華東理工大學

模式识别大作业

题　目	房价预测
学　院	信息科学与工程学院
专　业	控制科学与工程
组　员	邱穗庆、赵金敏
指导教师	赵海涛

完成日期：2019 年 12 月 5 日

模式识别大作业报告--房价预测

组员：邱穗庆、赵金敏

一、房价预测简介

提供 1460 组房价数据信息，也就是训练数据，每栋房子用 79 个特征变量来表征，我们需要通过对这 79 个特征变量进行分析，进而找出各个特征变量与真实房价之间的关系，并且建立一个模型，这个模型要尽可能准确地表达真实房价与特征变量之间的关系。建立好模型之后，用它来预测其他房子的价格，在测试集中同样用 79 个特征变量描述房子的特点。

二、需要解决的重难点

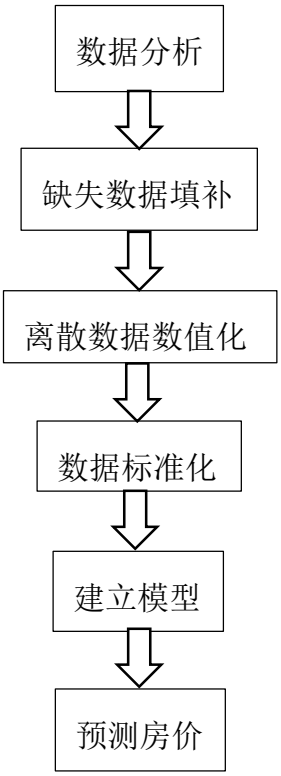
1、训练和测试数据集中都缺失了许多数据，需要根据不同情况用对应的方法进行数据填充；

2、各个特征变量的类型等不一致，要对数据进行数值化、标准化等处理；

3、房子的特征有 79 个，如何对其进行取舍；

4、回归模型的选取和建立，我们采用岭回归和 XGBoost 建立回归模型，最终模型由对上面两个模型进行加权平均得出，最终效果比单独模型更好，最终得分为 0.11938，将对这两种方法进行详细介绍。

三、整体解决方案



3.1 数据分析

1）先导入数据，如下：

```
train = pd.read_csv(r'F:\House price\train.csv')
test = pd.read_csv(r'F:\House price\test.csv')
```

2）观察数据，大致了解数据形式，有个整体印象，看需要做哪些处理，如下：

```
train.head(5)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...

5 rows × 81 columns

可以看到，除去行号、Id 和房价共有 79 个特征，其中有很多特征不是数值型，需要对它们进行数值化处理，各个特征数据相差也比较大，需要统一进行标准化处理。

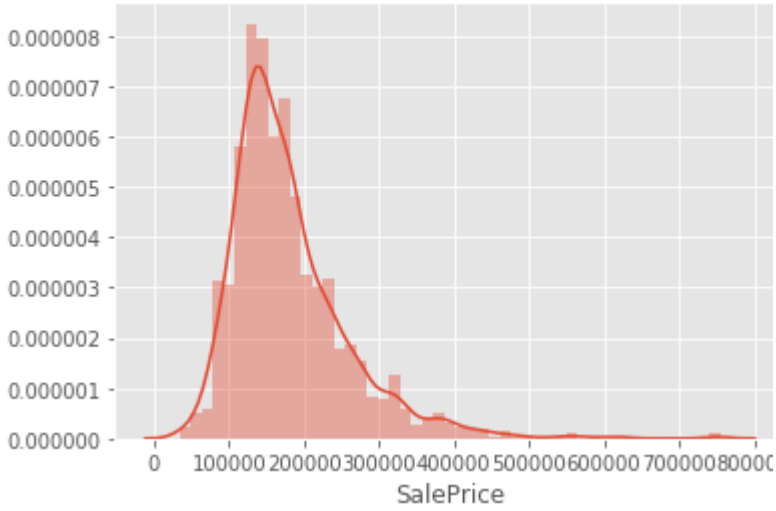
3）看看房价数据的分布情况，有无异常值等，如下：

```
sns.distplot(train['SalePrice'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xa87d0ebc08>
```

```
train['SalePrice'].describe()
```

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```



可见，训练集中共有 1460 个房价数据，说明房价数据没有缺失值，而且其平均值为 18.1 万美元，最低 3.49 万美元，最高为 75.5 万美元，其总体分布稍往右偏，为让其尽量呈正态分布，需要对其取对数处理。

4）因为测试集和训练集特征变量都一样，因此可以将两个数据合并起来进行数据处理，并将无关数据删除，如下：

