多元线性回归

董峦 新疆农业大学

引言

不论是酒还是其它商品，通常来说一分价钱一分货，那么是不是越贵的葡萄酒越好喝呢？近些年，有很多双盲实验说明仅仅通过品尝是无法分辨一种酒价格高低的。也就是说当我们没有酒标、价格等先入为主的信息时，酒的口感与价格没有关系，便宜酒也有好喝的。从酒的本身来说，到底什么决定了“好喝”呢？本文试图从数据中发掘影响葡萄酒口感的因素。

数据集

在 UCI Machine Learning Repository 有这样一份数据集：`Wine Quality Data Set`（https://archive.ics.uci.edu/ml/datasets/wine+quality），该数据集包含葡萄牙 *Vinho Verde* 产区葡萄酒的两组数据，一组是红葡萄酒的，有1599条数据，另一组是白葡萄酒的，有4898条数据。该数据集来自研究：**Modeling wine preferences by data mining from physicochemical properties**（http://dx.doi.org/10.1016/j.dss.2009.05.016）。如论文题目反映的，该数据集建立的目的是根据葡萄酒的物理化学属性给酒一个评级。数据集包含的葡萄酒物理化学属性有以下11项：

1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide
7. total sulfur dioxide
8. density
9. pH
10. sulphates
11. alcohol

对酒的评价是一个0~10之间的分值。作者认为个别评酒师对酒的评价可能存有差异，但大量的品酒师一定能形成某种共识，该共识就是影响葡萄酒品质的那些关键因素。本文首先用上述物理化学属性预测评分，然后找出影响评分的关键因素。

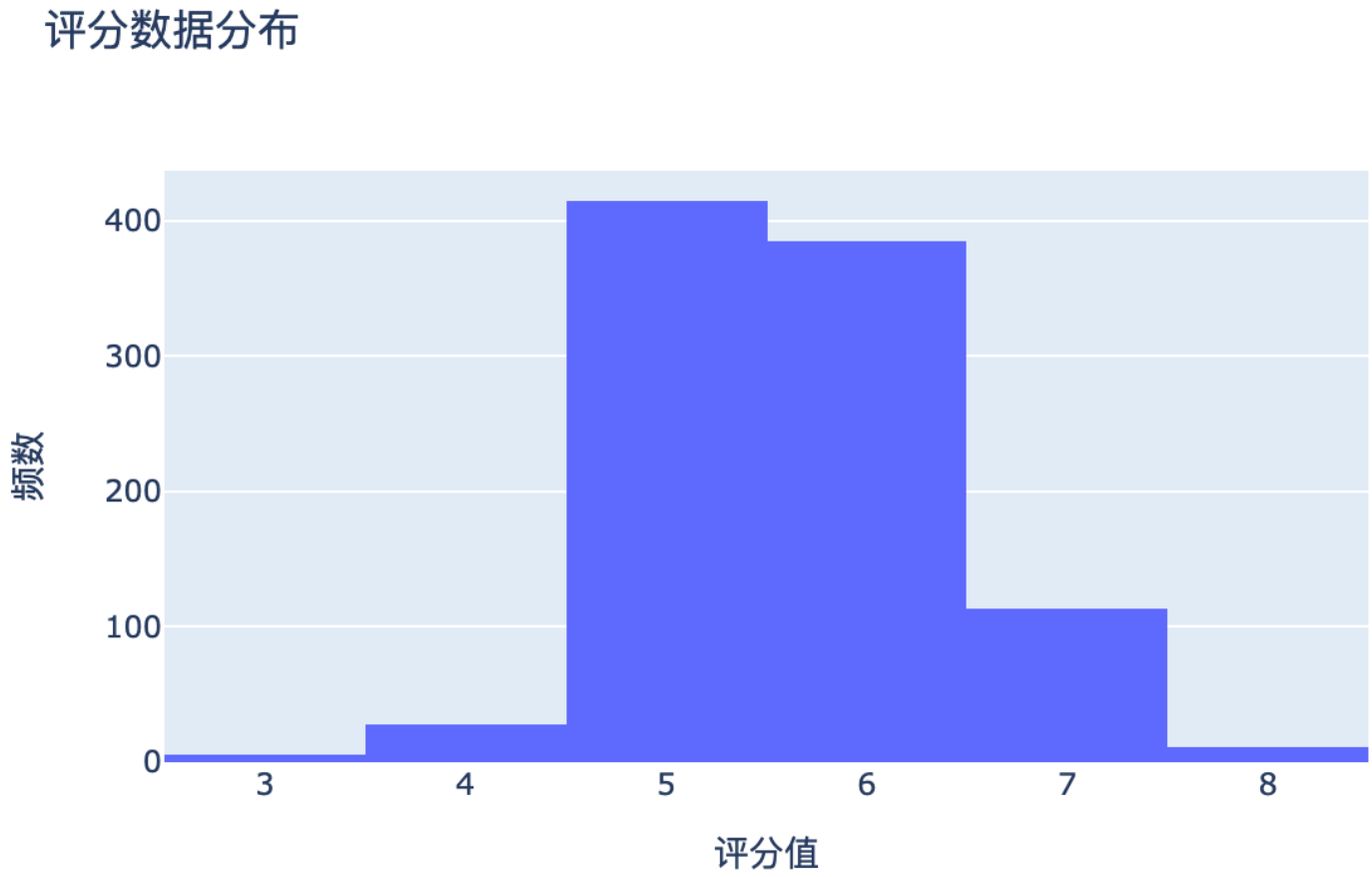
首先载入数据集，观察数据，对数据有一个总体认识。

```
import pandas as pd

data = pd.read_csv('winequality-red.csv', sep=';')
data.info()
data.describe()
```

```
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
```

```
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB


       fixed acidity  volatile acidity  ...      alcohol      quality
count    1599.000000       1599.000000  ...  1599.000000  1599.000000
mean        8.319637          0.527821  ...    10.422983     5.636023
std         1.741096          0.179060  ...     1.065668     0.807569
min         4.600000          0.120000  ...     8.400000     3.000000
25%         7.100000          0.390000  ...     9.500000     5.000000
50%         7.900000          0.520000  ...    10.200000     6.000000
75%         9.200000          0.640000  ...    11.100000     6.000000
max        15.900000          1.580000  ...    14.900000     8.000000

[8 rows x 12 columns]
```

