

**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 傅昊

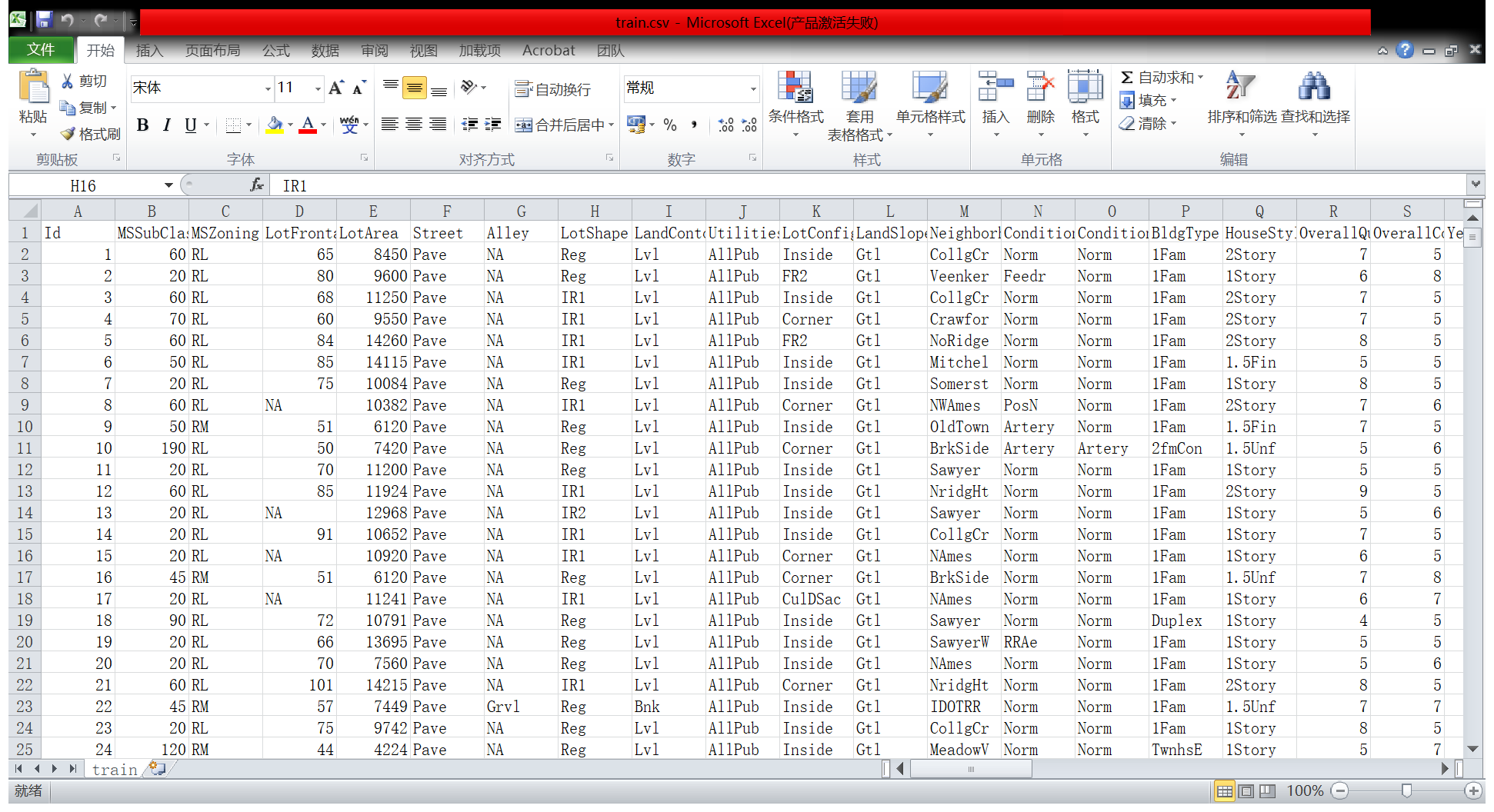
指导教师 赵海涛

**完成日期： 2019 年 12 月10日**

**模式识别作业报告——房屋价格预测**

一.项目介绍

House Prices数据集分为train（即训练）数据和test（即测试）数据，其中，训练集含有1460个样本，80个属性（包括序号），一个标签（SalePrice，即房价）；测试集含有1459个样本，80个属性。

