+ Supporting Information

Machine learning-aided catalysts screening and multi-objective optimization for the indirect CO_2 hydrogenation to methanol and ethylene glycol process

Qingchun Yang[a, b*], Yingjie Fan [a], Jianlong Zhou [a], Lei Zhao [a], Yichun Dong [a], Jianhua Yu[c], Dawei Zhang [a**]

[a] School of Chemistry and Chemical Engineering, Hefei University of Technology, Hefei, PR China, 230009

[b] Anhui HaoYuan Chemical Group Co., Ltd., Fuyang, PR China, 236023

[c] State Grid Anhui Electric Power Co., Ltd., Guangde Power Supply Company, Guangde, PR China, 242200

+ For publication in *Green Chemistry*

***Corresponding author:**

Dr. Qingchun Yang

Email: ceqcyang@hfut.edu.cn

****Corresponding author:**

Prof. Dawei Zhang

Email: zhangdw@utsc.edu.cn

Total number of pages: 17 (S1-S17)

Total number of figures: 7 (Figures S1-S7)

Total number of tables: 6 (Tables S1-S6)

Appendix A Supplementary material of the machine learning models

A.1 Random forest regression algorithm

Random forest algorithm, one of the representative Bagging integration algorithms, further introduces random feature selection in each round of decision tree training process by taking decision tree as the base learner. The random forest regression model

The algorithm principle of the random forest algorithm is as follows:

(1) The S sample points are randomly selected from the training sample set M to obtain a series of new M_1, ..., M_s sub training sets.

(2) Each regression tree is trained through the sub training set. The segmentation rule of each node of each tree is to select k features randomly from all the features. Then the optimal segmentation point (j,s) is selected from the k features according to the minimization of square error $L(j,s)_{\min}$, and divided in to left sub tress ($R_1(j,s)$) and right sub trees ($R_2(j,s)$), as shown in Eqs. (S1) - (S4).

$$L(j,s)=\sum_{x_i\in R_1(j,s)}(y_i-\hat{c}_1)^2+\sum_{x_i\in R_2(j,s)}(y_i-\hat{c}_2)^2 \tag{S1}$$

$$R_1(j,s)=\{x\,|\,x^{(j)}\le s\} \tag{S2}$$

$$R_2(j,s)=\{x\,|\,x^{(j)}> s\} \tag{S3}$$

$$\hat{C}_m=\frac{1}{N_m}\sum_{x_i\in R_m(j,s)}y_i,\ x\in R_m,\ m=1,2 \tag{S4}$$

Where, j represents the optimal partition feature, s denotes the optimal partition sample point under the optimal partition feature, x_i is the sample point, y_i is the output value corresponding to the sample point, $\hat{C}_m$ stands for the output value of the subtree, and N_m refers to the total number of features contained in the subtree.

(3) The prediction result of each regression tree is the mean value of leaf nodes reached by the sample point, as shown in Eqs. (S5) and (S6).

(4) The final prediction result of random forest is the unweighted mean value of all

regression tree prediction results, as shown in Eq. (S7).

$$h_s(x) = \sum_{m=1}^{M} \hat{c}_m I(x \in R_m) \tag{S5}$$

$$I = \begin{cases} 1 & if\,(x \in R_m) \\ 0 & if\,(x \notin R_m) \end{cases} \tag{S6}$$

$$\overline{h}(x) = \frac{1}{S} \sum_{s=1}^{S} h_s(x) \tag{S7}$$

Where *I* is the indicator function, $h_s(x)$ refers to the predicted result of each tree, $\overline{h}(x)$ stands for the final prediction result of random forest, and S denotes the number of regression trees.

A.2 Support vector regression algorithm

For a linear regression, the objective is to fit a regression line ($\hat{y} = W^T x + b$) for the data to minimize the error caused by the deviation. The process usually uses the least square method to determine the vector *W* and offset term *b*. The support vector regression model will set a threshold (ε, function interval) around the regression line. The point within the threshold will not be punished due to its error (no loss will be calculated). Only the support vector will have an impact on its functional model. Such a threshold area is commonly known as the ε-pipeline.

SVR maps the original space of the input data to a higher dimensional feature space through the kernel function, making the problem of nonlinear regression become a problem of approximate linear regression after the transformation of the kernel function. The common kernel functions of the SVR model are shown in Table S1.

