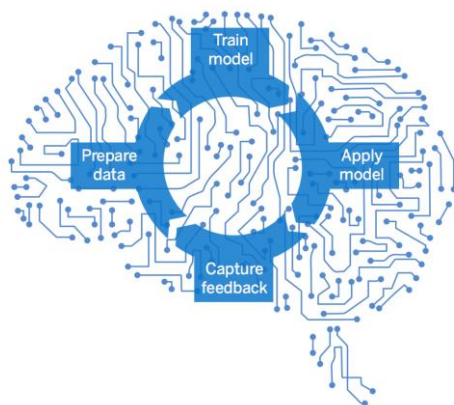


新能源汽车大数据创新创业大赛

电动汽车动力电池充电能量预测



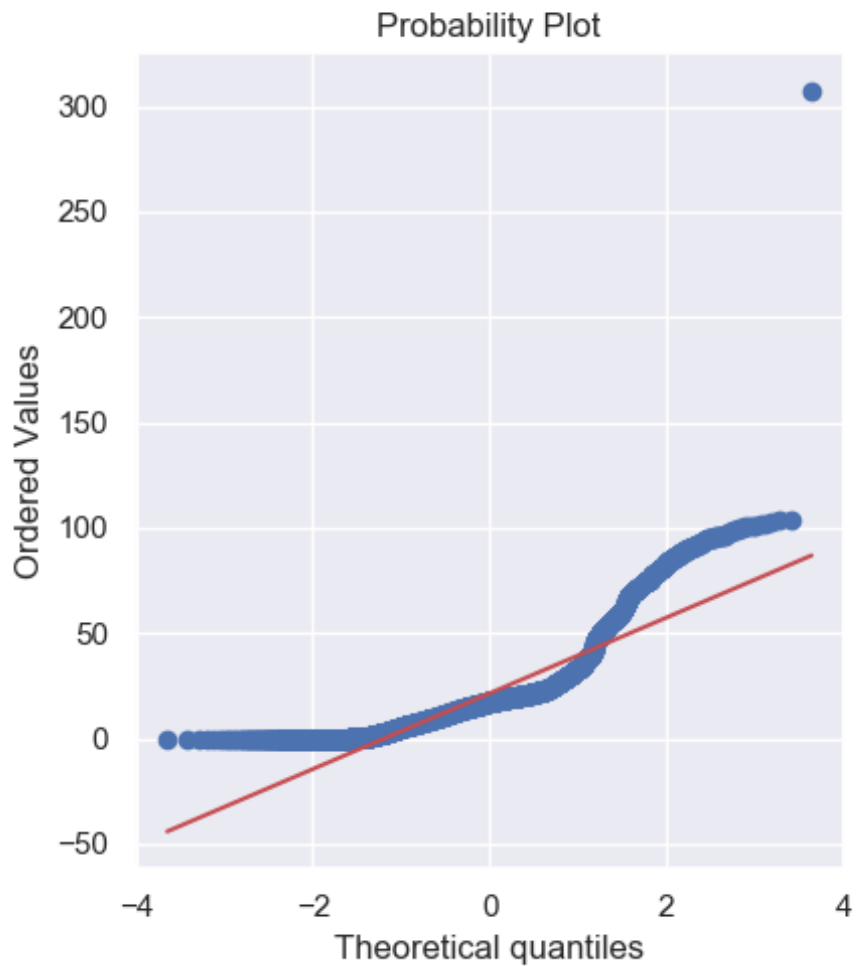
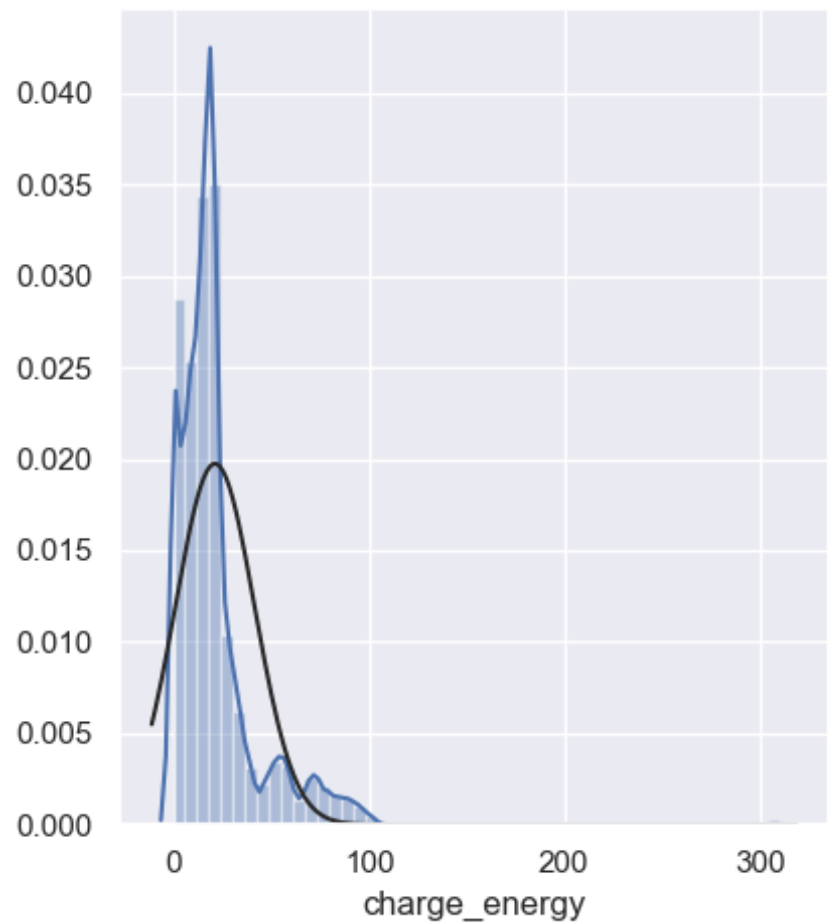
同济大学

刘晓曼 王昕杰 谭海宇

2018/11/22



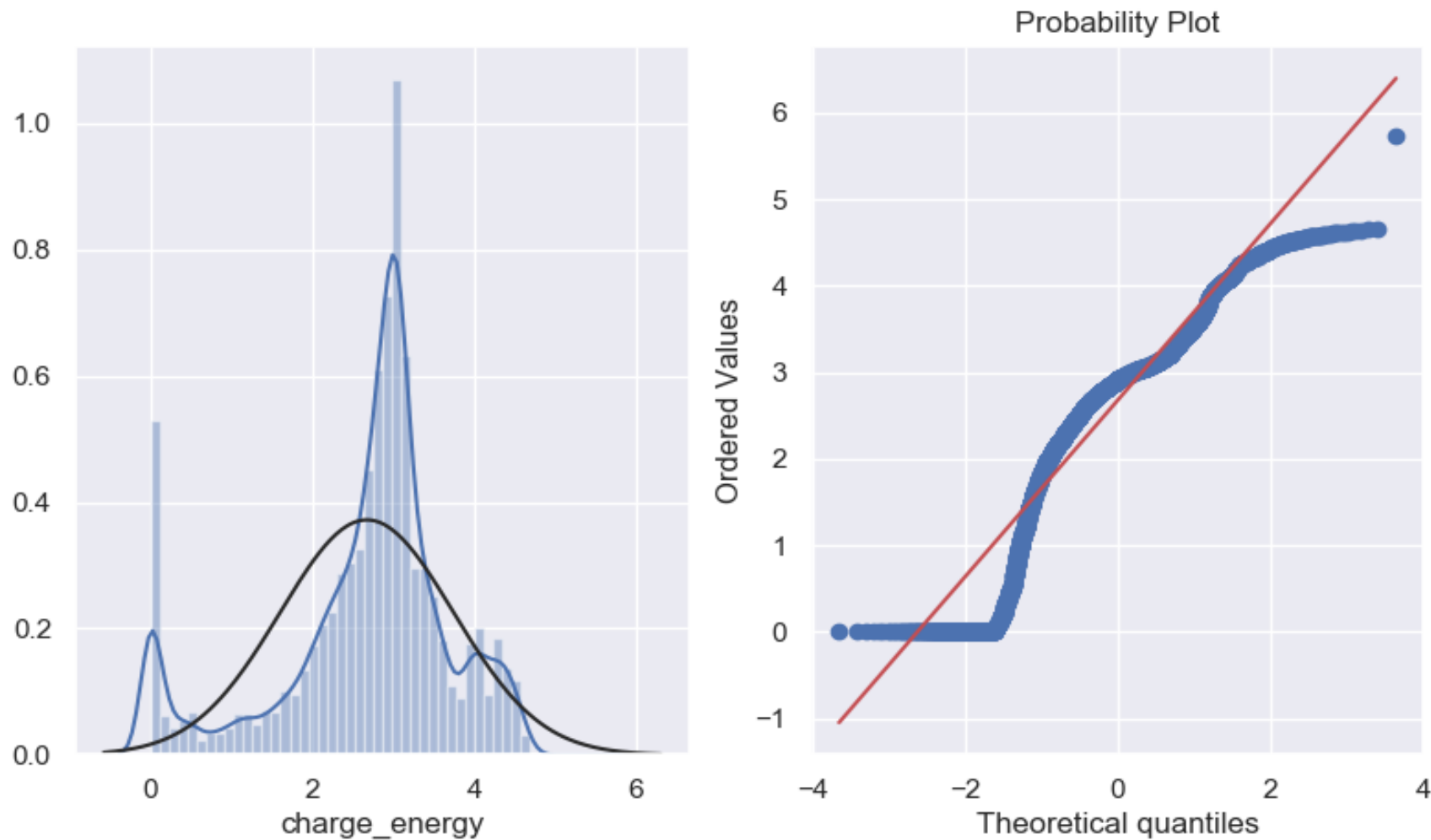
- 数据分析与清洗
- 模型设计
- 算法结构
- 可移植性与工程优化



对数正态分布

- 影响充电能量的因素不是彼此独立的，各种因素对充电能量的影响是相乘关系
- 数据量少

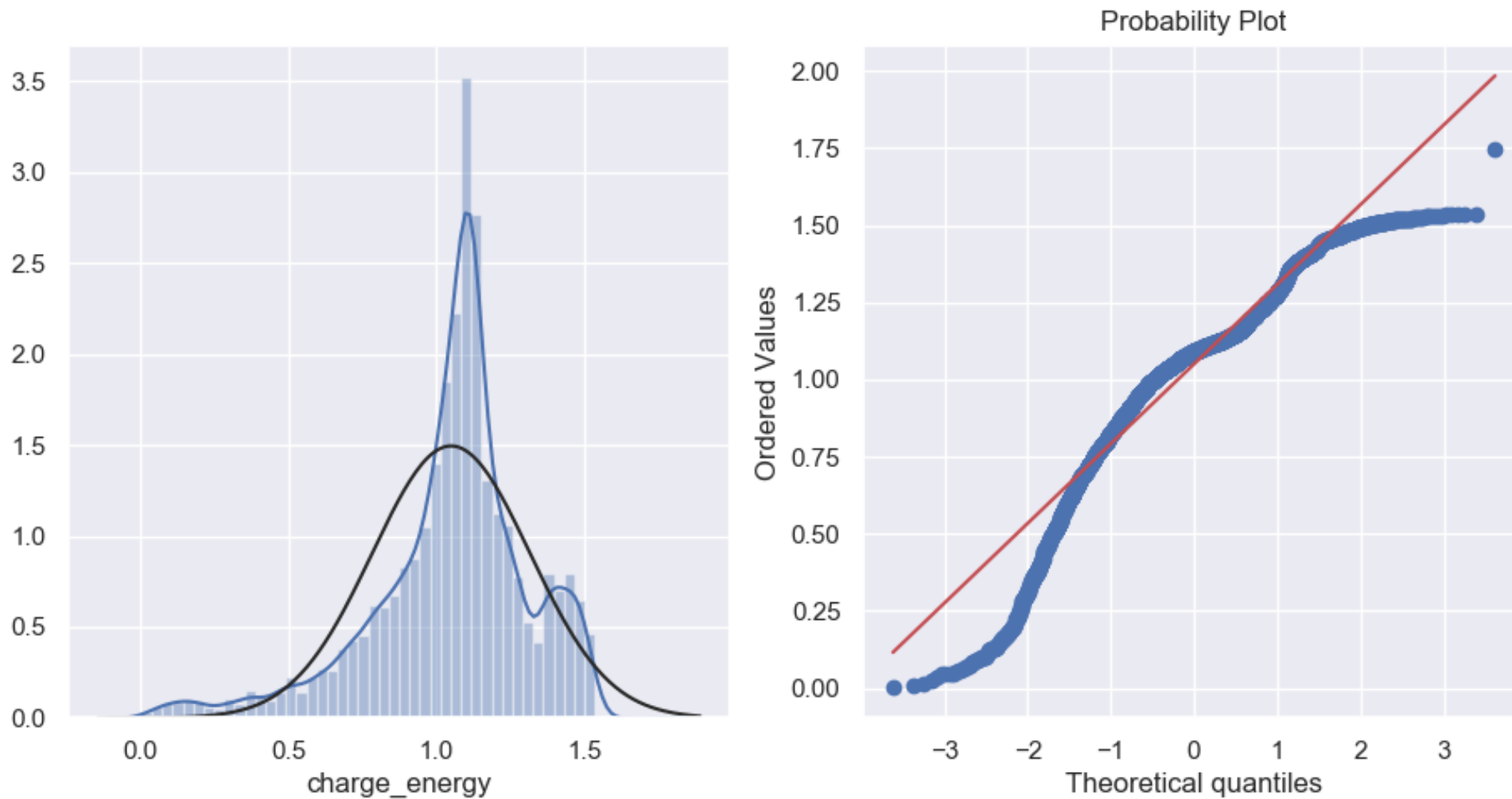
训练集能量原始分布



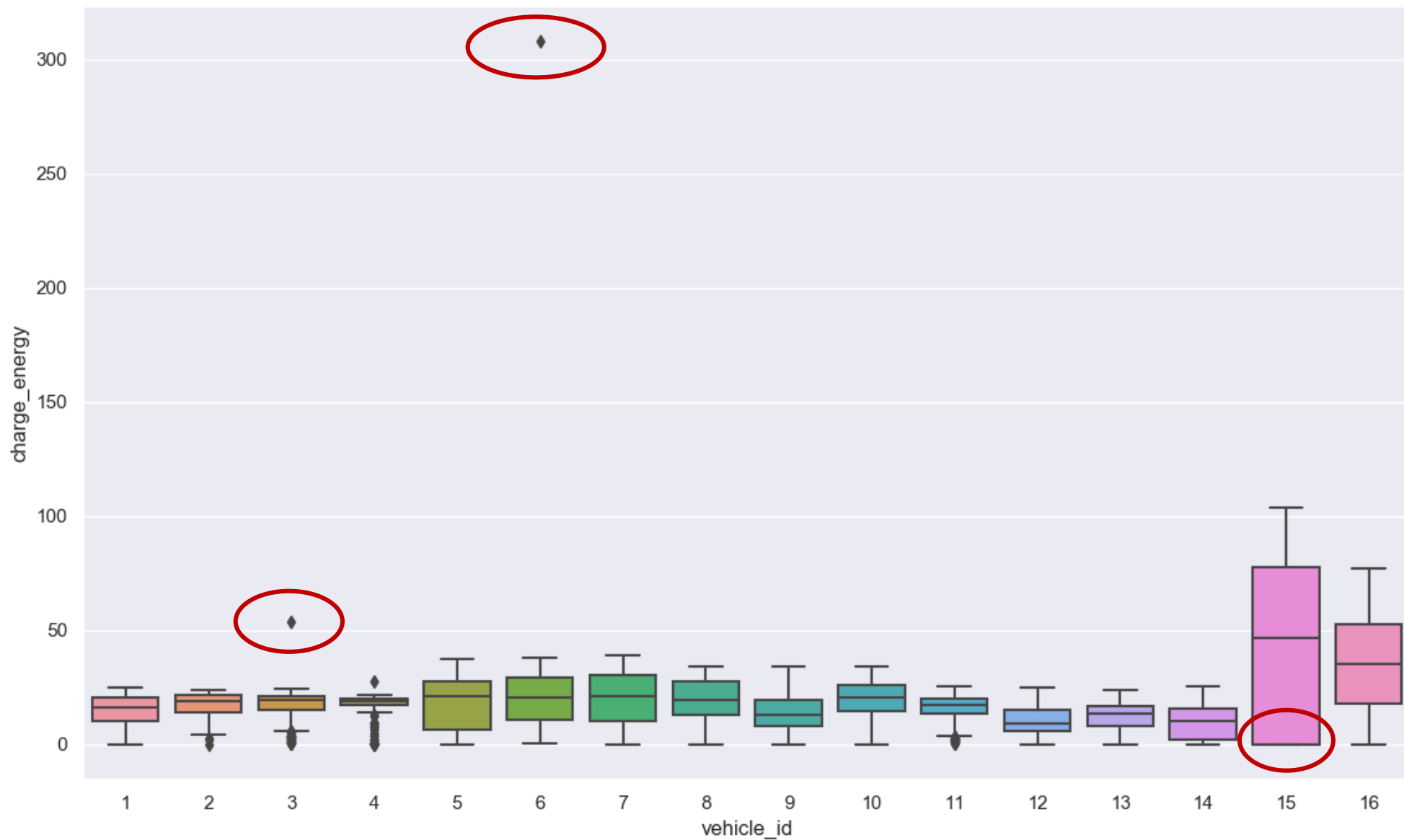
对数正态分布

• 许多小能量分布点

训练集能量 $\ln(x+1)$ 处理后分布



训练集能量In处理后>0部分分布




能量分布Box plot

过充保护或充电阶段转化



缺失值检测

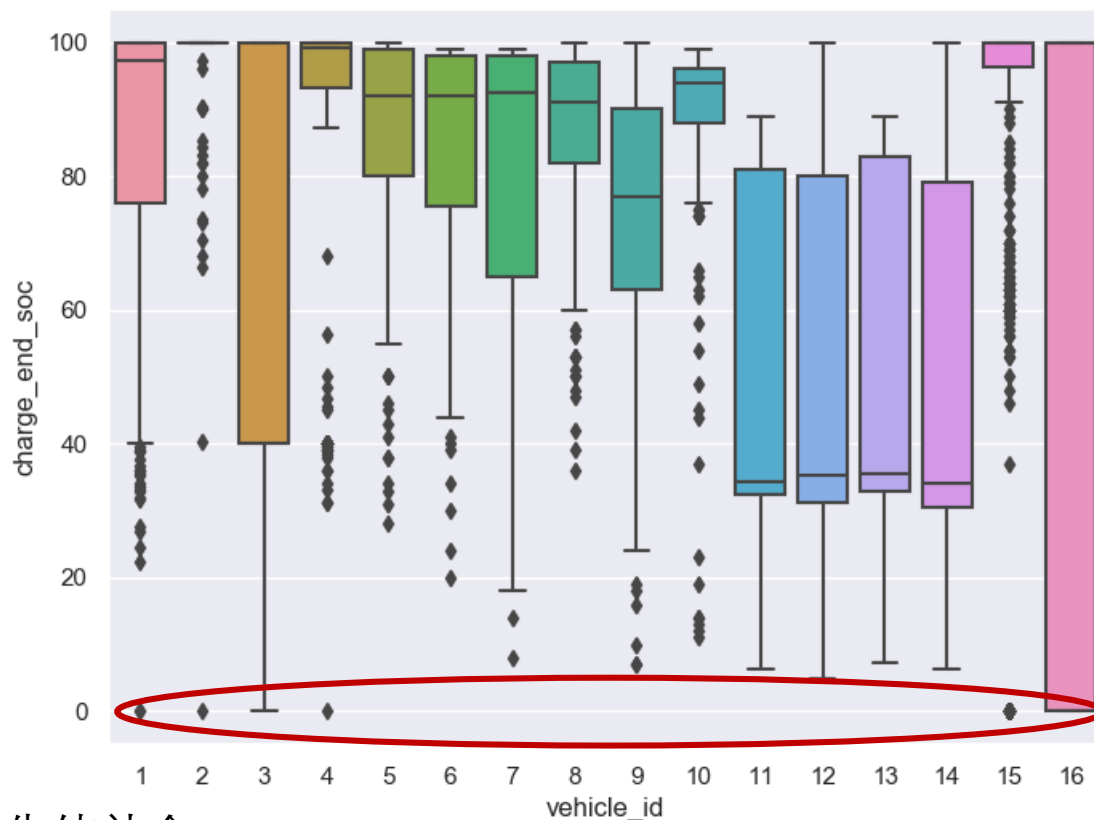
```
total = df_train.isnull().sum().sort_values(ascending=False)
percent = (df_train.isnull().sum()/df_train.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
print(missing_data)
```



