Supplementary file

Data-driven interpretable machine learning for prediction of porosity and permeability of tight sandstone reservoir

Liu Cao[1,2] , Fujie Jiang[1,2,*], Zhangxing Chen[1,3,4], Yang Gao[1,2,5], Lina Huo[1,2], Di Chen[1,2]

[1] *National Key Laboratory of Petroleum Resources and Engineering, China University of Petroleum (Beijing), Beijing, 102249, P. R. China.*

[2] *College of Geosciences, China University of Petroleum (Beijing), Beijing, 102249, P. R. China.*

[3] *Institute of Digital Twins, Eastern Institute of Technology, Ningbo, 315200, P. R. China.*

[4] *Chemical and Petroleum Engineering, Schulich School of Engineering, University of Calgary, T2N 1N4, Calgary, Canada.*

[5] *Research Institute of Petroleum Exploration and Development, PetroChina, Beijing, 100089, P. R. China.*

E-mail address: cl@student.cup.edu.cn (L. Cao); jfjhtb@163.com (F. Jiang); zxchen@eitech.edu.cn (Z. Chen); gaoyang1112024@163.com (Y. Gao); hln10171007@163.com (L. Huo); chendcup@cup.edu.cn (D. Chen).

[*] Corresponding author (ORCID: 0000-0002-0089-2972)

Cao, L., Jiang, F., Chen, Z., Gao, Y., Huo, L., Chen, D.. Data-driven interpretable machine learning for prediction of porosity and permeability of tight sandstone reservoir. Advances in Geo-Energy Research, 2025, 16(1): 21-35.

The link to this file is: https://doi.org/10.46690/ager.2025.04.04

Contents of This File

- Other figures and tables
- Mathematical theory of different algorithms
- Abbreviation
- Mathematical descriptions of evaluation metrics

1. Other figures and tables

Table S1. Hyperparameter setting of base learners RF.

Models	n_estimators	max_depth	random_state
RF1	222	15	42
RF2	100	None	35

Table S2. Hyperparameter setting of base learners CatBoost.

Models	iterations	depth	learning_rate	random_state
CatBoost1	535	9	0.01325	32
CatBoost2	600	6	0.03	42

Table S3. Hyperparameter setting of other base learners. (The types and default values of DT hyperparameters include criterion = mse, splitter = best, max_depth = None, min_samples_split = 2, min_samples_leaf = 1, and so on. The types and default values of KNN hyperparameters include *n*_neighbors = 5, weights = uniform, algorithm = auto, leaf_size = 30, and so on).

Models	n_estimators	max_depth	learning_rate	random_state
DT	No	None	No	None
KNN	No	No	No	None
LightGBM	100	9	0.037	None
XGBoost	924	6	0.06	None

