二手车交易价格预测

组　　员：曾　博 2019180060

组　　员：孙文利 2019180074

组　　员：刘德峰 2019180044

北京邮电大学

论文题目	二手车价格预测		
论文类型	工程项目	选题来源	阿里天池

1、研究背景和内容

1.1、研究内容简介

本项研究的题目为天池数据挖掘竞赛二手车交易价格预测。目的是通过二手车的相关特征，以此尽可能准确的预测出二手车的价格。通过该研究，加深对机器学习、python、数据分析方法、构造特征工程，建立学习模型，调整参数，融合模型的理解和熟练程度。

1.2、研究现状

20 世纪 90 年代，随着数据库系统的广泛应用和网络技术的高速发展，数据库技术也进入一个全新的阶段，即从过去仅管理一些简单数据发展到管理由各种计算机所产生的图形、图像、音频、视频、电子档案、Web 页面等多种类型的复杂数据，并且数据量也越来越大。

数据库在给我们提供丰富信息的同时，也体现出明显的海量信息特征。信息爆炸时代，海量信息给人们带来许多负面影响，最主要的就是有效信息难以提炼，过多无用的信息必然会产生信息距离（信息状态转移距离）是对一个事物信息状态转移所遇到障碍的测度，简称 DIST 或 DIT）和有用知识的丢失。这也就是约翰·内斯伯特(John Nalsbert)称为的“信息丰富而知识贫乏”窘境。

因此，人们迫切希望能对海量数据进行深入分析，发现并提取隐藏在其中的信息，以更好地利用这些数据。但仅以数据库系统的录入、查询、统计等功能，无法发现数据中存在的关系和规则，无法根据现有的数据预测未来的发展趋势，更缺乏挖掘数据背后隐藏知识的手段。正是在这样的条件下，数据挖掘技术应运而生。

对于给定的原始数据，我们需要通过一系列的方法对数据进行加工和处理，找到数据之间存在的关系，由此引申出本次研究的内容，需要利用已知的二手车数据通过数据分析、清洗、预处理、建立模型、评估模型等一系列的操作，最终可以实现项目落地，即将此模型应用的不同的数据集上。

1.3 研究内容与创新工作

1.3.1 研究内容

本文立足于我国二手车市场，通过给定的数据集，尝试使用人工神经网络方法,在字段一致的情况下，对二手车价格进行预测。此次比赛为回归问题，可以采用的算法有：感知机模型、决策树模型和神经网络，隐马尔可夫模型，条件随机场，确定数据挖掘的方向。其中隐马尔可夫模型和条件随机场主要用于分词，这里我们不打算采用，我们采用了线性回归模型、提升树模型和神经网络对该项目进行挖掘

1.3.2 创新工作

由于数据挖掘的项目已经有非常多成功的案例了，用的机器学习算法，数据预处理方法都已经非常完善了，本次项目的主要创新点在于使用了多种机器学习算法和深度学习神经网络的算法来解决回归问题，并对模型进行了融合，以此来提高模型的准确率。

1.3.3 研究思路

因为是第一次打比赛，之前也只是简单了解过一些基础的机器学习算法，所以对比赛项目的要如何去做不是很了解，随着时间线的推进，在对本次项目的深入了解之后，我总结了以下的研究思路：

1.根据赛题确定项目是属于哪种需要解决的问题，一般问题会归结于三类：回归问题、分类问题和聚类问题。

2.进行探索性数据分析，通过观察、熟悉并了解数据集，然后通过绘图进一步了解变量之间的相互关系以及变量与预测值之间的存在关系。

3.特征工程，对于特征进行进一步分析，并对于数据进行处理。

4.建模与调参根据数据的实际情况选用合适的机器学习或者深度学习算法建模。评估模型的准确率。

5.对于多种调参完成的模型进行模型融合。主要目的是为了提升模型的泛化性能。

2、二手车交易价格预测

2.1 赛题理解与数据分析

2.1.1 赛题概况

赛题以预测二手车的交易价格为任务，数据集报名后可见并可下载，该数据来自某交易平台的二手车交易记录，总数据量超过 40w，包含 31 列变量信息，其中 15 列为匿名变量。为了保证比赛的公平性，将会从中抽取 15 万条作为训练集，5 万条作为测试集 A，5 万条作为测试集 B，同时会对 name、model、brand 和 regionCode 等信息进行脱敏。

2.1.2 数据概况

通过 pandas 库来得到数据集的基本信息，了解列的性质会有助于我们对于数据的理解和后续分析，具体列信息如下图所示。

train.csv

- SaleID - 销售样本ID
- name - 汽车编码
- regDate - 汽车注册时间
- model - 车型编码
- brand - 品牌
- bodyType - 车身类型
- fuelType - 燃油类型
- gearbox - 变速箱
- power - 汽车功率
- kilometer - 汽车行驶公里
- notRepairedDamage - 汽车有尚未修复的损坏
- regionCode - 看车地区编码
- seller - 销售方
- offerType - 报价类型
- creatDate - 广告发布时间
- price - 汽车价格
- v_0', 'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6', 'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13','v_14' 【匿名特征，包含v0-14在内15个匿名特征】

数字全都脱敏处理，都为label encoding形式，即数字形式

2.1.3 预测指标

一般来说分类和回归问题的评价指标有如下一些形式：

1.分类算法常见的评估指标如下：

对于二类分类器/分类算法，评价指标主要有 accuracy， [Precision，Recall，F-score，Pr 曲线], ROC-AUC 曲线。

对于多类分类器/分类算法，评价指标主要有 accuracy， [宏平均和微平均，F-score]。

2.对于回归预测类常见的评估指标如下：

平均绝对误差（Mean Absolute Error，MAE），均方误差（Mean Squared Error，MSE），平均绝对百分误差(Mean Absolute Percentage Error，MAPE)，均方根误差(Root Mean Squared Error)，R2（R-Square）

平均绝对误差 平均绝对误差（Mean Absolute Error， MAE）:平均绝对误差，其能更好地反映预测值与真实值误差的实际情况，其计算公式如下：

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

均方误差 均方误差（Mean Squared Error， MSE）,均方误差,其计算公式为：

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

R2（R-Square）的公式为：

残差平方和:

$$SS_{res} = \sum (y_i - \hat{y}_i)^2$$

总平均值:

$$SS_{tot} = \sum (y_i - \bar{y}_i)^2$$

R2 表达式为:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

R2 用于度量因变量的变异中可由自变量解释部分所占的比例，取值范围是 0~1，R2 越接近 1,表明回归平方和占总平方和的比例越大,回归线与各观测点越接近，用 x 的变化来解释 y 值变化的部分就越多,回归的拟合程度就越好。所以 R2 也称为拟合优度（Goodness of Fit）的统计量。