	Total	Percent
charge_end_soc	154	0.029799
charge_start_soc	55	0.010642

所有车型的缺失值

数据清洗非常重要，大数据往往是大量有问题的数据

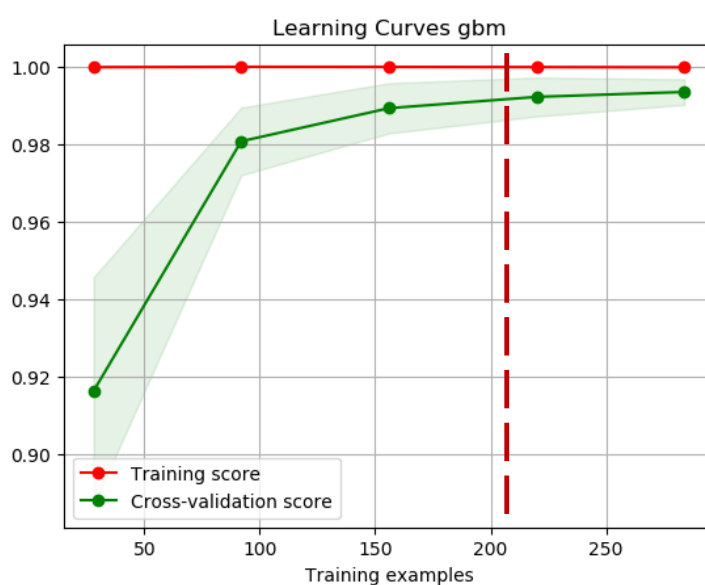
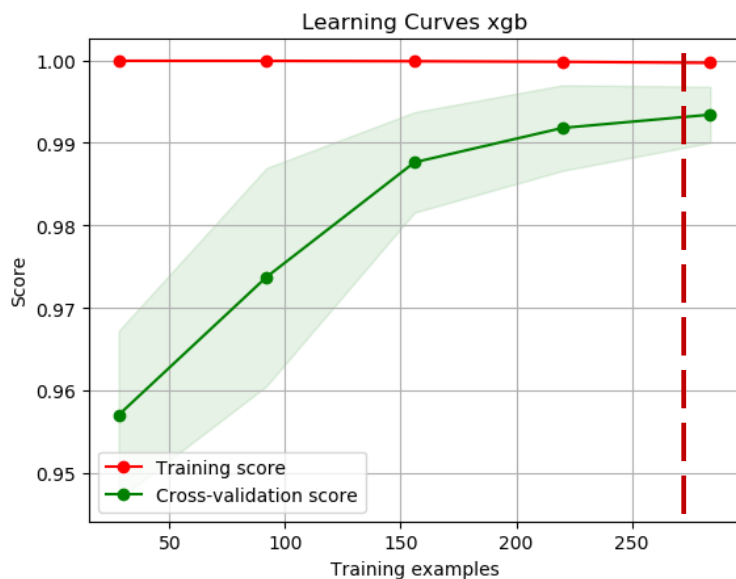
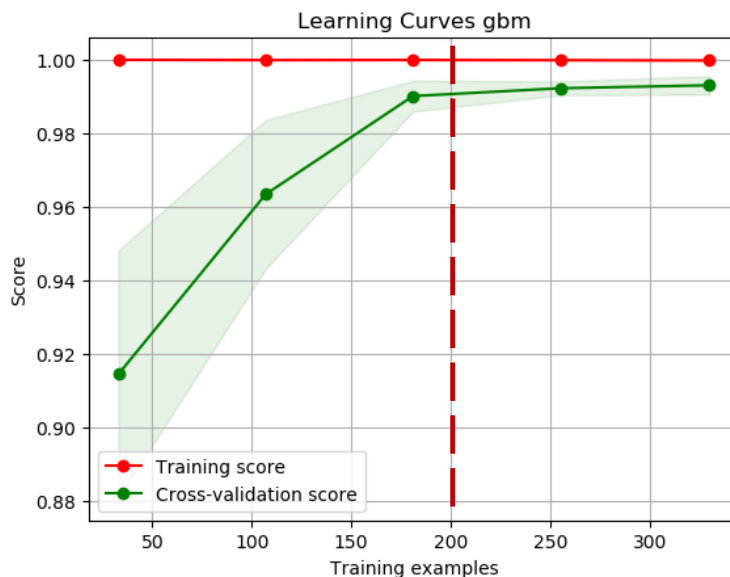
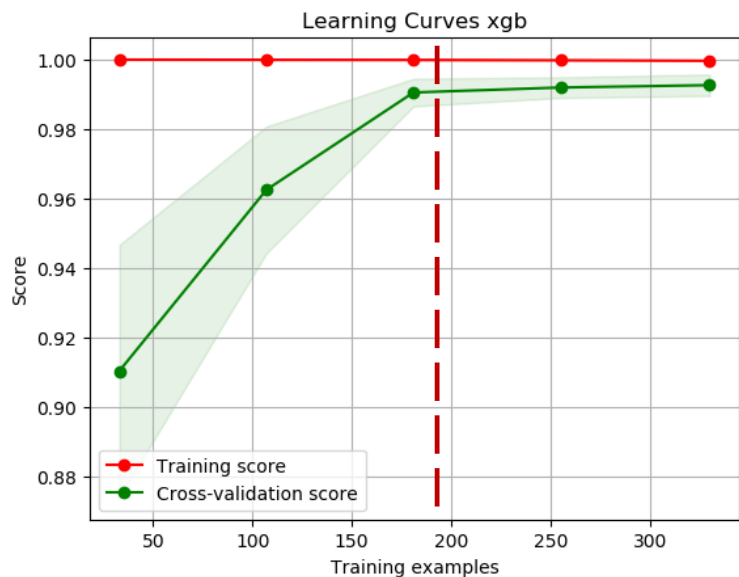


异常点修正，缺失值补全

Car1-3 `car_phase0.loc[(car_phase0.charge_end_U > 389) & (car_phase0.charge_end_L > -29), ['charge_end_soc']] = 100`

Car4, 15-16 使用随机森林与梯度提升树进行异常点修正，缺失值补全

└─ 增加数据预处理工作量，效果一般，最后选择用树算法进行提升



Car15-16

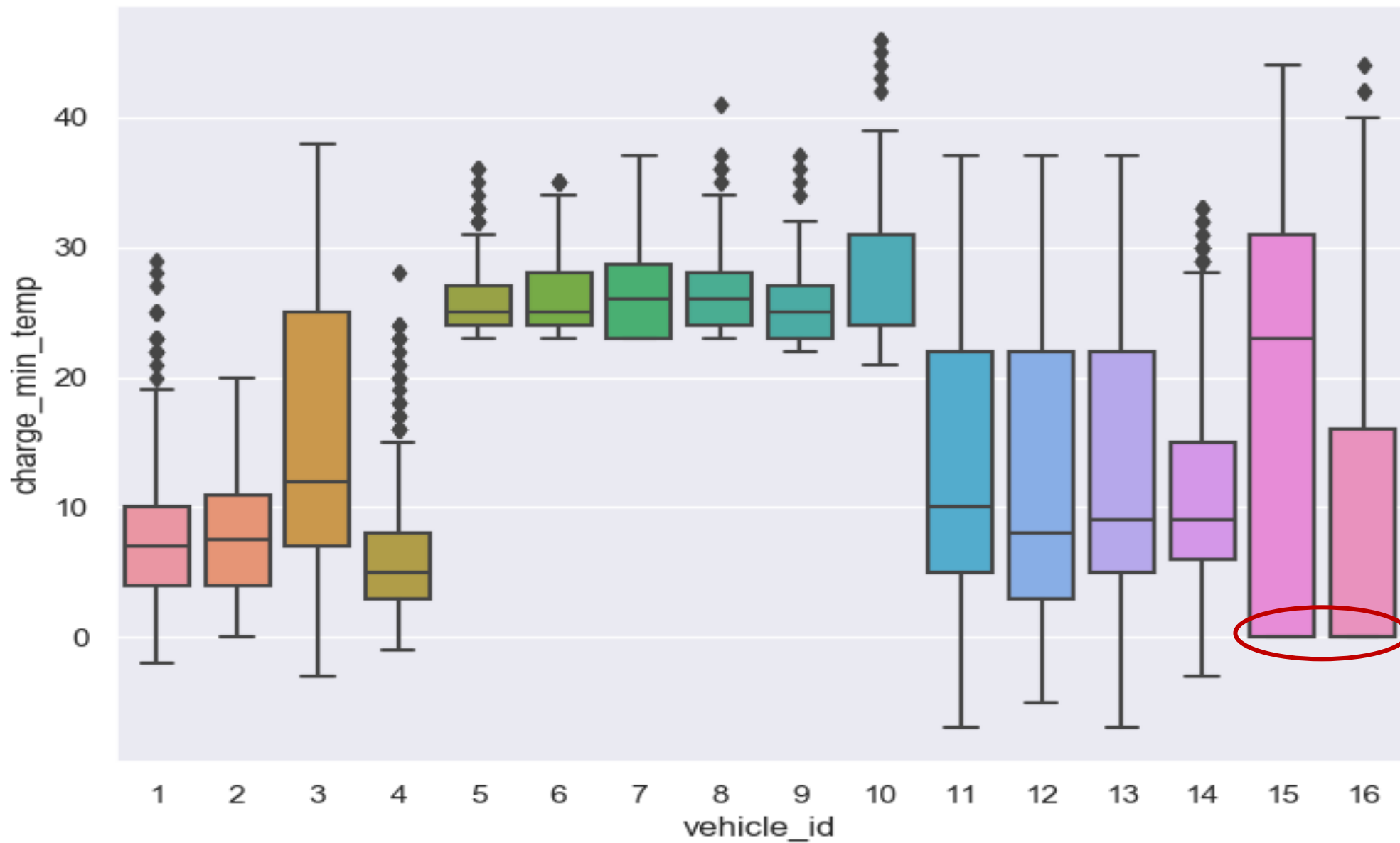
增加数据预处理工作量,
效果一般, 最后选择用
树算法进行提升

有效数据量依然足够

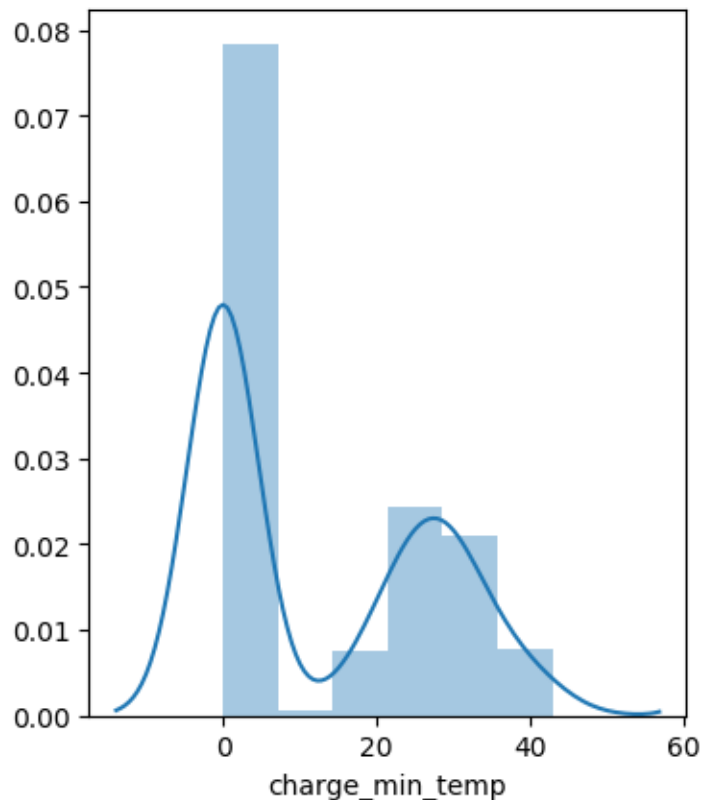
树型算法

- 善于处理缺失值
- 对离群点鲁棒

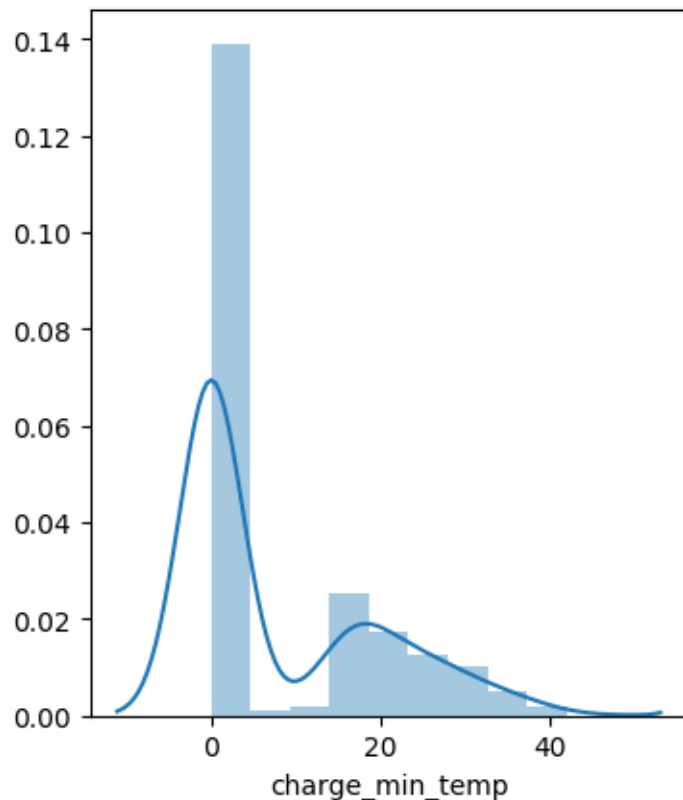
数据预处理要结合算法特性
不用overengineering, 人为引
入噪声



最低温度分布Box plot



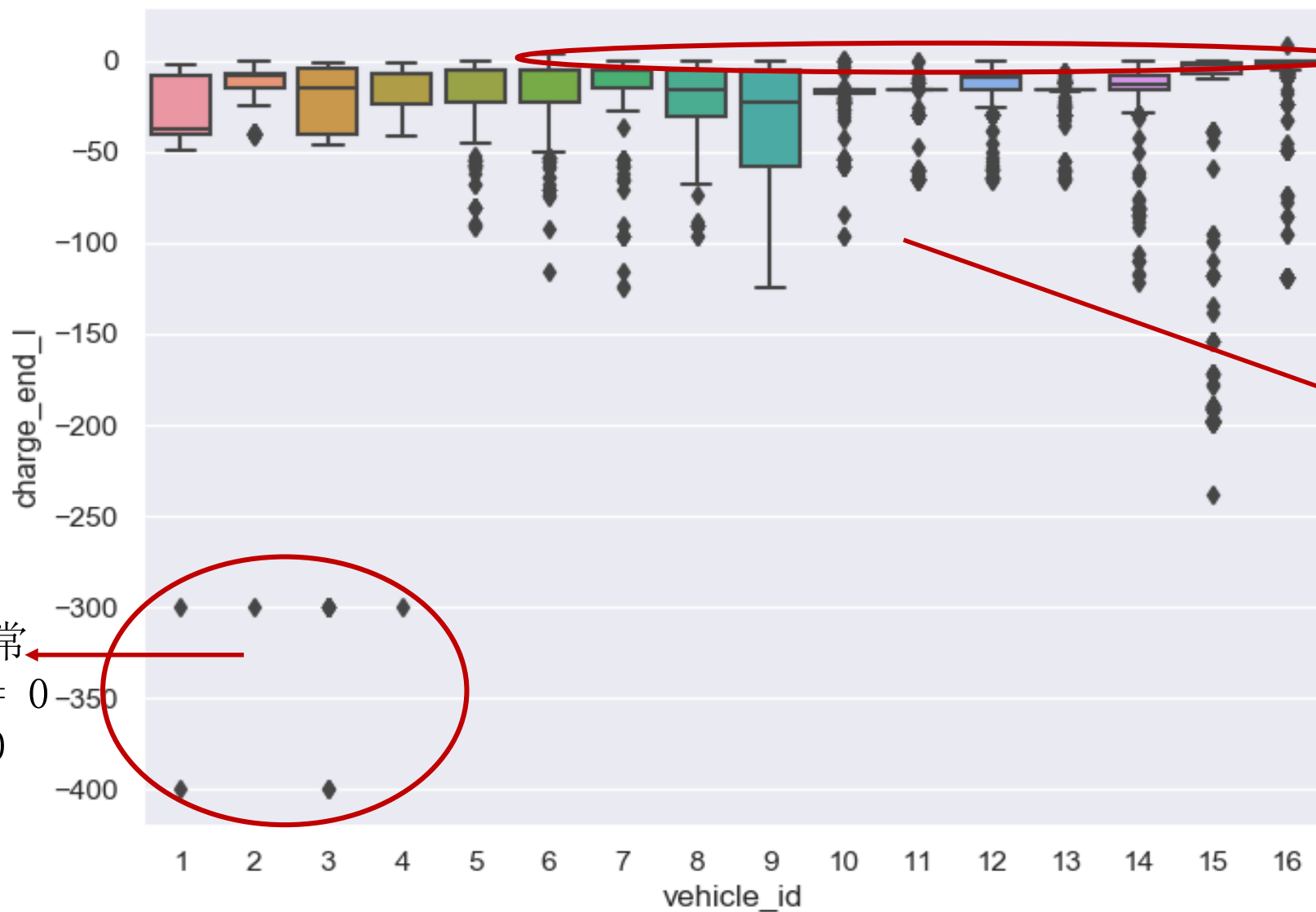
car15



car16

异常值修正

- car15 charge_min_temp为连续值，且为正态分布，使用均值修正，保持期望不变
- car16 charge_min_temp为连续值，且为长尾分布，使用中值修正，避免异常点影响
- 使用随机森林与梯度提升树进行异常点修正
- 取当月平均