需要做的工作：根据测试集的属性预测每个样本的房价。

二.数据处理

2.1 数据属性分析

对除编号外的79个属性进行分析，可以大致分为以下几类：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 标签 | | | | | | | | | | SalePrice 房价 | | | | | | | | | | | | | | | | | |
| 房子地理位置 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| MSSubClass建筑类 | | | LandContour  地产的扁平化 | | | | | | | LotShape  地产的外形 | | | | LotArea  方块大小 | | | | LotConfig  地产配置 | | | | | Alley  巷类型 | | | | |
| MSZoning  城市总体规划分区 | | | LotFrontage  连接物业的街道线 | | | | | | | Utilities  地产的公用事业类型 | | | | Street  道路入口类型 | | | | LandSlope地产的  坡度 | | | | | Neighborhood城市范围内的物理位置 | | | | |
| Condition1接近主干道或铁路 | | | | | | | | | | | | | | | Condition2接近主路或铁路 | | | | | | | | | | | | |
| 房屋建造风格 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| BldgType  住宅类型 | HouseStyle  居家风格 | | | | | | | | OverallQual  整体质量和表面质量 | | | | | | | | | | | OverallCond  总体状态额定值 | | | | | | | |
| 房屋装修详情 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| YearBuilt  原施工日期 | | | | YearRemodAdd 重塑日期 | | | | | | | RoofStyle 屋顶类型 | | RoofMatl  屋顶材料 | | | | | | | | Exterior1st 房屋外墙 | | | | | ExterQual外观材质 | |
| MasVnrType 圬工单板型 | | | | MasVnrArea 砌体覆盖面积 | | | | | | | | | ExterCond  外墙现状 | | | | | | | | Exterior2nd  房屋外部覆盖物 | | | | | | |
| 地下室 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Foundation 地基类型 | | | | BsmtQual  地下室的高度 | | | | | BsmtCond 地下室概况 | | | | BsmtFinType1  地下室竣工面积质量 | | | | | | | | | | | BsmtFinSF1 1型成品面积 | | | |
| BsmtFinType2 第二成品区域的质量 | | | | | BsmtFinSF2  2型成品面积 | | | | BsmtUnfSF 地下室面积 | | | | TotalBsmtSF 地下室面积总计面积 | | | | | | | | BsmtExposure:  走道或花园式地下室墙 | | | | | | |
| 冷暖气 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Heating  暖气方式 | | | | | CentralAir 空调 | | | | Electrical  电气系统 | | | | HeatingQC 暖气质量与条件 | | | | | | | | | | | | | | |
| 居住面积 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1stFlrSF  一楼面积 | | | | | 2ndFlrSF  二楼面积 | | | | GrLivArea  高档居住面积 | | | | | | | LowQualFinSF  低质量完工面积（所有楼层） | | | | | | | | | | | |
| 功能房间 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Functional 家庭功能  评级 | | KitchenQual 厨房品质 | | | | | FullBath  高档浴室 | | | | | HalfBath  半人以上  洗澡浴室 | | | | | Bedroom  地下室层以上的卧室数 | | | | | | | | Kitchen  厨房数量 | | |
| Fireplaces  壁炉数 | | FireplaceQu  壁炉质量 | | | | | PoolQC  泳池质量 | | | | | Fence  围栏质量 | | | | | MiscFeature  杂项特征 | | | | | | | | MiscVal  杂项价值 | | |
| BsmtFul lBath  地下室全浴室 | | | | | | | TotRmsAbvGrd  总房间（不包括浴室） | | | | | | | | | | BsmtHalfBath  地下室半浴室 | | | | | | | | | | |
| 车库 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| GarageType车库位置 | | GarageYrBlt车库建成年 | | | | | PavedDrive铺好的车道 | | | | | GarageCond  车库状况 | | | | | GarageArea  车库面积 | | | | | | | | GarageQual  车库质量 | | |
| GarageFinish车库的内饰 | | | | | | | | | | | | GarageCars车库容量大小 | | | | | | | | | | | | | | | |
| 其他面积 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| WoodDeckSF  木制甲板面积 | | | | | OpenPorchSF  外部面积 | | | EnclosedPorch  闭走廊面积 | | | | | | | 3SsnPorch  三季面积 | | | | ScreenPorch  屏风面积 | | | | | | | | PoolArea  泳池面积 |
| 销售 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| MoSold  月出售 | | | | | | YrSold  年出售 | | | | | | SaleType  销售类型 | | | | | | | | | | SaleCondition  销售条件 | | | | | |

2.2 格式转换

首先读取训练数据和测试数据。

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

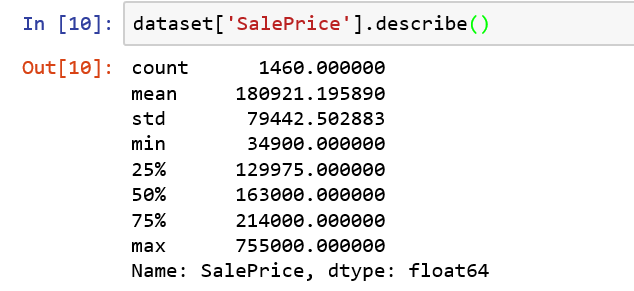
*%matplotlib inline*

*df\_train=pd.read\_csv('train.csv') #训练数据*

*df\_test=pd.read\_csv('test.csv') #测试数据*

之后分析要预测的标签数据SalePrice

*dataset['SalePrice'].describe()*



通过上图可以知道SalcePrice的基本数据特征，包括总计数值、平均值、最大最小值等。可以看出在训练数据中不存在无效或者其它非数据类型的数据。

2.3 缺失值处理

训练数据和预测数据中均有大量的缺失值，将训练数据和预测数据合并到一起，统计每列的缺失值数量。

*y\_train=df\_train.pop('SalePrice')*

*#删除并返回数据集中SalePrice标签列*

*all\_df=pd.concat((df\_train,df\_test),axis=0) #要处理的整体数据集*

*total=all\_df.isnull().sum().sort\_values(ascending=False) #每列缺失数量*

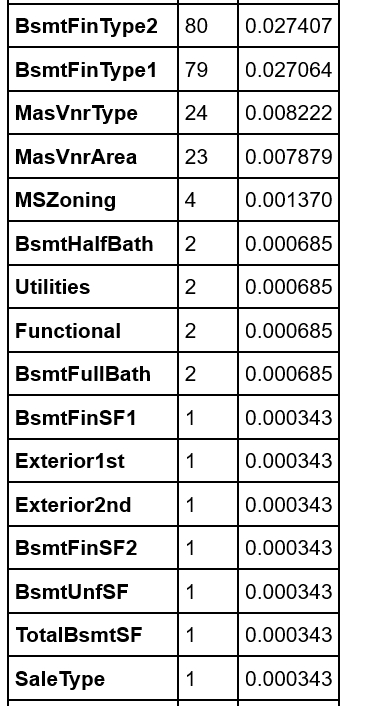
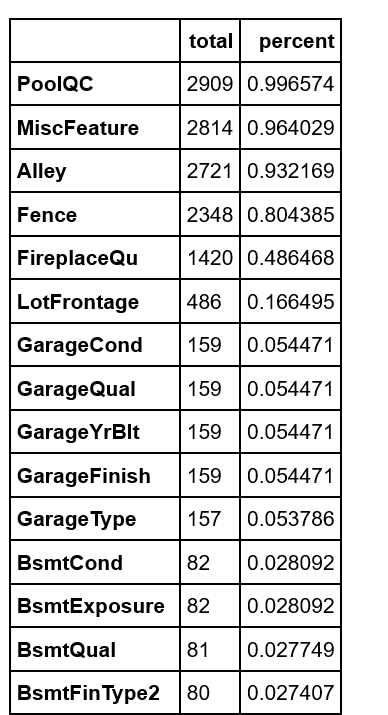
*percent=(all\_df.isnull().sum()/len(all\_df)).sort\_values(ascending=False)*