```
#列合并训练和测试数据
full=pd.concat([train,test], ignore_index=True)
```

```
#删除ID这一列数据
full.drop(['Id'],axis=1, inplace=True)
full.shape
```

```
(2919, 80)
```

可见，测试集包含 1459 个数据信息。

5）观察特征数据的缺失情况，如下：

```
#缺失数据个数
aa = full.isnull().sum()
aa[aa>0].sort_values(ascending=False)
```

```
PoolQC          2909
MiscFeature     2814
Alley           2721
Fence           2348
SalePrice       1459
FireplaceQu     1420
LotFrontage      486
GarageQual       159
GarageCond       159
GarageFinish     159
GarageYrBlt      159
GarageType       157
BsmtExposure      82
BsmtCond          82
BsmtQual          81
BsmtFinType2      80
BsmtFinType1      79
MasVnrType        24
MasVnrArea        23
MSZoning           4
BsmtFullBath       2
BsmtHalfBath       2
Utilities          2
Functional         2
Electrical         1
BsmtUnfSF          1
Exterior1st        1
Exterior2nd        1
TotalBsmtSF        1
GarageCars         1
BsmtFinSF2         1
BsmtFinSF1         1
KitchenQual        1
SaleType           1
GarageArea         1
dtype: int64
```

可见，有的数据缺失很多，仔细观察一下 data_description 里面的内容的话，发现很多缺失值都有迹可循，比如上表第一个 PoolQC，表示的是游泳池的质量，其值缺失代表的是这个房子本身没有游泳池，因此可以用“None”来填补。有的数据缺失比较少，可以用众数或平均数来填补。在进行数据填补时尤其要注意要考虑具体情况选择合适的方法进行数据填补。

3.2 缺失数据填补

对于"MasVnrArea", "BsmtUnfSF", "TotalBsmtSF", "GarageCars","BsmtFinSF2", "BsmtFinSF1", "GarageArea" 等特征多为表示面积，比如 TotalBsmtSF 表示地

下室的面积，如果一个房子本身没有地下室，则缺失值就用 0 来填补，如下：

```
# fill lost data with zero
cols=["MasVnrArea", "BsmtUnfSF", "TotalBsmtSF", "GarageCars", "BsmtFinSF2", "BsmtFinSF1", "GarageArea"]
for col in cols:
    full[col].fillna(0, inplace=True)
```

对于 "PoolQC","MiscFeature","Alley","Fence","FireplaceQu", "GarageQual", "GarageCond", "GarageFinish", "GarageYrBlt" 等特征数据，其值缺失代表的是这个房子本身没有该特征，因此可以用“None 来填补

```
# fill lost data with None
cols1 = ["PoolQC" , "MiscFeature", "Alley", "Fence", "FireplaceQu", "GarageQual", "GarageCond",
for col in cols1:
    full[col].fillna("None", inplace=True)
```

对于 "MSZoning", "BsmtFullBath", "BsmtHalfBath", "Utilities", "Functional", "Electrical", "KitchenQual", "SaleType","Exterior1st", "Exterior2nd" 这些特征数据缺失比较少，采用众数去填补，如下：

```
# fill lost with 众数（缺失值较少）
cols2 = ["MSZoning", "BsmtFullBath", "BsmtHalfBath", "Utilities", "Functional", "Electrical"
for col in cols2:
    full[col].fillna(full[col].mode()[0], inplace=True)
```

对于 LotFrontage 这个特征与 LotAreaCut 和 Neighborhood 有比较大的关系，所以这里用这两个特征分组后的中位数进行填补，如下：

```
full["LotAreaCut"] = pd.qcut(full.LotArea,10)
full['LotFrontage']=full.groupby(['LotAreaCut','Neighborhood'])['LotFrontage'].transform(lambda x: x.fillna(x.median()))
full['LotFrontage']=full.groupby(['LotAreaCut'])['LotFrontage'].transform(lambda x: x.fillna(x.median()))
```

数据填补完成后，观察一下是否所有缺失值都已填补完毕，如下：

```
aa = full.isnull().sum()
aa[aa>0].sort_values(ascending=False)
```

```
SalePrice    1459
dtype: int64
```

可见，此时所有缺失的特征数据都已填补完毕。

3.3 离散数据数值化

对于离散型特征，一般采用 pandas 中的 get_dummies 进行数值化，但这样可能还不够，所以下面采用的方法是按特征进行分组，计算该特征每个取值下 SalePrice 的平均数和中位数，再以此为基准排序赋值，这里只举一个例子：

