“Energy efficiency in buildings”

A Case study Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

(Specialization in Artificial Intelligence & Machine Learning)

Mentor – Dr. P. Sankara Rao

by

VU22CSEN0500043

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE
GITAM SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)
VISAKHAPATNAM
2025

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

DECLARATION

I hereby declare that the capstone project report entitled **Energy Efficiency in buildings** is an original work done in the Department of Artificial Intelligence and Data Science, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering (AIML). The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 23-02-2025

VU22CSEN0500043
K.Hrishikesh

Contents

Case Study: Energy Efficiency in buildings

Introduction

Energy consumption prediction is crucial for optimizing building designs, reducing operational costs, and improving energy efficiency. In this study, we leverage ensemble machine learning techniques, including **Bagging (Random Forest) and Stacking**, to predict **Heating Load (Y1) and Cooling Load (Y2)** based on architectural and environmental factors.

Dataset Overview

The dataset used for this study contains 768 samples with 8 predictive features and 2 target variables. The features include:

X1: Relative Compactness

X2: Surface Area

X3: Wall Area

X4: Roof Area

X5: Overall Height

X6: Orientation

X7: Glazing Area

X8: Glazing Area Distribution

Dataset Details

- **Total Records:** 768
- **Features:** 8
- **Target Variables:**
  - **Y1:** Heating Load (kWh/m²)
  - **Y2:** Cooling Load (kWh/m²)

Data source: https://archive.ics.uci.edu/dataset/242/energy

Data Preprocessing

1. Handling Missing Values

```
data1.fillna(data1.median(), inplace=True)
```

- **What it does?**
  - Fills missing values in `data1` using the **median** of each column.
- **Why?**
  - Median imputation is robust against outliers compared to mean imputation.
  - Ensures no missing values that could disrupt model training.

---

2. Removing Duplicates

```
data1.drop_duplicates(inplace=True)
```

- **What it does?**
  - Removes any duplicate rows from the dataset.
- **Why?**
  - Duplicates can introduce bias and redundancy, leading to overfitting.

---

3. Dropping Unnecessary Columns

```
data1.drop(columns=['Unnamed: 0'], inplace=True,
errors='ignore')
```

- **What it does?**
  - Removes a column named `'Unnamed: 0'` (often an index column from CSV/Excel files).
- **Why?**
  - It doesn't contribute to predictive modeling and should be removed.

---

4. Removing Highly Correlated Features

```
corr_matrix = data1.corr().abs()
```

```
upper =
corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
k=1).astype(bool))
to_drop = [column for column in upper.columns if
any(upper[column] > 0.9)]
data1.drop(columns=to_drop, inplace=True)
```

- **What it does?**
  - Computes the **correlation matrix** (`corr_matrix`) for all numerical features.
  - Identifies pairs of features that have a high correlation (**above 0.9**).
  - Drops one feature from each highly correlated pair.
- **Why?**
  - Highly correlated features cause **multicollinearity**, making models less interpretable.
  - Keeping redundant features does not add value and may lead to overfitting.

---

5. Splitting Features (X) and Target (y)

```
X = data1.iloc[:, :-1]  # All columns except the last
one as features
y = data1.iloc[:, -1]   # Last column as target
```

- **What it does?**
  - Assigns **all columns except the last one** to `X` (independent variables).
  - Assigns the **last column** to `y` (dependent variable / target).
- **Why?**
  - Ensures proper separation of input variables and the target variable.

---

6. Feature Scaling

```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X),
columns=X.columns)
```

- **What it does?**
  - Standardizes features by transforming them to have **zero mean** and **unit variance**.
  - Converts transformed data back into a DataFrame.
- **Why?**
  - Standardization improves model convergence for algorithms like **SVM, Perceptron, and Gradient Boosting**.
  - It ensures that all features are on a similar scale, preventing features with larger values from dominating the learning process.

Machine Learning Models

1.1 Ensemble Methods:

We implemented and compared the performance of different ensemble methods:

1.1.1 Bagging - Random Forest:

- Utilizes multiple Decision Trees to reduce overfitting.
- Aggregates predictions from various trees to improve stability.

[13]:
```
# Bagging with Random Forest
bagging_model = RandomForestRegressor(n_estimators=100, random_state=42)
bagging_model.fit(X_train, y_train)
# Predictions
y_pred_bagging = bagging_model.predict(X_test)
# Evaluation
rmse_bagging = np.sqrt(mean_squared_error(y_test, y_pred_bagging))
r2_bagging = r2_score(y_test, y_pred_bagging)
#accuracy_bagging = accuracy_score(y_test, y_pred_bagging)
print(f"Bagging (Random Forest) -> RMSE: {rmse_bagging:.4f}, R2 Score: {r2_bagging:.4f}")
```

```
Bagging (Random Forest) -> RMSE: 0.5263, R2 Score: 0.9973
```

1.1.2 Stacking Ensemble

Combines multiple models (Random Forest, XGBoost) with a meta-learner (Gradient Boosting Regressor) for improved accuracy.

[16]:
```
# Define base learners
base_learners = [
    ('rf', RandomForestRegressor(n_estimators=100, random_state=42)),
    ('xgb', XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42))
]
# Define Meta-learner
stacking_model = StackingRegressor(estimators=base_learners, final_estimator=XGBRegressor())
# Train Stacking Model
stacking_model.fit(X_train, y_train)
# Predictions
y_pred_stacking = stacking_model.predict(X_test)
# Evaluation
rmse_stacking = np.sqrt(mean_squared_error(y_test, y_pred_stacking))
r2_stacking = r2_score(y_test, y_pred_stacking)
print(f"Stacking -> RMSE: {rmse_stacking:.4f}, R2 Score: {r2_stacking:.4f}")
```

```
Stacking -> RMSE: 0.6046, R2 Score: 0.9965
```

Model Performance Comparison

[18]:
```
import numpy as np
def regression_accuracy(y_true, y_pred, tolerance=0.05):
    correct = np.abs(y_true - y_pred) <= (tolerance * np.abs(y_true))
    return np.mean(correct)
# Compute accuracy for Bagging
accuracy_bagging = regression_accuracy(y_test, y_pred_bagging)
print(f"Bagging Accuracy: {accuracy_bagging*100:.4f}")
# Compute accuracy for Boosting
#accuracy_boosting = regression_accuracy(y_test, y_pred_boosting)
#print(f"Boosting Accuracy: {accuracy_boosting*100:.4f}")
# Compute accuracy for Stacking
accuracy_stacking = regression_accuracy(y_test, y_pred_stacking)
print(f"Stacking Accuracy: {accuracy_stacking*100:.4f}")
```

```
Bagging Accuracy: 99.3506
Stacking Accuracy: 91.5584
```

Key Observations:

- The baseline Decision Tree had lower recall and precision compared to ensemble methods.
- Random Forest performed well due to the averaging effect of multiple decision trees.
- Stacking provided the best balance, leveraging multiple models to improve performance.

Conclusion

This study demonstrates the effectiveness of ensemble learning in predicting **energy consumption** in buildings. **Stacking Regressor** provided the best accuracy among the models tested. Future improvements can include:

- Exploring **Deep Learning** models for enhanced prediction.
- Incorporating additional environmental factors like **humidity, temperature, and insulation properties**.
- Optimizing hyperparameters using **Bayesian Optimization** for even better results.

This predictive model can assist architects, engineers, and energy analysts in designing more energy-efficient buildings, reducing costs, and contributing to sustainable development.