

**模式识别大作业**

题 目 **房屋价格预测**

学 院 **信息科学与工程**

专 业 **控制科学与工程**

组 员 **冯述放**

指导教师 **赵海涛**

**完成日期： 2019 年 12 月 12日**

房屋价格预测

组员：冯述放

经过半个学期的对模式识别课程的学习，在赵海涛老师的辛勤指导下，我们都对模式识别与机器学习方向有了一定的了解，但是理论终究是理论，只有能够把知识熟练运用到实践当中，才能算是真正掌握了这门知识，所以赵海涛老师布置的这个大作业能够很好的锻炼我们理论运用于实际的能力。

由于本人对于编写程序方面知识的匮乏，所以需要尽量选择一个可以有一些模板框架来支持我完成作业，因此选择了房屋价格预测的比赛。

1. 房屋价格预测

要求购房者描述他们的梦想中的房子，他们可能不会从地下室天花板的高度或与东西方铁路的距离开始。但是，这场运动场比赛的数据集证明，与价格的影响相比，卧室或白色栅栏的数量影响更大。借助79个解释变量（几乎）描述了爱荷华州埃姆斯市住宅的方方面面，本次竞赛要求您预测每个房屋的最终价格。

推荐使用的算法：随即森林、Boosting

1. 整体解决方案

首先，在对数据进行处理之前，要先思考清楚题目所给的79个特征中有哪些是最能引起购房者购买此房的兴趣的。在思考这个问题时，我们可以将自己代入其中，如果我们买房，那对房屋的哪些地方要求比较高呢？1、面积；2、房屋的类型；3、房屋存在年份等等，因此接下来在处理数据时，我们就可以做一些针对性的处理，比如人为的对人们在买房子的时候比较在意的细节加大更多的权重，并且对那些在购置房屋时考虑较少甚至没有考虑的方面可以将其缩小甚至消除，这样能够提升计算机的运算速度。

其次，房屋价格预测问题可以归结为一个回归问题，对于回归问题，就可以有比较多的算法去解决，比较常见的有支持向量回归（SVR）、Light Gradient Boosting Regression等等，接下来会去介绍这些算法。

2.1 数据结构分析

由于数据结构过大，在此就不将数据贴放出来了，本次数据有79个特征，1460个训练数据，1459个测试数据，数据量还是比较大的，其中数据的特征包含：MSSubClass：建筑类；MSZoning：常规分区分类；LotFrontage：连接到物业的街道的线性英尺；LotArea：平方英尺大小；街道：道路通行类型；胡同：胡同通道的类型；LotShape：属性的一般形状；LandContour：物业的平坦度；实用程序：可用的实用程序类型；LotConfig：批次配置；LandSlope：物业的坡度；邻里：艾姆斯市区内的地理位置；条件1：接近主干道或铁路；条件2：接近主要道路或铁路（如果有）；BldgType：住宅类型；HouseStyle：住宅风格；OverallQual：总体材料和加工质量；OverallCond：总体状况的评价；YearBuilt：原始施工日期；YearRemodAdd：改型日期；RoofStyle：屋顶类型；RoofMatl：屋顶材料；Exterior1st：房屋外墙；Exterior2nd：房屋的外墙覆盖物（如果使用一种以上的材料）；MasVnrType：砖石饰面类型；MasVnrArea：砌面贴面面积（平方英尺）；ExterQual：外部材料质量；ExterCond：外部材料的当前状态；基金会：基金会的类型；BsmtQual：地下室的高度；BsmtCond：地下室的一般状况；BsmtExposure：罢工或花园水平的地下室墙壁；BsmtFinType1：地下室成品区域的质量；BsmtFinSF1：1型成品平方英尺；BsmtFinType2：第二个完成区域的质量（如果存在）；BsmtFinSF2：2型成品平方英尺；BsmtUnfSF：地下室面积未完成的平方英尺；TotalBsmtSF：地下室总平方英尺；加热方式：加热方式；加热质量控制：加热质量和条件；CentralAir：中央空调；电气：电气系统；1stFlrSF：一楼平方英尺；2ndFlrSF：二楼平方英尺；LowQualFinSF：低质量成品平方英尺（所有楼层）；GrLivArea：地面（地面）以上居住级平方英尺；BsmtFullBath：地下室全浴室；BsmtHalfBath：地下室半浴室；FullBath：满级浴室；HalfBath：地上半浴；卧室：地下室以上的卧室数量；厨房：厨房数量；KitchenQual：厨房质量；TotRmsAbvGrd：上等客房总数（不包括浴室）；功能性：家庭功能等级；壁炉：壁炉数量；FireplaceQu：壁炉质量；车库类型：车库位置；GarageYrBlt：年车库已建成；GarageFinish：车库的内部装饰；GarageCars：车库中车库的大小；GarageArea：车库的大小（平方英尺）；GarageQual：车库质量；GarageCond：车库条件；PavedDrive：铺装的车道；WoodDeckSF：木制甲板面积（平方英尺）；OpenPorchSF：开放式阳台面积（平方英尺）；封闭的门廊：封闭的门廊面积（以平方英尺为单位）；3SsnPorch：三季门廊面积（以平方英尺为单位）；ScreenPorch：屏幕门廊面积（以平方英尺为单位）；PoolArea：平方英尺的游泳池面积；PoolQC：泳池质量；围栏：围栏质量；MiscFeature：杂项功能未包括在其他类别；MiscVal：杂项功能的$ Value；MoSold：已售一个月；年销售额：已售年份；SaleType：销售类型；SaleCondition：销售条件；

