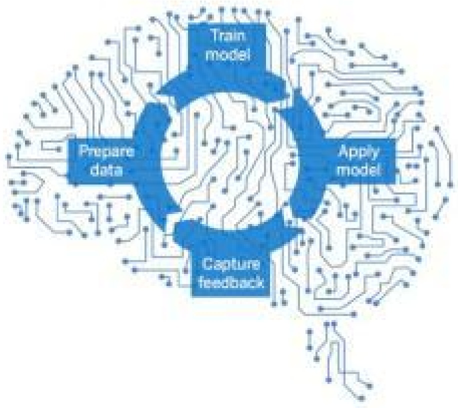
New Energy Vehicle Big Data Innovation and Entrepreneurship Competition

EV battery charging energy prediction



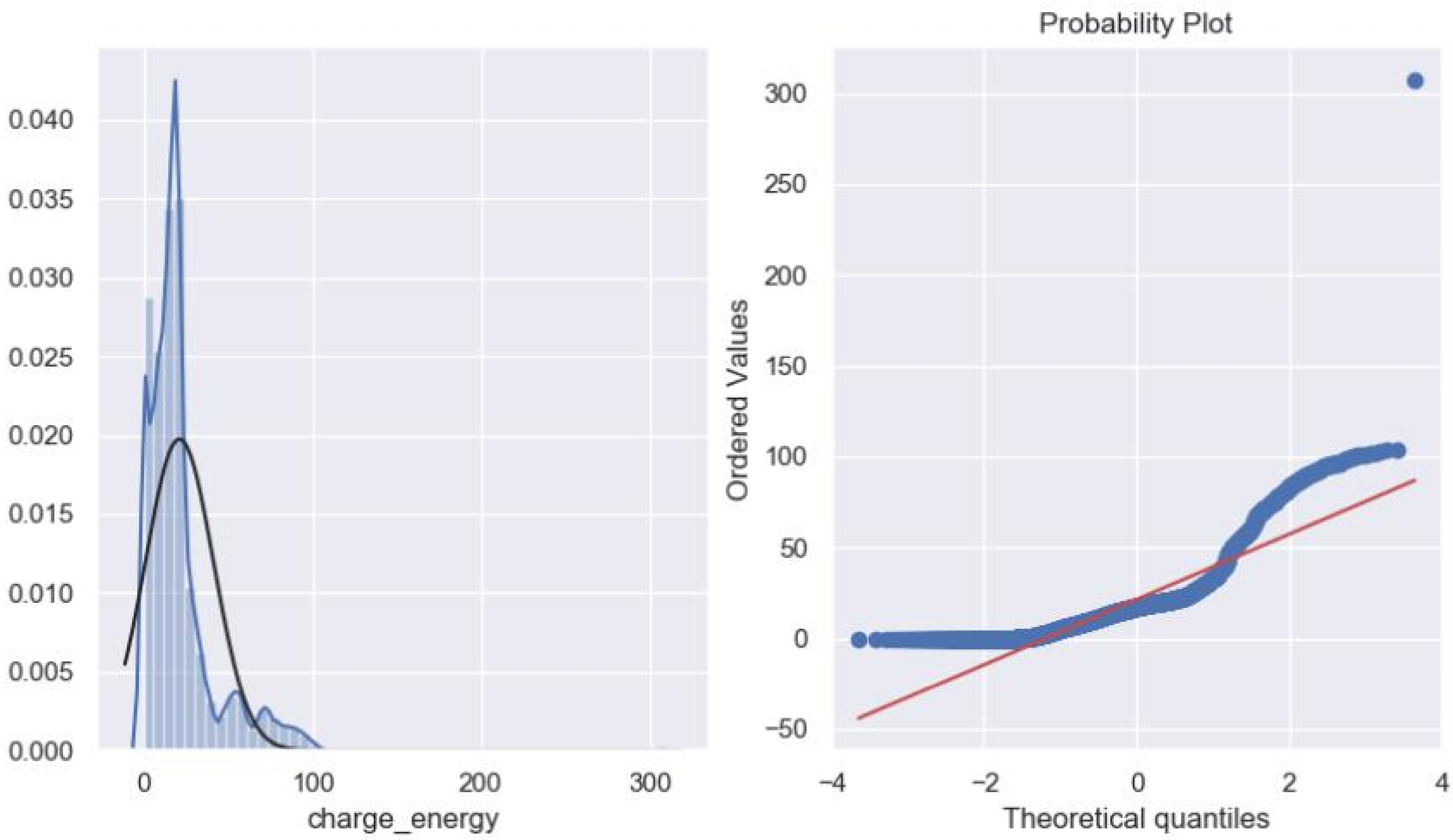
Tongji University

Liu Xiaoman,Wang Xinjie,Tan Haiyu

2018/11/22

- Data analysis and cleaning
- Model design
- Algorithm structure
- Portability & Engineering Optimization

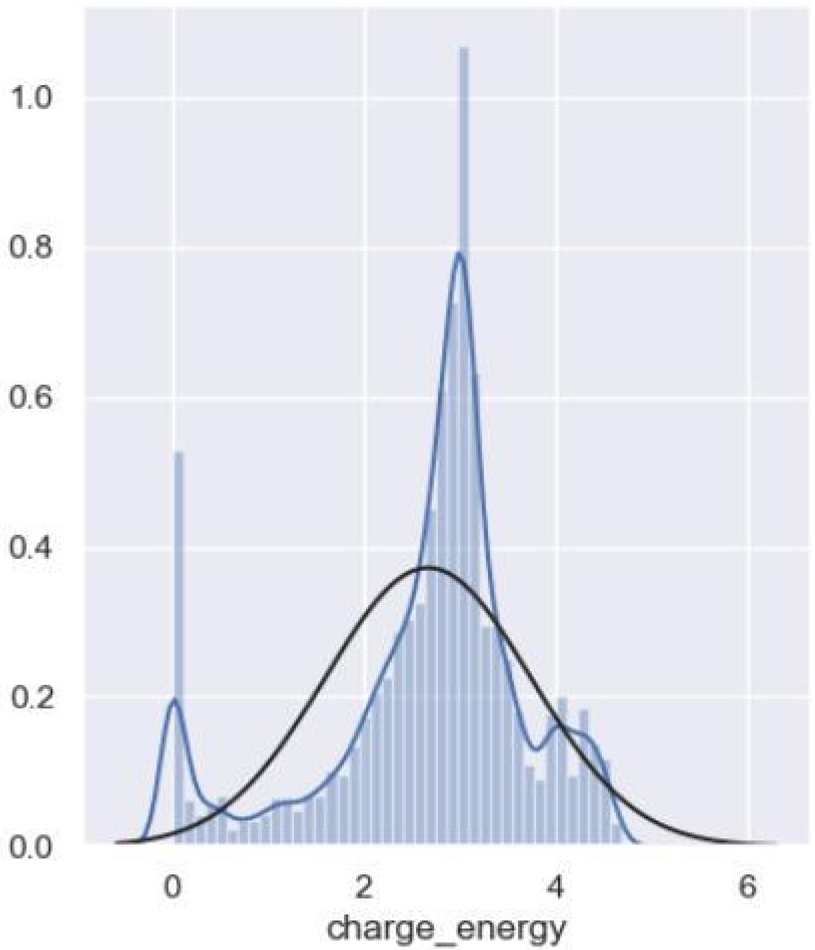
data analysis

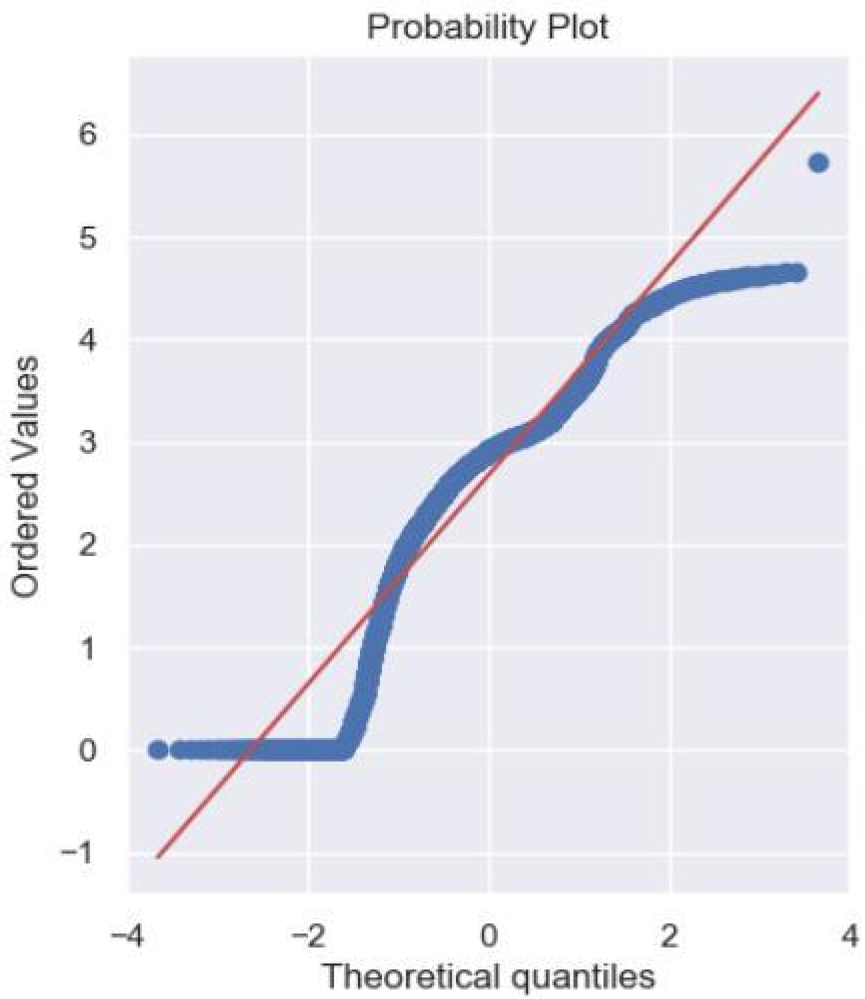


Raw distribution of training set

Lognormally distributed

- The factors that affect the charging energy are not independent of each other, and the influence of various factors on the charging energy is multiplicative
- Low amount of data

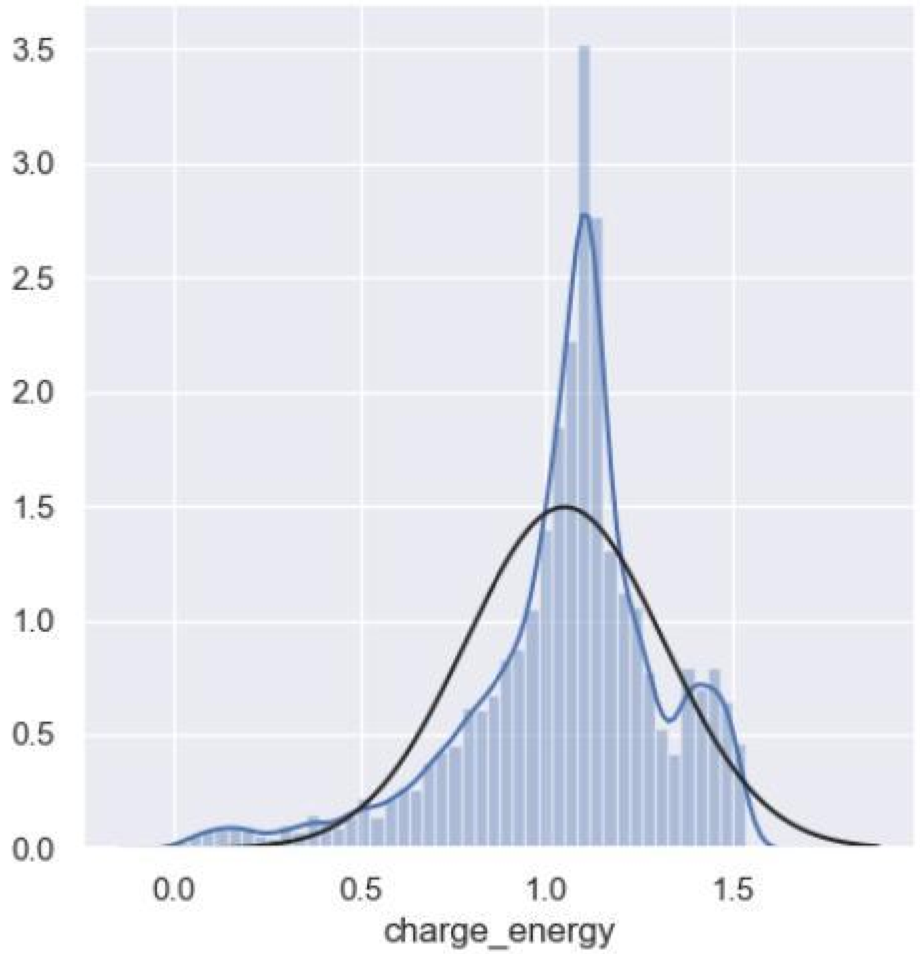


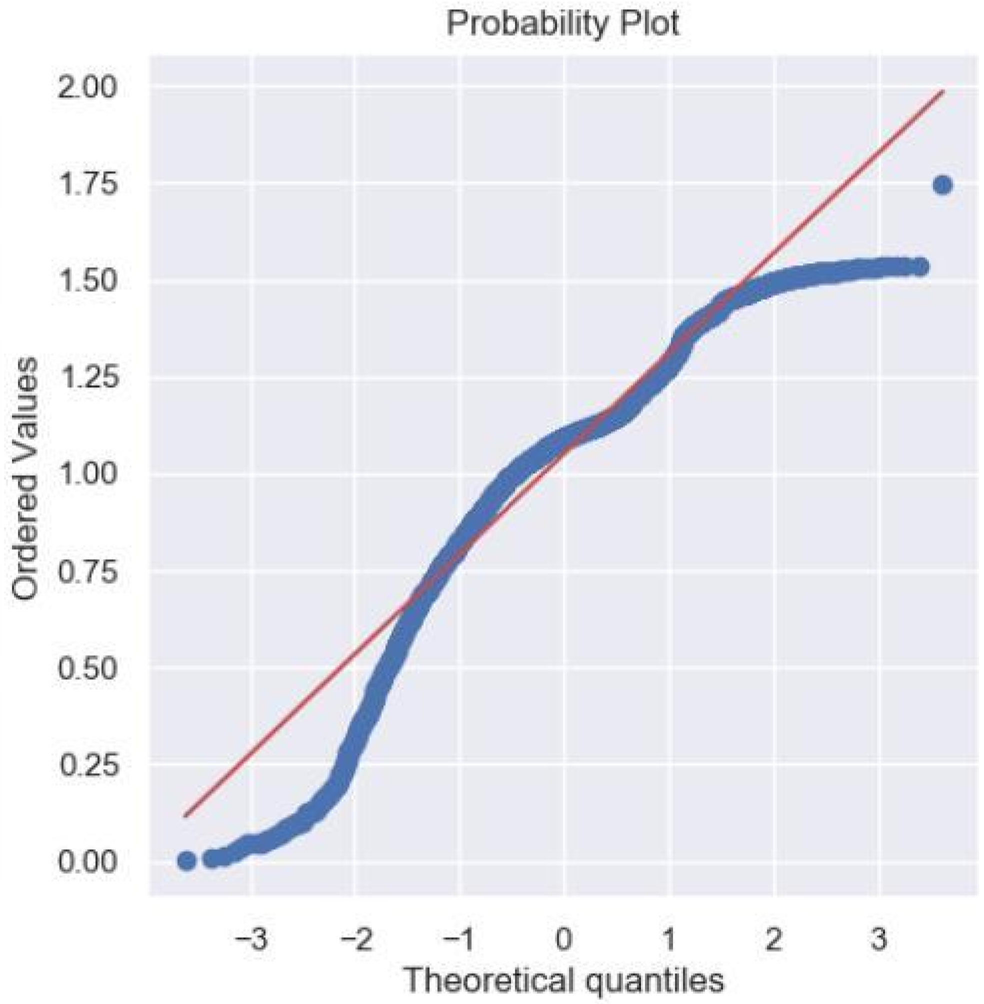


Lognormally distributed

- Many small energy distribution points

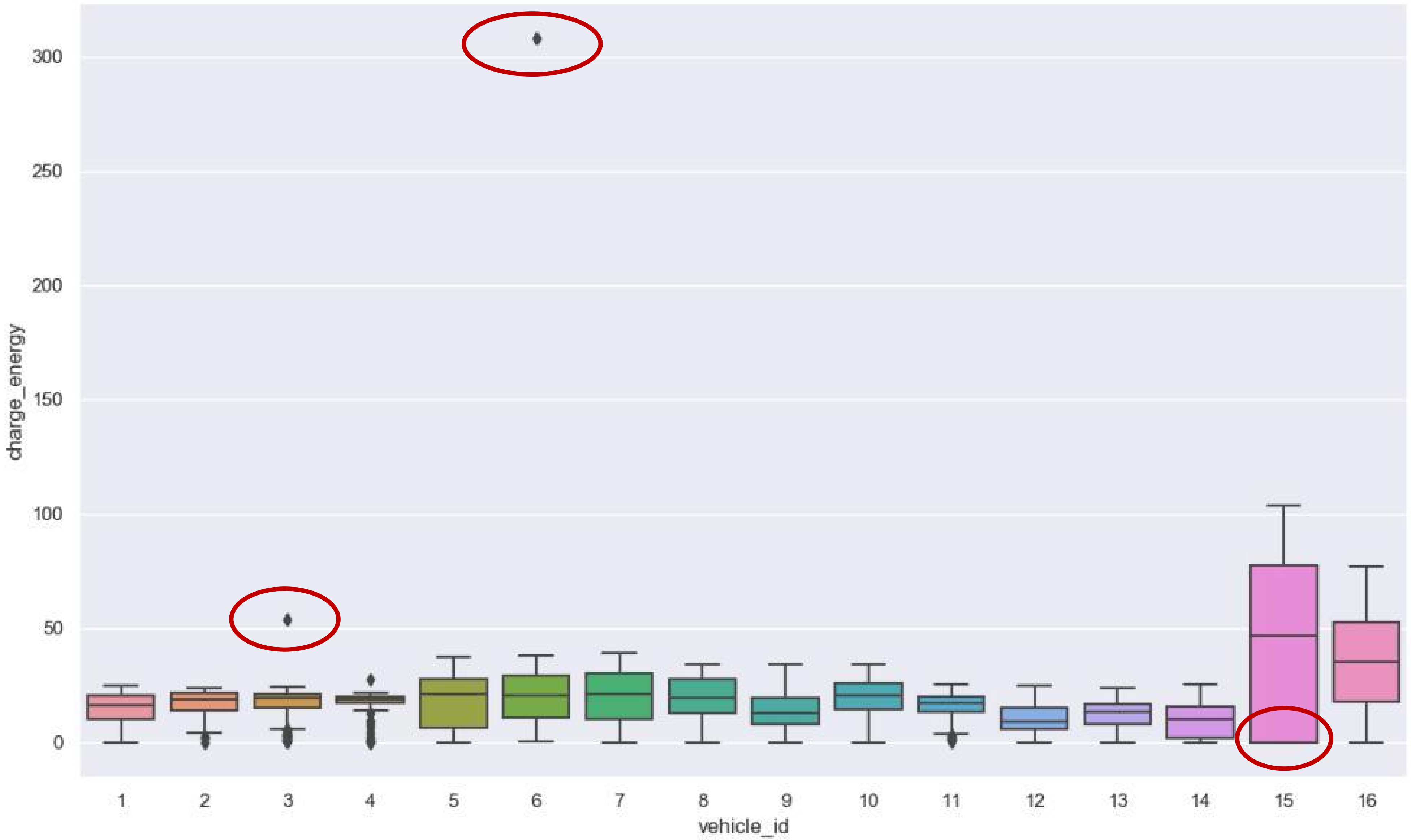
The distribution of the training set energy after In(x+1) transformation