In the feature space, to determine the optimal linear plane of fitting data, the optimized SVR model is obtained by minimizing the total loss and maximizing the pipe width ($\frac{\varepsilon}{\|W\|}$), as shown in Eq. (S8). Since the error of the predicted value of each training data is at most equal to ε, the smoothest function ($f(x)$) is found by

minimizing the vector norm ($\|W\|^2$). The objective function and constraints are given in Eqs. (S9) and (S10).

The smoothest function of SVR model: $f(x) = W^T \phi(x) + b$ (S8)

The objective function of SVR model: $\min_{W,b} \frac{1}{2}\|W\|^2$ (S9)

The constraints of SVR model: $\|y_i - [W\phi(x_i) + b]\| \le \varepsilon$ (S10)

where W is the weight, b is the deviation, $\phi(x)$ represents the mapping of feature x from low to high dimensions.

A.3 Neural network algorithm

Neural network algorithm is an information processing system that simulates the neural results of human brain to realize the intelligent activities of human brain. Its structure mainly includes input layer, hidden layer and output layer, and each layer is connected by the most basic element neuron. In the artificial neural network, each neuron receives the input signals (x_0, x_1, ..., x_n) transmitted from the upper layer of neurons and superposes with the weight (w_i). It is then to set a threshold (b) to ensure that the output value calculated through the input cannot be randomly activated, and finally output them through the activation function.

For a neuron: $y = f_{act}(\sum_{i=1}^{n} w_i x_i + b)$ (S11)

Where f_{act} is the activation function; x_i denotes the input of a neuron at the upper level; and y represents the output of the neuron.

From each layer of the neural network, the result is composed of the output of all neurons in this layer, as shown in Eq. (S12).

$h_{out}^{(i)} = f_{act}^{(i)}(h_{in}^{(i)} w^{(i)} + b^{(i)})$ (S12)

Where h_{in} is the output matrix of the upper layer, and its dimension is [N, D_{in}]; h_{out} is the output matrix of this layer whose dimension is [N, D_{out}]; $w^{(i)}$ is the weight matrix whose dimension is [D_{in}, D_{out}]; $b^{(i)}$ is the offset matrix whose dimension is [1, D_{out}];

$f_{act}^{(i)}$ denotes the activation function; N is the number of samples in the matrix; D_{in} represents the input sample dimension, and D_{out} represents the output sample dimension.

This paper introduces a back-propagation algorithm, which is a supervised learning method. It is constantly updated w and b through "learning from mistakes" to minimize the difference (loss function) between the final model output and the true value. Here, the gradient descent algorithm is the most commonly used optimization method for updating w and b. Then, a trained neural network is obtained by repeating the above process for the whole training set. Finally, to output the test set, it can get the prediction value through forward propagation to evaluate the prediction results of the trained neural network on the unknown data set.

