rename(columns={'brand\_id':'user\_merchant\_brand\_cnt'})

train=train.merge(t,on=['user\_id','merchant\_id'])

test=test.merge(t,on=['user\_id','merchant\_id'])

t=user\_info\_log.groupby(['user\_id','merchant\_id']).agg({'cat\_id':lambda x:len(set(x))}).reset\_index().rename(columns={'cat\_id':'user\_merchant\_cat\_cnt'})

train=train.merge(t,on=['user\_id','merchant\_id'])

test=test.merge(t,on=['user\_id','merchant\_id'])

#决策树

from graphviz import Digraph

from sklearn import tree

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

feature\_names=list(train) #获取特征名

data=train

x=data.iloc[:,data.columns!='label']

y=data.iloc[:,data.columns=='label']

Xtrain,Xtest,Ytrain,Ytest = train\_test\_split(x,y,test\_size=0.3)

for i in [Xtrain,Xtest,Ytrain,Ytest]:#纠正随机后的索引

i.reset\_index()

clf = tree.DecisionTreeClassifier()

clf = clf.fit(Xtrain, Ytrain)

score = clf.score(Xtest, Ytest) #返回预测的准确度

score

#score结果（accuracy）

import graphviz

feature\_name=list(Xtrain)

dot\_data=tree.export\_graphviz(clf,

feature\_names=feature\_name,

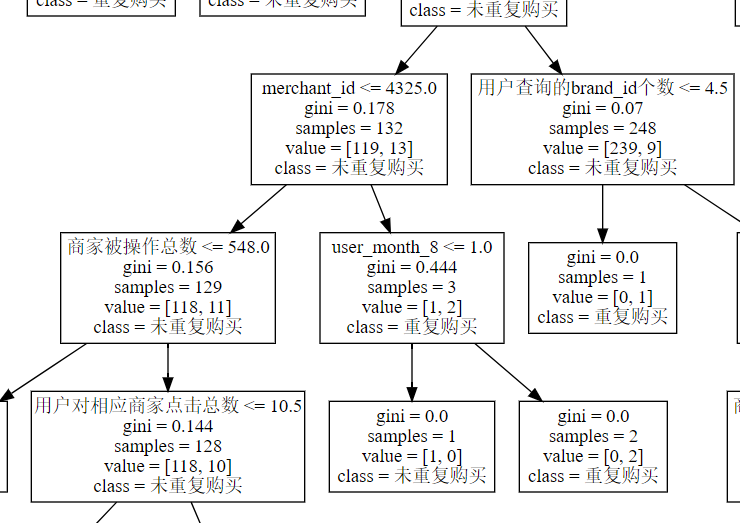
class\_names=["重复购买","未重复购买"]

,out\_file=None

)

graph=graphviz.Source(dot\_data)

#画图结果为（一部分）



#导出决策树的图

dot\_data=tree.export\_graphviz(clf,

class\_names=["repeat\_buyers","not\_repeat"]

)

graph=graphviz.Source(dot\_data)

graph.render("repeat buyers prediction")

#调参，查看最大深度变化时，会有什么变化，画出学习曲线

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.model\_selection import GridSearchCV

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

import matplotlib.pyplot as plt

tr=[]

te=[]

for i in range(10):

clf=tree.DecisionTreeClassifier(random\_state=30

,max\_depth=i+1

,splitter='random'

,min\_samples\_leaf=3

,max\_features=6

,criterion='gini'

)

clf=clf.fit(Xtrain,Ytrain)

score\_tr=clf.score(Xtrain,Ytrain)

score\_te=cross\_val\_score(clf,x,y,cv=10).mean()

tr.append(score\_tr)

te.append(score\_te)

print(max(te))

plt.plot(range(1,11),tr,color='red',label='train')

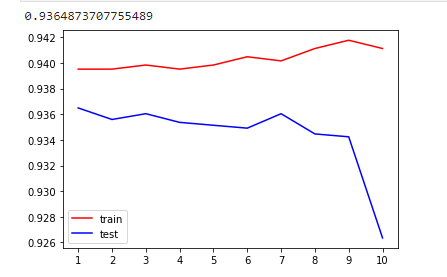
plt.plot(range(1,11),te,color='blue',label='test')

plt.xticks(range(1,11))

plt.legend()

plt.show()

#学习曲线



from sklearn.metrics import roc\_curve, auc

from sklearn import metrics

clf=DecisionTreeClassifier(random\_state=30

,max\_depth=7

,splitter='random'

,min\_samples\_leaf=4

,max\_features=6

,criterion='gini'

)

clf=clf.fit(Xtrain,Ytrain)

pred = clf.predict\_proba(Xtest)[:,1]

#############画图部分

fpr, tpr, threshold = metrics.roc\_curve(Ytest, pred)