*#每列缺失率*

*miss\_data=pd.concat([total,percent],axis=1,keys=['total','percent'])*

*miss\_data #显示每个列及其对应的缺失率*

所得的结果如下：



从上图中看出，“PoolQC”，“MiscFeature”，“Alley”，“Fence”，“FireplaceQu”这五个参数的缺失值过高，可以直接删除。

*all\_df.drop(['MiscFeature','Alley','Fence','FireplaceQu'],axis=1)*

车库类的四项缺失值相同，都指的是没有车库的情况。将缺失值定义为“missing”缺失。

*garage\_obj=['GarageType','GarageFinish','GarageQual','GarageCond']*

*#列出车库这一类*

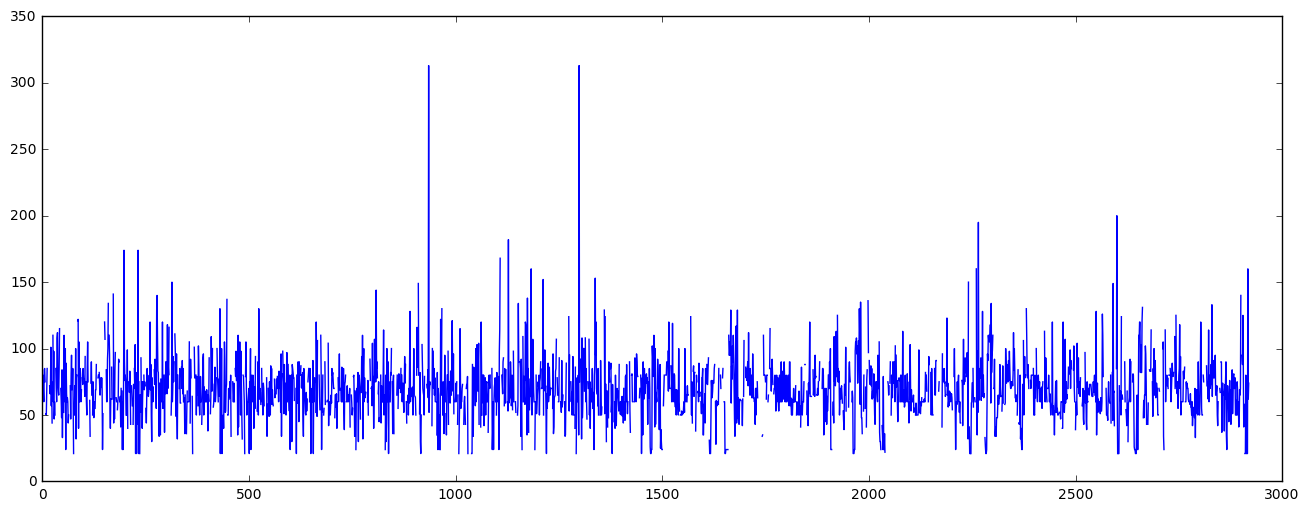
*for garage in garage\_obj:*

*all\_df[garage].fillna('missing',inplace=True)*

*#把1900标签填入空缺处表示年代久远*

*all\_df['GarageYrBlt'].fillna(1900.,inplace=True)*

然后表中缺失值较高的是LotFrontage，查看LotFrontage的分布情况，发现其大致是均匀分布。因此用平均值将缺失值补齐。



*all\_df['LotFrontage'].fillna(all\_df['LotFrontage'].mean(),inplace=True)*

还有部分少量的缺失值，不是很重要，可以用one-hotd转变离散值，然后均值补齐。

*all\_dummies\_df=pd.get\_dummies(all\_df)*

*mean\_col=all\_dummies\_df.mean()*

*all\_dummies\_df.fillna(mean\_col,inplace=True)*

*a=all\_dummies\_df.columns[all\_dummies\_df.dtypes=='int64'] #数值为int型*

*b=all\_dummies\_df.columns[all\_dummies\_df.dtypes=='float64'] #数值为float型*

*#进行标准化处理，符合0-1分布*

*a\_mean=all\_dummies\_df.loc[:,a].mean()*

*a\_std=all\_dummies\_df.loc[:,a].std()*

*all\_dummies\_df.loc[:,a]=(all\_dummies\_df.loc[:,a]-a\_mean)/a\_std*

*b\_mean=all\_dummies\_df.loc[:,b].mean()#使数值型为int的所有列标准化*

*b\_std=all\_dummies\_df.loc[:,b].std()*

*all\_dummies\_df.loc[:,b]=(all\_dummies\_df.loc[:,b]-b\_mean)/b\_std*

*#使数值型为float的所有列标准化*

*#处理后的训练集*

*df\_train1=all\_dummies\_df.iloc[:1460,:]*

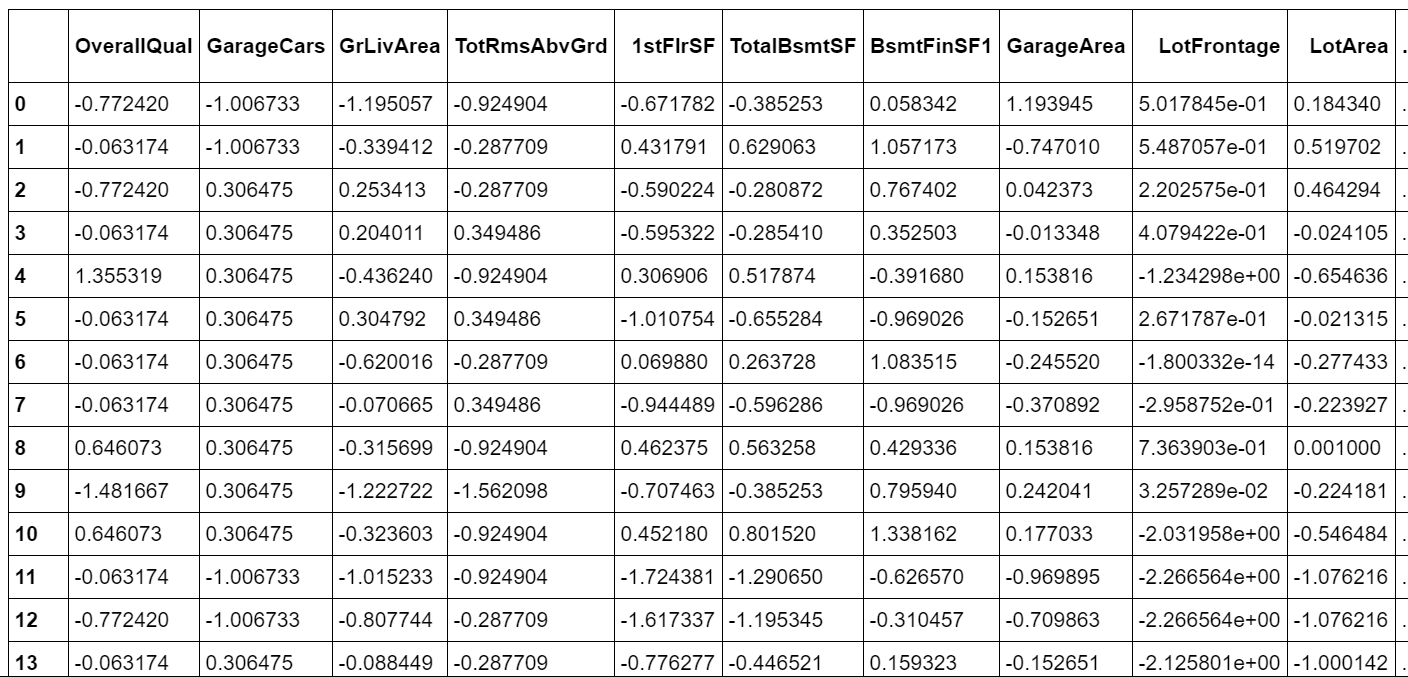
*x\_train =df\_train1.iloc[0:int(0.8\*len(df\_train1)),:] #train中的训练集*

