Predicting Superconducting Critical Temperature with Classical ML

Panita Vaishnavi Kanaguduru
University of Central Florida
STA 5703: Data Mining I
panita.vaishnavi@ucf.edu

***Abstract*—We study the supervised prediction of the critical temperature (T_c) of superconductors from engineered descriptors derived from chemical formulas. Using the UCI Superconductivity dataset (21,263 materials; 81 numeric features), we compare linear regression, Ridge regression, and Gradient Boosting under a unified preprocessing and evaluation protocol. Exploratory analysis confirms a right-skewed target distribution and reveals strong associations between T_c and thermal/valence descriptors. With a simple 3-fold cross-validation on the training set, Gradient Boosting consistently outperforms linear baselines; on a held-out test split, it achieves RMSE $\approx$ 11.6 and $R^2 \approx 0.88$. Feature-importance analysis highlights thermal conductivity range/dispersion and atomic radius as primary drivers. A short learning-curve study indicates diminishing returns beyond $\sim$10k samples. These observations align with and compactly reproduce key findings reported by Hamidieh (2018), while offering a lightweight workflow suitable for classroom projects.**

***Index Terms*—Superconductivity, Regression, Gradient Boosting, Materials Informatics, Model Evaluation**

I. Introduction

Superconductivity—the complete loss of electrical resistance below a critical temperature T_c—offers transformative technological potential in high-performance power transmission, magnetic levitation, particle accelerators, and quantum devices. Identifying compounds with elevated T_c is therefore a central challenge in materials science. Yet experimental discovery is expensive and slow; the chemical design space is effectively unbounded.

Data-driven modeling seeks to accelerate this search by learning mappings from readily computed descriptors (e.g., elemental statistics, valence, and thermal properties) to T_c. In a seminal data-mining study, Hamidieh [1] curated a large tabular dataset and established that tree ensembles (e.g., Gradient Boosting) substantially outperform naïve linear models for predicting T_c. Follow-on work across materials informatics has repeatedly found that nonlinear models leveraging interactions among descriptors improve accuracy and provide explanatory signals through feature importance.

This report presents a compact reproduction and extension of the classical baselines for T_c prediction using a modern, fast scikit-learn pipeline. Our contributions are: (i) a transparent preprocessing/evaluation protocol suitable for reproducible coursework; (ii) a focused comparison between linear, regularized linear, and Gradient Boosting models; (iii) explanatory analysis via correlation screening, feature importance, and a short learning-curve study.

II. Data and Methods

A. Dataset

We use the UCI Superconductivity dataset introduced in [1]. The table comprises 21,263 materials described by 81 numeric features derived from their chemical formulas, and a continuous target T_c (Kelvin). Features include counts/statistics over elemental properties (e.g., atomic mass, atomic radius, valence, electron affinity) as well as thermal conductivity descriptors (means, ranges, entropies, and weighted variants). In our copy, the data is provided as `train.csv` inside a compressed archive.

B. Preprocessing

All features are numeric. We apply a simple, consistent preprocessing pipeline: *median imputation* for missing values followed by *standardization* (zero mean, unit variance). We split the data into training (80%) and testing (20%) using a fixed random seed for reproducibility. Model selection uses only the training split via cross-validation; the test split is untouched until the final evaluation.

C. Models

We compare three representative regressors:

- **Linear Regression** (OLS). A transparent baseline assuming linear additive effects.
- **Ridge Regression**. A Tikhonov-regularized linear model that mitigates multicollinearity prevalent in descriptor families [4].
- **Gradient Boosting Regressor (GBR)**. An additive ensemble of shallow decision trees that captures nonlinearities and interactions with strong out-of-the-box performance on tabular data [2].

Hyperparameters are intentionally kept compact to satisfy runtime constraints of the project: Ridge uses a small log-grid over α with internal CV; GBR uses 200 trees, depth 3, learning rate 0.1. Implementations are from scikit-learn [3].