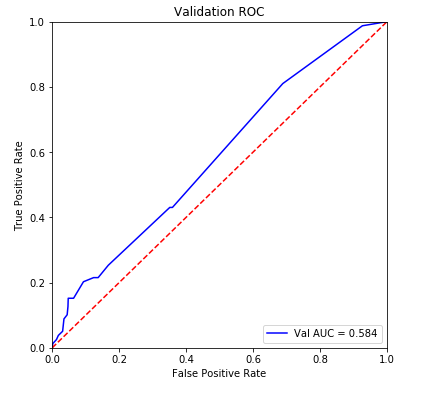
roc\_auc = metrics.auc(fpr, tpr)

plt.figure(figsize=(6,6))

plt.title('Validation ROC')

plt.plot(fpr, tpr, 'b', label = 'Val AUC = %0.3f' % roc\_auc)

plt.legend(loc = 'lower right')

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')

plt.show()

#AUC

#填写test中prob

clf=DecisionTreeClassifier( random\_state=30

,max\_depth=7

,splitter='random'

,min\_samples\_leaf=4

,max\_features=6

,criterion='gini'

)

train\_x=train.iloc[:,train.columns!='label']

train\_y=train.iloc[:,train.columns=='label']

test\_x=test.loc[:,test.columns!='prob']

test\_y=test.loc[:,test.columns=='prob']

clf=clf.fit(train\_x,train\_y)

test\_y=clf.predict\_proba(test\_x)

test\_=test

test\_y=pd.DataFrame(test\_y)

del test\_y[1]

del test\_['prob']

test\_=pd.concat([test\_,test\_y],axis=1)

test\_=test\_.rename(columns={0:'prob'})

test\_=test\_[['user\_id','merchant\_id','prob']]

test\_.to\_csv('repeat buyers.csv',index=False)

决策树提交结果：



#调用随机森林模型

#随机森林

from sklearn.ensemble import RandomForestClassifier

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

data=train

x=data.iloc[:,data.columns!='label']

t=data.iloc[:,data.columns=='label']

Xtrain, Xtest, Ytrain, Ytest = train\_test\_split(x,y,test\_size=0.3)

for i in [Xtrain,Xtest,Ytrain,Ytest]:#纠正随机后的索引

i.reset\_index()

rfc = RandomForestClassifier()

rfc = rfc.fit(Xtrain,Ytrain)

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

score\_r

#画学习曲线

superpa=[]

for i in range(190,200):

rfc=RandomForestClassifier(n\_estimators=i+1,n\_jobs=-1)

rfc\_s=cross\_val\_score(rfc,x,y.values.ravel(),cv=10).mean()

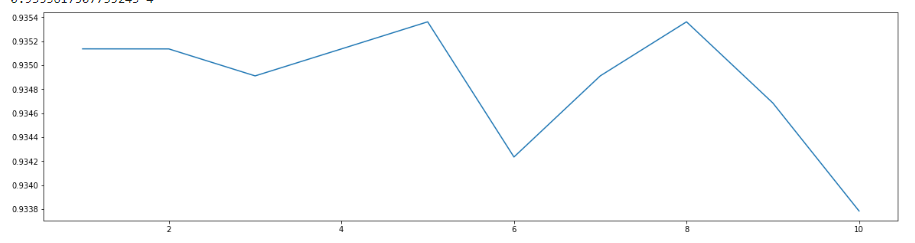
superpa.append(rfc\_s)

print(max(superpa),superpa.index(max(superpa)))

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

plt.plot(range(1,11),superpa)

plt.show()



#画auc图

from sklearn.metrics import roc\_curve, auc

from sklearn import metrics

rfc=RandomForestClassifier(n\_estimators=192

)

rfc=rfc.fit(Xtrain,Ytrain)

pred = clf.predict\_proba(Xtest)[:,1]

#############画图部分

fpr, tpr, threshold = metrics.roc\_curve(Ytest, pred)

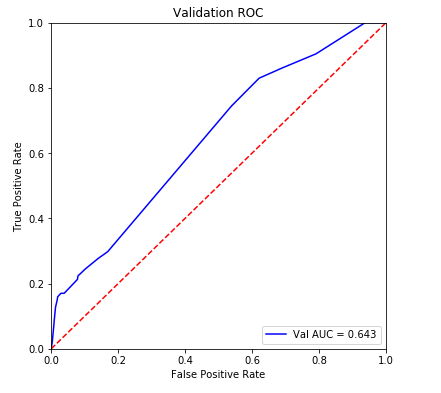
roc\_auc = metrics.auc(fpr, tpr)

plt.figure(figsize=(6,6))

plt.title('Validation ROC')

plt.plot(fpr, tpr, 'b', label = 'Val AUC = %0.3f' % roc\_auc)

plt.legend(loc = 'lower right')