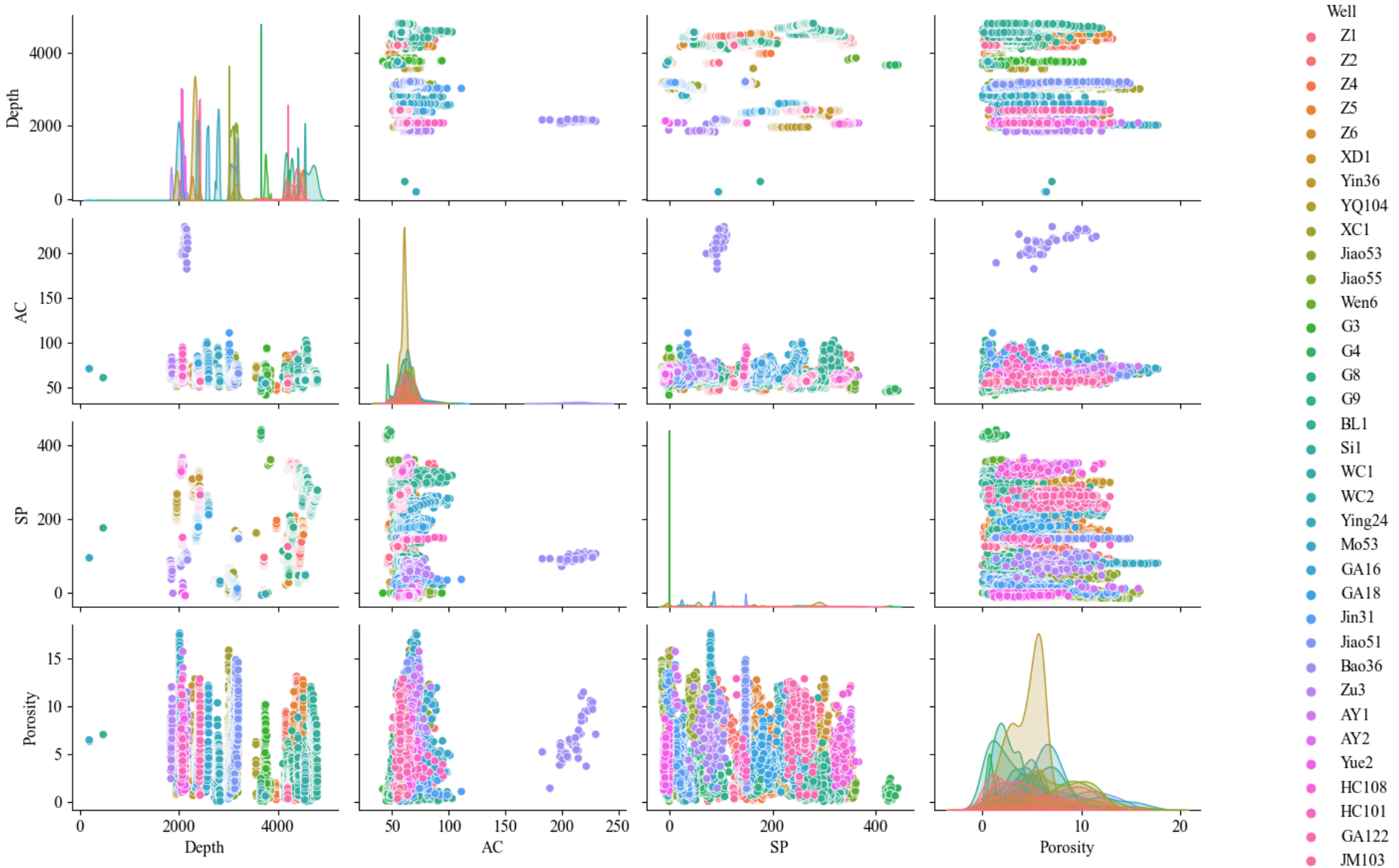

Fig. S1. The pairwise correlation plot of porosity after expanding the dataset.

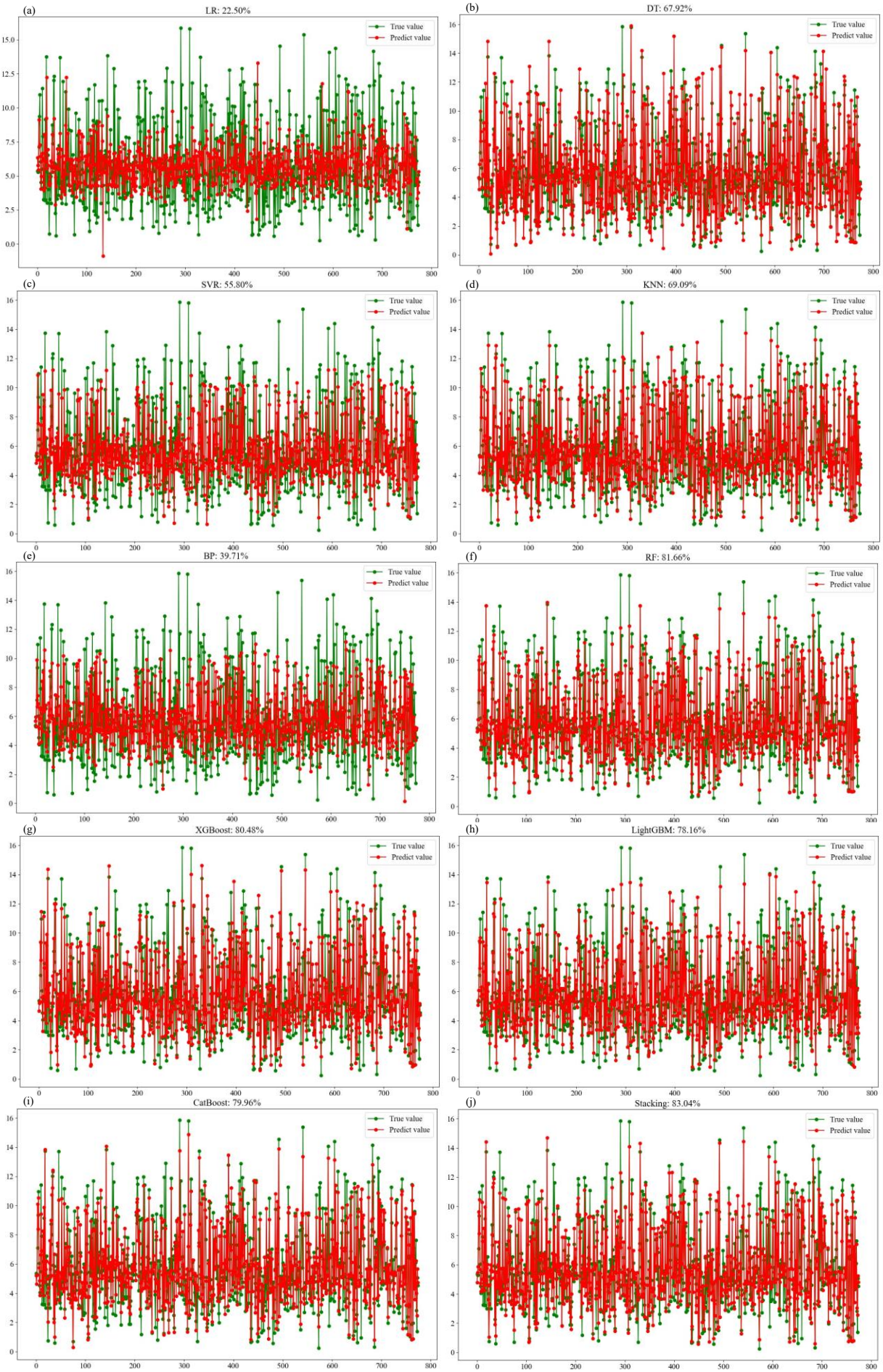

Fig. S2. The R^2 results of ten porosity ML prediction models.