可以看到在上述11个属性列之外还有一个 `quality` 列，该列中数值是整数，取值为 3~8，但评分分布并不均匀，大部分评分集中在5和6分，如下图所示



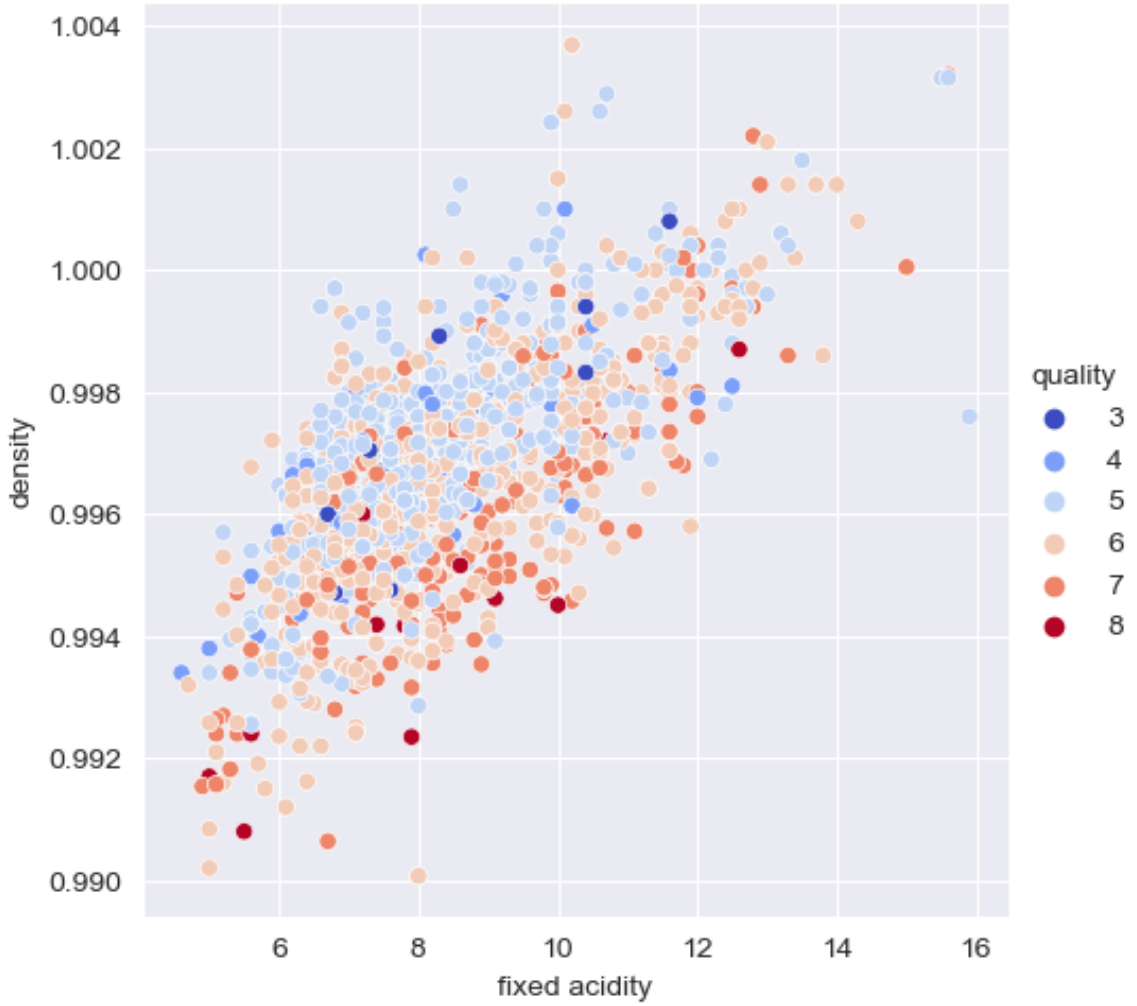
下面利用LDA技术将数据降到2维，然后可视化

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components=2)
feature2D = lda.fit(data.loc[:, :'alcohol'], data.loc[:, 'quality']).\
    transform(data.loc[:, :'alcohol'])

