Predictive Modeling of Steel Industry Energy Consumption

Kishan Kumar Upadhyay
Guide: `Prof. Urban Larsson`

Nov 15, 2024

Abstract

This report presents an end-to-end machine learning pipeline to predict energy consumption (`Usage_kWh`) in steel manufacturing. The work covers problem definition, dataset description, exploratory data analysis, preprocessing (missing values, outliers, encoding), feature engineering, selection, model benchmarking, tuning, evaluation, and deployment considerations. Key results, limitations and recommended next steps are discussed.

Contents

1 Introduction

Energy efficiency is critical in the steel industry both for cost savings and environmental impact. Accurate short-term and medium-term forecasting of plant energy consumption enables better operational planning, load balancing, predictive maintenance and demand-side management. This project focuses on building a robust regression model to predict energy consumption (Usage_kWh) using plant telemetry and operational features.

2 Problem Statement

2.1 Background

Steel manufacturing is energy intensive. Plants generate a wealth of sensor and process data (power, voltages, currents, temperatures, load types, operational states). Translating this telemetry into reliable energy forecasts supports process optimization and decarbonization efforts.

2.2 Objective

Formally, given historical plant telemetry and auxiliary features $X = \{x_1, x_2, ..., x_n\}$, predict the continuous target variable $y =$ `Usage_kWh` for a future time-step. The model should generalize across operational regimes, be robust to missing data and outliers, and provide actionable performance metrics (RMSE, MAE, R^2).

2.3 Scope and Success Criteria

- Build an end-to-end pipeline from data ingestion to model artifact saving.
- Achieve stable hold-out performance with low RMSE and high R^2 compared to baseline linear models.
- Provide interpretable feature importance and reproducible training artifacts.

3 Dataset

3.1 Description

The dataset (`Steel_industry_data.csv`) contains time-indexed plant telemetry and operational variables. The most important column is `Usage_kWh` (continuous target). Other columns include reactive/active power measurements, power factor, load type and potential categorical identifiers.

3.2 Data Quality Notes

- Missing values: numerical and categorical missingness handled via median and mode imputation respectively.

- Outliers: detected using IQR; outliers are capped to preserve data while avoiding extreme influence.
- Temporal aspects: date column removed if not useful; lag and rolling features are engineered to capture temporal dependencies.

4 Exploratory Data Analysis

Summarize distributions, correlations and time patterns. Typical EDA steps implemented:

- Univariate summaries for all numeric variables (mean, median, skewness).
- Missingness heatmap and counts.
- Boxplots to identify outliers and support the choice of IQR capping.
- Pairwise correlation matrix to detect multicollinearity and important predictors.
- Target distribution analysis to understand heteroscedasticity or heavy tails.

5 Preprocessing

5.1 Missing Value Treatment

- Numerical columns imputed with median using `SimpleImputer(strategy='median')`.
- Categorical columns (if any) imputed with most frequent value.

5.2 Outlier Handling

Outliers detected using the IQR method were capped at the 1.5*IQR fences to reduce undue influence while retaining observations.

5.3 Encoding Categorical Variables

- Ordinal-like features (e.g. `Load_Type`) encoded with label encoding.
- Nominal features one-hot encoded with drop-first to avoid dummy variable trap.

5.4 Feature Scaling and Transformation

StandardScaler applied to ensure zero mean and unit variance. Yeo-Johnson power transform used to reduce skew and better satisfy algorithmic assumptions.

6 Feature Engineering

Key engineered features included:

- Lag features: previous-step values (e.g., column_lag1) to capture temporal dependence.
- Rolling statistics: short-window rolling mean and standard deviation (window=3) to capture local trends.
- Interaction features: totals and ratios (e.g., total reactive power, power factor ratio) to capture domain-informed relationships.

7 Feature Selection

Two complementary methods were used to select a compact, informative feature set:

- Mutual Information (SelectKBest with `mutual_info_regression`) to capture non-linear dependencies.
- Recursive Feature Elimination (RFE) with a Random Forest base estimator to account for interactions.

The union of both methods produced the final feature set used for modeling.