本赛题的评价标准为 MAE(Mean Absolute Error):

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

2.1.4 分析赛题

1.此题为传统的数据挖掘问题，通过数据科学以及机器学习深度学习的办法来进行建模得到结果。

2.此题是一个典型的回归问题。

3.主要应用 xgb、lgb、catboost，以及 pandas、numpy、matplotlib、seabon、sklearn、keras 等等数据挖掘常用库或者框架来进行数据挖掘任务。

4.通过 EDA 来挖掘数据的联系和自我熟悉数据。

代码如下:

1.简略观察数据

```
Train_data = pd.read_csv('used_car_train_20200313.csv', sep=' ',encoding="utf-8")
Test_data = pd.read_csv('used_car_testA_20200313.csv', sep=' ',encoding="utf-8")

print('Train data shape:', Train_data.shape)
print('TestA data shape:', Test_data.shape)

print(Train_data.head())
print(Test_data.head())
```

```
Train data shape: (150000, 31)
TestA data shape: (50000, 30)
   SaleID    name   regDate  model  ...      v_11      v_12      v_13      v_14
0       0     736  20040402   30.0  ...  2.804097 -2.420821  0.795292  0.914762
1       1    2262  20030301   40.0  ...  2.096338 -1.030483 -1.722674  0.245522
2       2   14874  20040403  115.0  ...  1.803559  1.565330 -0.832687 -0.229963
3       3   71865  19960908  109.0  ...  1.285940 -0.501868 -2.438353 -0.478699
4       4  111080  20120103  110.0  ...  0.910783  0.931110  2.834518  1.923482

[5 rows x 31 columns]
   SaleID    name   regDate  model  ...      v_11      v_12      v_13      v_14
0  150000   66932  20111212  222.0  ... -0.854626  4.800151  0.620011 -3.664654
1  150001  174960  19990211   19.0  ... -0.729039 -3.796107 -1.541230 -0.757055
2  150002    5356  20090304   82.0  ...  1.001106  0.826562  0.138226  0.754033
3  150003   50688  20100405    0.0  ...  0.505493  1.870379  0.366038  1.312775
4  150004  161428  19970703   26.0  ... -0.673236 -3.197685 -0.025678 -0.101290

[5 rows x 30 columns]
```

describe 中有每列的统计量，个数 count、平均值 mean、方差 std、最小值 min、中位数 25% 50% 75% 、以及最大值 看这个信息主要是瞬间掌握数据的大概的范围以及每个值的异常值的判断，比如有的时候会发现 999 9999 -1 等值这些其实都是 nan 的另外一种表达方式，有的时候需要注意下。

```
              SaleID           name  ...           v_13           v_14
count  150000.000000  150000.000000  ...  150000.000000  150000.000000
mean    74999.500000   68349.172873  ...       0.000313      -0.000688
std     43301.414527   61103.875095  ...       1.288988       1.038685
min         0.000000       0.000000  ...      -4.153899      -6.546556
25%     37499.750000   11156.000000  ...      -1.057789      -0.437034
50%     74999.500000   51638.000000  ...      -0.036245       0.141246
75%    112499.250000  118841.250000  ...       0.942813       0.680378
max    149999.000000  196812.000000  ...      11.147669       8.658418

[8 rows x 30 columns]
```

	SaleID	name	...	v_13	v_14
count	50000.000000	50000.000000	...	50000.000000	50000.000000
mean	174999.500000	68542.223280	...	-0.003147	0.001516
std	14433.901067	61052.808133	...	1.286597	1.027360
min	150000.000000	0.000000	...	-4.123333	-6.112667
25%	162499.750000	11203.500000	...	-1.060428	-0.437920
50%	174999.500000	52248.500000	...	-0.035956	0.138799
75%	187499.250000	118856.500000	...	0.941469	0.681163
max	199999.000000	196805.000000	...	5.913273	2.624622

[8 rows x 29 columns]