可以看到，这么多的特征，如果全部用于训练数据不仅速度慢，而且由于某些特征对于结果的产生的影响不大，这也浪费了资源，因此在接下来会对某些特征进行处理，甚至对某些特征进行删除操作。

2.2 数据读入

数据分为训练数据和测试数据，训练数据为train.csv，测试数据为test.csv。利用python打开训练数据后发现，训练数据其实1460个。测试数据有1459个。

train = pd.read\_csv("train.csv") #加载训练集

test = pd.read\_csv("test.csv") #加载测试集

train\_size = train.shape[0] #训练集的规模

submission = pd.read\_csv("sample\_submission.csv") #加载测试集的结果

y\_test = submission['SalePrice']

2.3 数据处理

由于数据中含有缺失值，有些特征的值并不是数字，因此需要对数据进行处理。首先，这里需要对每个特征进行分析，因为在这79个特征中不同的特征对于不同的特征的协方差不同，因此在补缺失值时，需要根据不同的特征针对性进行插补。

print("Total Number of missing value {} before Imputation".format(sum(all\_features.isnull().sum())))

def fill\_missing\_values():

fillSaleType = all\_features[all\_features['SaleCondition'] == 'Normal']['SaleType'].mode()[0] #选择SaleCondition为normal时，SaleType列出现次数最多的值

all\_features['SaleType'].fillna(fillSaleType, inplace=True)

fillElectrical = all\_features[all\_features['Neighborhood'] == 'Timber']['Electrical'].mode()[0]

all\_features['Electrical'].fillna(fillElectrical, inplace=True)

exterior1\_neighbor = all\_features[all\_features['Exterior1st'].isnull()]['Neighborhood'].values[0] #选择Exteriorlst列为NA时，Neigbourhood列最小的值

fillExterior1 = all\_features[all\_features['Neighborhood'] == exterior1\_neighbor]['Exterior1st'].mode()[0]

all\_features['Exterior1st'].fillna(fillExterior1, inplace=True)

exterior2\_neighbor = all\_features[all\_features['Exterior2nd'].isnull()]['Neighborhood'].values[0]

fillExterior2 = all\_features[all\_features['Neighborhood'] == exterior1\_neighbor]['Exterior1st'].mode()[0]