```
#MSSubClass这个特征表示房子的类型，将数据按其分组：
full.groupby(['MSSubClass'])[['SalePrice']].agg(['mean','median','count'])
```

	SalePrice		
	mean	median	count
MSSubClass			
120	200779.080460	192000.0	87
150	NaN	NaN	0
160	138647.380952	146000.0	63
180	102300.000000	88500.0	10
190	129613.333333	128250.0	30

按各个特征输出表中的数值进行排序，完成数据的数值化操作，这里只截图部分操作，完整操作见代码：

```
#数值化映射-升序排列
def map_values():
    full["oMSSubClass"] = full.MSSubClass.map({'180':1,
                                        '30':2, '45':2,
                                        '190':3, '50':3, '90':3,
                                        '85':4, '40':4, '160':4,
                                        '70':5, '20':5, '75':5, '80':5, '150':5,
                                        '120': 6, '60':6})
```

3.4 数据标准化

这里用到 python 中的 Pipeline 管道机制对数据进行标准化：

```
#pipeline 数据预处理
class labelenc(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass
    def fit(self,X,y=None):
        return self
    def transform(self,X):
        lab=LabelEncoder()
        X["YearBuilt"] = lab.fit_transform(X["YearBuilt"])
        X["YearRemodAdd"] = lab.fit_transform(X["YearRemodAdd"])
        X["GarageYrBlt"] = lab.fit_transform(X["GarageYrBlt"])
        return X
```

```
class skew_dummies(BaseEstimator, TransformerMixin):
    def __init__(self,skew=0.5):
        self.skew = skew
    def fit(self,X,y=None):
        return self
    def transform(self,X):
        X_numeric=X.select_dtypes(exclude=["object"])  #只包含字符串或多种类型时为object
        skewness = X_numeric.apply(lambda x: skew(x))
        skewness_features = skewness[abs(skewness) >= self.skew].index
        X[skewness_features] = np.log1p(X[skewness_features])
        X = pd.get_dummies(X)
        return X
```

```
#针对离群点做标准化处理
scaler = RobustScaler()
```

```
pipe = Pipeline([
    ('labenc', labelenc()),
    ('skew_dummies', skew_dummies(skew=1))
    ])
```

```
n_train=train.shape[0]