Appendix B Supplementary figures and tables

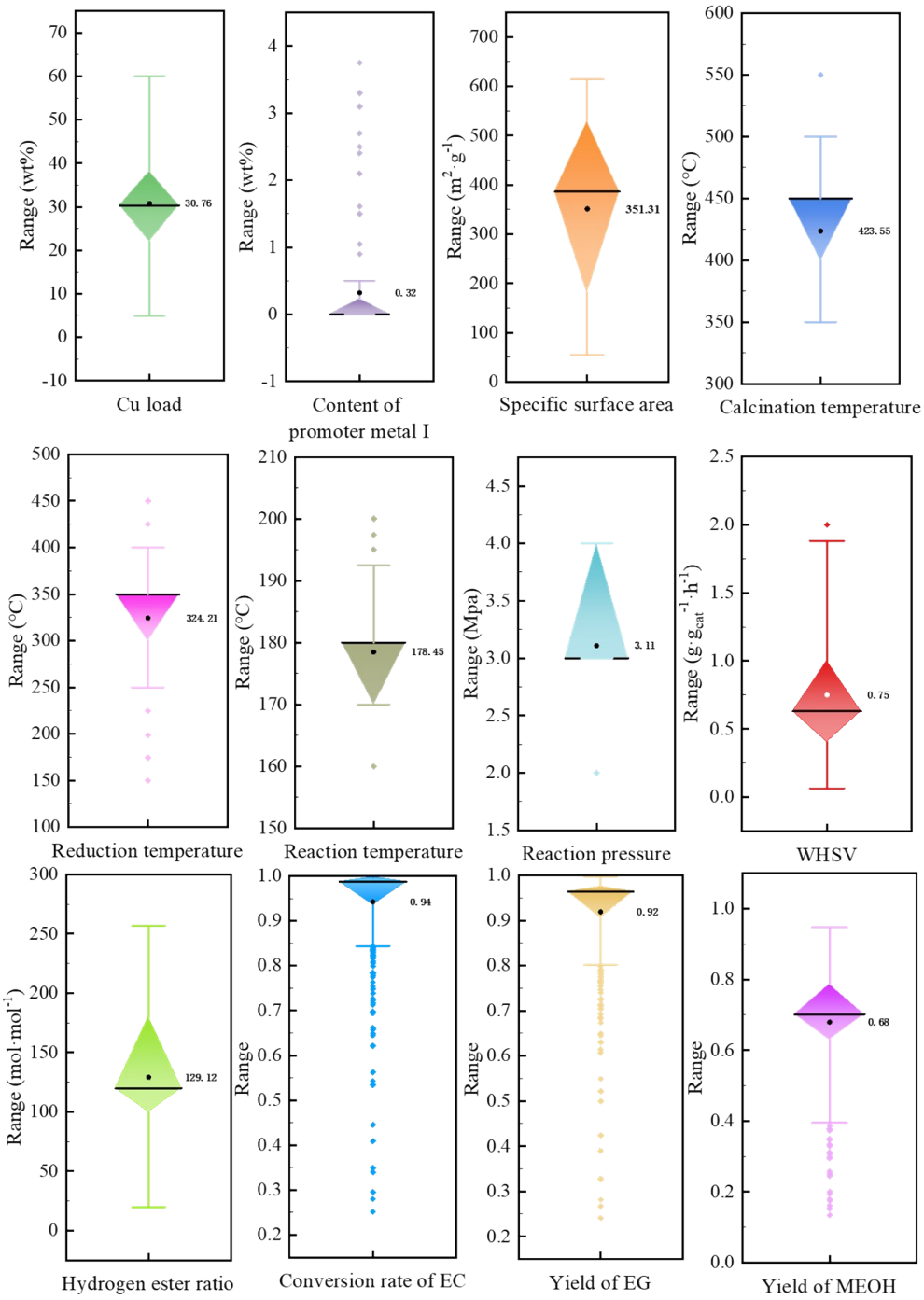

Fig. S1. Data frequency(distribution) analysis of the collected indirect CO_2 hydrogenation catalysts