一些充电结束电流
被记为0，尤其是
Car15, 16

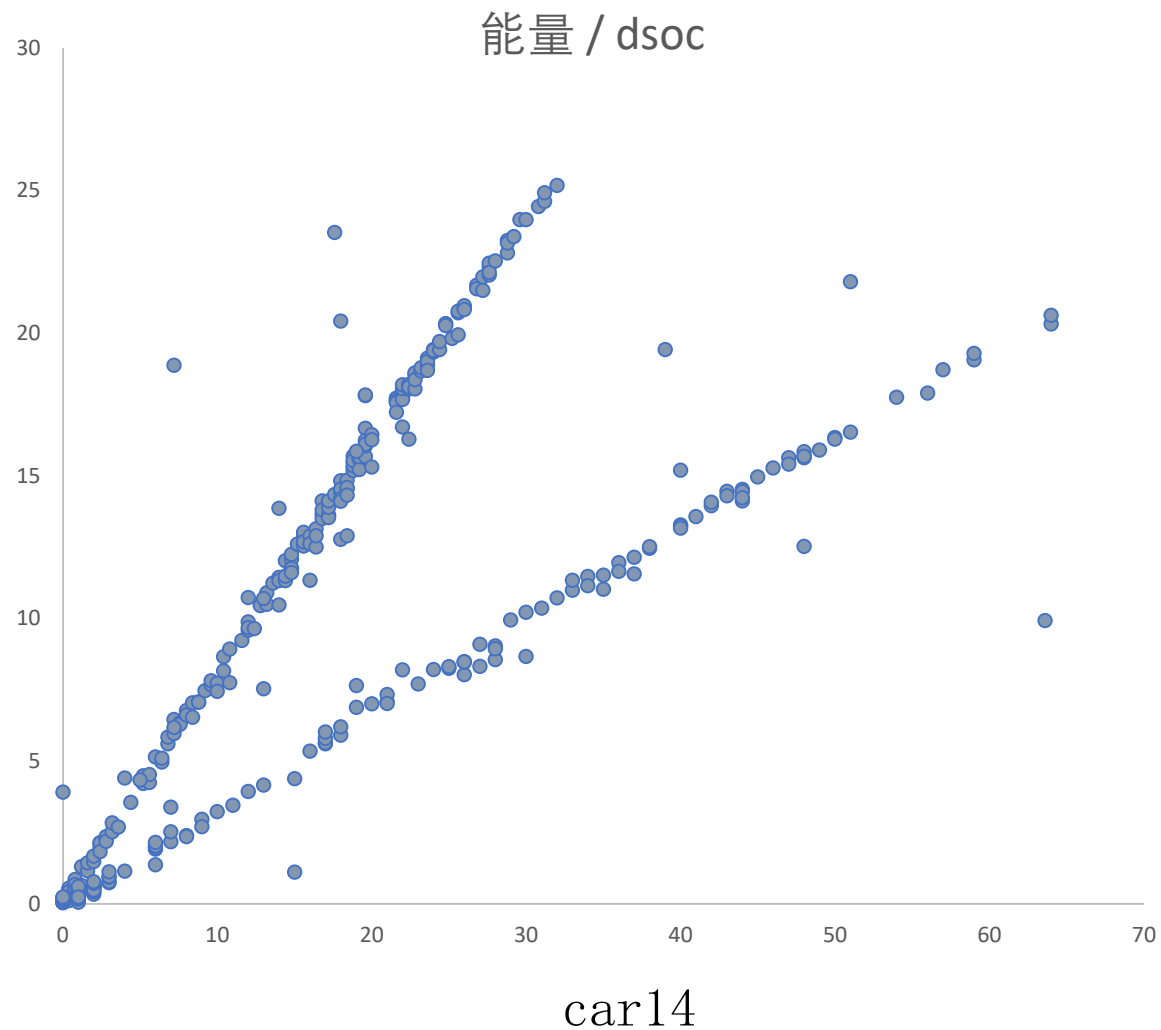
离群点 \neq 异常点
却会显著影响数据
分布，影响特征标
准化效果，信息量
却异常丰富

伴随性异常

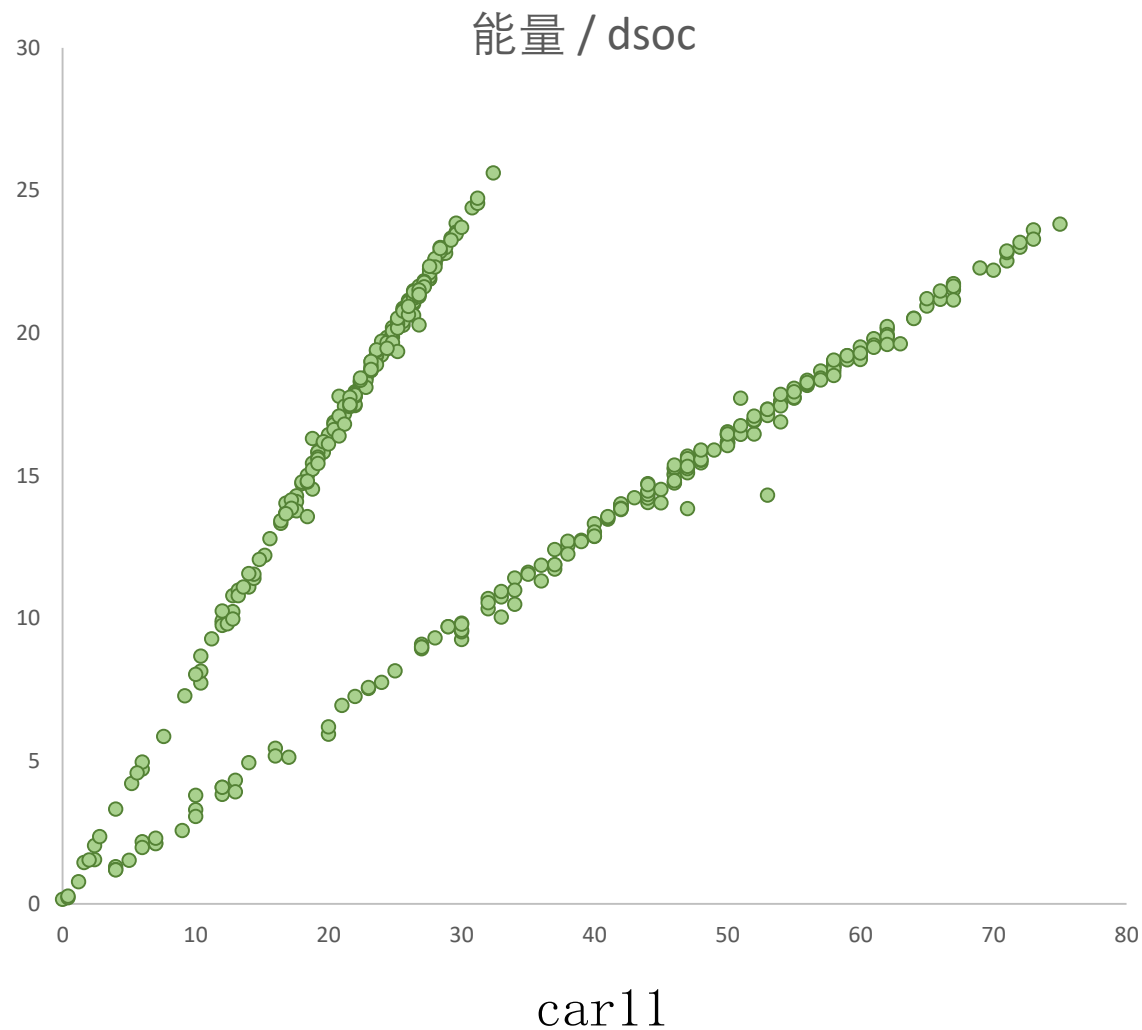
End_soc = 0

End_U = 0

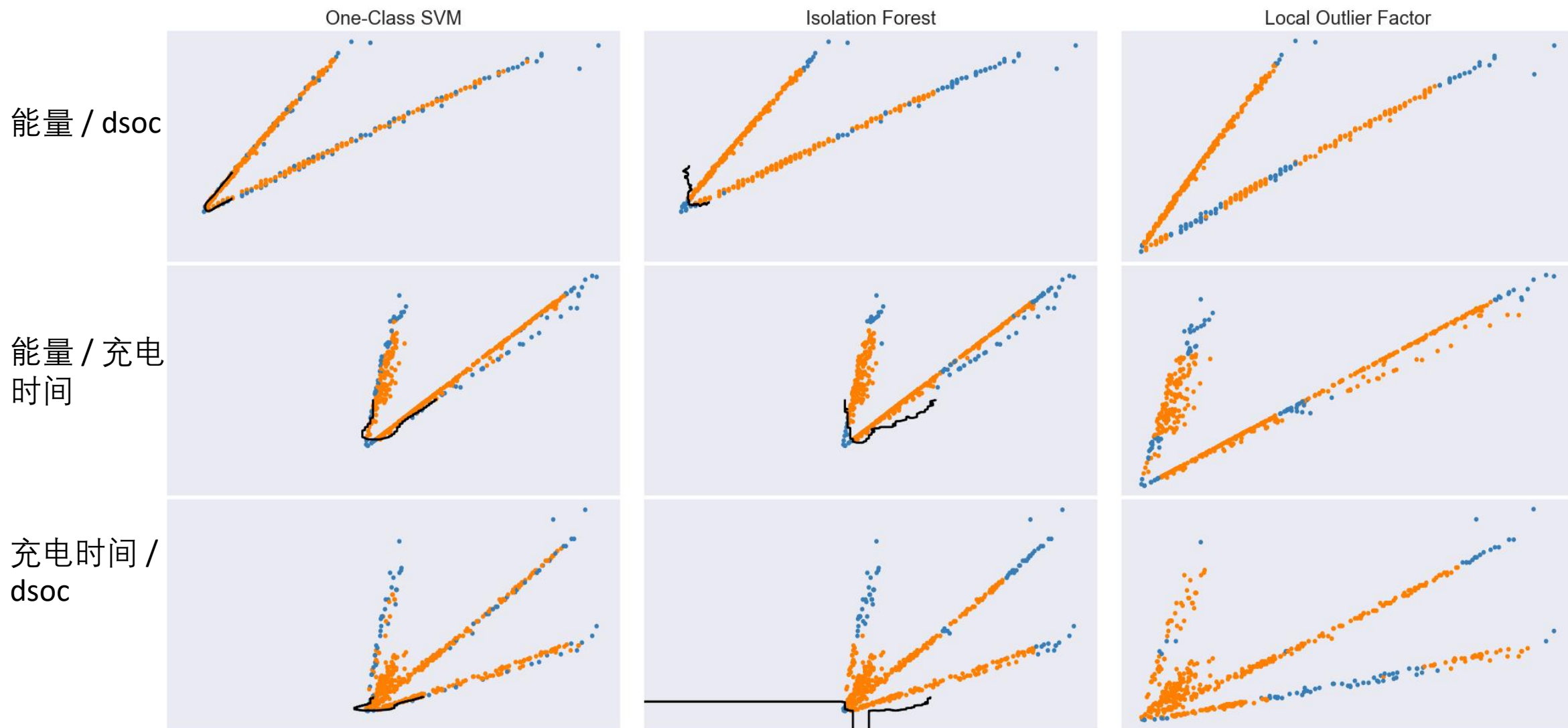
结束电流分布Box plot



结合新特征dsoc / charge_hour 剔除异常点



* dsoc 充电结束soc - 充电起始soc



在充电时间短，dsoc小的情况下，数据分布差异很大

car12



- 数据分析与清洗
- 模型设计
- 算法结构
- 可移植性与工程优化



一类基础特征群

原始特征



二类表征特征群

表征性特征



三类时间特征群

时间维特征





基础特征群

- 原始特征 12维

vehicle_id,
charge_start_time, charge_end_time
charge_start_soc, charge_end_soc,
charge_start_U, charge_end_U
charge_start_I, charge_end_I,
charge_max_temp, charge_min_temp,
mileage

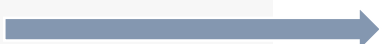


表征特征群

- 更有表现力的特征 11维
 - 差值特征: charge_hour, dsoc, dU, dtemp
 - 比率特征: $dU/dsoc$, $dsoc/hour$, $dmileage/soc$
 - 记忆特征: ddsoc, dmileage,
 - 类别特征: phase, charge_mode



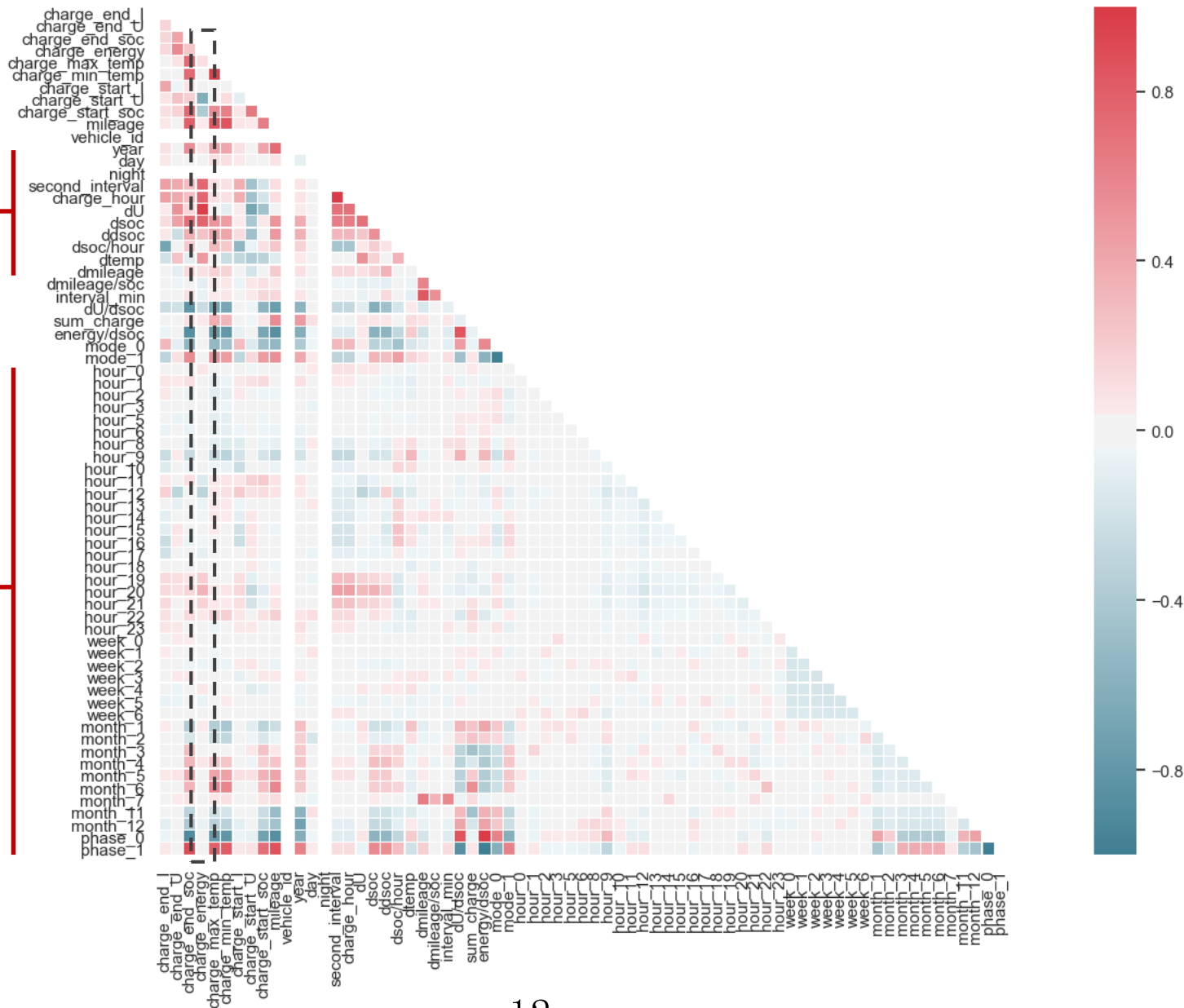
时间特征群

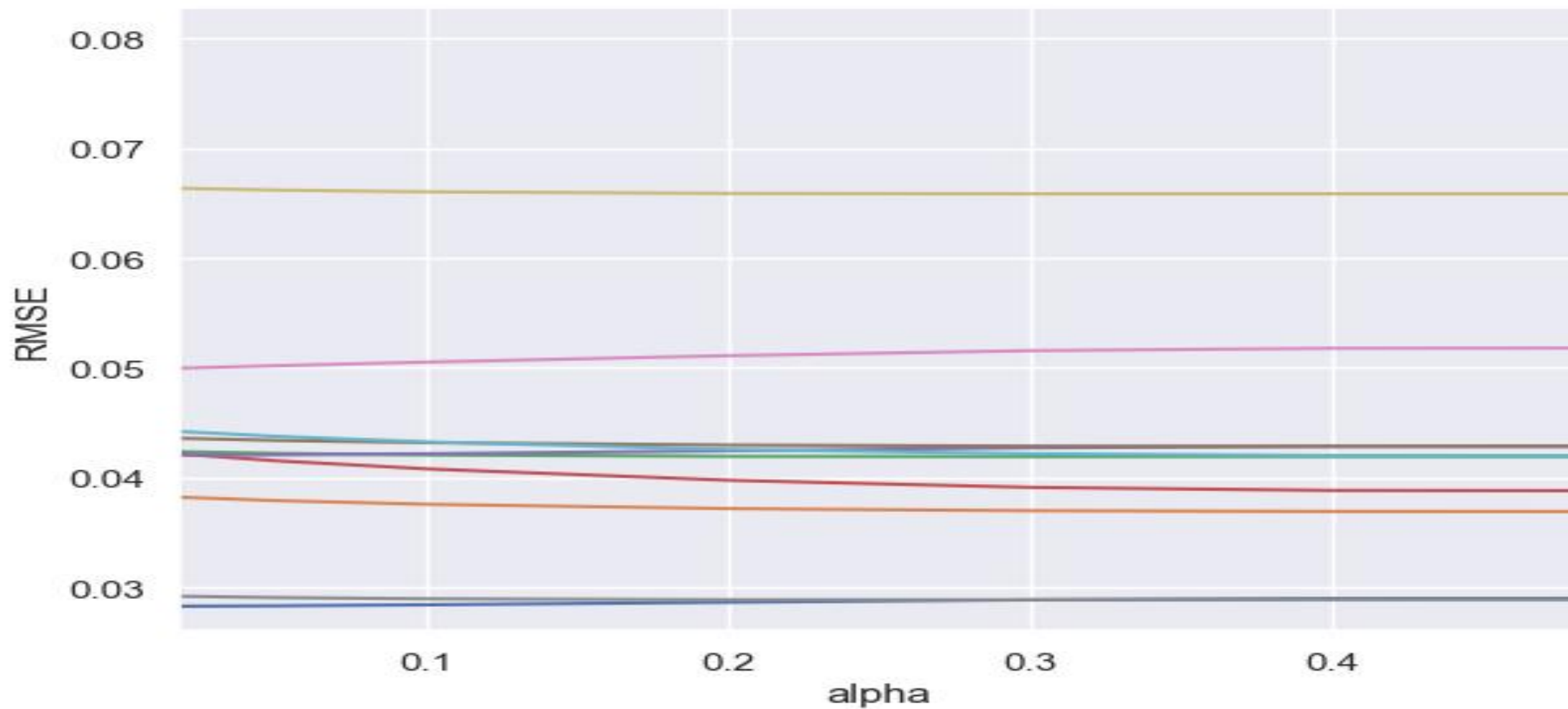
- 能量的时间维度特征
 - 时间维特征: year, month, day
 - 用户习惯特征: week, hour, night
 - 电池时间维特征: interval_min, sum_charge
- 
- one-hot 编码