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')

plt.show()

#用随机森林进行预测

rfc=RandomForestClassifier()

train\_x=train.iloc[:,train.columns!='label']

train\_y=train.iloc[:,train.columns=='label']

test\_x=test.loc[:,test.columns!='prob']

test\_y=test.loc[:,test.columns=='prob']

rfc=rfc.fit(train\_x,train\_y)

test\_y=rfc.predict\_proba(test\_x)

test\_=test

test\_y=pd.DataFrame(test\_y)

test\_=pd.concat([test\_,test\_y],axis=1)

test\_=test\_.rename(columns={0:'prob'})

test\_=test\_[['user\_id','merchant\_id','prob']]

test\_.to\_csv('repeat buyers\_random forest.csv',index=False)

#调用xgboost模型

"""

training xgboost

"""

import xgboost as xgb

models\_xgb = []

params={'booster':'gbtree',

'objective': 'binary:logistic',

'eval\_metric': 'auc',

'max\_depth':4,

'lambda':10,

'subsample':0.75,

'colsample\_bytree':0.75,

'min\_child\_weight':2,

'eta': 0.05,

'seed':0,

# 'nthread':8,

'silent':1}

for i in range(1, 6):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train\_x, train\_y, test\_size=0.33, random\_state=620 + 2 \* i)

dtrain = xgb.DMatrix(X\_train, y\_train)

deval = xgb.DMatrix(X\_test, y\_test)

print('Start training...')

# train

evals\_result={}

watchlist = [(dtrain,'train'), (deval, 'eval')]

bst=xgb.train(params, dtrain, evals\_result=evals\_result,

num\_boost\_round=10000,

evals=watchlist,

early\_stopping\_rounds=100,

verbose\_eval=1)

models\_xgb.append((i, bst, evals\_result))

#选取其中最好分数



#进行调参

import xgboost as xgb

from sklearn.model\_selection import StratifiedKFold

params={'booster':'gbtree',

'objective': 'binary:logistic',

'eval\_metric': 'auc',

'max\_depth':5,

'lambda':10,

'subsample':0.75,

'colsample\_bytree':0.75,

'min\_child\_weight':4,

'eta': 0.025,

'seed':0,

'nthread':40,

'silent':1}

model\_xgb\_base\_feat = []

skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=620)

for train\_index, test\_index in skf.split(x, y):

print()

dtrain = xgb.DMatrix(x, y)

deval = xgb.DMatrix(x, y)

print('Start training...')

watchlist = [(dtrain,'train'), (deval, 'eval')]

bst=xgb.train(params, dtrain,

num\_boost\_round=10000,

evals=watchlist,

early\_stopping\_rounds=100,

verbose\_eval=1)

model\_xgb\_base\_feat.append(bst)

#用xgboost进行预测

"""

xgboost预测

"""

dtest = xgb.DMatrix(data=test\_x)

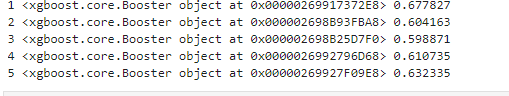
for i, model, r in models\_xgb:

col = "prob\_xgb\_" + str(i)

print(i, model, model.best\_score)

columns.append(col)

test[col] = model.predict(data=dtest, ntree\_limit=model.best\_iteration)



## Conclusion

经过试验，解决预测复购问题较好的模型是xgboost，随机森林这样的集成算法。xgboost效果会比随机森林更好。是由于xgboost采用迭代方法构造树，所以每一次构造树预测结果都会参照之前的结果，效果会更好。本次实验中数据预处理和特征工程还有可以改进，和增加的地方，若将数据处理的更好，可能会得到更好的分数和结果。

经过这一两个月的学习，在机器学习实践方面，我们累积了许多宝贵的经验。体会了从无到有处理数据得出预测结果的过程。我们感受到了数据的魅力和数据可以带给我们的价值。一些不起眼的数据，经过机器学习的整理和改变后，可以得出许多意想不到的结果，这是多么不可思议啊。以后我们还会继续在机器学习的道路上学习和探索。此次竞赛与我们而言是机器学习的开始。未来我们会继续学习更多机器学习宝贵的知识和经验。

## Reference

<https://www.leiphone.com/news/201701/ZJMTak4Y8ch3Nwd0.html>

<https://blog.csdn.net/zw0Pi8G5C1x/article/details/102693901>

<https://blog.csdn.net/weixin_34041003/article/details/87987523>

<https://www.cnblogs.com/zongfa/p/9304785.html>

https://blog.csdn.net/qq\_24519677/article/details/82117406

<https://blog.csdn.net/ae5555/article/details/73849647?utm_source=app>

菜菜sklearn课堂

七月在线机器学习训练营