房价预测项目总结

房价预测项目总结目录 1

1 引言

1.1 项目背景

2019 年初机缘巧合，进入《开课吧》参加了黄老师的机器学习实践课程，在黄老师的带领下初步认识了数据分析、机器学习、神经网络等人工智能相关内容，并深入学习并实践了放假预测项目。基于老师提供的住宅数据信息，预测每间房屋的销售价格，很明显是一个回归问题。

1.2 参考文档

黄老师课程及笔记

2 项目完成情况总结

2.1 最开始获取过程

2.1.1 导入各种包

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import matplotlib.pyplot as plt
```

```
%matplotlib inline

import seaborn as sns

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

from sklearn.pipeline import make_pipeline

from sklearn.preprocessing import RobustScaler

from sklearn.linear_model import Ridge

from sklearn.model_selection import KFold

from sklearn.linear_model import RidgeCV

from sklearn.linear_model import LassoCV

from sklearn.linear_model import ElasticNetCV

import xgboost as xgb

from mlxtend.regressor import StackingCVRegressor

from mlxtend.regressor import StackingCVRegressor
```

神经网络部分：

```
import keras

import tensorflow as tf

from keras.models import Sequential

from keras.layers import Dense,Dropout

from keras import metrics

from keras import backend as K
```

```
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV
from keras.layers import BatchNormalization,Activation
from keras import optimizers
```

2.1.2 了解算法的 Pipeline 和 Baseline

1）Pipeline

数据采集->数据清洗->数据仓库->数据挖掘->数据标注->数据集市(训练集/评测集)->训练->评测->模型工程(Int8 定点化)->模型的集成和应用

2)Baseline

训练集的制作->训练->评测

在形成一个 BaseLine 之后，模型迭代的过程进步的标志是评测指标的提升.

3）评测指标

Regression 回归问题：MSE(Mean Squared Error)-L2/MAE(Mean Absolute Error) –L1

Classification 分类问题：Cross-Entropy(交叉熵)

2.1.3 数据收集及初探

data.csv数据集中包含了1460行和81列。RangeIndex: 1460 entries, 0 to 1459，Data columns (total 81 columns):

往往关于数据的填充和Outlier（离群值）的处理是清洗的第一步，数据类型一般分为两种：Numerical 和 Categorial，连续型数据需要进行归一化处理，离散型数据需要进行 LabelEncode 编码城数字。

1）数据类型统计：train.get_dtype_counts(),得到数据的特征中有 3 个 float 类型，35 个 int 类型，43 个 object 类型

2）数据的缺失率统计：

```
nullRate = train.isnull().sum()
(i/len(train) for I in nullRate)
tmpNullNum = train.isnull().sum().sort_values()
tmpNullNum[tmpNullNum>0]
```

得到：

```
Electrical        1
MasVnrType        8
MasVnrArea        8
BsmtQual         37
BsmtCond         37
BsmtFinType1     37
BsmtFinType2     38
BsmtExposure     38
GarageQual       81
GarageFinish     81
GarageYrBlt      81
GarageType       81
GarageCond       81
LotFrontage     259
FireplaceQu     690
Fence          1179
Alley          1369
```

```
MiscFeature     1406
PoolQC          1453
```

2.1.4 数据清洗

1）缺失值类型填充

Numerical

1.均值填充:

train1=train1.fillna(train1.mean())

fillna 函数会找到对应列的均值或者是中位数，对于该列进行相应的填充

2.中位数填充

3.高频填充--高频会"稍微"少一些--连续型数据很难数字相等，但是很容易接近)

4.分布拟合填充：回归一个分布，或者是均值和标准差+x 范围进行随机

5.内在关系填充(根据列本身存在的意义和数据集内其他列的相关性进行构建数学映射模型进行填充)

Categorial

1.高频填充

"男"|"女"-->0|1

"小孩"|"青年"|"中年"|"老年"-->0|1|2|3

No Free Lunch Theory-不存在超级机器学习模型

2.内在关系填充

3.填充'None'

train1=train1.fillna('None')

2.1.5 训练集的制作

y = train1['SlaePrice']

train1 = train1.drop(['Id','SalePrice'],axis = 1)

X = pd.get_dummies(train1)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state = 123)

2.1.6 BaseLine 模型的训练及评测

xg_xgb = xgb.XGBRegressor(objective = 'reg:linear',colsample_bytree = 0.6,learning_rate = 0.01,max_depth = 5,alpha = 10,n_estimators = 1000,subsample=0.7,random_state = 123)

xg_reg.fit(X_train,y_train)

pred = xg_reg.predict(X_test)

rmse = np.sqrt(mena_squared_error(y_test,pred))

logrmse = np.sqrt(mean_squared_error(np.log(y_test),np.log(pred)))

2.1.7 交叉验证

1.将数据集拆成 K 份->(首先进行 shuffle)

2.规定 k-1 份进行训练，剩下的 1 份进行评测，总共训练 k 次，轮流每个子数据集作为评测集

数据集被分成 K 份，1.....K 第一次训练使用第 1 份数据集作为评测集，剩余的 k-1 份作为训练集 第 i 次训练使用第 i 份数据集作为评测集，剩余的 k-1 份作为训练集 做 k 次训练

3.k 次训练之后，评测的分值=k 次评测结果的平均

4.不同的模型会提供不同的子模型的合并方法，会将所有的 k 个子模型进行合并

2.2 数据 EDA

2.2.1 可视化探索性分析的要点

1.数据属性分布分析(概率密度分析/是否呈现正态分布)

2.数据属性与 Target 之间的相关性[correlation]

3.缺失属性相关性分析--进行合理的填充

4.构造属性和构造属性的相关性分析

2.2.2 相关性分析

值域：[-1 , +1]

|r|>0.95 显著关系 显著的关系需要概率该特征是否与目标有很强的关联关系(报价和成交价,这样的话可能考虑去掉，否则会影响判断的走向)

|r|>=0.8 强相关

|r|>=0.5 中度相关

0.5>=|r|>=0.3 弱相关 (待挖掘，可以尝试去掉)部分数据与该特征可能存在较强的相关性(楼房屋顶和顶楼的成交价的关系)

|r|<0.3 极弱相关

1) 协方差矩阵：train.corr

2) 与 Target 相关的相关系数：corr = train.corr()[‘SalePrice’]

corr[corr>0.5].sort_values()

两个特征之间的相关系数

