Supporting Information
Title: Ensemble Machine Learning for CO_2 Corrosion Rate Prediction with Heterogeneous Datasets
Authors list

Joan Ejeta+
Department of Computational Data Science and Engineering
North Carolina A&T State University
Greensboro, NC 27411, USA
Email: joejeta@aggies.ncat.edu

Tolu Emiola-Sadiq +
Department of Chemical and Biological Engineering
University of Saskatchewan
Saskatoon, S7N 5A9, Canada
Email: tolu.emiola@usask.ca

Robert Eshun +
Department of Computational Data Science and Engineering
North Carolina A&T State University
Greensboro, NC 27411, USA
Email: rbeshun@ncat.edu

Kristen Rhinehardt *
Department of Computational Data Science and Engineering
North Carolina A&T State University
Greensboro, NC 27411, USA
Email: klrhineh@ncat.edu

*corresponding author
+ these authors contributed equally to this work

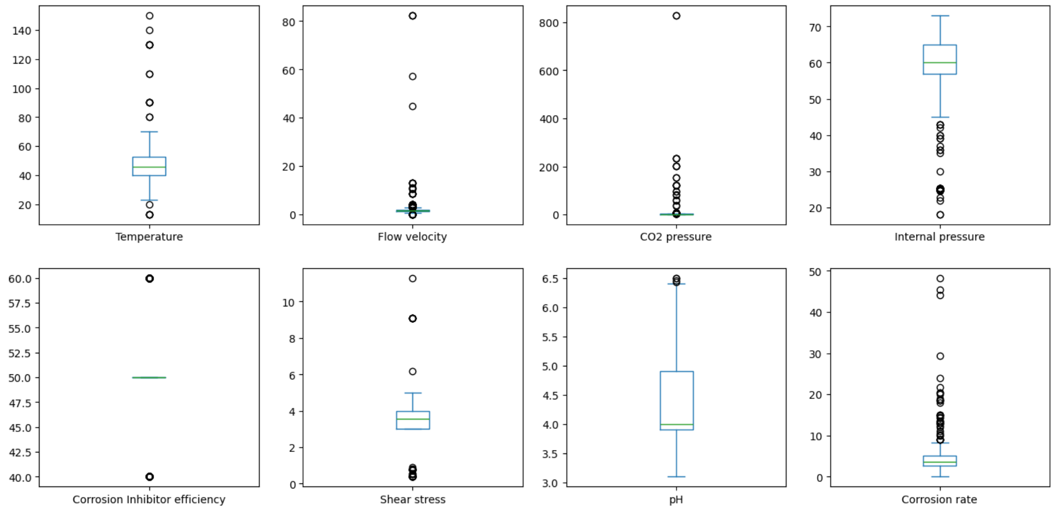

Figure 1: Box plot before outlier removal

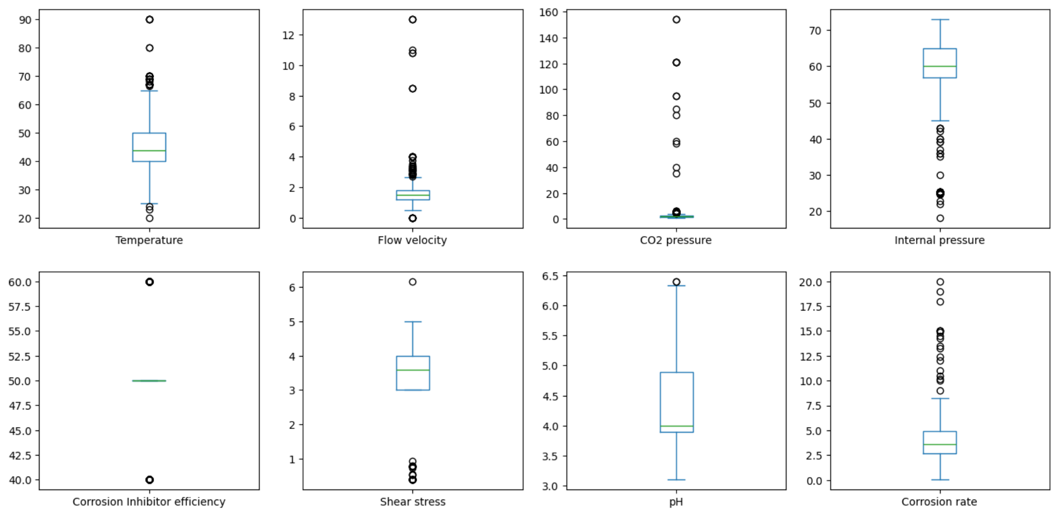

Figure 2: Box plot after outlier removal

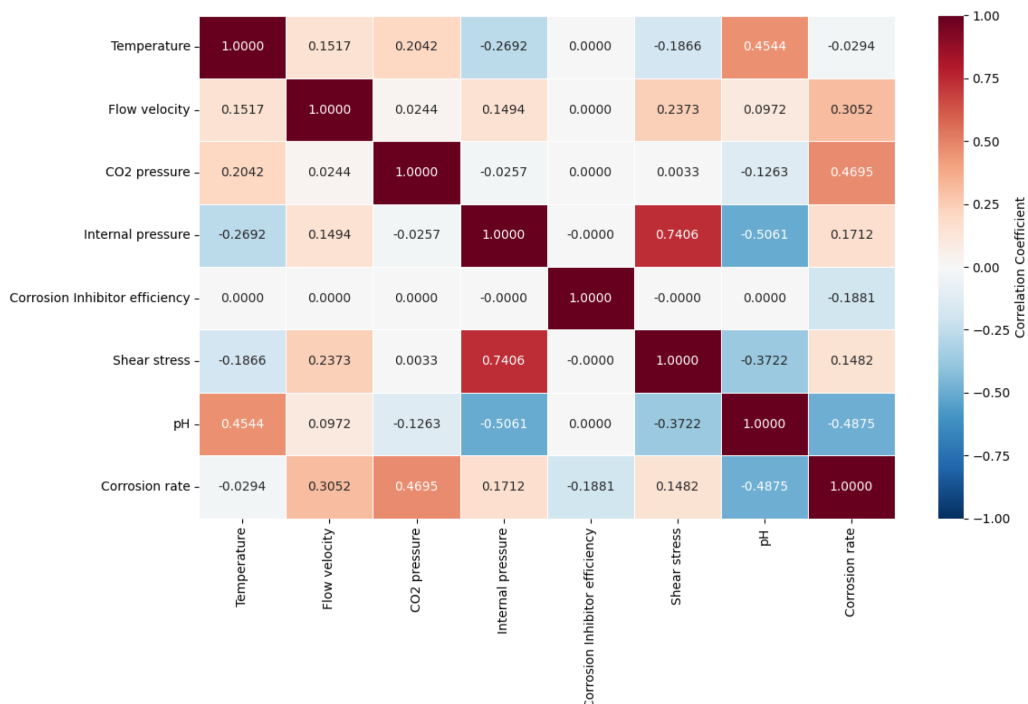

Figure 3: Correlation Matrix before SMOGN Class Imbalance

```python
# Define the three columns to impute
columns_to_impute = [
    'Corrosion Inhibitor',
    'Internal pressure',
    'Shear stress'
]

# Extract column names (handling typos/variations)
selected_columns = []
for col_pattern in columns_to_impute:
    matching_cols = df.columns[df.columns.str.contains(col_pattern, case=False)]
    if len(matching_cols) > 0:
        selected_columns.append(matching_cols[0])
    else:
        print(f"Warning: Column matching '{col_pattern}' not found")

print(f"Columns to impute: {selected_columns}")

# Replace missing values with initial estimates (mean)
imputer = SimpleImputer(strategy='mean')
data_imputed = imputer.fit_transform(df[selected_columns])

# Define the range for the number of components to test
n_components_range = range(1, 18)
bic_scores = []
aic_scores = []

# Fit GMM for each number of components and record BIC and AIC
for n_components in n_components_range:
    gmm = GaussianMixture(n_components=n_components, max_iter=100, random_state=42)
    gmm.fit(data_imputed)
    bic_scores.append(gmm.bic(data_imputed))
    aic_scores.append(gmm.aic(data_imputed))
```

Figure 4a: Screenshot of Code Showing the Criteria for Handling Missing Data Using EM and Simple Imputer

```python
# Plot BIC and AIC scores
plt.figure(figsize=(10, 5))
plt.plot(n_components_range, bic_scores, label='BIC', marker='o')
plt.plot(n_components_range, aic_scores, label='AIC', marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Score')
plt.legend()
plt.title('BIC and AIC Scores for GMM (3 Columns)')
plt.show()

# Determine optimal number of components
optimal_bic_components = n_components_range[np.argmin(bic_scores)]
optimal_aic_components = n_components_range[np.argmin(aic_scores)]

print(f"Optimal number of components based on BIC: {optimal_bic_components}")
print(f"Optimal number of components based on AIC: {optimal_aic_components}")

# Fit GMM with optimal components and impute missing values
optimal_components = optimal_bic_components
gmm_optimal = GaussianMixture(n_components=optimal_components, max_iter=100, random_state=42)
gmm_optimal.fit(data_imputed)

# Create a copy of the dataframe with imputed values
df_imputed = df.copy()
df_imputed[selected_columns] = data_imputed

print("\nImputation complete!")
print(f"Shape of imputed data: {df_imputed[selected_columns].shape}")
```

Figure 4b: Screenshot Of Codes Showing the Criteria for Handling Missing Data Using EM and Simple Imputer

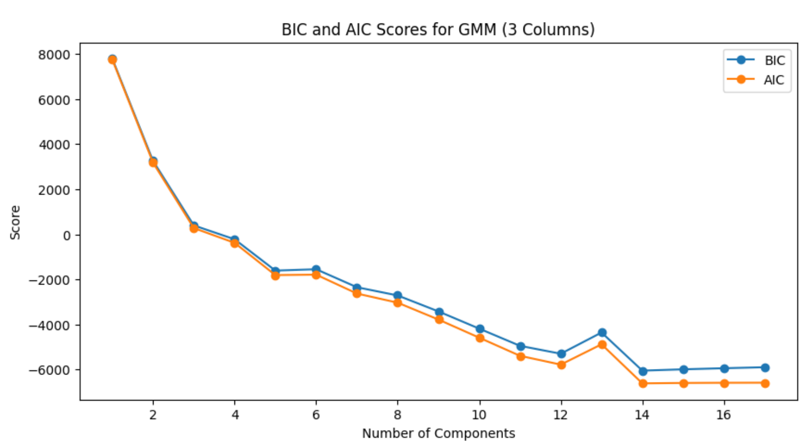

Figure 5: Stopping Criteria for the BIC and AIC Using EM for Missing Data Handling

```python
## load libraries
import smogn
import pandas
import seaborn

# Separate features and target
X = cleaned_df.drop(columns=['Corrosion rate'])
y = cleaned_df['Corrosion rate']

## conduct smogn
cleaned_df_smogn = smogn.smoter(
    data = cleaned_df,
    y = "Corrosion rate",
    k = 7,                      ## positive integer (k < n)
    pert = 0.04,                ## real number (0 < R < 1)
    samp_method = 'balance',    ## string ('balance' or 'extreme')

    ## phi relevance arguments
    rel_thres = 0.20,           ## positive real number (0 < R < 1)
    rel_method = 'auto',        ## string ('auto' or 'manual')
    rel_xtrm_type = 'both',     ## string ('low' or 'both' or 'high')
    rel_coef = 2.5
)
print("Original samples:", len(cleaned_df))
print("Resampled samples:", len(cleaned_df_smogn))
```

Figure 6: Screenshot of Codes for Handling Class Imbalance Using SMOGN

Compute Comparison for Ensemble Learning Models

Dataset Specifications

This analysis evaluates computational efficiency for predicting CO_2 corrosion rates using ensemble learning methods. After preprocessing (imputation, outlier removal, and SMOGN rebalancing), the final dataset contains **n = 320 observations** with **d = 4 features** (CO_2 partial pressure, pH, flow velocity, and temperature).

Parameters:

- $n = 320$, $d = 4$, $\log_2 n \approx 8.32$
- For ensembles: $k = 100$–400 trees (varies by model after hyperparameter tuning)
- Boosting rounds $M = 100$–400
- Average tree depths: forests $h \approx 7$–10; boosting trees $h \approx 5$–7
- m = features tried per split; with $d = 4$, typical $m = \sqrt{d} \approx 2$

Computational Complexity Comparison Table

Random Forest Regressor (RFR)

Configuration: k=400, max_depth=90

- **Train Complexity:** $O(k \cdot n \cdot m \cdot \log n) \rightarrow (400 \cdot 320 \cdot 2 \cdot 8.32) \approx$ **2.13M units**
- **Predict Cost (per sample):** $O(k \cdot h) \approx$ **36,000** tests (assuming $h \approx 90$)
- **Memory (model only): ~12–20 MB**
- **Parallelism/Notes:** Embarrassingly parallel across trees; strong baseline accuracy

Gradient Boosting Regressor (GBR)

Configuration: M=400, max_depth=7

- **Train Complexity:** $O(M \cdot n \cdot d \cdot \log n) \rightarrow (400 \cdot 320 \cdot 4 \cdot 8.32) \approx$ **4.25M units**
- **Predict Cost (per sample):** $O(M \cdot h) \approx$ **2,800** tests
- **Memory (model only): ~0.5–1.2 MB**
- **Parallelism/Notes:** Sequential across M rounds; shallow trees reduce overfitting

Extreme Gradient Boosting (XGBR)

Configuration: M=100, max_depth default

- **Train Complexity:** $O(M \cdot n \cdot m \cdot B)$ (histogram-based) $\rightarrow (100 \cdot 320 \cdot 2 \cdot 256) \approx$ **16.4M "bin ops"** (cache-friendly)
- **Predict Cost (per sample):** $O(M \cdot h) \approx$ **500–700** tests
- **Memory (model only): ~0.4–1.0 MB**
- **Parallelism/Notes:** Multicore/GPU support; regularization (L1/L2) adds minimal cost

Extra Trees Regressor (ETR)

Configuration: k=50, max_features=200, min_samples_leaf=5

- **Train Complexity:** $O(k \cdot n \cdot m \cdot h) \rightarrow (50 \cdot 320 \cdot 2 \cdot 9) \approx$ **0.29M units**
- **Predict Cost (per sample):** $O(k \cdot h) \approx$ **450** tests
- **Memory (model only): ~2–4 MB**
- **Parallelism/Notes:** Faster training than RFR (random thresholds); less prone to overfitting

AdaBoost Regressor

Configuration: base: DecisionTreeRegressor, M=50–100

- **Train Complexity:** $O(M \cdot n \cdot d \cdot \log n)$ + reweighting $\rightarrow (100 \cdot 320 \cdot 4 \cdot 8.32) \approx$ **1.06M units**
- **Predict Cost (per sample):** $O(M \cdot h) \approx$ **500–900** tests
- **Memory (model only): ~0.3–0.8 MB**
- **Parallelism/Notes:** Sequential; focuses on hard samples; sensitive to noise/outliers

CatBoost Regressor

Configuration: M=100–300, depth=6–10

- **Train Complexity:** $O(M \cdot n \cdot d \cdot \log n)$ (ordered boosting + symmetric trees) → similar to GBDT but with overhead for categorical handling
- **Predict Cost (per sample):** $O(M \cdot h) \approx$ **600–2,100** tests
- **Memory (model only): ~0.5–1.5 MB**
- **Parallelism/Notes:** Built-in categorical encoding; GPU support; symmetric trees improve consistency

Practical Implications for 320-Sample Corrosion Dataset

Training Efficiency

- **Fastest training:** ETR (random splits, no search for optimal thresholds) < XGBR/GBR (histogram-based, efficient caching) < RFR (must search for best splits) < AdaBoost (sequential, reweighting overhead)
- **All models train in seconds to minutes** on this dataset size, making hyperparameter tuning via GridSearchCV feasible
- **CatBoost** adds preprocessing overhead but eliminates manual feature engineering for categorical variables (not applicable here with continuous features only)

Model Size & Deployment

- **Smallest models:** AdaBoost/GBR/XGBR (hundreds of KB with shallow trees) < ETR (2–4 MB with 50 trees) < RFR (12–20 MB with 400 deep trees)
- **RFR's large model size** reflects max_depth=90 from hyperparameter tuning, creating very deep trees
- For embedded systems or edge deployment, **boosting methods** are preferable due to compact representation