D. Evaluation Protocol

Within the training set, we perform *3-fold cross-validation* and report the mean and standard deviation of root mean squared error (RMSE). After selecting the best model by mean CV RMSE, we refit it on the full training split and estimate generalization on the hold-out test split using RMSE, mean

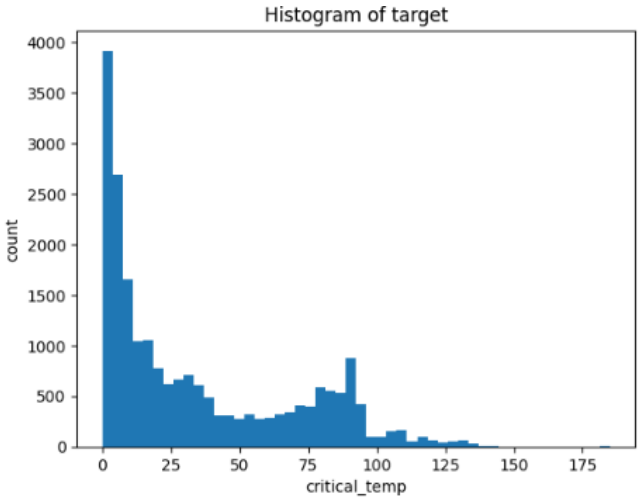

Fig. 1. Distribution of critical temperature (T_c). Most compounds have modest T_c, with a long high-temperature tail.

TABLE I
TOP CORRELATED DESCRIPTORS WITH T_c (ABSOLUTE CORRELATION).

Feature	|corr|
`wtd_std_ThermalConductivity`	0.72
`range_ThermalConductivity`	0.69
`range_atomic_radius`	0.65
`std_ThermalConductivity`	0.65
`wtd_mean_Valence`	0.63
`wtd_entropy_atomic_mass`	0.63
`wtd_gmean_Valence`	0.62
`wtd_entropy_atomic_radius`	0.60
`number_of_elements`	0.60
`range_fie`	0.60

absolute error (MAE), and R^2. We also report (i) a univariate correlation screen between features and T_c to provide intuition, (ii) impurity-based feature importances from the best tree ensemble, and (iii) a short learning-curve diagnostic (four subsample sizes) to assess the bias–variance regime.

III. RESULTS

A. Exploratory Data Analysis

Figure 1 shows a right-skewed distribution of T_c with a long tail: the majority of compounds lie below 50 K, whereas relatively few exceed 100 K.

Table I lists ten descriptors with the strongest absolute Pearson correlations with T_c. Thermal-conductivity dispersion statistics dominate (e.g., weighted standard deviation and range), followed by atomic radius and valence descriptors.

B. Cross-Validation Comparison

Table II summarizes 3-fold CV results on the training split. Gradient Boosting attains a mean RMSE near 11.9, substantially better than linear and Ridge baselines (both about 17.7), reflecting the benefit of modeling nonlinearities and interactions among descriptors.

TABLE II
3-FOLD CV RMSE ON THE TRAINING SPLIT. MEAN (STD).

Model	**Mean RMSE**	**Std**
Gradient Boosting	11.90	0.05
Linear Regression	17.70	0.28
Ridge Regression	17.70	0.29

TABLE III
HOLD-OUT TEST PERFORMANCE (BEST MODEL: GRADIENT BOOSTING).

Model	**RMSE**	**MAE**	$\boldsymbol{R^2}$
Gradient Boosting	11.59	7.72	0.885

C. Hold-Out Test Performance

After refitting on the full training split, Gradient Boosting yields the metrics in Table III. The R^2 of ≈ 0.88 indicates that the model explains the bulk of the variance in T_c on unseen compounds under this split.

D. Feature Importance

Figure 2 displays the top ten features by GBR importance. The *range of thermal conductivity* is the most influential, followed by *range of atomic radius* and several thermal-conductivity dispersion/mean descriptors and valence statistics.

E. Learning Curve

To probe data sufficiency, Figure 3 plots train and CV RMSE versus training size. CV error improves rapidly up to $\sim$10k samples and then flattens; the gap between curves indicates a modest variance component.

IV. DISCUSSION

Thermal-conductivity dispersion measures and atomic-radius statistics appear consistently important. This aligns with physical intuition regarding electron–phonon interactions and structural effects. Linear models underfit, whereas GBR captures nonlinear interactions. Our compact configuration reproduces prior rankings [1] while staying within classroom runtime limits. Limitations include a single random split, impurity-importance bias, and limited hyperparameter tuning. Future work: stratified evaluation for the high-T_c tail, permutation/SHAP importances, and stacked or calibrated ensembles.

V. CONCLUSION

A compact ML pipeline with standard preprocessing and Gradient Boosting provides accurate T_c predictions on the UCI Superconductivity dataset and recovers physically plausible drivers. The workflow balances interpretability, runtime, and accuracy, serving as a sound baseline for further materials-informatics exploration.