info 通过 info 来了解数据每列的 type，有助于了解是否存在除了 nan 以外的特殊符号异常。

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 31 columns):
SaleID               150000 non-null int64
name                 150000 non-null int64
regDate              150000 non-null int64
model                149999 non-null float64
brand                150000 non-null int64
bodyType             145494 non-null float64
fuelType             141320 non-null float64
gearbox              144019 non-null float64
power                150000 non-null int64
kilometer            150000 non-null float64
notRepairedDamage    150000 non-null object
regionCode           150000 non-null int64
seller               150000 non-null int64
offerType            150000 non-null int64
creatDate            150000 non-null int64
price                150000 non-null int64
v_0                  150000 non-null float64
v_1                  150000 non-null float64
v_2                  150000 non-null float64
v_3                  150000 non-null float64
v_4                  150000 non-null float64
v_5                  150000 non-null float64
v_6                  150000 non-null float64
v_7                  150000 non-null float64
v_8                  150000 non-null float64
v_9                  150000 non-null float64
v_10                 150000 non-null float64
v_11                 150000 non-null float64
v_12                 150000 non-null float64
v_13                 150000 non-null float64
v_14                 150000 non-null float64
dtypes: float64(20), int64(10), object(1)
memory usage: 35.5+ MB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 30 columns):
SaleID               50000 non-null int64
name                 50000 non-null int64
regDate              50000 non-null int64
model                50000 non-null float64
brand                50000 non-null int64
bodyType             48587 non-null float64
fuelType             47107 non-null float64
gearbox              48090 non-null float64
power                50000 non-null int64
kilometer            50000 non-null float64
notRepairedDamage    50000 non-null object
regionCode           50000 non-null int64
seller               50000 non-null int64
offerType            50000 non-null int64
creatDate            50000 non-null int64
v_0                  50000 non-null float64
v_1                  50000 non-null float64
v_2                  50000 non-null float64
v_3                  50000 non-null float64
v_4                  50000 non-null float64
v_5                  50000 non-null float64
v_6                  50000 non-null float64
v_7                  50000 non-null float64
v_8                  50000 non-null float64
v_9                  50000 non-null float64
v_10                 50000 non-null float64
v_11                 50000 non-null float64
v_12                 50000 non-null float64
v_13                 50000 non-null float64
v_14                 50000 non-null float64
dtypes: float64(20), int64(9), object(1)
memory usage: 11.4+ MB
```

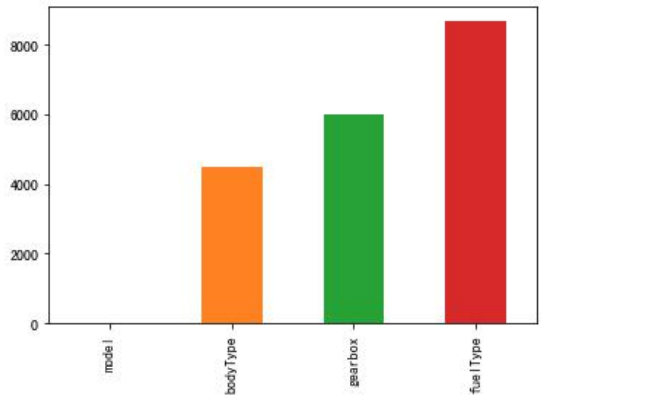
2.处理数据异常值

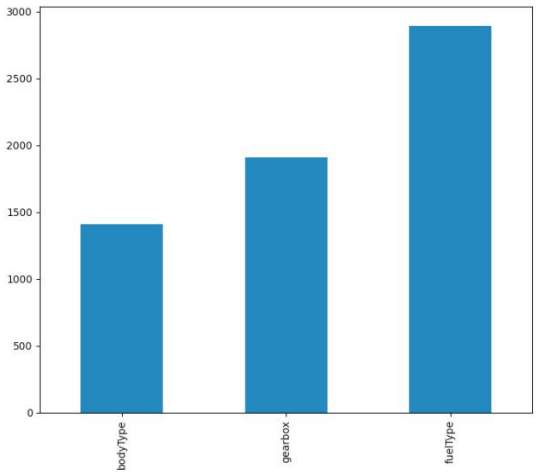
根据上面的 info 信息判断数据的缺失和异常情况，用 matplotlib 绘图，通过图像可以很直观的了解哪些列存在 “nan”，并可以把 nan 的个数打印，主要的目的在于 nan 存在的个数是否真的很大，如果很小一般选择填充，如果使用 lgb 等树模型可以直接空缺，让树自己去优化，但如果 nan 存在的过多、可以考虑删掉。

```
# nan可视化
missing = Train_data.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()
plt.show()
```

```
# nan可视化
missing = Test_data.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()
plt.show()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff019312b38>
```





通过上面的 info 信息可以发现除了 notRepairedDamage 为 object 类型其他都为数字 这里我们把他的几个不同的值都进行显示就知道了。

```
print(Train_data['notRepairedDamage'].value_counts())
Train_data['notRepairedDamage'].replace('-', np.nan, inplace=True)
print(Train_data['notRepairedDamage'].value_counts())

print(Test_data['notRepairedDamage'].value_counts())
Test_data['notRepairedDamage'].replace('-', np.nan, inplace=True)
print(Test_data['notRepairedDamage'].value_counts())
```

从代码运行结果上可以看出来 ‘ - ’ 也为空缺值，因为很多模型对 nan 有直接的处理，这里我们先不做处理，先替换成 nan，将训练集和测试集的 notRepairedDamage 列全部替换。

```
0.0    111361
-       24324
1.0     14315
Name: notRepairedDamage, dtype: int64
0.0    111361
1.0     14315
Name: notRepairedDamage, dtype: int64
0.0    37249
-       8031
1.0     4720
Name: notRepairedDamage, dtype: int64
0.0    37249
1.0     4720
Name: notRepairedDamage, dtype: int64
```

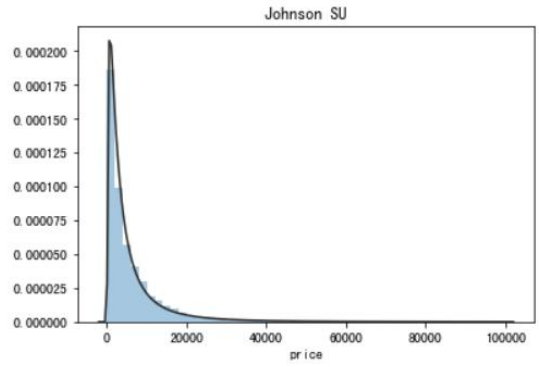
由数据可以知道，seller 和 offerType 两个类别特征严重倾斜，一般不会对预测有什么帮助，故这边先删掉，当然你也可以继续挖掘，但是一般意义不大

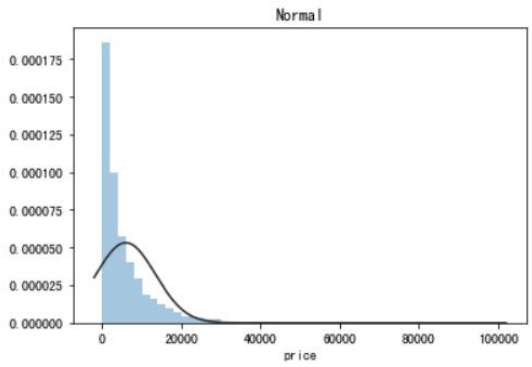
```
print(Train_data["seller"].value_counts())
print(Train_data["offerType"].value_counts())

del Train_data["seller"]
del Train_data["offerType"]
del Test_data["seller"]
del Test_data["offerType"]
```

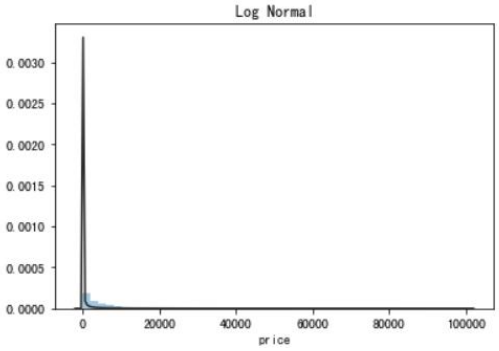
```
0    149999
1         1
Name: seller, dtype: int64
0    150000
Name: offerType, dtype: int64
```

3.了解预测值的分布