```
np.corrcoef(train['GarageCars'],train['GarageArea'])
```

3）热度图

```
plt.figure(figsize=(30,30))

sns.heatmap(train.corr(),linewidths=0.01,sq
uare=True,cmap='viridis',annot=True)
```

4） 特征观察：观察特征与目标值之间的关系[使用统计图来进行观察]

```
corr=train.corr()['SalePrice']

corr[corr>0.5].sort_values()
```

得到：

```
YearRemodAdd    0.507101

YearBuilt       0.522897

TotRmsAbvGrd    0.533723

FullBath        0.560664

1stFlrSF        0.605852

TotalBsmtSF     0.613581

GarageArea      0.623431

GarageCars      0.640409

GrLivArea       0.708624

OverallQual     0.790982

SalePrice       1.000000
```

不要将连续的数据进行 groupby，这样做没有意义,如果想要对连续性的数据进行 group，那么就需要对该数据进行分段

```
train[['OverallQual','SalePrice']].groupby(['OverallQual']).mean().plot.bar()

train[['GarageCars','SalePrice']].groupby(['GarageCars']).mean().plot.bar()

train[['GarageCars','GarageArea']].groupby(['GarageCars']).mean().plot.bar()
```

2.2.3 正相关数据分析

OverallQual 装修质量

YearBuilt 建筑年代

YearRcemodAdd 重新改造时间(晚于或者等于建筑时间)

TotalBsmtSF 地下室的面积

1stFlrSF 1 楼面积

GrLivArea 地上整体面积

FullBath 总体卫生间数量

TotRmsAbvGrd 总体卧室数量

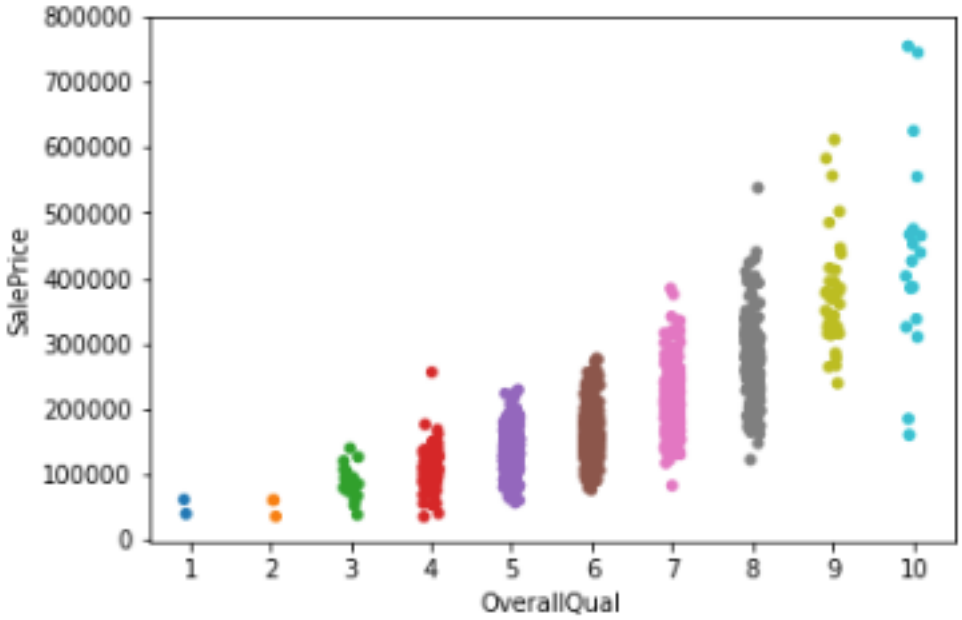
GarageCars 车库容车量

GarageArea 车库面积

1） OverallQual 与 SalePrice 的相关性分析

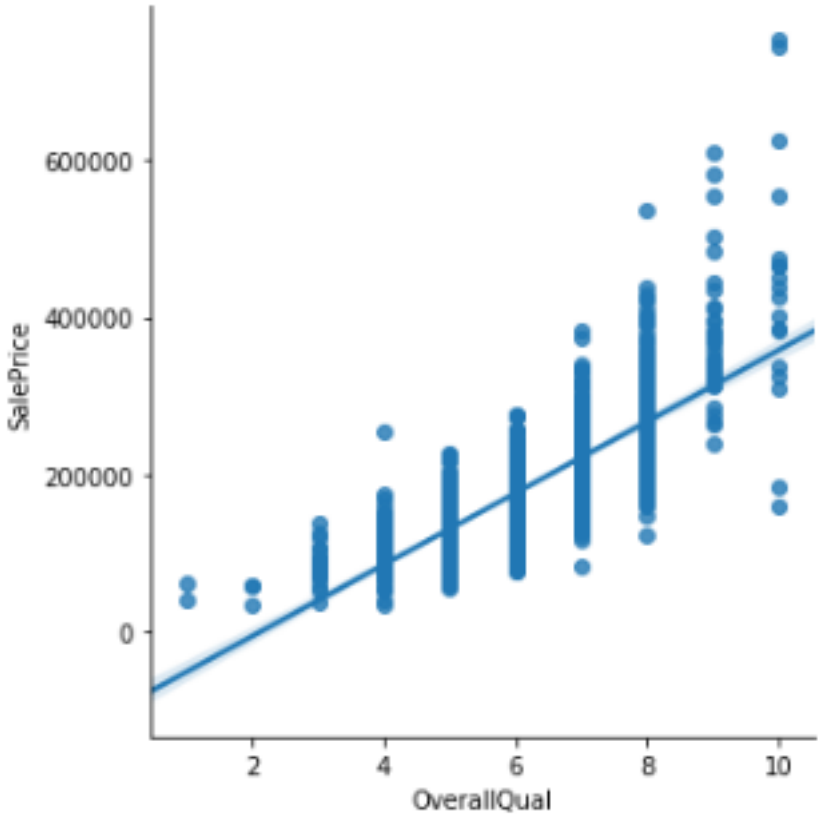