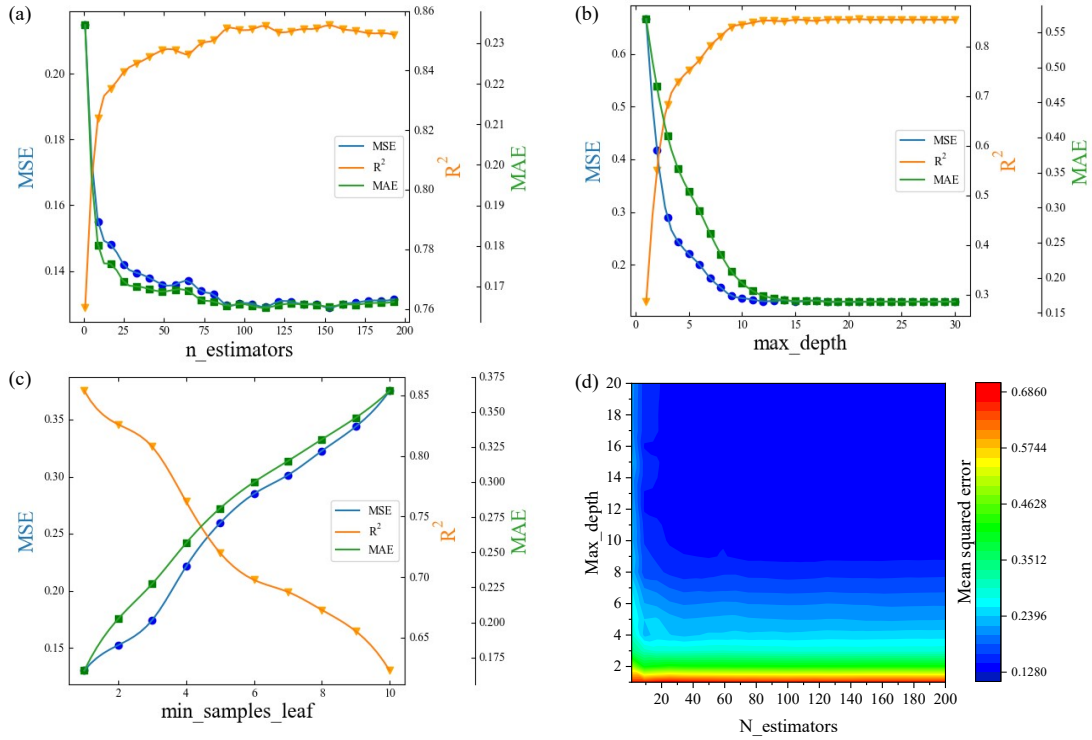

Fig. S2. Optimization of the hyperparameters of the RFR model under 5-fold

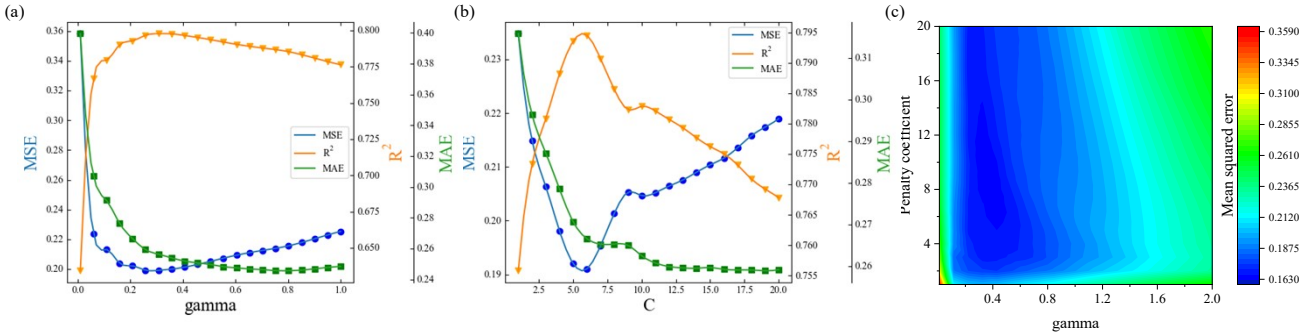

Fig. S3. Optimization of the hyperparameters of the SVR model under 5-fold cross-validation

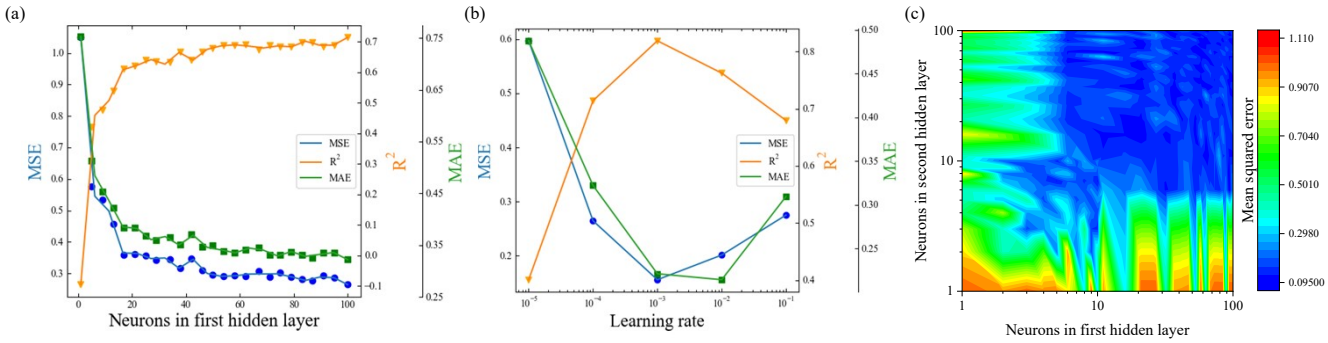

Fig. S4. Optimization of the hyperparameters of the NN model under 5-fold cross-validation

Appendix C Code of the PCA, SHAP, and GA optimization methods

Appendix C.1 Code of the PCA method

```
# data standardization
data = df.values.astype(np.float32)
data = shuffle(data, random_state=2022)
scaler = StandardScaler()
data = scaler.fit_transform(data)

# Mean and variance of data
data_mean = np.mean(data, axis = 0)
data_std = np.std(data, axis = 0)

# covariance matrix
cov_mat = (data - data_mean).T.dot((data - data_mean))/(data.shape[0]-1)

# PCA
ew, ev = np.linalg.eig(cov_mat)  #ew——eigenvalue，ev——Eigenvector
ew_order = np.argsort(ew)[::-1] #Arrange from large to small
ew_sort = ew[ew_order]
ev_sort = ev[:, ew_order]

# Each principal component contribution
gx = ew_sort/np.sum(ew_sort)

# Accumulated contribution
lgx = np.cumsum(gx)

# Draw principal component variance diagram
x1 = np.linspace(1, 26, 26)
x2 = x1 +0.5
y1 = gx
y2 = lgx
plt.bar(x=x1, height=y1, width=0.9, label="Individual explained variance", color="teal", alpha=0.8)
plt.step(x2, y2, label="Cumulative explained variance", color="teal", alpha=0.8)

# Set x-axis scale
xticks(np.linspace(1, 26, 26, endpoint=True))

# Set Title
# plt.title("Principal component variance")

# Set coordinate axis
plt.xticks(fontsize=12)
```

Fig. S5. Key code of the PCA method

Appendix C.2 Code of the SHAP method

```
# SHAP Value Interpretation Machine Learning Model
x_train = pd.DataFrame(x_train)
x_train.columns=['Catalyst','Cu (wt.%)','Promoter Ⅰ','Promoter Ⅰ(wt.%)','Promoter Ⅱ','Promoter Ⅱ(wt.%)','Support','Prepa
x_test = pd.DataFrame(x_test)
x_test.columns=['Catalyst','Cu (wt.%)','Promoter Ⅰ','Promoter Ⅰ(wt.%)','Promoter Ⅱ','Promoter Ⅱ(wt.%)','Support','Prepar
explainer = shap.KernelExplainer(regr.predict, x_train)
shap_values = explainer.shap_values(x_test)# 传入特征矩阵，计算SHAP值

# SHAP value of EC conversion
shap_values_EC = shap_values[0]
shap_sum_EC = np.sum(abs(shap_values_EC), axis=0)
shap_sum_EC = pd.DataFrame(shap_sum_EC)
shap_sum_EC.index = ['Catalyst type','Cu load','Promoter Ⅰ','Content of\npromoter metal Ⅰ','Promoter Ⅱ','Content of\np

# SHAP value of EG yield
shap_values_EG = shap_values[1]
shap_sum_EG = np.sum(abs(shap_values_EG), axis=0)
shap_sum_EG = pd.DataFrame(shap_sum_EG)
shap_sum_EG.index = ['Catalyst type','Cu load','Promoter Ⅰ','Content of\npromoter metal Ⅰ','Promoter Ⅱ','Content of\np

# SHAP value of MEOH yield
shap_values_MEOH = shap_values[2]
shap_sum_MEOH = np.sum(abs(shap_values_MEOH), axis=0)
shap_sum_MEOH = pd.DataFrame(shap_sum_MEOH)
shap_sum_MEOH.index = ['Catalyst type','Cu load','Promoter Ⅰ','Content of\npromoter metal Ⅰ','Promoter Ⅱ','Content of\

# EC feature importance ranking
shap.summary_plot([shap_values_EC,shap_values_EG,shap_values_MEOH], x_test, plot_type="bar", class_inds=[0,1,2], class_n
font = {'family':'Times New Roman', 'weight':'normal', 'size':14}
plt.xlabel("mean(|Shap value|)(average impact on model output magnitude)", fontdict=font)
plt.title("Importance ranking of input features in machine learning model", fontsize=16)
plt.legend()
plt.figure(figsize=(16,16))
plt.show()

shap.summary_plot(shap_values_EC, x_test, plot_type="bar", color='darkcyan', show=False)
font = {'family':'Times New Roman', 'weight':'normal', 'size':16}
plt.xlabel("mean(|Shap value|)(average impact on model output magnitude)", fontdict=font)
plt.title("The Importance Ranking of Input Features of\nMachine Learning Model for EC Conversion", fontsize=18)
plt.figure(figsize=(16,16))
```

Fig. S6. Key code of the SHAP interpreter

How to explain the built machine learning model by importing the corresponding SHAP package is detailed in: https: //Welcome to the SHAP documentation — SHAP latest documentation.

Appendix C.3 Code of the GA optimization method

```python
class MyProblem(ea.Problem):
    def __init__(self):
        name = 'MyProblem'
        M = 1  # Initialize M (target dimension)
        maxormins = [-1]  # Initialize maxormins (target minimizing and maximizing tag list, 1: minimize target; -1: max
        Dim = 16  # Initialize Dim (decision variable dimension)
        varTypes = [1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1]  # Initialize varTypes (the type of decision variabl
        lb = [17, 40, 6, 0.12, 0, 0, 4, 1, 518, 350, 350, 180, 3, 1, 45, 0]  # Lower bound of decision variables
        ub = [17, 41.9, 6, 0.48, 0, 0, 4, 1, 578, 350, 350, 180, 3, 1, 180, 0]  # Upper bound of decision variables
        lbin = [1] * Dim  # The lower boundary of the decision variable, where 1 represents the boundary of the inclusio
        ubin = [1] * Dim  # The upper boundary of the decision variable, where 1 represents the boundary of the inclusio
        # Instantiate
        ea.Problem.__init__(self, name, M, maxormins, Dim, varTypes, lb, ub, lbin, ubin)

    def aimFunc(self, pop):  # objective function
        Vars = pop.Phen  # Obtain the decision variable matrix
        #         print(pop.Phen.size)
        y_Phen = np.zeros((40, 3))
        pop.Phen1 = np.hstack([pop.Phen[:, 0:16], y_Phen])
        pop.Phen1 = pop.Phen1.astype(np.float32)
        pop.Phen1 = scaler.transform(pop.Phen1)
        p_Phen = regr.predict(pop.Phen1[:, 0:16])
        p_data = np.hstack([pop.Phen1[:, 0:16], p_Phen])
        p_ks = p_data[:, 13].reshape(-1, 1)
        p_qzb = p_data[:, 14].reshape(-1, 1)
        p_EG = p_data[:, 17].reshape(-1, 1)
        p_MEOH = p_data[:, 18].reshape(-1, 1)
        p_data = scaler.inverse_transform(p_data)
        EC_p = p_data[:, 16].reshape(-1, 1)
        EG_p = p_data[:, 17].reshape(-1, 1)
        MEOH_p = p_data[:, 18].reshape(-1, 1)
        pop.CV = np.hstack([EC_p - 1, EG_p - 1, MEOH_p - 1])
        pop.ObjV = 0.25 * p_MEOH + 0.25 * p_EG + 0.25 * p_ks - 0.25 * p_qzb  # Calculate the objective function value an

# Instantiate the problem object
problem = MyProblem()

# Population settings
Encoding = 'RI'# Encoding method
NIND = 40# population size
Field = ea.crtfld(Encoding, problem.varTypes, problem.ranges, problem.borders)
population = ea.Population(Encoding, Field, NIND) # Instantiating a population object (at this point, the population has

# Constructive Algorithms
myAlgorithm = ea.soea_DE_rand_1_L_templet(problem, population)# Instantiating an Algorithm Template Object
myAlgorithm.MAXGEN = 1000# Maximum Generation
myAlgorithm.mutOper.F = 0.5# Parameter F in Differential Evolution
myAlgorithm.recOper.XOVR = 0.7# Set crossover probability
myAlgorithm.logTras = 1# Set how many generations to record logs every, if set to 0, it means no logs will be recorded
myAlgorithm.verbose = True# Set whether to print out log information
myAlgorithm.drawing = 1# Set drawing method (0: No drawing; 1: Draw result graph; 2: Animate target space process; 3: A

# Calling algorithm templates for population evolution
[BestIndi, population] = myAlgorithm.run() # Execute the algorithm template to obtain the optimal individual and the las

# Output Results
print('Number of evaluations: %s' % myAlgorithm.evalsNum)
print('Time spent %s 秒' % myAlgorithm.passTime)
if BestIndi.sizes != 0:
    print('The optimal objective function value is: %s' % BestIndi.ObjV[0][0])
    print('Optimal control variable value为: ')
    for i in range(BestIndi.Phen.shape[1]):
        print(BestIndi.Phen[0, i])
else:
    print('No feasible solution was found this time.')
```

Fig. S7. Key code of the GA optimization method

Using the Geatpy genetic algorithm template to optimize and screen catalyst parameters is detailed in: https: //Geatpy – The Genetic and Evolutionary Algorithm Toolbox for Python with High Performance.

Table S1 Common kernel functions and parameters of the SVR model

Function	Expression	Parameters
Linear Kernel	$k(x_i, x_j) = (x_i, x_j)$	/
Polynomial Kernel	$k(x_i, x_j) = ((x_i, x_j) + 1)^d$	$d \geq 1$
Gaussian Kernel (RBF)	$k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$	$\sigma > 0$
Laplacian Kernel	$k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|}{\sigma})$	$\sigma > 0$
Sigmoid Kernel	$k(x_i, x_j) = \tanh(\lambda(x_i, x_j) + \theta$	$\lambda > 0,\ \theta < 0$

Table S2 Main information of the original dataset

Features	Type	Item	Unit
Input	Catalyst Descriptor	Catalyst type	/
		Cu load	wt%
		Content of promoter metal I	wt%
		Content of promoter metal II	wt%
	Preparation conditions	Calcination temperature	℃
		reduction temperature	℃
	Operating conditions	Reaction temperature	℃
		Reaction pressure	MPa
		Weight hourly space velocity	$g \cdot g_{cat}^{-1} \cdot h^{-1}$
	Feeding conditions	Hydrogen ester ration	$mol \cdot mol^{-1}$
		Solvent type	/
Output	Conversion	Conversion rate of EC	/
	Yield	Yield of EG	/
		Yield of MEOH	/

Table S3 Main information of the dataset improved by PCA

Features	Type	Item	Unit
Input	Catalyst Descriptor	Catalyst type	/
		Cu load	wt%
		Atomic weight of promoter metal I	/
		Radius of promoter atomic (ion) I	/
		Electronegativity of promoter metal I	/
		Status of promoter metal I	/
		Content of promoter metal I	wt%
		Atomic weight of promoter metal II	/
		Radius of promoter atomic (ion) II	/
		Electronegativity of promoter metal II	/
		Status of promoter metal II	/
		Content of promoter metal II	wt%
		Support type	/
		Preparation method	/
		Specific surface area	m^2/g
		Pore volume	cm^3/g
	Preparation conditions	Calcination temperature	℃
		reduction temperature	℃
	Operating conditions	Reaction temperature	℃
		Reaction pressure	MPa
		Weight hourly space velocity	$g \cdot g_{cat}^{-1} \cdot h^{-1}$
	Feeding conditions	Hydrogen ester ration	$mol \cdot mol^{-1}$
		Solvent type	/
Output	Conversion	Conversion rate of EC	/
	Yield	Yield of EG	/
		Yield of MEOH	/

Table S4 Definition of the optimized hyperparameters of the established ML models

ML model	Hyperparameters	Definition
RFR	n_estimators	Number of decision trees
	max_features	Maximum depth of tree
	min_samples_leaf	Minimum samples contained in leaf nodes
SVR	kernel	Functions that map samples to higher dimensions
	gamma	Coefficients in kernel functions
	C	Penalty coefficient (regularization coefficient)
NN	Neuron	Basic Unit of neural network
	learning rate	Tuning parameters to determine the step size in each iteration

Table S5 Evaluation of the SVR model with different kernel functions by 5-fold cross-validated average MSE, R^2, and MAE

Kernel function	MSE	R^2	MAE
Linear Kernel	0.4493	0.5280	0.4371
Polynomial Kernel	0.2827	0.7037	0.3387
Gaussian Kernel (RBF)	0.2347	0.7559	0.3158
Sigmoid Kernel	1.0355	-0.076	0.7017

Table S6 Supplementary results of optimized parameters of various types of catalysts

Catalysts	Promoter II	Content of promoter metal II	Support type	Preparation method	Specific surface area
Cu/SiO_2	None	0	SiO_2	AE	489
Cu/HMS	None	0	HMS	AE	225.49
Cu/SiO_2-S	None	0	SiO_2-S	AE	55
Cu-C/SiO_2-R	None	0	SiO_2-C	AE	201.3
Cu-MgO/SBA-15	None	0	SBA-15	AE	429.17
Ni-Cu/SiO_2	None	0	SiO_2	AE	613.53
Cu/SiO_2-F	None	0	SiO_2-F-3.75	OSHP	112
Cu_8-Mg_1-Zr_z/SiO_2	Zr	4.2	SiO_2	DP	82
Cu_x-Mg_1/SiO_2	None	0	SiO_2	DP	56
S-1-210@$CuSiO_3$	None	0	S-1-210	OSHT	363
xMo-Cu/SiO_2	None	0	SiO_2	AE-IWI	448
$xMoO_x$-Cu/SiO_2	None	0	SiO_2	OPMHT	413
xPt-Cu/SiO_2	None	0	SiO_2	AE-IWI	578