feature2D = pd.DataFrame(feature2D, columns=['x1', 'x2'])
feature2D['quality'] = data['quality']
sns.relplot(data=feature2D, x='x1', y='x2', hue='quality', palette='coolwarm')
```

从上图可以看出数据分布呈现聚集态势，从左侧向右侧数据点的评分逐渐增加。这至少反映了两个情况：第一评酒师的打分是有规律的，相似品质的酒在评分上不会有太大差异；第二数据分布有规律，建立模型是可能的。

上述11个葡萄酒的物理化学属性是特征（feature），评分值是标签（label），本文将建立多元线性回归模型试图根据11个物理化学属性预测评分值。

在建模之前有必要观察各特征之间有没有相关性，特征之间彼此相关的现象叫多重共线性。适度的多重共线性不成问题，但当出现严重共线性问题时，会导致分析结果不稳定。绘制 density 和 fixed acidity 两个属性的散点图可以看到两个属性程正相关关系，这说明多重共线性问题在本数据集中是存在的，但这个问题并不严重，因为后面的实验将发现这两个特征对模型预测能力的影响微乎其微，而且利用正则化技术可以有效抑制多重共线性。



本文利用 PyTorch 的 Dataset 类来隐藏数据的采样、置乱等操作细节。定义适配特定数据集的 Dataset 类需要实现两个方法：`__len__` 和 `__getitem__`，如下所示

```
import torch as th
from torch.utils.data import Dataset

class WineQualityDataset(Dataset):
    def __init__(self, dataframe, tag, feature_mean=None, feature_std=None):
        self.data = dataframe
        self.feature = self.data.loc[:, :'alcohol'].to_numpy()
        self.label = self.data.loc[:, 'quality':'quality'].to_numpy()

        if tag == 'train':
            self.mean = self.feature.mean(axis=0)
            self.std = self.feature.std(axis=0)
        else:
            self.mean = feature_mean
            self.std = feature_std

        try:
            self.feature = (self.feature - self.mean) / self.std
        except Exception as e:
            print('请指定训练集上的特征均值与标准差，类型是 Numpy 数组', str(e))
```

```
    def __len__(self):
        return self.data.shape[0]

    def __getitem__(self, idx):
        feature = self.feature[idx, :]
        label = self.label[idx]
        return th.tensor(feature, dtype=th.float32), th.tensor(label, dtype=th.float32)
```

注意，由于各特征的取值范围大相径庭，所以要对特征做标准化操作，且在验证集和测试集上做标准化时要使用训练集上的特征均值与标准差。

对特征做标准化操作不仅有利于训练，各个特征标准化后，权重的大小还能定量反映出对应特征对评分值预测所起作用的大小。

模型

本文将评分的预测当做一个回归问题，将样本 i （第i条数据）的评分值 y_i 看做各种物理化学属性（特征）的加权和，即

$$y_i = \sum_{j=1}^{n} x_{ij} w_j + b$$

其中 x_{ij} 是第 i 个样本的第 j 个特征，w_j 是第 j 个特征的权重，b 是偏置项，$n = 11$ 是特征的个数。本文使用 PyTorch 提供的线性模型，如下

```
from torch import nn

model = nn.Linear(11, 1) # 特征维度为11，输出维度为1
```

在 PyTorch 中 model 是可运行的，因此像调用函数那样在特征上执行模型即可得到预测值，如下所示

```
prediction = model(feature)
```

损失函数

和一元线性回归模型不同的是，本文在损失函数中增加了正则化项（Regularization），如下

$$L_{LASSO} = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{n} |w_j|$$

损失函数由两部分组成，第一部分是均方误差，第二部分是正则化项，其中 y_i 是预测值，$\hat{y}_i$ 是真值（ground truth）或标签（label），m 是样本个数。正则化项是权重的L1范数，其重要性由系数 λ 来控制。以该函数为损失函数的回归叫LASSO回归，LASSO是`Least Absolute Shrinkage and Selection Operator`的简称。LASSO回归可以使权重的学习结果稀疏化，即部分权重为零，方便选出重要的特征。该特性适合本文的目标：找出影响葡萄酒品质的关键因素。

以权重的L2范数为正则化项的回归叫岭回归（ridge regression），定义如下

$$L_{ridge} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{n} w_j^2$$

岭回归的作用主要是防止模型过拟合（over-fitting）。

注意，上述损失函数中的正则化项仅包含模型权重不包含偏置 b 。

在 PyTorch 中损失函数这样定义：