a)带状图 stripplot

sns.stripplot('OverallQual','SalePrice',data = train)



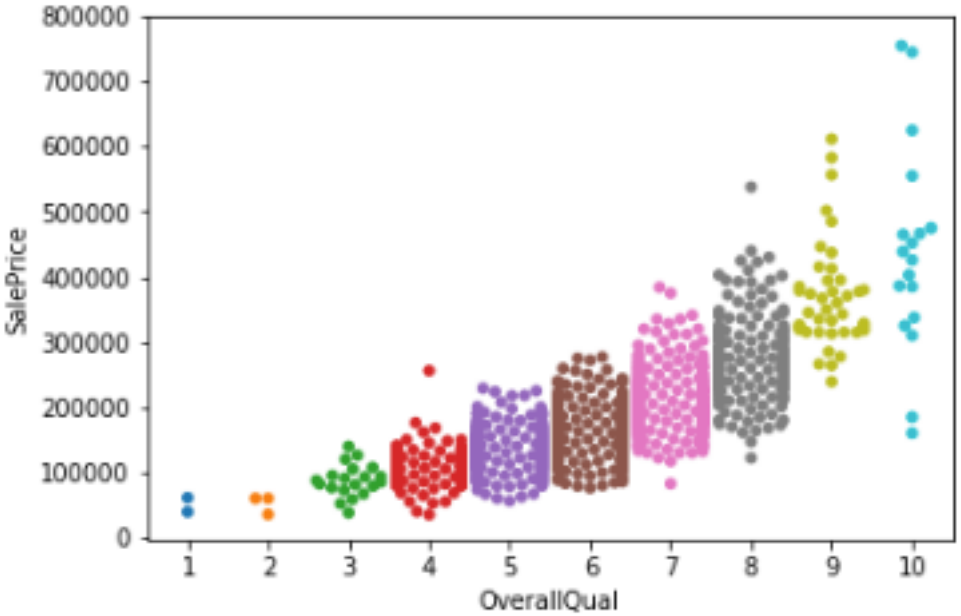
b)带回归的散点图 lmplot

sns.lmplot("OverallQual",'SalePrice',data=train)



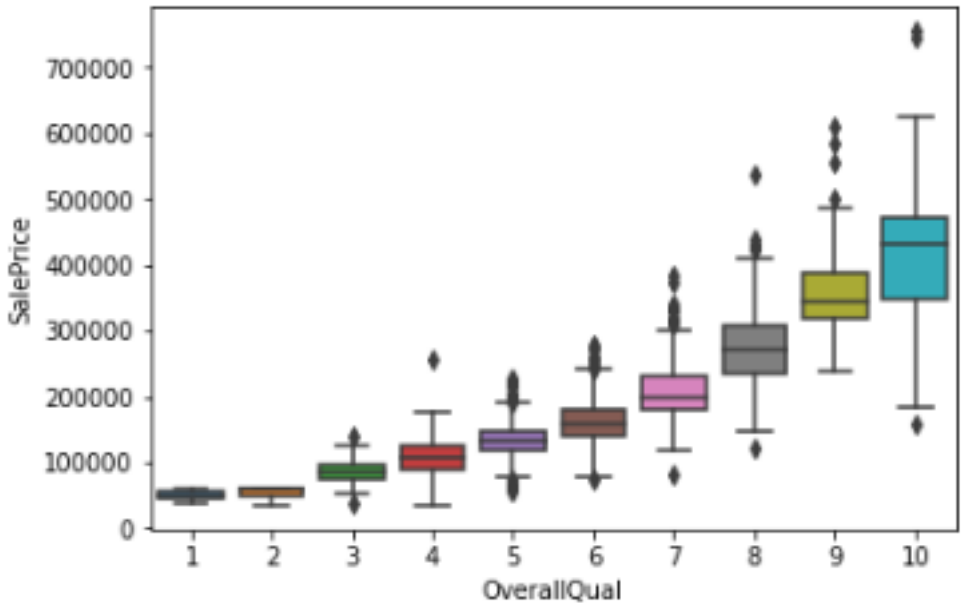
c) 蜂群图 swarmplot

sns.swarmplot("OverallQual","SalePrice",data=train)



d) 箱型图 boxplot

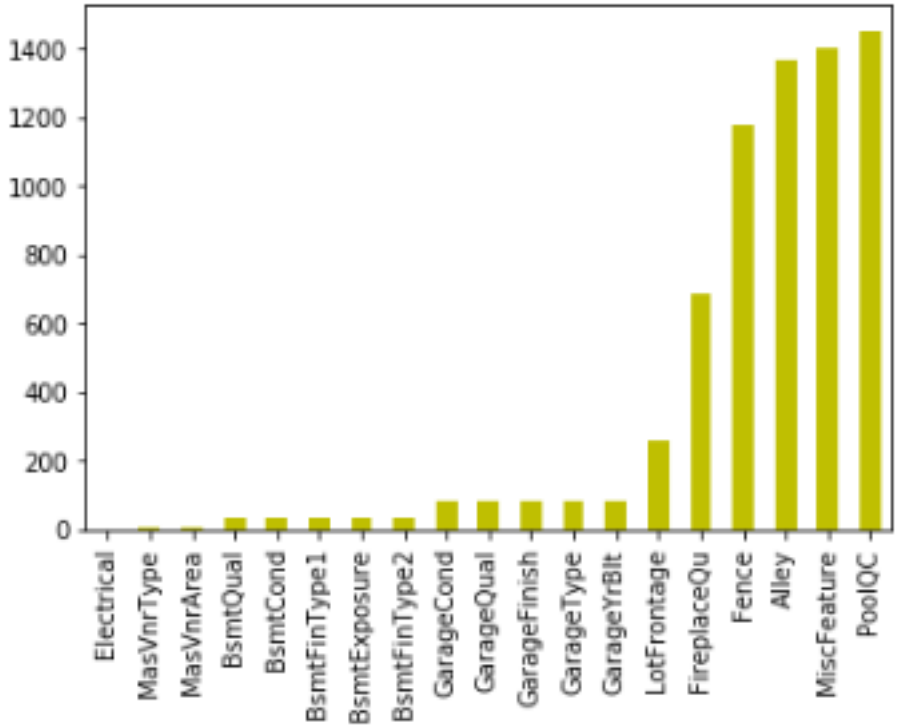
sns.boxplot("OverallQual","SalePrice",data=train)



2.2.4 根据可视化和相关性分析进行数据填充

1）查看空值数据

```
def getnullcount():
    x=train.isnull().sum()
    print(x[x>0])
    x[x>0].sort_values().plot.bar()
```



2) **LotFrontage 的填充**

方案一：取不同的 Neighborhood 的均值和中位数

```
sns.violinplot(train['LotFrontage'],train['Neighborhood'])

neighborhood_group=train.groupby('Neighborhood')

lot_medians=neighborhood_group['LotFrontage'].median()

lot_mean=neighborhood_group['LotFrontage'].mean()

train[train['LotFrontage'].isnull()]['Neighborhood']
```

方案二：通过 LotArea 进行填充

```
np.corrcoef(train['LotArea'],train['SalePrice'])
```

```
train['LotFrontage'].corr(train['LotArea'])

train['LotFrontage'].corr(np.sqrt(train['LotArea']))

train['SqrtLotArea']=np.sqrt(train['LotArea'])

sns.lmplot('SqrtLotArea','LotFrontage',data=train)

sns.jointplot('SqrtLotArea','LotFrontage',data=train)