all\_features['Exterior2nd'].fillna(fillExterior2, inplace=True)

bsmtNeigh = all\_features[all\_features['BsmtFinSF1'].isnull()]['Neighborhood'].values[0]

fillBsmtFinSf1 = all\_features[all\_features['Neighborhood'] == bsmtNeigh]['BsmtFinSF1'].mode()[0]

all\_features['BsmtFinSF1'].fillna(fillBsmtFinSf1, inplace=True)

kitchen\_grade = all\_features[all\_features['KitchenQual'].isnull()]['KitchenAbvGr'].values[0]

fillKitchenQual = all\_features[all\_features['KitchenAbvGr'] == kitchen\_grade]['KitchenQual'].mode()[0]

all\_features['KitchenQual'].fillna(fillKitchenQual, inplace=True)

all\_features['MSZoning']=all\_features.groupby('MSSubClass')['MSZoning'].transform(lambdax:x.fillna(x.mode()[0]))#利用MSSubClass对MSZoning列进行分组，随后对每组的缺失值进行赋予本组出现次数最多的值

all\_features['LotFrontage']=all\_features.groupby('Neighborhood')['LotFrontage'].transform(lambdax:x.fillna(x.median())) #利用Neighborhood对LotFrontage进行分组，随后将每组中的缺失值赋予本组的中位数值

for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'BsmtQual', 'BsmtCond', 'BsmtExposure',

'BsmtFinType1', 'BsmtFinType2', 'PoolQC']:

all\_features[col] = all\_features[col].fillna('None')

categorical\_cols = all\_features.select\_dtypes(include='object').columns

all\_features[categorical\_cols]=all\_features[categorical\_cols].fillna('None') #将所有以上未涉及的特征为非数的向量的缺失值设为None

numeric\_cols = all\_features.select\_dtypes(include='number').columns

all\_features[numeric\_cols]=all\_features[numeric\_cols].fillna(0) #将所有以上未涉及的特征为数的向量的缺失值设为0

all\_features['Shed']=np.where(all\_features['MiscFeature']=='Shed',1,0) #建立新特征Shed

# GarageYrBlt - missing values there for the building which has no Garage, imputing 0 makes huge difference with other buildings,

# imputing mean doesn't make sense since there is no Garage. So we'll drop it

all\_features.drop(['GarageYrBlt', 'MiscFeature'], inplace=True, axis=1) #将GarageYrBlt与MiscFeature列删除

all\_features['QualitySF'] = all\_features['GrLivArea'] \* all\_features['OverallQual']

fill\_missing\_values()

print("Total Number of missing value {} after Imputation".format(sum(all\_features.isnull().sum()))) #检查是否还有特征为空的数据

all\_features = all\_features.drop(['PoolQC'], axis=1) #PoolQC与PoolArea重复，删去

以上，就可以将数据的缺失值全部补好，而且补缺失值的原则就是利用与此特征关联性最大的特征对此特征进行分组，分完组之后，将对应组的缺失值填补成本组中出现次数最多的数，但如果有特殊情况时，将此分组下数据最小的赋予缺失值。接下来，就是要将特征中不是数字的特征改为数字特征，且将一些重复的特征结合起来，或者说是将本应该合在一起的特征重新整合起来，这么做完全是因为本次处理的数据是人们在购置房屋时会对房屋的特征进行评价，从而得出心目中的价格，所以很多时候，人们在购置房屋时对某些特征并不会分开去看，而是会将其合在一起，比如说住房面积，浴室数；而机器学习本身就是比较容易淡化人类的主观概念，因此我们就可以将这些特征结合起来而不会对数据的产出就太大的影响，并且可以使得计算机的运算时间减少。

all\_features['YearsSinceRemodel'] = all\_features['YrSold'].astype(int) - all\_features['YearRemodAdd'].astype(int) #出售日期-改性年份 可以表示房屋的新旧程度