X = data_pipe[:n_train]
test_X = data_pipe[n_train:]
y= train.SalePrice

X_scaled = scaler.fit(X).transform(X)
y_log = np.log(train.SalePrice)
test_X_scaled = scaler.transform(test_X)
```

3.5 建立模型

数据处理完之后，便可以建立模型了，采用了岭回归和 xgboost 两种方法，下面主要介绍这两种方法。

3.5.1 岭回归

在线性回归中，容易出现过拟合现象，解决办法一般有两种：(1)：丢弃一些对我们最终预测结果影响不大的特征，具体哪些特征需要丢弃可以通过 PCA 算法来实现；(2)：使用正则化技术，保留所有特征，但是减少特征前面的参数 θ 的大小，具体就是修改线性回归中的损失函数形式即可，岭回归就是这么做的。

岭回归的损失函数：

$$J(\theta)=\frac{1}{2n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2+\lambda\sum_{j=1}^{n}\theta_j^2 \qquad (1)$$

其中 λ 称为正则化参数，如果 λ 选取过大，会把所有参数 θ 均最小化，造成欠拟合，如果 λ 选取过小，会导致对过拟合问题解决不当，因此 λ 的选取是一个技术活。

采用基于最小二乘法的正则化算法对参数 θ 进行求解，对上式求偏导：

$$\frac{\partial}{\partial\theta}J(\theta)=\frac{1}{n}\left[\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)x_j^{(i)}+\lambda\theta_j\right],\ j=1,2,\cdots,n \qquad (2)$$

令上式为零，即：

$$\frac{\partial}{\partial\theta}J(\theta)=X^T\left(X\theta-Y\right)+\lambda\theta=0 \qquad (3)$$

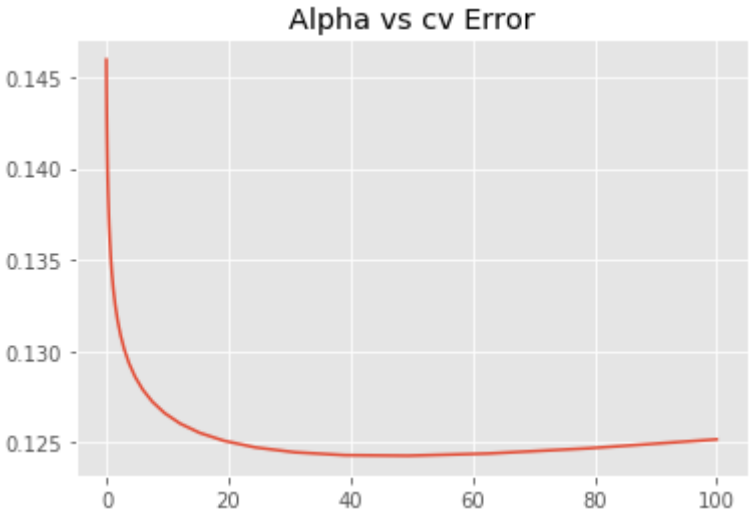
得出：

$$X^TX\theta+\lambda\theta=X^TY \qquad (4)$$

从而得出：

$$\theta=\left(X^TX+\lambda I_n\right)^{-1}X^TY \qquad (5)$$

这里的重点就是要确定最优的λ，要确定λ首先要从评价指标入手，这里的评分标准是均方根误差（RMSE），痛过实验得出本实验中λ与 RMSE 之间的关系如下图所示：



可见，这里当λ=40 时，单独的岭回归模型能让 RMSE 取得的最小值为 0.124315，结果如下：

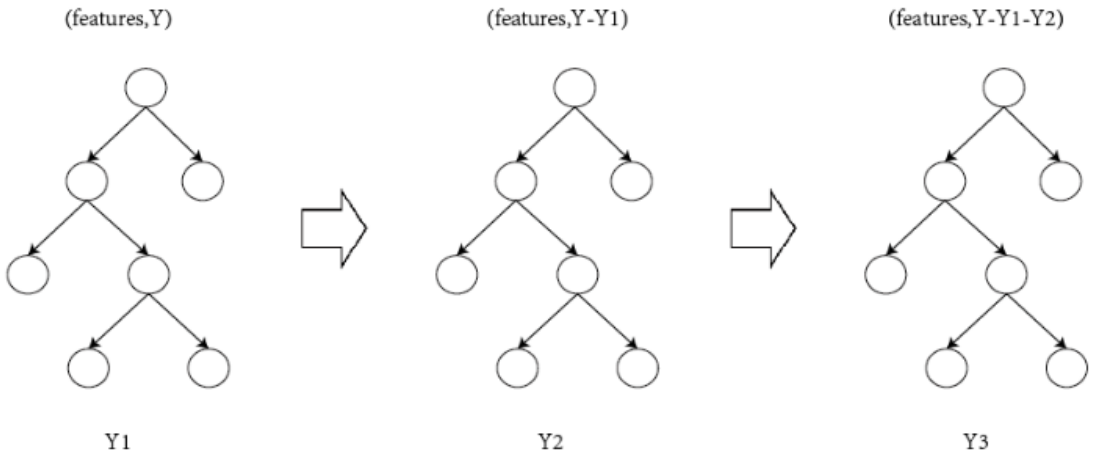
```
score = rmse_cv(Ridge(40),X_scaled,y_log)
print(score.mean())
```