```
# 总体分布概况（无界约翰逊分布等）
y = Train_data['price']
plt.figure(1); plt.title('Johnson SU')
sns.distplot(y, kde=False, fit=st.johnsonsu)
plt.figure(2); plt.title('Normal')
sns.distplot(y, kde=False, fit=st.norm)
plt.figure(3); plt.title('Log Normal')
sns.distplot(y, kde=False, fit=st.lognorm)
```



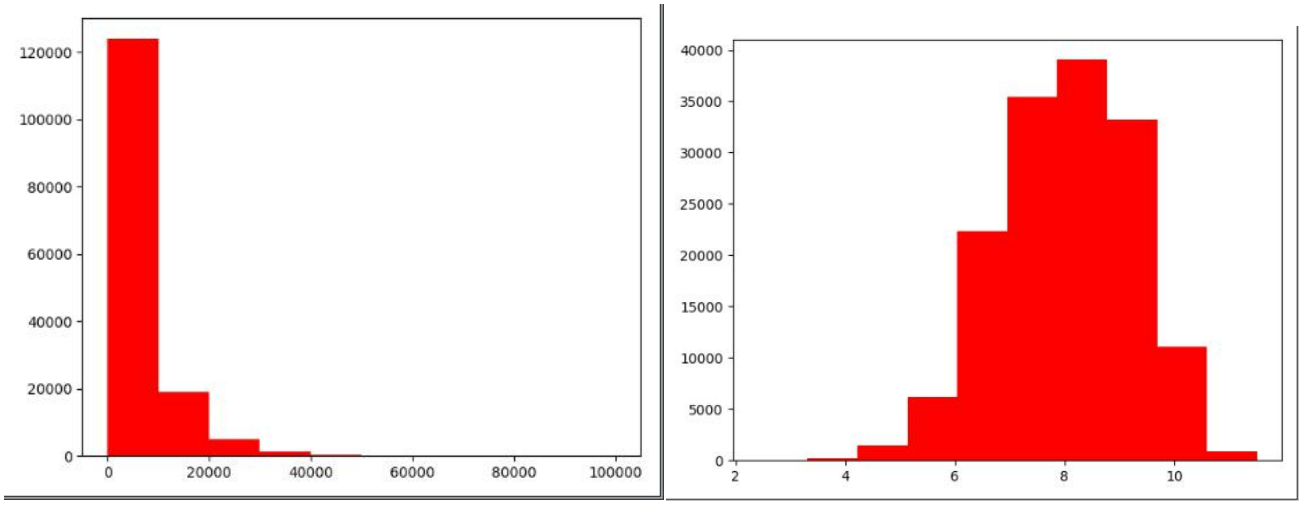




由图可知，价格不服从正态分布，所以在进行回归之前，它必须进行转换。虽然对数变换做得很好，但最佳拟合是无界约翰逊分布。

查看价格的频数，可以看到大于 20000 得值极少，其实这里也可以把这些当作特殊得值（异常值）直接用填充或者删掉，再进行操作。这里可以看到 price 列进行 log 变换之后的分布较均匀，可以进行 log 变换进行预测，这也是预测问题常用的 trick。

```
# 查看预测值的具体频数
plt.hist(Train_data['price'], orientation='vertical', histtype='bar',
         color='red')
plt.show()

# log变换之后的分布较均匀，可以进行log变换进行预测，这也是预测问题常用的trick
plt.hist(np.log(Train_data['price']), orientation='vertical', histtype='bar',
         color='red')
plt.show()
```



4.特征分为类别特征和数字特征，并对类别特征查看 unique 分布

按照特征的取值类型不同，分为类别特征和数字特征，类别特征取值为离散值，数字特征取值为连续值

```
# 分离label即预测值
Y_train = Train_data['price']

numeric_features = ['power', 'kilometer', 'v_0', 'v_1', 'v_2', 'v_3',
                    'v_4', 'v_5', 'v_6', 'v_7', 'v_8', 'v_9', 'v_10',
                    'v_11', 'v_12', 'v_13', 'v_14']