#填充

filter=train['LotFrontage'].isnull()

train.LotFrontage[filter]=0.6*train.SqrtLotArea[filter]

filter=train['LotFrontage'].isnull()

filter.sum()
```

补充分类交叉熵损失函数概念：

在图像分类的过程中，比如说猫狗分类，我们分类的交叉熵可以定义成：

```
H(p,q)=-p_cat * log(q_cat)-p_dog * log(q_dog)
```

一张图片上面画了一只猫：

```
p_cat=1
```

p_dog=0

如果我的预测期：

q_cat=0.2

q_dog=0.8

H(p,q)=-1 * log(0.2)-0 * log(0.8)

=-1*log(0.2)=-log(0.2)=log(5)

对于分类模型的交叉熵=-log(q_label)-->分类的 Loss 函数

对于一张图片上画了一只猫的真实的熵本身是多少？

H(p,q)-H(p)=D(p||q)

3）**MasVnrType 和 MasVnrArea 的填充**

x=train.isnull().sum()

x[x>0]

plt.scatter(train['MasVnrArea'],train['SalePrice'])

sns.jointplot(train['MasVnrArea'],train['SalePrice'])

sns.boxplot('MasVnrType','SalePrice',data=train)

train.groupby(['MasVnrType']).count()

```
train[train.MasVnrType=='None'][['MasVnrType','MasVnrArea']]

sns.lmplot('MasVnrArea','SalePrice',hue='MasVnrType',data=train
)
```

填充：

```
filter=train['MasVnrArea'].isnull()

train.MasVnrArea[filter]=0.0

filter=train['MasVnrType'].isnull()

train.MasVnrType[filter]='None'
```

4） **Electrical 的填充**

```
sns.boxplot('Electrical','SalePrice',data=train)
train.groupby(['Electrical']).count()
filter=train['Electrical'].isnull()
train['Electrical'][filter]='SBrkr'
```

5) **Alley 填充**

```
train['Alley'].value_counts()
train['Alley']=train['Alley'].fillna('None')
```

6) BaseMent 群填充

TotalBsmt 是一个完整的关于 Basement 的列，可以拿出来进行与 SalePrice 相关性分析

```
plt.scatter(train['TotalBsmtSF'],train['SalePrice'])
basement_cols=['BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','BsmtFinSF1','BsmtFinSF2']
print(train[basement_cols+['TotalBsmtSF']][train['BsmtQual'].isnull()==True])
```

填充：

```
for col in basement_cols:
    if 'FinSF' not in col:
        train[col]=train[col].fillna('None')
```

7）FirePlace 填充

```
sns.lmplot('Fireplaces','SalePrice',data=train)
sns.lmplot('Fireplaces','SalePrice',data=train,hue='FireplaceQu')
train['FireplaceQu']=train['FireplaceQu'].fillna('None')
```

8）**Garage 列群填充**

```
sns.lmplot('GarageArea','SalePrice',data=train)
sns.distplot(train['GarageArea'],color='r',kde=True)
sns.violinplot(train['GarageCars'],train['SalePrice'])
garage_cols=['GarageType','GarageQual','GarageCond','GarageYrBlt','
```

GarageFinish','GarageCars','GarageArea']

train[garage_cols][train['GarageType'].isnull()]

填充：

```
for col in garage_cols:
    if train[col].dtype==np.object:
        train[col]=train[col].fillna('None')
    else:
        train[col]=train[col].fillna(0)
```

9）PoolQC 填充

train.filter(like='Pool',axis=1)

sns.distplot(train['PoolArea'],color='g',kde=True)

train.PoolQC=train.PoolQC.fillna('None')

10) **Fence 填充**

sns.violinplot(train['Fence'],train['SalePrice'])

因为总共有 5 个类型，然而可以统计出来的类型一共只有 4 类，因此，可以断定最后一个类 NA 用空值代替了

train['Fence']=train['Fence'].fillna('None')

回看填充后的数据与 SalePrice 的关系

sns.violinplot(train['Fence'],train['SalePrice'])

10）**MiscFeature 填充**

sns.violinplot(train['MiscFeature'],train['SalePrice'])

train['MiscFeature']=train['MiscFeature'].fillna('None')

11）保存数据到本地

```
train.to_csv('./train_zl.csv')
```

2.3 机器学习

数据填充完成后 ，将进入机器学习部分。需要考虑各种单独的模型的各自的表现，筛选出表现较好的模型，并进行集成学习。

此项目中我将依次尝试线性回归模型、树回归模型、SVM 模型、神经网络、集成学习。其中线性回归模型包括：朴素线性回归，基于 L1 的线性回归，基于 L2 的线性回归，ElasticNet.

树回归模型包括：CART，RF（随机森林），AdaBoost，GBDT（XGBoost，LightGBM）。

SVM：SVR。神经网络使用全连接神经网络。以及集成学习：Stacking Ensemble。

2.3.1 数据集的准备

```
train=pd.read_csv('train_zl.csv')

y=train['SalePrice']

train1=train.drop(['Id','SalePrice','Unnamed: 0'],axis=1)

X=pd.get_dummies(train1).reset_index(drop=True)
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,rand
om_state=123)
```

2.3.2 基础线性回归

```
lm=LinearRegression()

p=lm.fit(X_train,y_train)

pred=lm.predict(X_test)

def benchmark(model):

    pred=model.predict(testset)

    if pred[pred<0].shape[0]>0:

        print('Neg Value')

    rmse=np.sqrt(mean_squared_error(label,pred))

    lrmse=np.sqrt(mean_squared_error(np.log(label),np.log(pred)))



    print('RMSE:',rmse)

    print('LRMSE:',lrmse)
```

return lrmse 得到的结果：0.1262780962215732

2.3.3 数据的 Preprocessing 和 Pipline 创建

lm_model=make_pipeline(RobustScaler(),LinearRegression())

lm_model.fit(X_train,y_train)

补充：jupter notebook 中 忽略警告

import warnings

warnings.filterwarnings('ignore')

2.3.4 RidgeRegression

2.3.4.1 朴素的岭回归

ridge_model=Ridge(alpha=0.1)

ridge_model.fit(X_train,y_train)

benchmark(ridge_model)

得到的结果：0.12658320875064974

2.3.4.2 带有 RobustScaler 的岭回归