```
import torch as th
from torch.nn.functional import mse_loss

mse = mse_loss(prediction, label)
penalty = th.linalg.vector_norm(model.weight, ord=1) # 模型权重的L1范数

loss = mse + lambd * penalty
```

优化

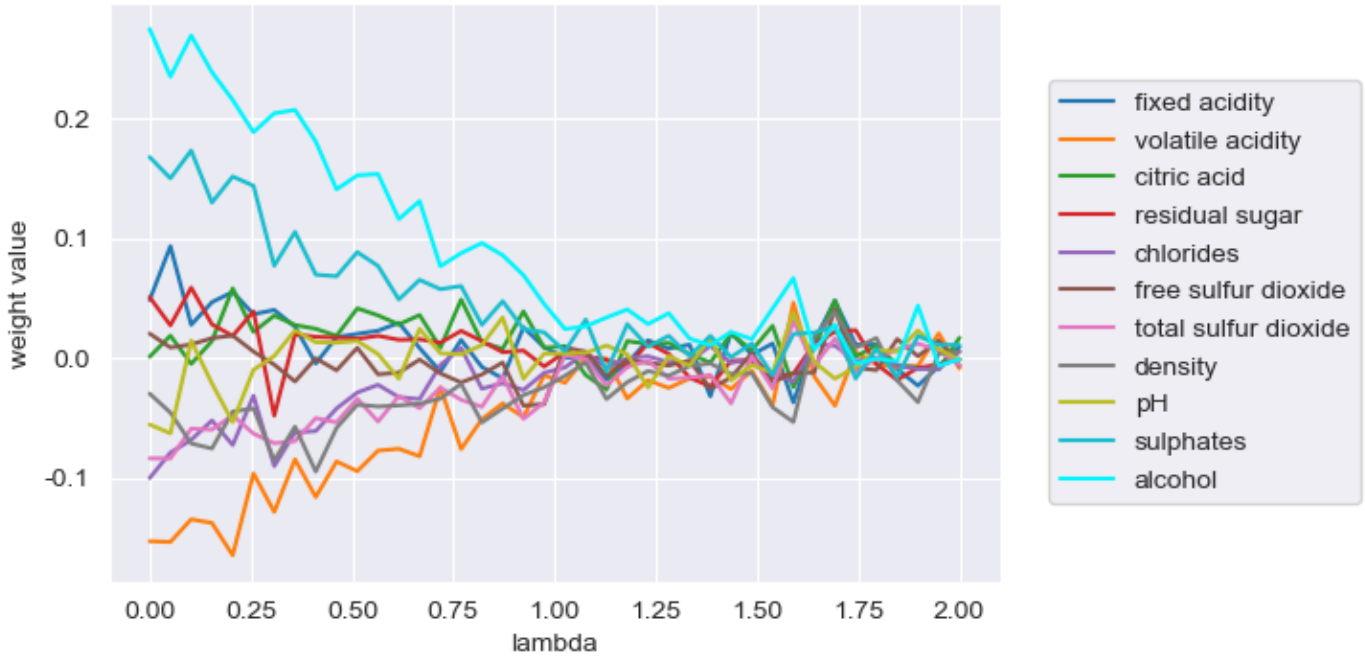
本文仍使用梯度下降方法优化模型，不同的是在一元线性回归模型中使用所有样本计算梯度，在本文里仅使用一个批次（batch）里的样本计算梯度然后做梯度下降，这叫做 mini-batch gradient descent，也叫随机梯度下降（Stochastic Gradient Descent，简称 SGD）。如果数据集里有1000个样本，批次大小是50的话，则每次用50个样本计算梯度然后做一次梯度下降，20次迭代后遍历了一遍数据集（一个 epoch 结束）。 下一轮迭代开始前通常要把样本置乱，避免同样的一组特征被拿来训练，导致模型过拟合。

由于使用了 PyTorch 所以不再需要手动计算梯度了，利用自动求导技术即可实现梯度计算，使用优化器进一步隐藏了梯度下降的实现细节。模型训练部分的关键代码如下

```
from torch import optim
from torch.utils.data import DataLoader

# 定义优化器。优化器包裹模型参数
optimizer = optim.SGD(model.parameters(), lr=0.02)
# 定义数据装载器。它包裹自定义的 Dataset 类
train_loader = DataLoader(trainset, batch_size=16, shuffle=True)

# 一轮迭代过程
for feature, label in train_loader:
    # forward step: 计算损失值
    prediction = model(feature)
    loss = ridge_regression_loss(prediction, label)