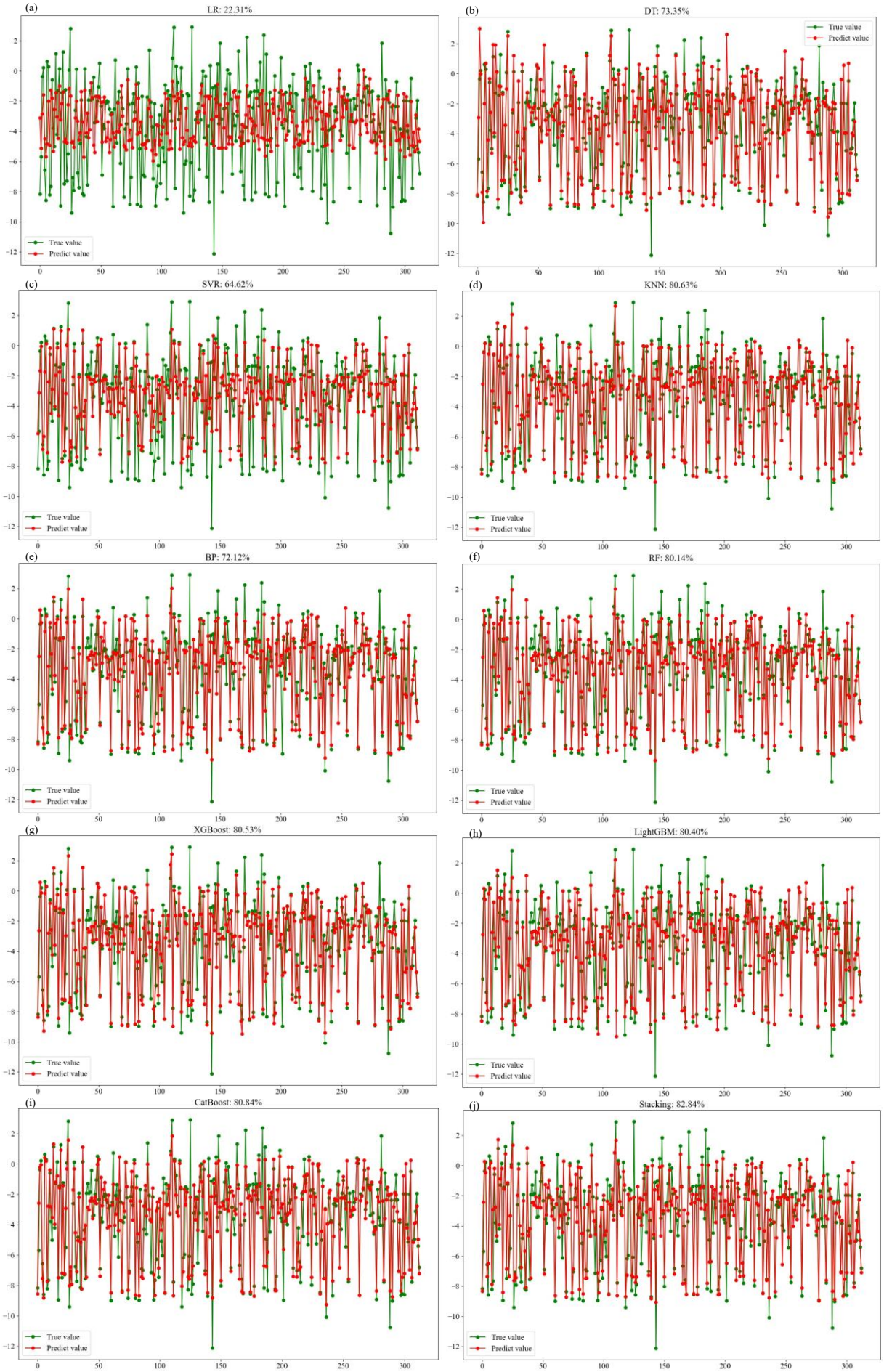

Fig. S3. The R^2 results of ten permeability ML prediction models.