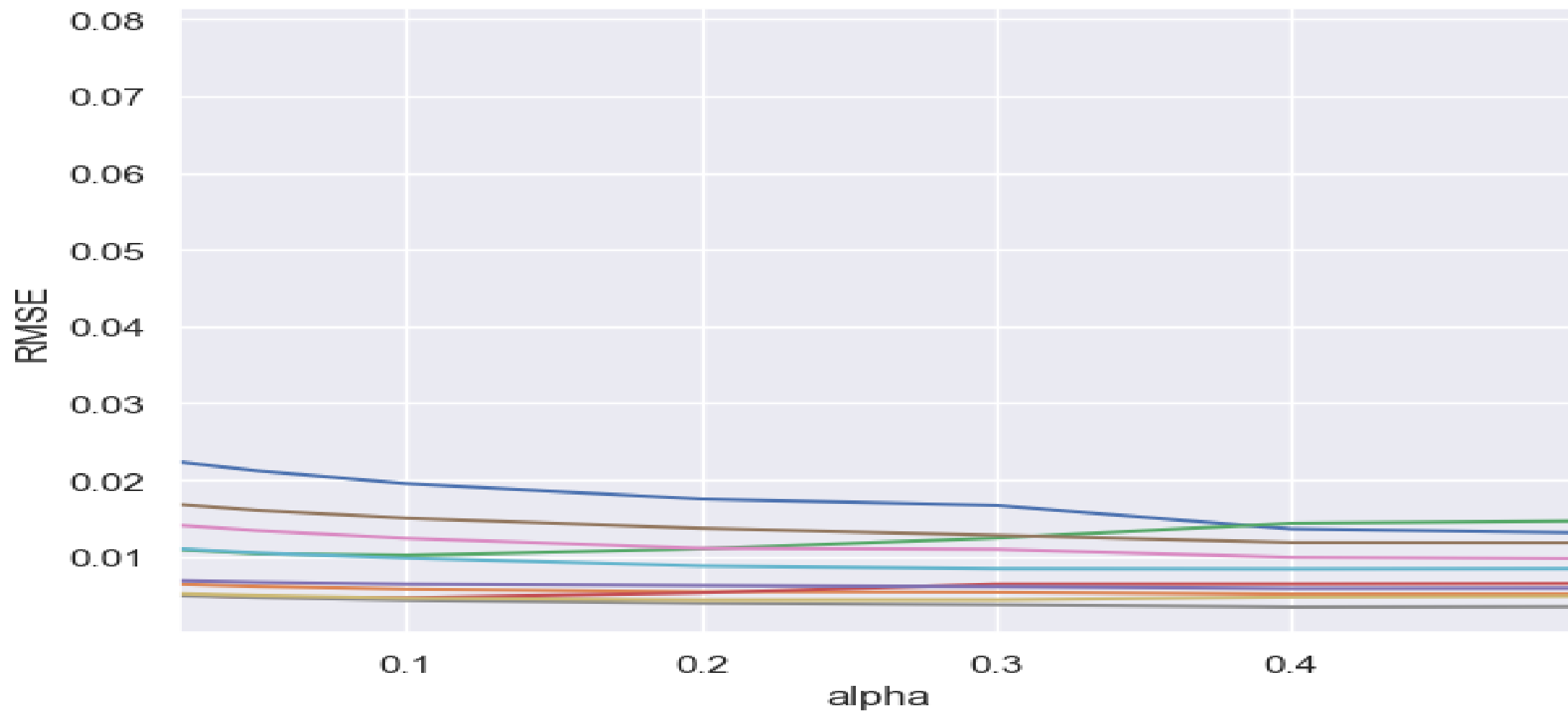
表征性特征与能量相关性更强

时间维特征one-hot编码后形成稀疏矩阵





Car12 交叉验证结果
原始特征



Car12 交叉验证结果

多维特征



泛化特征

```
select_list = [  
    'charge_hour',  
    'dsoc',  
    'dsoc/hour',  
    'charge_min_temp',  
    'charge_end_U',  
    'charge_start_U',  
    'dU',  
    'charge_start_I',  
    'charge_end_I']
```

回归方法对训练集的数据量要求高。百万级别的训练集都无法支持数百维特征进行训练。特征数量的限制导致无法充分利用信息，进而影响了回归模型的精度。

小能量特征

```
select_list = [  
    'charge_hour',  
    'dsoc',  
    'dtemp',  
    'dU',  
    'charge_start_I',  
    'charge_end_I']
```

决赛倒数第二天，引入的小能量模型，并进行系数融合后，效果提升9%，当天排名第一



gbm model

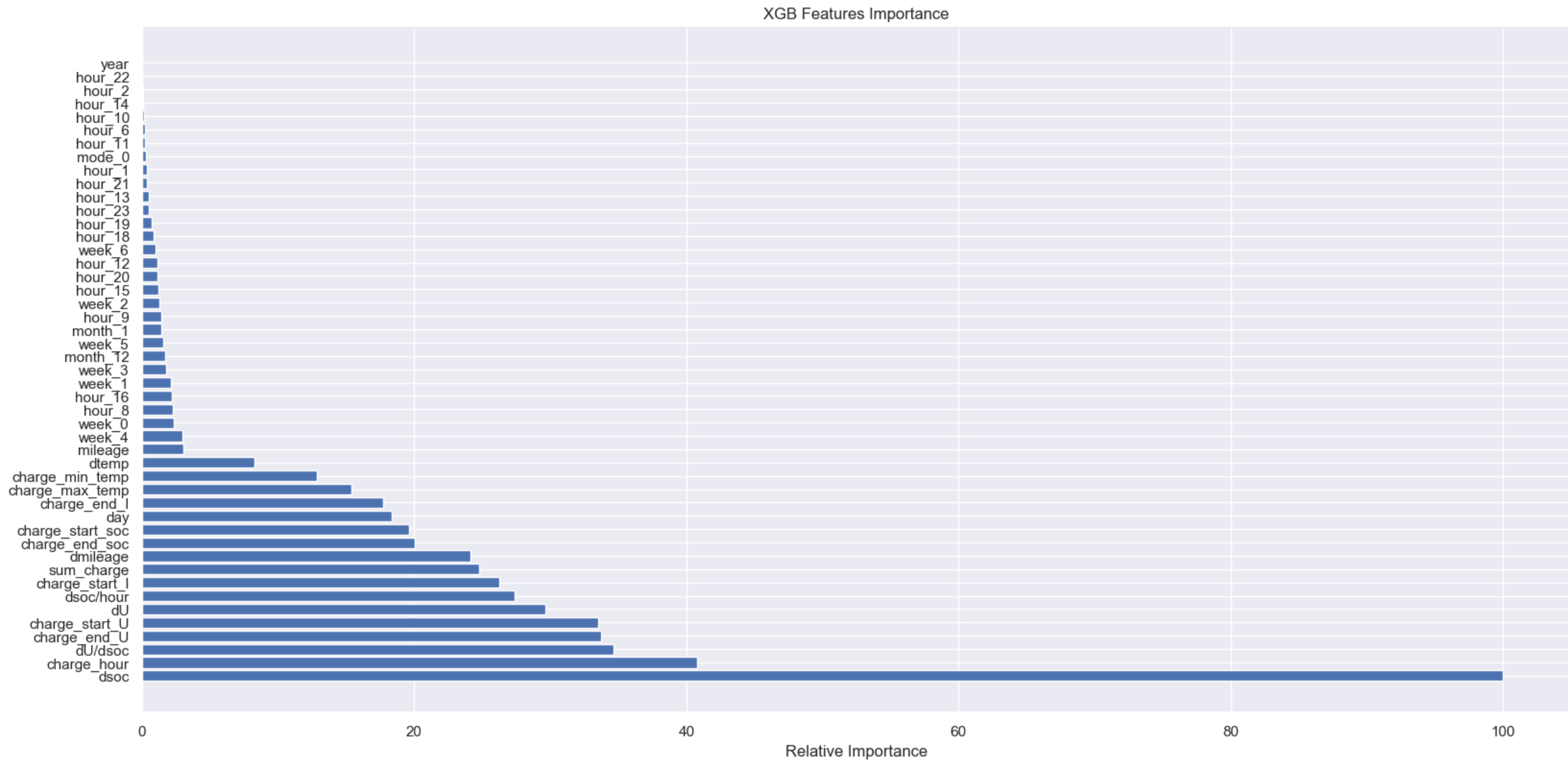
```
gbm_model = GradientBoostingRegressor(n_estimators=2000, max_depth=4,  
min_samples_split=2, min_samples_leaf=2, max_features='auto', subsample=0.6,  
learning_rate=0.008) # 少样本
```

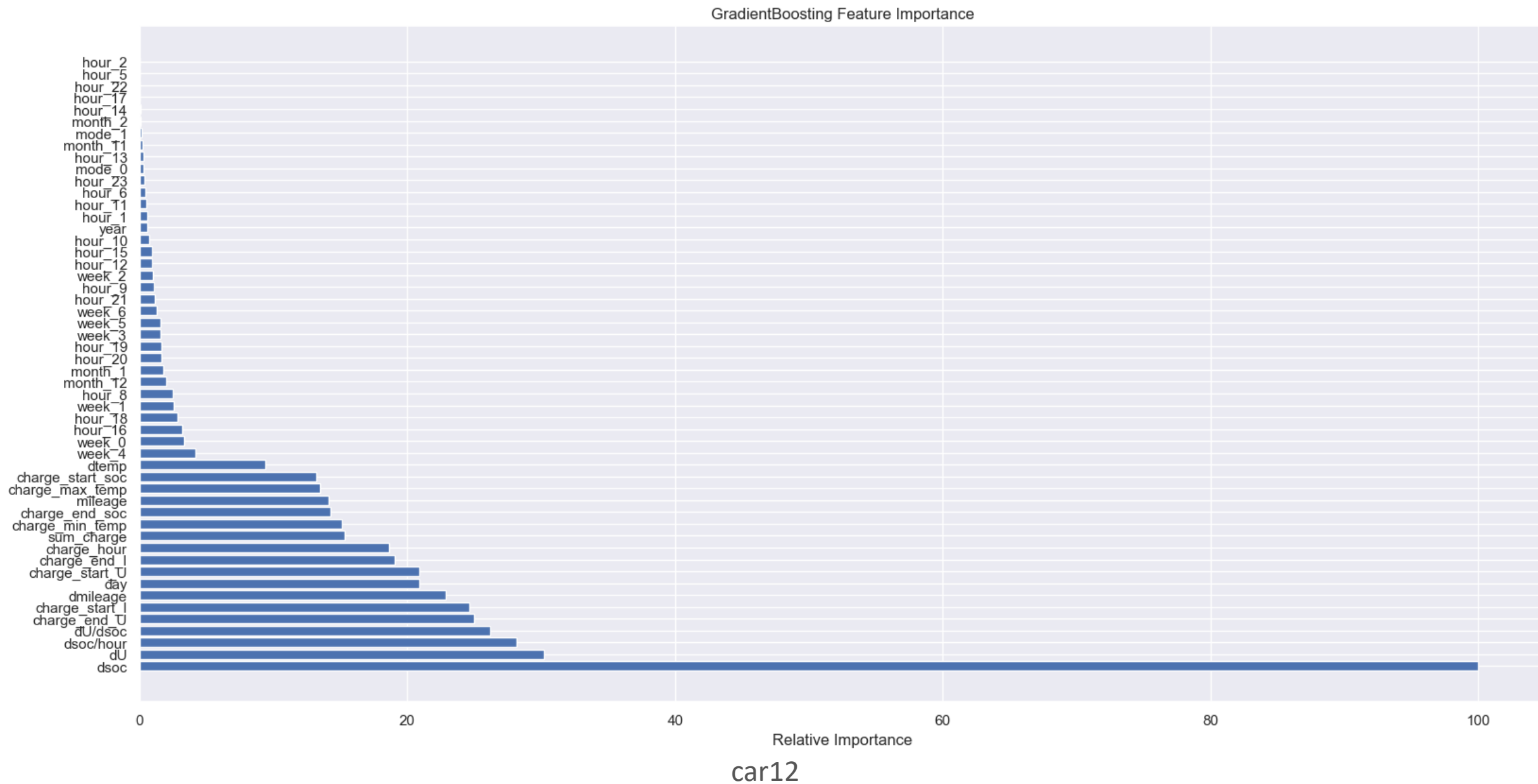
```
gbm_model = GradientBoostingRegressor(n_estimators=2000, max_depth=4,  
min_samples_split=15, min_samples_leaf=2, max_features='auto', subsample=0.6,  
learning_rate=0.008) # 多样本
```

xgboost_model

```
xgboost_model = xgb.XGBRegressor(max_depth=3, min_child_weight=0.9,  
gamma=0.0001, subsample=0.55, scale_pos_weight=1, learning_rate=0.008,  
reg_alpha=0.001, colsample_bytree=0.9, booster='gbtree', n_estimators=3000) # 泛化
```

```
xgboost_model = xgb.XGBRegressor(max_depth=6, min_child_weight=0.9,  
gamma=0.0001, subsample=0.55, scale_pos_weight=1, learning_rate=0.008,  
reg_alpha=0.001, colsample_bytree=0.9, booster='gbtree', n_estimators=3000) # 深层
```



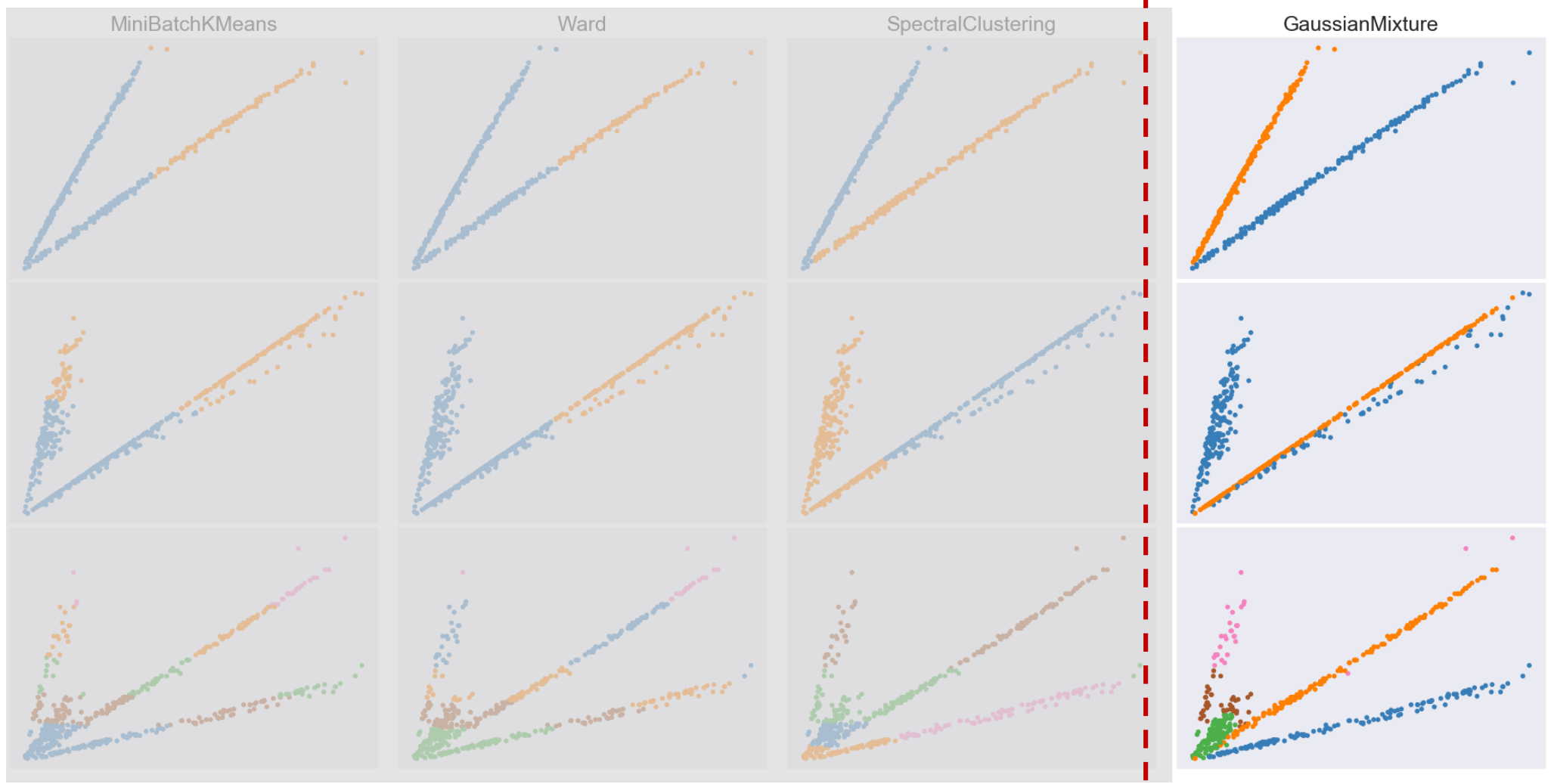


高斯混合模型

能量 / dsoc

能量 / 充电
时间

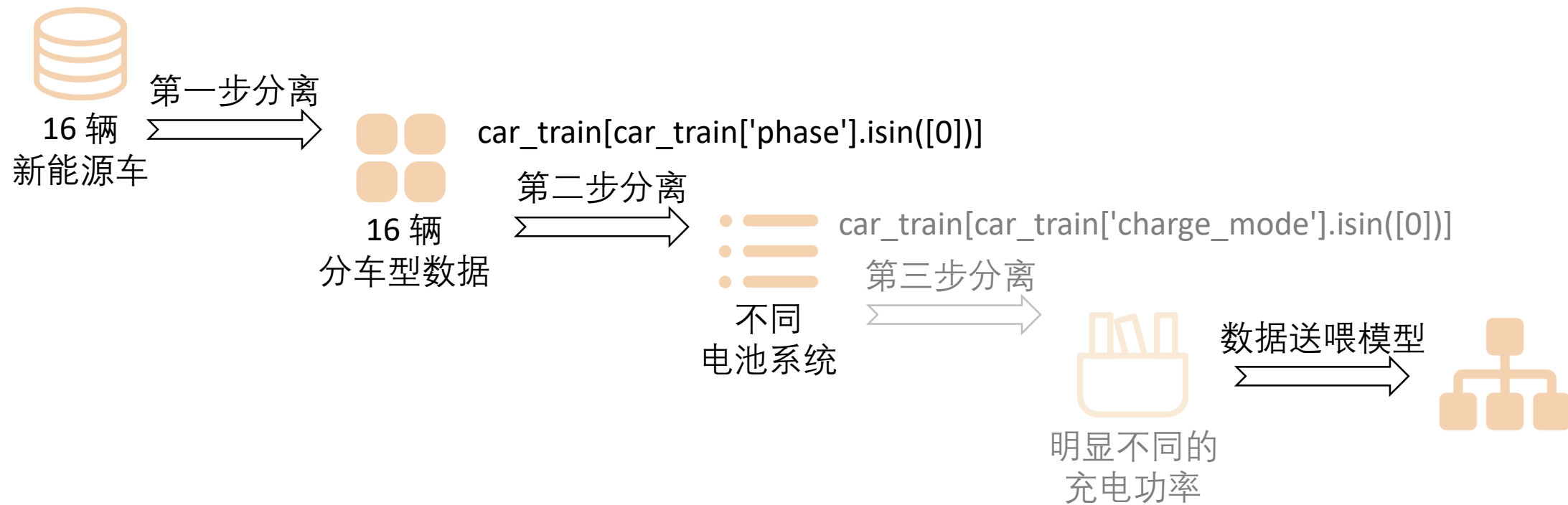
dsoc / 充电
时间



car12

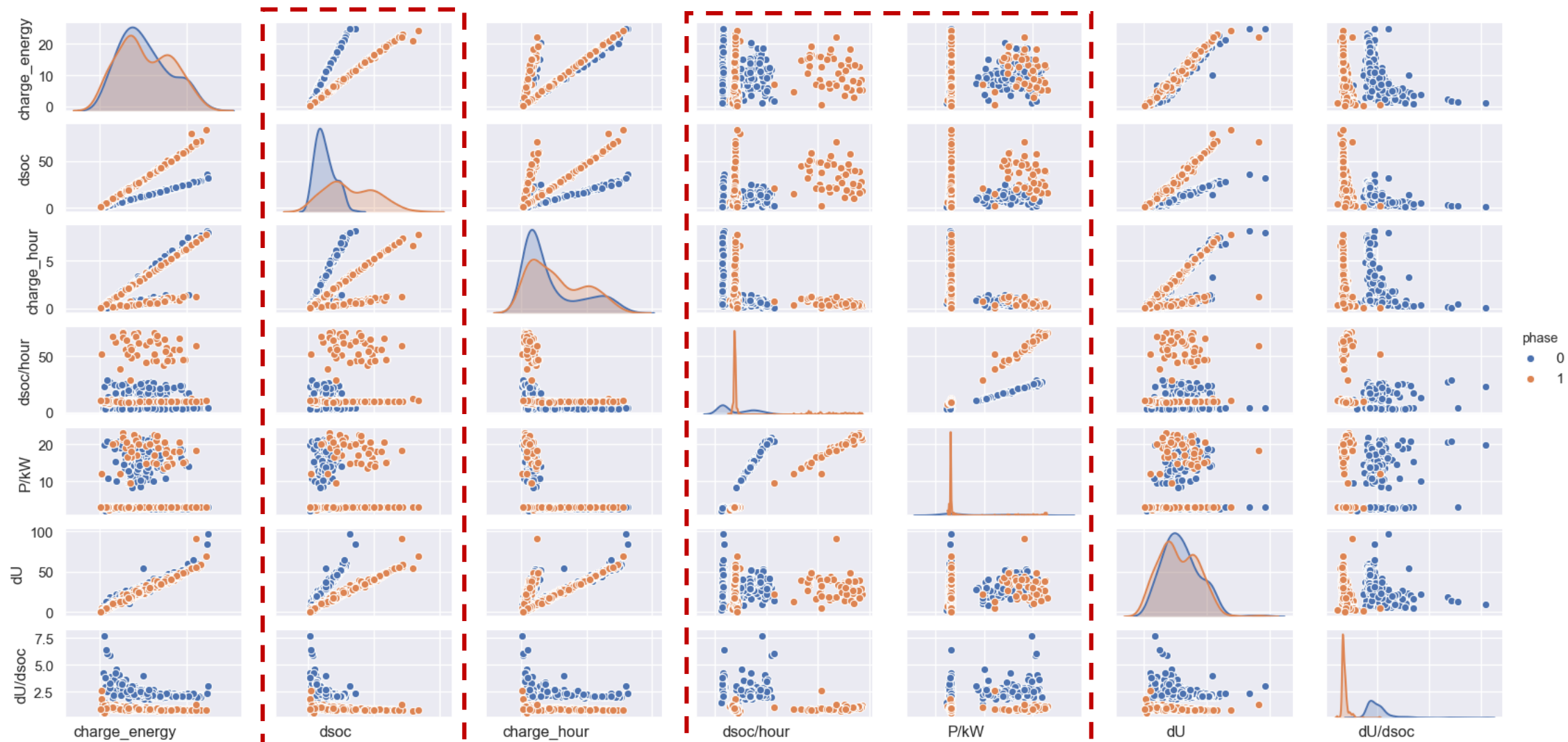


```
train_features[train_features['vehicle_id'].isin([1])]
```