    # backward step: 计算梯度
    optimizer.zero_grad()
    loss.backward()

    # update step: 优化器进行参数更新
    optimizer.step()
```

上一节提到，以L1范数作正则化项时模型是LASSO回归，此模型具有特征筛选的功能。绘制模型权重随 λ 改变而变化的规律，如下图所示。可见随着λ 变大，模型权重向零值靠近（应该变成零，但由于PyTorch的实现原因权值在零值附近摆动）。但各个权值归于零的速度不同，越重要的特征其权值越是最后归于零，这是我们识别特征重要性的依据。从图上看 `alcohol` `volatile acidity` 和 `sulphates` 权值最大是最重要的三个特征，即这三个特征是评判酒品质的最重要指标。但要注意 `alcohol` 和 `sulphates` 特征的权重是正值，说明这两个指标对评分的贡献是正向的；`volatile acidity` （挥发酸）的权重是负数，说明它对评分的贡献是反向的，即该指标超过指标均值越多越拉低评分。经过资料核实，挥发酸过多对葡萄酒来说确实是一种缺陷。

把正则化项换成L2范数，观察模型权重随 λ 变化的规律如下图所示，可以看到所有权重都以同样的速度衰减。岭回归惩罚过大的权重，是一种对抗过拟合的技术。



超参数优化

模型权重和偏置项属于参数（parameter），另一些可配置的参数叫超参数（hyper-parameter），比如学习率、批次大小和代价函数里的 λ。这些参数对模型的训练存在不可忽视的影响，因此配置这些参数是需要认真对待的。然而这些参数的选择目前并没有简便的方法，仍然需要依靠重复实验，下面描述超参数的选择过程。

首先要从数据集里分出一个叫验证集（validation set）的部分，此时数据集由训练集、测试集和验证集三个集合组成。验证集用来在训练中评估模型性能，模型最终性能仍然由测试集来评估。简单地理解，验证集是一个较小的测试集。

然后用不同超参数的组合训练模型，记录每一组超参数和此时模型在验证集上的表现，由此确定最佳超参数组合。

最后用上述最佳超参数组合训练模型，并在测试集上报告模型性能。

例如，为了检验何种学习率和批次大小最有利于模型训练，编写如下代码

```
learning_rates = [0.01, 0.02, 0.03, 0.04, 0.05]
batchsizes = [8, 16, 32, 64, 128]


for i, lr in enumerate(learning_rates):
    for j, batchsize in enumerate(batchsizes):
        # 训练模型
        # 记录模型在验证集上性能
        # 打印结果
```

结果如下：

```
lr: 0.01, batchsize:   8, minimum validation loss: 0.486
lr: 0.01, batchsize:  16, minimum validation loss: 0.489
lr: 0.01, batchsize:  32, minimum validation loss: 0.525
lr: 0.01, batchsize:  64, minimum validation loss: 1.036
lr: 0.01, batchsize: 128, minimum validation loss: 2.225
lr: 0.02, batchsize:   8, minimum validation loss: 0.484
lr: 0.02, batchsize:  16, minimum validation loss: 0.486
lr: 0.02, batchsize:  32, minimum validation loss: 0.489
lr: 0.02, batchsize:  64, minimum validation loss: 0.512
lr: 0.02, batchsize: 128, minimum validation loss: 0.944
lr: 0.03, batchsize:   8, minimum validation loss: 0.484
lr: 0.03, batchsize:  16, minimum validation loss: 0.483
lr: 0.03, batchsize:  32, minimum validation loss: 0.492
lr: 0.03, batchsize:  64, minimum validation loss: 0.487
lr: 0.03, batchsize: 128, minimum validation loss: 0.599
lr: 0.04, batchsize:   8, minimum validation loss: 0.482
lr: 0.04, batchsize:  16, minimum validation loss: 0.483
lr: 0.04, batchsize:  32, minimum validation loss: 0.489
lr: 0.04, batchsize:  64, minimum validation loss: 0.494
lr: 0.04, batchsize: 128, minimum validation loss: 0.504
lr: 0.05, batchsize:   8, minimum validation loss: 0.484
lr: 0.05, batchsize:  16, minimum validation loss: 0.484
lr: 0.05, batchsize:  32, minimum validation loss: 0.485
lr: 0.05, batchsize:  64, minimum validation loss: 0.491
```