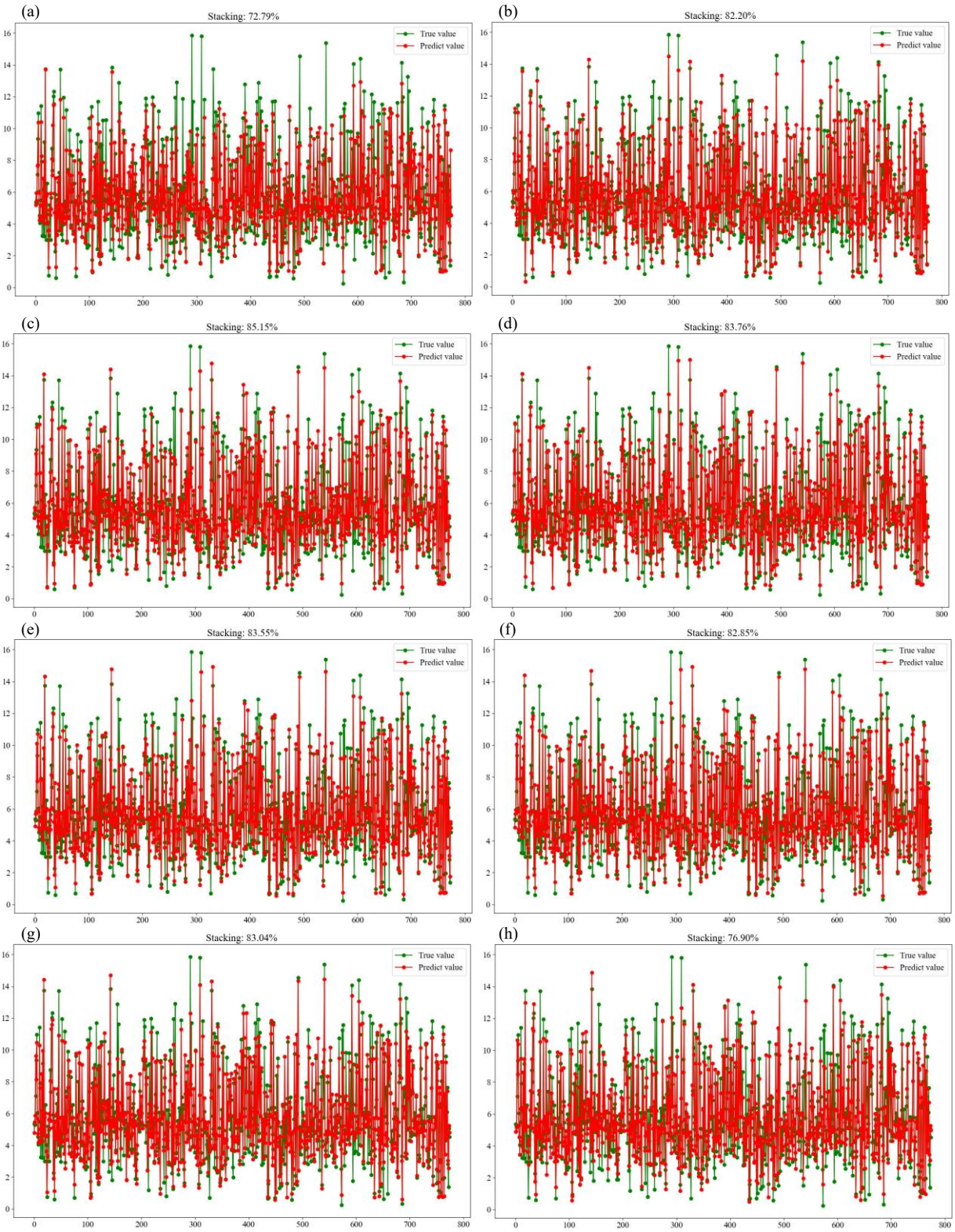

Fig. S4. The R^2 results of feature ablation study for Stacking porosity prediction models. (A)Depth; (B)Depth, AC; (C) Depth, AC, SP; (D) Depth, AC, SP, CAL; (E) Depth, AC, SP, CAL, GR; (F) Depth, AC, SP, CAL, GR, DEN; (G) Depth, AC, SP, CAL, GR, DEN, CNL; (H) AC, SP, CAL, GR, DEN, CNL.

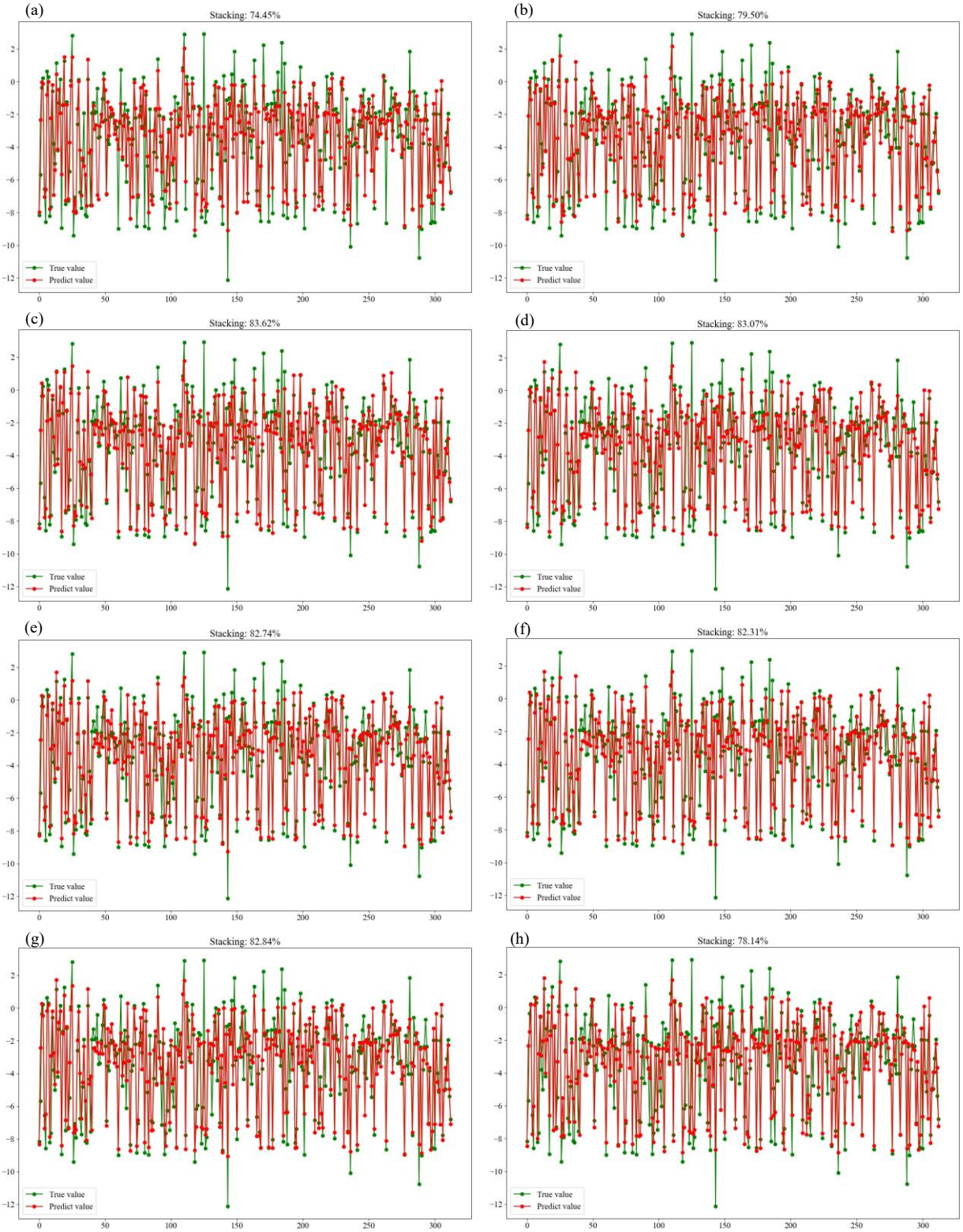

Fig. S5. The R^2 results of feature ablation study for Stacking permeability prediction models. (A)Depth; (B)Depth, DEN; (C) Depth, DEN, SP; (D) Depth, DEN, SP, GR; (E) Depth, DEN, SP, GR, CNL; (F) Depth, DEN, SP, GR, CNL, CAL; (G) Depth, DEN, SP, GR, CNL, CAL, AC; (H) DEN, SP, GR, CNL, CAL, AC.

