Assignment: Decision Trees, Random Forests, and Boosting in Practice

Name: Xiaochen Cui
ID: 2101390

Background

This homework is about applying different regression models to predict income based on a given dataset. I use Python and scikit-learn to train and evaluate models like Linear Regression, Decision Tree, Random Forest, and Gradient Boosting. The goal is to understand how different models perform and what features are important. Through this project, I learned how to preprocess data, train models, and compare results in a practical way.

Data

The dataset comes from the Labour Force Survey and includes personal and job-related information such as age, education level, employment status, hours worked, and wage. It contains both numerical and categorical variables, which are described in the official codebook. Before modeling, I cleaned the data and selected relevant features for predicting income.

Result

I trained several regression models including Linear Regression, Decision Tree, Random Forest, and Gradient Boosting. The performance of each model was evaluated using metrics such as mean absolute error and R-squared. Gradient Boosting gave the best result, showing lower prediction error and better fit compared to other models.

Key Findings

- I found that categorical features like education level and employment status have strong impact on income prediction.
- Tree-based models performed better than linear models, especially for capturing non-linear patterns.
- Gradient Boosting was the most accurate among all models I tried, but it also took the longest time to train.

Appendix

Implementation

This is the core implementation:

```
def train_and_evaluate_model(df, features, target, model, model_name, method="Basic",
                             parameters=None, cv_score=None, show_plot=True):
    """
    Unified function to train and evaluate any regression model.

    Parameters:
    - df: DataFrame containing the data
    - features: List of feature column names
    - target: Target column name
    - model: Instantiated sklearn model
    - model_name: Name of the model for reporting
    - method: Method used ("Basic", "Manual Tuning", "GridSearchCV")
    - parameters: String representation of parameters (for tuning methods)
    - cv_score: Cross-validation score (for GridSearch)
    - show_plot: Whether to show visualization plots
    """
    start_time = time.time()

    # Preprocess data
    df_clean = df.copy()
    df_clean["HRLYEARN"] = df_clean["HRLYEARN"] / 100

    if "AGE_12" in features:
        df_clean["AGE_12"] = 10 + df_clean["AGE_12"] * 5

    # Select required columns and drop nulls
    df_model = df_clean[features + [target]].dropna()
    print(f"Data after preprocessing: {df_model.shape[0]} rows")

    if len(df_model) == 0:
        print("ERROR: No data remaining after preprocessing!")
        return None

    X, y = df_model[features], df_model[target]

    # Split and train
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Handle GridSearchCV differently

# Evaluate
if hasattr(fitted_model, 'predict'):
    y_pred = fitted_model.predict(X_test)
elif hasattr(fitted_model, 'best_estimator_'):
    y_pred = fitted_model.best_estimator_.predict(X_test)
else:
    print("ERROR: Cannot make predictions!")
    return None

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
mean_error_percent = (rmse / np.mean(y_pred)) * 100

# Print results

# Visualization

# Save results using unified function
```