Inference Speed

- **Fastest prediction:** Shallow boosting ensembles (GBR: 2,800 tests, XGBR: 500–700 tests) < ETR (450 tests) < RFR (36,000 tests due to depth)
- **RFR's deep trees** (max_depth=90) significantly increase prediction latency
- For real-time corrosion monitoring, **GBR or ETR** offer best speed-accuracy trade-off

Model-Specific Considerations

Random Forest (RFR)

- Highest computational cost due to deep trees (max_depth=90)
- Best for offline analysis where training time is not critical
- Strong resistance to overfitting through averaging

Gradient Boosting (GBR)

- Optimal balance: R^2_test = 0.700, minimal overfitting ($\Delta R^2 = 0.165$)
- Sequential training limits parallelism but produces compact models
- Ideal for production deployment

XGBoost (XGBR)

- Despite advanced features (regularization, pruning), showed overfitting ($\Delta R^2 = 0.313$)
- Requires careful hyperparameter tuning for small heterogeneous datasets
- Hardware acceleration (GPU) overkill for n=320

Extra Trees (ETR)

- Most generalizable ($\Delta R^2 = 0.114$) with fastest training
- Random threshold selection reduces variance
- Recommended for initial exploration and cross-validation

AdaBoost

- Poor performance on this dataset (highest errors)
- Sensitivity to outliers problematic even after preprocessing
- Not recommended for noisy corrosion data

CatBoost

- Moderate performance; symmetric trees increase stability
- Automatic handling of missing values (redundant after GMM-EM imputation)
- Useful if dataset expansion includes categorical variables (e.g., pipeline material grades)

Corrosion-Specific Recommendations

1. Start with GBR (Validated Optimal)

Configuration: learning_rate=0.2, max_depth=7, n_estimators=400

- Best test performance (R^2=0.700) with reasonable training time
- Captures non-linear interactions (CO_2 pressure × temperature, pH × flow velocity)

2. Use ETR for Robustness Checks

- Minimal overfitting makes it ideal for uncertainty quantification
- Fast k-fold cross-validation (k=3 used in study)

3. Leverage Monotonic Constraints (GBR/XGBR)

- Expected trends: CO_2 pressure ↑ → corrosion rate ↑, pH ↑ → corrosion rate ↓
- Improves physical interpretability without asymptotic cost increase

4. Avoid RFR for Real-Time Applications

- Despite R^2_train=0.881, max_depth=90 creates impractical inference latency
- Reserve for comprehensive offline integrity assessments

5. Hyperparameter Tuning is Critical

- 3-fold CV with GridSearchCV increased computational cost by ~3× but prevented overfitting
- Total training time still <10 minutes for all models on standard hardware

6. Future Scaling Considerations

- For expanded datasets (n>1000), histogram-based methods (XGBR, CatBoost) become increasingly advantageous
- Consider LightGBM for massive datasets (not evaluated in current study)

Summary

For this **320-sample, 4-feature CO_2 corrosion prediction task**, all ensemble methods train efficiently (seconds to minutes). **Model selection should prioritize generalization over raw training speed**.

The study identified **Gradient Boosting Regressor** as optimal, balancing:

- Strong predictive accuracy (R^2_test = 0.700)
- Minimal overfitting (ΔR^2 = 0.165)
- Compact model size (~0.5–1.2 MB)
- Reasonable inference speed (~2,800 node tests per prediction)

This computational profile supports both **offline integrity management** (comprehensive risk assessment) and **near-real-time monitoring** (inspection interval optimization) for oil and gas pipeline systems.

Performance Metrics Summary

Model	R^2_train	R^2_test	ΔR^2 (Overfitting)	Training Time	Model Size
GBR	0.865	0.700	0.165	Moderate	Small
ETR	0.779	0.665	0.114	Fast	Medium
RFR	0.881	0.646	0.235	Slow	Large
XGBR	0.901	0.588	0.313	Fast	Small

Recommendation: Use **GBR** for production deployment and **ETR** for model validation and uncertainty analysis.