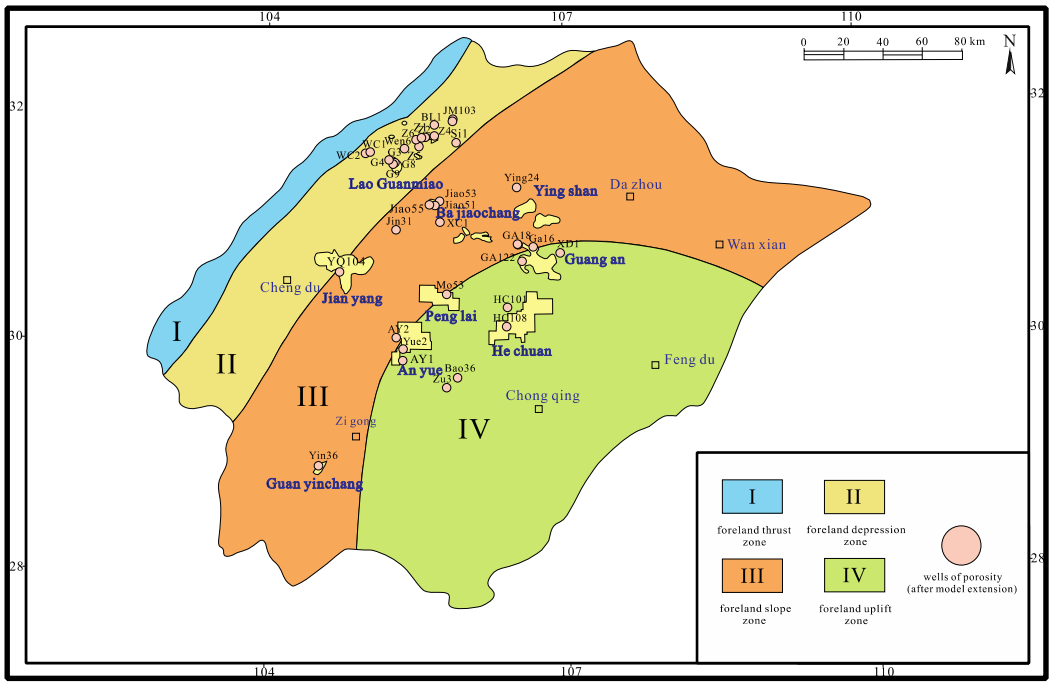

Fig. S6. The tectonic zone map of Sichuan Basin (After the expansion of porosity prediction dataset.).