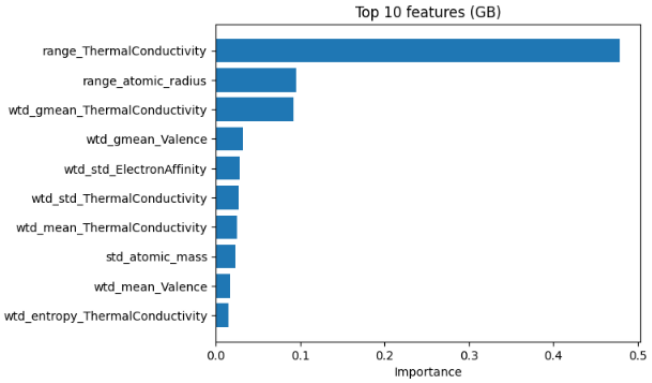

Fig. 2. Top 10 features by Gradient Boosting importance. Thermal-conductivity dispersion dominates, with atomic radius and valence following.

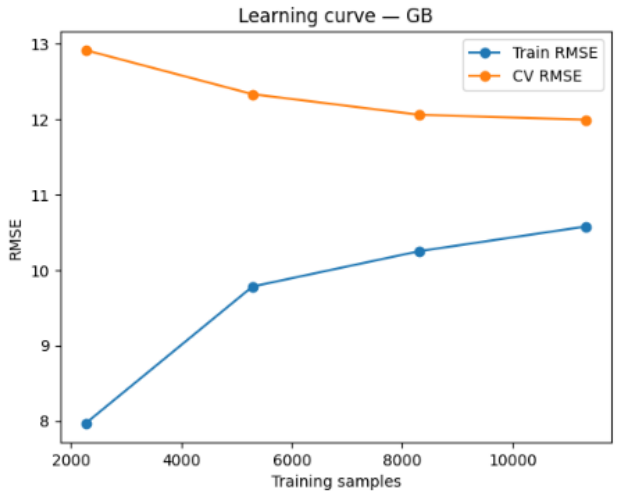

Fig. 3. Learning curve for the selected Gradient Boosting model (4 subsample sizes).

Reproducibility Statement

We fixed random seeds, standardized features after median imputation, separated model selection (3-fold CV on training) from final evaluation (held-out test), and generated all artifacts in a single Colab notebook using scikit-learn [3]. Package versions were printed in the notebook.

References

[1] K. Hamidieh, "A data-driven statistical model for predicting the critical temperature of a superconductor," *Computational Materials Science*, vol. 154, pp. 346–354, 2018.

[2] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[3] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer, 2009.

[5] L. Ward, A. Agrawal, A. Choudhary, and C. Wolverton, "A general-purpose machine learning framework for predicting properties of inorganic materials," *npj Computational Materials*, vol. 2, 2016.