categorical_features = ['name', 'model', 'brand', 'bodyType', 'fuelType',
                        'gearbox', 'notRepairedDamage', 'regionCode']
numeric_features.append('price')
```

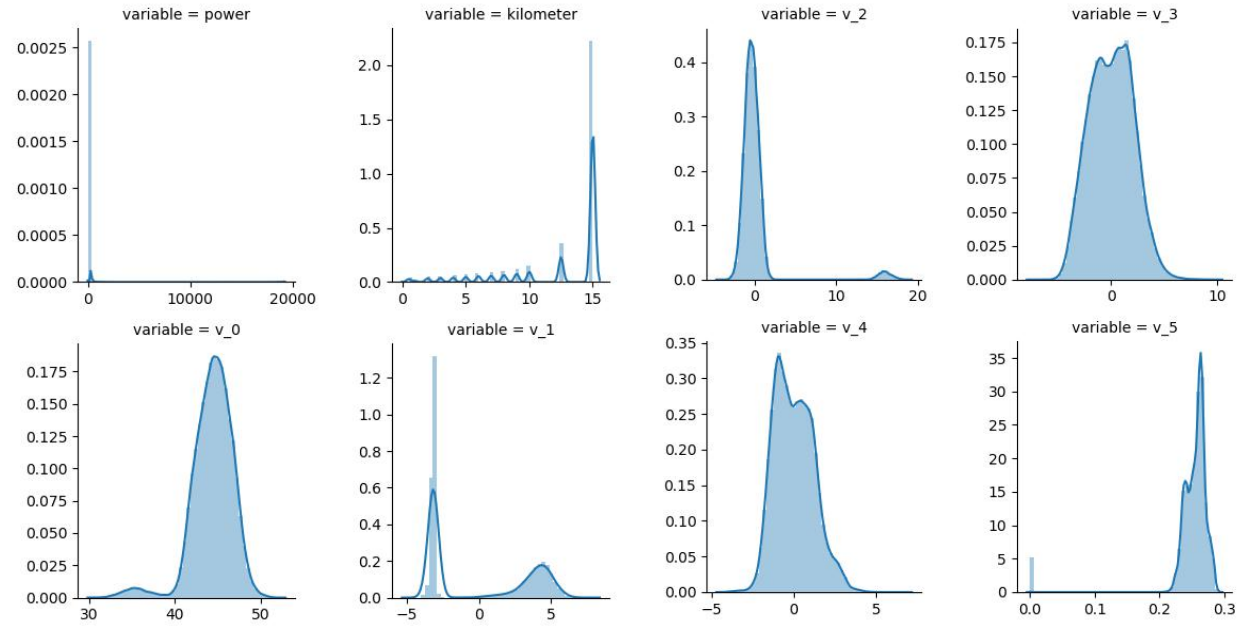
分析所有数字特征跟标签特征 price 的相关性。用 matplotlib 绘图可以清晰的看出，每个特征与标签特征到底是正相关还是负相关。

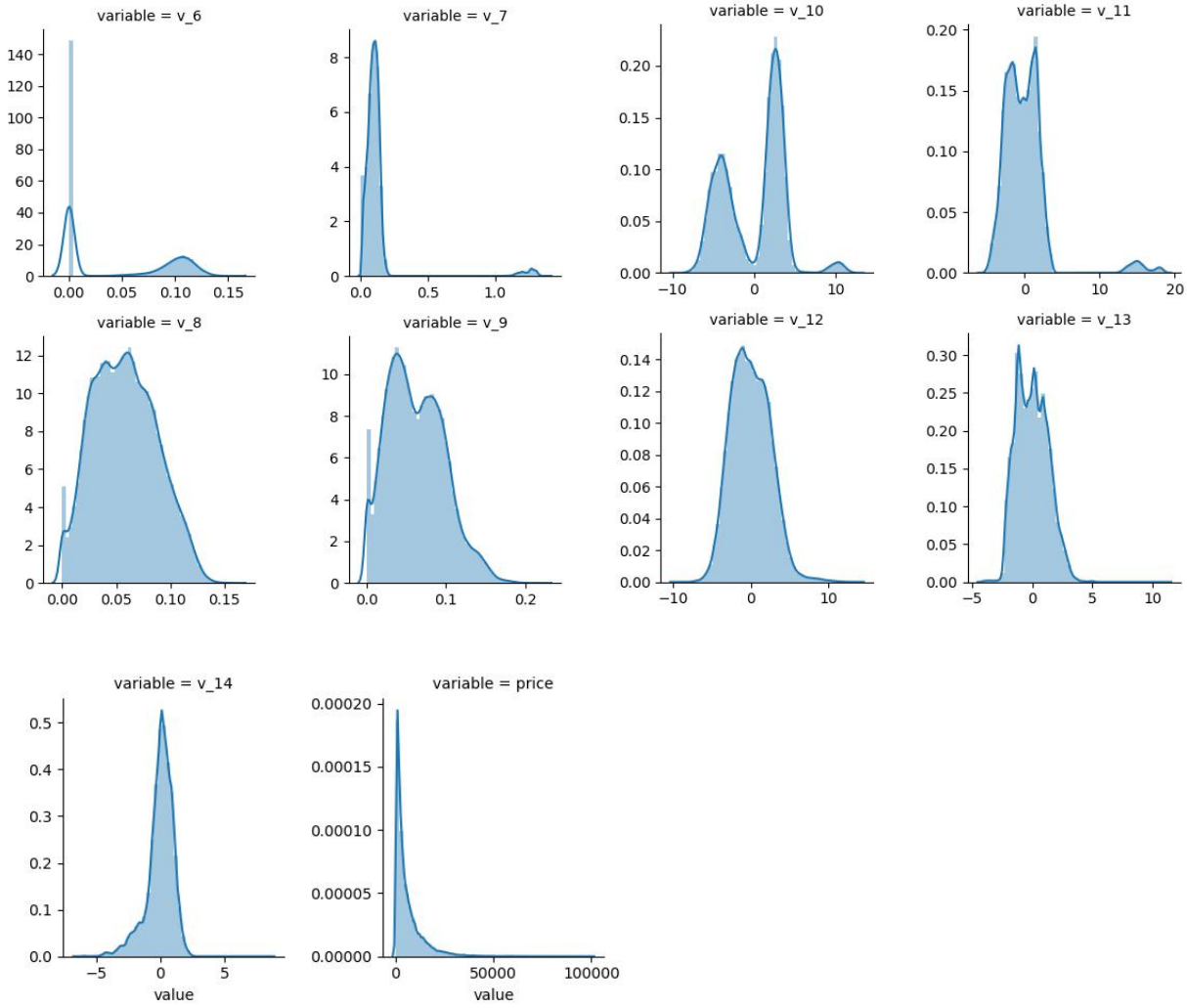
```
price_numeric = Train_data[numeric_features]
correlation = price_numeric.corr()
print(correlation['price'].sort_values(ascending=False), '\n')

f, ax = plt.subplots(figsize=(7, 7))

plt.title('Correlation of Numeric Features with Price', y=1, size=16)

sns.heatmap(correlation, square=True,  vmax=0.8)

del price_numeric['price']
```

将每个数字特征的分布进行可视化

```
# 每个数字特征得分布可视化
f = pd.melt(Train_data, value_vars=numeric_features)
g = sns.FacetGrid(f, col="variable",  col_wrap=2, sharex=False, sharey=False)
g = g.map(sns.distplot, "value")
```

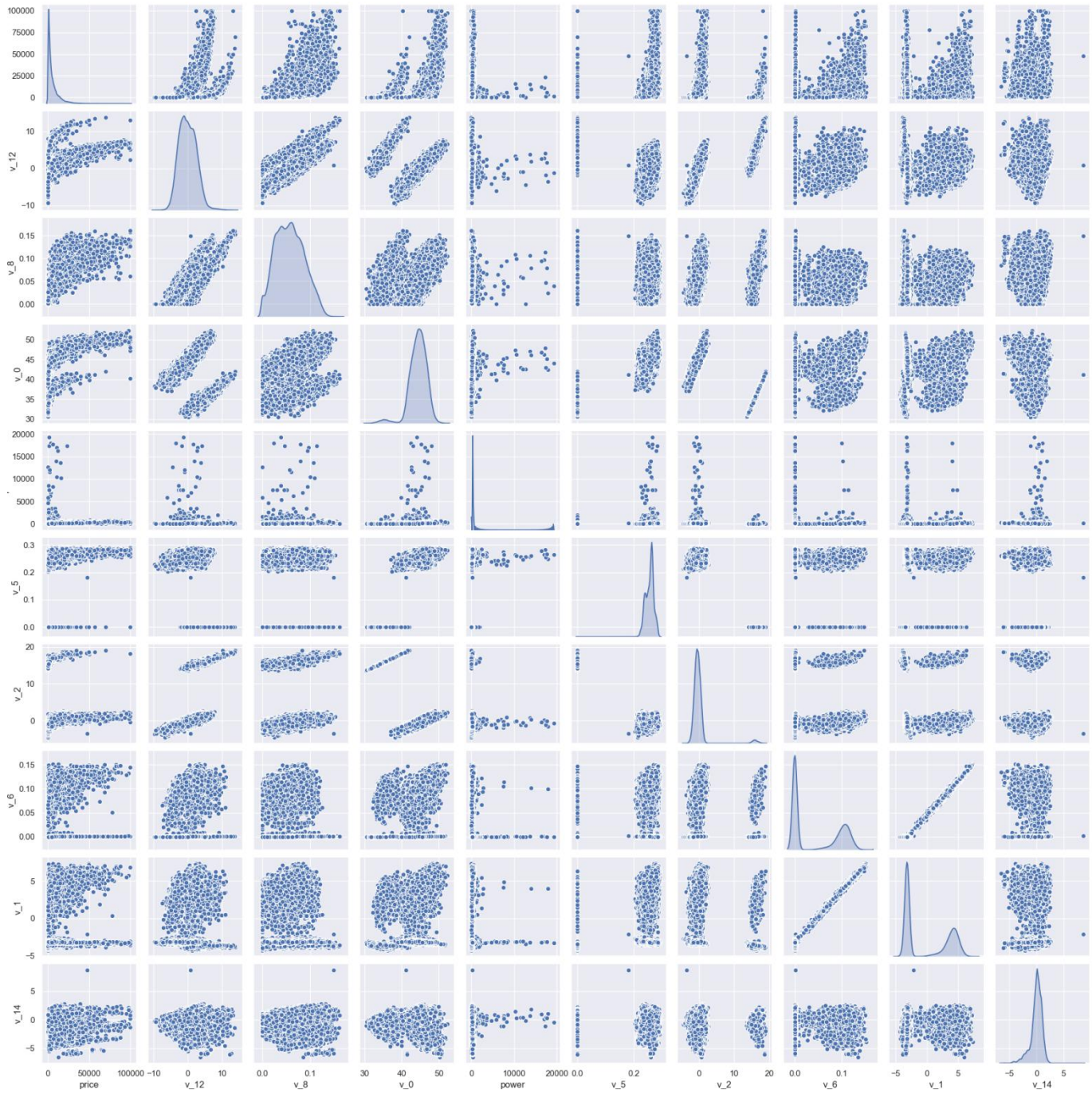
从图中可以看出匿名特征分布相对均匀。





数字特征相互之间的关系可视化