all\_features['Total\_Home\_Quality'] = all\_features['OverallQual'] + all\_features['OverallCond'] #总体材料和加工质量+总体状况的评价 = 总体房屋质量

all\_features['TotalSF'] = all\_features['TotalBsmtSF'] + all\_features['1stFlrSF'] + all\_features['2ndFlrSF'] #地下室总平方英尺+一楼平方英尺+二楼平方英尺 = 房屋总面积

all\_features['YrBltAndRemod'] = all\_features['YearRemodAdd'] - all\_features['YearBuilt'] #房屋改型日期 -原始建房日期 可以表征房屋的使用情况

all\_features['BsmtFinType1\_Unf'] = 1\*(all\_features['BsmtFinType1'] == 'Unf')

all\_features['Total\_sqr\_footage']=(all\_features['BsmtFinSF1']+all\_features['BsmtFinSF2'] +all\_features['1stFlrSF']+all\_features['2ndFlrSF']) #1型成品平方英尺+2型成品平方英尺+一楼平方英尺+二楼平方英尺 = 房屋总落脚面积

all\_features['Total\_Bathrooms'] = (all\_features['FullBath'] + (0.5 \* all\_features['HalfBath']) +

all\_features['BsmtFullBath']+(0.5\*all\_features['BsmtHalfBath'])) #总浴室数目

all\_features['Total\_porch\_sf'] = (all\_features['OpenPorchSF'] + all\_features['3SsnPorch'] +

all\_features['EnclosedPorch'] + all\_features['ScreenPorch'] +

all\_features['WoodDeckSF']) #开放式阳台面积+3季门廊面积+封闭门廊面积+屏幕门廊面积+木制甲板面积 = 门廊面积总和

all\_features = all\_features.drop(['OverallQual','OverallCond','TotalBsmtSF','1stFlrSF'], axis=1)

all\_features = all\_features.drop(['2ndFlrSF','BsmtFinSF1','BsmtFinSF2','FullBath'], axis=1)

all\_features = all\_features.drop(['HalfBath', 'BsmtFullBath', 'BsmtHalfBath', 'OpenPorchSF'], axis=1)

all\_features = all\_features.drop(['3SsnPorch', 'EnclosedPorch', 'ScreenPorch', 'WoodDeckSF'], axis=1)

# Exponential features

另外，需要对不是数字形式的特征进行改编，本次改编将以1-n进行分组，同时也将数据中数值比较大的特征减小甚至消除。

# There is a natural order in their values for few categories, so converting them to numbers gives more meaning

quality\_map = {'None': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5}

quality\_cols = ['BsmtQual', 'BsmtCond', 'ExterQual', 'ExterCond', 'FireplaceQu', 'GarageQual', 'GarageCond',

'KitchenQual', 'HeatingQC']

for col in quality\_cols:

all\_features[col] = all\_features[col].replace(quality\_map)

all\_features['BsmtExposure'] = all\_features['BsmtExposure'].replace({"No": 0, "Mn": 1, "Av": 2, "Gd": 3})

all\_features["PavedDrive"] = all\_features["PavedDrive"].replace({"N": 0, "P": 1, "Y": 2})

bsmt\_ratings = {"None": 0, "Unf": 1, "LwQ": 2, "Rec": 3, "BLQ": 4, "ALQ": 5, "GLQ": 6}

bsmt\_col = ['BsmtFinType1', 'BsmtFinType2']

for col in bsmt\_col:

all\_features[col] = all\_features[col].replace(bsmt\_ratings)

all\_features["GarageScore"] = all\_features["GarageQual"] \* all\_features["GarageCond"]

all\_features["ExterScore"] = all\_features["ExterQual"] \* all\_features["ExterCond"]

all\_features = pd.get\_dummies(all\_features).reset\_index(drop=True)