*x\_test=df\_train1.iloc[int(0.8\*len(df\_train1)):,:] #train中的测试集*

*y\_train =y\_train.iloc[0:int(0.8\*len(y\_train))] #train中训练集的lable*

*y\_test = y\_train.iloc[int(0.8\*len(df\_train1)):] #train中测试集的lable*

至此，训练数据均已预处理完毕，所有数据为没有缺失值且在[0,1]区间的标准化数据。大致如下图所示。



三.实验过程

3.1 模型对比

本次实验挑选了四种树模型进行对比训练，分别是。DecisionTreeRegressor、RandomForestRegressor，AdaBoostRegressor GradientBoostingRegressor。

*from sklearn.tree import DecisionTreeRegressor*

*from sklearn.ensemble import RandomForestRegressor*

*from sklearn.ensemble import ExtraTreesRegressor*

*from sklearn.ensemble import AdaBoostRegressor*

*from sklearn.ensemble import GradientBoostingRegressor*

*cv\_params = {'n\_estimators': [550, 575, 600, 650, 675]}*

*other\_params = {'learning\_rate': 0.1, 'n\_estimators': 600, 'max\_depth': 5, 'min\_child\_weight': 1, 'seed': 0, 'subsample': 0.8, 'colsample\_bytree': 0.8, 'gamma': 0, 'reg\_alpha': 0, 'reg\_lambda': 1}*

*clfs = [DecisionTreeRegressor(),*

*RandomForestRegressor(),*

*AdaBoostRegressor(),*

*GradientBoostingRegressor() ]*

将处理好的数据放入模型中训练，得出训练集中测试样本（验证集）的预测值，得到模型在验证集上的准确度，并提取出权重占比大的特征。

def plot\_feature\_importances(clf, X\_train, y\_train, X\_test, y\_test, top\_n=10,

figsize=(12,12), print\_table=False, title="Feature Importances", logger = None):

\_\_name\_\_ = "plot\_feature\_importances"

try:

if not hasattr(clf, 'feature\_importances\_'):

clf.fit(X\_train.values, y\_train.values.ravel())

if not hasattr(clf, 'feature\_importances\_'):

raise AttributeError("{} does not have feature\_importances\_ attribute".format(clf.\_\_class\_\_.\_\_name\_\_))

model\_name = clf.\_\_class\_\_.\_\_name\_\_

# score

logger.info("Model name: \n {}".format(clf.\_\_class\_\_.\_\_name\_\_))

score = clf.score(X\_test, y\_test)

y\_pred = clf.predict(x\_test)

logger.info("Model score: \n {}".format(score))

model\_path = pjoin(MODEL\_CP\_DIR, clf.\_\_class\_\_.\_\_name\_\_ + '-'+str(score)+ '.pkl')

logger.info("Model path: \n {}".format(model\_path))

joblib.dump(clf, model\_path)

# Feature Importance

feat\_imp = pd.DataFrame({'Importance':clf.feature\_importances\_})

feat\_imp['Feature'] = X\_train.columns

feat\_imp.sort\_values(by='Importance', ascending=False, inplace=True)

feat\_imp = feat\_imp.iloc[:top\_n]

feat\_imp.sort\_values(by='Importance', inplace=True)

feat\_imp = feat\_imp.set\_index('Feature', drop=True)

feat\_imp.plot.barh(title=title, figsize=figsize)

plt.xlabel('Feature Importance Score')

plt.savefig(pjoin(MODEL\_CP\_DIR, clf.\_\_class\_\_.\_\_name\_\_ + '-'+str(score)+"-Feature Importance.png"))

plt.show()

logger.info("Feature Importance Score: \n {} \n\n".format(feat\_imp))

if print\_table:

from IPython.display import display

print("Top {} features in descending order of importance".format(top\_n))

display(feat\_imp.sort\_values(by='Importance', ascending=False))

return feat\_imp, score

best\_score = 0

best\_model = None

for clf in clfs:

try:

feat\_imp, score = plot\_feature\_importances(

clf, x\_train, y\_train, x\_test, y\_test,

top\_n=x\_train.shape[1], title=clf.\_\_class\_\_.\_\_name\_\_, logger=logger)

if best\_score < score:

best\_score = score

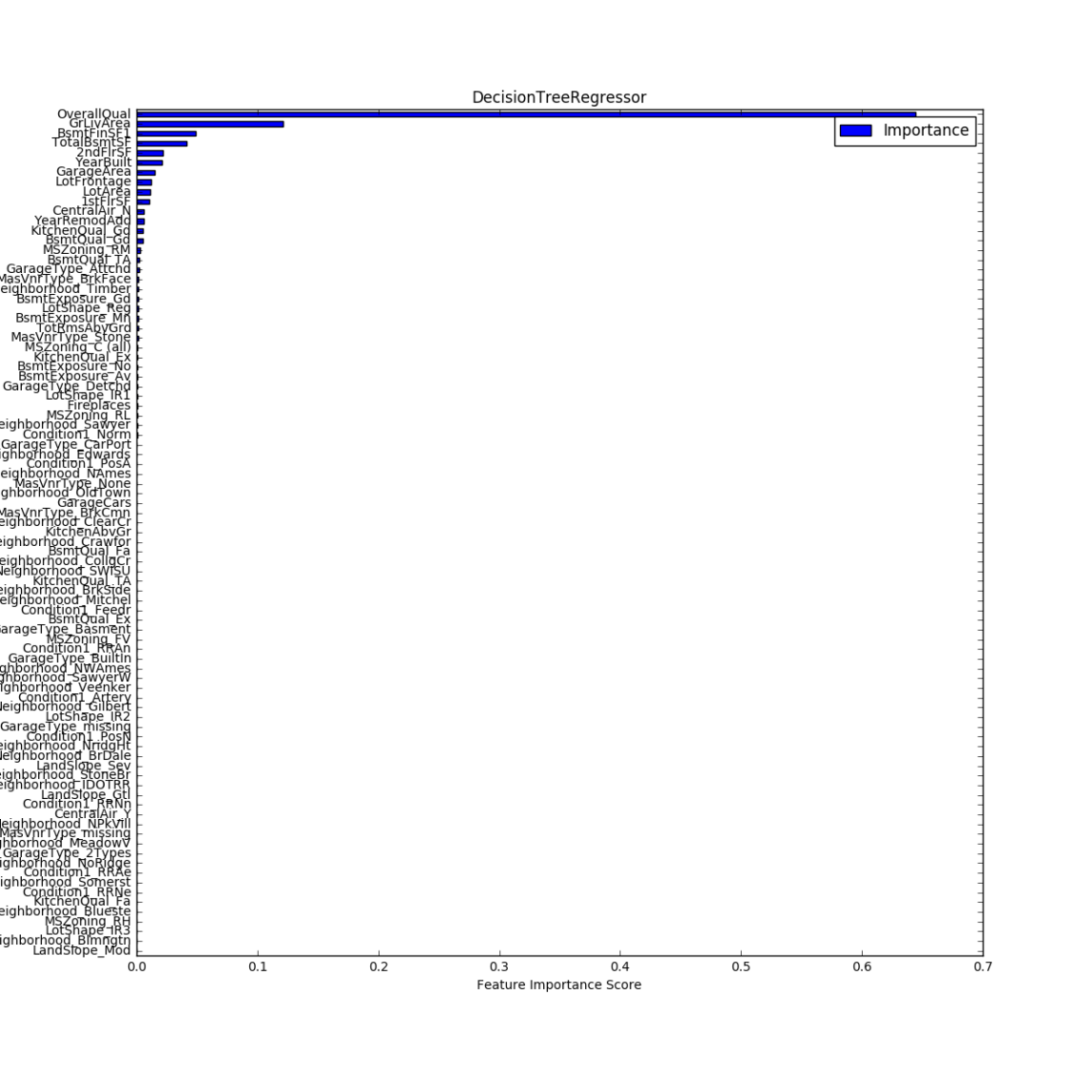
best\_model = clf.\_\_class\_\_.\_\_name\_\_

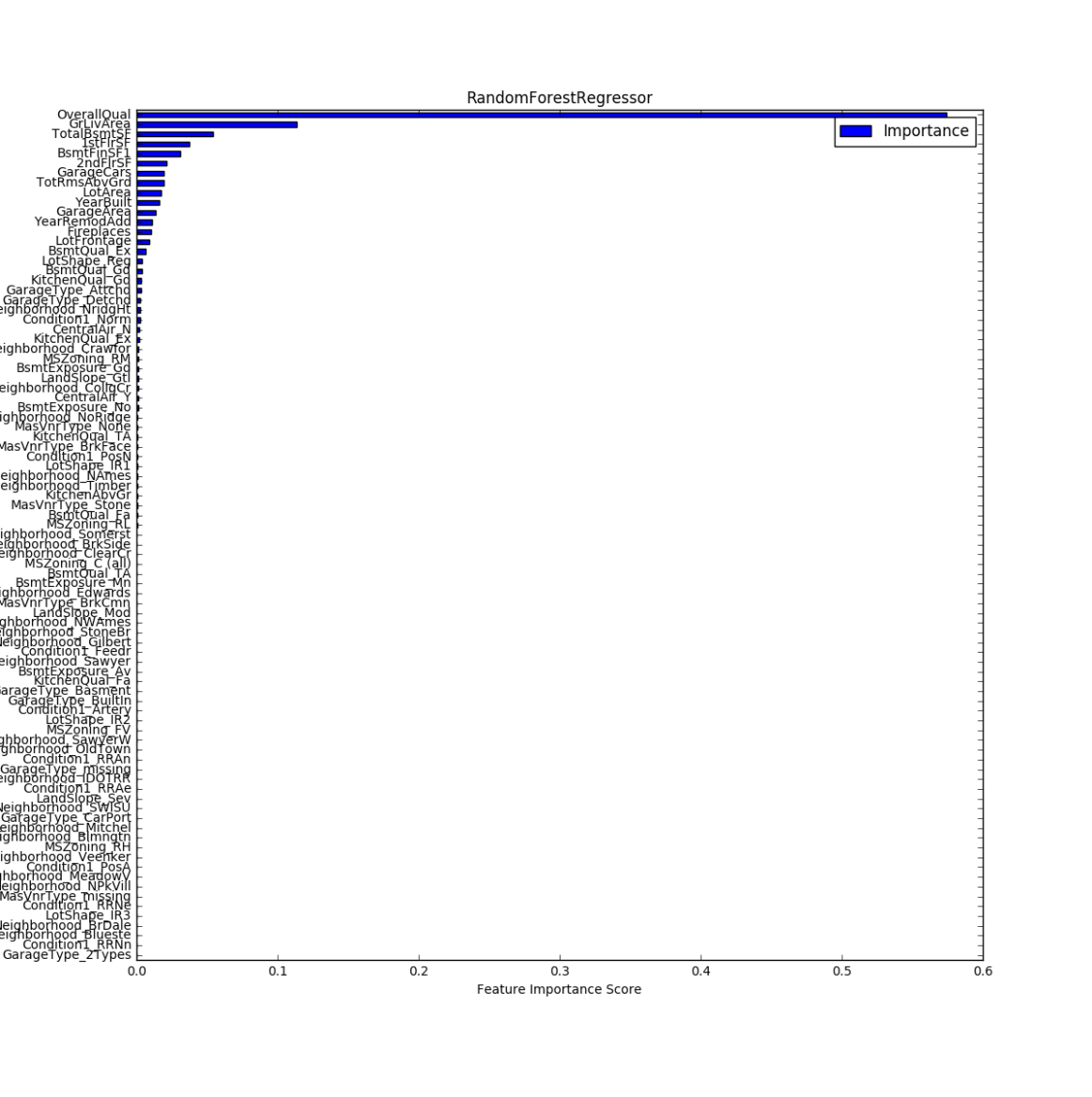
except AttributeError as e:

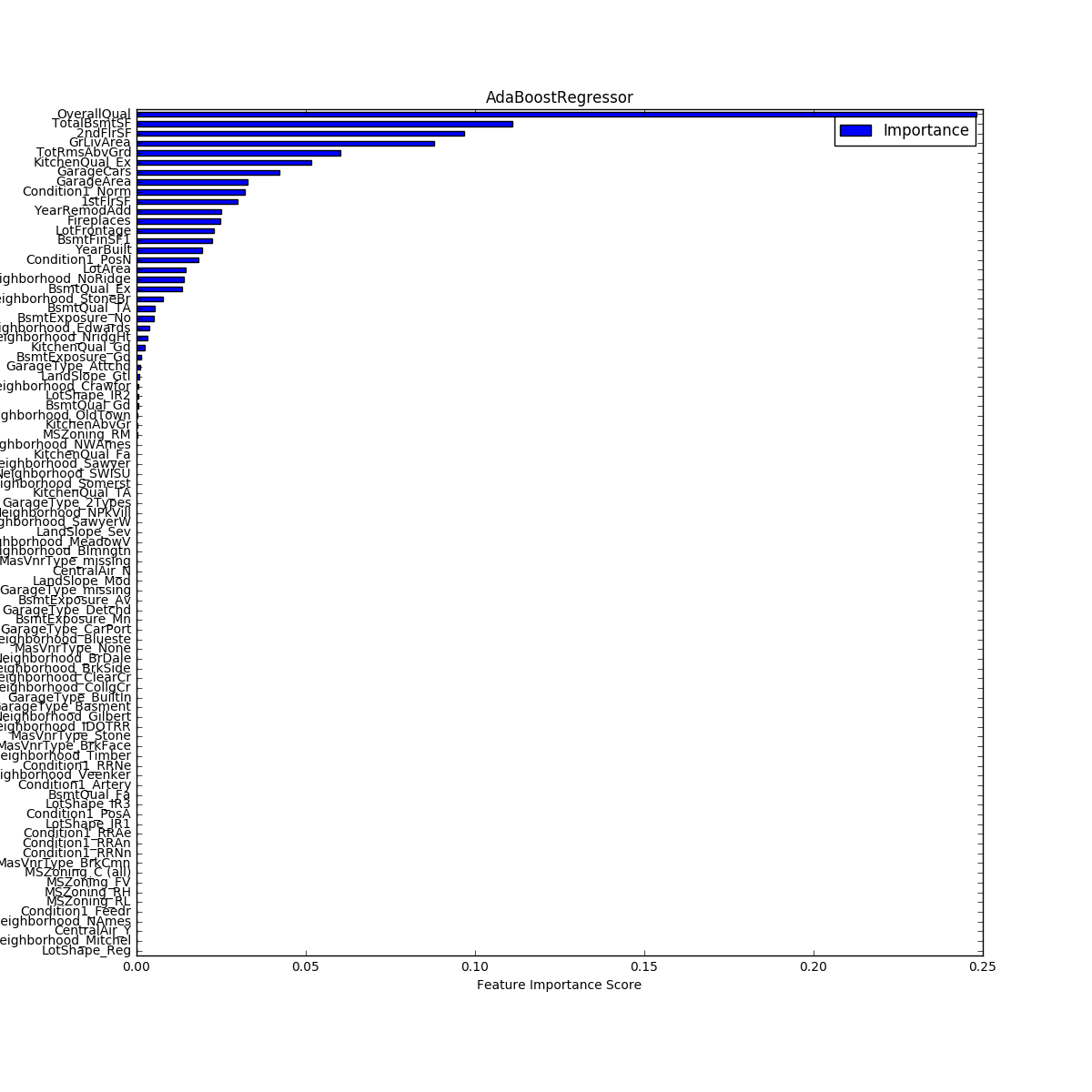
print(e)