```
0.1243150232420096
```

即，单独用岭回归方法建立回归模型时，取得最好分数是 0.124315。

3.5.2 XGBoost

说到 xgboost，不得不说 gbdt，两者都是 boosting 方法如下图所示：



$$Y = Y_1 + Y_2 + Y_3 \qquad (6)$$

如果不考虑工程实现、解决问题上的一些差异，xgboost 与 gbdt 比较大的不同就是目标函数的定义。

- 目标 $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
- 用泰勒展开来近似我们原来的目标
  - 泰勒展开：$f(x+\Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$
  - 定义：$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}),\quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

$$Obj^{(t)} \simeq \sum_{i=1}^{n}\left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)\right] + \Omega(f_t) + constant$$

其中，红色箭头指向的 l 即为损失函数；红色方框为正则项，包括 L1、L2；红色圆圈为常数项。xgboost 利用泰勒展开三项，做一个近似，我们可以很清晰地看到，最终的目标函数只依赖于每个数据点的在误差函数上的一阶导数和二阶导数。

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2$$

叶子的个数　　w的L2模平方

方框部分在最终的模型公式中控制这部分的比重，对应模型参数中的 lambda ，gamma。在这种新的定义下，我们可以把目标函数进行如下改写，其中 I 被定义为每个叶子上面样本集合 $I_j = \{i|q(x_i) = j\}$。

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)\right] + \Omega(f_t) \\ &= \sum_{i=1}^{n}\left[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2\right] + \gamma T + \lambda\frac{1}{2}\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}\left[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2\right] + \gamma T \end{aligned}$$

这一个目标包含了 T 个相互独立的单变量二次函数。我们可以定义：

$$G_j = \sum_{i\in I_j} g_i \quad H_j = \sum_{i\in I_j} h_i$$

最终公式可以化简为：

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T \\ &= \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T \end{aligned}$$

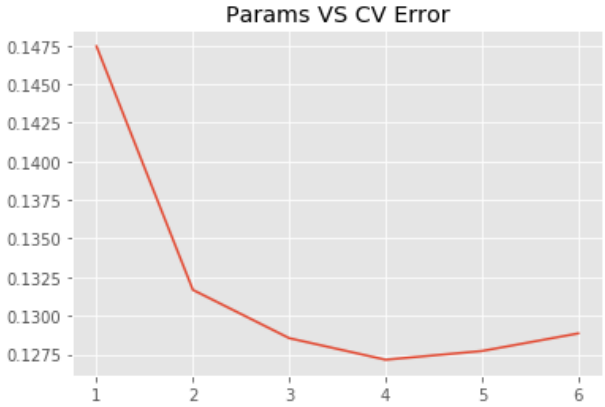
通过对 ω_j 求导等于 0，可以得到：

$$w_j^* = -\frac{G_j}{H_j+\lambda}$$

然后把 ω_j 最优解代入得到：

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

同理，通过实验求得：



可见，这里当 param=4 时，单独用 XGBoost 建立的模型能让 RMSE 取得的最小值为 0.12715，结果如下：

```
score = rmse_cv(XGBRegressor(max_depth=4),X_scaled,y_log)
print(score.mean())
```

```
[15:27:52] WARNING: C:/Jenkins/workspace/xgboost-win64_rel
or of reg:squarederror.
[15:27:55] WARNING: C:/Jenkins/workspace/xgboost-win64_rel
or of reg:squarederror.
[15:27:57] WARNING: C:/Jenkins/workspace/xgboost-win64_rel
or of reg:squarederror.
[15:28:00] WARNING: C:/Jenkins/workspace/xgboost-win64_rel
or of reg:squarederror.
[15:28:03] WARNING: C:/Jenkins/workspace/xgboost-win64_rel
or of reg:squarederror.
0.12715393737996425
```

对以上两种模型加权平均，经过反复调整，取 0.53*岭回归+0.47*XGBoost 效果最佳，最终得分为 0.11938，结果如下。

```
weight_avg = AverageWeight(mod = [Ridge(40),XGBRegressor(max_depth=4)],weight=[0.53,0.47])
score = rmse_cv(weight_avg,X_scaled,y_log)
print(score.mean())
```

```
[15:28:12] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regressio
or of reg:squarederror.
[15:28:14] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regressio
or of reg:squarederror.
[15:28:17] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regressio
or of reg:squarederror.
[15:28:19] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regressio
or of reg:squarederror.
[15:28:22] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regressio
or of reg:squarederror.
0.11938315368843806
```

按照比赛要求，预测结果保存在"Prediction_Result.csv"文件中。

本来想上传至 Kaggle 看一下排名，但是一直上传不了。。。

4、小组分工

XGBoost 部分程序设计、编写与调试，XGBoost 部分报告：赵金敏

其余部分程序设计、编写与调试，其余报告部分：邱穗庆

5、作业总结

岭回归因为引入了 L2 范数惩罚项，可以有效的防止过拟合，从而获得更可靠的回归模型。同样，XGBoost 包含的正则化步骤对减少过拟合也有很大的帮助，同时因为 XGBoost 可以实现并行处理，相比于 GMB 有了速度的飞跃，而且，XGBoost 允许在每一轮 boosting 迭代中使用交叉验证。因此，可以方便地获得最优 boosting 迭代次数。

本次作业中采用了岭回归和 XGBoost 两种方法进行建模，取得了不错的效果，在后续的学习中，也可以加入新的建模方法，比如：随机森林、Lasso 回归、SVM 等方法，再对各种方法进行加权平均，看最终的效果。因此可以激励我们不断的去学习新的机器学习方法。

附:文件说明

本次附件一共包含有：

1、大作业报告：House price 报告

2、Python 实现程序：House price.ipynb

3、最终房价预测结果：Prediction_Result.csv

4、测试集：train.csv

5、训练集：test.csv

6、数据描述：data_description.txt

7、输出结果样本示例：sample_submission.csv