2.4 评定测试

本次实验以计算 RMSE 来评判预测评分的准确度，RMSE 越小则说明评分越正确。RMSE 的计算公式如下：



本次程序中，每计算一次预测的成绩，即与真实成绩作差再平方，依次累计

直到预测完毕，最终除以总数再开根号即为所述 RMSE。

def cv\_rmse(model, X=X):

rmse=np.sqrt(-cross\_val\_score(model,X,train\_labels,scoring="neg\_mean\_squared\_error", cv=kf)) #对模型进行评分，利用模型和数据之间距离的度量的负数作为模型得分，越小越好

return (rmse)

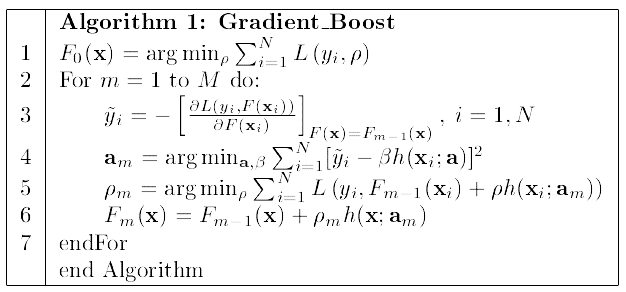
2.5 算法介绍

2.5.1 SVR

主要是通过升维后，在高维空间中构造线性决策函数来实现线性回归，用e不敏感函数时，其基础主要是 e 不敏感函数和核函数算法。

若将拟合的数学模型表达多维空间的某一曲线，则根据e 不敏感函数所得的结果，就是包括该曲线和训练点的“ e管道”。在所有样本点中，只有分布在“管壁”上的那一部分样本点决定管道的位置。这一部分训练样本称为“支持向量”。为适应训练样本集的非线性，传统的拟合方法通常是在线性方程后面加高阶项。此法诚然有效，但由此增加的可调参数未免增加了过拟合的风险。支持向量回归算法采用核函数解决这一矛盾。用核函数代替线性方程中的线性项可以使原来的线性算法“非线性化”，即能做非线性回归。与此同时，引进核函数达到了“升维”的目的，而增加的可调参数是过拟合依然能控制。

2.5.2 GBR(Gradient Boosting Regression)



2.5.3 XGBoost Regression

XGBoost算法是集成学习算法的一种，并且运用比较广泛，本质上，对误差函数进行二阶泰勒近似展开，实质上就是这个算法的作者自己定义了误差函数，算是一种很好的突破与尝试。

总结起来，XGBoost做了以下改进：

1、目标函数通过二阶泰勒展开式做近似

2、定义了树的复杂度，并应用到目标函数中

3、分裂结点处通过结构打分和分割损失动态生长

4、分裂结点的候选集合通过一种分布式Quantile Sketch得到

5、可以处理稀疏、缺失数据

6、可以通过特征的列采样防止过拟合

XGBoost的优势是：

1.传统GBDT以CART作为基分类器，xgboost还支持线性分类器，这个时候xgboost相当于带L1和L2正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。 —可以通过booster [default=gbtree]设置参数:gbtree: tree-based models/gblinear: linear models

2.传统GBDT在优化时只用到一阶导数信息，xgboost则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。顺便提一下，xgboost工具支持自定义代价函数，只要函数可一阶和二阶求导。 —对损失函数做了改进（泰勒展开，一阶信息g和二阶信息h,上一章节有做介绍）

3.xgboost在代价函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的score的L2模的平方和。从Bias-variance tradeoff角度来讲，正则项降低了模型variance，使学习出来的模型更加简单，防止过拟合，这也是xgboost优于传统GBDT的一个特性

1. shrinkage and column subsampling —还是为了防止过拟合
2. split finding algorithms(划分点查找算法)：—理解的还不够透彻，需要进一步学习

（1）exact greedy algorithm—贪心算法获取最优切分点

（2）approximate algorithm— 近似算法，提出了候选分割点概念，先通过直方图算法获得候选分割点的分布情况，然后根据候选分割点将连续的特征信息映射到不同的buckets中，并统计汇总信息。