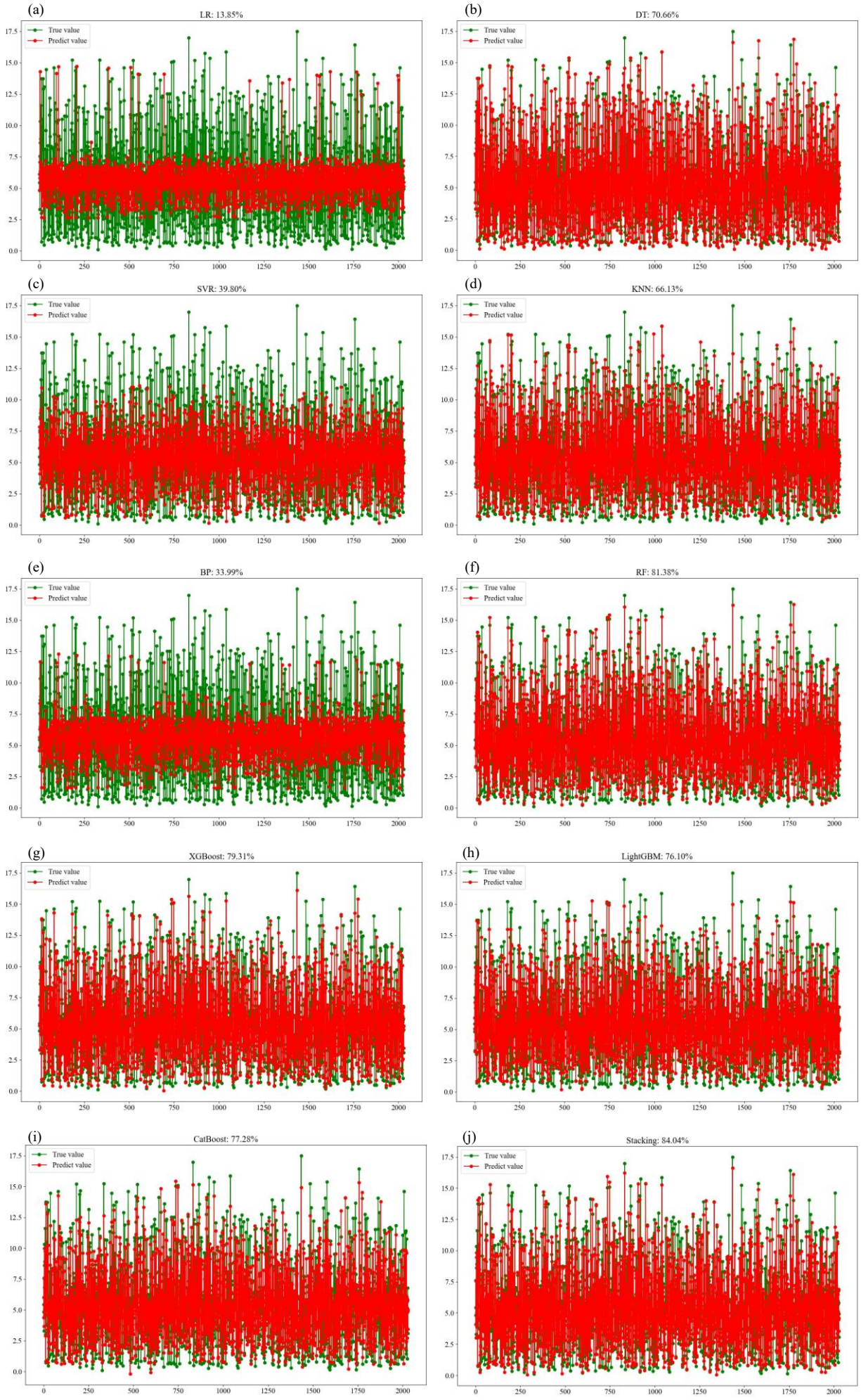

Fig. S7. The R^2 results of ten porosity ML prediction models after model extension(Input features were Depth, AC, SP).

2. Mathematical theory of different algorithms

2.1 LR

The LR algorithm is among the most widely utilized regression algorithms for identifying the linear relationship between two or more variables. LR aims to establish the optimal linear correlation between the independent variable (feature) and the dependent variable (target variable) by fitting a line that minimizes the distance between the data points and itself. The central concept involves determining the weight vector by minimizing the loss function,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n. \quad (1)$$

where n represents the number of features.

2.2 SVR

The SVR algorithm is a regression analysis tool derived from the Support Vector Machine (SVM) framework. Unlike the linear regression algorithm, which attempts to fit all data points, the SVR introduces an "insensitive band" around the prediction line. Data points within this band are considered acceptable and do not contribute to the loss calculation, whereas only data points outside this band impact the loss function. The SVR algorithm's advantage lies in its sparsity and robustness. Owing to its design, where only data points outside the epsilon-insensitive band influence the loss function, SVR can inherently disregard the effects of noise and outliers. This capability leads to a more stable and reliable regression model,

$$f(x) = \sum_{i=1}^{m} (\hat{\alpha}_i - \alpha_i)\kappa(x_i^T x) + b. \quad (2)$$

Where $\kappa(x_i^T x)$ is the kernel function, $\hat{\alpha}_i$ is the predicted value, α_i is the true value，and b is the biased term.

2.3 KNN

KNN is an instance-based learning algorithm. The fundamental principle of KNN is that, within the feature space, if the majority of a sample's K nearest neighbors belong to a certain category, then the sample is also classified into that category. Here, "nearest neighbors" are defined as the closest sample points within the feature space. The KNN algorithm' s strengths include its simplicity and intuitiveness, the lack of need for parameter estimation, and its applicability to both classification and regression tasks. Furthermore, KNN demonstrates a certain degree of robustness to outliers and noisy data.

Algorithm S1. The pseudocode of KNN.

Algorithm K-Nearest Neighbors

Input: Training dataset: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, $x_n, y_n \in R$

$x_n \in$ R were variables, $y_n \in$ R = $\{c_1, c_2, \ldots, c_k\}$ were values or classes of objective variables. i = 1, 2, …, N.

Output: $y = \underset{cj}{argmax} \sum_{x_{i \in N_k(x)}} I(y_i - c_i), i = 1, 2, 3 \ \ldots \ K$

I is the indicator function

$$I(y_i - c_i) = \begin{cases} 1, y_i = c_i \\ 0, otherwise \end{cases}$$

2.4 DT

The DT algorithm is a widely utilized supervised learning method for addressing classification and regression issues. It constructs a DT by iteratively partitioning the dataset into smaller subsets, thereby facilitating predictions and decisions on new data instances. The essential concept of the DT algorithm involves segmenting the dataset into distinct subsets based on data attributes and recursively developing the DT on each subset until a stopping criterion is reached.

ID3 algorithm is the first work of DT algorithm. It utilizes information gain as the criterion for selecting partition attributes. Specifically, it selects the attributes that can maximize the information gain of the partitioned sample set. When the discrete attribute a possesses V possible values $\{a^1, a^2, ..., a^v\}$, and the dataset D is segmented based on attribute a, then V branch nodes are generated. Each V branch node encapsulates all samples in D that correspond to the value a^v of attribute a, denoted as D^V. Subsequently, the information gain from partitioning the sample set D using attribute a can be determined:

$$Gain(D, a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} Ent(D^v). \quad (3)$$

Generally, the higher the information gain, the more significant the "purity boost" achieved through partitioning with attribute a. However, utilizing information gain for attribute division poses a potential issue: when an attribute contains a large number of values, the information entropy for the sample subset corresponding to each attribute value may become minimal. To mitigate the negative impact of this bias, the C4.5 DT algorithm does not directly employ information gain. Instead, it utilizes the gain ratio to select the optimal partition attribute.