```
lr: 0.05, batchsize: 128, minimum validation loss: 0.512
```

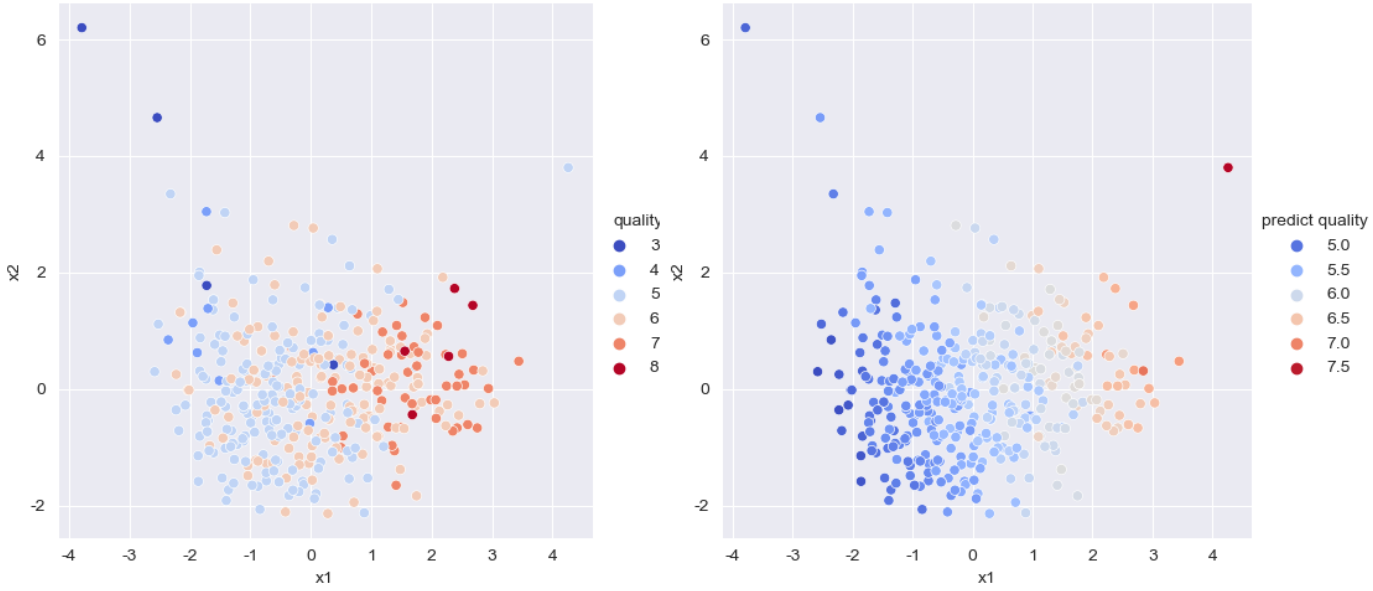
可见在学习率为 0.04，批次大小为 8时，模型在验证集上可以取到最小损失（MAE）0.482，在其他配置下都不能达到这个水平。下面就按这个超参数配置训练模型观察模型在测试集上的表现。

测试

以验证集上检验过的最佳超参数组合训练模型，每一次迭代记录验证集上表现和模型参数，伪代码如下所示

```
foreach epoch:
    foreach batch:
        进行一次迭代
        记录模型在验证集上性能
        记录模型参数

把验证集上最佳表现对应的模型参数作为最终模型的参数
```

模型在测试集上的MAE是0.515。把测试集上的预测结果可视化，如下所示。左侧是测试样本实际评分的可视化，右侧是预测评分的可视化。结合图例中颜色的定义可以看到预测的结果与真实值大致相同，但样本的实际评分较大或较小时，预测效果不佳。



过拟合与欠拟合

当模型过于简单不足以描述数据分布的时候会发生欠拟合（underfitting）。构造以下测试数据，其中 y 是一个叠加了噪声的二次函数，而数据集里只含有特征 x 的一次项。

```
m = 15
x = np.arange(m).reshape((m,1))
err = np.random.randn(m, 1) * 40 # 标准差40
y = x**2 - 6*x + 10 + err

data = pd.DataFrame(data=np.hstack((y,x)), columns=['y', 'x'])
```

定义数据集

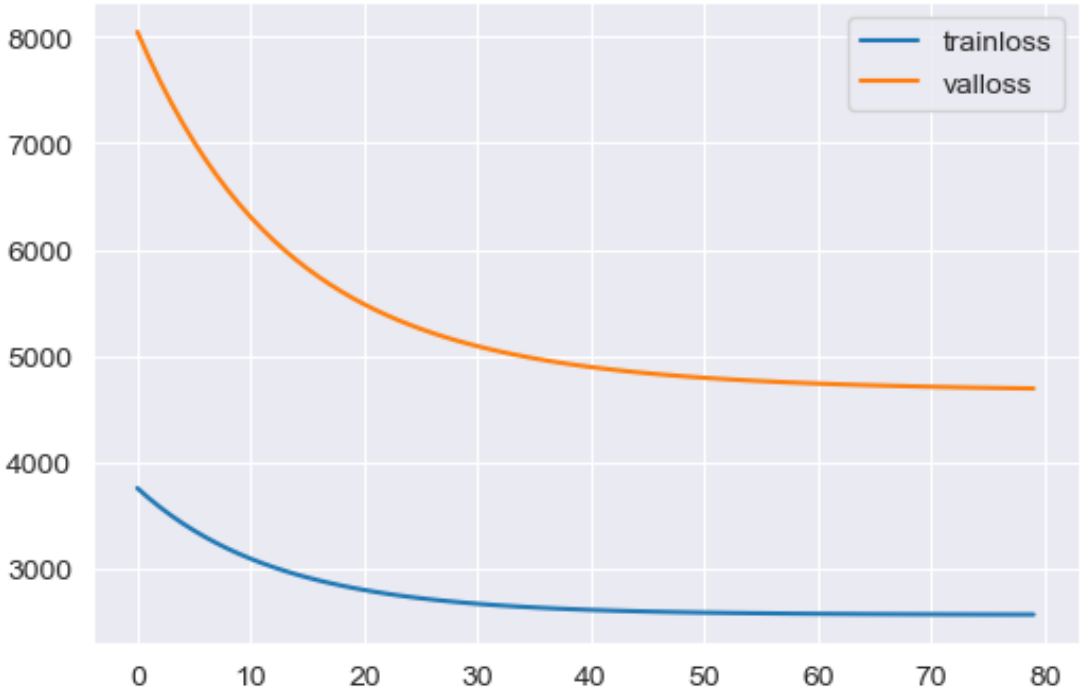
```
import torch as th
from torch.utils.data import Dataset