```
ridge_model_pipe=make_pipeline(RobustScaler(),Ridge(alpha=0.1
))

ridge_model_pipe.fit(X_train,y_train)

benchmark(ridge_model_pipe)
```

得到的结果：0.12658566764241527

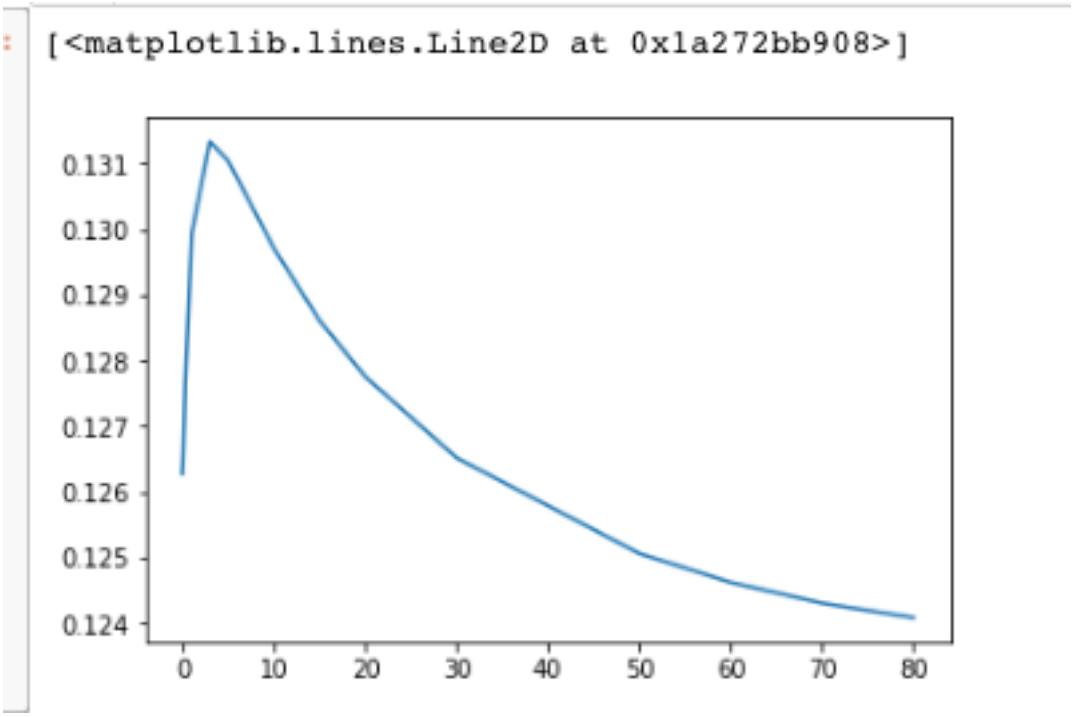
2.3.5 带有 CV 的 Ridge 回归

```
kfolds=KFold(n_splits=10,shuffle=True,random_state=123)

r_alphas=[0.01,0.1,1,3,5,7,10,100]

ridge_model_cv=make_pipeline(RobustScaler(),RidgeCV(alphas=r
_alphas,cv=kfolds))

ridge_model_cv.fit(X_train,y_train)

benchmark(ridge_model_cv)
```

得到的结果：0.12385966794851846

```
r_alphas=[.0001, .0003, .0005, .0007, .0009,

          .01, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 20, 30, 50, 60, 70, 80]
```

```
def ridge_train_test(alpha):


m=make_pipeline(RobustScaler(),RidgeCV(alphas=[alpha],cv=kfol
ds))
    m.fit(X_train,y_train)
    return benchmark1(m,X_test,y_test)
scores=[]
for k in r_alphas:
    scores.append(ridge_train_test(k))


plt.plot(r_alphas,scores)
```



2.3.5.1 RidgeCV 自动筛选参数

```
r_alphas2=np.logspace(-10,2.8,150)

scores=[]

for k in r_alphas2:

    scores.append(ridge_train_test(k))


plt.plot(r_alphas2,scores)
```

补充：可以使用自动化筛选出最优的 Alpha

```
ridge_model2=make_pipeline(RobustScaler(),RidgeCV(
```

```
        alphas=r_alphas2,cv=kfolds

)).fit(X_train,y_train)
```

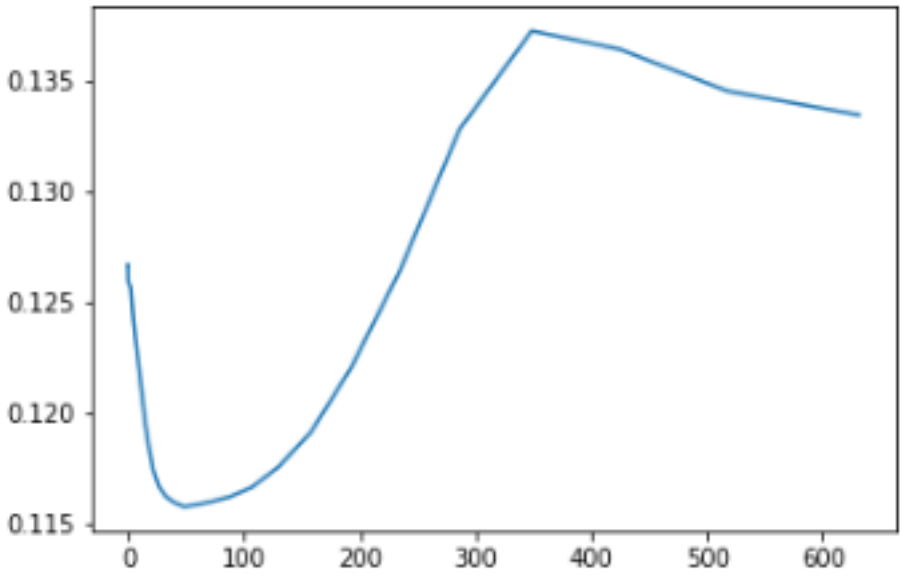
最好的 alpha：

```
ridge_model2.steps[1][1].alpha_
```

2.3.6 Lasso Regression

```
l_alphas=np.logspace(-10,2.8,150)

def lasso_train_test(alpha):


lasso_model=make_pipeline(RobustScaler(),LassoCV(alphas=[alp
ha],cv=kfolds))

        lasso_model.fit(X_train,y_train)

        lrmse=benchmark1(lasso_model,X_test,y_test)

        return lrmse

scores=[]