APPENDIX A
CODE LISTING

```
import sys, math, zipfile, time
from pathlib import Path
import numpy as np, pandas as pd, matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, KFold, \
    cross_val_score, learning_curve
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, RidgeCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, \
    mean_absolute_error, r2_score


ZIP_PATH = "/content/superconductivty+data.zip"
TARGET   = "critical_temp"
RANDOM_STATE = 42
OUTDIR   = Path("/content/sta5703_submit")
(OUTDIR / "figures").mkdir(parents=True, exist_ok=True)
(OUTDIR / "results").mkdir(parents=True, exist_ok=True)
(OUTDIR / "tables").mkdir(parents=True, exist_ok=True)

t0 = time.time()

extract_dir = OUTDIR / "data"
with zipfile.ZipFile(ZIP_PATH, "r") as z:
    z.extractall(extract_dir)
train_path = next(extract_dir.rglob("train.csv"))
df = pd.read_csv(train_path)

y = df[TARGET].astype(float)
X = df.drop(columns=[TARGET]).select_dtypes(include=[np.number])
print(f"Data: {df.shape[0]} rows, {X.shape[1]} numeric features")

# Histogram (figure)
fig = plt.figure()
plt.hist(y, bins=50)
plt.xlabel(TARGET); plt.ylabel("count"); plt.title("Histogram of target")
fig.savefig(OUTDIR / "figures" / "hist_target.png",
            bbox_inches="tight", dpi=150)
plt.show()

# Top-15 correlations (table)
corr = X.copy(); corr["__t__"] = y
```

```
corr_tbl = (
    corr.corr(numeric_only=True)["__t__"]
    .dropna().drop(labels="__t__", errors="ignore")
    .sort_values(key=lambda s: s.abs(), ascending=False)
    .head(15).reset_index()
    .rename(columns={"index":"feature","__t__":"corr_with_target"})
)
corr_tbl.to_csv(OUTDIR / "tables" / "top15_corr.csv", index=False)
print("\nTop 15 correlated features with target:")
print(corr_tbl)

# Split & preprocessing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=RANDOM_STATE
)
pre = ColumnTransformer([
    ("num", Pipeline([("imp", SimpleImputer(strategy="median")),
                      ("scaler", StandardScaler())]), X.columns)
])

# Models (small + fast)
models = {
    "Linear": Pipeline([("pre", pre), ("model", LinearRegression())]),
    "Ridge": Pipeline([("pre", pre),
    ("model", RidgeCV(alphas=np.logspace(-3,3,7), cv=3))]),
    "GB":     Pipeline([("pre", pre),
    ("model", GradientBoostingRegressor(random_state=RANDOM_STATE,
        n_estimators=200, learning_rate=0.1, max_depth=3))])

}

# 3-fold CV
cv = KFold(n_splits=3, shuffle=True, random_state=RANDOM_STATE)
rows = []
for name, pipe in models.items():
    neg_mse = cross_val_score(pipe, X_train, y_train,
    scoring="neg_mean_squared_error", cv=cv, n_jobs=-1)
    rmse = np.sqrt(-neg_mse)
    rows.append({"model": name, "cv_mean_RMSE": rmse.mean(),
    "cv_std_RMSE": rmse.std()})
comp = pd.DataFrame(rows).sort_values("cv_mean_RMSE").reset_index(drop=True)
comp.to_csv(OUTDIR / "results" / "model_comparison_cv.csv", index=False)
print("\nCross-validation comparison (3-fold):")
print(comp)

# Final hold-out with best CV model
best_name = comp.iloc[0]["model"]
best_pipe = models[best_name].fit(X_train, y_train)
```

```
pred = best_pipe.predict(X_test)
final = pd.DataFrame([{
    "model": best_name,
    "test_RMSE": math.sqrt(mean_squared_error(y_test, pred)),
    "test_MAE":  mean_absolute_error(y_test, pred),
    "test_R2":   r2_score(y_test, pred)
}])
final.to_csv(OUTDIR / "results" / "final_holdout_metrics.csv", index=False)
print("\nFinal hold-out evaluation:")
print(final)

# Feature importance (if GB wins)
model = best_pipe.named_steps["model"]
if hasattr(model, "feature_importances_"):
    fi = pd.DataFrame({"feature": X.columns,
    "importance": model.feature_importances_"})
    fi = fi.sort_values("importance", ascending=False).head(10)
    fi.to_csv(OUTDIR / "tables" / f"feature_importance_{best_name}.csv",
    index=False)
    print(f"\nTop 10 features ({best_name}):")
    print(fi)
    fig = plt.figure()
    plt.barh(fi["feature"], fi["importance"])
    plt.gca().invert_yaxis()
    plt.xlabel("Importance"); plt.title(f"Top 10 features ({best_name})")
    fig.savefig(OUTDIR / "figures" / f"feature_importance_{best_name}.png",
    bbox_inches="tight", dpi=150)
    plt.show()

# learning curve for the best model
ts, train_scores, val_scores = learning_curve(
    best_pipe, X_train, y_train, cv=3, scoring="neg_mean_squared_error",
    train_sizes=np.linspace(0.2, 1.0, 4), n_jobs=-1,
    shuffle=True, random_state=RANDOM_STATE
)
train_rmse = np.sqrt(-train_scores).mean(axis=1)
val_rmse   = np.sqrt(-val_scores).mean(axis=1)

fig = plt.figure()
plt.plot(ts, train_rmse, marker="o", label="Train RMSE")
plt.plot(ts, val_rmse, marker="o", label="CV RMSE")
plt.xlabel("Training samples"); plt.ylabel("RMSE")
plt.title(f"Learning curve | {best_name}")
plt.legend()
fig.savefig(OUTDIR / "figures" / f"learning_curve_{best_name}.png",
    bbox_inches="tight", dpi=150)
plt.show()
```