class TestDataset(Dataset):
    def __init__(self, data, tag, feature_mean=None, feature_std=None):
        self.data = data
        self.feature = self.data[:,1:]
        self.label = self.data[:, :1]

        if tag == 'train':
            self.mean = self.feature.mean(axis=0)
            self.std = self.feature.std(axis=0)
        else:
            self.mean = feature_mean
            self.std = feature_std

        try:
            self.feature = (self.feature - self.mean) / self.std
        except Exception as e:
            print('请指定训练集上的特征均值与标准差，类型是 Numpy 数组', str(e))


    def __len__(self):
        return self.data.shape[0]


    def __getitem__(self, idx):
        feature = self.feature[idx, :]
        label = self.label[idx]
        return th.tensor(feature, dtype=th.float32), \
               th.tensor(label, dtype=th.float32)
```

分割数据集，建立模型并训练

```
from dataset_test import TestDataset
from torch import nn
from train import trainer

population = data.shape[0]
idx = np.random.permutation(population)

data = data.to_numpy()
# 分割数据集
trainset = TestDataset(data[idx[:10]], tag='train')
valset = TestDataset(data[idx[10:]], tag='test', feature_mean=trainset.mean, \
                     feature_std=trainset.std)
```

```
# 建立模型
feature_nums = 1
model = nn.Linear(feature_nums, 1)

# 训练
config = {'epoches': 70, 'batchsize':10, 'lr':0.02, 'lambd': 0, 'norm':'L1'}
trainloss, valloss, _ = trainer(trainset, valset, model, config)
```

训练损失和验证损失如下图所示

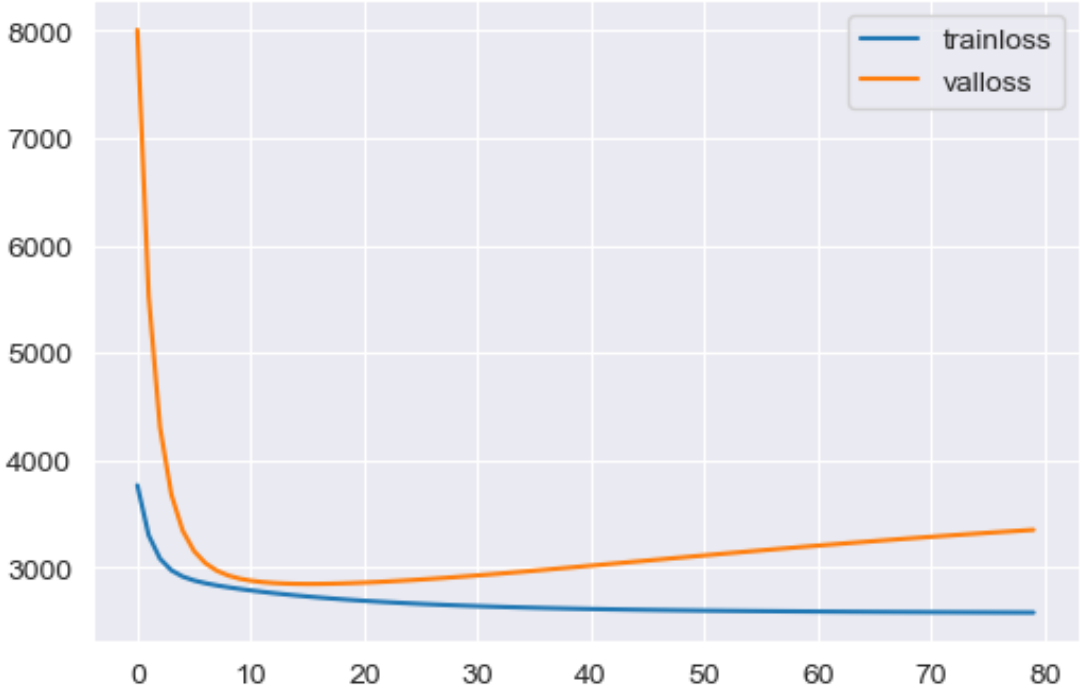


可以看到两个损失间存在较大的间隙，这说明直线模型并不能很好地抓住数据分布的特点，发生了欠拟合。过拟合则是另一番情景，下面增加自变量的高阶项：$\{x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9\}$