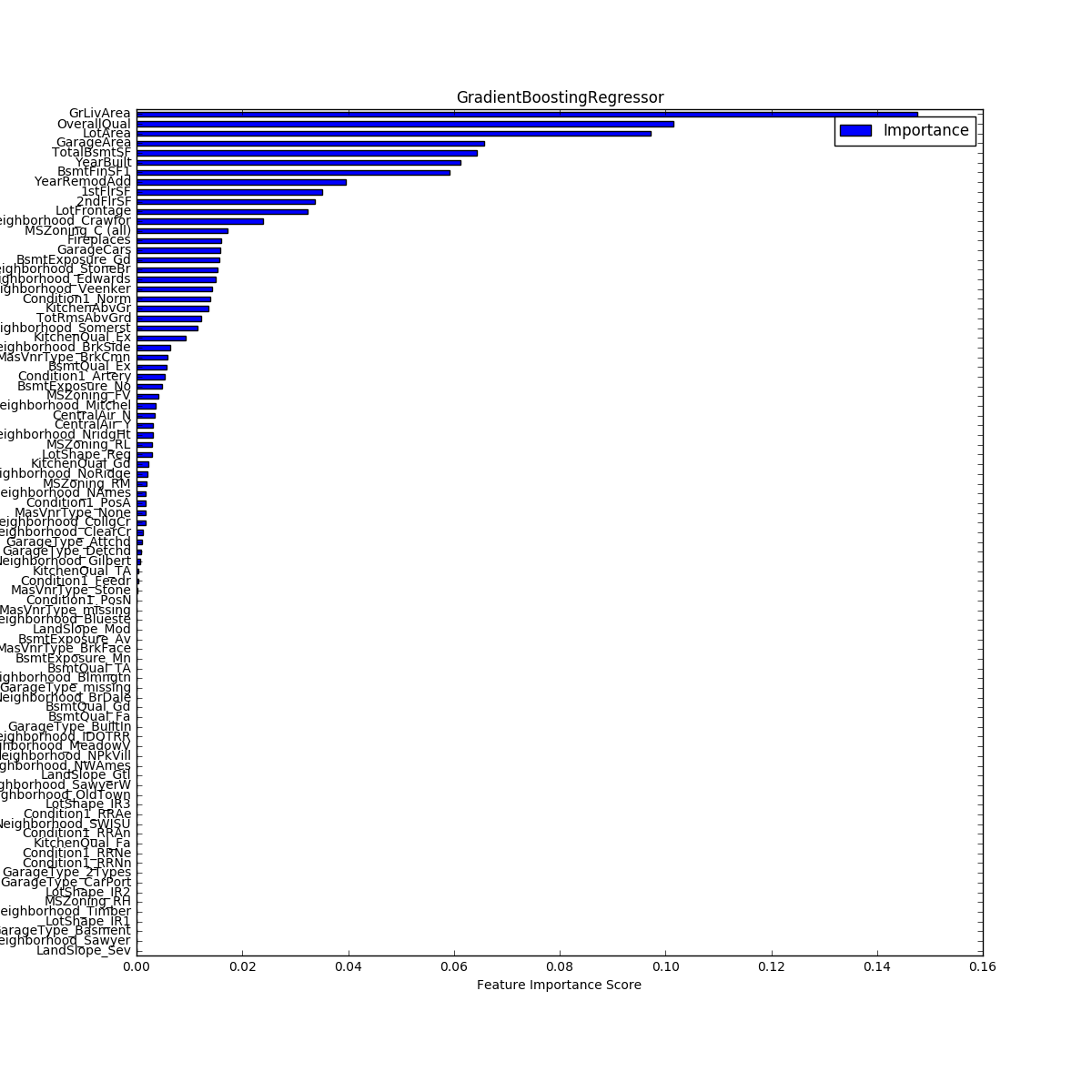
logger.info("Best model: \n {} : {}\n\n".format(best\_model, best\_score))

得到每个模型各个特征的权重值如下图所示：









在验证集上的准确度为：

|  |  |
| --- | --- |
| 模型 | 准确度（验证集）% |
| DecisionTreeRegressor | 64.90 |
| RandomForestRegressor | 81.15 |
| AdaBoostRegressor | 76.86 |
| GradientBoostingRegressor | 88.70 |

如上表可得，GradientBoostingRegressor的准确度最高，其次是RandomForestRegressor，因此在预测时选用GradientBoostingRegressor梯度提升回归树。接下来对这这个模型进行简要的介绍。

3.2 GradientBoostingRegressor梯度提升回归树

GradientBoostingRegressor是集成学习Boosting家族的成员，但是却和传统的Adaboost有很大的不同。Adaboost是利用前一轮迭代弱学习器的误差率来更新训练集的权重，这样一轮一轮的迭代下去。GradientBoostingRegressor也是迭代，使用了前向分布算法，但是弱学习器限定了只能使用CART回归树模型，同时迭代思路和Adaboost有所不同。

在GradientBoostingRegressor的迭代中，假设前一轮迭代得到的强学习器是ft-1(x)，损失函数是 L(y, ft-1(x)) ，本轮迭代的目标是找到一个CART回归树模型的弱学习器 ht(x) ，让本轮的损失函数 L(y,ft(x)) = L(y,ft-1(x) + ht(x)) 最小 。即本轮迭代找到决策树，要让样本的损失尽量变得更小。

GradientBoostingRegressor回归算法:

1. 初始化弱学习器
2. 对迭代轮数t = 1,2,3 … m，有：

a）对样本i = 1,2,...,m，计算负梯度

b）利用（xi，rti），i = 1,2,3 … m，拟合一棵CART回归树，得到第t棵回归树，其对应的叶结点区域为Rtj， j= 1,2, … J。其中其中j为回归树t的叶结点的个数。

c) 对叶子区域j= 1,2, … J，计算最佳拟合值。

d) 更新强学习器

1. 得到强学习器f(x)的表达式

创建决策树子类

"""

-----------

n\_estimators: int #树的数量

learning\_rate: float #梯度下降的学习率

min\_samples\_split: int #每棵子树的节点的最小数目（小于后不继续切割）

min\_impurity: float #每颗子树的最小纯度（小于后不继续切割）

max\_depth: int #每颗子树的最大层数（大于后不继续切割）

regression: Boolean #是否为回归问题

"""

def \_\_init\_\_(self, n\_estimators, learning\_rate, min\_samples\_split,

min\_impurity, max\_depth, regression):

#导入参数

self.n\_estimators = n\_estimators

self.learning\_rate = learning\_rate

self.min\_samples\_split = min\_samples\_split

self.min\_impurity = min\_impurity

self.max\_depth = max\_depth

self.regression = regression

# 进度条 processbar

self.bar = progressbar.ProgressBar(widgets=bar\_widgets)

# 定义损失函数

self.loss = SquareLoss() #平方损失（残差的平方和），用于回归问题

if not self.regression:

self.loss = SotfMaxLoss()#用于分类

#回归树，利用残差去学习

self.trees = []

for i in range(self.n\_estimators):

self.trees.append(RegressionTree(min\_samples\_split=self.min\_samples\_split,

min\_impurity=self.min\_impurity,

max\_depth=self.max\_depth))

def fit(self, X, y):

self.trees[0].fit(X, y) # 让第一棵树去拟合模型

y\_pred = self.trees[0].predict(X)

for i in self.bar(range(1, self.n\_estimators)): #进入循环

gradient = self.loss.gradient(y, y\_pred)

self.trees[i].fit(X, gradient)

y\_pred -= np.multiply(self.learning\_rate, self.trees[i].predict(X))

for循环的过程就是不断让下一棵树拟合上一颗树的"残差"(梯度)。

而残差是由梯度求出。在square loss中，gradient = yi - F(xi),此时梯度刚好等于残差。

def predict(self, X):

y\_pred = self.trees[0].predict(X)

for i in range(1, self.n\_estimators):