for k in l_alphas:

        print("Alpha:",k)
```

```
    scores.append(lasso_train_test(k))

plt.plot(l_alphas,scores)
```




```
lasso_model2=make_pipeline(RobustScaler(),LassoCV(

    alphas=l_alphas,cv=kfolds

)).fit(X_train,y_train)

benchmark1(lasso_model2,X_test,y_test)

lasso_model2.steps[1][1].alpha_
```

2.3.7 ElasticNet

```
e_l1ratio=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.85,0.9,0.95,1]

e_alphas=l_alphas
```

```
def elastic_train_test(alpha,l1ratio):


e_model=make_pipeline(RobustScaler(),ElasticNetCV(alphas=[alp
ha],l1_ratio=[l1ratio]))

    e_model.fit(X_train,y_train)

    lrmse=benchmark1(e_model,X_test,y_test)

    return lrmse

elastic_model3=make_pipeline(RobustScaler(),ElasticNetCV(alpha
s=e_alphas,l1_ratio=e_l1ratio)).fit(X_train,y_train)

benchmark1(elastic_model3,X_test,y_test)
```

得到的结果：0.12203176481488646

```
elastic_model3.steps[1][1].alpha_
```

得到的结果：192.5589718453296

```
elastic_model3.steps[1][1].l1_ratio_
```

得到的结果 ：1.0

2.3.8 XGBoost 训练

Xgboost 部分参数解释：

2.3.8.1 eta [default=0.3, alias: learning_rate]

ETA 类似于学习率，将每一步迭代出的特征的残差项进行缩放

range: [0,1]

2.3.8.2 gamma [default=0, alias: min_split_loss]

range: [0,∞]

代表最小的 Loss 阈值，只有高于这个阈值的时候才可以进行树的分裂

2.3.8.3 max_depth [default=6]

最大的树的深度，用来防止过拟合

range: [0,∞]

2.3.8.4 min_child_weight [default=1]

最小的叶子节点权重，通过每一个节点内的数据的二阶导求和后，得到的一个数字跟阈值进行比较，如果小于该值，则放弃分裂；对于回归问题，Loss 函数是叶子节点的方差，二阶导是 1，所以节点中的个数小于某个值的时候就放弃分裂。分类可以根据 loss 的不通，等价于一个二阶导的阈值（纯度函数）

range: [0,∞]

2.3.8.5 subsample [default=1]

样本采样率 range: (0,1]

2.3.8.6 colsample_bytree, colsample_bynode [default=1] -

列采样的一群系数

colsample_bytree 发生在树构建的时候进行采样

colsample_bynode 节点分裂的时候进行新的采样

2.3.8.7 lambda [default=1, alias: reg_lambda]

对于权重的 L2 正则化

2.3.8.8 alpha [default=0, alias: reg_alpha]

权重的 L1 正则化

2.3.8.9 n_estimators

相当于步数

```
xg_reg = xgb.XGBRegressor(
    booster='dart',
    objective='reg:linear',
    colsample_bytree=0.7,
    learning_rate=0.01,
    max_depth=3,
```

```
        n_estimators=3400,
        subsample=0.7,
        nthread=6,
        seed=123)
xg_reg.fit(X_train,y_train)
benchmark1(xg_reg,X_test,y_test)
```

得到的结果：0.10103616336254848

2.3.9 Stacking 集成算法

1）底层算法

```
alphas_alt = np.logspace(-10, 2.8, 150)
ridge = make_pipeline(RobustScaler(), RidgeCV(alphas=alphas_alt, cv=5))
lasso = make_pipeline(RobustScaler(), LassoCV(alphas=alphas_alt, cv=5))
elasticnet = make_pipeline(
        RobustScaler(), ElasticNetCV(alphas=e_alphas, cv=5,
l1_ratio=e_l1ratio))
xgboost = make_pipeline(
        RobustScaler(),
        xgb.XGBRegressor(
                objective='reg:linear',
```

```
        colsample_bytree=0.7,
            learning_rate=0.01,
            max_depth=3,
            n_estimators=3460,
            subsample=0.7,
            reg_alpha=0.00006,
            gamma=0,
            nthread=6,
            scale_pos_weight=1,
            seed=27))
```

2）上层算法

```
stack_alg = StackingCVRegressor(
      regressors=(ridge, lasso, elasticnet, xgboost),
      meta_regressor=xgboost,
      use_features_in_secondary=True)
stackX=np.array(X_train)
stacky=np.array(y_train)
stack_alg.fit(stackX,stacky)
benchmark1(stack_alg,X_test,y_test)
```

得到的结果：0.10218243402850173

2.4 特征工程

基于可视化探索性分析的结果，我们可以进一步对数据进行处理。

训练及评测方法：