8 Modeling Strategy

8.1 Candidate Models

Benchmarked a diverse set of regressors: Linear Regression, Ridge, Lasso, Decision Tree, Random Forest, Gradient Boosting, XGBoost, KNN, and SVR.

8.2 Cross-Validation and Metric

Cross-validation used (5 folds). Primary metric: RMSE (root mean squared error). Secondary: MAE and R^2. For cross-validation, negative RMSE scoring was used then negated to obtain positive RMSE.

8.3 Selection and Tuning

The best performing model on CV RMSE was selected and, when applicable, hyperparameter tuning via GridSearchCV applied on a reduced parameter grid to balance compute and performance.

9 Evaluation

Report training and test metrics (MSE, RMSE, MAE, R^2). Typical evaluation steps:

1. Train best/tuned model on full training data.

2. Predict on training and hold-out test sets.
3. Report and compare metrics to detect overfitting (difference in R^2 greater than 0.1 flagged).
4. Visual diagnostics: predicted vs actual scatter, residual histogram and residuals vs fitted.

10 Results

Summarize best model name, tuned parameter set (if tuned), and final test metrics. For example (replace with actual experiment outputs):

- Best model: XGBoost (after tuning)
- Test RMSE: 123.45
- Test MAE: 98.76
- Test R^2: 0.87

Also include the top-10 feature importances table or plot when available.

11 Discussion

Interpretation points:

- Which features most influence energy consumption (e.g., active/reactive power readings, power factor ratio, load type).
- Model stability across operational regimes and possible causes of error (sensor drift, unobserved control actions).
- Data limitations (temporal granularity, missing context about maintenance/events) and how they affect forecasts.

12 Deployment Considerations

For productionizing the model consider:

- Packaging preprocessing and model artifacts together (example: joblib dump of scaler, transformers and model).
- Serving via a lightweight API (FastAPI/Flask) or embedding in edge controllers for on-prem inference.
- Monitoring: data drift detection, periodic retraining schedule, and alerting for anomalous predictions.
- Latency and memory constraints if deploying on edge devices.

13 Limitations and Future Work

- Incorporate richer temporal modeling (e.g., sequence models, state-space models or Prophet) for longer horizon forecasting.
- Explore advanced ensembling and stacking to improve robustness.
- Integrate external covariates (weather, electricity tariff, production schedule) to reduce unexplained variance.
- Use probabilistic forecasting to quantify uncertainty (e.g., quantile regression, Bayesian models).

14 Conclusion

This project implements a reproducible and interpretable ML pipeline for predicting steel industry energy consumption. Through careful preprocessing (imputation, outlier capping), domain-informed feature engineering (lags, rolling statistics, interaction ratios), and rigorous model selection (mutual information, RFE, cross-validation), the pipeline yields a performant regression model with actionable metrics (RMSE, MAE, R^2). The final artifacts (trained model and transformers) are ready for deployment, with recommendations on monitoring and future enhancements to improve accuracy and reliability.

Appendix A: Key Code Snippets

Data Imputation and Outlier Capping

```python
from sklearn.impute import SimpleImputer
import numpy as np

imputer = SimpleImputer(strategy='median')
df[numeric_cols] = imputer.fit_transform(df[numeric_cols])

# IQR capping
for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    df[col] = np.clip(df[col], lower, upper)
```

Listing 1: Imputation and IQR capping

Feature Selection Combination

```
from sklearn.feature_selection import SelectKBest, mutual_info_regression, RFE
from sklearn.ensemble import RandomForestRegressor

selector_mi = SelectKBest(mutual_info_regression, k=15)
X_mi = selector_mi.fit_transform(X, y)

rf = RandomForestRegressor(n_estimators=100, random_state=42)
rfe = RFE(rf, n_features_to_select=15)
rfe.fit(X, y)

final_features = list(set(X.columns[selector_mi.get_support()]) | set(X.columns
    [rfe.support_]))
```

Listing 2: Mutual information + RFE

References

- Hastie, T., Tibshirani, R., Friedman, J. (2009). The Elements of Statistical Learning.
- Molnar, C. (2019). Interpretable Machine Learning.
- Documentation: scikit-learn, XGBoost.