# for循环的过程就是汇总各棵树的残差得到最后的结果

y\_pred -= np.multiply(self.learning\_rate, self.trees[i].predict(X))

if not self.regression:

# 变成概率分布

y\_pred = np.exp(y\_pred) / np.expand\_dims(np.sum(np.exp(y\_pred), axis=1), axis=1)

# 将标签设置为概率最大的值

y\_pred = np.argmax(y\_pred, axis=1)

return y\_pred

四. 实验结果

上述步骤已经处理好了实验数据，并且在四个模型中挑选出了最优的GradientBoostingRegressor模型。将测试数据放入训练好的模型中进行预测，得到最终的实验预测结果，并且保存。

def predict\_result(clf,x\_train,y\_train,x\_test):

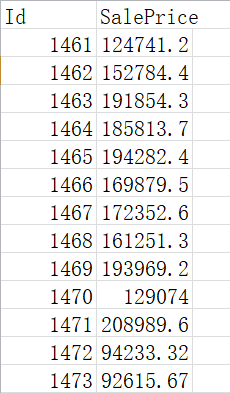
rf = clf.fit(x\_train,y\_train)

preds = rf.predict(x\_test)

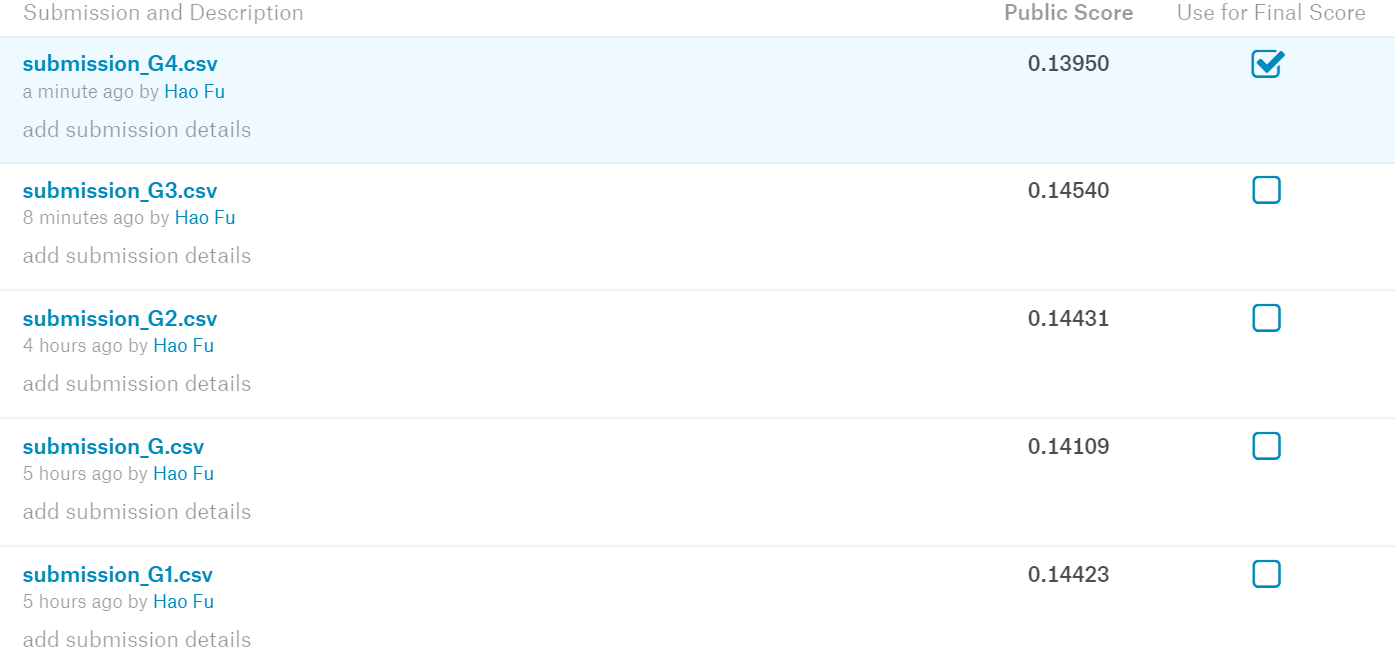
result = pd.DataFrame({'SalePrice':preds})

return result

得到的实验结果大致如下：



将实验结果上传至kaggle，得分为0.139，排名约半数。



五. 实验心得

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。

这次选的题目有很大一部分的数据处理内容，在刚开始接手的时候比较吃力。以前没有接触过这样的python编程练习，每一种实现方法、每一条代码函数都要自己去查资料。

在学习中我知道，特征工程是很重要的部分，好的数据是必要基础。因此，在以后的学习研究中，也要好好学习如何进行数据分析，做好特征工程的工作。

这次的实验结果并不是太好，得分0.139。在这次实验中我还有个遗憾，本来想在各个特征提取时采用pac算法，但是由于数据格式或是其他的问题，一直实现不了。程序无法跑通，也一直报错。应该还是在数据处理方面出现了问题，还是要提升自己的编程能力。

实验过程中我参考了《机器学习》、《机器学习实战》与CS321课程，让我知道这短短半学期的课程我学到的知识远远不够，我还要学习好多别的知识才能摸到模式识别的皮毛。这次的大作业让增加了我对模式识别的兴趣，同时原来神秘的推荐算法变得立体清晰，感谢赵老师教学帮助。

附：文件说明

由于之前没有使用过github，不知道如何创建文件夹，因此文件比较多比较杂，给老师的审阅带去麻烦，我表示很抱歉。

1.最终提交数据submission\_G4.csv

2.实验代码Kaggle-housePrices.py

3.各个模型的输出图像在fig文件夹中

4.实验过程记录house-p-\_20191121-015431.log

5.大作业报告