```
for i in range(2,10):
    data.loc[:, f'x^{i}'] = data['x'] ** i
data = data.to_numpy()

# ......

# 模型要相应修改
feature_nums = 9
model = nn.Linear(feature_nums, 1)
```

训练损失和验证损失如下图所示，可见在训练损失还在下降的时候验证损失不降反升了，这是发生过拟合的显著特征。

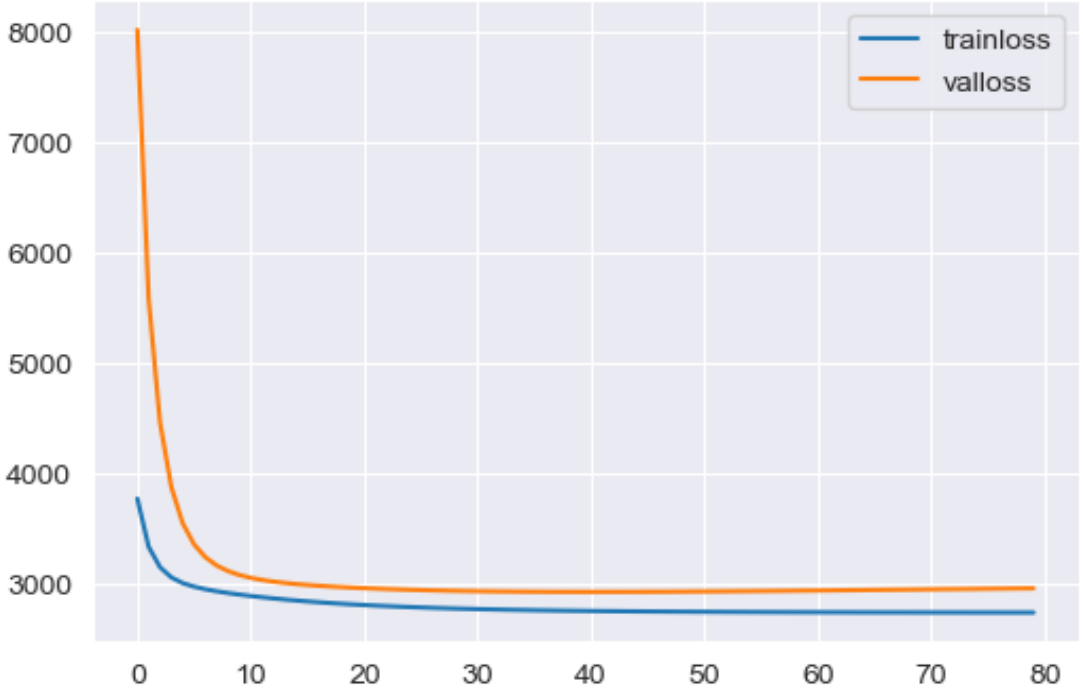
数据分布是二次的，但通过特征工程（feature engineering）使特征的阶次上升到9次，远远高于实际所需，因此模型在训练集上发生了过拟合，或者说把噪声的特点也学习了，但这个“经验”并不适用于验证集，所以预测效果反而变差了。



在机器学习中，偏差（Bias）指**所有可能的训练数据集**训练出的**所有模型**的输出**平均值**与**真实模型**的输出值之间的差异。方差（Variance）是**不同的训练数据集训练出的模型**输出值之间的差异。欠拟合的时候是 high bias, low variance；过拟合的时候是 low bias, high variance。

对上述过拟合情况，选择L2范数并设置 λ 参数后再次训练模型，损失函数变成下图所示样子，可见过拟合情况被消除了。

```
config = {'epoches': 80, 'batchsize':10, 'lr':0.02, 'lambd': 10, 'norm':'L2'}
```



小结

本文在葡萄酒评分数据集上建立多元线性回归模型预测评分，在代价函数中添加L1范数正则化项识别对评分最有影响的特征，利用验证集选择合适的超参数。在编程中实现了自定义 Dataset 类并利用 PyTorch 的优化器做参数优化。

从测试结果上可以看到，模型对评分为5和6的样本预测较好，对其它评分预测效果不佳。一个解释是样本在评分上分布不平衡，从本文图1可以看到，评分为3、4、7、8的样本很少，大部分样本的评分集中在5和6。对模型来说牺牲前者的预测准确性不影响模型总体表现，因此造成预测评分集中在5~7分之间。为了克服样本不平衡问题，可以补充相应样本弱化该影响，还可以改用基于局部信息做预测的方法，比如KNN，

思考与实践

- 分析白葡萄酒数据集，分析影响白葡萄酒评分的最主要特征
- 以L2范数为正则化项，利用验证集确定最佳超参数 λ
- 把代价函数中的均方误差改成加权平均，给评分较少的样本（评分为3、4、7、8的样本）分配更大的权重能否克服样本不平衡问题?