```
columns = ['price', 'v_12', 'v_8', 'v_0', 'power', 'v_5',  'v_2', 'v_6',
           'v_1', 'v_14']
sns.pairplot(Train_data[columns], size=2, kind='scatter', diag_kind='kde')
plt.show()
```



2.2 数据预处理

1.box-cox 变换目标值“price”，解决长尾分布。

```
一、预测值处理，处理目标值长尾分布的问题
"""
Train_data['price'] = np.log1p(Train_data['price'])

# 合并方便后面的操作
df = pd.concat([Train_data, Test_data], ignore_index=True)
```

2.异常值处理，按照题目要求“power”位于 0～600，因此把“power”>600 的值截断至 600，把 "notRepairedDamage"的非数值的值替换为 np.nan，让模型自行处理。

```
# 异常值就目前初步判断，只有notRepairedDamage的值有问题，还有题目规定了范围的power。处理一下
df['power'] = df['power'].map(lambda x: 600 if x > 600 else x)
df['notRepairedDamage'] = df['notRepairedDamage'].astype('str').apply(lambda x:
                                                    x if x != '-' else None).astype('float32')
```

3.缺失值处理，分类特征填充众数，连续特征填充平均值。由上面的数据分析可以知道，fuelType，gearbox，bodyType，model 的众数为 0。

```
# 以下特征全部填充众数
df['fuelType'] = df['fuelType'].fillna(0)
df['gearbox'] = df['gearbox'].fillna(0)
df['bodyType'] = df['bodyType'].fillna(0)
df['model'] = df['model'].fillna(0)
```

4.删除与目标值无关的列，例如“SaleID”，“name”。这里可以挖掘一下“name”的频度作为新的特征。

```
# SaleID肯定没用，但是我们可以用来统计别的特征的group数量
# name一般没什么好挖掘的，不过同名的好像不少，可以挖掘一下
df['name_count'] = df.groupby(['name'])['SaleID'].transform('count')
del df['name']
```

5.异常点处理，删除训练集特有的数据，例如删除“seller”==1 的值。

6.其他特别处理，把取值无变化的列删掉。像 seller 跟 offerType 这种倾斜太严重的列对数据挖掘没有什么作用，故删去。

```
# seller有一个特殊值，训练集特有测试集没有，把它删除掉
df.drop(df[df['seller'] == 1].index, inplace=True)
del df['offerType']
del df['seller']
```

2.3 特征工程

1.地区时间类

首先对通过上面的数据分析可以看到汽车注册时间、广告发布时间和看车地区编码都包含了很多隐藏信息可以挖掘，“regDate”，“creatDate”可以获得年、月、日等一系列的新特征，然后做差可以获得使用年长和使用天数这些新特征。

```python
def date_process(x):
    """
    切分字符串
    :param x: 字符串
    :return:
    """
    year = int(str(x)[:4])
    month = int(str(x)[4:6])
    day = int(str(x)[6:8])

    if month < 1:
        month = 1

    date = datetime(year, month, day)
    return date


df['regDate'] = df['regDate'].apply(date_process)
df['creatDate'] = df['creatDate'].apply(date_process)
df['regDate_year'] = df['regDate'].dt.year
df['regDate_month'] = df['regDate'].dt.month
df['regDate_day'] = df['regDate'].dt.day
df['creatDate_year'] = df['creatDate'].dt.year
df['creatDate_month'] = df['creatDate'].dt.month
df['creatDate_day'] = df['creatDate'].dt.day
df['car_age_day'] = (df['creatDate'] - df['regDate']).dt.days
df['car_age_year'] = round(df['car_age_day'] / 365, 1)
```

“regionCode”没有保留。因为尝试了一系列方法，并且发现了可能会泄漏“price”，因此最终没保留该特征。

```python
# 地区
df['regionCode_count'] = df.groupby(['regionCode'])['SaleID'].transform('count')
df['city'] = df['regionCode'].apply(lambda x: str(x)[:2])
```

2.对可分类的连续特征进行分桶，kilometer 是已经分桶了。然后对"power"和"model"进行了分桶。

数据分桶 以 power 为例，这时候我们的缺失值也进桶了，为什么要做数据分桶呢，原因有很多:

(1). 离散后稀疏向量内积乘法运算速度更快，计算结果也方便存储，容易扩展;

(2). 离散后的特征对异常值更具鲁棒性，如 age>30 为 1 否则为 0，对于年龄为 200 的也不会对模型造成很大的干扰;

(3). LR 属于广义线性模型，表达能力有限，经过离散化后，每个变量有单独的权重，这相

当于引入了非线性，能够提升模型的表达能力，加大拟合；

(4). 离散后特征可以进行特征交叉，提升表达能力，由 M+N 个变量编程 M*N 个变量，进一步引入非线形，提升了表达能力；

(5). 特征离散后模型更稳定，如用户年龄区间，不会因为用户年龄长了一岁就变化

当然还有很多原因, LightGBM 在改进 XGBoost 时就增加了数据分桶, 增强了模型的泛化性能。