```
def TestDataSet(train):
    y = train['SalePrice']
    train2 = train.drop(['SalePrice'], axis=1)
    X = pd.get_dummies(train2)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=123)
    xg_reg = xgb.XGBRegressor(
        objective='reg:linear',
        colsample_bytree=0.7,
        learning_rate=0.01,
        max_depth=3,
        n_estimators=3000,
        subsample=0.7,
        reg_alpha=0.0006,
        nthread=6)
    xg_reg.fit(X_train, y_train)
  benchmark1(xg_reg, X_test, y_test)
```

获取与 SalePrice 的相关系数

x = train.corr()['SalePrice']

1） 去掉极弱相关（|corr|<0.3）

TestDataSet(train.drop(x[abs(x)<0.3].index.tolist(),axis=1))

RMSE: 24493.170974380915 LRMSE: 0.1092756690472248

实验结果标注名，草率去掉低于 0.3 的极弱相关属性就目前的数据集不会改进算法的结果

2） 去掉绝对值小于 0.1 的列

TestDataSet(train.drop(x[abs(x)<0.1].index.tolist(),axis=1))

RMSE: 23953.771426822474 LRMSE: 0.10829674662542016

3） 去掉相关性小于 0.1 的列

TestDataSet(train.drop(x[x<0.1].index.tolist().axis=1))

RMSE: 23408.818211833623 LRMSE: 0.10631323797733613

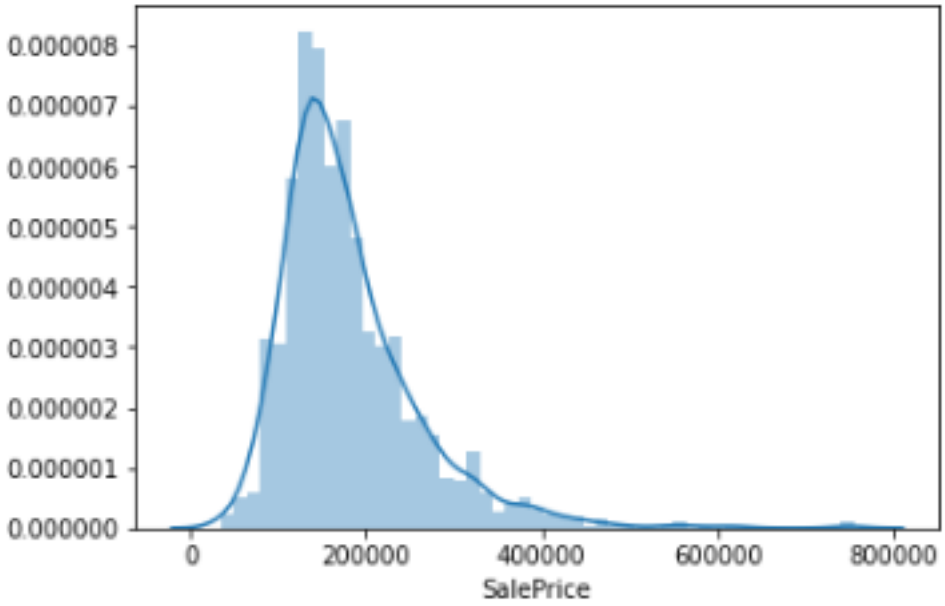
4） 去掉正相关但是小于 0.1 的列

TestDataSet(train.drop(x[(x<0.1) & (x>0)].index.tolist(),axis=1))

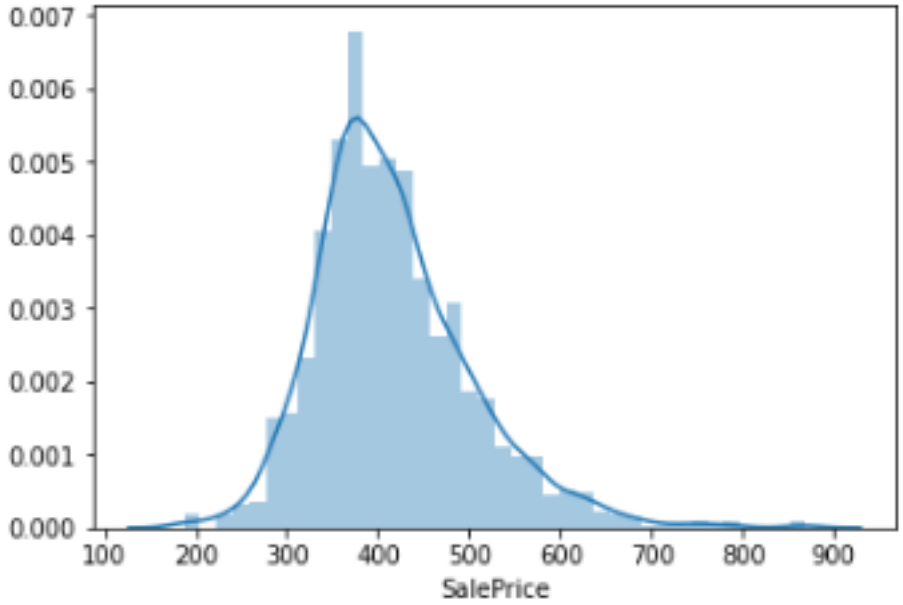
RMSE: 22897.938327661526 LRMSE: 0.10212401106719694

5） Target 的偏度分析

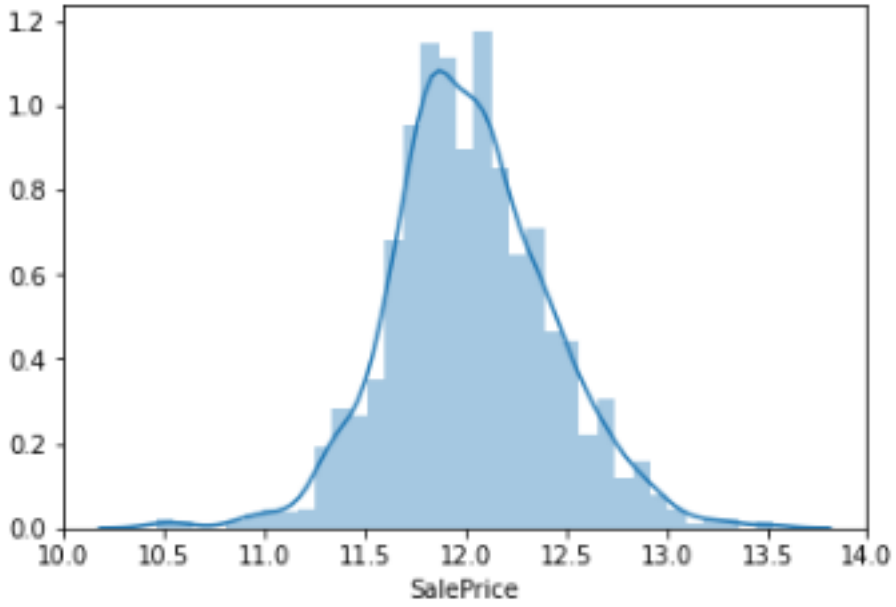
sns.displot(train[‘SalePrice’])



sns.distplot(np.sqrt(train['SalePrice']))



sns.distplot(np.log(train['SalePrice']))



尝试 Target 取 log 进行训练：

```
def TestDataSet2(train):
    y = np.log(train['SalePrice'])
    train2 = train.drop(['Id', 'SalePrice'], axis=1)
    X = pd.get_dummies(train2)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=123)
    xg_reg = xgb.XGBRegressor(
        objective='reg:linear',
        colsample_bytree=0.7,
        learning_rate=0.01,
      max_depth=3,
        n_estimators=3000,
        subsample=0.7,
```

```
            reg_alpha=0.0006,
            nthread=6)
        xg_reg.fit(X_train, y_train)
        benchmark1(xg_reg, X_test, y_test)
```

RMSE: 0.10575709981973441 LRMSE: 0.008893218608158249

尝试 Target 取平方根进行训练:

```
def TestDataSet3(train):
    y = np.sqrt(train['SalePrice'])
    if 'Id' in train.columns:
        train2 = train.drop(['Id', 'SalePrice'], axis=1)
    else:
        train2 = train.drop(['SalePrice'], axis=1)
    X = pd.get_dummies(train2)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=123)
    xg_reg = xgb.XGBRegressor(
        objective='reg:linear',
        colsample_bytree=0.7,
        learning_rate=0.01,
        max_depth=3,
        n_estimators=3000,
        subsample=0.7,
```