Distribution of the positive part of the training set energy after In transformation

Data cleaning



Energy distribution Box plot

Overcharge protection or charging phase conversion

缺失值检测

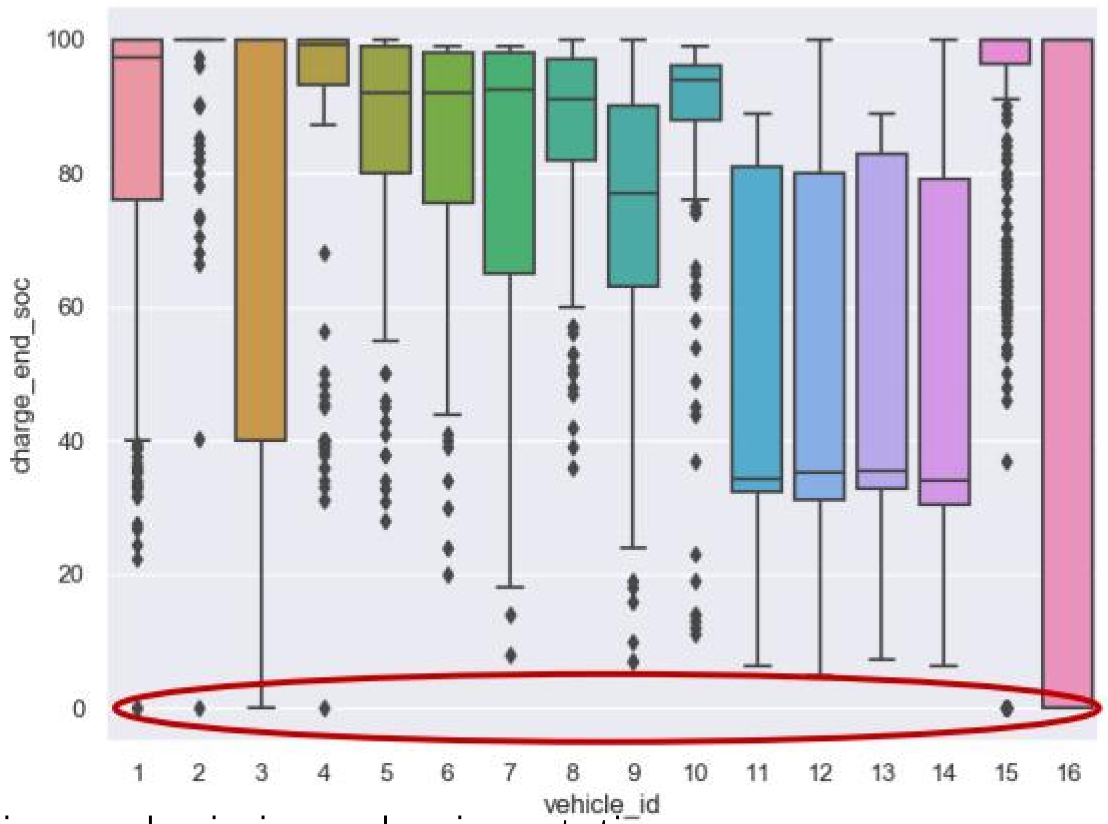
```
total = df_train.isnull().sum().sort_values(ascending=False)
percent = (df_train.isnull().sum()/df_train.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
print(missing_data)
```

```
                  Total   Percent
charge_end_soc      154  0.029799
charge_start_soc     55  0.010642
```

Missing values for all vehicle models.

Data cleaning is very important, as big data often consists of a large amount of problematic data.

Data cleaning



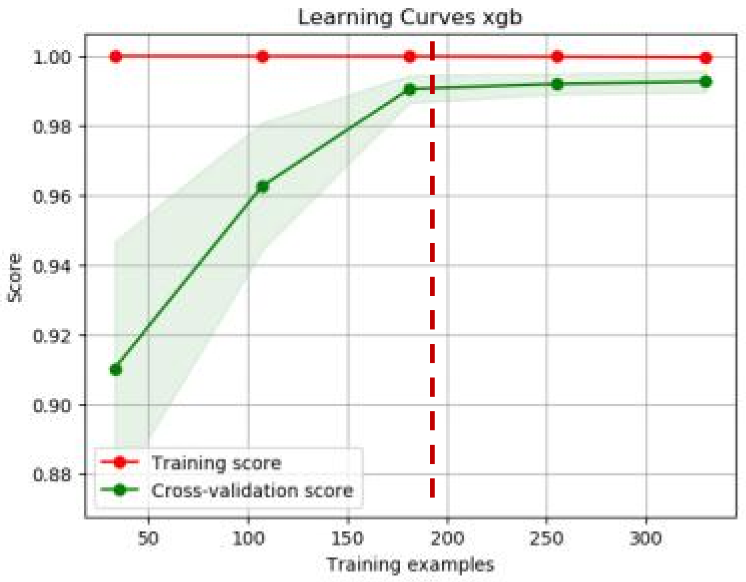
Anomaly correction and missing value imputation.

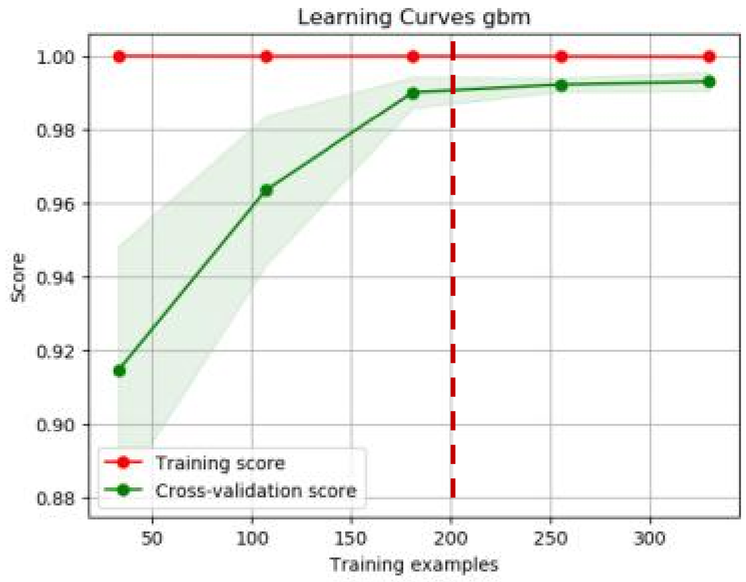
Car1-3 car_phase0.loc[(car_phase0.charge_end_U > 389) & (car_phase0.charge_end_I > -29), ['charge_end_soc']] = 100

Car4 , 15-16 Using random forests and gradient boosting trees for anomaly correction and missing value imputation.

L—— Increasing data preprocessing workload had a moderate effect; ultimately, tree algorithms were chosen for enhancement.

Data cleaning

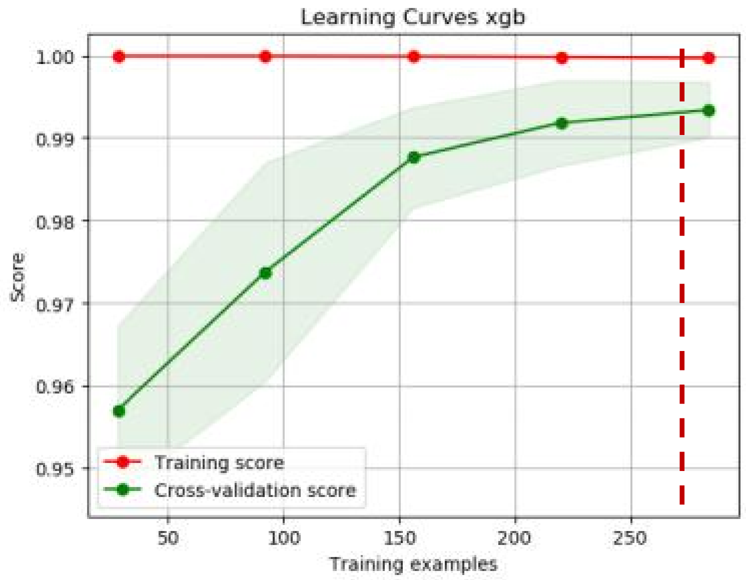


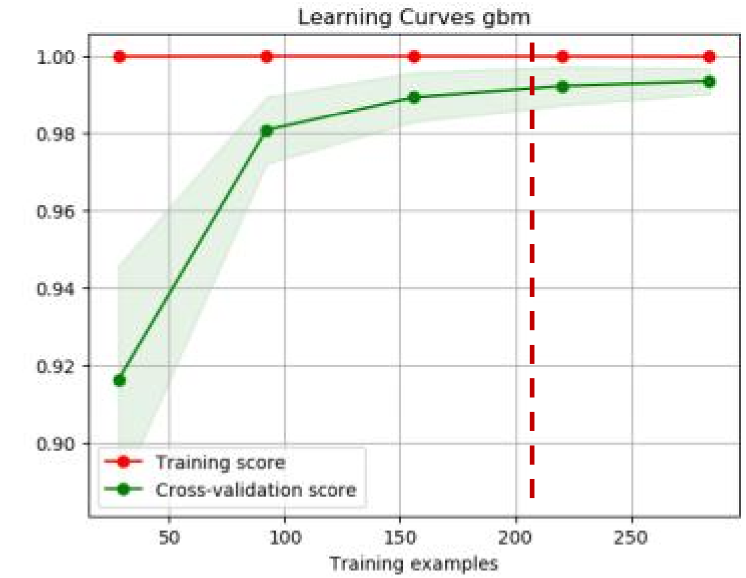


Car15-16

Increasing data preprocessing workload had a moderate effect; ultimately, tree algorithms were chosen for enhancement.

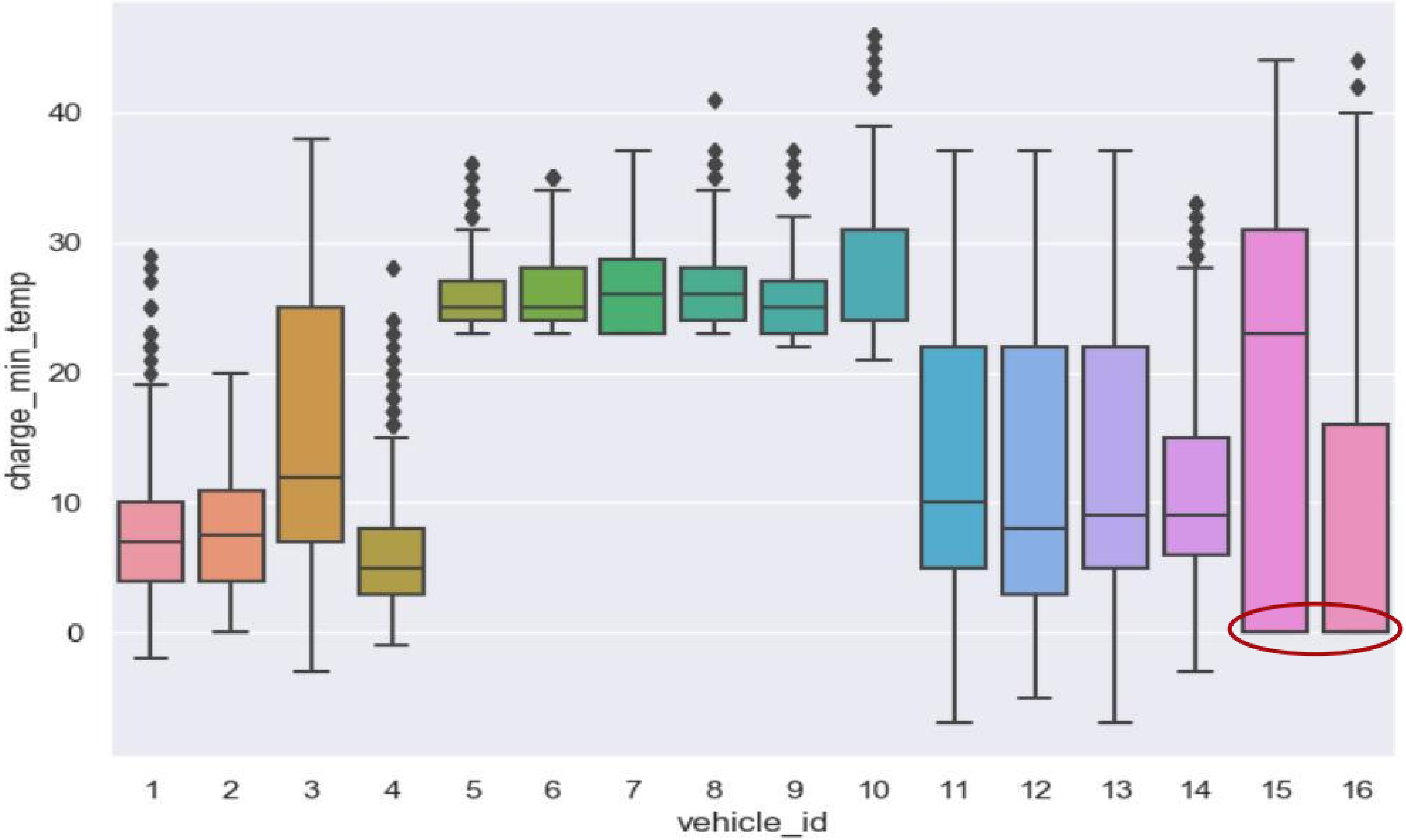
The effective data volume remains sufficient.





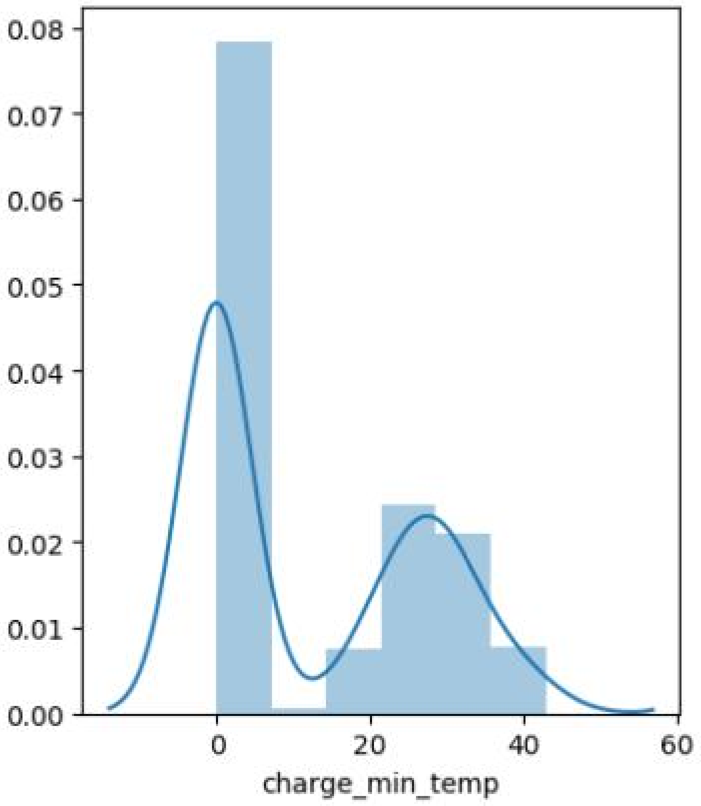
Tree algorithms:
- Good at handling missing values
- Robust to outliers

Data preprocessing should align with algorithm characteristics, avoiding overengineering and the introduction of artificial noise.

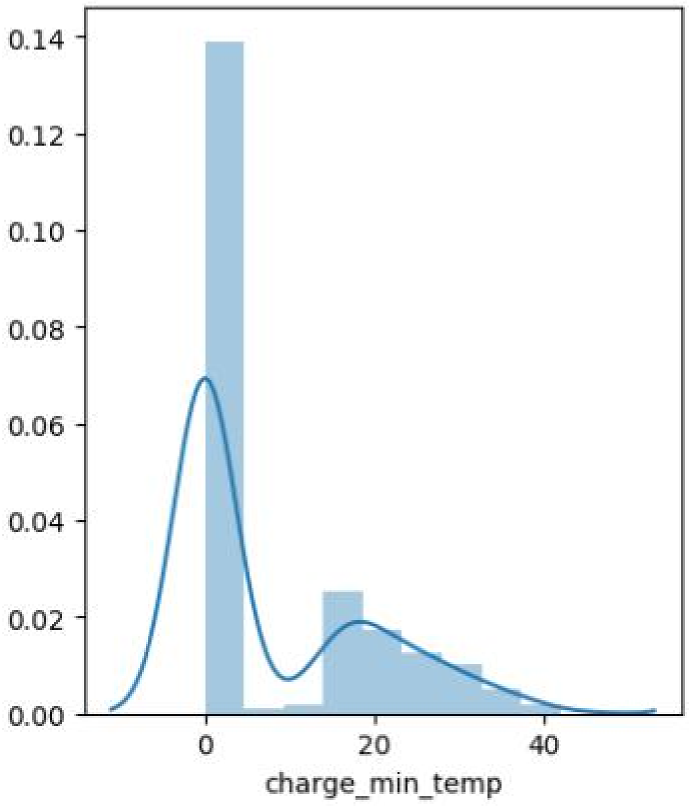


Box plot of the minimum temperature distribution.

Data cleaning



car15



car16

Anomaly correction

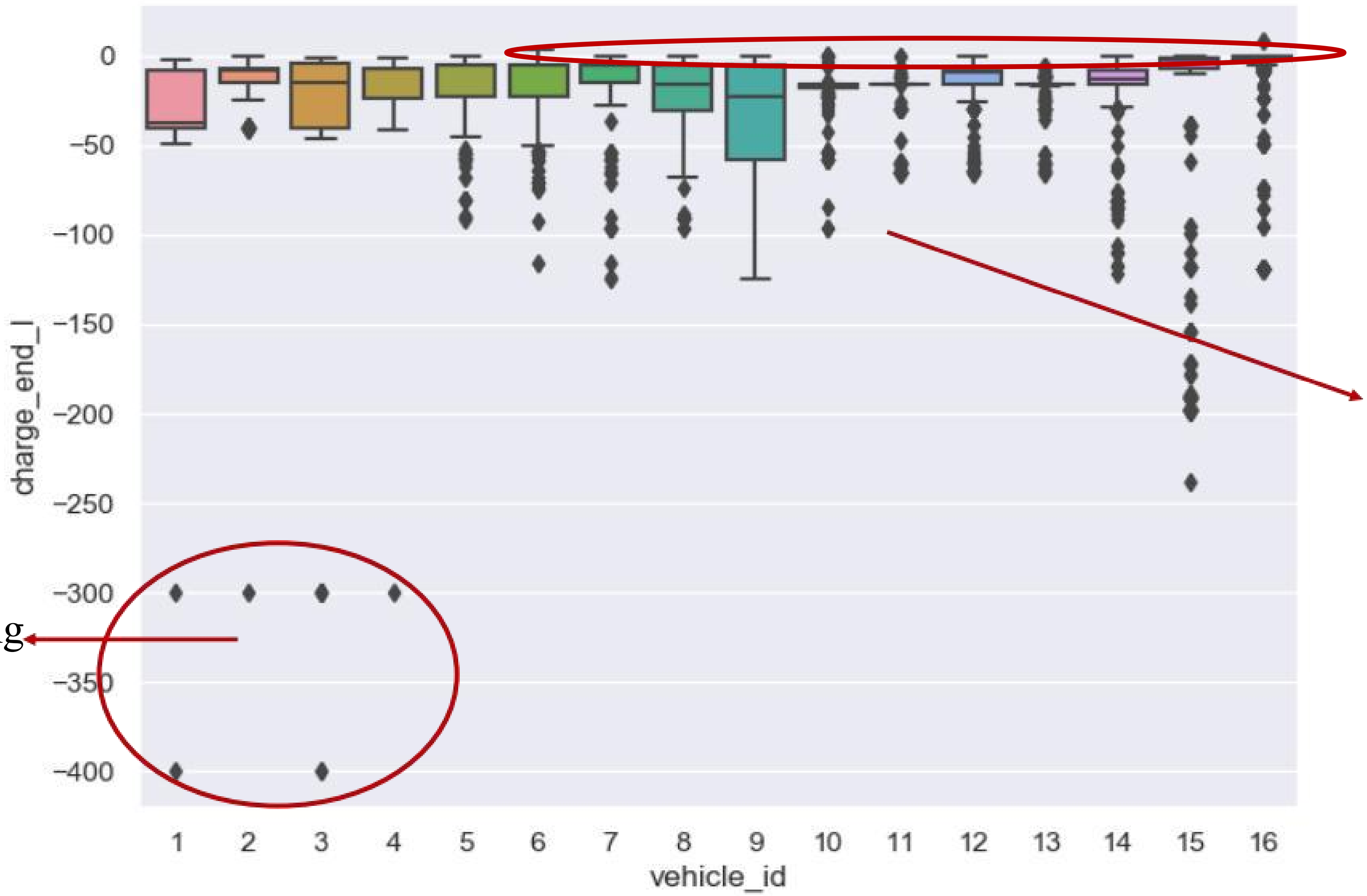
- For car15, `charge_min_temp` is a continuous value and follows a normal distribution; mean correction is used to maintain the expected value.

- For car16, `charge_min_temp` is a continuous value with a long-tail distribution; median correction is used to avoid the impact of outliers.

- Anomaly correction is performed using random forests and gradient boosting trees.

- The monthly average is taken.

Data cleaning



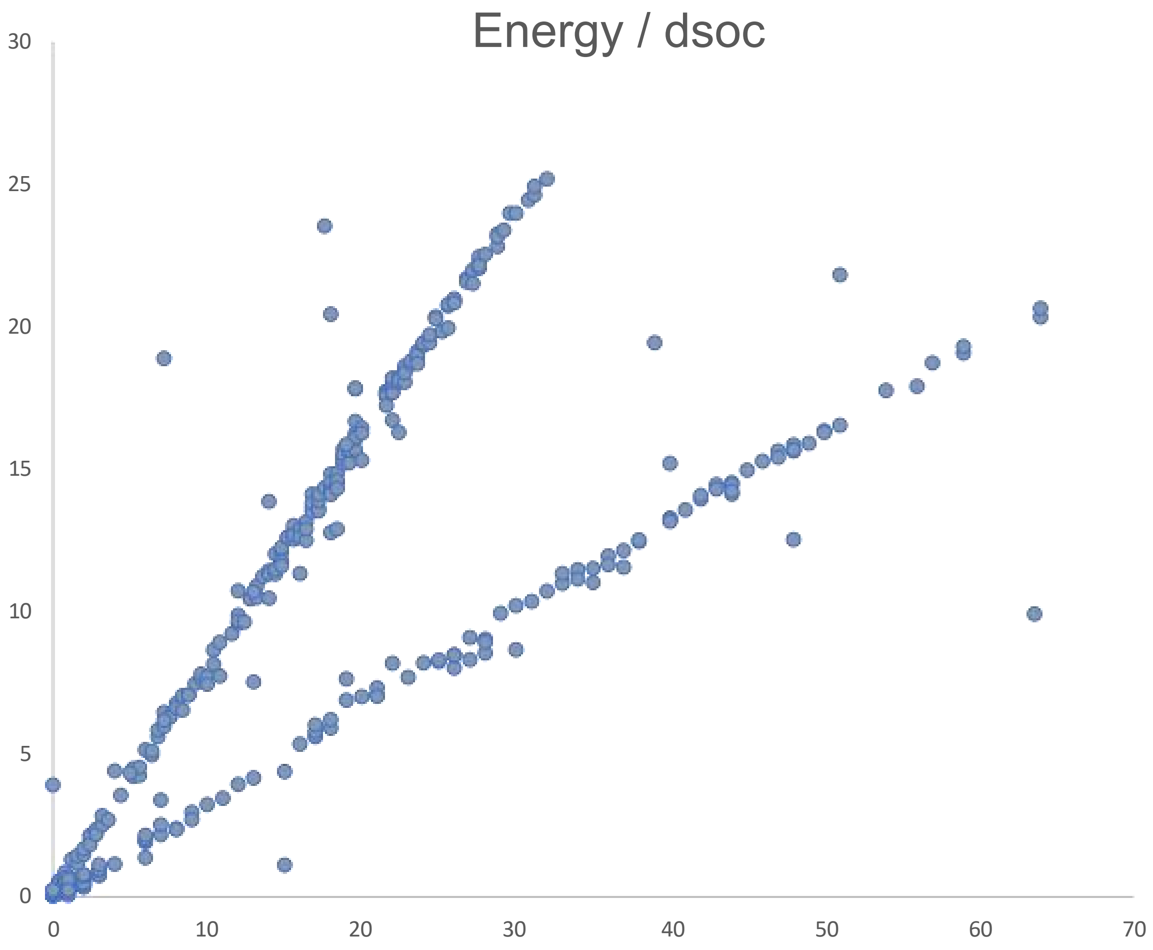
Some charging end currents are recorded as 0, especially for Car15 and Car16.

Outliers ≠ Anomalies, but they can significantly affect data distribution and impact feature normalization, while containing exceptionally rich information.

Accompanying anomalies:
End_soc = 0,
End_U = 0

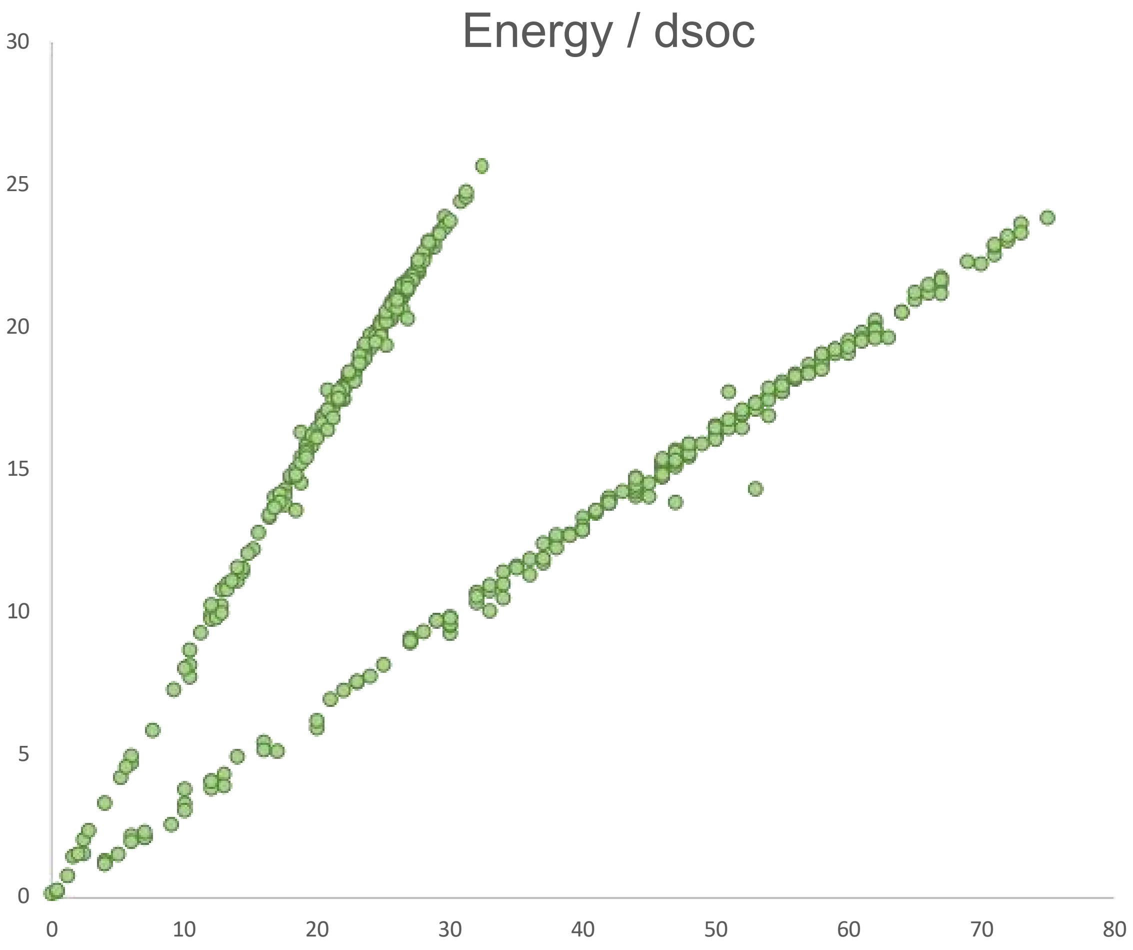
Box plot of the end current distribution

Data cleaning



car14

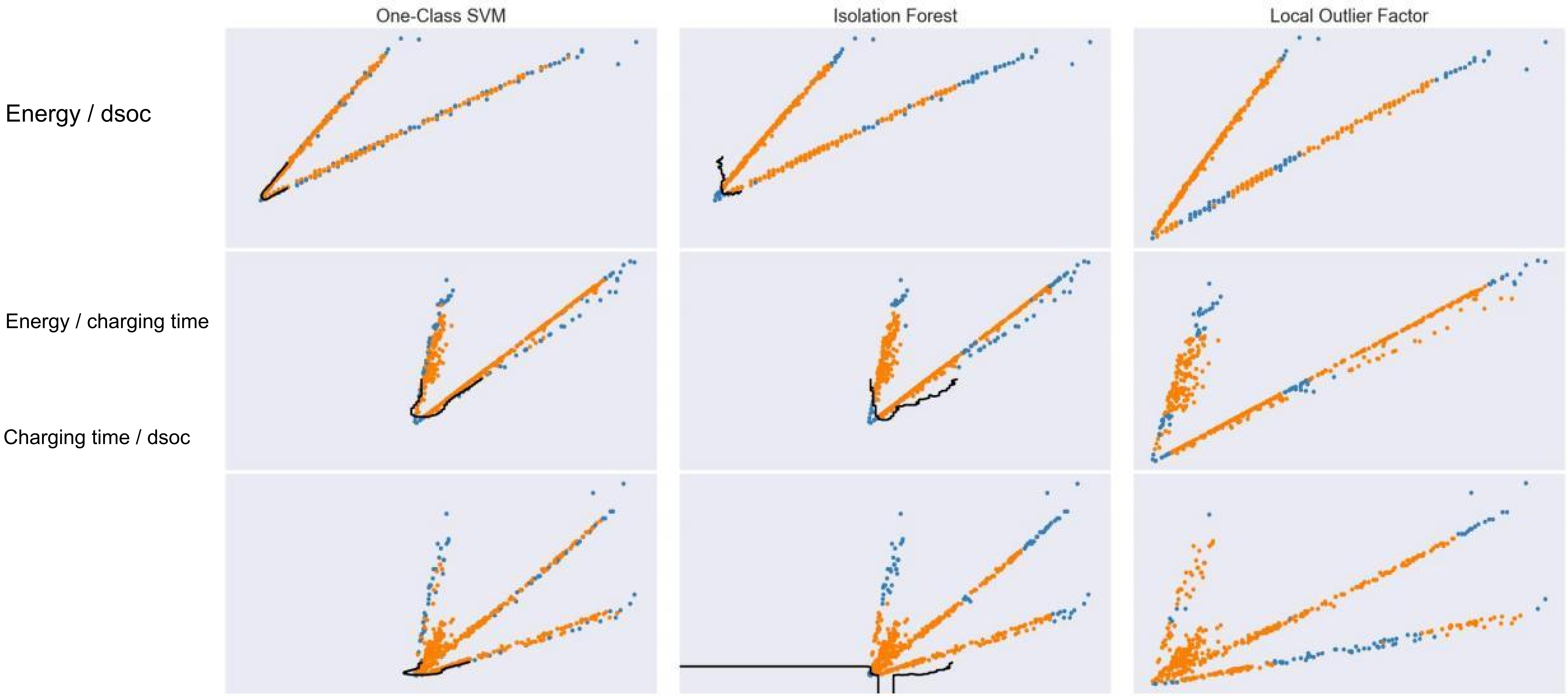
Combine the new feature dsoc / charge_hour to eliminate anomalies



car11

dsoc: Charging end soc - Charging start soc

Data cleaning



In situations where the charging time is short and dsoc is small, the data distribution varies greatly, especially for car12.

- Data analysis and cleaning
- Model design
- Algorithm structure
- Portability & Engineering Optimization

Model design

basic feature group

Original features

representative feature group

Representative features

temporal feature group

Temporal features

Basic feature group

- Original features 12 dimensions

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

Model design

Representative feature group

- More expressive features: 11 dimensions
    - Difference features: `charge_hour`, `dsoc`, `dU`, `dtemp`
    - Ratio features: `dU/dsoc`, `dsoc/hour`, `dmileage/soc`
    - Memory features: `ddsoc`, `dmileage`
    - Categorical features: `phase`, `charge_mode`

Model design

Temporal feature group.

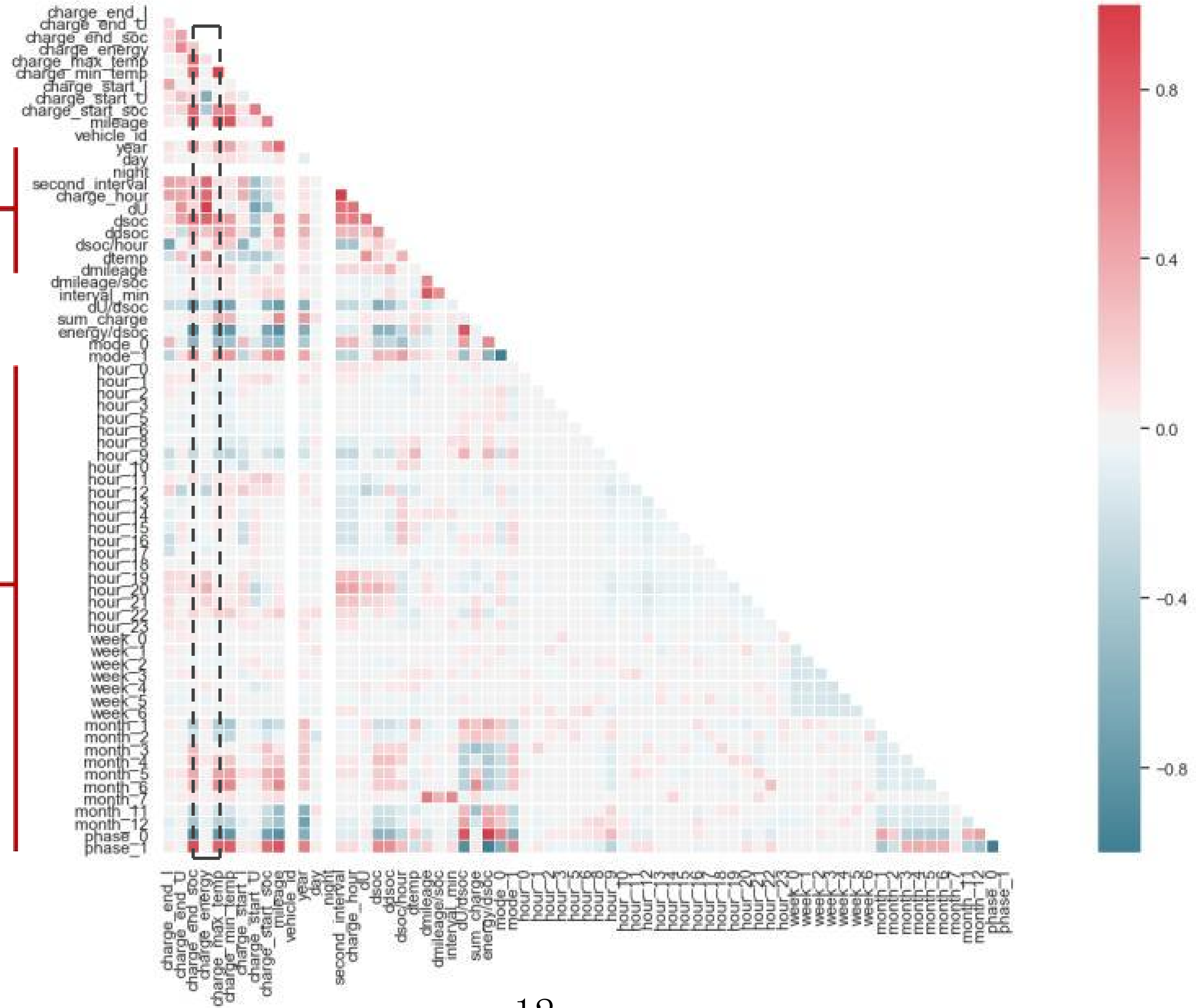
- Temporal features of energy:
- Temporal features: `year`, `month`, `day`
- User habit features: `week`, `hour`, `night`
- Battery temporal features: `interval_min`, `sum_charg`

- one-hot encoding

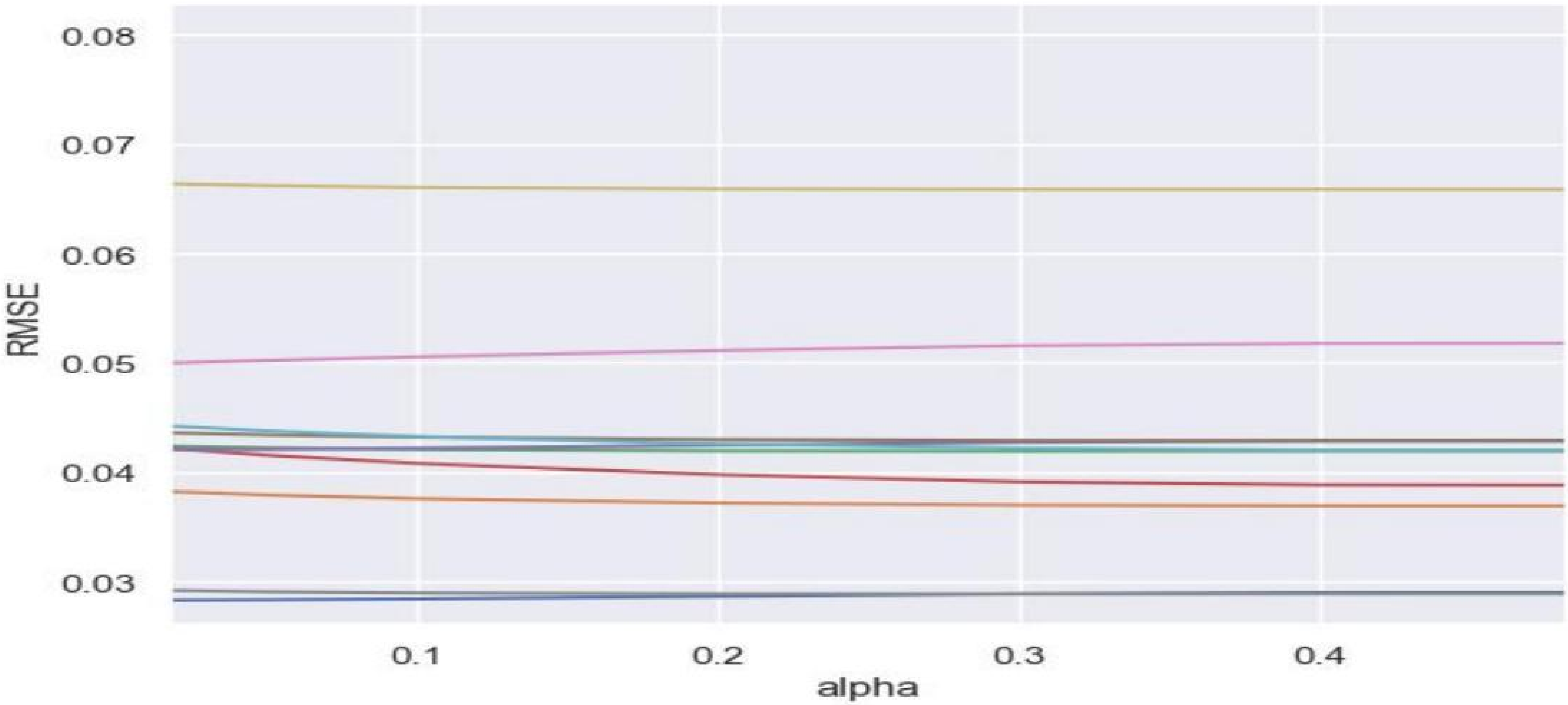
Model design

Representative features have a stronger correlation with energy

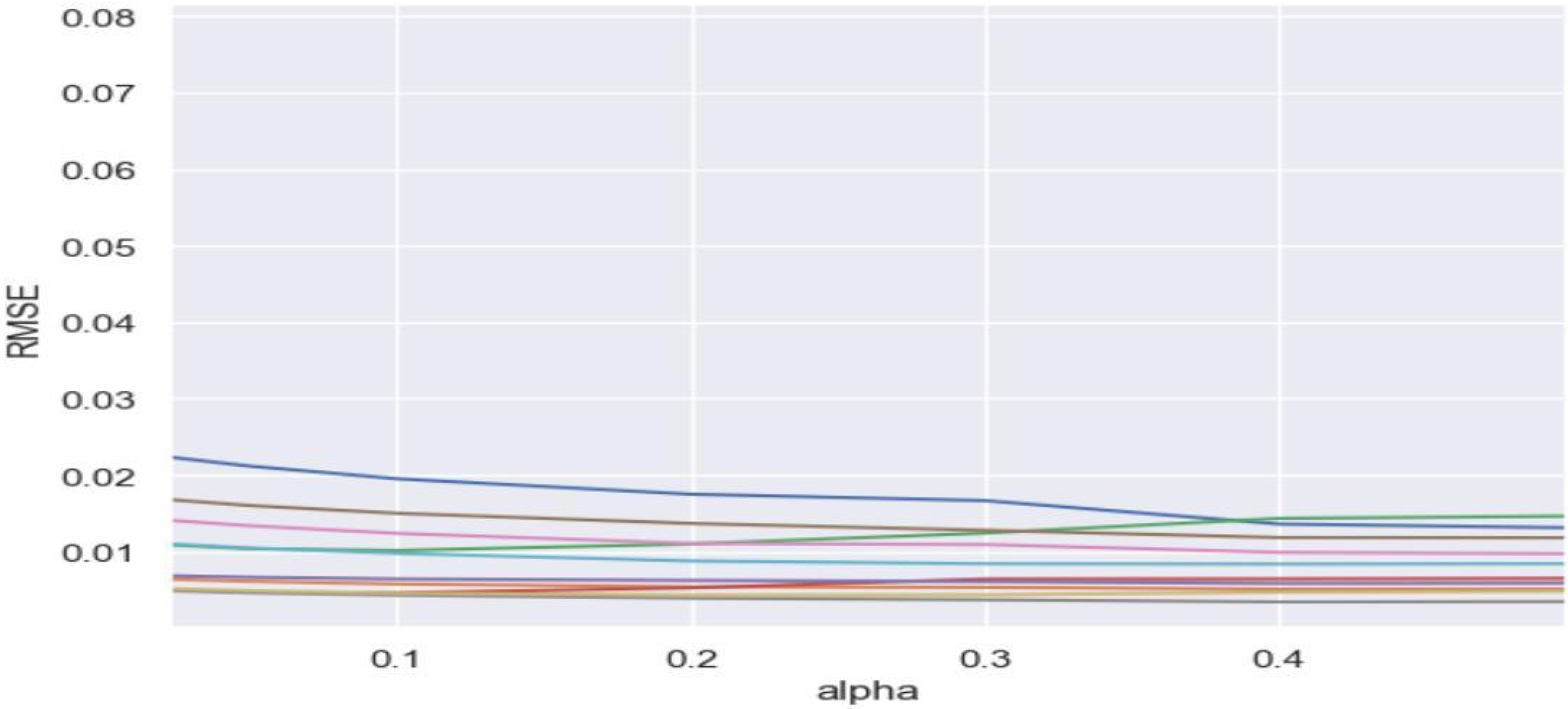
Temporal features are one-hot encoded to form a sparse matrix



car12



Car12 cross-validation results with original features



Car12 cross-validation results with multi-dimensional features.

Model design

Generalized features

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

Regression methods require a large amount of data in the training set. Even a dataset with millions of entries cannot support training with hundreds of dimensions. The limitation on the number of features prevents full utilization of the information, thereby affecting the accuracy of the regression model.

Small energy features

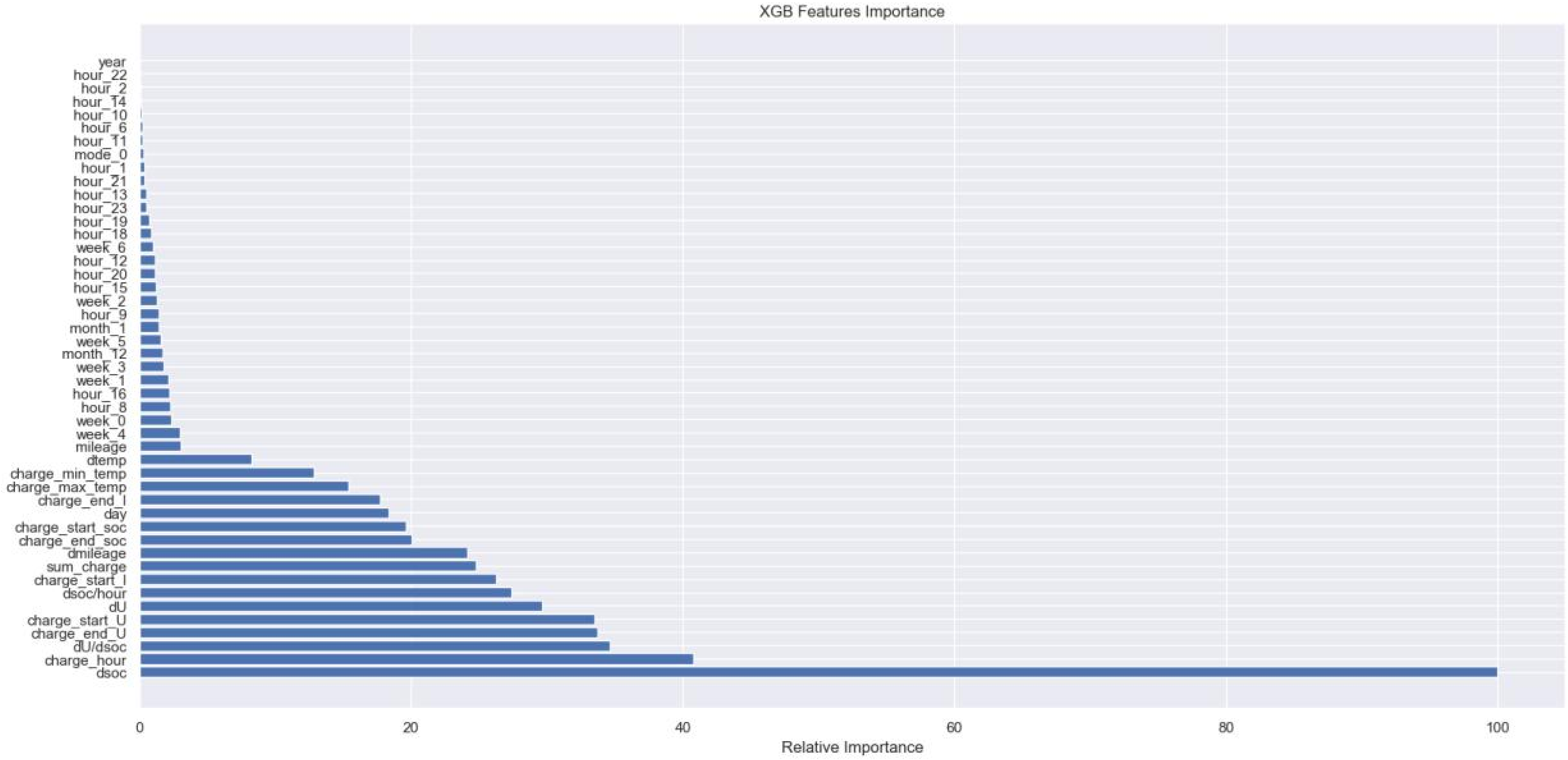
```
select_list = [
    'charge_hour',
    'dsoc',
    'dtemp',
    'dU',
    'charge_start_I',
    'charge_end_I' ]
```

On the second to last day of the finals, a small energy model was introduced, and after coefficient fusion, the performance improved by 9%, achieving first place on that day

Model design

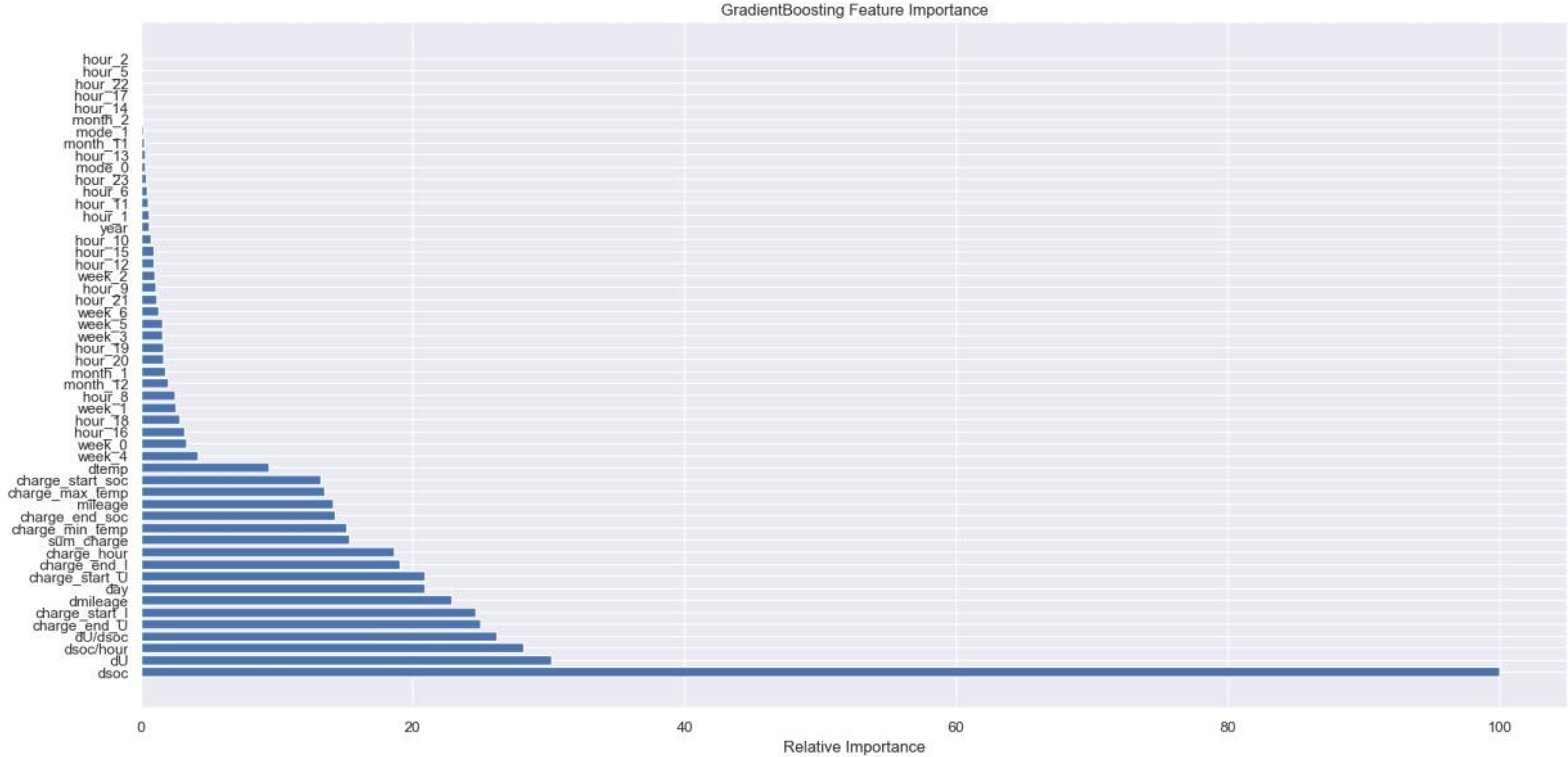
```
# gbm model
   gbm_model = GradientBoostingRegressor(n_estimators=2000, max_depth=4,
min_samples_split=2, min_samples_leaf=2, max_features='auto', subsample=0.6,
learning_rate=0.008) # 少样本

   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
```

Model design



car12

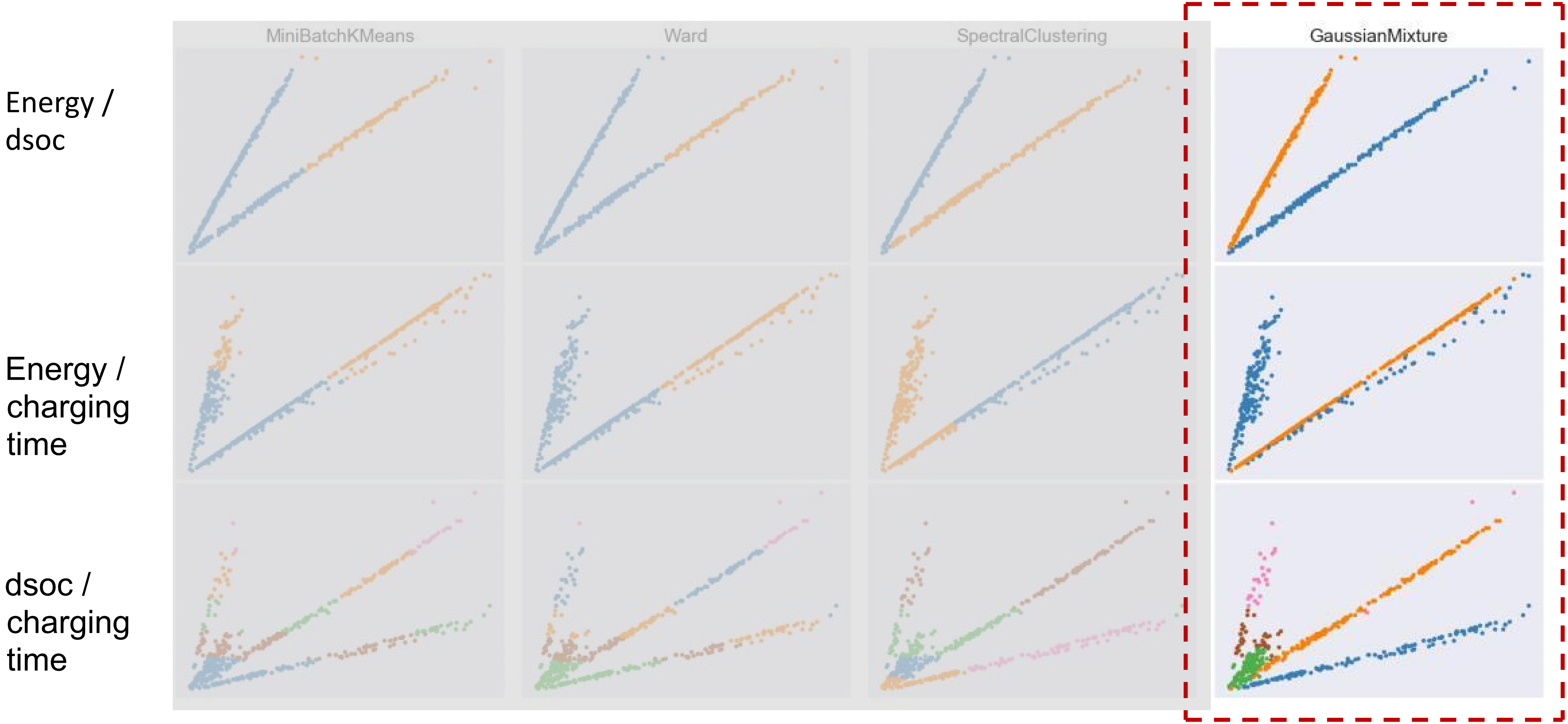
Model design



car12

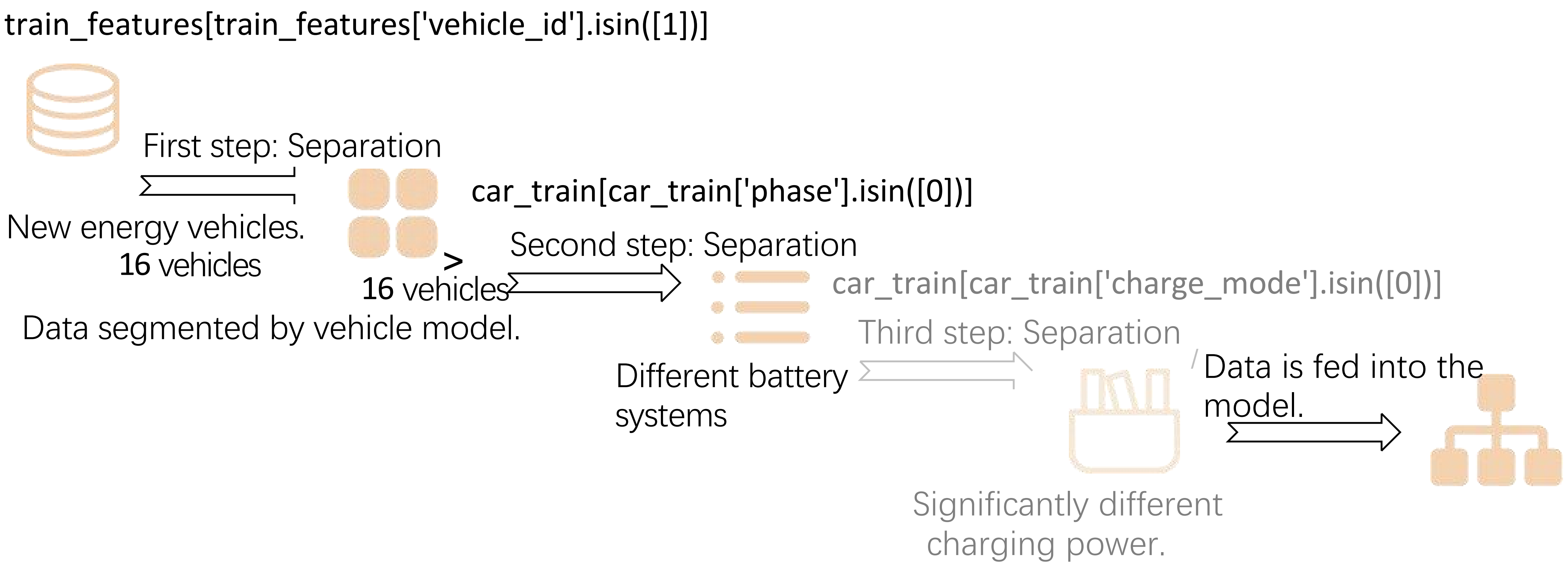
Model design

Gaussian Mixture Model (GMM)

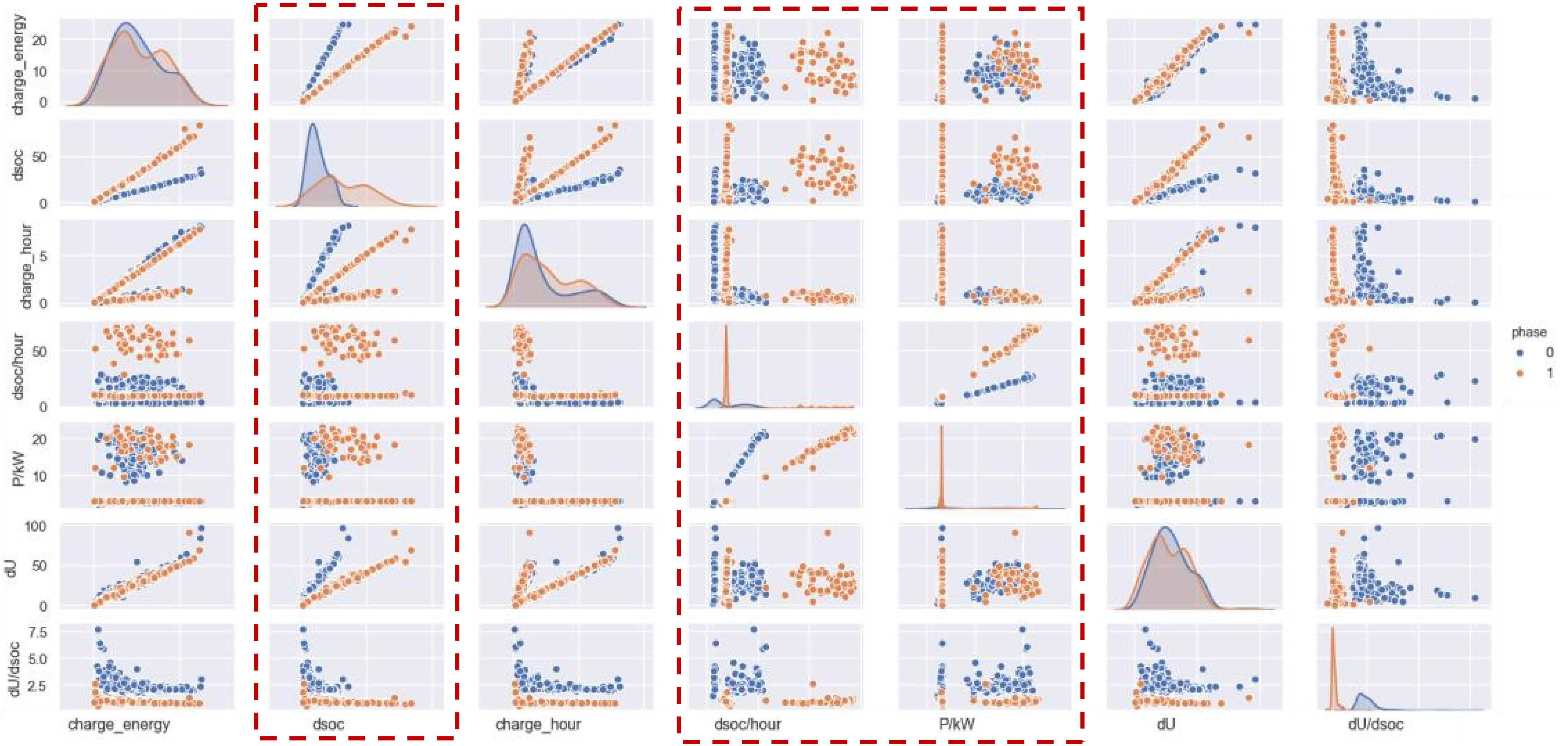


car12

Model design



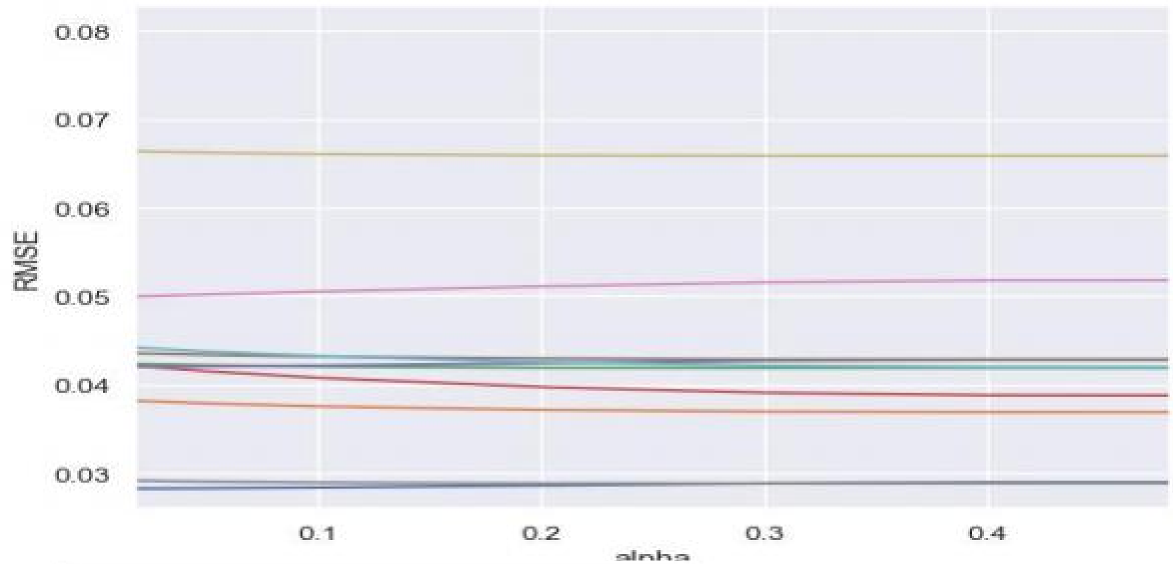
Model design

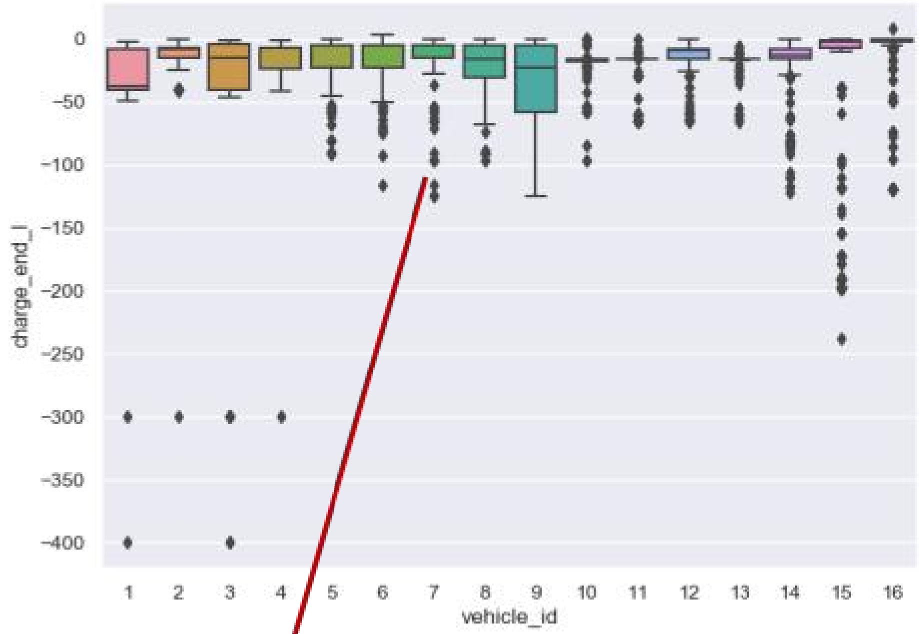


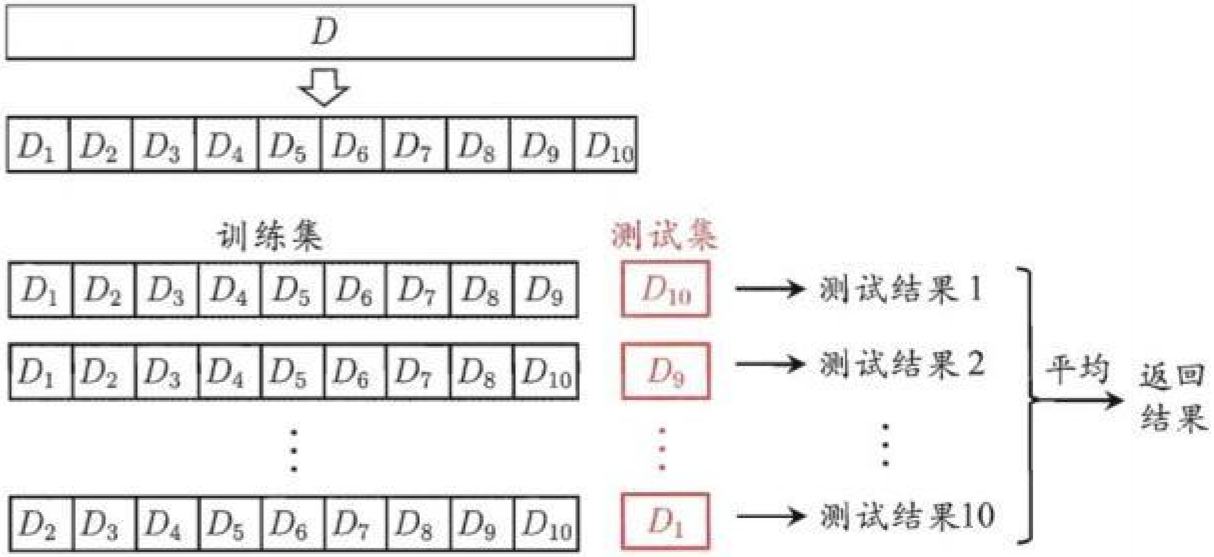
car12

- Data analysis and cleaning
- Model design
- Algorithm structure
- Portability & Engineering Optimization

Algorithm structure







A more robust standardization method is used for outliers. Each feature is independently centered and scaled, ensuring that outlier feature points do not affect the standardization results while maintaining their outlier characteristics.

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

K-fold cross-validation and cross-training based on K-fold

Algorithm structure

Linear algorithm

Ridge Regression

Used for data with multicollinearity (highly correlated independent variables). L2 regularization penalty distributes the weights during shrinkage, reducing the sum of squared weights.

Lasso Regression

When a set of predictor variables is highly correlated, Lasso helps with feature selection. L1 regularization penalty concentrates the weights during shrinkage, resulting in sparse solutions, and extracts features for sparsity.

ElasticNet Regression

Use L1 to train and prioritize L2 as the regularization matrix. When there are multiple correlated features, Lasso randomly selects one of them, while ElasticNet tends to select both.

SVM Regression

Suitable for high-dimensional feature spaces, it solves a convex quadratic programming problem and is sensitive to missing values.

Ensemble learning algorithm

Gradient boosting

It can fit complex nonlinear relationships and flexibly handle various types of data, including continuous and discrete values. There is dependency among weak learners, which makes it prone to overfitting.

XGboost

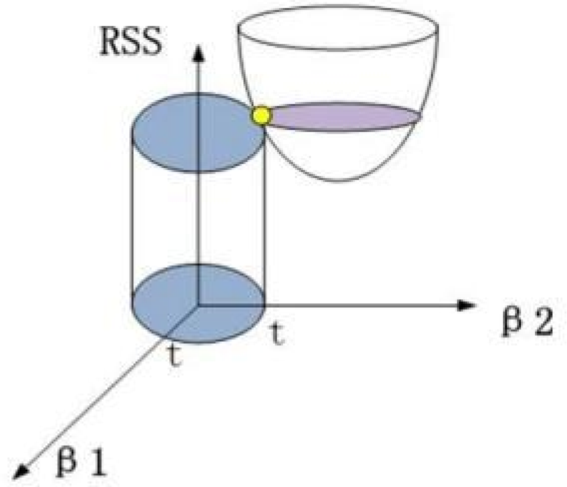
Handles samples with missing feature values, uses regularization to prevent overfitting, and supports parallel processing.

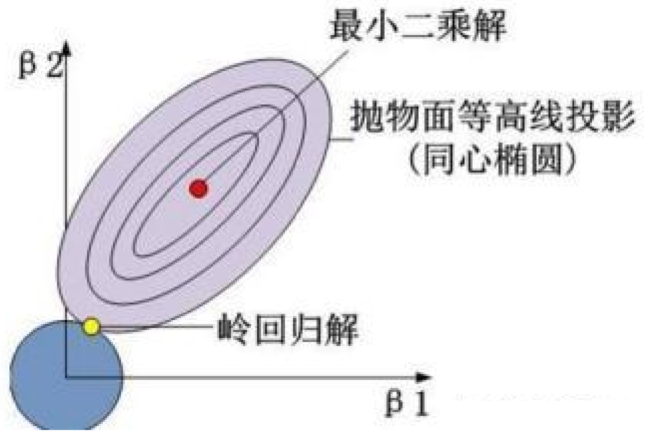
Ridge Regression

$$\min_w ||xw - y||_2^2 + \underline{\alpha||w||_2^2}$$

- Based on the optimization objective of minimizing the sum of squared residuals using the least squares method, an L2 regularization penalty term is introduced to control the complexity of shrinkage, making the weights more robust to collinear features.
- To shrink the weights, (α * weight) is added to the least squares term to achieve a very low variance.

- L2 norm: Represents the square root of the sum of squares of elements in vector x, similar to Euclidean distance, measuring differences between vectors, such as the sum of squared differences. The function of the L2 norm is to prevent the model from becoming overly complex to fit the training data, thus improving the model's generalization ability.



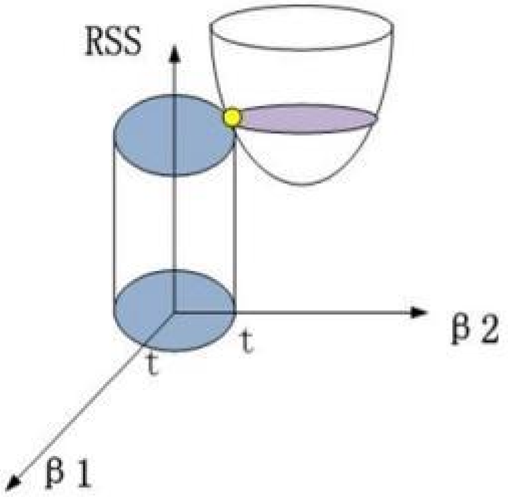


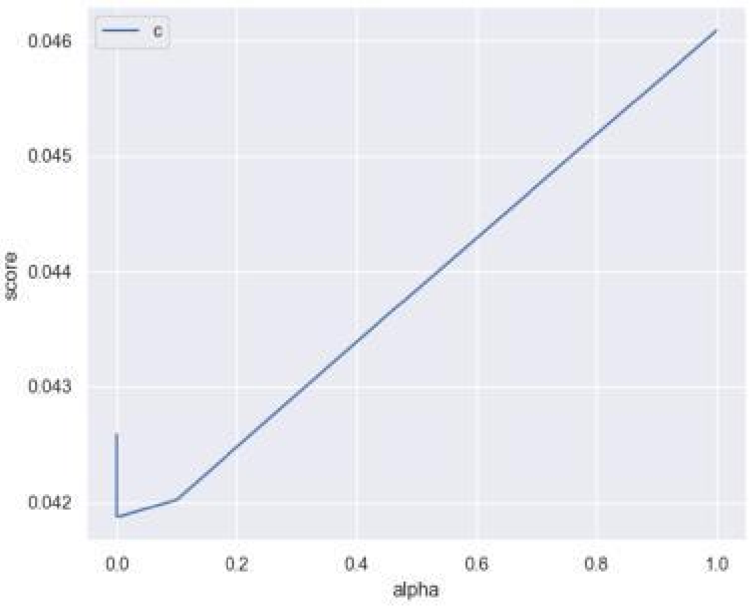
Algorithm structure

Ridge Regression

$$\min_w ||xw - y||_2^2 + \alpha||w||_2^2$$

```python
#### 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 = kfolds))
    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()))
```



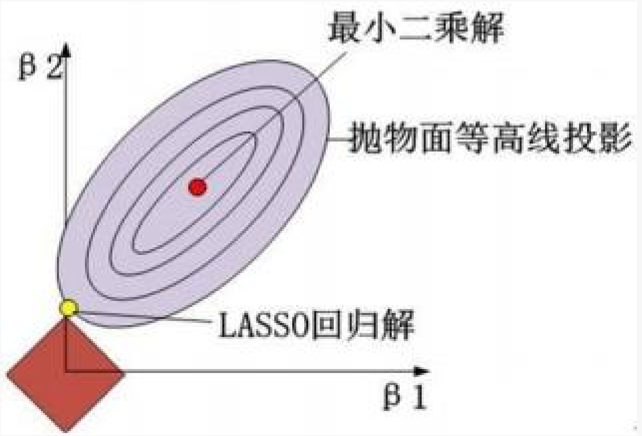


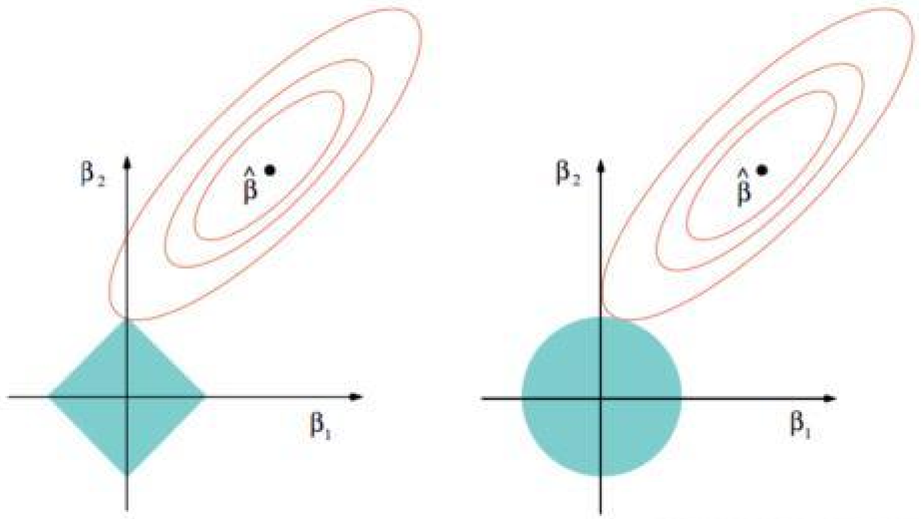
Algorithm structure

Lasso Regression

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

- Based on the optimization objective of minimizing the sum of squared residuals using the least squares method, an L1 regularization penalty term is introduced to control the complexity of shrinkage. This often results in sparse weights, achieving the effect of variable selection.
- L1 norm: Represents the sum of the absolute values of non-zero elements in vector x, also known as the Manhattan distance or minimum absolute error. It measures differences between vectors, such as the sum of absolute errors. For vectors x1 and x2, the L1 norm can achieve feature sparsity by eliminating features that carry no information.





The tangent point between the contour and the constraint region is the optimal solution of the objective function. The Lasso constraint region is a square, which allows for tangency with the coordinate axes, resulting in some feature weights being zero and achieving variable selection through sparsity.

In contrast, the Ridge method's constraint region is circular, with tangency points only on the circumference and not with the coordinate axes. Although it also shrinks the original coefficients, none of the values in any dimension are zero, so the final model retains all variables.

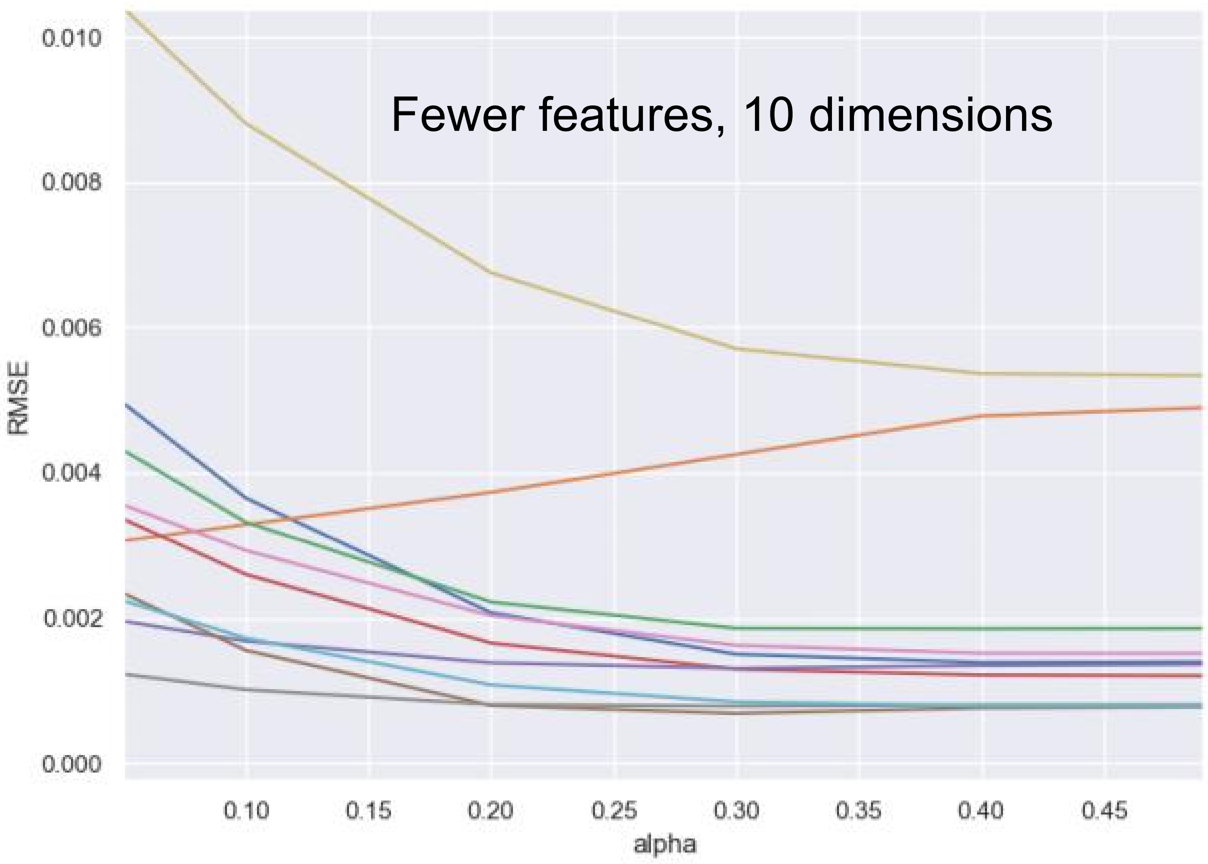
Algorithm structure

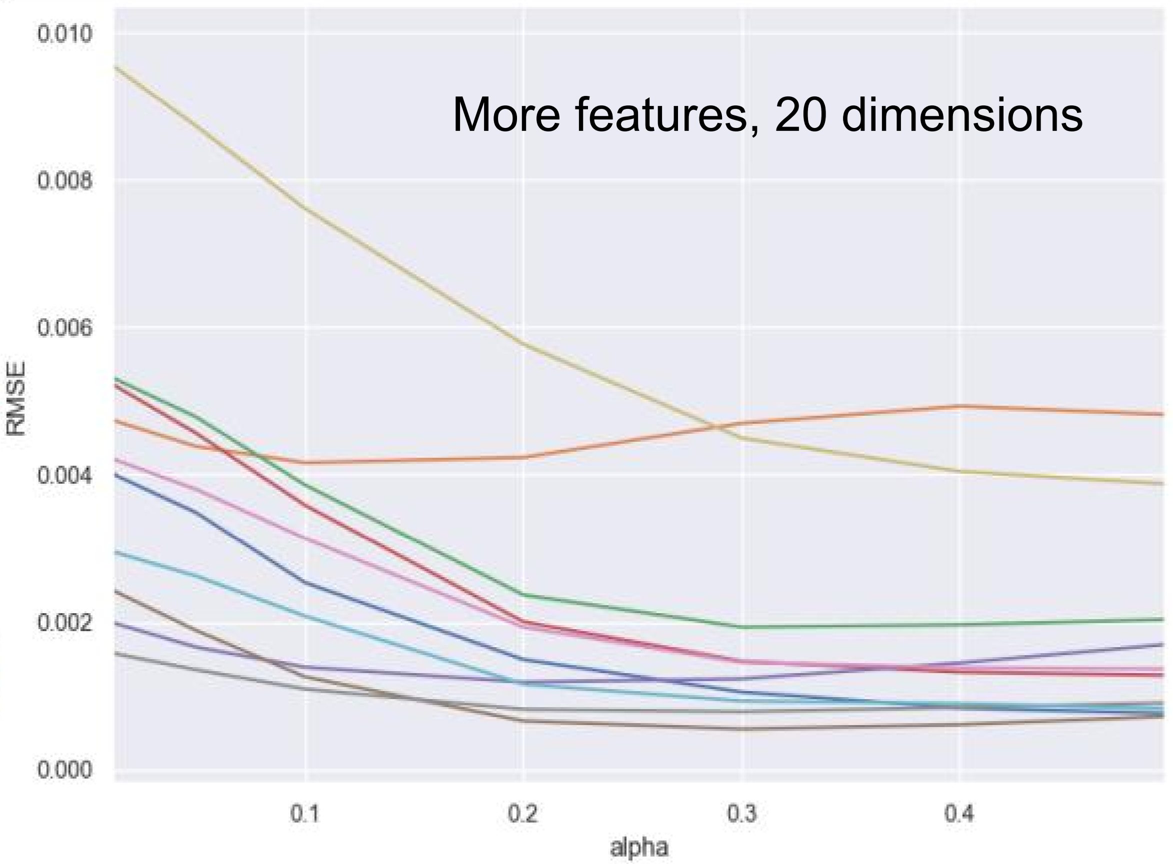
Lasso Regression

```python
#### 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 = kfolds
                                )).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']
```

Algorithm structure





ElasticNet Regression

- Based on the optimization objective of minimizing the sum of squared residuals using the least squares method, both L1 and L2 regularization penalty terms are introduced to control the complexity of shrinkage. This is managed through the `l1_ratio` and `alphas` parameters.

$$min_{\omega}\{\frac{1}{2n_{samples}}||X\omega - y||_2{}^2 + \underline{\alpha\rho||\omega||_1} + \underline{\frac{\alpha(1-\rho)}{2}||\omega||_2{}^2}\}$$

Algorithm structure

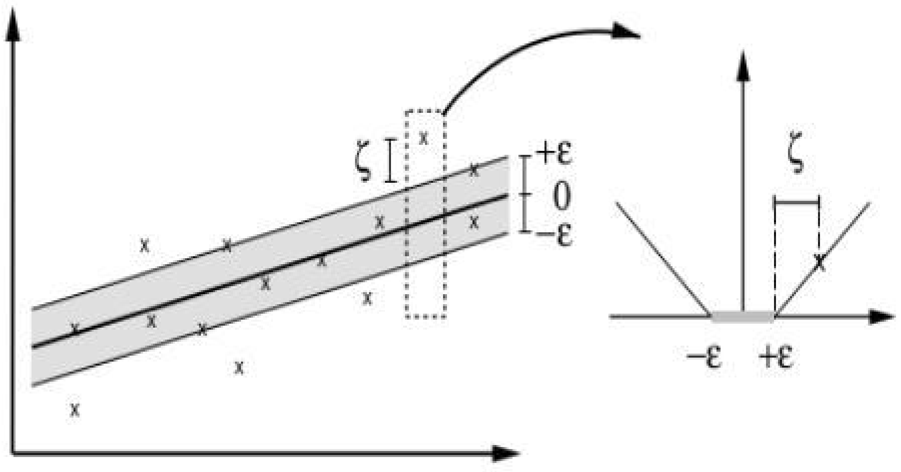
SVR Regression

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

Support Vector Regression (SVR) can tolerate a deviation of ε between f(x) and y. It constructs a margin band centered around f(x) with a width of 2ε. If the training samples fall within this margin, the prediction is considered correct.

Data points outside the dashed region have residuals, which are the distances to the boundary of the margin. Similar to linear models, the aim is to minimize these residuals.

SVR transforms the actual problem into a high-dimensional feature space through a nonlinear transformation. In this high-dimensional space, a linear decision function is constructed to achieve a nonlinear decision function in the original space. This elegantly solves the dimensionality problem, making the algorithm's complexity independent of the sample dimensions.



Suitable for:

- High-dimensional feature spaces, remaining effective even when the data dimensionality exceeds the number of samples.
- Solving a convex quadratic programming problem, which yields a global optimum and addresses the issue of local extrema that cannot be avoided in neural network methods.

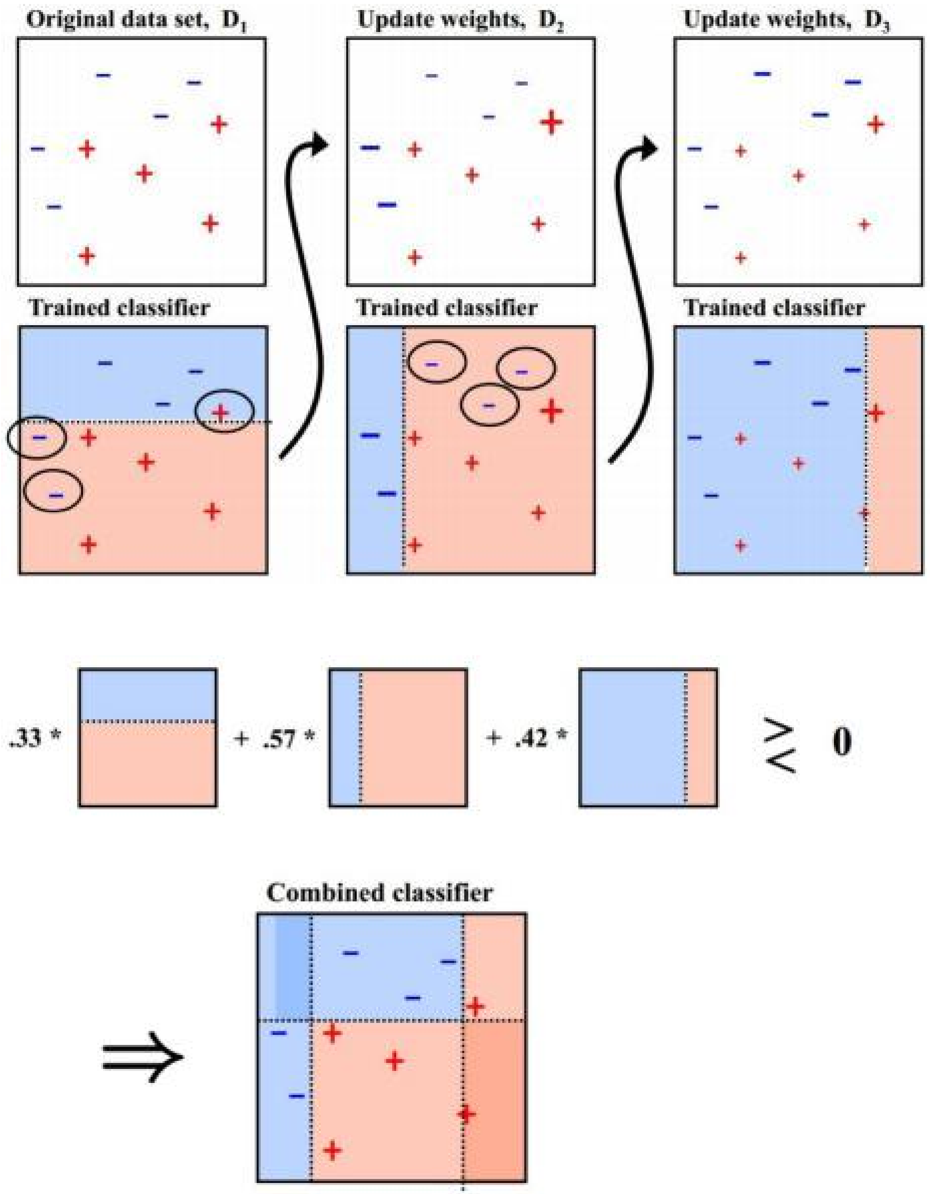
Not suitable for:

- The need to correctly choose the kernel function.
- Sensitivity to missing data.

Gradient boosting

Ensemble learning combines base learners with different weights in a linear combination, allowing well-performing learners to be reused. It calculates pseudo-residuals based on the initial model, then builds a base learner to explain these pseudo-residuals, reducing them in the gradient direction. The base learner is then multiplied by a weight coefficient (learning rate) and linearly combined with the original model to form a new model. This iterative process continues to find a model that minimizes the expected value of the loss function.

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



Suitable for:

- Fitting complex nonlinear relationships.
- Flexibly handling various types of data, including continuous and discrete values.

Not suitable for:

- Dependency among weak learners, which can lead to overfitting.
- Weak resistance to noise.

Algorithm structure

XGboost

Ensemble learning combines predictions from multiple base learner trees to obtain the final result. Initially, a tree is trained using the training set and the true values (i.e., the correct answers). This tree then predicts the training set, resulting in predicted values for each sample. The difference between these predicted values and the true values is the "residual." The next step involves training a second tree, but instead of using the true values, the residuals are used as the target. After training the two trees, you can calculate the residuals for each sample again and proceed to train a third tree, continuing this process iteratively.

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

Suitable for:

- Samples with missing feature values. XGBoost can automatically learn the direction of splits for features with missing values.
- Regularization to prevent overfitting. XGBoost includes a regularization term in the cost function to control model complexity.
- Parallel processing.

Not suitable for:

- Complex models.
- Poor interpretability.
- High maintenance costs.

Algorithm structure

Linear algorithm

Ridge Regression

Used for data with multicollinearity (highly correlated independent variables). L2 regularization penalty distributes the weights during shrinkage, reducing the sum of squared weights.

Lasso Regression

When a set of predictor variables is highly correlated, Lasso helps with feature selection. L1 regularization penalty concentrates the weights during shrinkage, resulting in sparse solutions, and extracts features for sparsity.

ElasticNet Regression

Use L1 to train and prioritize L2 as the regularization matrix. When there are multiple correlated features, Lasso randomly selects one of them, while ElasticNet tends to select both.

SVM Regression

Suitable for high-dimensional feature spaces, it solves a convex quadratic programming problem and is sensitive to missing values.

Ensemble learning algorithm

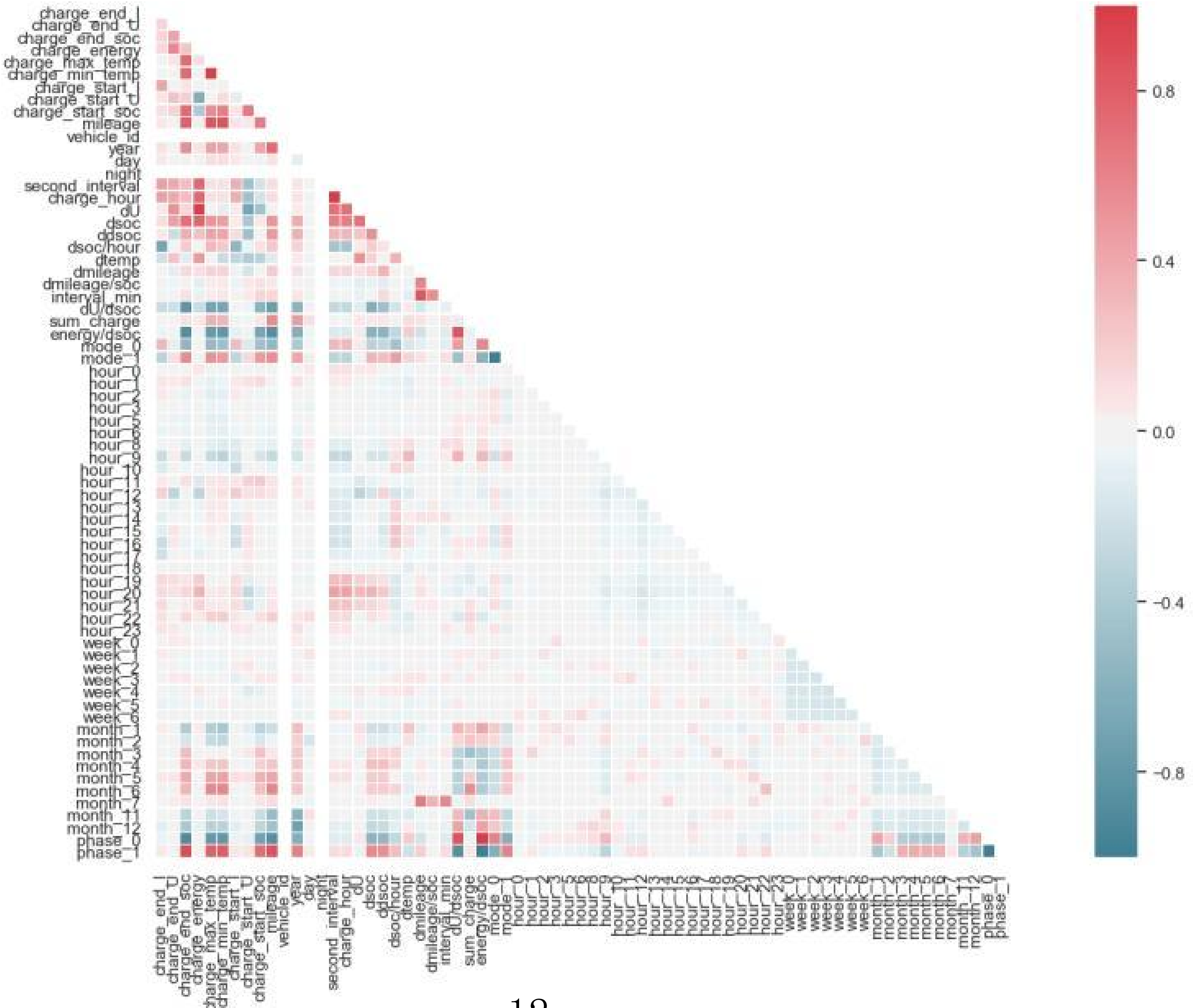
Gradient boosting

It can fit complex nonlinear relationships and flexibly handle various types of data, including continuous and discrete values. There is dependency among weak learners, which makes it prone to overfitting.

XGboost

Handles samples with missing feature values, uses regularization to prevent overfitting, and supports parallel processing.

Algorithm structure



car12

Algorithm structure

Car 10

Training set original data: 187

After data processing: 180

Feature dimensions: 9

Features: 'charge_hour', 'dsoc', 'dsoc/hour', 'charge_min_temp', 'charge_end_U', 'charge_start_U', 'dU', 'charge_start_I', 'charge_end_I'

Test set data: 14

Feature dimensions: 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 →

K-fold Random shuffle.

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

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

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

$$\vdots$$

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

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

Algorithm structure

$\text{Model1}_{\text{k-fold1}}$

K-fold
Random
shuffle.

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

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

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

$$\vdots$$

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

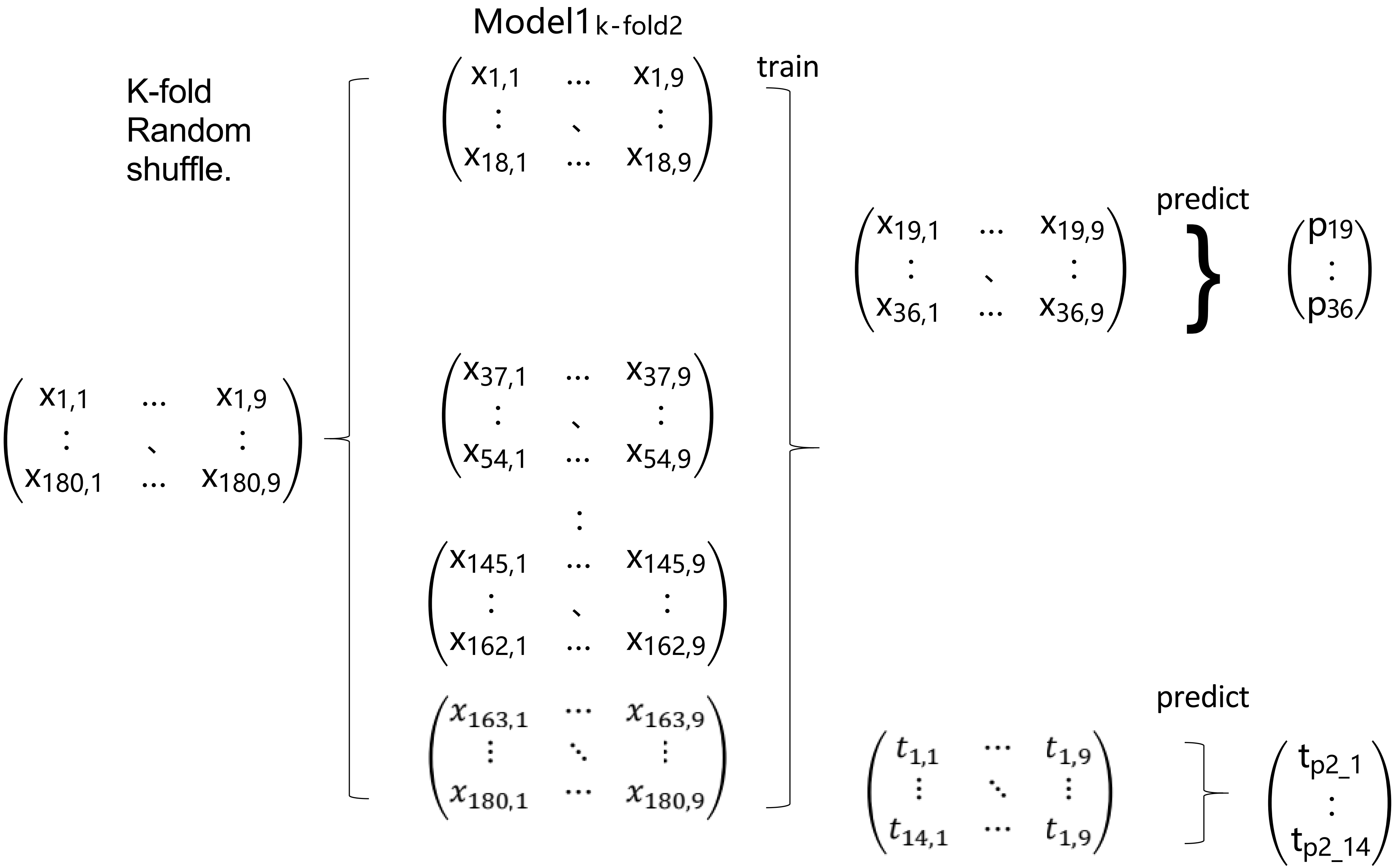
$$\begin{pmatrix} x_{163,1} & \cdots & x_{163,9} \\ \vdots & \ddots & \vdots \\ x_{180,1} & \cdots & x_{180,9} \end{pmatrix}$$

train

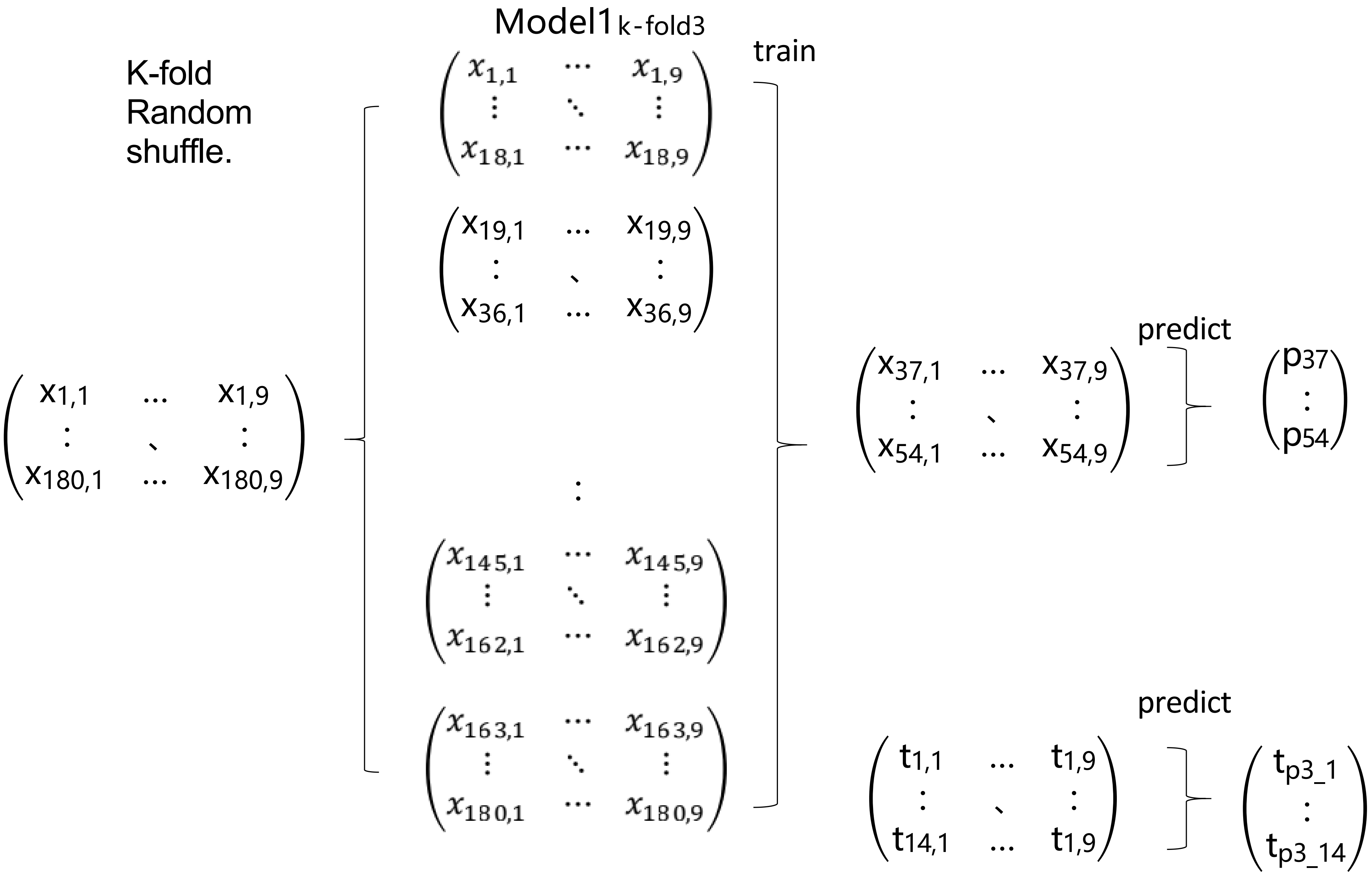
$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,9} \\ \vdots & \ddots & \vdots \\ x_{18,1} & \cdots & x_{18,9} \end{pmatrix} \xrightarrow{\text{predict}} \begin{pmatrix} p_{1} \\ \vdots \\ p_{18} \end{pmatrix}$$

$$\begin{pmatrix} t_{1,1} & \cdots & t_{1,9} \\ \vdots & \ddots & \vdots \\ t_{14,1} & \cdots & t_{1,9} \end{pmatrix} \xrightarrow{\text{predict}} \begin{pmatrix} t_{p1\_1} \\ \vdots \\ t_{p1\_14} \end{pmatrix}$$

Algorithm structure



Algorithm structure



K-fold
Random
shuffle.

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

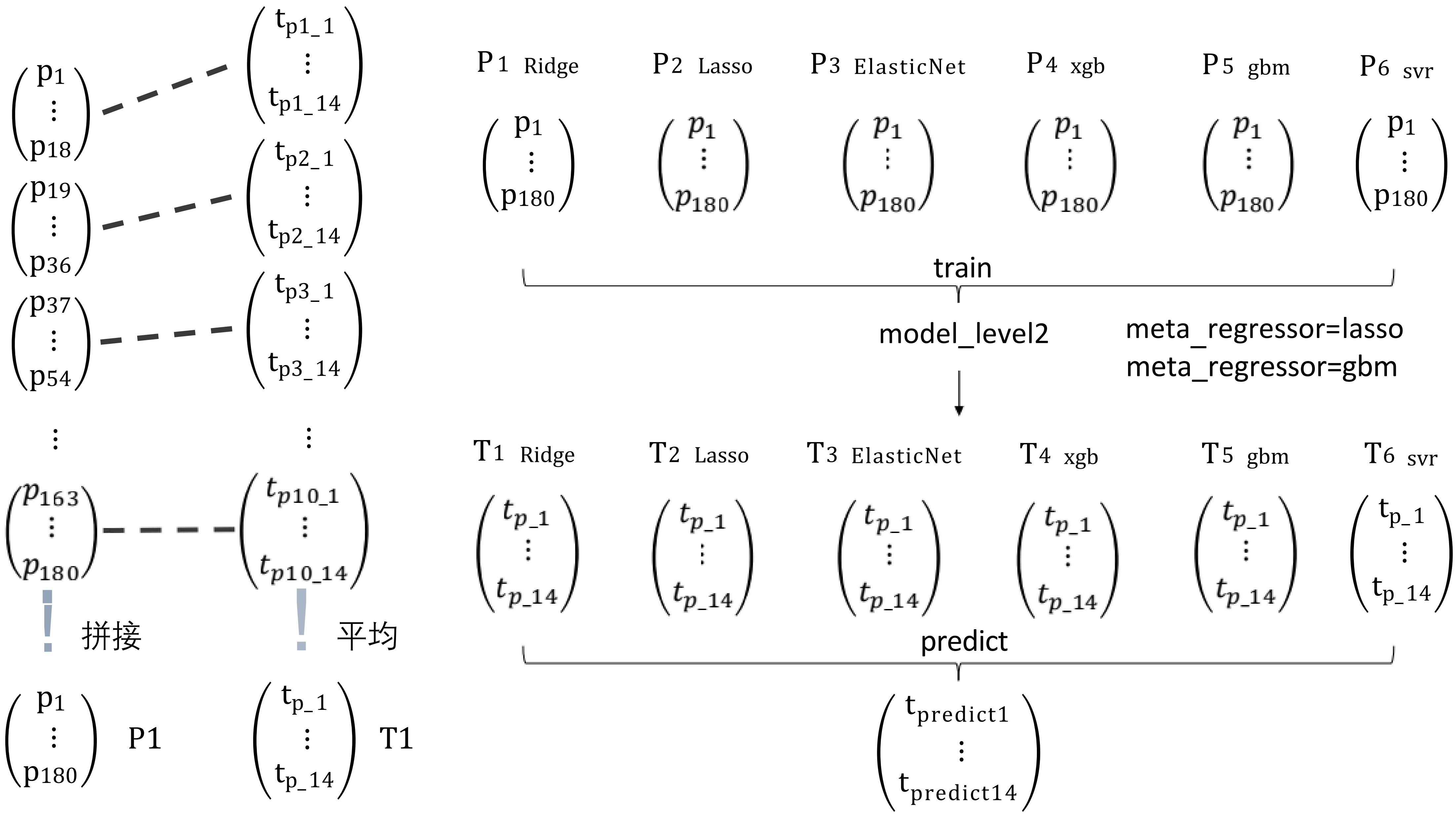
Model1$_{\text{k-fold10}}$ train

$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,9} \\ \vdots & \ddots & \vdots \\ x_{18,1} & \cdots & x_{18,9} \end{pmatrix}$$
$$\begin{pmatrix} x_{19,1} & \cdots & x_{19,9} \\ \vdots & \ddots & \vdots \\ x_{36,1} & \cdots & x_{36,9} \end{pmatrix}$$
$$\begin{pmatrix} x_{37,1} & \cdots & x_{37,9} \\ \vdots & \ddots & \vdots \\ x_{54,1} & \cdots & x_{54,9} \end{pmatrix}$$
$$\vdots$$
$$\begin{pmatrix} x_{145,1} & \cdots & x_{145,9} \\ \vdots & \ddots & \vdots \\ x_{162,1} & \cdots & x_{162,9} \end{pmatrix}$$

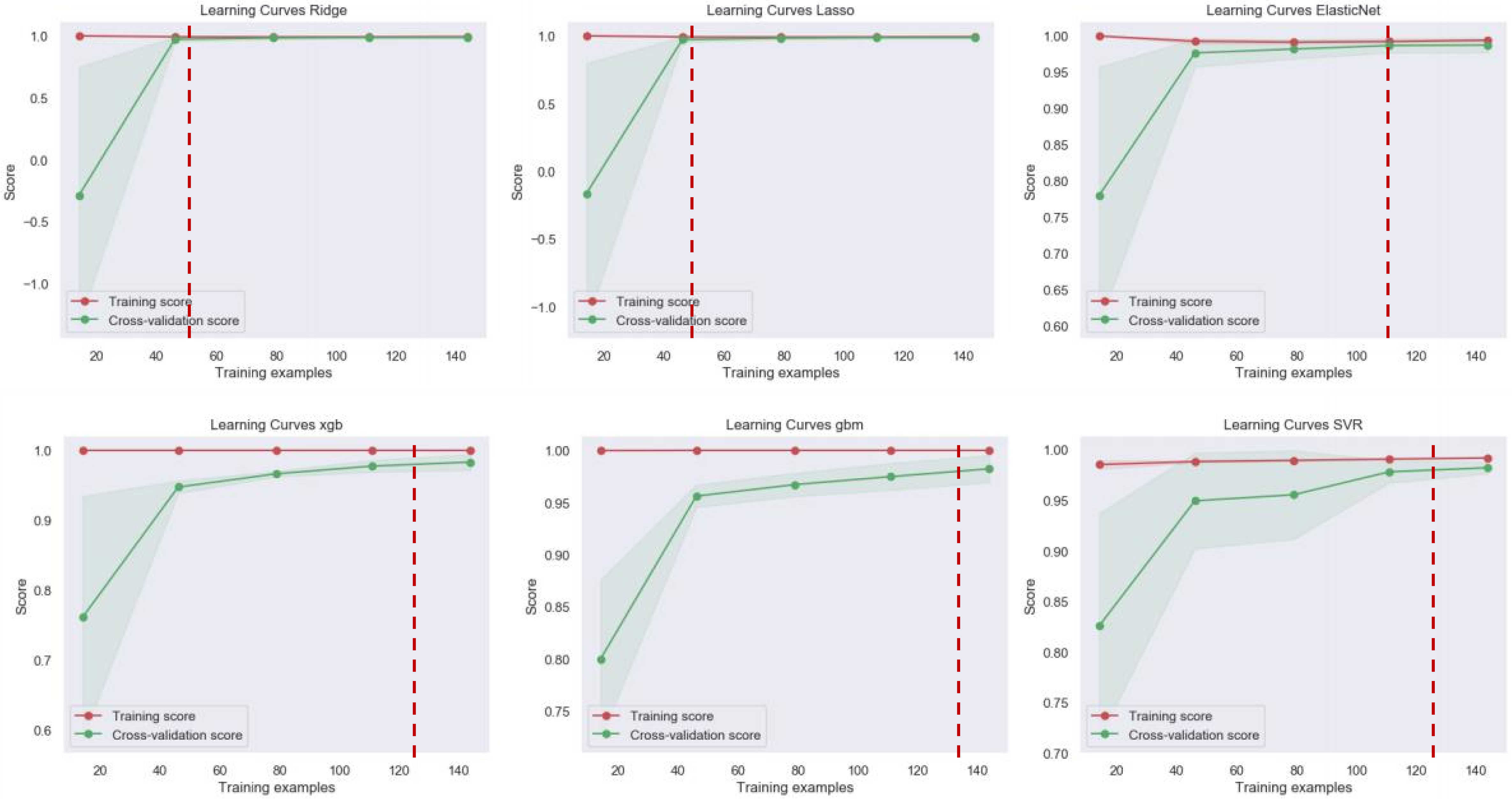
$$\begin{pmatrix} x_{163,1} & \cdots & x_{163,9} \\ \vdots & \ddots & \vdots \\ x_{180,1} & \cdots & x_{180,9} \end{pmatrix} \xrightarrow{\text{predict}} \begin{pmatrix} p_{163} \\ \vdots \\ p_{180} \end{pmatrix}$$

$$\begin{pmatrix} t_{1,1} & \cdots & t_{1,9} \\ \vdots & \ddots & \vdots \\ t_{14,1} & \cdots & t_{1,9} \end{pmatrix} \xrightarrow{\text{predict}} \begin{pmatrix} t_{p10\_1} \\ \vdots \\ t_{p10\_14} \end{pmatrix}$$

Algorithm structure



Algorithm structure



- Data analysis and cleaning
- Model design
- Algorithm structure
- Portability & Engineering Optimization

Portability & Engineering Optimization

Preliminary stage

- DNN: Deep Neural Network
- Complex Gradient Boosted Trees

Preliminary stage:
Final B leaderboard rank: 2nd
Gap with 1st place: 0.002

Disadvantages:
- Long training time, difficult to find a global optimum.
- Heavy model with poor responsiveness.
- Insufficient data volume to support the model.
- Model is significantly affected by data distribution.

Final stage: Simplifying complexity.

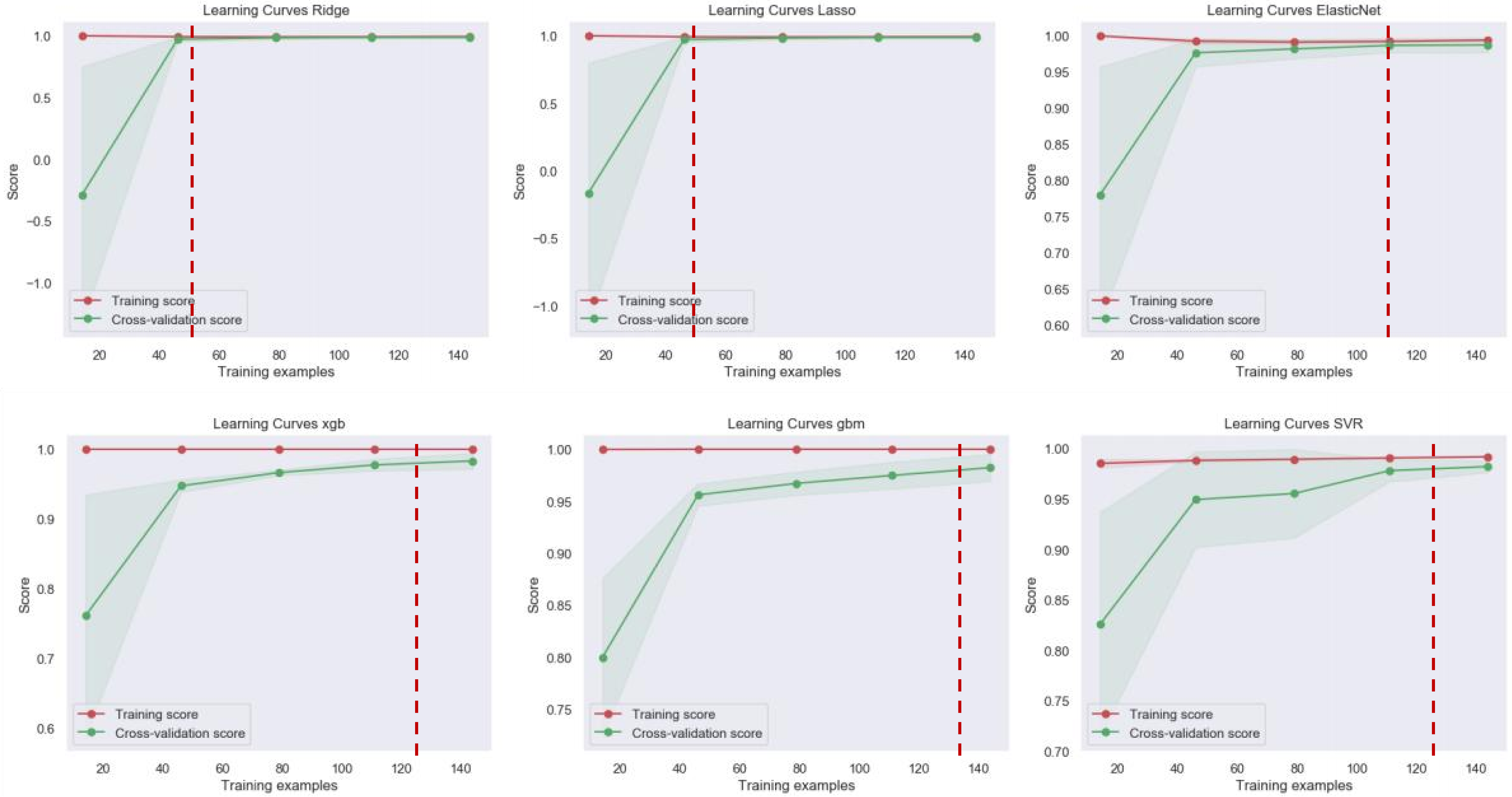
- Faster training speed, in tens of seconds.
- Stronger interpretability with a lighter model and fusion algorithm.
- Greater robustness to differences in data distribution, suitable for a wider range of data distributions.
- Simpler outlier handling by leveraging algorithm characteristics.
- Model is not sensitive to its own parameters, ensuring that score remains high even with parameter changes within a certain range.
- Model is not sensitive to dataset changes, achieving leading scores with less data consistently.

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

Low-energy model features.

Reasonable tree construction

Portability & Engineering Optimization



Portability & Engineering Optimization

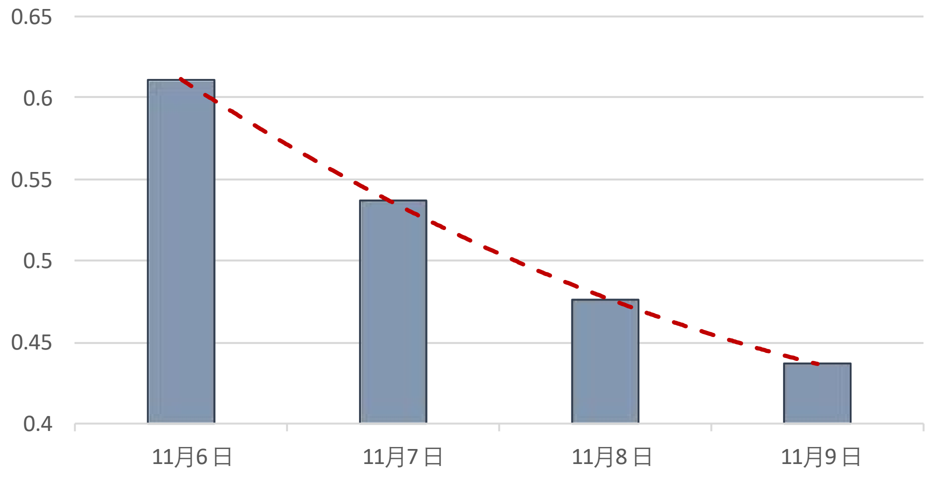
```python
def car10_feature(ca, train_ornot):
    car = ca.copy()
    car['sum_charge'] = car['charge_hour'].cumsum()

    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
```

```python
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
```

```python
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
```

Portability & Engineering Optimization

Final evaluation metrics.



The type of data in new energy vehicles is determined, and you cannot rely solely on piling up data. The quality of data preprocessing and the choice of model algorithms are the core aspects.

Thoughts on engineering practice.

- Edge computing scenario: Based on linear models and weak learners, it requires only small computational power and cost to be applied on vehicles, and can complete predictions in seconds.

- Cloud integration: Internet of Vehicles, where big data is aggregated on a unified cloud platform to process similar data distributions for different vehicle models together.

New Energy Vehicle Big Data Innovation and Entrepreneurship Competition

Electric Vehicle Power Battery Charging Energy Prediction

Thank you to the judges and fellow students for listening.

Thank you!

Speaker: Liu Xiaoman