```
# 2、分类特征
# 对可分类的连续特征进行分桶，kilometer是已经分桶了
bin = [i*10 for i in range(31)]
df['power_bin'] = pd.cut(df['power'], bin, labels=False)
bucket_power = df[['power_bin', 'power']].head()


bin = [i*10 for i in range(24)]
df['model_bin'] = pd.cut(df['model'], bin, labels=False)
bucket_model = df[['model_bin', 'model']].head()
```

使用分类特征“brand”、“model”、“kilometer”、“bodyType”、“fuelType”与“price”、“days”、“power”进行特征交叉。交叉主要获得的是后者的总数、方差、最大值、最小值、平均数、众数、峰度等等。

这里可以获得非常多的新特征，挑选的时候，直接使用 lightgbm 帮我们去选择特征，一组组的放进去，最终保留了以下特征。（注意：这里使用 1/4 的训练集进行挑选可以帮助我们更快的锁定真正 Work 的特征）

'model_power_sum','model_power_std',

'model_power_median', 'model_power_max',

'brand_price_max', 'brand_price_median',

'brand_price_sum', 'brand_price_std',

'model_days_sum','model_days_std',

'model_days_median', 'model_days_max',

'model_amount','model_price_max',

'model_price_median','model_price_min',

'model_price_sum', 'model_price_std',

'model_price_mean'

3.数字特征

使用了置信度排名靠前的匿名特征“v_0”、“v_3”与“price”进行交叉，测试方法以上述一样，效果并不理想。

因为都是匿名特征，比较训练集和测试集分布，分析完基本没什么问题，并且它们在 lightgbm 的输出的重要性都是非常高的，所以先暂且全部保留。对简易 lgb 模型输出的特征重要度较高的几个连续数值特征对 price 进行刻画。

```
# 3、连续数值特征
# 都是匿名特征 比较训练集和测试集分布 分析完 基本没什么问题 先暂且全部保留咯
# 后期也许得对相似度较大的进行剔除处理
# 对简易lgb模型输出的特征重要度较高的几个连续数值特征对price进行刻画

dd = 'v_3'
Train_gb = df.groupby(kk)
all_info = {}
for kind, kind_data in Train_gb:
    info = {}
    kind_data = kind_data[kind_data[dd] > -10000000]
    info[kk+'_'+dd+'_max'] = kind_data.v_3.max()
    info[kk+'_'+dd+'_min'] = kind_data.v_3.min()
    info[kk+'_'+dd+'_std'] = kind_data.v_3.std()
    info[kk+'_'+dd+'_mean'] = kind_data.v_3.mean()
    info[kk+'_'+dd+'_median'] = kind_data.v_3.median()
    info[kk+'_'+dd+'_sum'] = kind_data.v_3.sum()
    all_info[kind] = info
brand_fe = pd.DataFrame(all_info).T.reset_index().rename(columns={"index": kk})
df = df.merge(brand_fe, how='left', on=kk)
```

```
dd = 'v_0'
Train_gb = df.groupby(kk)
all_info = {}
for kind, kind_data in Train_gb:
    info = {}
    kind_data = kind_data[kind_data[dd] > -10000000]
    info[kk+'_'+dd+'_max'] = kind_data.v_0.max()
    info[kk+'_'+dd+'_min'] = kind_data.v_0.min()
    info[kk+'_'+dd+'_std'] = kind_data.v_0.std()
    info[kk+'_'+dd+'_mean'] = kind_data.v_0.mean()
    info[kk+'_'+dd+'_median'] = kind_data.v_0.median()
    info[kk+'_'+dd+'_sum'] = kind_data.v_0.sum()
    all_info[kind] = info
    all_info[kind] = info
brand_fe = pd.DataFrame(all_info).T.reset_index().rename(columns={"index": kk})
df = df.merge(brand_fe, how='left', on=kk)
```

接下来主要是对输出重要度非常高的特征进行处理。对 14 个匿名特征使用乘法处理得到 14*14 个特征，使用 sklearn 的自动特征选择帮我们去筛选，大概运行了半天的时间。大致方法如下：

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

from sklearn.linear_model import LinearRegression

sfs = SFS(LGBMRegressor(n_estimators = 1000,objective='mae' ),

          k_features=50,

          forward=True,

          floating=False,

          cv = 0)

sfs.fit(X_data, Y_data)

print(sfs.k_feature_names_)
```

最终筛选得到：'new3*3', 'new12*14', 'new2*14','new14*14'。

接下来对 14 个匿名特征使用加法处理得到 14*14 个特征，这次不选择使用自动特征选择了，因为运行实在太慢了，笔记本耗不起。使用的方法是删除相关性高的变量,把要删除的特征记录下来，大致方法如下：（剔除相关度>0.95 的）

```
corr = X_data.corr(method='spearman')

feature_group = list(itertools.combinations(corr.columns, 2))

print(feature_group)

# 删除相关性高的变量,调试好直接去主函数进行剔除

def filter_corr(corr, cutoff=0.7):

    cols = []

    for i,j in feature_group:

        if corr.loc[i, j] > cutoff:

            print(i,j,corr.loc[i, j])

            i_avg = corr[i][corr[i] != 1].mean()

            j_avg = corr[j][corr[j] != 1].mean()

            if i_avg >= j_avg:

                cols.append(i)

            else:

                cols.append(j)

    return set(cols)

drop_cols = filter_corr(corr, cutoff=0.95)

print(drop_cols)
```