```
            reg_alpha=0.0006,
            nthread=6)
        xg_reg.fit(X_train, y_train)
        #benchmark1(xg_reg,X_test,y_test)
        pred = xg_reg.predict(X_test)
        pred = pred**2
        y_test = y_test**2
        print(np.sqrt(mean_squared_error(np.log(pred),
np.log(y_test))))
```

得到的结果：0.10459702326285117

```
TestDataSet3(train.drop(x[(x<0.1) & (x>0)].index.tolist(),axis=1))
```

得到的结果：0.10193706843462444

2.5 神经网络

关于房价预测问题，我们完成了机器学习部分的尝试，下面将使用全连接神经网络进行房价回归预测。

2.5.1 创建全连接模型

```
def create_model(shape_col_num:int):
    model=Sequential()
    model.add(Dense(10,input_dim=shape_col_num,activation='relu'))
```

```
    model.add(Dense(300,activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(50,activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam',loss='mean_squared_error')
return model

model = create_model(X_train.shape[1])
model.summary()
history = model.fit(X_train, y_train, epochs=500, batch_size=32,
verbose=0)
benchmark2(model,X_test,y_test)
plt.plot(history.history['loss'])
model2 = create_model()
history2 =
model2.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=3000,
batch_size=32,verbose=0)
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
```

2.5.2 加入 BatchNormalization 归一化和 SGD 优化器

```
y_train_wan = y_train/10000
```

```
y_test_wan = y_test/10000
def create_model02():
    model = Sequential()
    model.add(Dense(10,input_dim=X_train.shape[1]))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dense(300))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
#     model.add(Dropout0.2) ## 加 BatchNormlization 时，最好不要加 Dropout 层
    model.add(Dense(50,activation='relu'))
    model.add(Dense(1))
    sgd = optimizers.SGD(lr=0.001,decay=1e-6,momentum=0.9,nesterov=True)
#     model.compile(optimizer=sgd,loss='mean_squared_error')
return model

model = create_model02()
sgd = optimizers.SGD(lr = 0.000001,decay=1e-6,momentum=0.9,nesterov=True)
model.compile(optimizer=sgd,loss='mean_squared_error')
tmp =
```

```
model.fit(X_train,y_train_wan,validation_data=(X_test,y_test_wan),epo
chs=500,batch_size=32,verbose=1)
history02 = tmp
history02.history['loss'] += tmp.history['loss']
history02.history['val_loss'] += tmp.history['val_loss']
plt.plot(history02.history['loss'])
plt.plot(history02.history['val_loss'])
```

2.5.3 RobustScaler 归一化

```
transformer = RobustScaler().fit(X_train)
X_train_norm = transformer.transform(X_train)
X_test_norm = transformer.transform(X_test)
def create_model04():
    model = Sequential()
    model.add(Dense(10,input_dim=X_train.shape[1]))
    model.add(Activation('relu'))
    model.add(Dense(300))
    model.add(Activation('relu'))
    model.add(Dense(50,activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam',loss='mean_squared_error')
    return model
```

```
model = create_model04()
tmp =
model.fit(X_train_norm,y_train,validation_data=(X_test_norm,y_test),e
pochs=2020,verbose=0,batch_size=32)
plt.plot(history04.history['loss'])
plt.plot(history04.history['val_loss'])
benchmark3(model,X_test_norm,y_test)
```

2.5.4 Target 进行归一化和反归一化

```
transformer_y = RobustScaler().fit(np.asanyarray(y_train).reshape(-1, 1))
y_train_norm =
transformer_y.transform(np.asanyarray(y_train).reshape(-1, 1))
y_test_norm =
transformer_y.transform(np.asanyarray(y_test).reshape(-1, 1))
transformer_x = RobustScaler().fit(X_train)
X_train_norm = transformer_x.transform(X_train)
X_test_norm = transformer_x.transform(X_test)
model05 = create_model04()
history05 =
model05.fit(X_train_norm,y_train_norm,validation_data=(X_test_norm,
y_test_norm),verbose=0,batch_size=32,epochs=3000)
pred_norm = model05.predict(X_test_norm)
```

```
pred = transformer_y.inverse_transform(pred_norm).flatten()
np.std(pred-y_test)
np.std(np.log(pred)-np.log(y_test))
```

3 开发工作总结

3.1 在开发过程中学到什么

黄老师的“道法术器“体系中，截止到今日我一直在努力学习关于AI职业道路中的”器“这一层。包括数学基础：概率论，线性代数和微积分课程中与 AI 相关的基础课程，基础不够扎实需要加倍努力。Python 语言编程中学习了基础语法并贯穿了整个课程实践。数据分析基础工具：Numpy，Pandas，Seaborn,Matplotlib 等常用分析工具。机器学习体系中主要学习了决策树，随机森林，GBDT，线性回归，逻辑回归等基础算法。神经网络主要学习了全连接神经网络，CNN 网络。

3.2 希望在未来的开发过程中学到什么

希望黄老师可以再帮我们整理一下 SVM 模型，HMM 模型。关于深度学习可以在扩展一下目前应用较为广泛的神经网络（小白无法列举）。

3.3 开发学习过程中优点和缺点

缺点：目前知识还处在各个孤立的点的阶段，还没办法进行融会贯通，举一反三。

3.4 还希望开发过程中在哪方便做更多尝试

基础算法掌握较为薄弱，希望更多的区尝试推导和实践。

4 经验和教训

4.1 项目中习得经验

百看不如一练。静心去实践和总结

4.2 项目中习得教训

学习新知识，一定要系统的，长时间的，重复的去看书，练习和复习。

5 建议和展望

5.1 对于今后项目开发工作的建议

尽量少用贪心算法去思考项目，而是动态规划算法去理解和实践。

5.2 其他建议

5.3 展望

首先感谢黄老师的辛苦教导。作为已经工作几年时间的我来说，中途转行，风险还是很大的。几乎从零基础开始学习 AI 相关课程，从刚开始的兴奋好奇但又无从下手，到丧失信心几乎放弃，再到面纱慢慢清晰重燃斗志。未来一段时间要加强基础理论学习，夯实基础，慢慢

填充体系中薄弱的知识点，逐步去探索机器学习中的“法”和“术“以最快的时间跨过 AI 行业的门槛，并开始新的职业道路。唯有坚持才有希望。