（3）Weighted Quantile Sketch—分布式加权直方图算法。

2.5.4 LightGBM（Light Gradient Boosting Machine）

LightGBM（Light Gradient Boosting Machine）是微软开源的一个实现GBDT算法的框架，支持高效率的并行训练。

GBDT (Gradient Boosting Decision Tree)是机器学习中一个长盛不衰的模型，其主要思想是利用弱分类器（决策树）迭代训练以得到最优模型，该模型具有训练效果好、不易过拟合等优点。GBDT在工业界应用广泛，通常被用于点击率预测，搜索排序等任务。GBDT也是各种数据挖掘竞赛的致命武器，据统计Kaggle上的比赛有一半以上的冠军方案都是基于GBDT。

LightGBM提出的主要原因是为了解决GBDT在海量数据遇到的问题，让GBDT可以更好更快地用于工业实践。其具有以下优点：

更快的训练速度；更低的内存消耗；更好的准确率；分布式支持，可以快速处理海量数据。

2.6 训练模型

对模型的训练采用的是集成方法，方法是采用上述几种算法分别训练模型，随后将这些模型加权结合，加权参数即是我们在训练模型是可以调控的参数，并且根据不同的算法训练出不同的模型，随后根据其RMSE分数得出不同算法的分数，并且这个分数越低，那么效果越好。

得出结果，可以看出不同算法对于此模型的不同得分：

score = cv\_rmse(lightgbm)

lgb\_model\_full\_data = lightgbm.fit(X, train\_labels)

print("lightgbm: {:.4f}".format(score.mean()))

model\_score['lgb'] = score.mean()

score = cv\_rmse(xgboost)

xgb\_model\_full\_data = xgboost.fit(X, train\_labels)

print("xgboost: {:.4f})".format(score.mean()))

model\_score['xgb'] = score.mean()

score = cv\_rmse(svr)

svr\_model\_full\_data = svr.fit(X, train\_labels)

print("SVR: {:.4f} ({:.4f})".format(score.mean(), score.std()))

model\_score['svr'] = score.mean()

score = cv\_rmse(gbr)

gbr\_model\_full\_data = gbr.fit(X, train\_labels)

print("gbr: {:.4f}".format(score.mean()))

model\_score['gbr'] = score.mean()







def blended\_predictions(X,weight):

return ( (weight[0] \* svr\_model\_full\_data.predict(X)) + \

(weight[1] \* gbr\_model\_full\_data.predict(X)) + \

(weight[2] \* xgb\_model\_full\_data.predict(X)) + \

(weight[3] \* lgb\_model\_full\_data.predict(X)))

# Get final precitions from the blended model

blended\_score = rmsle(train\_labels, blended\_predictions(X,[0.15,0.2,0.1,0.15,0.1,0.3]))

print("blended score: {:.4f}".format(blended\_score))

model\_score['blended\_model'] = blended\_score



可以看出，训练结果是比较不错的，因此本次算法还算成功。

三、总结

由于我是第一次接触大数据的处理，所以在刚开始做这个作业时会比较迷茫，但是经过深思考虑后确定了一个整体的方向，然后朝着这个方向不断的去努力，最后结果虽然不是十分的理想，可能是由于算法比较简单，不能很全面的考虑到所有的因素。

由于也是第一次接触大数据的处理，因此很多数据的处理可能比较粗糙，并没有完全按照真实的协方差进行处理，更多时候是一些主观的印象，因此虽然最后得到的效果不错，但总体而言本次由于是第一次接触大数据的处理，因此很多事物做的并不好。希望以后能够继续努力。

这次大作业让我直接接触了真正的用程序语言来解决生活中的问题，接触了大数据，虽然只是进行了简单的尝试，但是也能够让我学到许多经验，并且培养了我对机器学习这一方向的兴趣。感谢赵老师在这一学期的非常认真的教学，并且在作业过程中给了我大家很多指导。