The C4.5 algorithm represents an enhancement of the ID3 algorithm, addressing several of ID3's limitations, such as the inability to handle continuous attributes and missing values. C4.5 employs the information gain rate as the criterion for selecting partition attributes, circumventing the ID3 algorithm's tendency to favor attributes with a large number of values. The information gain rate is defined as:

$$Gain_{r}\ ati\ o(D,a) = \frac{\mathrm{Gain}(D,a)}{\mathrm{IV}(a)}, \tag{4}$$

here,

$$IV(a) = -\sum_{v=1}^{V} \frac{|D^{v}|}{|D|} log_{2} \frac{|D^{v}|}{|D|}. \tag{5}$$

This is referred to as the intrinsic value *IV* of attribute *a*. The greater the number of possible values for attribute *a* (that is, the larger *V* is), the higher the value of *IV*(*a*)) tends to be.

The CART (Classification and Regression Trees) algorithm is another widely utilized DT methodology. It employs a binary tree structure and is applicable to both classification and regression tasks. The CART algorithm utilizes the Gini index as the criterion for selecting partition attributes. The Gini coefficient is calculated as follows:

$$Gin\ i(D) = \sum_{k=1}^{|y|} \sum_{k' \neq k} p_{k} p_{k'} = 1 - \sum_{k=1}^{|y|} p_{k}^{2}, \tag{6}$$

where *k* is the number of sample types in the data set; p_k is the proportion of the number of Class *i* samples to the total number of samples.

Gini(*D*) reflects the probability that two samples taken at random from dataset *D* have inconsistent category labels. Therefore, the smaller *Gini*(*D*) is, the higher the purity of the dataset *D*. The result is the Gini index:

$$Gini\ index(D\ ,a) = \sum_{v=1}^{V} \frac{|D^{v}|}{|D|} Gini(D^{v}). \tag{7}$$

Algorithm S2. The pseudocode of DT.

Algorithm Decision Tree

Input: Training dataset: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, $x_n, y_n \in R$

if termination criteria met

return base hypothesis $g_t(x)$

else

learn branching citeria $b(x)$

split *D* to *C* parts $D_C = \{(x_n, y_n): b(x_n) = c\}$

build sub-tree $G_c \leftarrow$ Decision Tree (D_c)

return $G(x) = \sum_{c=1}^{C} [b(x) = c] G_C(x)$

2.5 RF

The foundational principle of RF involves generating several subsamples sets from the original training dataset through bootstrap resampling. A DT model is then built for each subsample set, with the final prediction result obtained by amalgamating the outputs from all the DTs. In a Random Forest, each DT is trained on a random subset of features, meaning only a fraction of the features are selected for partitioning at each node. This approach boosts the model's diversity and, consequently, its integrated prediction accuracy. The benefits of the RF algorithm include its capability to manage high-dimensional data, resistance to overfitting, robustness to outliers and noisy data, and its ability to provide variable importance evaluations.

Algorithm S3. The pseudocode of RF.

Algorithm Random Forests

Input: Training dataset: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, $x_n, y_n \in R$

Base learning algorithm $\mathfrak{L}$

Number of trainings *T*.

Algorithm:

1. **For** t = 1, 2, ..., *T*, **do**
2. $h_t = L(D, D_{bs})$
3. **End for**

Output: $H(x) = \underset{y \in Y}{argmax} \sum_{t=1}^{T} \prod(h_t(x) = y)$

2.6 XGBoost

XGBoost is an ensemble learning (boosting) algorithm that is based on Gradient Boosting Decision Trees (GBDT). The fundamental principle of XGBoost is to enhance the DT model through gradient boosting.

Algorithm S4. The pseudocode of XGBoost.

Algorithm XGBoost (Exact Greedy Algorithm for Split Finding)

Input: Training dataset: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, $x_n, y_n \in R$

I, instance set of current nodes

d, feature dimension

$gain \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$

for $k = 1$ **to** d **do**

$G_L \leftarrow 0$, $H_L \leftarrow 0$

for j in *sorted* (I, by x_{jk}) **do**

$G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$

$score \leftarrow max\ (score,\ \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$

end

end

Output: Split with max score

1.7 LightGBM

The LightGBM algorithm shares similarities with XGBoost, as both are ensemble learning (boosting) algorithms grounded in GBDT. The primary advantages of LightGBM lie in its efficiency and scalability.

Algorithm S5. The pseudocode of LightGBM.

Algorithm LightGBM (Gredient-based One-Side Sampling)

Input: I: training data, d: iterations

Input: a: sampling ratio of large gradient data

Input: b: sampling ratio of small gradient data

Input: *loss*: loss function, L: weak learner

models ← { }, fact ← $\frac{1-a}{b}$ (I)

topN ← a × len(I), randN ← b × len(I)

for $i = 1$ **to** d **do**

preds ← models.predict(I)

g ← loss(I,preds), w ← { 1, 1, ... }

sorted ← GetSortedIndices(abs(g))

topSet ← sorted[1:topN]

randSet ← RandomPick(sorted[topN:len(I)],randN)

usedSet ← topSet + randSet

w[randSet] × = fact ▷ Assign weight *fact* to the small gradient data.

newModel ← L(I[usedSet], − g[usedSet], w[usedSet])

models.append(newModel)

2.8 CatBoost

CatBoost is an ensemble learning (boosting) algorithm that utilizes symmetric trees. Its principal strength lies in its capability to efficiently handle categorical features and mitigate overfitting through the reduction of gradient bias and prediction shift, enhancing both the model's accuracy and its ability to generalize. CatBoost employs symmetric trees as base learners, meaning that at each step, the leaves of the preceding tree are split based on identical conditions. This uniformity simplifies the model's structure, contributing to its stability.

Algorithm S6. The pseudocode of CatBoost.