以上特征工程处理都是针对于树模型来进行的，接下来，简单说明神经网络的数据预处理。

大家都知道由于 NN 的不可解释性，可以生成大量的我们所不清楚的特征，因此我们对于 NN 的数据预处理只要简单处理异常值以及缺失值。

大部分的方法都包含在以上针对树模型数据处理方法中，重点讲述几个不同点：

在对于“notRepairedDamage”的编码处理，对于二分类的缺失值，往往取其中间值。

在对于其他缺失值的填充，在测试了效果后，发现填充众数的效果比平均数更好，因此均填充众数。

2.4 模型选择

本次比赛，我选择的是 lightgbm+catboost+neural network。

本来也想使用 XGBoost 的，不过因为它需要使用二阶导，因此目标函数没有 MAE，并且用于逼近的一些自定义函数效果也不理想，因此没有选择使用它。

经过上述的数据预处理以及特征工程：

树模型的输入有 83 个特征；神经网络的输入有 29 个特征。

1、lightgbm 和 catboost：

因为它们都是树模型，因此我同时对这两个模型进行分析

第一：lgb 和 cab 的训练收敛速度非常快，比同样参数的 xgb 快非常多。

第二：它们可以处理缺失值，计算取值的增益，择优录取。

第三：调整正则化系数，均使用正则化，防止过拟合。

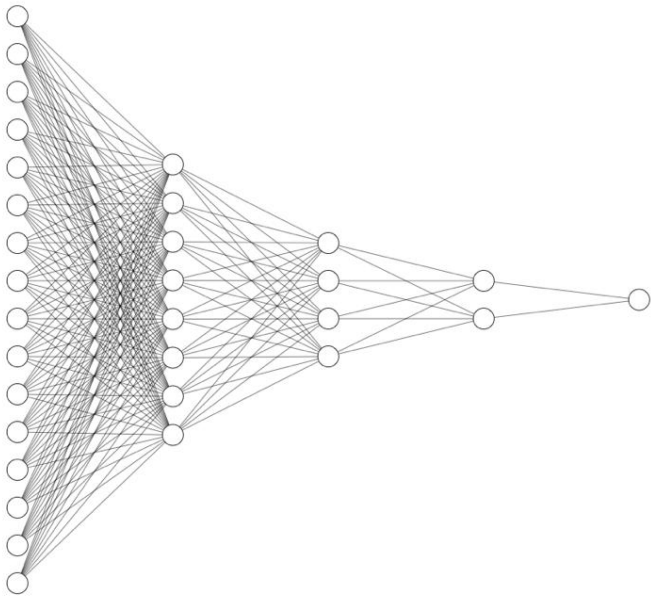
第四：降低学习率，获得更小 MAE 的验证集预测输出。

第五：调整早停轮数，防止陷入过拟合或欠拟合。

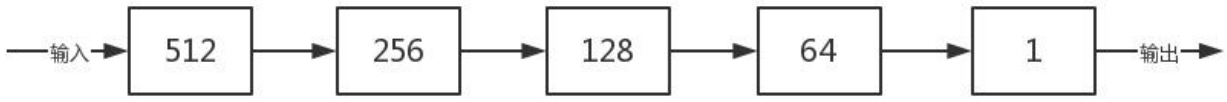
第六：均使用交叉验证，使用十折交叉验证，减小过拟合。

其他参数设置无明显上分迹象，以代码为准，不一一阐述。

2.neural network：



设计了一个五层的神经网络，大致框架如上图所示，但结点数由于太多只是展示部分结点画图。以下为全连接层的结点个数设置，具体实施可参考代码。



接下来对神经网络进行具体分析:

第一：训练模型使用小 batchsize，512，虽然在下降方向上可能会出现小偏差，但是对收敛速度的收益大，2000 代以内可以收敛。

第二：神经网络对于特征工程这一类不用操心很多，就能达到与树模型相差无几的精度。

第三：调整正则化系数，使用正则化，防止过拟合。

第四：调整学习率，对训练过程的误差进行分析，选择学习率下降的时机进行调整。

第五：使用交叉验证，使用十折交叉验证，减小过拟合。

第六：选择梯度下降的优化器为 Adam，它是目前综合能力较好的优化器，具备计算高效，对内存需求少等等优点。

对神经网络的数据预处理和特征工程与提升树模型略有不同，选择需要使用的特征标签，由于神经网络会生成大量的特征，我们只需要保留原始特征和刻画几个明显特征即可。

数据预处理方面，跟提升树模型相差无几，主要是处理掉异常数据，缺失值处理，分类特征填充众数，按照题目要求“power”位于 0～600，因此把“power”>600 的值截断至 600。这里需要注意的是 notRepairedDamage 的值是 0 和 1，然后为-的值设置为 0.5，在将它进行标签转换，0->1;0.5->2;1->3;这样符合神经网络的特征提取，不确定值位于两个确定值的中间。最后对数据进行了归一化处理。

特征工程方面、主要对“regDate”，“creatDate”构造特征，可获得年、月、日等一系列的新特征，然后做差可以获得使用年长和使用天数这些新特征。

2.5 模型融合

模型融合是比赛后期一个重要的环节，大体来说有如下的类型方式。

简单加权融合:

回归（分类概率）：算术平均融合（Arithmetic mean），几何平均融合（Geometric mean）；

分类：投票（Voting）

综合：排序融合(Rank averaging)，log 融合

stacking/blending:构建多层模型，并利用预测结果再拟合预测。

boosting/bagging（在 xgboost，Adaboost,GBDT 中已经用到）:多树的提升方法

其中 stacking 就是当用初始训练数据学习出若干个基学习器后，将这几个学习器的预测结果作为新的训练集，来学习一个新的学习器。

将个体学习器结合在一起的时候使用的方法叫做结合策略。对于分类问题，我们可以使用投票法来选择输出最多的类。对于回归问题，我们可以将分类器输出的结果求平均值。

对于如果训练集和测试集分布不那么一致的情况下是有一点问题的，其问题在于用初始模型训练的标签再利用真实标签进行再训练，毫无疑问会导致一定的模型过拟合训练集，这样或许模型在测试集上的泛化能力或者说效果会有一定的下降，因此现在的问题变成了如何降低再训练的过拟合性，这里我们一般有两种方法。

1.次级模型尽量选择简单的线性模型

2.利用 K 折交叉验证

由于两个树模型的训练数据一样且结构相似，首先对两个树模型进行 stacking，然后再与神经网络的输出进行 mix。

由于树模型和神经网络是完全不同的架构，它们得到的分数输出相近，预测值差异较大，往往在 MAE 上差异为 200 左右，因此将他们进行 MIX 可以取到一个更好的结果，加权平均选择系数选择 0.5，虽然神经网络的分数确实会比树模型高一点点，但是我们的最高分是多组线上最优输出的结合，因此可以互相弥补优势。

这里 lgb 和 catb 的模型融合采用的是: