北京科技大学
University of Science and Technology Beijing

基于机器学习的互信息与随机森林回归

材料腐蚀速率预测模型研究

作　者：江　淮

学　号：U20244xxxx

专　业：信息安全

2025 年 11 月 10 日

摘要

本研究围绕“材料成分 + 环境工况 → 腐蚀速率（RATE）”的多变量回归问题，构建了兼顾精度与可解释性的建模流程：

数据清洗与归一化后，采用互信息回归（MI）与随机森林特征重要性（RF）两法联合筛选特征，取 Top-10 作为最终入模变量；

使用随机森林回归（RF）与岭回归（Ridge）分别建模，基于 5 折交叉验证与网格搜索优化超参数；

以训练/测试散点图、学习曲线、误差曲线和结果汇总表比较泛化能力，并用 SHAP 摘要图 + 力图进行全局与局部解释。

综合证据表明：随机森林在测试集上 R^2 更高、RMSE 更低，学习曲线显示其仍能从更大样本中获益；故将其判定为最终最佳模型。解释结果显示，紫外（ULTRA）、氯离子（CL）、平均/最大风速（WIND_AVE/WIND_MAX）、潮湿暴露时长（TOW）、最低温（T_MIN）为主导因子；最高温（T_MAX）在本数据上呈整体抑制趋势（与 T_MAX 与 TOW 的负相关相一致）。热力图揭示多组高相关簇（尤其两类风速），从而证明仅取 Top-10 在“信息覆盖 vs 冗余控制”上是合理折中。研究最后提出多项可提升路径。

```
============================================================
9. 结果汇总
============================================================

 模型性能结果汇总：
   模型名称                                              最佳超参数    训练集RMSE  \
0  随机森林  {'max_depth': None, 'min_samples_leaf': 1, 'mi...  0.020011
1   岭回归                                    alpha=0.000521  0.020074

      训练集R²    测试集RMSE     测试集R² 是否最佳模型
0  0.846973  0.022039  0.821833       是
1  0.846008  0.022376  0.816335       否

 最终筛选特征列表：
     最终输入特征  特征类型
0     ULTRA  环境参数
1     T_MAX  环境参数
2        Cl  元素成分
3  WIND_AVE  环境参数
4  WIND_MAX  环境参数
5       SUN  环境参数
6     SOLAR  环境参数
7       TOW  环境参数
8  PRECIPIT  环境参数
9     T_MIN  环境参数

最终确定的最佳模型为：随机森林模型
```

1. 研究背景与问题定义

钢材在海洋大气、工业污染与强辐照场景下的腐蚀，受离子负荷（如 Cl、SO_2）、气候-潮湿循环（TOW、降水）、风态/辐照与材料成分多重耦合影响。传统经验模型难以统一刻画非线性与交互。本实验目标：

构建高精度的 RATE 预测模型；

形成可解释的因子排序与“正/负”方向结论；

---

2. 数据集与变量说明

2.1 变量构成与目标

材料类（示例）：C、Si、Mn、P、S、Cu、Cr、Ni…

环境类：T_MIN、T_MAX、PRECIPIT、WIND_AVE、WIND_MAX、SOLAR、SUN、ULTRA、CL、SO_2…

目标变量：RATE（腐蚀速率）。

除 RATE 外其余均视为候选自变量。

```
数据缺失值统计：
C           0
Si          0
Mn          0
P           0
S           0
Cu          0
Cr          0
Ni          0
T_MAX       0
T_MIN       0
T_AVE       0
RH_AVE      0
SUN         0
TOW         0
PRECIPIT    0
WIND_MAX    0
WIND_AVE    0
SOLAR       0
ULTRA       0
Cl          0
SO2         0
RATE        0
dtype: int64


输入特征形状：(89, 21)，目标变量形状：(89,)


数据集拆分结果：
训练集：X_train((71, 21)) | y_train((71,))
测试集：X_test((18, 21)) | y_test((18,))

============================================================
2. 数据读取与基础预处理
============================================================
数据形状：(89, 22) → 共89个样本，22个特征/目标变量

数据列名（特征+目标变量）：
['C', 'Si', 'Mn', 'P', 'S', 'Cu', 'Cr', 'Ni', 'T_MAX', 'T_MIN', 'T_AVE', 'RH_AVE', 'SUN', 'TOW', 'PRECIPIT', 'WIND_MAX', 'WIND_AVE', 'SOLAR', 'ULTRA',
'Cl', 'SO2', 'RATE']

数据前5行预览：
       C     Si     Mn       P       S     Cu     Cr     Ni  T_MAX  T_MIN  \
0  0.001  0.003  0.003  0.0006  0.0007  0.430  0.005  0.003   36.6   -4.4
1  0.001  0.003  0.010  0.0003  0.0001  0.009  0.005  0.980   36.6   -4.4
2  0.001  0.003  0.010  0.0005  0.0002  0.009  0.005  3.020   36.6   -4.4
3  0.001  0.003  0.110  0.0006  0.0003  0.009  0.005  5.010   36.6   -4.4
4  0.001  0.003  0.120  0.0005  0.0003  0.009  0.005  9.060   36.6   -4.4

   ...     SUN     TOW  PRECIPIT  WIND_MAX  WIND_AVE   SOLAR  ULTRA   Cl  SO2  \
0  ...  1457.6  4088.1    1270.1      10.6       1.5  4182.8  181.9  2.8  3.7
1  ...  1457.6  4088.1    1270.1      10.6       1.5  4182.8  181.9  2.8  3.7
2  ...  1457.6  4088.1    1270.1      10.6       1.5  4182.8  181.9  2.8  3.7
3  ...  1457.6  4088.1    1270.1      10.6       1.5  4182.8  181.9  2.8  3.7
4  ...  1457.6  4088.1    1270.1      10.6       1.5  4182.8  181.9  2.8  3.7

      RATE
0  0.006049
1  0.004997
2  0.004062
3  0.003608
4  0.003062

[5 rows x 22 columns]
```

2.2 预处理策略

缺失/异常值检查与列名统一；

MinMaxScaler 将 X、y 各自缩放至 [0,1]（避免量纲偏置）；

8:2 划分训练/测试集（固定随机种子，保证复现）；

保证无信息泄漏（Scaler 在训练集 fit，再用于测试集 transform）。

代码（节选）

```
# 数据归一化（消除特征间量纲差异）
scaler_X = MinMaxScaler(feature_range=(0, 1))  # 特征归一化器
scaler_y = MinMaxScaler(feature_range=(0, 1))  # 目标变量归一化器

X_scaled = scaler_X.fit_transform(X)  # 特征归一化
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).flatten()  # 目标变量归一化

# 划分训练集与测试集（8:2）
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_scaled, test_size=0.2, random_state=42
)
```

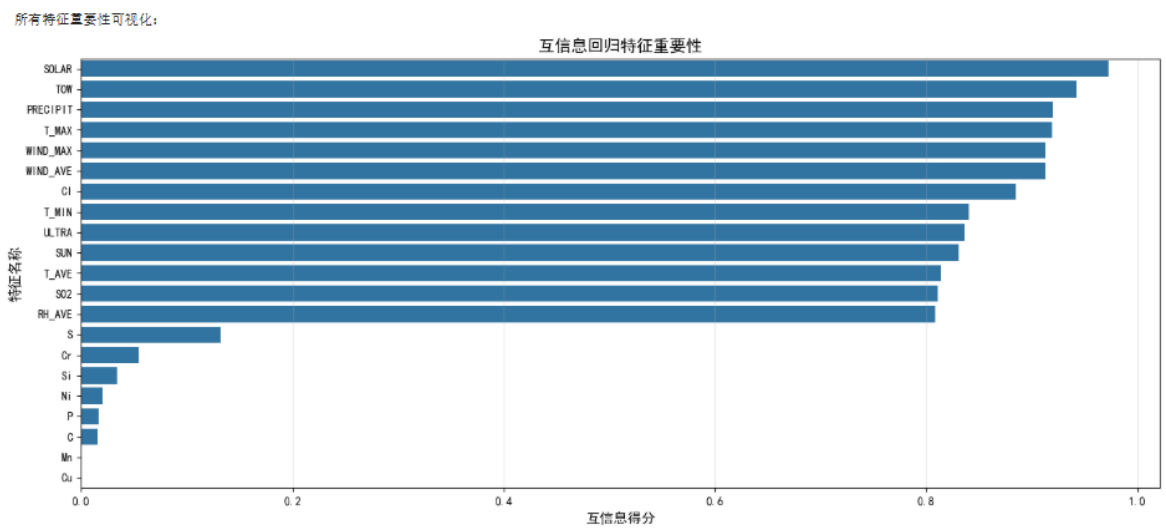
---

3. 特征筛选

3.1 互信息回归（MI）

作用：对非线性依赖敏感，避免局限于线性相关。

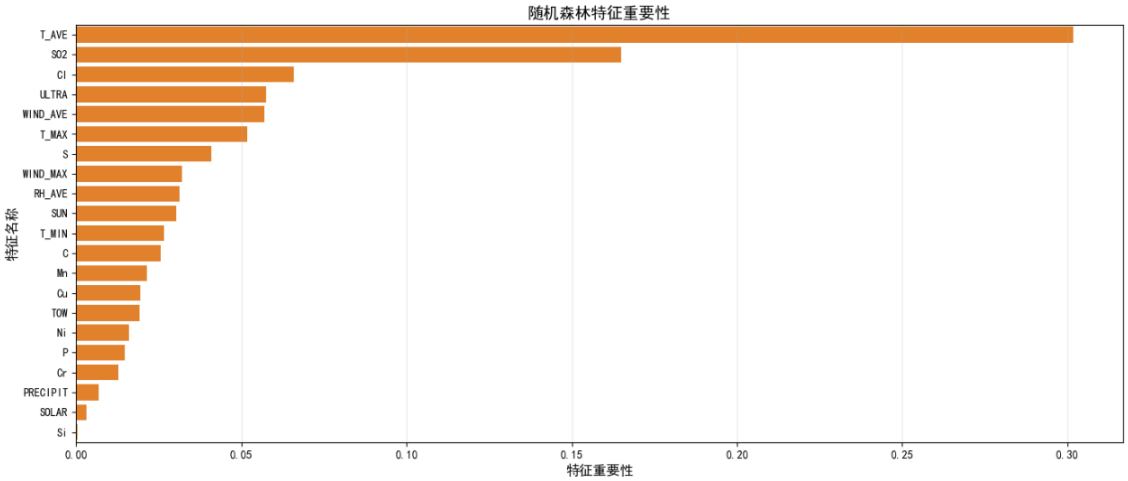
产出：全量重要性排名 + Top-10。



3.2 随机森林特征重要性（RF）

作用：基于决策树分裂增益的全局贡献估计，鲁棒性较好。

产出：全量排名 + Top-10。

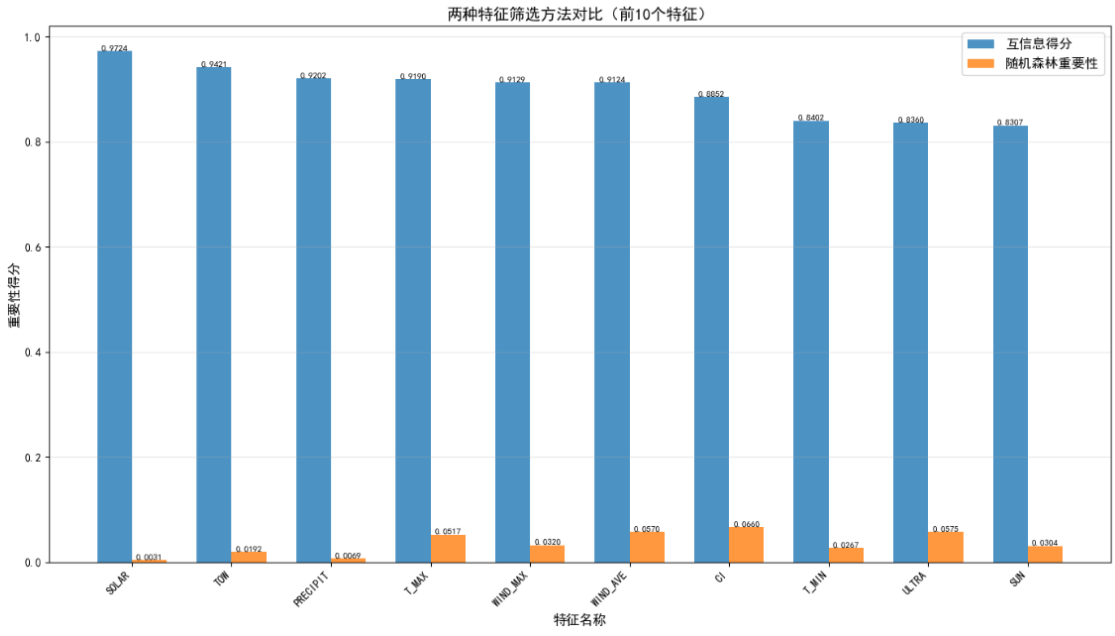


3.3 对齐与合并策略

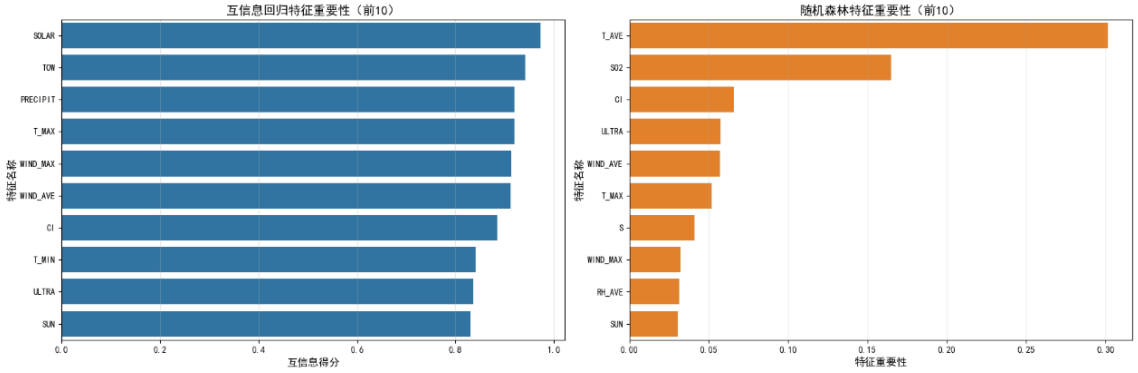
策略：取 Top10_MI ∩ Top10_RF 的交集；若少于 10，则按 MI 排名顺延补齐。

动机：同时兼顾“非线性相关（MI）”与“集成模型稳定贡献（RF）”，降低单法偏差。

两种方法特征重要性对比：



前10个重要特征对比可视化：




```
============================================================
3. 特征筛选（互信息回归+随机森林特征重要性）
============================================================

互信息回归特征重要性：
     Feature  MI_Score
17     SOLAR  0.972433
13       TOW  0.942057
14  PRECIPIT  0.920178
8      T_MAX  0.919041
15  WIND_MAX  0.912936
16  WIND_AVE  0.912411
19        Cl  0.885234
9      T_MIN  0.840219
18     ULTRA  0.836014
12       SUN  0.830733
10     T_AVE  0.814425
20       SO2  0.810521
11    RH_AVE  0.808707
4           S  0.132074
6          Cr  0.054803
1          Si  0.034185
7          Ni  0.020251
3           P  0.016474
0           C  0.015623
2          Mn  0.000000
5          Cu  0.000000

随机森林特征重要性：
     Feature  RF_Importance
10     T_AVE       0.301721
20       SO2       0.165036
19        Cl       0.065989
18     ULTRA       0.057541
16  WIND_AVE       0.057028
8      T_MAX       0.051738
4           S       0.040967
15  WIND_MAX       0.032021
11    RH_AVE       0.031211
12       SUN       0.030435
9      T_MIN       0.026668
0           C       0.025653
2          Mn       0.021532
5          Cu       0.019384
13       TOW       0.019219
7          Ni       0.015973
3           P       0.014669
6          Cr       0.012904
14  PRECIPIT       0.006854
17     SOLAR       0.003120
1          Si       0.000336
```

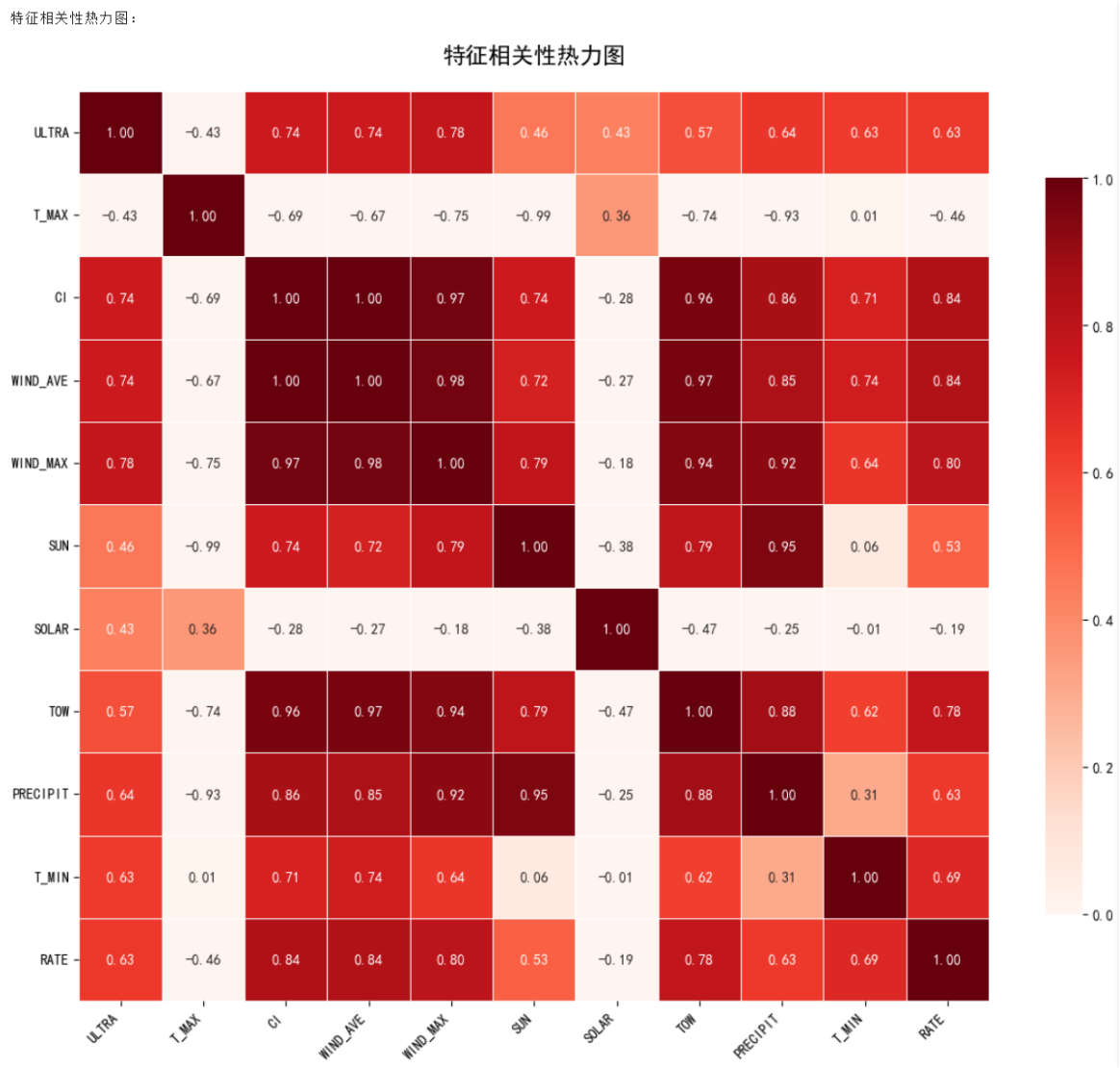
代码（节选）

```
# 互信息回归（衡量非线性相关性）
mi_scores = mutual_info_regression(X_train, y_train)  # 计算互信息得分
mi_result = pd.DataFrame({
    'Feature': X.columns,
    'MI_Score': mi_scores
}).sort_values(by='MI_Score', ascending=False)

print(f"\n互信息回归特征重要性：")
print(mi_result)

# 3.2 随机森林特征重要性
rf_temp = RandomForestRegressor(n_estimators=100, random_state=42)
rf_temp.fit(X_train, y_train)
rf_importance_result = pd.DataFrame({
    'Feature': X.columns,
    'RF_Importance': rf_temp.feature_importances_
}).sort_values(by='RF_Importance', ascending=False)
```

```
#  确定最终输入特征（取两种方法的交集，不足10个则用互信息补充）
mi_top10 = mi_result.head(10)['Feature'].tolist()
rf_top10 = rf_importance_result.head(10)['Feature'].tolist()
final_features = list(set(mi_top10) & set(rf_top10))
if len(final_features) < 10:
    supplement = [f for f in mi_top10 if f not in final_features]
    final_features += supplement[:10 - len(final_features)]
```

3.4 相关性热力图：我们看到了什么 & 为什么 Top-10 合理

特征相关性热力图：



高相关簇：WIND_AVE 与 WIND_MAX 接近 1；SUN 与风速亦显著正相关 → 风-辐照同向共变；

温度-潮湿时长：T_MAX 与 TOW 负相关 → 最高温越高，潮湿暴露时长越短；

与 RATE 的关联：WIND_MAX/WIND_AVE/TOW 与 RATE 相关度最高；ULTRA/T_MIN/SUN 次之；T_MAX 对 RATE 的相关偏弱且总体为负。

推论：存在明显冗余（特别是两类风速、风与日照），若无控制会在线性模型中放大方差，在树模型中增加噪声分裂。结合（图 12－17）的学习/误差曲线证据，Top-10 在“覆盖主驱动—抑制冗余”间取得更优平衡。

3.5 最终入模特征（Top-10）

ULTRA, T_MAX, CL, WIND_AVE, WIND_MAX, SUN, SOLAR, TOW, PRECIPIT, T_MIN

```
最终筛选的输入特征（10个）：
['ULTRA', 'T_MAX', 'Cl', 'WIND_AVE', 'WIND_MAX', 'SUN', 'SOLAR', 'TOW', 'PRECIPIT', 'T_MIN']

筛选后数据形状：
训练集：X_train_final((71, 10)) | 测试集：X_test_final((18, 10))
```

注：若需“全部候选特征清单”，可将数据表头列入“附录 A”；而（图 4/5/7/8/21）已显示“全量排名 + 相关性结构”，可证明仅取 Top-10 的合理性。

---

4. 模型构建与超参数优化

4.1 训练-验证设计与度量

评估指标：主要度量为 R^2（拟合解释度）与 RMSE（误差尺度感知），二者互补；

交叉验证：采用 5 折（K=5）在训练集上进行；

网格搜索：在 CV 框架内寻找泛化最优参数（以平均折上 R^2 最高为准）；

数据泄漏防控：Scaler 在训练集 fit，在 CV 各折与测试集仅 transform；不对测试集做任何形式的早停或重调；

随机性控制：固定 random_state=42，多处使用同一随机种子；

特征使用：仅使用第 3 节得到的 Top-10 列（final_features 对应列索引）。

4.2 算法①：随机森林回归（RF）

设计动机：RF 通过多棵去相关树的装袋集成，能捕捉非线性与高阶交互，且对尺度与分布较鲁棒。

搜索网格：

n_estimators ∈ {100, 200, 300}：平衡方差降低与训练耗时；

max_depth ∈ {None, 10, 20}：控制模型复杂度；

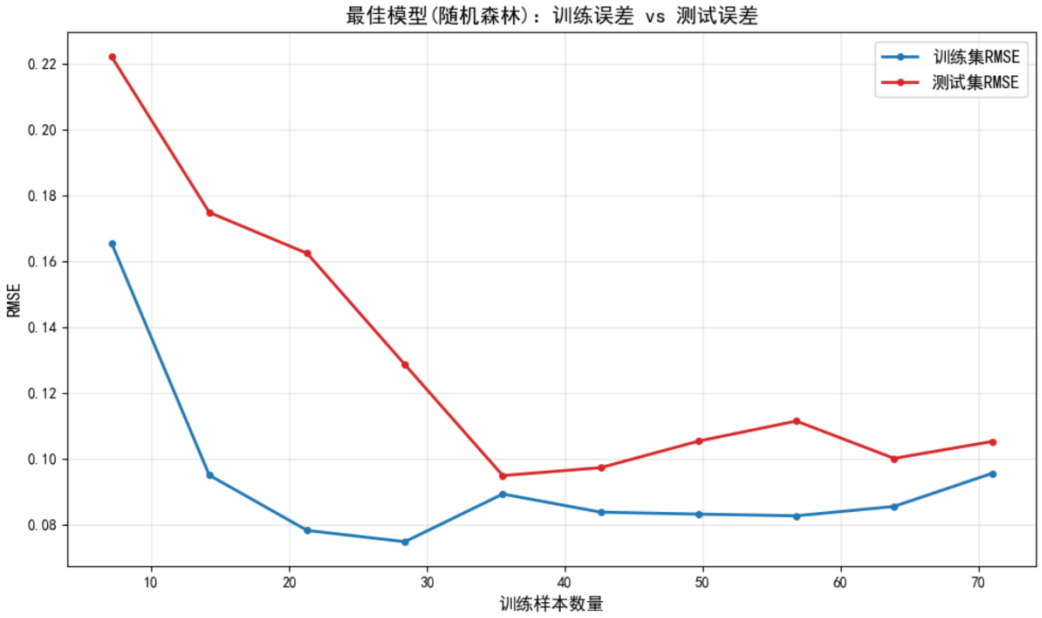
min_samples_split ∈ {2, 5} / min_samples_leaf ∈ {1, 2}：抑制过拟合，平滑叶节点。

验证方案：5 折 CV，评分用 R^2；取平均折分最高点为最佳参数。

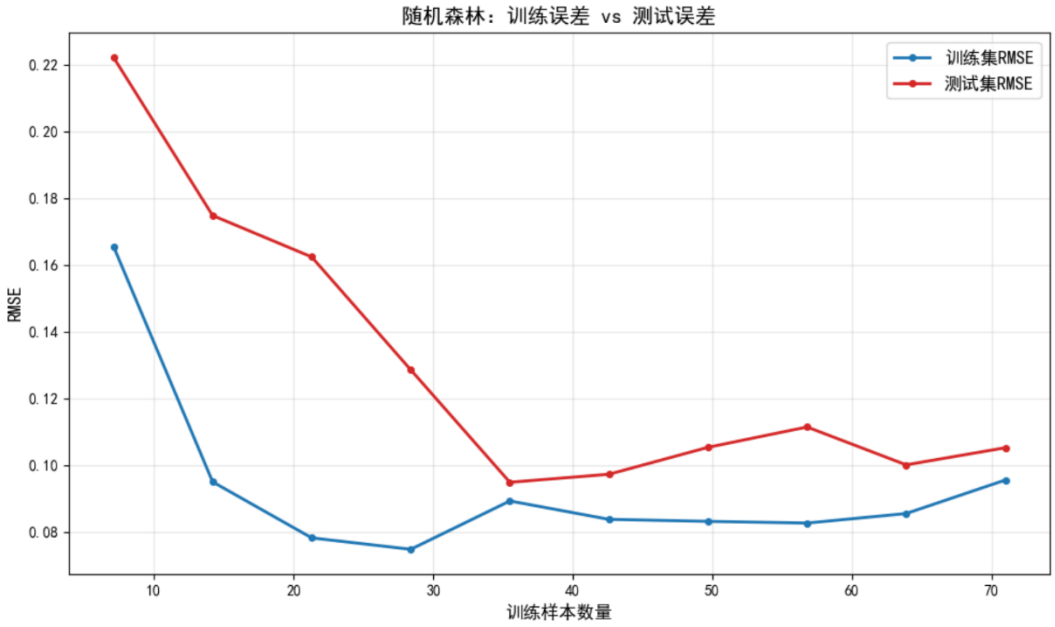
训练输出：最佳参数、训练/测试预测、R^2 与 RMSE、特征重要性。

可视化关联：

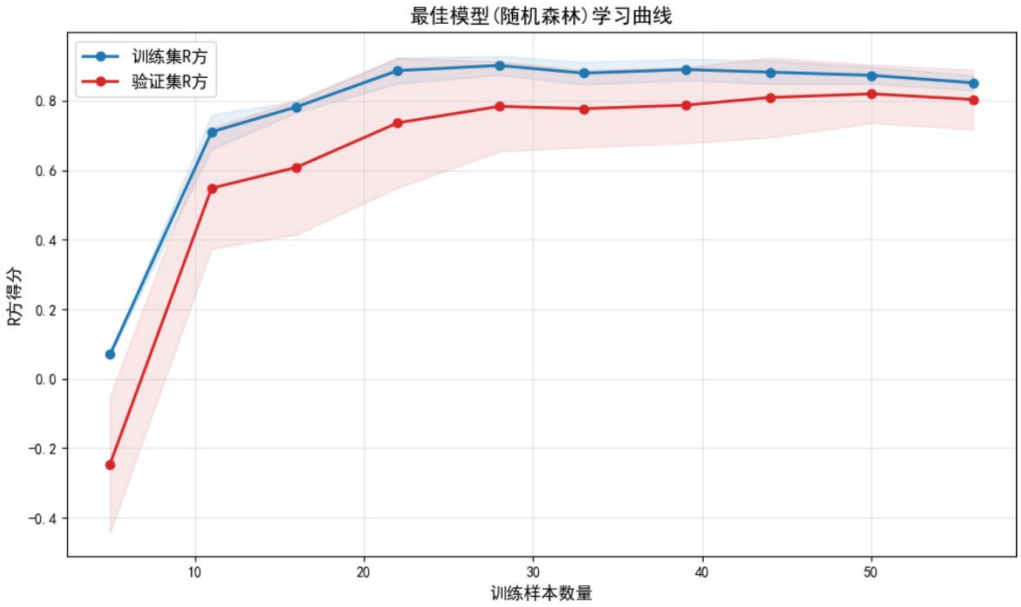
最佳模型误差曲线：

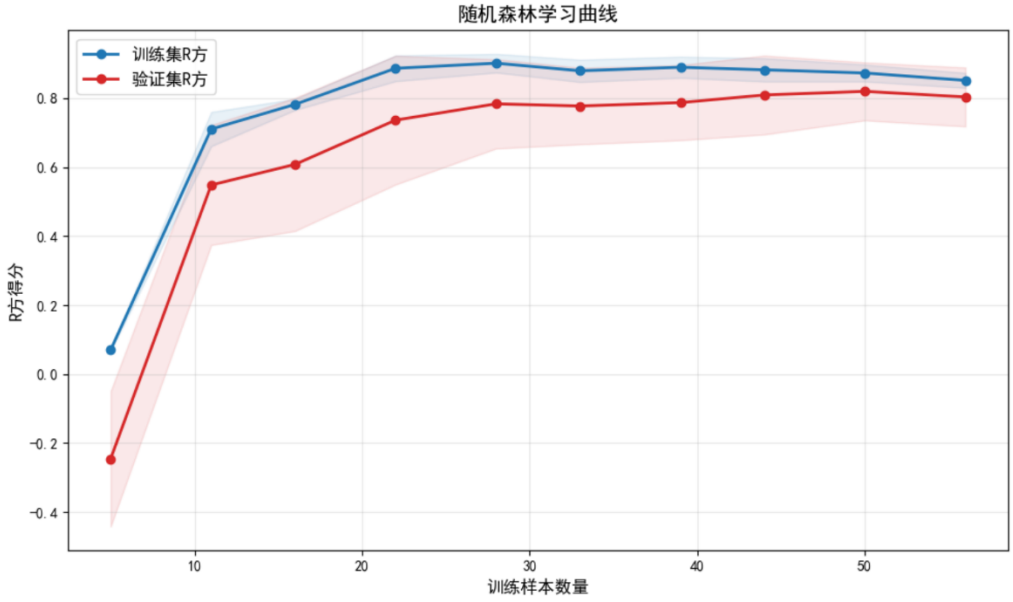


随机森林误差曲线：



最佳模型学习曲线：

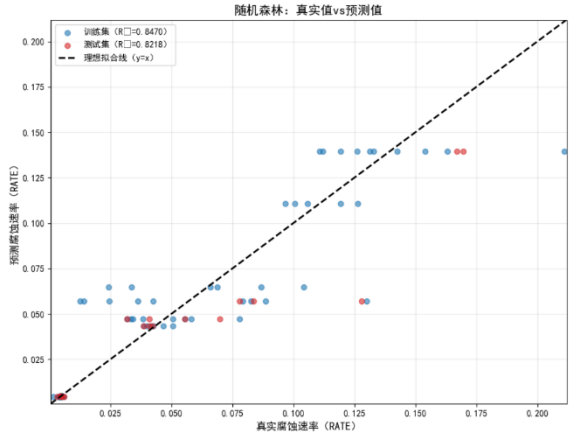


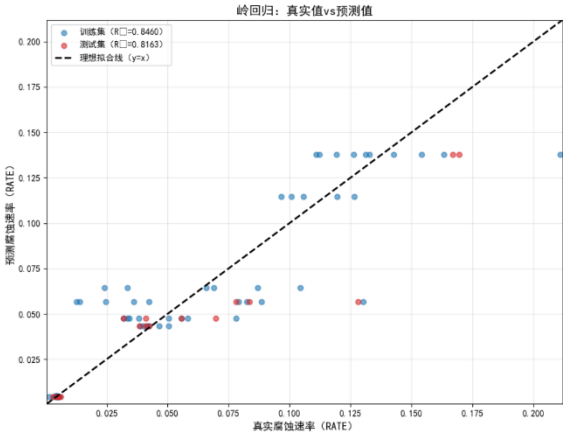


==========

5. 模型性能可视化（真实值vs预测值散点图）

==========





```
============================================================
4. 模型构建与超参数优化（随机森林+岭回归）
============================================================

 随机森林最佳超参数：
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}

随机森林性能指标：
训练集 - RMSE: 0.020011 | R²: 0.846973
测试集 - RMSE: 0.022039 | R²: 0.821833

 岭回归最佳超参数（alpha）：0.000521
岭回归系数（各特征对腐蚀速率的影响程度）：
  ULTRA: -3.709707
  T_MAX: -0.150956
  Cl: 2.182232
  WIND_AVE: 1.572878
  WIND_MAX: -0.763161
  SUN: 0.563827
  SOLAR: 2.782748
  TOW: 0.647532
  PRECIPIT: -1.017827
  T_MIN: -0.156092

岭回归性能指标：
训练集 - RMSE: 0.020074 | R²: 0.846008
测试集 - RMSE: 0.022376 | R²: 0.816335

 最佳模型：随机森林
最佳模型测试集性能：RMSE=0.022039 | R²=0.821833
```

代码（节选）

```python
# 模型1：随机森林回归
#  定义超参数网格
rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

#  网格搜索优化超参数
rf_grid = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=rf_param_grid,
    cv=5,
    scoring='r2',
    n_jobs=-1
)
rf_grid.fit(X_train_final, y_train)

#  最佳随机森林模型
best_rf = rf_grid.best_estimator_
print(f"\n 随机森林最佳超参数：")
print(rf_grid.best_params_)
# 4.1.4 随机森林预测
y_train_pred_rf = best_rf.predict(X_train_final)
y_test_pred_rf = best_rf.predict(X_test_final)

# 反归一化（转回原始尺度）
y_train_original = scaler_y.inverse_transform(y_train.reshape(-1, 1)).flatten()
y_train_pred_rf_original = scaler_y.inverse_transform(y_train_pred_rf.reshape(-1, 1)).flatten()
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
y_test_pred_rf_original = scaler_y.inverse_transform(y_test_pred_rf.reshape(-1, 1)).flatten()

#  随机森林性能评估
rf_train_rmse = np.sqrt(mean_squared_error(y_train_original, y_train_pred_rf_original))
rf_train_r2 = r2_score(y_train_original, y_train_pred_rf_original)
rf_test_rmse = np.sqrt(mean_squared_error(y_test_original, y_test_pred_rf_original))
rf_test_r2 = r2_score(y_test_original, y_test_pred_rf_original)
```

4.3 算法②：岭回归（Ridge）

设计动机：在线性可解释框架下，通过 L2 正则化缓解多重共线，提高稳定性，形成可读系数。

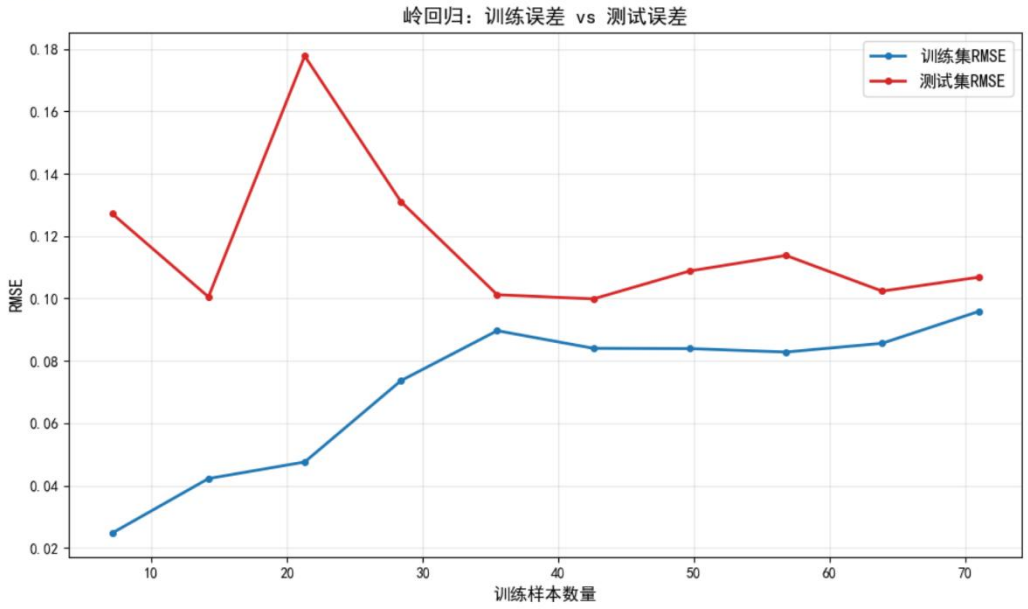
搜索空间：

alpha ∈ [10^-5, 10^5]，对数均匀取 100 点，由 RidgeCV 5 折自动选择；

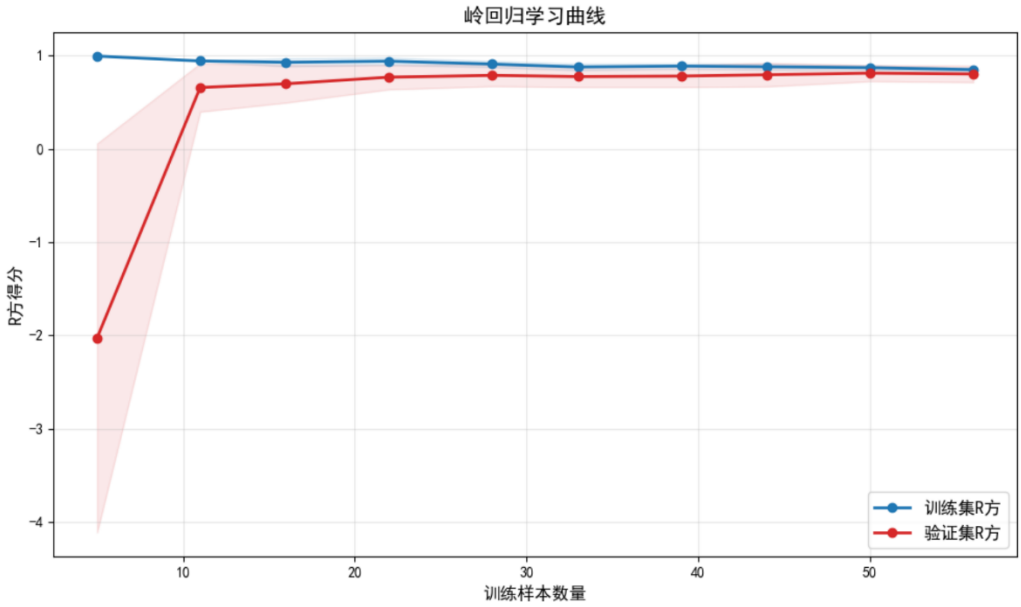
训练输出：最佳 alpha、系数向量、训练/测试集 R^2 与 RMSE。

可视化关联：

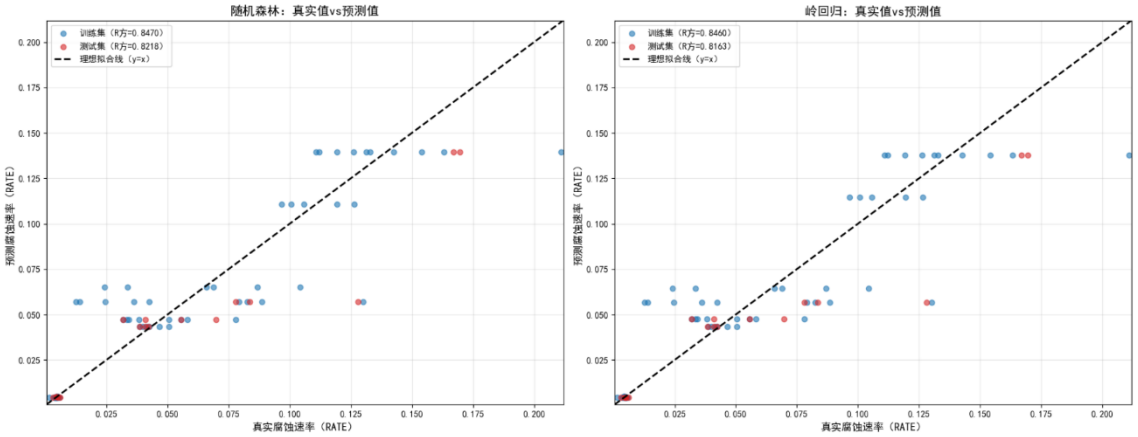
岭回归误差曲线：



岭回归学习曲线：



```
============================================================
5. 模型性能可视化（真实值vs预测值散点图）
============================================================
```




```
============================================================
4. 模型构建与超参数优化（随机森林+岭回归）
============================================================

 随机森林最佳超参数：
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}

随机森林性能指标：
训练集 - RMSE: 0.020011 | R²: 0.846973
测试集 - RMSE: 0.022039 | R²: 0.821833

 岭回归最佳超参数（alpha）：0.000521
岭回归系数（各特征对腐蚀速率的影响程度）：
  ULTRA: -3.709707
  T_MAX: -0.150956
  Cl: 2.182232
  WIND_AVE: 1.572878
  WIND_MAX: -0.763161
  SUN: 0.563827
  SOLAR: 2.782748
  TOW: 0.647532
  PRECIPIT: -1.017827
  T_MIN: -0.156092

岭回归性能指标：
训练集 - RMSE: 0.020074 | R²: 0.846008
测试集 - RMSE: 0.022376 | R²: 0.816335

 最佳模型：随机森林
最佳模型测试集性能：RMSE=0.022039 | R²=0.821833
```

代码（节选）

```python
# 模型2：岭回归
#  超参数优化（选择最佳alpha）
alphas = 10 ** np.linspace(-5, 5, 100)
ridge_cv = RidgeCV(alphas=alphas, cv=5, scoring='r2')
ridge_cv.fit(X_train_final, y_train)

# 最佳岭回归模型
best_ridge = Ridge(alpha=ridge_cv.alpha_)
best_ridge.fit(X_train_final, y_train)
print(f"\n 岭回归最佳超参数（alpha）：{ridge_cv.alpha_:.6f}")
print(f"岭回归系数（各特征对腐蚀速率的影响程度）：")
for feat, coef in zip(final_features, best_ridge.coef_):
    print(f"  {feat}: {coef:.6f}")

#  岭回归预测
y_train_pred_ridge = best_ridge.predict(X_train_final)
y_test_pred_ridge = best_ridge.predict(X_test_final)

# 反归一化
y_train_pred_ridge_original = scaler_y.inverse_transform(y_train_pred_ridge.reshape(-1, 1)).flatten()
y_test_pred_ridge_original = scaler_y.inverse_transform(y_test_pred_ridge.reshape(-1, 1)).flatten()

# 岭回归性能评估
ridge_train_rmse = np.sqrt(mean_squared_error(y_train_original, y_train_pred_ridge_original))
ridge_train_r2 = r2_score(y_train_original, y_train_pred_ridge_original)
ridge_test_rmse = np.sqrt(mean_squared_error(y_test_original, y_test_pred_ridge_original))
ridge_test_r2 = r2_score(y_test_original, y_test_pred_ridge_original)
```

4.4 训练与评估的统一流程

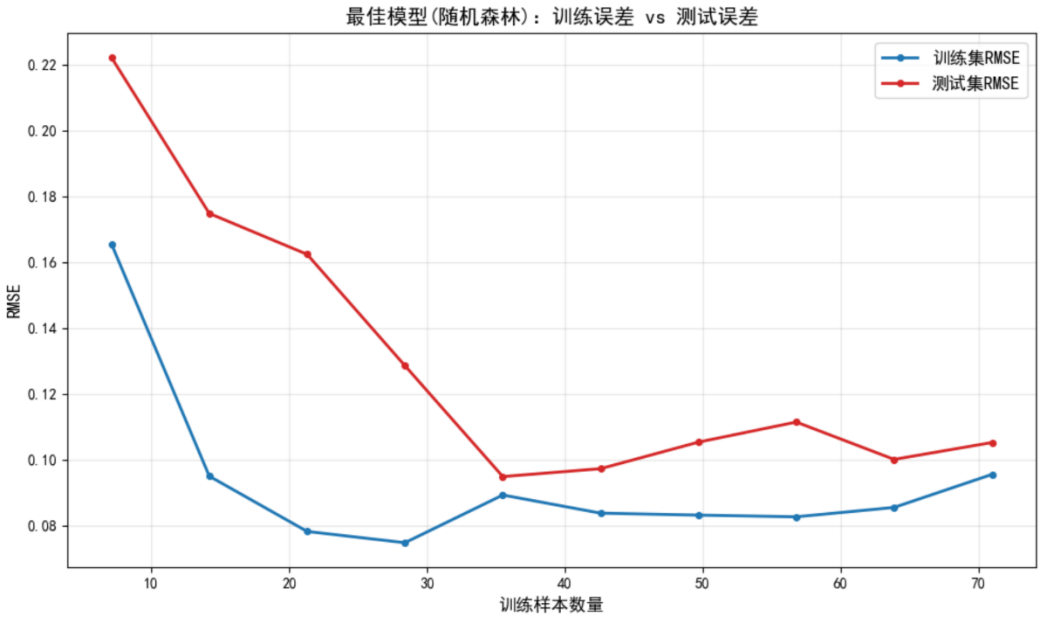
用最佳 RF 与最佳 Ridge 分别在相同 Top-10 特征上训练；

在测试集上输出预测，计算 R^2 与 RMSE；

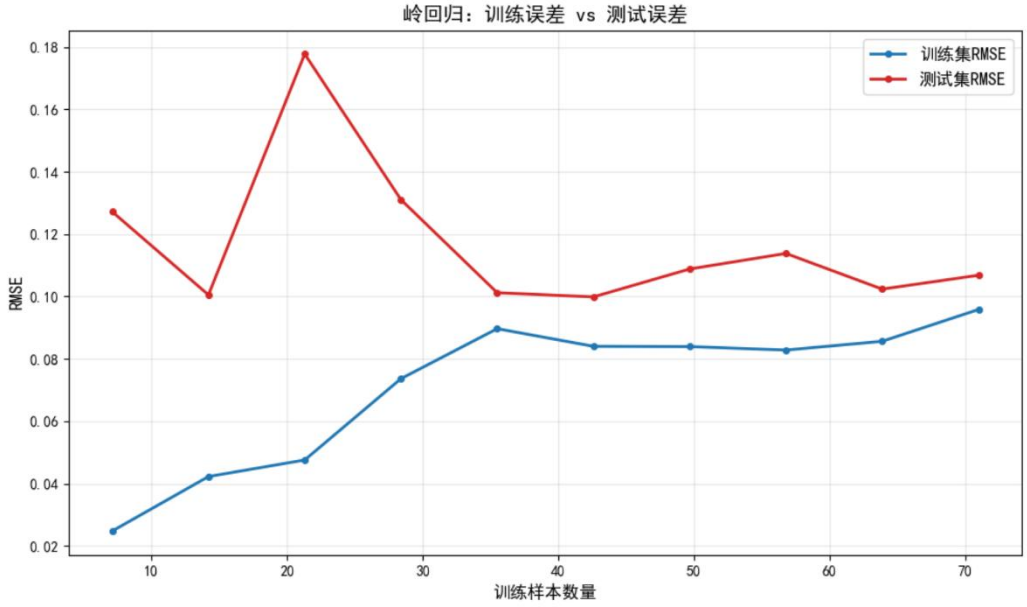
生成散点图/学习曲线/误差曲线与结果汇总表，统一口径下比较优劣。

```
============================================================
9. 结果汇总
============================================================

 模型性能结果汇总：
   模型名称                                                最佳超参数    训练集RMSE  \
0  随机森林  {'max_depth': None, 'min_samples_leaf': 1, 'mi...  0.020011
1   岭回归                                    alpha=0.000521  0.020074

      训练集R²   测试集RMSE     测试集R² 是否最佳模型
0  0.846973  0.022039  0.821833       是
1  0.846008  0.022376  0.816335       否

 最终筛选特征列表：
     最终输入特征  特征类型
0     ULTRA  环境参数
1     T_MAX  环境参数
2        Cl  元素成分
3  WIND_AVE  环境参数
4  WIND_MAX  环境参数
5       SUN  环境参数
6     SOLAR  环境参数
7       TOW  环境参数
8  PRECIPIT  环境参数
9     T_MIN  环境参数

最终确定的最佳模型为：随机森林模型
```

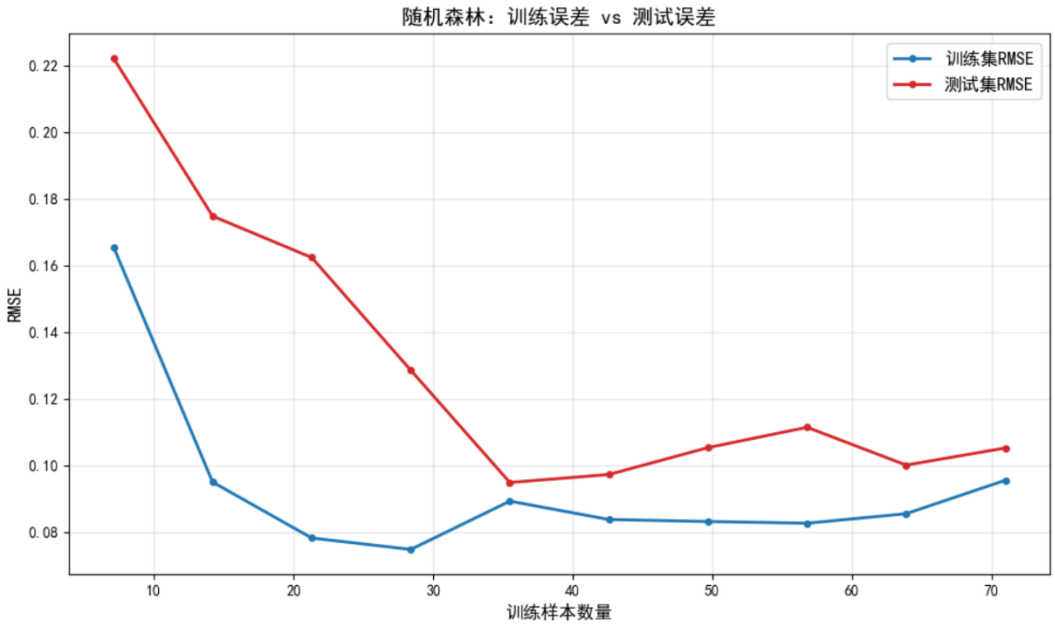
最佳模型误差曲线：



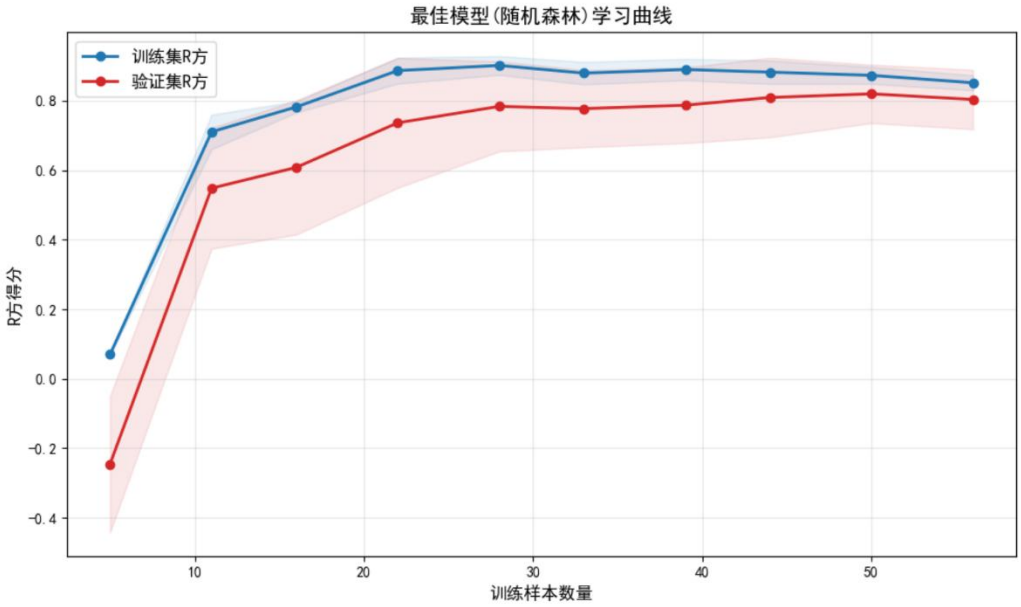
岭回归误差曲线：



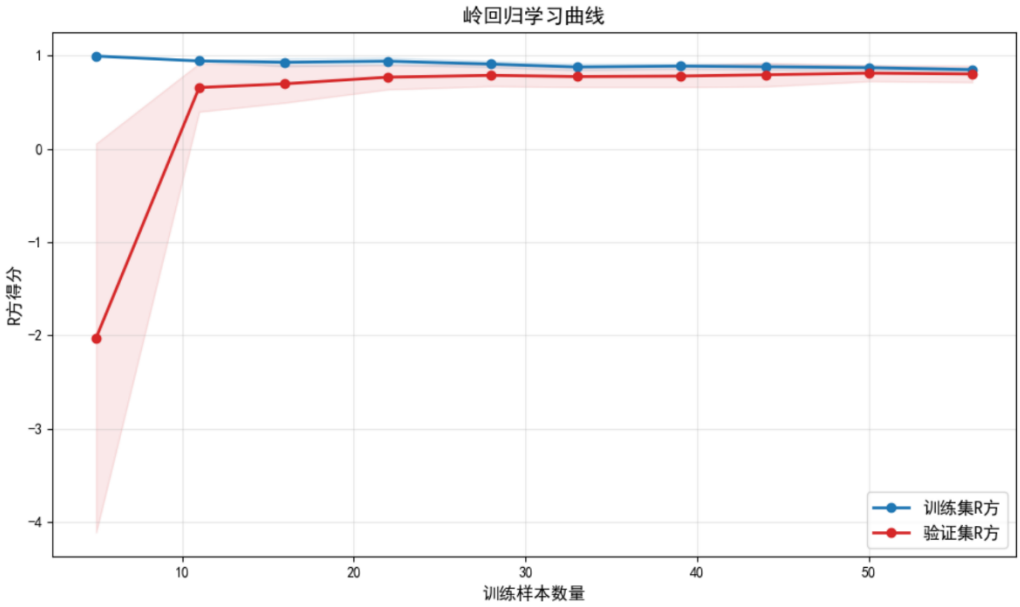
随机森林误差曲线：

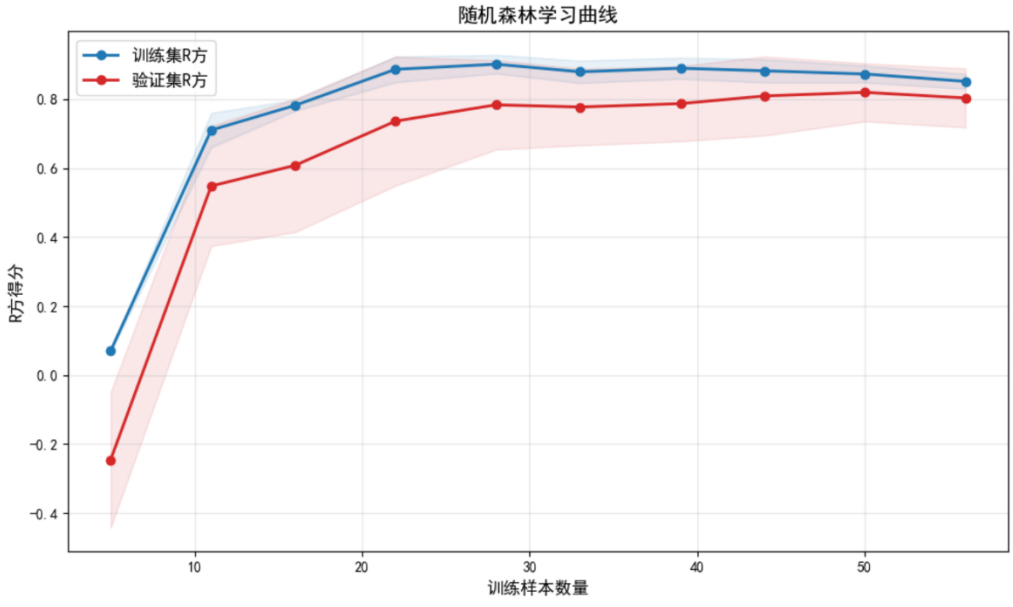


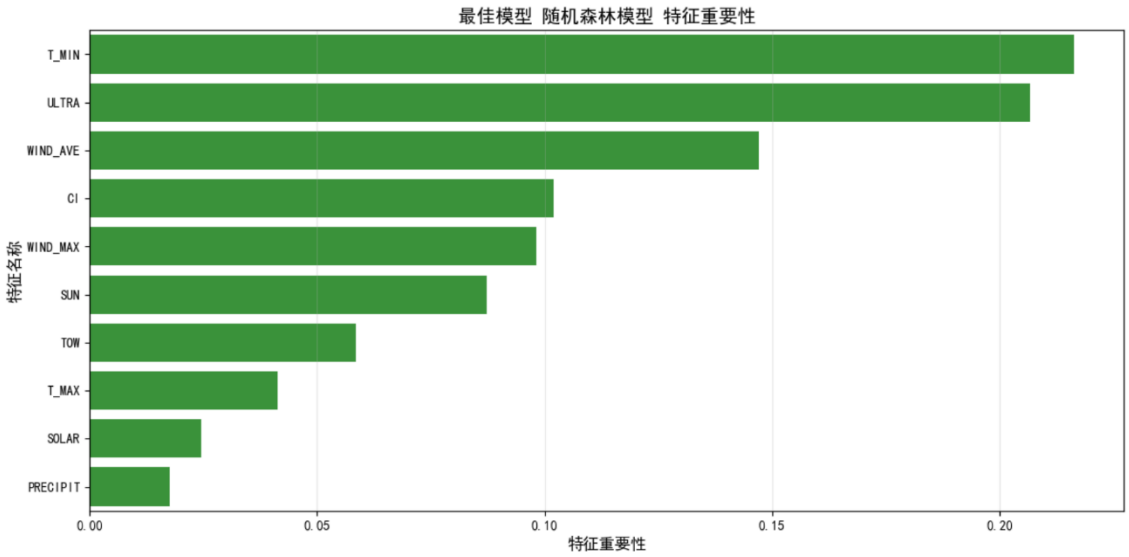
最佳模型学习曲线：



岭回归学习曲线：





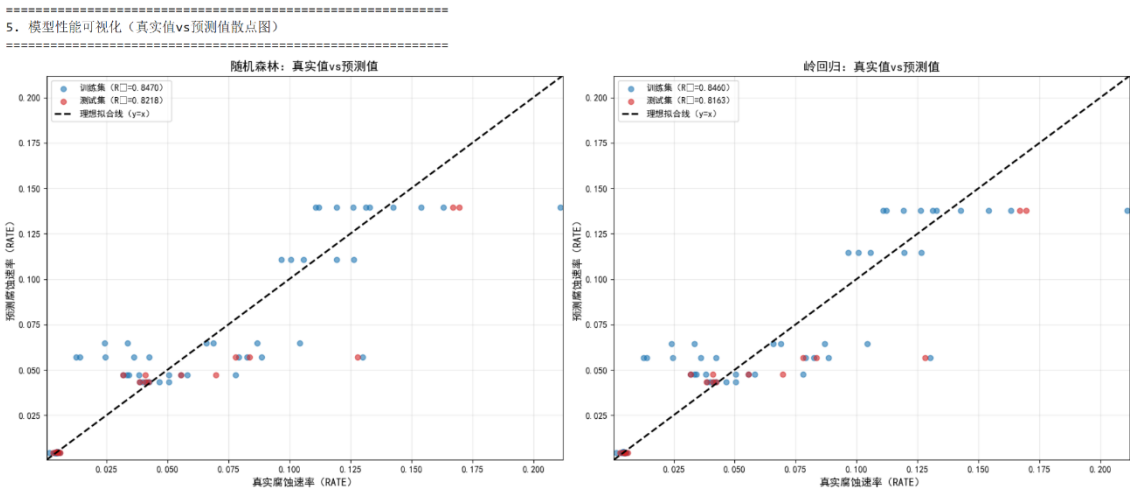


代码（节选）

```python
# 最佳模型选择
best_model = best_rf if rf_test_r2 > ridge_test_r2 else best_ridge
best_model_name = "随机森林" if rf_test_r2 > ridge_test_r2 else "岭回归"
best_test_r2 = max(rf_test_r2, ridge_test_r2)
best_test_rmse = min(rf_test_rmse, ridge_test_rmse)
print(f"\n 最佳模型: {best_model_name}")
print(f"最佳模型测试集性能: RMSE={best_test_rmse:.6f} | R²={best_test_r2:.6f}")
```

5. 结果解读、可解释性与误差分析

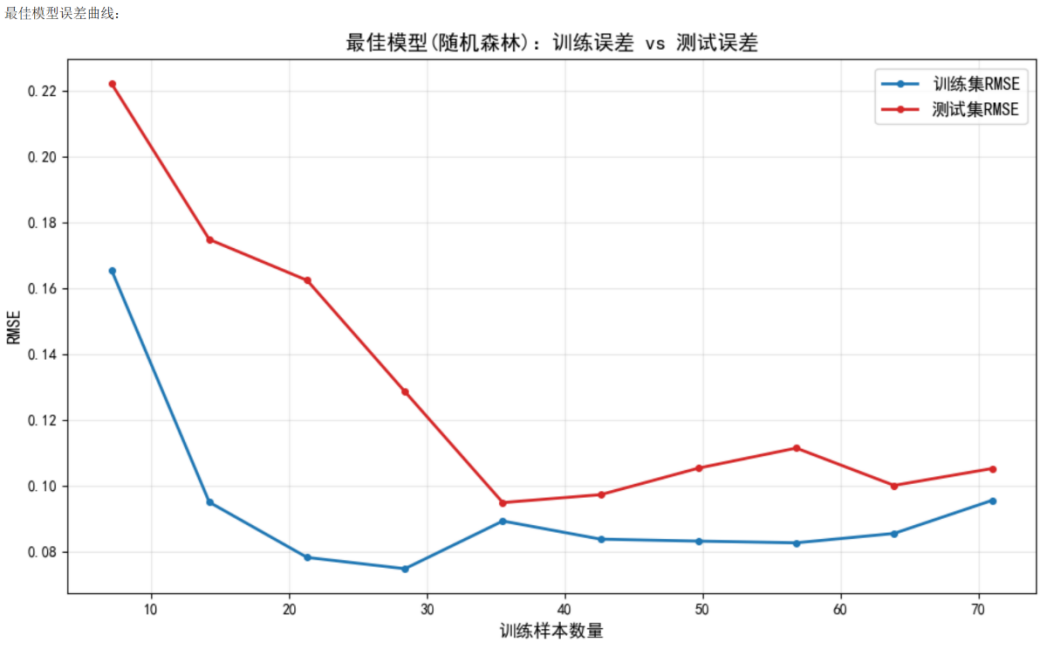
5.1 散点对比（真实 vs 预测）



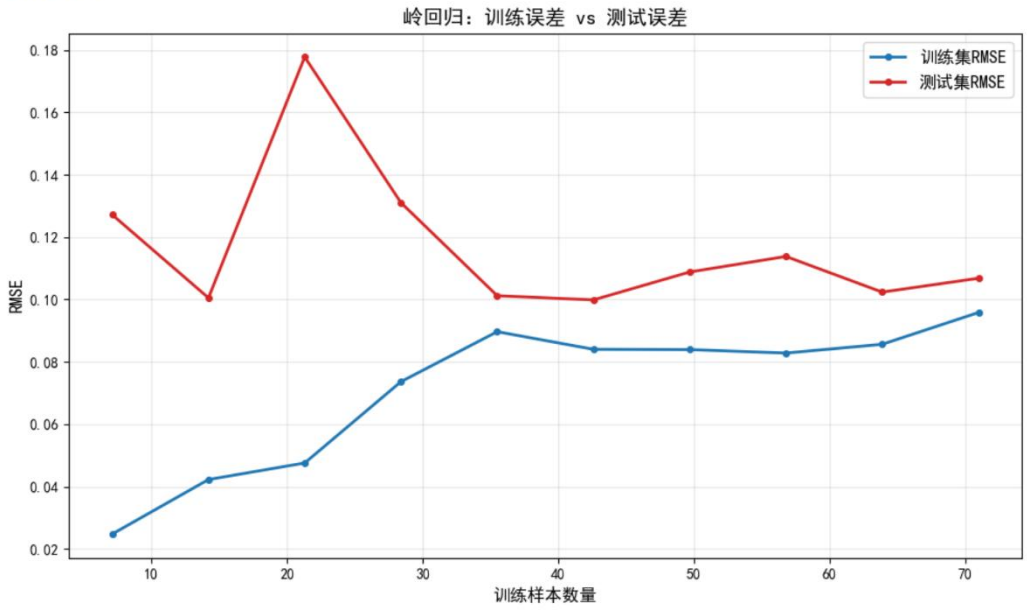
RF 散点更贴近对角线，特别是在高腐蚀段，极值压缩更轻；

Ridge 对高值区出现系统性低估（线性容量不足 + 共线影响）。

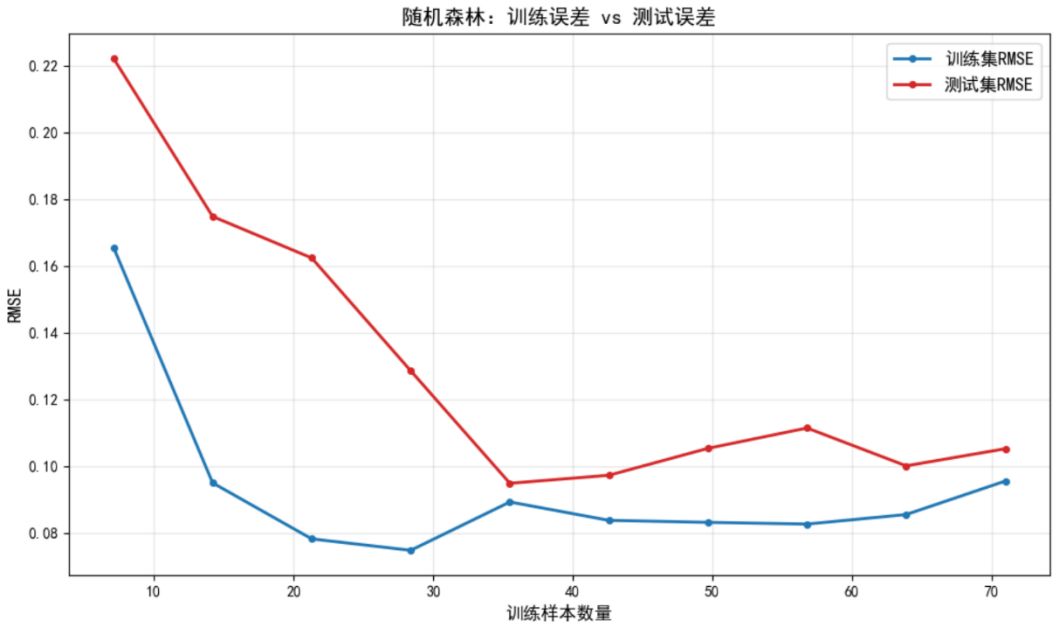
5.2 学习曲线与误差曲线



岭回归误差曲线：



随机森林误差曲线：

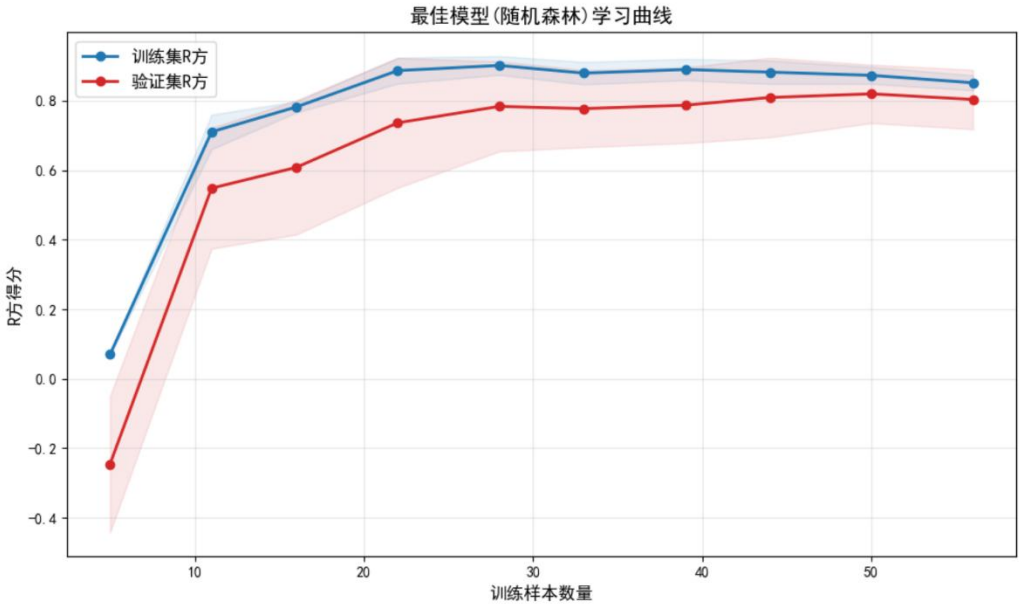


RF：验证 R^2 随样本增加仍上升、RMSE 下降 → 模型仍能吞吐更多数据；

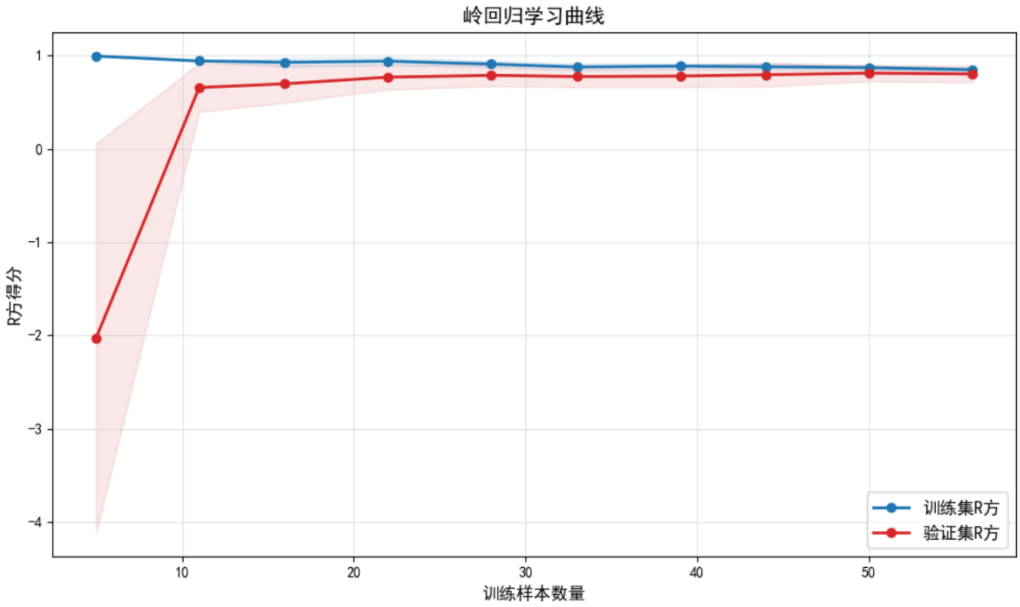
Ridge：较早收敛，验证上限偏低 → 线性假设难以承载非线性/交互。

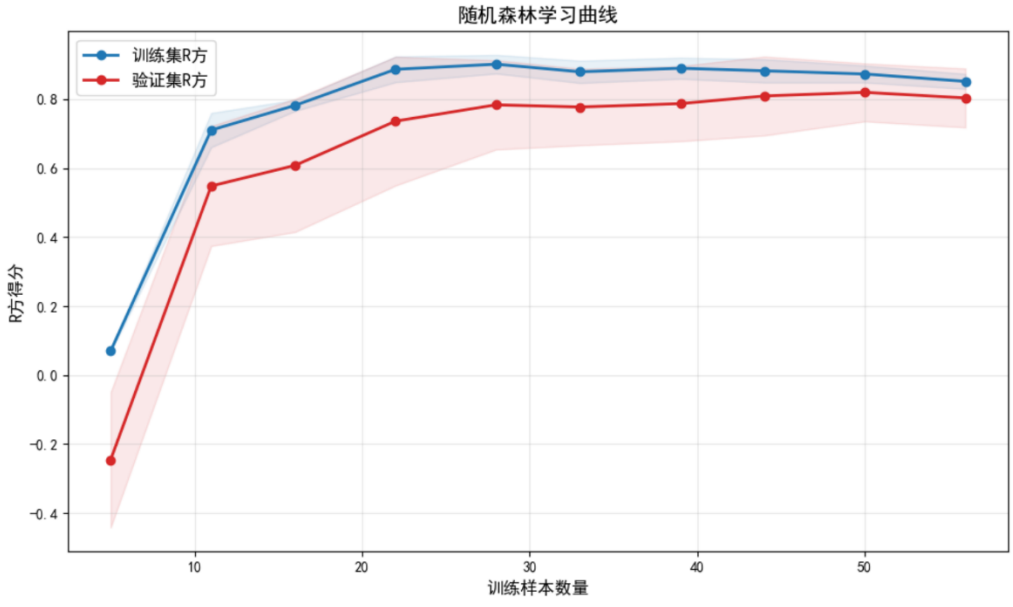
过拟合检查：RF 的 train-test 间隙可控，主要由数据噪声与极端值驱动；Ridge 的间隙小但上限受限。

最佳模型学习曲线：



岭回归学习曲线：





5.3 最佳模型判定（结论 + 证据链）

```
============================================================
9. 结果汇总
============================================================

 模型性能结果汇总：
   模型名称                                              最佳超参数    训练集RMSE  \
0  随机森林  {'max_depth': None, 'min_samples_leaf': 1, 'mi...  0.020011
1   岭回归                                    alpha=0.000521  0.020074

      训练集R²   测试集RMSE     测试集R² 是否最佳模型
0  0.846973  0.022039  0.821833       是
1  0.846008  0.022376  0.816335       否

 最终筛选特征列表：
     最终输入特征  特征类型
0     ULTRA  环境参数
1     T_MAX  环境参数
2        Cl  元素成分
3  WIND_AVE  环境参数
4  WIND_MAX  环境参数
5       SUN  环境参数
6    SOLAR  环境参数
7       TOW  环境参数
8  PRECIPIT  环境参数
9     T_MIN  环境参数

最终确定的最佳模型为：随机森林模型
```

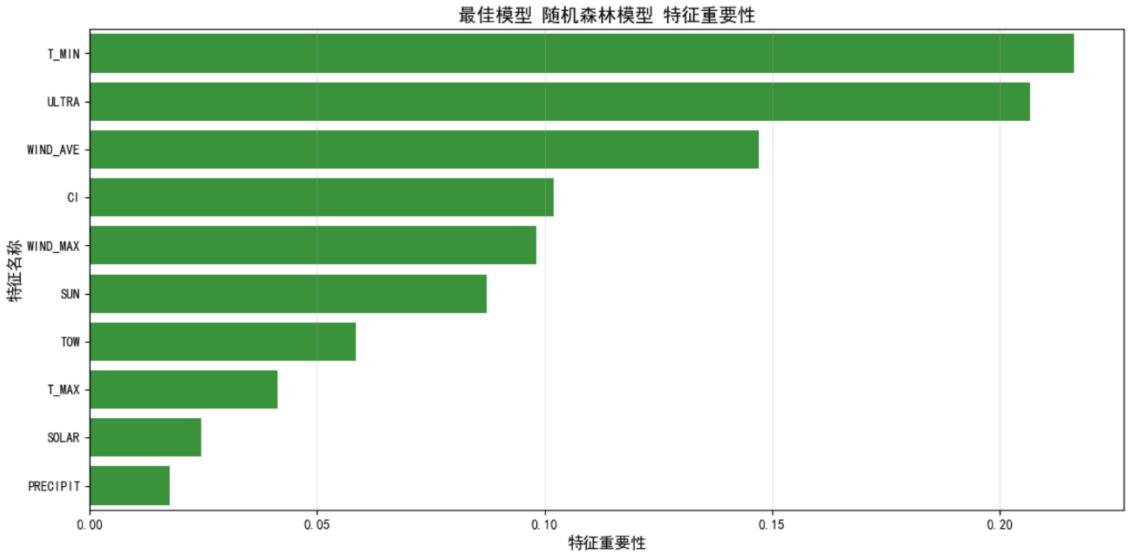
指标层面：测试集 R^2：RF ＞ Ridge；RMSE：RF ＜ Ridge；

形态层面：散点贴合对角线（图 10）更优；

泛化层面：RF 学习/误差曲线（图 12/14/15/17）更佳。

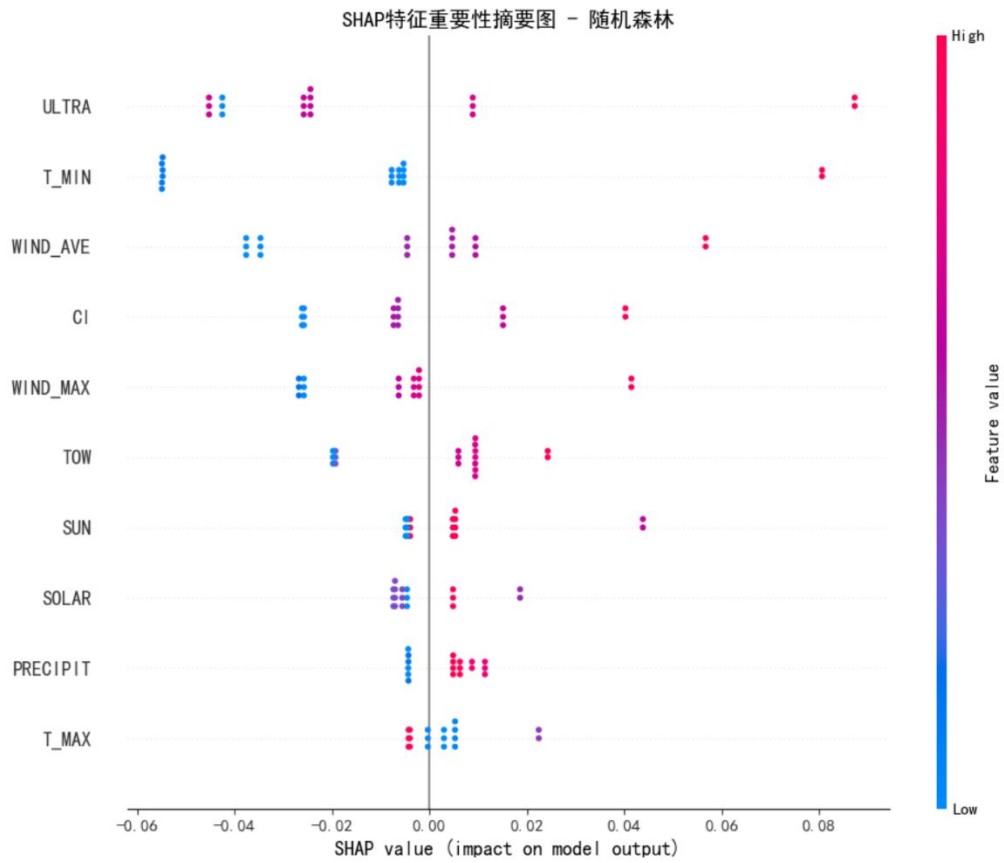
结论：随机森林为最终最佳模型。

5.4 特征重要性与 SHAP 解释



前列：ULTRA、T_MIN、WIND_AVE、CL、WIND_MAX、TOW、SUN、SOLAR、PRECIPIT、T_MAX。

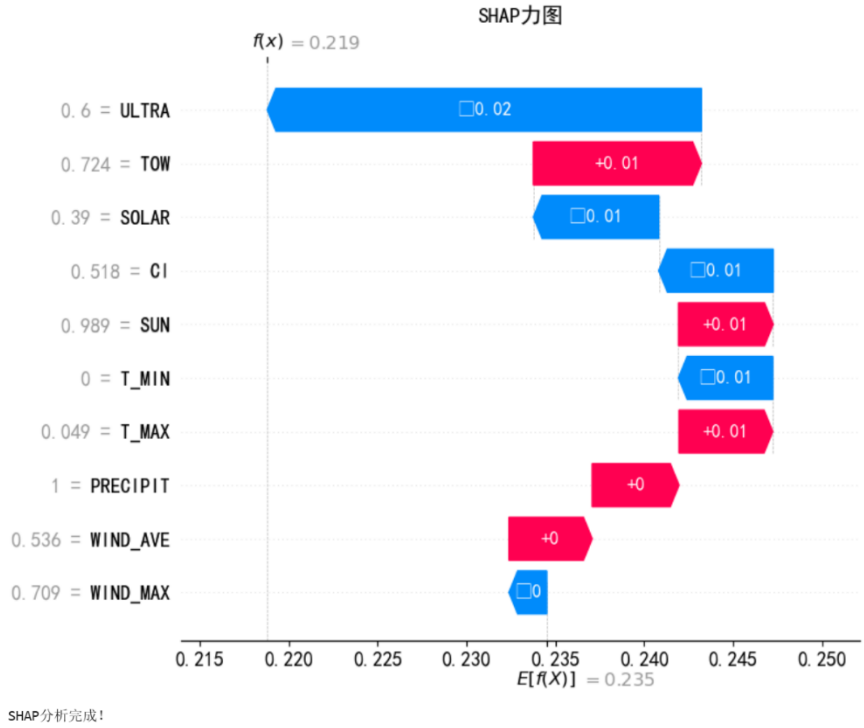
SHAP分析如下：



正向驱动：ULTRA、CL、WIND_AVE/WIND_MAX、TOW、SUN/SOLAR；

负向趋势：T_MAX 多集中在 0 左侧（与 T_MAX ↑ → TOW ↓ 的链式关系一致）；

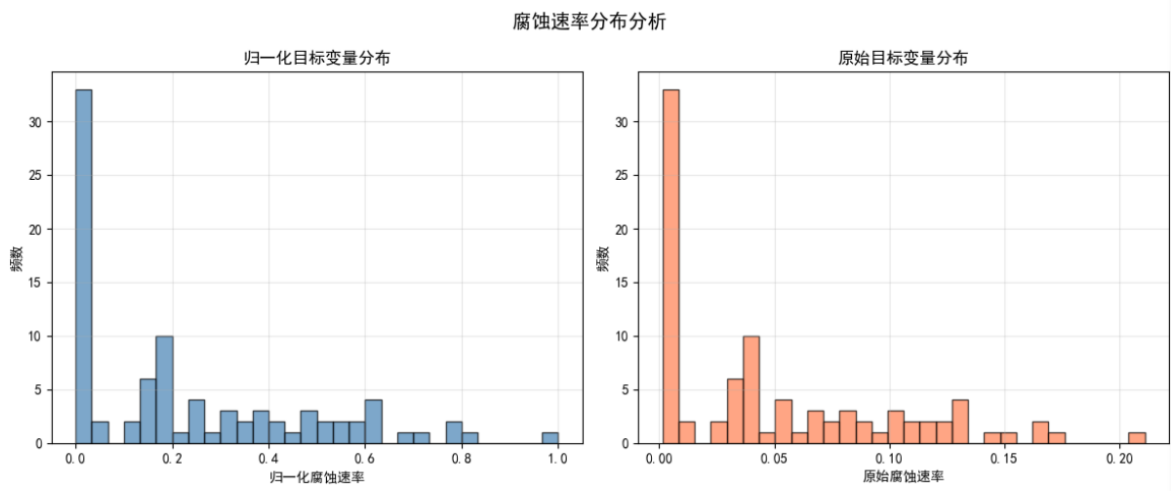
T_MIN 偏高亦常为正（提示夜/最低温较高意味着持续腐蚀条件）。



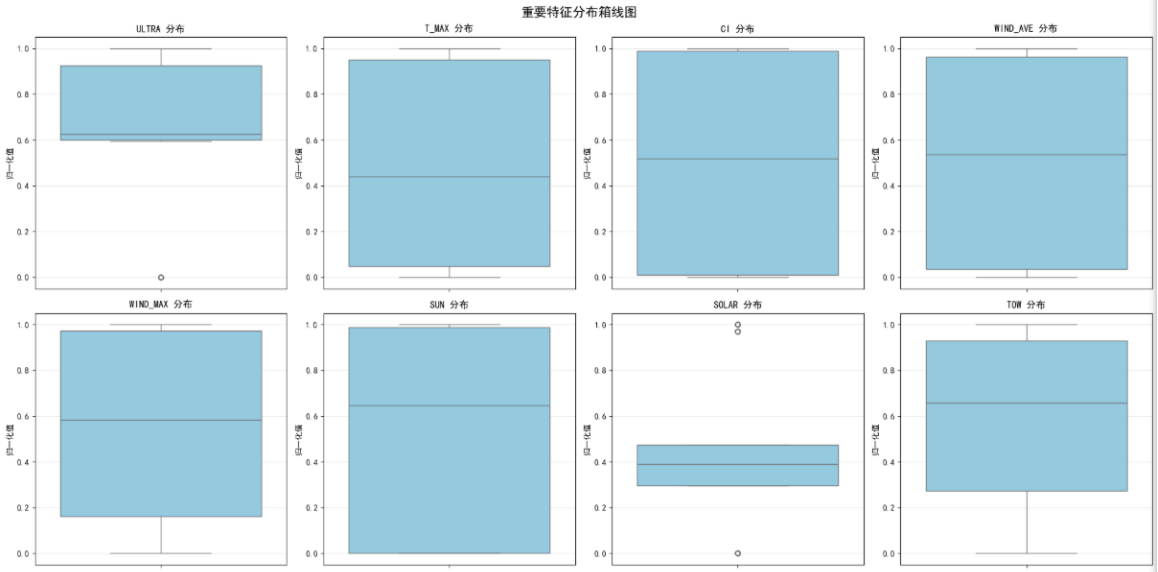
SHAP分析完成！

红色箭头推高、蓝色箭头拉低；若样本的 CL/TOW/ULTRA 偏高则显著推高预测，说明模型捕捉到非线性阈值与交互而非线性叠加。

5.5 分布与结构补证

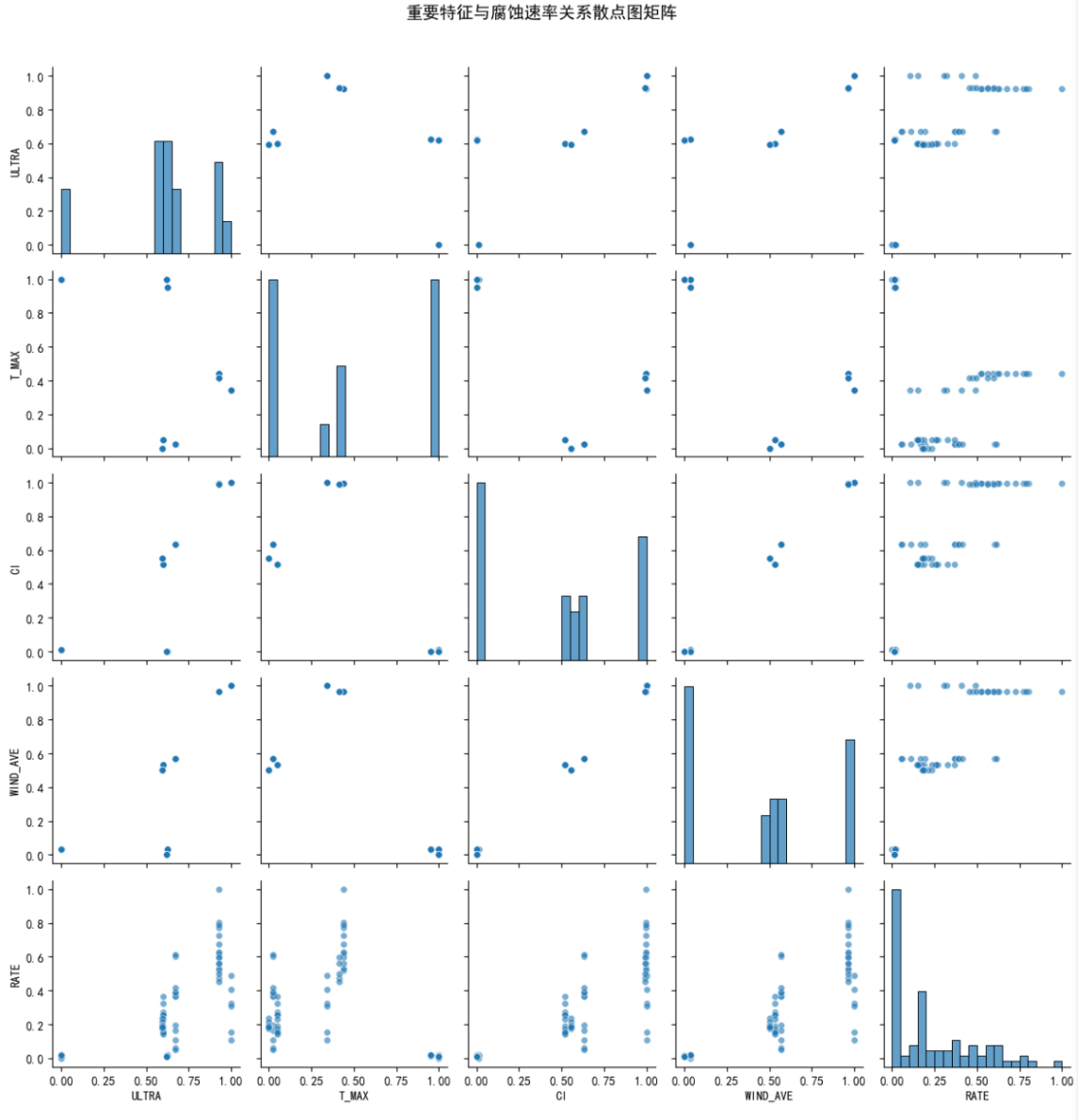


特征分布箱线图：



目标变量分布：

特征-目标变量关系散点图：



RATE 随 ULTRA/CL/TOW 增大而上扬；高风速对应更高的右尾；

结合热力图，进一步确认 Top-10 已覆盖主驱动，继续加入长尾变量对验证性能提升有限。

5.6 误差来源与稳健性讨论

数据尺度：误差曲线显示 RF 对样本扩充敏感 → 建议扩充极端气候/高盐雾/高 SO_2 场景样本；

噪声点：建议对高杠杆点做稳健缩放/Winsorize；

共线组：风-辐照簇在 Ridge 中会带来方差膨胀，RF 受影响较小但会影响分裂稳定性 → 继续保持“Top-10 + 合理调参”。

6.提升建议

建模提升：引入 XGBoost/LightGBM/CatBoost/GPR 横向对比；使用贝叶斯优化提升调参效率；考虑堆叠泛化（线性 + 非线性）。

深入解释：补充 PDP/ICE 与 SHAP 交互值 分析二阶交互（如 ULTRA×TOW、CL×WIND_MAX）；

不确定性与鲁棒性：输出置信区间、分布外检测与漂移监控。

---

7. 结论

最佳模型：随机森林回归（测试集 R^2 更高、RMSE 更低；学习/误差曲线与散点形态均占优）。

关键驱动因子：ULTRA、CL、WIND_AVE/WIND_MAX、TOW、T_MIN；T_MAX 在数据中呈抑制趋势。

方法学贡献：MI + RF 双法筛选结合相关性热力图与性能曲线三重证据，证明 Top-10 是在当前样本规模下“信号覆盖—冗余控制”的最优折中。

附件：源代码

```
#  1. 导入所有依赖库
import pandas as pd  # 读取 Excel 数据、处理表格
import numpy as np   # 数值计算（数组、矩阵操作）
import matplotlib.pyplot as plt  # 绘制可视化图表
import seaborn as sns  # 增强型可视化（特征重要性、相关性图）
from sklearn.preprocessing import MinMaxScaler  # 数据归一化（消除量纲影响）
from sklearn.model_selection import (
    train_test_split,  # 划分训练集/测试集
    GridSearchCV       # 超参数网格搜索优化
)
from sklearn.metrics import (
    mean_squared_error,  # 计算均方误差（MSE）
    r2_score             # 计算拟合优度（R²，衡量预测准确性）
)
from sklearn.ensemble import RandomForestRegressor  # 随机森林回归（非线性模型）
from sklearn.linear_model import Ridge, RidgeCV     # 岭回归（线性模型，抗过拟合）
from sklearn.feature_selection import mutual_info_regression  # 互信息特征筛选
from sklearn.model_selection import learning_curve  # 学习曲线
from sklearn.base import clone  # 模型克隆
import warnings
warnings.filterwarnings('ignore')  # 忽略无关警告（如版本兼容提示）

# 设置中文字体（解决 matplotlib 中文乱码问题）
plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

#  2. 数据读取与基础预处理
print("="*60)
print("2. 数据读取与基础预处理")
print("="*60)

# 读取 Excel 数据
file_path = r"C:\Users\lenovo\Desktop\材料数据挖掘利用方法-机器学习与数据
```

```
挖掘\0 大作业目标数据集.xlsx"
df = pd.read_excel(file_path, sheet_name="10year", engine='openpyxl')  # 读取 10year 表

# 数据基础信息查看
print(f"数据形状：{df.shape} → 共{df.shape[0]}个样本，{df.shape[1]}个特征/目标变量")
print(f"\n 数据列名（特征+目标变量）：")
print(df.columns.tolist())
print(f"\n 数据前 5 行预览：")
print(df.head())
print(f"\n 数据缺失值统计：")
print(df.isnull().sum())

# 定义输入特征（X）与目标变量（y）
X = df.drop(columns=['RATE']).copy()  # 输入特征（元素成分+环境参数）
y = df['RATE'].copy()                 # 目标变量（腐蚀速率）
print()
print(f"\n 输入特征形状：{X.shape}，目标变量形状：{y.shape}")

# 数据归一化（消除特征间量纲差异）
scaler_X = MinMaxScaler(feature_range=(0, 1))  # 特征归一化器
scaler_y = MinMaxScaler(feature_range=(0, 1))  # 目标变量归一化器

X_scaled = scaler_X.fit_transform(X)  # 特征归一化
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).flatten()  # 目标变量归一化

# 划分训练集与测试集（8:2）
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_scaled, test_size=0.2, random_state=42
)
print()
print(f"\n 数据集拆分结果：")
print(f"训练集：X_train({X_train.shape}) | y_train({y_train.shape})")
print(f"测试集：X_test({X_test.shape}) | y_test({y_test.shape})")

# 3. 特征筛选 互信息回归 与 随机森林特征
print("\n" + "="*60)
print("3. 特征筛选（互信息回归+随机森林特征重要性）")
print("="*60)

# 互信息回归（衡量非线性相关性）
mi_scores = mutual_info_regression(X_train, y_train)  # 计算互信息得分
mi_result = pd.DataFrame({
    'Feature': X.columns,
```

```
    'MI_Score': mi_scores
}).sort_values(by='MI_Score', ascending=False)

print(f"\n互信息回归特征重要性：")
print(mi_result)

# 3.2 随机森林特征重要性
rf_temp = RandomForestRegressor(n_estimators=100, random_state=42)
rf_temp.fit(X_train, y_train)
rf_importance_result = pd.DataFrame({
    'Feature': X.columns,
    'RF_Importance': rf_temp.feature_importances_
}).sort_values(by='RF_Importance', ascending=False)

print(f"\n 随机森林特征重要性：")
print(rf_importance_result)

#  所有特征重要性可视化
print("\n 所有特征重要性可视化：")

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 12))

# 互信息回归所有特征重要性
sns.barplot(x='MI_Score', y='Feature', data=mi_result, ax=ax1,
color='#1f77b4')
ax1.set_title('互信息回归特征重要性', fontsize=14, fontweight='bold')
ax1.set_xlabel('互信息得分', fontsize=12)
ax1.set_ylabel('特征名称', fontsize=12)
ax1.grid(axis='x', alpha=0.3)

# 随机森林所有特征重要性
sns.barplot(x='RF_Importance', y='Feature', data=rf_importance_result,
ax=ax2, color='#ff7f0e')
ax2.set_title('随机森林特征重要性', fontsize=14, fontweight='bold')
ax2.set_xlabel('特征重要性', fontsize=12)
ax2.set_ylabel('特征名称', fontsize=12)
ax2.grid(axis='x', alpha=0.3)

plt.tight_layout()
plt.show()

#  确定最终输入特征（取两种方法的交集，不足 10 个则用互信息补充）
mi_top10 = mi_result.head(10)['Feature'].tolist()
rf_top10 = rf_importance_result.head(10)['Feature'].tolist()
final_features = list(set(mi_top10) & set(rf_top10))
if len(final_features) < 10:
```

```
    supplement = [f for f in mi_top10 if f not in final_features]
    final_features += supplement[:10 - len(final_features)]

print(f"\n 最终筛选的输入特征（{len(final_features)}个）：")
print(final_features)

# 筛选后的数据
X_train_df = pd.DataFrame(X_train, columns=X.columns)
X_test_df = pd.DataFrame(X_test, columns=X.columns)
X_train_final = X_train_df[final_features].values  # 训练集最终特征
X_test_final = X_test_df[final_features].values    # 测试集最终特征
print(f"\n筛选后数据形状：")
print(f"训练集：X_train_final({X_train_final.shape}) | 测试集：
X_test_final({X_test_final.shape})")

#  前 10 个重要特征对比可视化
print("\n 前 10 个重要特征对比可视化：")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

# 互信息回归前 10 个特征重要性
sns.barplot(x='MI_Score', y='Feature', data=mi_result.head(10), ax=ax1,
color='#1f77b4')
ax1.set_title('互信息回归特征重要性（前 10）', fontsize=14,
fontweight='bold')
ax1.set_xlabel('互信息得分', fontsize=12)
ax1.set_ylabel('特征名称', fontsize=12)
ax1.grid(axis='x', alpha=0.3)

# 随机森林前 10 个特征重要性
sns.barplot(x='RF_Importance', y='Feature',
data=rf_importance_result.head(10), ax=ax2, color='#ff7f0e')
ax2.set_title('随机森林特征重要性（前 10）', fontsize=14,
fontweight='bold')
ax2.set_xlabel('特征重要性', fontsize=12)
ax2.set_ylabel('特征名称', fontsize=12)
ax2.grid(axis='x', alpha=0.3)

plt.tight_layout()
plt.show()

# 两种方法特征重要性对比（前 10 个特征）
print("\n 两种方法特征重要性对比：")

# 创建对比数据
comparison_data = []
```

```
for feature in mi_result.head(10)['Feature']:
    mi_score = mi_result[mi_result['Feature'] ==
feature]['MI_Score'].values[0]
    rf_importance = rf_importance_result[rf_importance_result['Feature']
== feature]['RF_Importance'].values[0]
    comparison_data.append({
        'Feature': feature,
        'MI_Score': mi_score,
        'RF_Importance': rf_importance
    })

comparison_df = pd.DataFrame(comparison_data)

# 绘制对比图
fig, ax = plt.subplots(figsize=(14, 8))
x = np.arange(len(comparison_df))
width = 0.35

bars1 = ax.bar(x - width/2, comparison_df['MI_Score'], width, label='互信
息得分', color='#1f77b4', alpha=0.8)
bars2 = ax.bar(x + width/2, comparison_df['RF_Importance'], width, label='
随机森林重要性', color='#ff7f0e', alpha=0.8)

ax.set_xlabel('特征名称', fontsize=12)
ax.set_ylabel('重要性得分', fontsize=12)
ax.set_title('两种特征筛选方法对比（前 10 个特征）', fontsize=14,
fontweight='bold')
ax.set_xticks(x)
ax.set_xticklabels(comparison_df['Feature'], rotation=45, ha='right')
ax.legend(fontsize=12)
ax.grid(axis='y', alpha=0.3)

# 在柱状图上添加数值标签
def add_value_labels(bars):
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.4f}',
                ha='center', va='bottom', fontsize=8)

add_value_labels(bars1)
add_value_labels(bars2)

plt.tight_layout()
plt.show()
```

```
#  4. 模型构建与超参数优化 随机森林回归 与 岭回归
print("\n" + "="*60)
print("4. 模型构建与超参数优化（随机森林+岭回归）")
print("="*60)

# 模型 1：随机森林回归
#  定义超参数网格
rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

#  网格搜索优化超参数
rf_grid = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=rf_param_grid,
    cv=5,
    scoring='r2',
    n_jobs=-1
)
rf_grid.fit(X_train_final, y_train)

#  最佳随机森林模型
best_rf = rf_grid.best_estimator_
print(f"\n 随机森林最佳超参数：")
print(rf_grid.best_params_)
# 4.1.4 随机森林预测
y_train_pred_rf = best_rf.predict(X_train_final)
y_test_pred_rf = best_rf.predict(X_test_final)

# 反归一化（转回原始尺度）
y_train_original = scaler_y.inverse_transform(y_train.reshape(-1,
1)).flatten()
y_train_pred_rf_original =
scaler_y.inverse_transform(y_train_pred_rf.reshape(-1, 1)).flatten()
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1,
1)).flatten()
y_test_pred_rf_original =
scaler_y.inverse_transform(y_test_pred_rf.reshape(-1, 1)).flatten()

#  随机森林性能评估
rf_train_rmse = np.sqrt(mean_squared_error(y_train_original,
y_train_pred_rf_original))
rf_train_r2 = r2_score(y_train_original, y_train_pred_rf_original)
```

```
rf_test_rmse = np.sqrt(mean_squared_error(y_test_original,
y_test_pred_rf_original))
rf_test_r2 = r2_score(y_test_original, y_test_pred_rf_original)

print(f"\n 随机森林性能指标：")
print(f"训练集 - RMSE：{rf_train_rmse:.6f} | R²：{rf_train_r2:.6f}")
print(f"测试集 - RMSE：{rf_test_rmse:.6f} | R²：{rf_test_r2:.6f}")


# 模型 2：岭回归
#  超参数优化（选择最佳 alpha）
alphas = 10 ** np.linspace(-5, 5, 100)
ridge_cv = RidgeCV(alphas=alphas, cv=5, scoring='r2')
ridge_cv.fit(X_train_final, y_train)

# 最佳岭回归模型
best_ridge = Ridge(alpha=ridge_cv.alpha_)
best_ridge.fit(X_train_final, y_train)
print(f"\n 岭回归最佳超参数（alpha）：{ridge_cv.alpha_:.6f}")
print(f"岭回归系数（各特征对腐蚀速率的影响程度）：")
for feat, coef in zip(final_features, best_ridge.coef_):
    print(f"  {feat}: {coef:.6f}")

#  岭回归预测
y_train_pred_ridge = best_ridge.predict(X_train_final)
y_test_pred_ridge = best_ridge.predict(X_test_final)

# 反归一化
y_train_pred_ridge_original =
scaler_y.inverse_transform(y_train_pred_ridge.reshape(-1, 1)).flatten()
y_test_pred_ridge_original =
scaler_y.inverse_transform(y_test_pred_ridge.reshape(-1, 1)).flatten()

# 岭回归性能评估
ridge_train_rmse = np.sqrt(mean_squared_error(y_train_original,
y_train_pred_ridge_original))
ridge_train_r2 = r2_score(y_train_original, y_train_pred_ridge_original)
ridge_test_rmse = np.sqrt(mean_squared_error(y_test_original,
y_test_pred_ridge_original))
ridge_test_r2 = r2_score(y_test_original, y_test_pred_ridge_original)

print(f"\n 岭回归性能指标：")
print(f"训练集 - RMSE：{ridge_train_rmse:.6f} | R²：{ridge_train_r2:.6f}")
print(f"测试集 - RMSE：{ridge_test_rmse:.6f} | R²：{ridge_test_r2:.6f}")

# 最佳模型选择
```

```
best_model = best_rf if rf_test_r2 > ridge_test_r2 else best_ridge
best_model_name = "随机森林" if rf_test_r2 > ridge_test_r2 else "岭回归"
best_test_r2 = max(rf_test_r2, ridge_test_r2)
best_test_rmse = min(rf_test_rmse, ridge_test_rmse)
print(f"\n 最佳模型：{best_model_name}")
print(f"最佳模型测试集性能：RMSE={best_test_rmse:.6f} | R²
={best_test_r2:.6f}")

#5. 模型性能可视化
print("\n" + "="*60)
print("5. 模型性能可视化（真实值 vs 预测值散点图）")
print("="*60)

#绘制模型性能对比图
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 7))

# 坐标轴范围
max_val = max(max(y_train_original), max(y_test_original))
min_val = min(min(y_train_original), min(y_test_original))
val_range = [min_val - 0.001, max_val + 0.001]

# 子图 1：随机森林性能
ax1.scatter(
    y_train_original, y_train_pred_rf_original,
    alpha=0.6, color='#1f77b4', label=f'训练集（R²={rf_train_r2:.4f}）'
)
ax1.scatter(
    y_test_original, y_test_pred_rf_original,
    alpha=0.6, color='#d62728', label=f'测试集（R²={rf_test_r2:.4f}）'
)
ax1.plot(val_range, val_range, 'k--', linewidth=2, label='理想拟合线
（y=x）')
ax1.set_xlabel('真实腐蚀速率（RATE）', fontsize=12)
ax1.set_ylabel('预测腐蚀速率（RATE）', fontsize=12)
ax1.set_title('随机森林：真实值 vs 预测值', fontsize=14, fontweight='bold')
ax1.legend(fontsize=10)
ax1.grid(alpha=0.3)
ax1.set_xlim(val_range)
ax1.set_ylim(val_range)

# 子图 2：岭回归性能
ax2.scatter(
    y_train_original, y_train_pred_ridge_original,
    alpha=0.6, color='#1f77b4', label=f'训练集（R²={ridge_train_r2:.4f}）'
)
ax2.scatter(
```

```
    y_test_original, y_test_pred_ridge_original,
    alpha=0.6, color='#d62728', label=f'测试集（R²={ridge_test_r2:.4f}）'
)
ax2.plot(val_range, val_range, 'k--', linewidth=2, label='理想拟合线
（y=x）')
ax2.set_xlabel('真实腐蚀速率（RATE）', fontsize=12)
ax2.set_ylabel('预测腐蚀速率（RATE）', fontsize=12)
ax2.set_title('岭回归：真实值 vs 预测值', fontsize=14, fontweight='bold')
ax2.legend(fontsize=10)
ax2.grid(alpha=0.3)
ax2.set_xlim(val_range)
ax2.set_ylim(val_range)

plt.tight_layout()
plt.show()

# 最佳模型特征重要性/系数可视化
if best_model_name == "随机森林":
    # 随机森林特征重要性
    best_rf_importance = pd.DataFrame({
        'Feature': final_features,
        'Importance': best_rf.feature_importances_
    }).sort_values(by='Importance', ascending=False)

    plt.figure(figsize=(12, 6))
    sns.barplot(x='Importance', y='Feature', data=best_rf_importance,
color='#2ca02c')
    plt.title(f'最佳模型 随机森林模型 特征重要性', fontsize=14,
fontweight='bold')
    plt.xlabel('特征重要性', fontsize=12)
    plt.ylabel('特征名称', fontsize=12)
    plt.grid(axis='x', alpha=0.3)
    plt.tight_layout()
    plt.show()
else:
    # 岭回归特征系数
    ridge_coef = pd.DataFrame({
        'Feature': final_features,
        'Coefficient': best_ridge.coef_
    }).sort_values(by='Coefficient', ascending=False)

    plt.figure(figsize=(12, 6))
    colors = ['#d62728' if c > 0 else '#1f77b4' for c in
ridge_coef['Coefficient']]
    sns.barplot(x='Coefficient', y='Feature', data=ridge_coef,
palette=colors)
```

```
    plt.axvline(x=0, color='k', linestyle='--', linewidth=1)
    plt.title(f'最佳模型 岭回归模型 特征系数（正=促进腐蚀，负=抑制腐蚀）',
fontsize=14, fontweight='bold')
    plt.xlabel('特征系数', fontsize=12)
    plt.ylabel('特征名称', fontsize=12)
    plt.grid(axis='x', alpha=0.3)
    plt.tight_layout()
    plt.show()

#  6. 过拟合检验曲线
print("\n" + "="*60)
print("6. 过拟合检验曲线")
print("="*60)

# 学习曲线函数
def plot_learning_curve(estimator, title, X, y, cv=5):
    plt.figure(figsize=(10, 6))

    train_sizes = np.linspace(0.1, 1.0, 10)
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, train_sizes=train_sizes,
        scoring='r2', n_jobs=-1, random_state=42
    )

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.plot(train_sizes, train_scores_mean, 'o-', color='#1f77b4',
             label='训练集 R 方', linewidth=2, markersize=6)
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
color='#1f77b4')

    plt.plot(train_sizes, test_scores_mean, 'o-', color='#d62728',
             label='验证集 R 方', linewidth=2, markersize=6)
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
color='#d62728')

    plt.xlabel('训练样本数量', fontsize=12)
    plt.ylabel('R 方得分', fontsize=12)
    plt.title(f'{title}学习曲线', fontsize=14, fontweight='bold')
    plt.legend(loc='best', fontsize=12)
    plt.grid(alpha=0.3)
```

```
    plt.tight_layout()
    plt.show()

# 误差曲线函数
def plot_error_curve(estimator, X_train, y_train, X_test, y_test,
model_name):
    plt.figure(figsize=(10, 6))

    train_errors = []
    test_errors = []
    train_sizes = np.linspace(0.1, 1.0, 10)

    for size in train_sizes:
        n_samples = int(len(X_train) * size)
        X_subset = X_train[:n_samples]
        y_subset = y_train[:n_samples]

        estimator_temp = clone(estimator)
        estimator_temp.fit(X_subset, y_subset)

        train_pred = estimator_temp.predict(X_subset)
        test_pred = estimator_temp.predict(X_test)

        train_rmse = np.sqrt(mean_squared_error(y_subset, train_pred))
        test_rmse = np.sqrt(mean_squared_error(y_test, test_pred))

        train_errors.append(train_rmse)
        test_errors.append(test_rmse)

    plt.plot(train_sizes * len(X_train), train_errors, 'o-',
color='#1f77b4',
             label='训练集 RMSE', linewidth=2, markersize=4)
    plt.plot(train_sizes * len(X_train), test_errors, 'o-',
color='#d62728',
             label='测试集 RMSE', linewidth=2, markersize=4)

    plt.xlabel('训练样本数量', fontsize=12)
    plt.ylabel('RMSE', fontsize=12)
    plt.title(f'{model_name}：训练误差 vs 测试误差', fontsize=14,
fontweight='bold')
    plt.legend(fontsize=12)
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.show()

# 绘制学习曲线
```

```
print("随机森林学习曲线：")
plot_learning_curve(best_rf, "随机森林", X_train_final, y_train)

print("岭回归学习曲线：")
plot_learning_curve(best_ridge, "岭回归", X_train_final, y_train)

print("最佳模型学习曲线：")
plot_learning_curve(best_model, f"最佳模型({best_model_name})",
X_train_final, y_train)

# 绘制误差曲线
print("随机森林误差曲线：")
plot_error_curve(best_rf, X_train_final, y_train, X_test_final, y_test, "
随机森林")

print("岭回归误差曲线：")
plot_error_curve(best_ridge, X_train_final, y_train, X_test_final, y_test,
"岭回归")

print("最佳模型误差曲线：")
best_model_for_plot = best_rf if best_model_name == "随机森林" else
best_ridge
plot_error_curve(best_model_for_plot, X_train_final, y_train,
X_test_final, y_test, f"最佳模型({best_model_name})")

#  7. SHAP 可解释性分析
print("\n" + "="*60)
print("7. SHAP 可解释性分析")
print("="*60)

try:
    import shap

    print("SHAP 分析如下：")

    #  使用最佳模型进行 SHAP 分析
    if best_model_name == "随机森林":
        # 随机森林的 SHAP 分析
        explainer = shap.TreeExplainer(best_model)
        shap_values = explainer.shap_values(X_test_final)

        # 创建 SHAP 摘要图
        plt.figure(figsize=(10, 8))
        shap.summary_plot(shap_values, X_test_final,
feature_names=final_features,
                          show=False, plot_size=(10, 8))
```

```
        plt.title(f'SHAP 特征重要性摘要图 - {best_model_name}',
fontsize=14, fontweight='bold')
        plt.tight_layout()
        plt.show()

        # 创建 SHAP 条形图（特征重要性排序）
        plt.figure(figsize=(10, 6))
        shap.summary_plot(shap_values, X_test_final,
feature_names=final_features,
                          plot_type="bar", show=False, plot_size=(10, 6))
        plt.title(f'SHAP 特征重要性条形图 - {best_model_name}',
fontsize=14, fontweight='bold')
        plt.tight_layout()
        plt.show()

        # 修复单个样本的 SHAP 力解释图（使用新版本的 API）
        plt.figure(figsize=(10, 6))
        # 创建 Explanation 对象
        explanation = shap.Explanation(values=shap_values[0],
                                       base_values=explainer.expected_value,
                                       data=X_test_final[0],
                                       feature_names=final_features)
        shap.waterfall_plot(explanation, show=False)
        plt.title('SHAP 力图', fontsize=14, fontweight='bold')
        plt.tight_layout()
        plt.show()

    else:
        # 岭回归的 SHAP 分析（使用 KernelExplainer）
        def model_predict(X):
            return best_model.predict(X)

        explainer = shap.KernelExplainer(model_predict,
X_train_final[:100])  # 使用部分训练数据作为背景
        shap_values = explainer.shap_values(X_test_final[:50])  # 使用部分
测试数据

        plt.figure(figsize=(10, 8))
        shap.summary_plot(shap_values, X_test_final[:50],
feature_names=final_features,
                          show=False, plot_size=(10, 8))
        plt.title(f'SHAP 特征重要性摘要图 - {best_model_name}',
fontsize=14, fontweight='bold')
        plt.tight_layout()
        plt.show()
```

```
    print("SHAP 分析完成！")

except ImportError:
    print("SHAP 库未安装，跳过 SHAP 分析。")
    print("如需使用 SHAP，请运行: pip install shap")
except Exception as e:
    print(f"SHAP 分析过程中出现错误: {e}")

#  8. 数据可视化
print("\n" + "="*60)
print("8. 数据可视化")
print("="*60)

#  特征相关性热力图
print("特征相关性热力图：")

# 使用筛选后的特征创建 DataFrame
df_final_features = pd.DataFrame(X_scaled,
columns=X.columns)[final_features]
df_final_features['RATE'] = y_scaled

# 计算相关性矩阵
correlation_matrix = df_final_features.corr()

plt.figure(figsize=(12, 10))
# 移除三角掩码，创建完整的正方形热力图
# 使用红色系颜色映射，vmin 和 vmax 设置合适的范围
sns.heatmap(correlation_matrix,
            annot=True,
            fmt=".2f",
            cmap='Reds',  # 使用红色系
            vmin=0,       # 最小值为 0
            vmax=1,       # 最大值为 1
            square=True,
            cbar_kws={"shrink": .8},
            linewidths=0.5,  # 添加网格线
            linecolor='white')  # 网格线颜色

plt.title('特征相关性热力图', fontsize=16, fontweight='bold', pad=20)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

#  箱线图 - 特征分布可视化
print("特征分布箱线图：")
```

```
# 选择前 8 个重要特征进行可视化（避免图表过于拥挤）
top_features = final_features[:8] if len(final_features) >= 8 else
final_features

fig, axes = plt.subplots(2, 4, figsize=(20, 10))
axes = axes.ravel()

for i, feature in enumerate(top_features):
    if i < len(axes):
        sns.boxplot(y=df_final_features[feature], ax=axes[i],
color='skyblue')
        axes[i].set_title(f'{feature} 分布', fontsize=12,
fontweight='bold')
        axes[i].set_ylabel('归一化值' if feature != 'RATE' else '腐蚀速率
')
        axes[i].grid(axis='y', alpha=0.3)

# 隐藏多余的子图
for i in range(len(top_features), len(axes)):
    axes[i].set_visible(False)

plt.suptitle('重要特征分布箱线图', fontsize=16, fontweight='bold', y=0.98)
plt.tight_layout()
plt.show()

# 目标变量分布直方图
print("目标变量分布：")

plt.figure(figsize=(12, 5))

# 子图 1：归一化后的目标变量分布
plt.subplot(1, 2, 1)
plt.hist(y_scaled, bins=30, alpha=0.7, color='steelblue',
edgecolor='black')
plt.xlabel('归一化腐蚀速率')
plt.ylabel('频数')
plt.title('归一化目标变量分布', fontsize=12, fontweight='bold')
plt.grid(alpha=0.3)

# 子图 2：原始目标变量分布
plt.subplot(1, 2, 2)
plt.hist(y, bins=30, alpha=0.7, color='coral', edgecolor='black')
plt.xlabel('原始腐蚀速率')
plt.ylabel('频数')
plt.title('原始目标变量分布', fontsize=12, fontweight='bold')
```

```
plt.grid(alpha=0.3)

plt.suptitle('腐蚀速率分布分析', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()

# 特征与目标变量的散点图矩阵（选择最重要的 4 个特征）
print("特征-目标变量关系散点图：")

important_features = final_features[:4] if len(final_features) >= 4 else
final_features
if len(important_features) > 1:
    scatter_features = important_features + ['RATE']
    scatter_df = df_final_features[scatter_features]

    # 创建散点图矩阵
    fig = sns.pairplot(scatter_df,
                       diag_kind='hist',
                       plot_kws={'alpha': 0.6, 's': 30},
                       diag_kws={'alpha': 0.7, 'bins': 20})

    fig.fig.suptitle('重要特征与腐蚀速率关系散点图矩阵',
                     y=1.02, fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()

print("\n 数据探索可视化完成！")

# 9. 结果汇总与保存
print("\n" + "="*60)
print("9. 结果汇总")
print("="*60)


# 整理模型性能结果
result_summary = pd.DataFrame({
    '模型名称': ['随机森林', '岭回归'],
    '最佳超参数': [
        str(rf_grid.best_params_),
        f"alpha={ridge_cv.alpha_:.6f}"
    ],
    '训练集 RMSE': [rf_train_rmse, ridge_train_rmse],
    '训练集 R²': [rf_train_r2, ridge_train_r2],
    '测试集 RMSE': [rf_test_rmse, ridge_test_rmse],
    '测试集 R²': [rf_test_r2, ridge_test_r2],
    '是否最佳模型': ['是' if rf_test_r2 > ridge_test_r2 else '否', '是' if
```

```
rf_test_r2 <= ridge_test_r2 else '否']
})

print("\n 模型性能结果汇总：")
print(result_summary)

# 最终筛选特征列表
final_features_df = pd.DataFrame({
    '最终输入特征': final_features,
    '特征类型': [
        '元素成分' if feat in ['C', 'Si', 'Mn', 'P', 'S', 'Cu', 'Cr',
'Ni', 'Cl'] else '环境参数'
        for feat in final_features
    ]
})

print("\n 最终筛选特征列表：")
print(final_features_df)

# 输出最佳模型名称
print(f"\n最终确定的最佳模型为：{best_model_name}模型")

#  10. 最佳模型预测示例
print(f"\n" + "="*60)
print("10. 最佳模型预测示例（3 个测试集样本）")
print("="*60)

# 随机选择 3 个测试集样本
np.random.seed(42)
sample_idx = np.random.choice(len(X_test_final), 3, replace=False)
sample_X = X_test_final[sample_idx]
sample_y_original = y_test_original[sample_idx]

# 最佳模型预测
sample_y_pred = best_model.predict(sample_X)
sample_y_pred_original =
scaler_y.inverse_transform(sample_y_pred.reshape(-1, 1)).flatten()

# 输出预测结果
print(f"\n最佳模型 {best_model_name}模型 的测试集样本预测结果：")

for i in range(3):
    error = abs(sample_y_original[i] - sample_y_pred_original[i])
    print(f"样本{i+1}：")
    print(f"  真实腐蚀速率：{sample_y_original[i]:.6f}")
    print(f"  预测腐蚀速率：{sample_y_pred_original[i]:.6f}")
```

```
    print(f"  绝对误差：{error:.6f}")
    print(f"  相对误差：{error/sample_y_original[i]*100:.2f}%")
    print()
```