Algorithm CatBoost

Input: Training dataset: $D = \{(x_i, y_i)\}_{i=1}^{n}$, I, a, L, s, $Mode$

$\sigma_r \leftarrow$ random permutation of $[1, n]$ for r = 0 … s;

$M_0(i) \leftarrow 0$ for $i = 1 \ldots n$;

if $Mode$ = $Plain$ **then**

$M_r(i) \leftarrow 0$ for $r = 1 \ldots s$, i: $\sigma_r(i) \le 2^{j+1}$;

if $Mode$ = Ordered then

for $j \leftarrow 1$ **to** $\lceil \log_2 n \rceil$ **do**

$M_{r,j}(i) \leftarrow 0$ for $r = 1 \ldots s$, $i = 1..2^{j+1}$;

for $t \leftarrow 1$ **to** I **do**

T_t, $\{M_r\}_{r=1}^{s} \leftarrow$ BuildTree $(\{M_r\}_{r=1}^{s}, \{(x_i, y_i)\}_{i=1}^{n}, a, L, \{\sigma_i\}_{i=1}^{s}, Mode)$;

$leaf_0(i) \leftarrow GetLeaf(x_i, T_t, \sigma_0)$ for i = 1… n);

$grad_0 \leftarrow CalcGradient(L, M_0, y)$

foreach leaf j in T_t **do**

$b_j^t \leftarrow -\text{avg}(grad_0(i)$ for i : $leaf_0(i) = j)$;

$M_0(i) \leftarrow M_0(i) + a b_{leaf_0(i)}^t$ for $i = 1 \ldots n$;

return $F(x) = \sum_{t=1}^{I} \sum_j \alpha b_j^t 1_{\{GetLeaf(x, T_t, ApplyMode) = j\}}$

2.9 Bayesian optimization

Bayesian optimization is a powerful black-box optimization algorithm based on probability theory. The core idea is to guide the search process by building a probabilistic model of the objective function, thereby finding the parameter configuration that optimizes the objective function.

Algorithm S7. The pseudocode of Bayesian optimization.

Algorithm Bayesian optimization

for n = 1, 2, ... , **do**

select new x_{n+1} by optimizing acquisition function α

$$x_{n+1} = arg \max_x \alpha(x; D_n)$$

query objective function to obtain y_{n+1}

augment data $D_{n+1} = \{D_n, (x_{n+1}, y_{n+1})\}$

update statistical model

end for

3. Abbreviation

AI - Artificial Intelligence

AC - Acoustic Log (μs/m)

AHP - Analytic Hierarchy Process

BPP - Bivariate Predictive Patterns

BPNN - Back Propagation Neural Network

CAL - Caliper (cm)

CNL - Compensated Neutron Log (%)

CatBoost - Category Boosting

DEN - Density Log (g/cm^3)

DMML - Data Matching of Measured Data and Logging Data

DT - Decision Tree

KD - Kernel Density

KNN - K-Nearest Neighbors

LR - Linear Regression

LightGBM - Light Gradient Boosting Machine

ML - Machine Learning

PCA - Principal Component Analysis

PI - Permutation Importance

PI-Set Permutation Importance - Set

SP - Spontaneous Potential (mV)

SVR - Support Vector Regression

GR - Gamma Ray (API)

RF - Random Forest

RT - Resistivity Log (OMM)

RLLD - Deep Resistivity Log (OMM)

RLLS - Shallow Resistivity Log (OMM)

UPP – Univariate Predictive Patterns

Xu FM - Xujiahe Formation

XGBoost - eXtreme Gradient Boosting

4. Mathematical descriptions of evaluation metrics

4.1 R^2

R^2 signifies the proportion of the variance in the dependent variable that is predictable from the independent variable(s). A value closer to 1 indicates a model that can more accurately explain the observed data, thus providing insights into the model's adequacy in capturing information. However, R^2 is influenced by the sample size and the complexity of the model. Consequently, a high R^2 does not necessarily guarantee strong predictive power in some scenarios.

4.2 MSE

MSE calculates the average of the squared differences between predicted and actual values. Its function curve is smooth, continuous, and differentiable everywhere, facilitating the use of the gradient descent algorithm for optimization. As the error decreases, the gradient diminishes accordingly, aiding in convergence. However, MSE is highly sensitive to outliers, potentially exaggerating errors. When the actual values deviate significantly from the predicted ones, MSE imposes a substantial penalty, which could lead the model to overly concentrate on outliers at the detriment of the predictive accuracy for other, more typical data points.

4.3 RMSE

RMSE is the square root of the average of the squared differences between the predicted and actual values. It is straightforward to interpret, sensitive to outliers, and provides an intuitive measure of the average error between predicted and actual values. However, RMSE does not account for the variability of the target variables, which may not accurately represent the model' s performance. Its sensitivity to outliers can result in unstable evaluation outcomes.

4.4 MAE

MAE calculates the average of the absolute differences between predicted and actual values. It demonstrates robustness against outliers, simplicity in calculation, and clarity in interpretation. MAE assigns equal weight to all errors, ensuring stable evaluation outcomes. However, it fails to account for the variability of the target variable, potentially misrepresenting the model' s performance.

4.5 MAPE

MAPE measures the average percentage difference between predicted and actual values. MAPE quantifies errors in percentage terms, emphasizing relative errors, making it highly suitable for comparing predictions across datasets of varying scales. However, MAPE encounters limitations when actual values approach zero, as this leads to division by zero issues, rendering the metric inapplicable. Additionally, MAPE is particularly sensitive to minor errors, potentially exaggerating the magnitude of errors in cases with smaller actual values. The closer the values of MSE, RMSE, MAE, and MAPE are to 0, the superior the model's performance is considered to be.