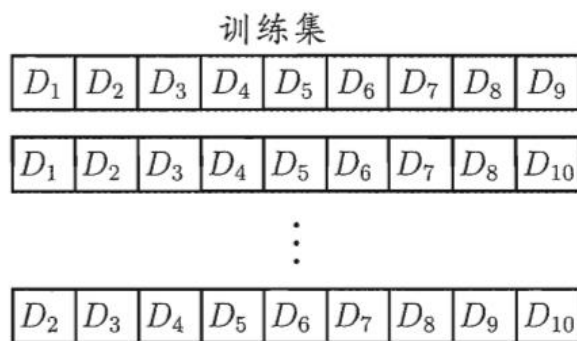
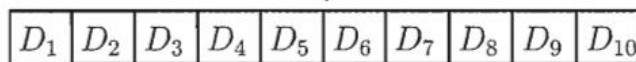
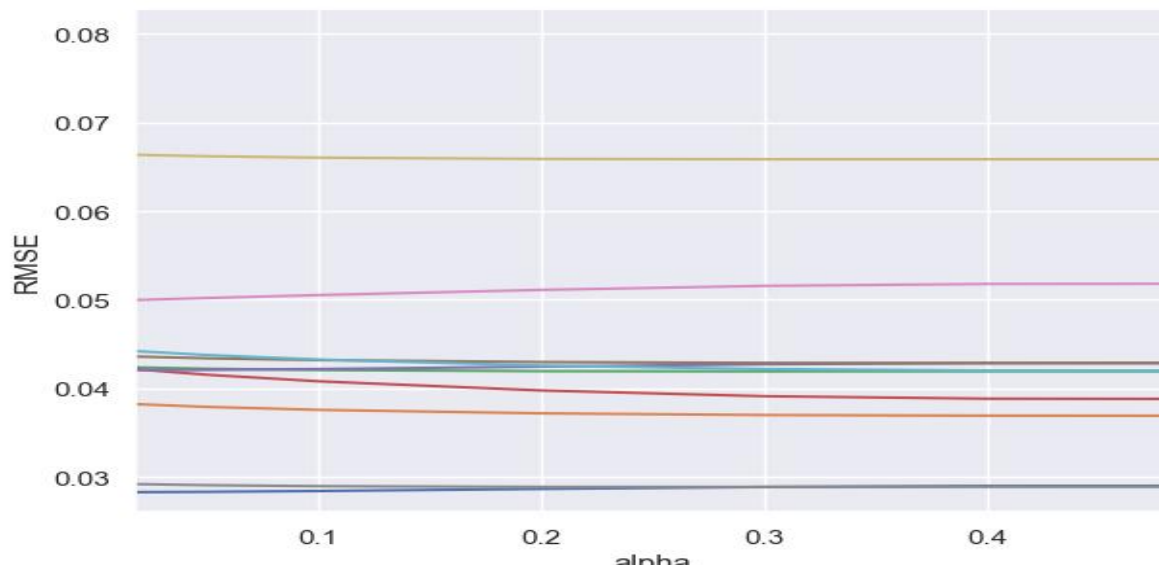
模型设计



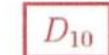
car12



- 数据分析与清洗
- 模型设计
- **算法结构**
- 可移植性与工程优化



测试集



→ 测试结果 1



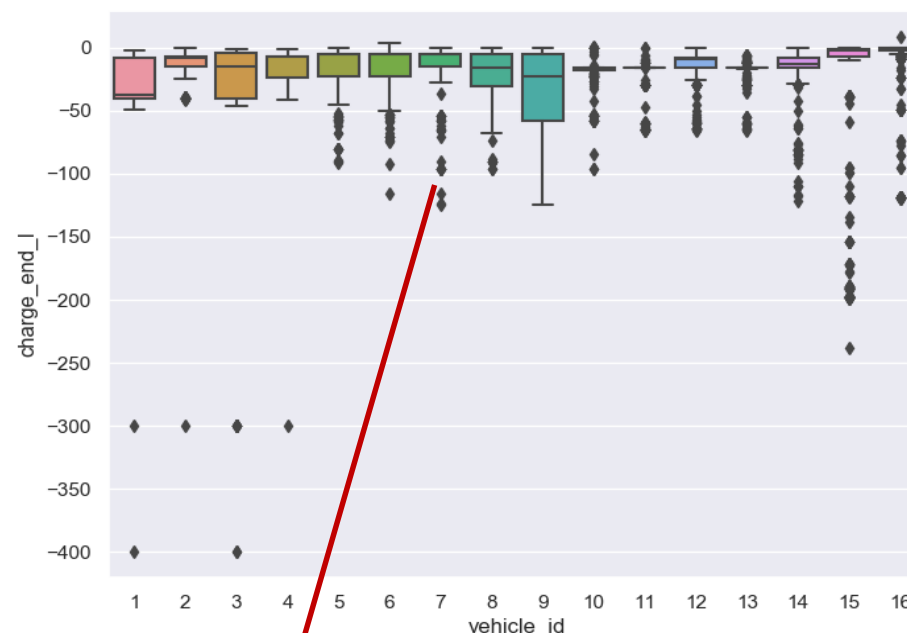
→ 测试结果 2



→ 测试结果 10

平均 返回结果

k折交叉验证与基于k折的交叉训练



对离群点采用鲁棒性更强的标准化方法。对每个特征独立地进行居中和缩放。离群特征点不影响标准化效果，并且可以保持离群特性。

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$



线性算法

Ridge Regression

用于存在多重共线性（自变量高度相关）数据。
L2正则化惩罚在收缩的时候权重会分散，让权重平方和变小。

Lasso Regression

预测的一组变量是高度相关的，Lasso 有助于特征选择
L1正则化惩罚在收缩的时候权重会集中，会出现稀疏解，为了稀疏而提取特征。

ElasticNet Regression

使用L1来训练并且L2优先作为正则化矩阵。
当有多个相关的特征时，Lasso 会随机挑选他们其中的一个，而ElasticNet则会选择两个。

SVM Regression

适用于高维特征空间，解决的是一个凸二次规划问题，对缺失值敏感

集成学习算法

Gradient boosting

能拟合复杂的非线性关系。
可以灵活处理各种类型的数据，包括连续值和离散值。
弱学习器之间存在依赖关系，容易过拟合

XGboost

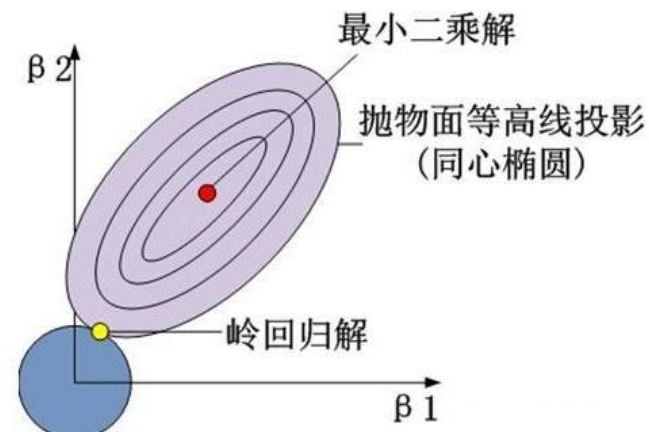
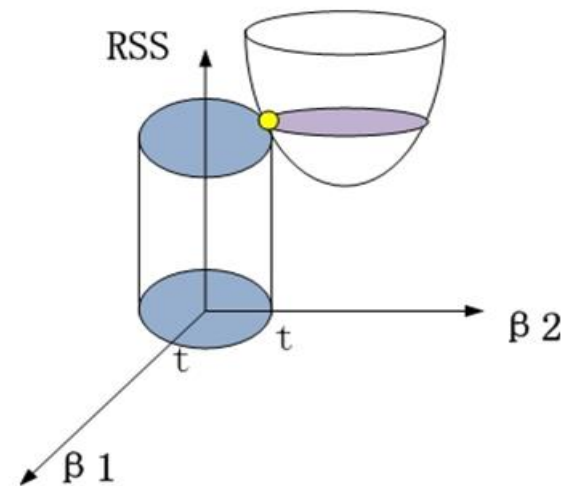
特征的值有缺失的样本。正则化，防止过拟合。并行处理。



Ridge Regression 岭回归

$$\min_{\omega} ||X\omega - y||_2^2 + \alpha ||\omega||_2^2$$

- 基于最小二乘法残差平方和最小的优化目标，引入L2正则惩罚项控制收缩量复杂度，使权重对共线性特征变得更加鲁棒。
- 为了收缩权重把(α * 权重) 添加到最小二乘项中以得到一个非常低的方差。
- * L2范数：表示向量x中元素的平方和再开平方，如欧式距离，度量向量间差异，如平方差和 L2范数功能：防止模型为了迎合模型训练而过于复杂出现过拟合能力，提高模型的泛化能力。

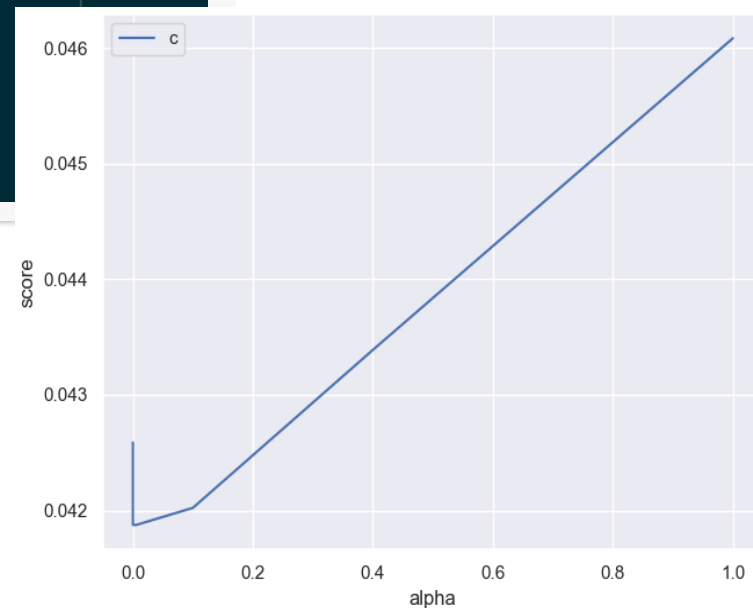
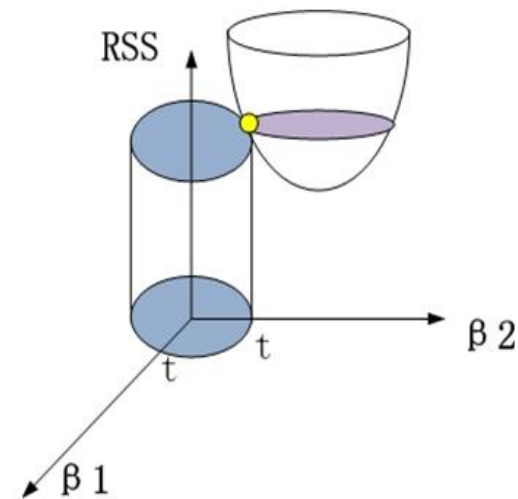




Ridge Regression 岭回归

$$\min_{\omega} ||X\omega - y||_2^2 + \alpha ||\omega||_2^2$$

```
#### Ridge 选取最佳参数
r_alphas = [0,0.00001,0.0001,0.0008,0.001,0.005,0.1,0.4,1,10,15,20,30,40,50]
ridge_scores = []
for alpha in r_alphas:
    score = ridge_selector(alpha, norm_X_train, train_target)
    ridge_scores.append(score)
ridge_score_table = pd.DataFrame(ridge_scores, r_alphas, columns=['Ridge_RMSE'])
print(ridge_score_table)
# 用最佳参数进行计算
r_alphas_best = [0.0008]
ridge = make_pipeline(RidgeCV(alphas = r_alphas_best, cv = kfold))
ridge_model_score = cv_rmse(ridge, norm_X_train, train_target)
plt.plot(r_alphas, ridge_scores, label='Ridge')
plt.legend('center')
plt.xlabel('alpha')
plt.ylabel('score')
print("ridge cv score: {0:.6f}".format(ridge_model_score.mean()))
```

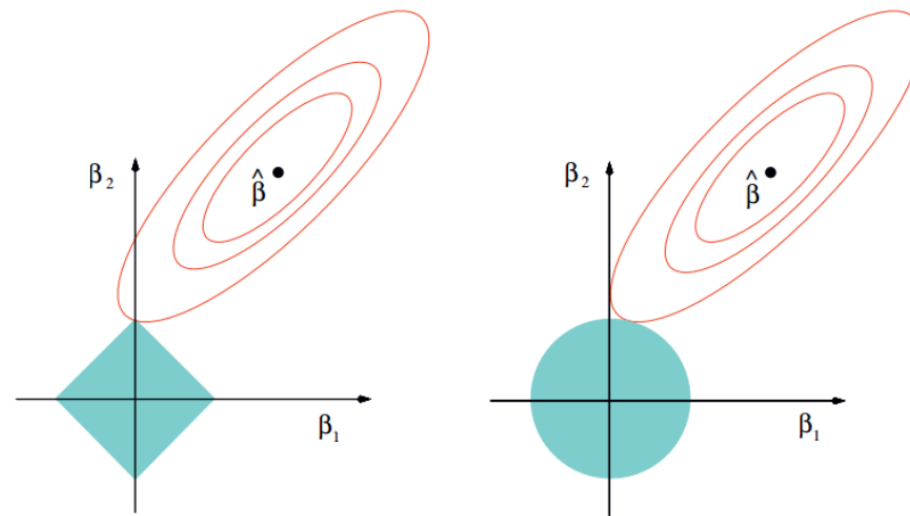
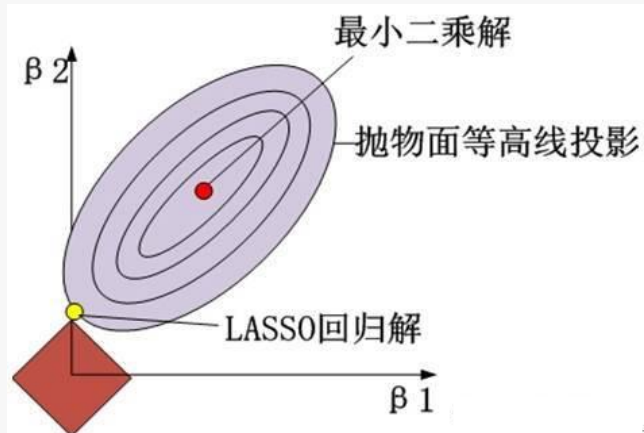




Lasso Regression Lasso回归

$$\min_m \left\{ \frac{1}{2N} \|X^T \omega - y\|_2^2 + \alpha \|\omega\|_1 \right\}$$

- 基于最小二乘法残差平方和最小的优化目标，引入L1正则惩罚项控制收缩量复杂度，容易产生稀疏的权重，达到变量选择的效果。
- * L1范数：表示向量x中非零元素的绝对值之和，又称为曼哈顿距离，最小绝对误差等，用于度量向量间差异，如：绝对误差和 x1和x2为两个向量 L1范数功能：可以实现特征稀疏，去掉一些没有信息的特征



等高线和约束域的切点就是目标函数的最优解。Lasso约束域是正方形，存在与坐标轴的切点，使得部分维度特征权重为0，产生稀疏的权重，达到变量选择的效果。

Ridge方法约束域是圆，其切点只存在圆周上，不与坐标轴相切，虽然也对原本的系数进行压缩，但任一维度上的取值都不为0，最终模型保留了所有的变量。

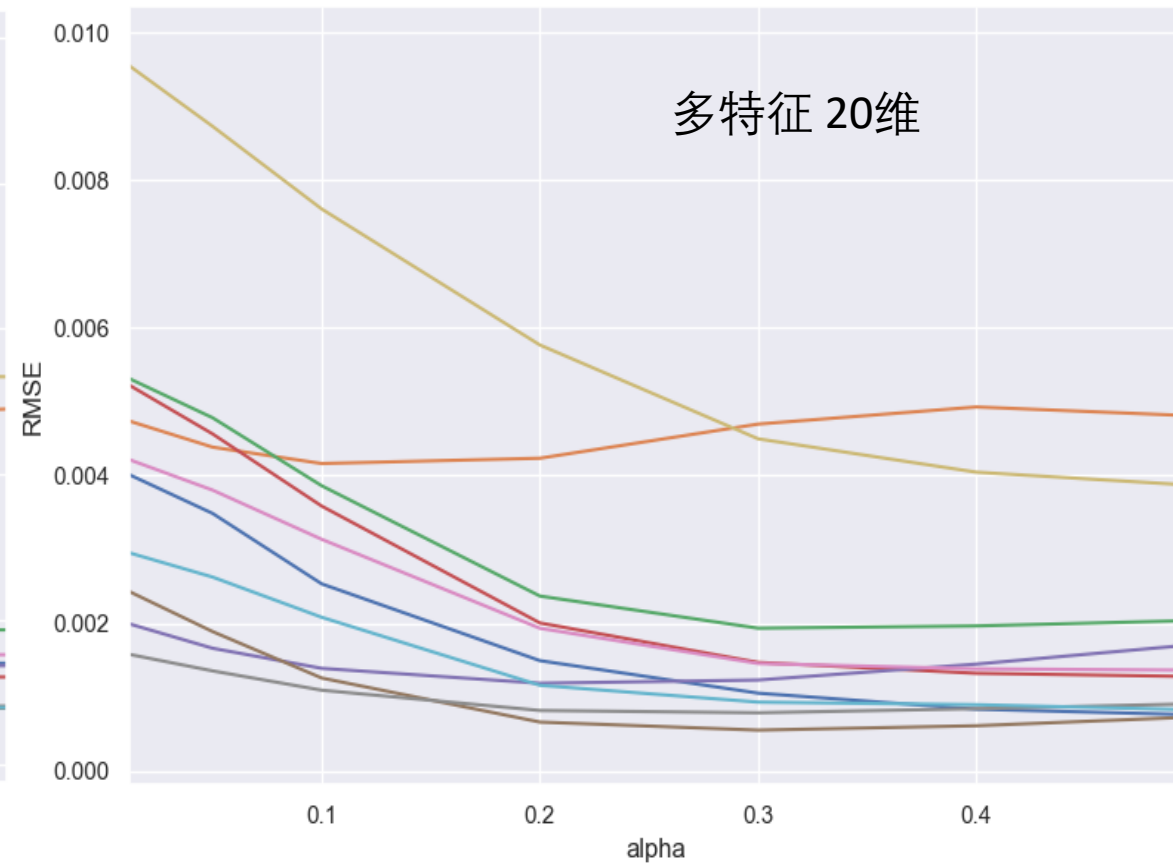
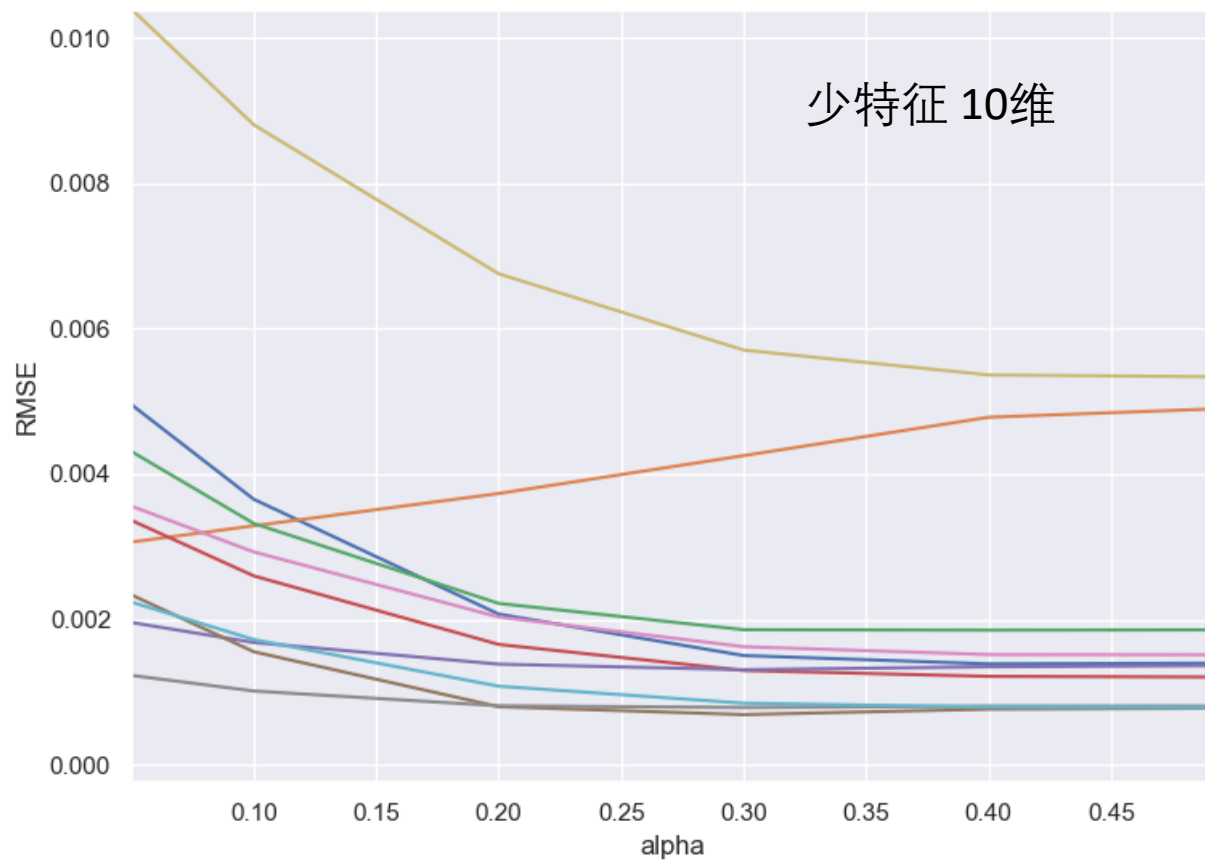


Lasso Regression

```
#### 分析Lasso k-fold 过拟合 特征
alphas_mse = [0.00001,0.0001,0.006,0.001,0.003,0.008,0.01,0.05,0.1,0.2,0.3,0.4,0.5]
lasso_model_mse = make_pipeline(RobustScaler(), LassoCV(max_iter=1e7, alphas = alphas_mse, cv = kfold
    )).fit(norm_X_train, train_target)
lasso_model_score = cv_rmse(lasso_model_mse, norm_X_train, train_target)
print("Lasso cv score: {0:.6f}".format(lasso_model_score.mean()))
lcv_scores = lasso_model_mse.steps[1][1].mse_path_
plt.plot(alphas_mse, lcv_scores, label='Lasso')
coeffs = pd.DataFrame(list(zip(norm_X_train.columns, lasso_model_mse.steps[1][1].coef_)), columns=['Features', 'Coefficients'])
used_coeffs = coeffs[coeffs['Coefficients'] != 0].sort_values(by='Coefficients', ascending=False)
print(used_coeffs.shape)
print(used_coeffs)
used_coeffs_values = norm_X_train[used_coeffs['Features']]
used_coeffs_values.shape
overfit_test2 = []
for i in used_coeffs_values.columns:
    counts2 = used_coeffs_values[i].value_counts()
    zeros2 = counts2.iloc[0]
    if zeros2 / len(used_coeffs_values) * 100 > 40:
        overfit_test2.append(i)
print('Overfit Features')
print(overfit_test2)
```

	Features	Coefficients
1	dsoc	0.530155
5	charge_end_U	0.064867
11	charge_end_I	0.030761
17	day	0.016488
0	charge_hour	0.016295
16	month	0.015879
9	dsoc/hour	0.008148
18	hour	0.007684
13	charge_max_temp	0.005540
3	charge_end_soc	0.000267
14	charge_min_temp	0.000127
10	charge_start_I	-0.026549
12	sum_charge	-0.057633
20	charge_mode	-0.076111
4	charge_start_U	-0.100276
2	charge_start_soc	-0.106421

Overfit Features
['charge_min_temp', 'charge_mode']





ElasticNet Regression 弹性网络

- 基于最小二乘法残差平方和最小的优化目标，同时引入L1和L2正则惩罚项控制收缩量复杂度，通过l1_ratio与alphas控制。

$$\min_{\omega} \left\{ \frac{1}{2n_{\text{samples}}} \|X\omega - y\|_2^2 + \underbrace{\alpha \rho \|\omega\|_1}_{\text{L1 penalty}} + \underbrace{\frac{\alpha(1-\rho)}{2} \|\omega\|_2^2}_{\text{L2 penalty}} \right\}$$



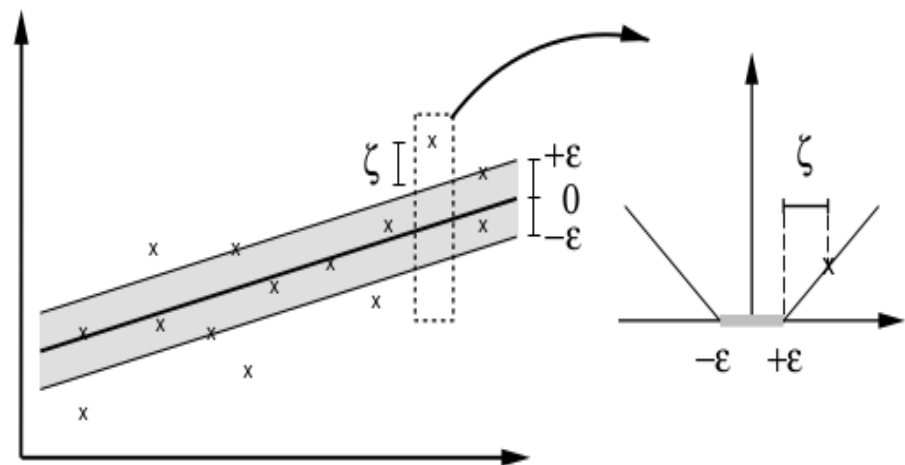
SVR Regression 支持向量机回归

$$\min_{\omega, b, \varepsilon_i, \hat{\varepsilon}_i} \frac{1}{2} \|\omega\|^2 + C \sum_{i=1}^m (\varepsilon_i + \hat{\varepsilon}_i)$$

支持向量回归能容忍 $f(x)$ 与 y 之间有 ε 的偏差，以 $f(x)$ 为中心，构建一个宽度为 2ε 的间隔带，若训练样本落入此间隔带，认为预测正确。

虚线区域外的数据点到虚线的边界的距离为残差，与线性模型类似，希望这些残差最小。

将实际问题通过非线性变换转换到高维的特征空间，在高维空间中构造线性决策函数来实现原空间中的非线性决策函数，巧妙地解决了维数问题，算法复杂度与样本维数无关。



适用于

- 高维特征空间，在数据维度比样本数量大的情况下仍然有效
- 解决的是一个凸二次规划问题，得到的将是全局最优解，解决了在神经网络方法中无法避免的局部极值问题

不适用于

- 要正确选择核函数
- SVM对缺失数据敏感



Gradient boosting 梯度提升树

集成学习法（Ensemble），它用不同的权重将基学习器进行线性组合，使表现优秀的学习器得到重用。根据初始模型计算伪残差，之后建立一个基学习器来解释伪残差，该基学习器是在梯度方向上减少残差。再将基学习器乘上权重系数（学习速率）和原来的模型进行线性组合形成新的模型。这样反复迭代就可以找到一个使损失函数的期望达到最小的模型。

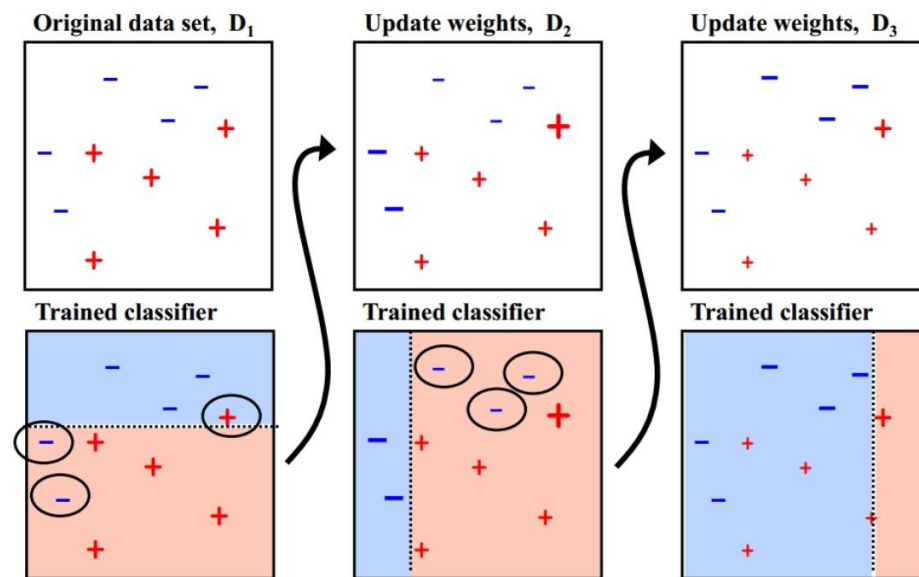
$$F_{m+1}(x) = F_m(x) + h(x)$$

适用于：

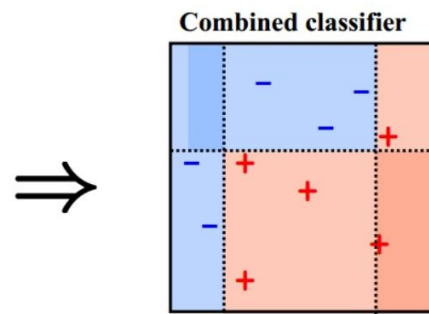
- 拟合复杂的非线性关系。
- 可以灵活处理各种类型的数据，包括连续值和离散值。

不适用于：

- 弱学习器之间存在依赖关系，容易出现过拟合现象。
- 抗干扰能力不强。



$$.33 * \begin{array}{|c|} \hline \text{Blue} \\ \hline \text{Red} \\ \hline \end{array} + .57 * \begin{array}{|c|} \hline \text{Blue} \\ \hline \text{Red} \\ \hline \end{array} + .42 * \begin{array}{|c|} \hline \text{Blue} \\ \hline \text{Red} \\ \hline \end{array} \geq 0$$





XGboost

集成学习法（Ensemble），将基学习器多棵树的预测加到一起，得到最终结果。首先使用训练集和样本真值（即标准答案）训练一棵树，然后使用这棵树预测训练集，得到每个样本的预测值，由于预测值与真值存在偏差，所以二者相减可以得到“残差”。接下来训练第二棵树，此时不再使用真值，而是使用残差作为标准答案。两棵树训练完成后，可以再次得到每个样本的残差，然后进一步训练第三棵树，以此类推。

适用于：

- 特征的值有缺失的样本。对于特征的值有缺失的样本，XGboost可以自动学习出它的分裂方向。
- 正则化，防止过拟合。XGboost在代价函数里加入了正则项，用于控制模型的复杂度。
- 并行处理。

$$\text{obj}(\theta) = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

不适用于：

- 模型复杂
- 解释性差
- 维护成本高



线性算法

Ridge Regression

用于存在多重共线性（自变量高度相关）数据。
L2正则化惩罚在收缩的时候权重会分散，让权重平方和变小。

Lasso Regression

预测的一组变量是高度相关的，Lasso 有助于特征选择
L1正则化惩罚在收缩的时候权重会集中，会出现稀疏解，为了稀疏而提取特征。

ElasticNet Regression

使用L1来训练并且L2优先作为正则化矩阵。
当有多个相关的特征时，Lasso 会随机挑选他们其中的一个，而ElasticNet则会选择两个。

SVM Regression

适用于高维特征空间，解决的是一个凸二次规划问题，对缺失值敏感

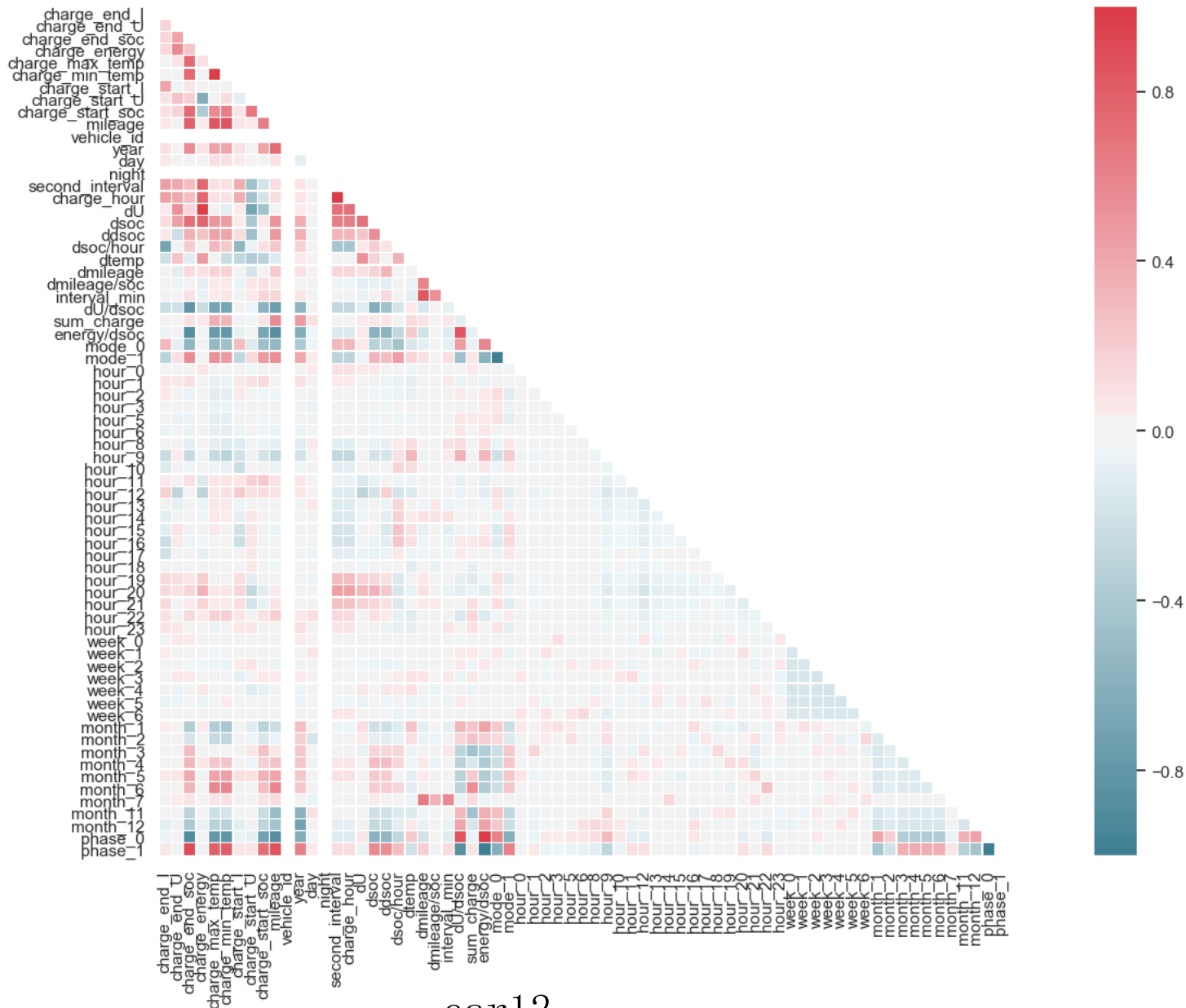
集成学习算法

Gradient boosting

能拟合复杂的非线性关系。
可以灵活处理各种类型的数据，包括连续值和离散值。
弱学习器之间存在依赖关系，容易过拟合

XGboost

特征的值有缺失的样本。正则化，防止过拟合。并行处理。





Car 10

训练集原始数据: 187

数据处理后: 180

特征维度: 9

'charge_hour', 'dsoc', 'dsoc/hour', 'charge_min_temp',

'charge_end_U', 'charge_start_U', 'dU',

'charge_start_I', 'charge_end_I'

测试集数据: 14

特征维度: 9

$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,9} \\ \vdots & \ddots & \vdots \\ x_{180,1} & \cdots & x_{180,9} \end{pmatrix}$$

K-fold

10折



K-fold随机打散

$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,9} \\ \vdots & \ddots & \vdots \\ x_{18,1} & \cdots & x_{18,9} \end{pmatrix}$$

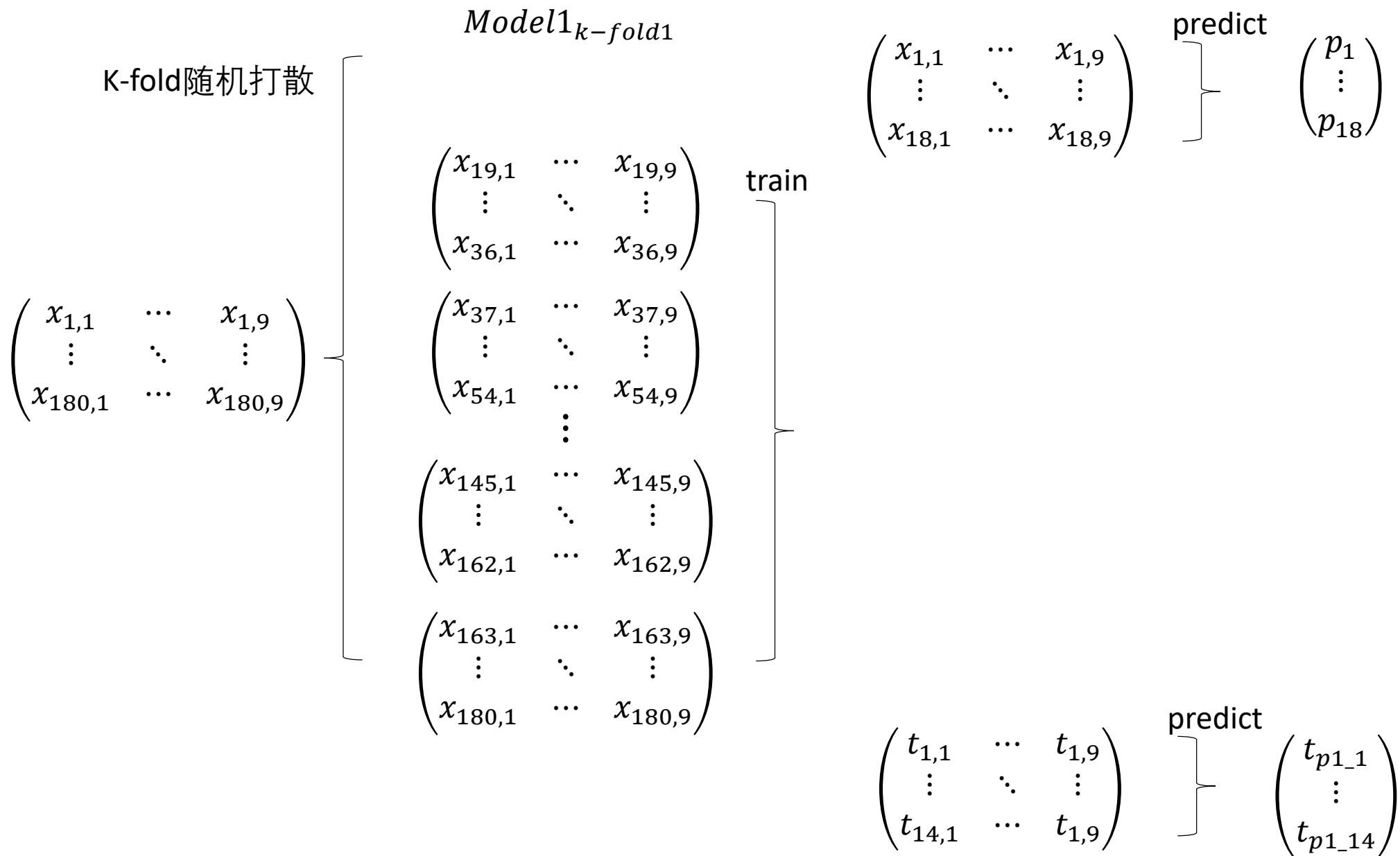
$$\begin{pmatrix} x_{19,1} & \cdots & x_{19,9} \\ \vdots & \ddots & \vdots \\ x_{36,1} & \cdots & x_{36,9} \end{pmatrix}$$

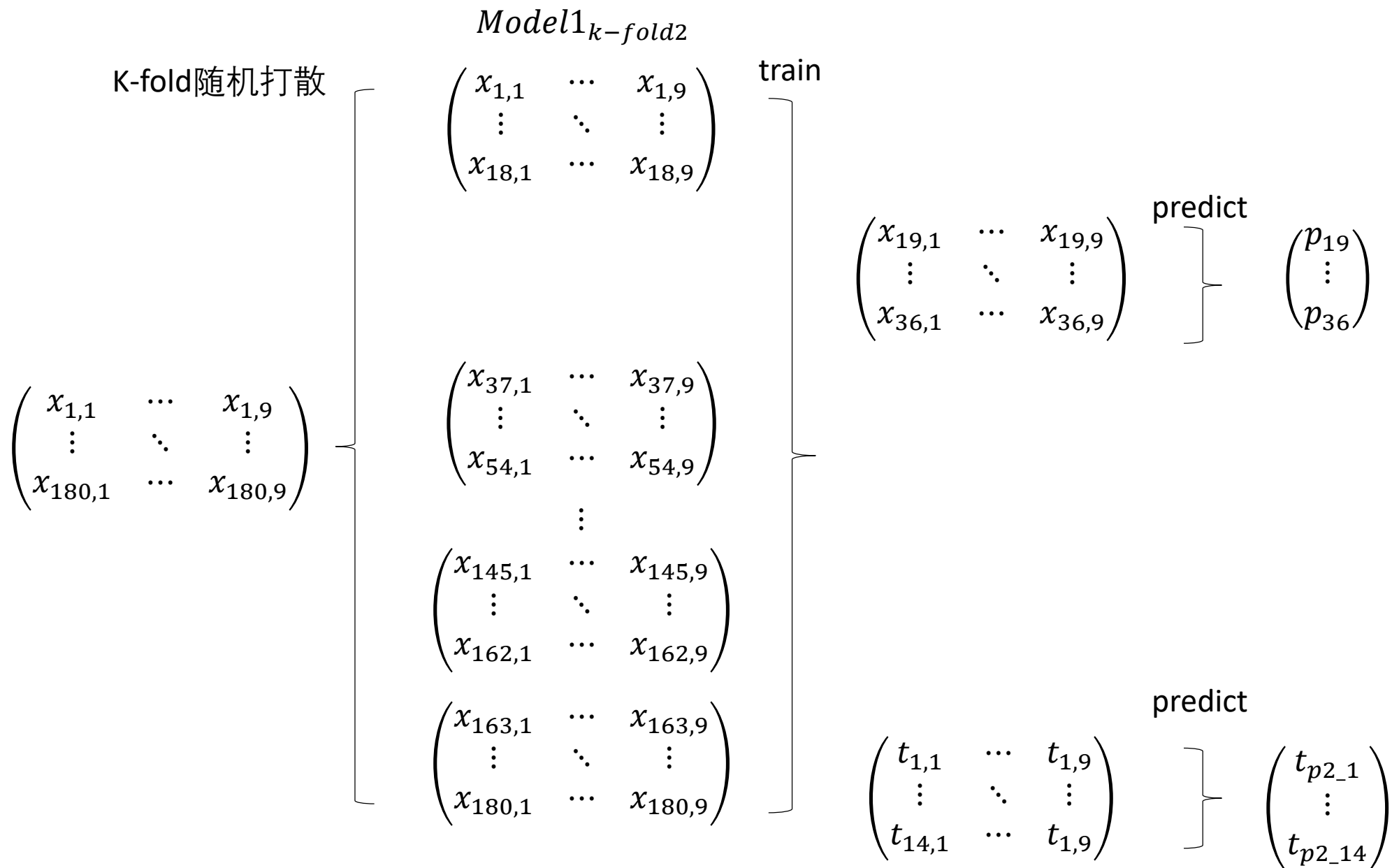
$$\begin{pmatrix} x_{37,1} & \cdots & x_{37,9} \\ \vdots & \ddots & \vdots \\ x_{54,1} & \cdots & x_{54,9} \end{pmatrix}$$

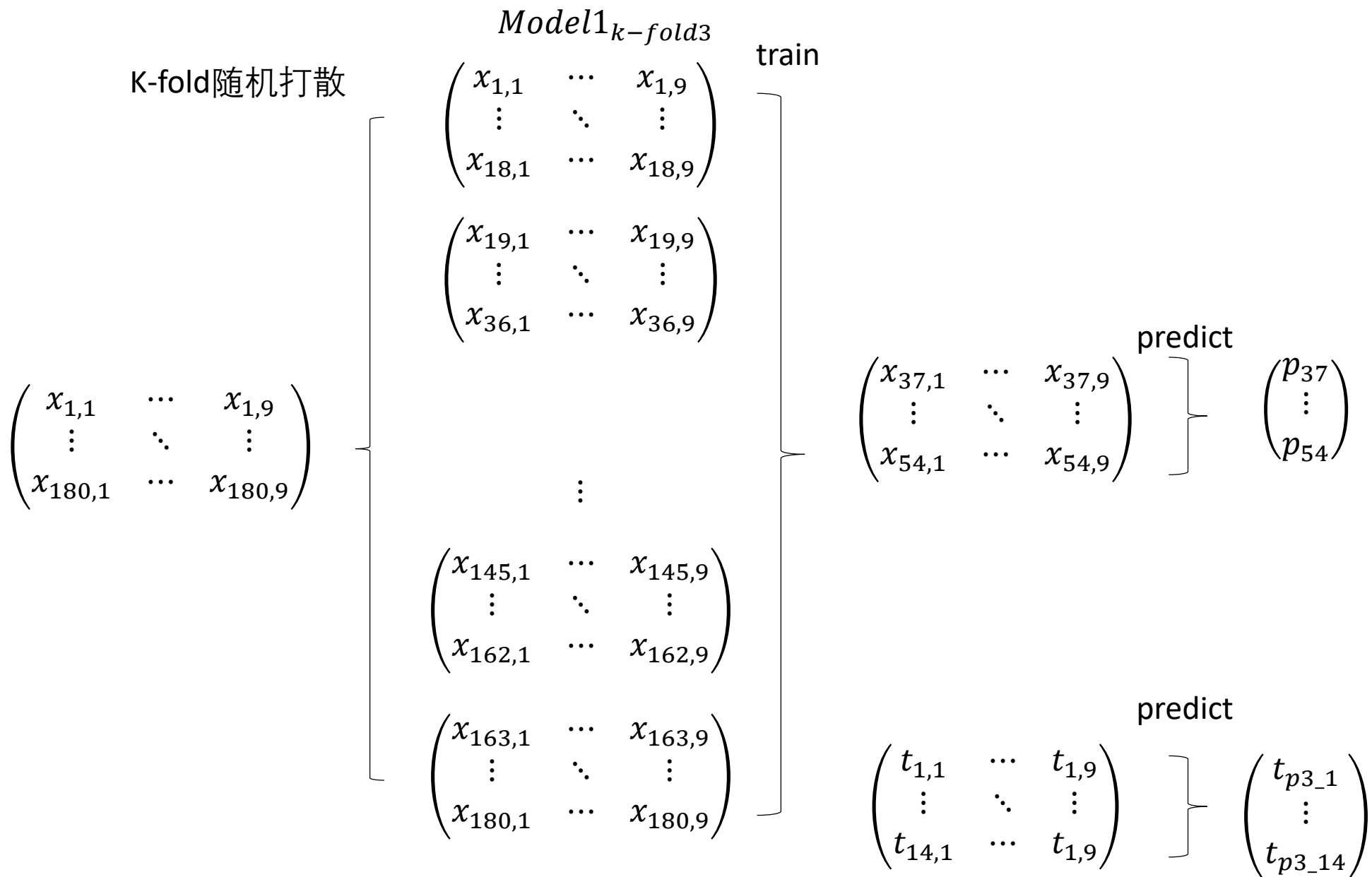
\vdots

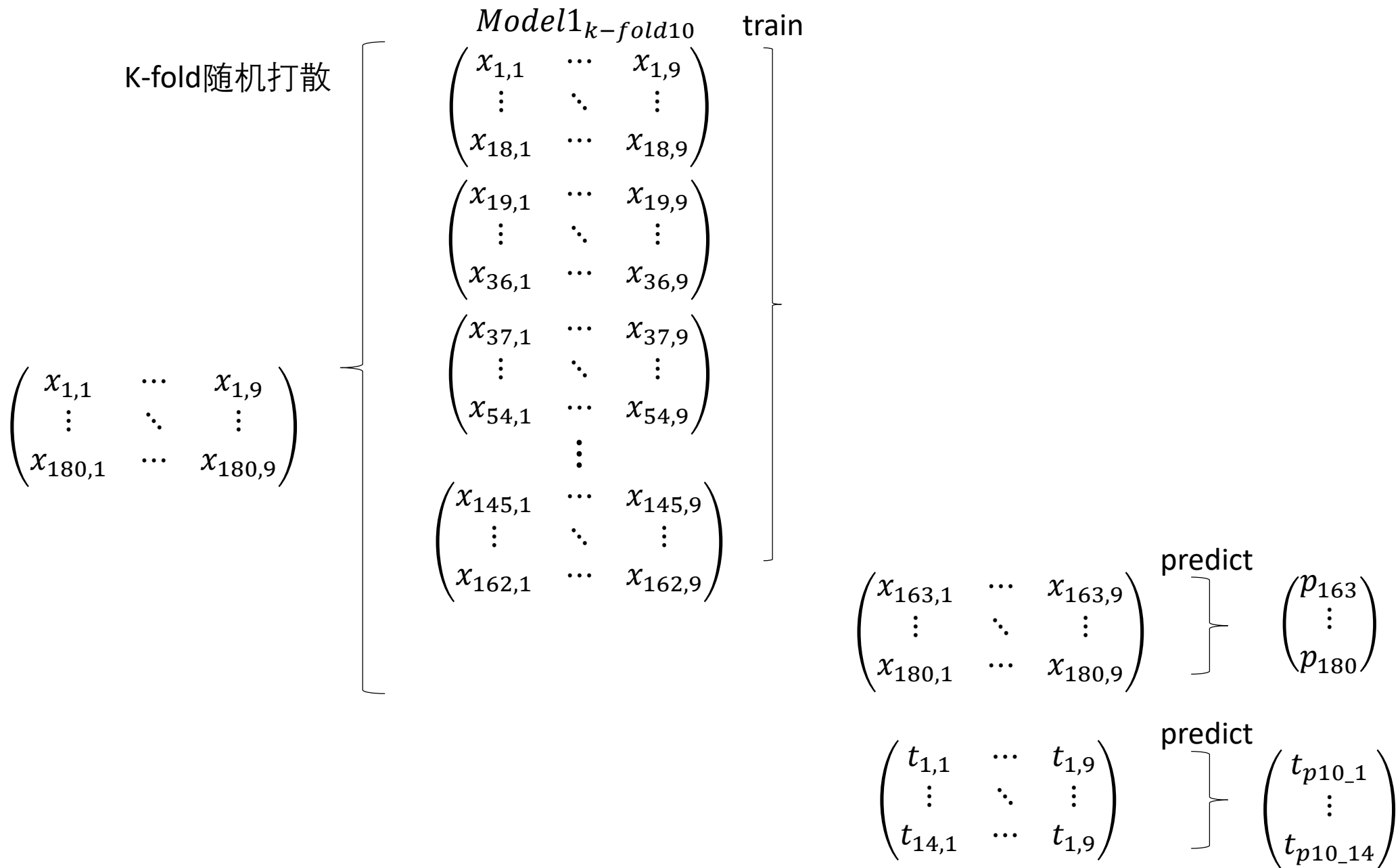
$$\begin{pmatrix} x_{145,1} & \cdots & x_{145,9} \\ \vdots & \ddots & \vdots \\ x_{162,1} & \cdots & x_{162,9} \end{pmatrix}$$

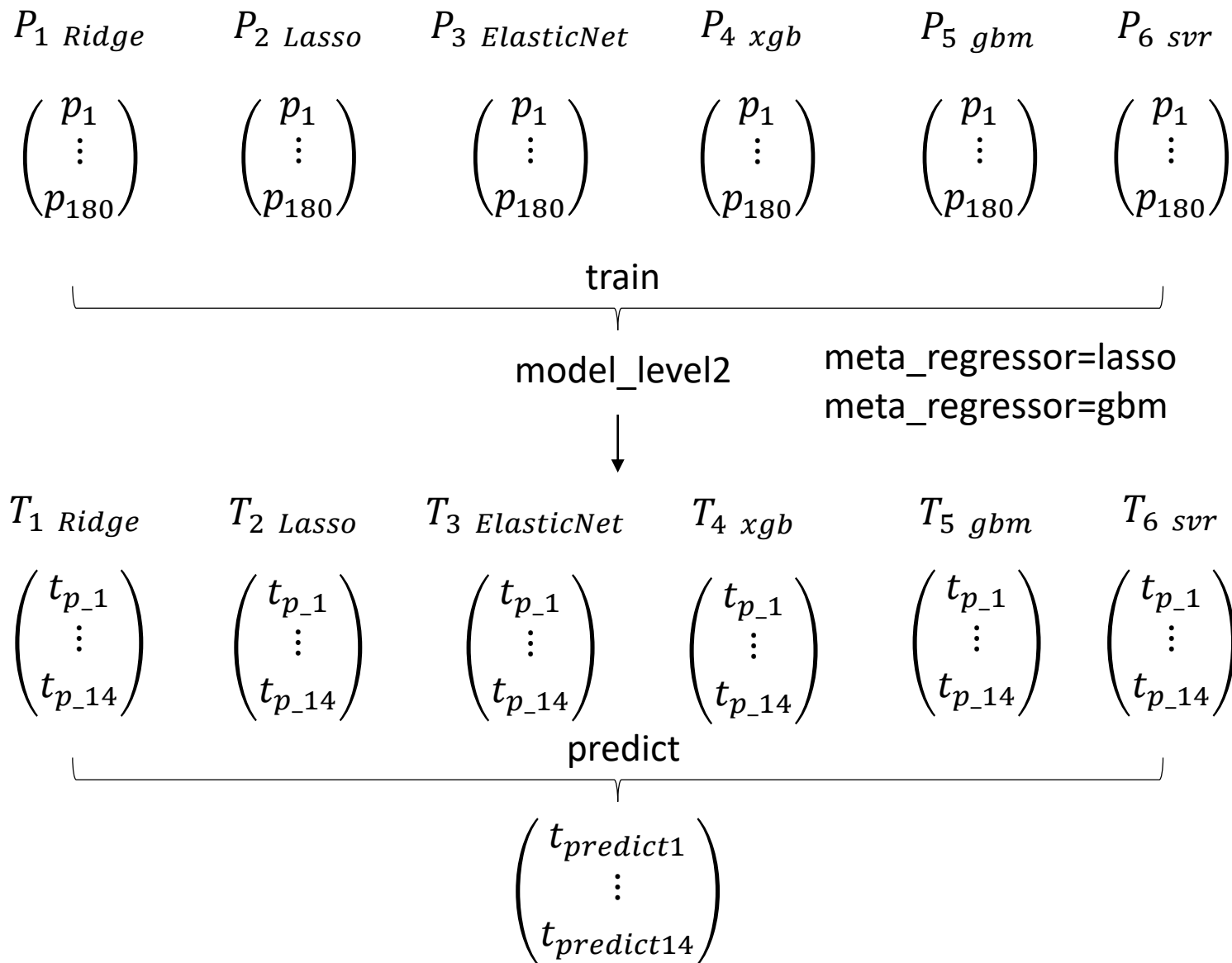
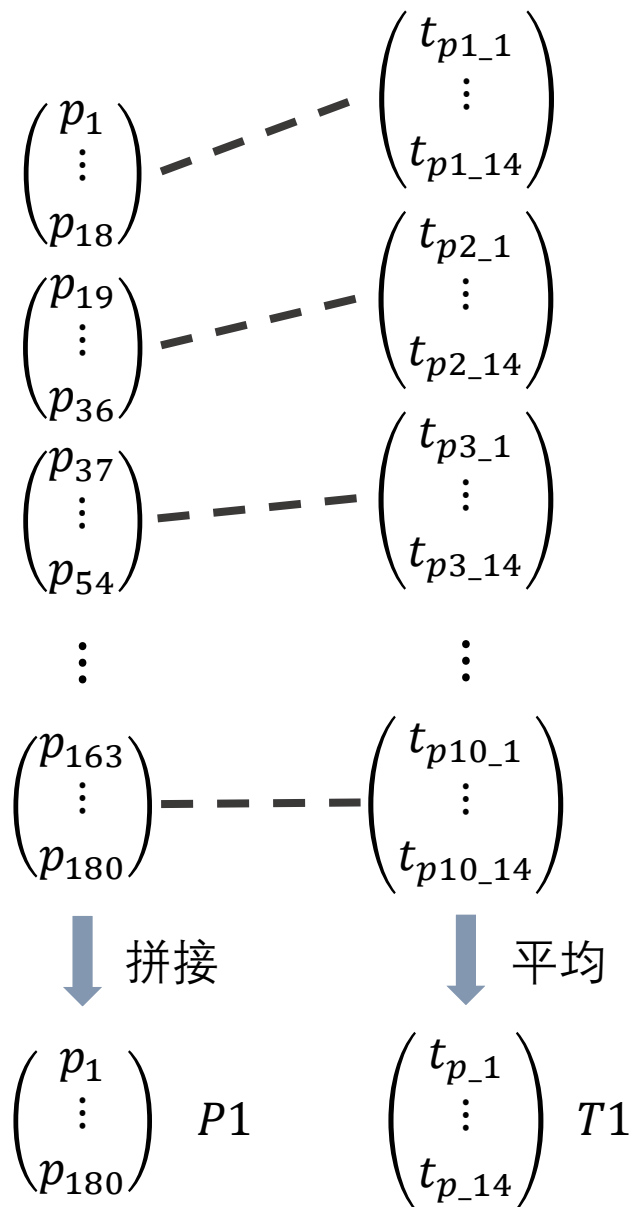
$$\begin{pmatrix} x_{163,1} & \cdots & x_{163,9} \\ \vdots & \ddots & \vdots \\ x_{180,1} & \cdots & x_{180,9} \end{pmatrix}$$





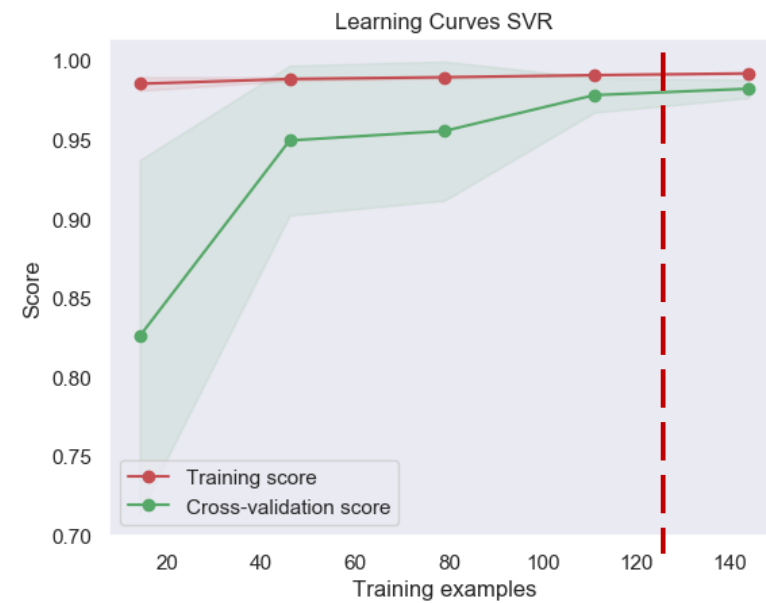
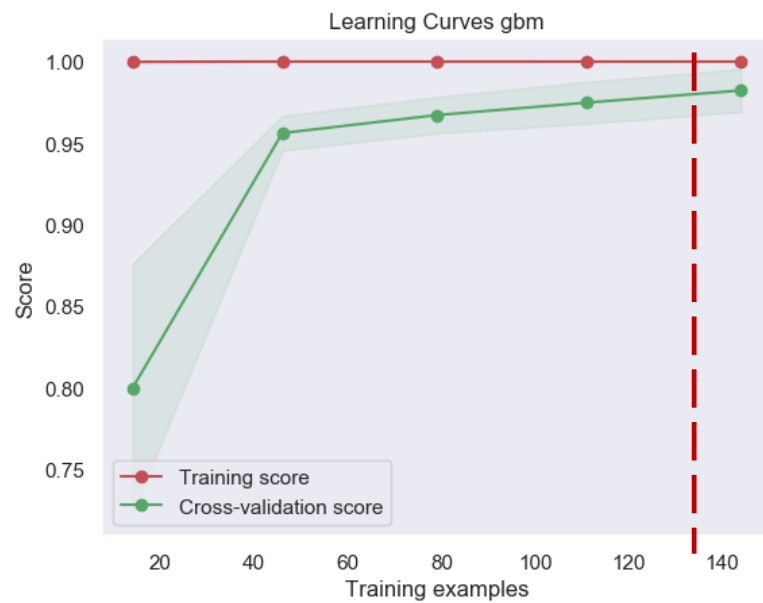
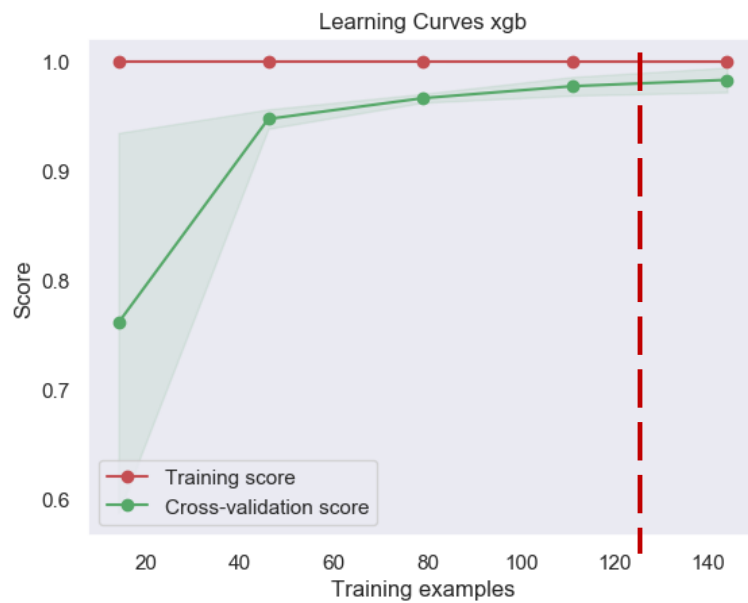
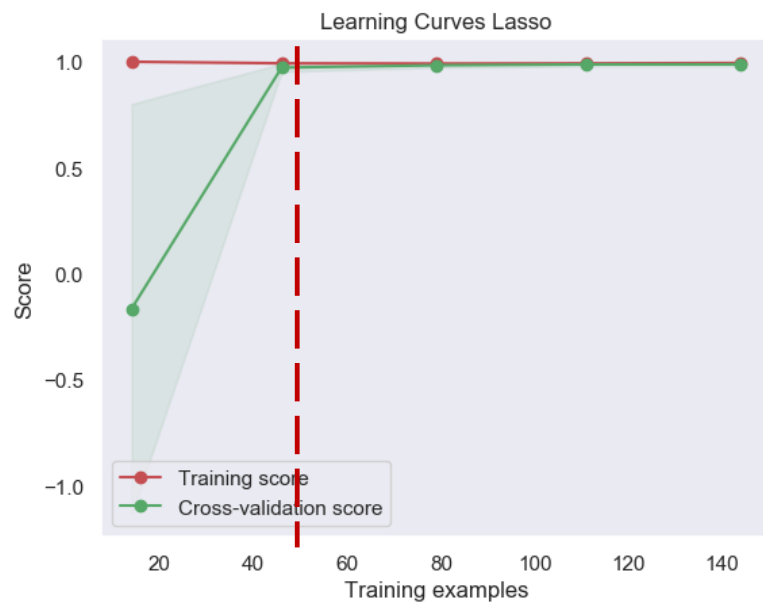
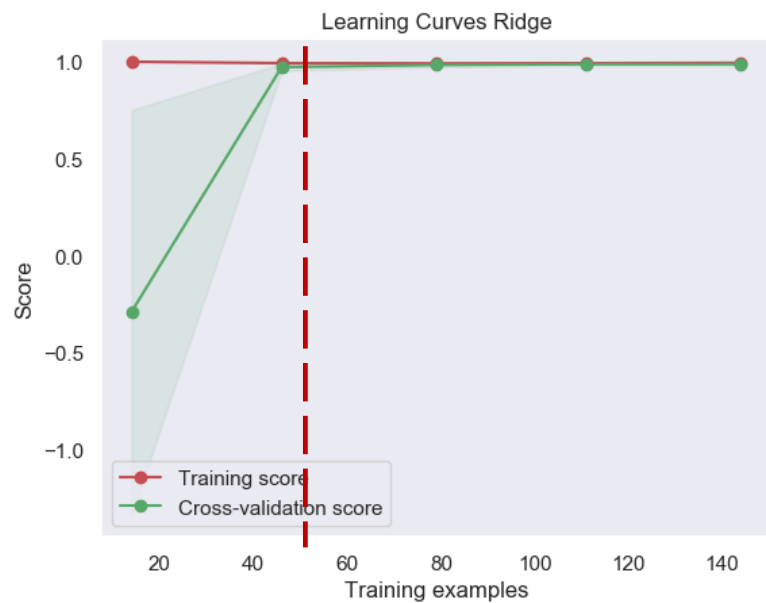








算法结构





- 数据分析与清洗
- 模型设计
- 算法结构
- 可移植性与工程优化



初赛阶段

- DNN 深度神经网络
- 复杂的梯度提升树

初赛阶段：
最终的B榜排名第二
与第一差距0.002

缺点：

- 训练时间长，难以找到全局最优解
- 模型重，灵敏性差
- 数据量不足够支撑模型
- 模型受数据分布影响明显

决赛阶段：化繁为简

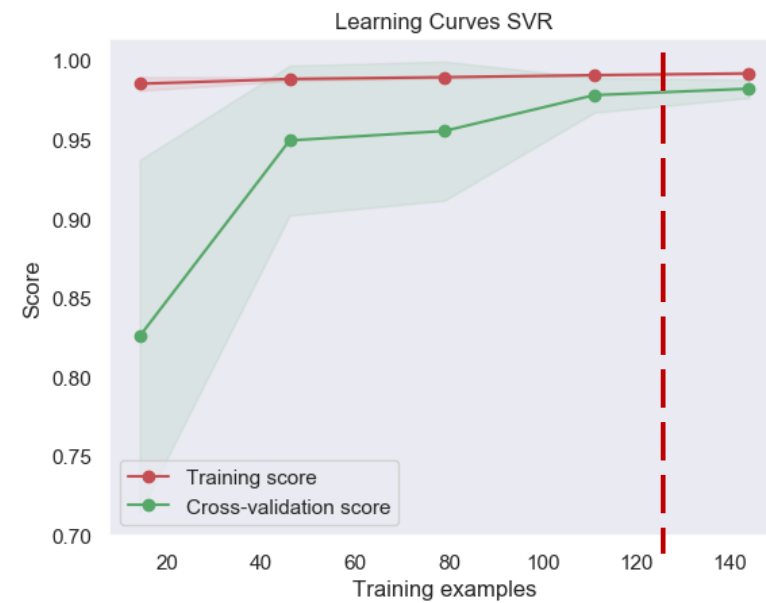
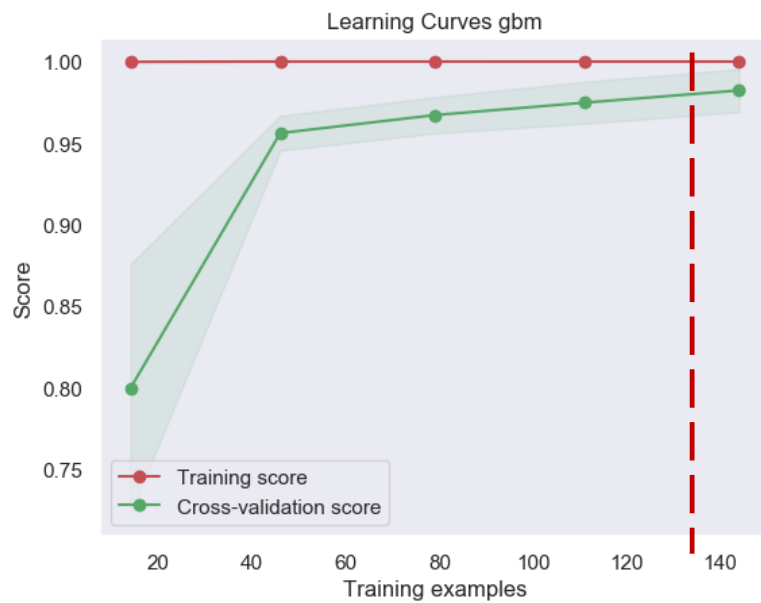
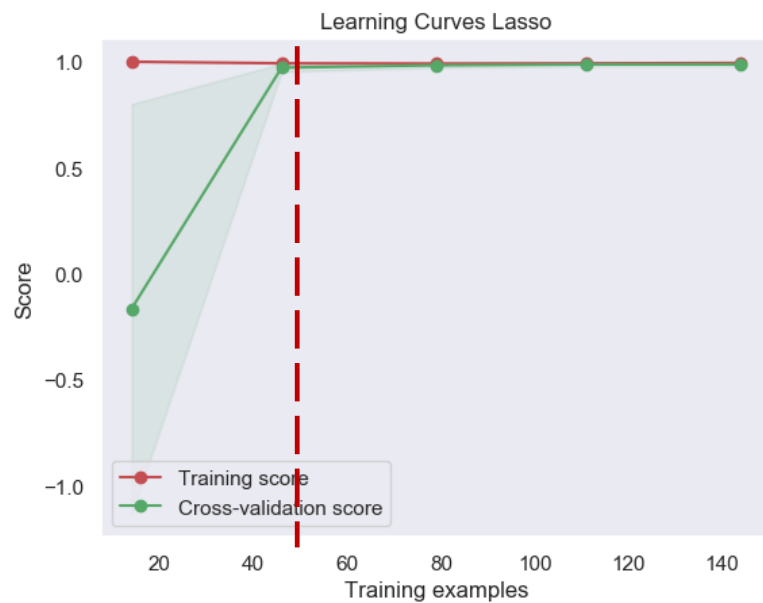
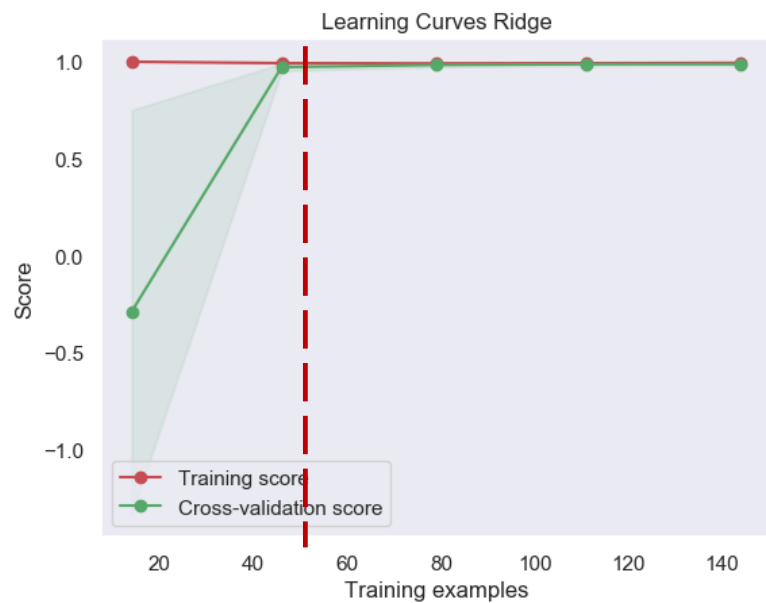
- 更快的训练速度，数十秒
- 解释性更强，更轻的模型，融合算法
- 数据分布差异鲁棒性更强，适用于更广的数据分布
- 结合算法特点更简洁的异常值处理
- 模型对自身参数不敏感，可以保证参数在一定范围内变化，得分保持在一个很高的水平上
- 模型对更换数据集不敏感，以更少的数据量稳定的得到领先的分数

→ $\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$ 小能量模型特征

→ 合理建树



算法结构





可移植性与工程优化

```
def car10_feature(ca, train_ornot):
    car = ca.copy()
    car['sum_charge'] = car['charge_hour'].cumsum()

    if train_ornot == True:
        # 第一个interval_min为上车数据
        car.loc[0, 'interval_min'] = np.nan
        # 不区分快慢充电
        car['charge_mode'] = 3
        # 删除异常数据
        car.drop(car.loc[car['dsoc'] <= 1].index.tolist(), inplace=True)

    if train_ornot == False:
        car['charge_mode'] = 3
    return car
```

```
def car15_feature(ca, train_ornot):
    car = ca.copy()
    car['sum_charge'] = car['charge_hour'].cumsum()

    if train_ornot == True:
        # 第一个interval_min为上车数据
        car.loc[0, 'interval_min'] = np.nan
        # 不区分快慢充电
        car['charge_mode'] = 3
        # 删除异常数据
        car.drop(car.loc[car['dsoc'] <= 1].index.tolist(), inplace=True)
        car.drop(car.loc[car['charge_hour'] > 5].index.tolist(), inplace=True)

    if train_ornot == False:
        car['charge_mode'] = 3
    return car
```

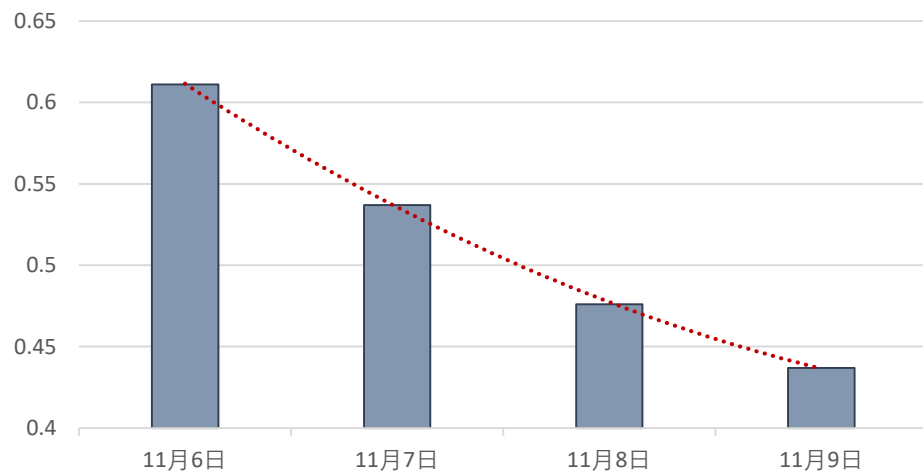
```
def car16_feature(ca, train_ornot):
    car = ca.copy()
    car['sum_charge'] = car['charge_hour'].cumsum()

    if train_ornot == True:
        # 第一个interval_min为上车数据
        car.loc[0, 'interval_min'] = np.nan
        # 不区分快慢充电
        car['charge_mode'] = 3
        # 删除异常数据
        car.drop(car.loc[car['dsoc'] <= 1].index.tolist(), inplace=True)
        car.drop(car.loc[car['charge_hour'] > 5].index.tolist(), inplace=True)
        car.drop(car.loc[car['dsoc/hour'] > 100].index.tolist(), inplace=True)

    if train_ornot == False:
        car['charge_mode'] = 3
    return car
```



决赛评价指标



新能源汽车数据类型决定了，不能靠堆砌数据，数据预处理质量与模型算法方法才是核心

关于工程实践的思考

- 边缘计算场景：基于线性模型与弱学习器，可只需较小的算力和较小的成本在车辆上应用，并能在数秒内完成预测。
- 上云：车联网，大数据统一汇总云平台，对数据分布类似的车型一起处理。

新能源汽车大数据创新创业大赛

电动汽车动力电池充电能量预测

感谢评委与同学们的聆听

谢谢



刘晓曼 王昕杰 谭海宇

2018/11/22