Exercise 2

Nashkerskyi Volodymyr 12407818
Avakian Valeriia 12432869
Dobrynina Ekaterina 12332276

VU Machine Learning

TU Wien, Austria

Overview

- Implementation of Custom Random Forest algorithm based on regression trees for predicting numeric values
- Implementation of Random Forest algorithm with LLM tool
- Comparison of metrics and execution time of Custom Random Forest with existing regression tools from Sklearn
- Comparison of results of different hyperparameters for Random Forest

Chosen Datasets

The datasets are:

- Housing Prices
- Life Expectancy

Datasets overview:

- Housing Prices is a wider dataset with 1460 instances and 80 features (43 categorical and 37 numeric), missing and rare values
- Life Expectancy is a bit longer dataset 2938 instances and 21 features (19 numeric and only 2 categorical), missing and rare values

Preprocessing

Housing prices:

- Rare values were combined in one group “Other”
- Missing values in numeric columns were replaced with medians
- LabelEncoder was applied to ordered columns
- OneHotEncoder was applied to other categorical columns
- StandardScaler was applied to target variable

Life Expectancy:

- High correlated features were dropped
- LabelEncoder was applied to categorical features
- Missing values in numeric columns were replaced with medians, grouped by year
- StandardScaler was applied to target variable

Decision Tree from scratch (custom)

class Node

- each node stores information: feature_index, threshold, left and right references to subtrees, var_red

class DecisionTreeRegressor

def __init__

- with max_depth, min_samples_split, min_samples_leaf, num_features

def fit

- input data X (feature matrix) and y (array of values to predict), builds a decision tree using the build_tree

def build_tree

- recursively builds a decision tree, creates a leaf if the stop criteria is reached

def best_split

def variance_reduction

def split

- splits the data into two parts depending on the threshold

def predict

def traverse_tree

- recursively goes from the root to the leaf and returns a prediction (the value from the leaf of the tree)

Random Forest from scratch (custom)

- using decision trees (DecisionTreeRegressor) as base models

Class RandomForestRegressor

def __init__

- with num_trees, max_depth, min_samples_split, min_samples_leaf, num_features

def fit

- with transformation of input X and y to numpy arrays if they are in pandas
- for each tree in the for loop a DecisionTreeRegressor object is created with the specified parameters and trained on the bootstrap sample

def bootstrap

- which is randomly created from the training data

def predict

- average predictions of all trees in the ensemble for each sample to get the final result

Comparison with LLMs implementation

Splitting Criteria:

- Custom implementation uses variance reduction for splitting
- LLM uses mean squared error (MSE)

Node representation:

- In the custom implementation, the Node class stores the variance reduction value (var_red)
- In the LLM, it stores the predicted value (value) at the leaves of the tree

Input data types:

- Custom implementation is more flexible and supports Pandas data, automatically converting it to NumPy
- LLM implementation requires that the input data be in the NumPy arrays format beforehand

Running the LLM code, the KeyError occurs because it tries to use indices to select rows from a pandas.DataFrame (X) using the X[indices], what is problematic because X expects column indices, not row indices

```
KeyError: "None of [Index([ 16, 464, 388, 315, 301,  74, 326, 534, 421, 863,\n       ...\n       870, 653, 831, 143, 837, 523, 858, 723, 366,
381],\n      dtype='int64', length=875)] are in the [columns]"
```

Comparison with other algorithms: Housing Prices

Before hyperparameters tuning:

Model	RMSE	MSE	R^2	Train MSE	Test MSE	Test/Train MSE Ratio
Random Forest (sklearn)	0.3934	0.1548	0.8380	0.0276	0.1548	5.6004
Custom Random Forest	0.5243	0.2749	0.7123	0.0489	0.2749	5.6197
Linear Regression	0.3937	0.1550	0.8378	0.2221	0.1550	0.6979
XGBoost	0.4109	0.1688	0.8233	0.0676	0.1688	2.4973
SVR	0.3628	0.1316	0.8623	0.1620	0.1316	0.8123

- SVR shows the best accuracy with the lowest RMSE (0.3628) and MSE (0.1316) and the high R^2 (0.8623) indicates that the model explains the data variance well
- Custom Random Forest has the highest RMSE (0.5243) and the lowest R^2 (0.7123), indicating its limited ability to explain the data

Experiments with hyperparameters: Housing Prices

Before hyperparameters tuning:

Model	RMSE	MSE	R^2	Train MSE	Test MSE	Test/Train MSE Ratio
Random Forest (sklearn)	0.3934	0.1548	0.8380	0.0276	0.1548	5.6004
Custom Random Forest	0.5243	0.2749	0.7123	0.0489	0.2749	5.6197

After hyperparameters tuning:

Model	RMSE	MSE	R^2	Train MSE	Test MSE	Test/Train MSE Ratio
Random Forest (sklearn)	0.3595	0.1293	0.8645	0.0632	0.1294	1.5161
Custom Random Forest	0.4041	0.1633	0.8291	0.0541	0.1633	3.0139

RandomForest sklearn:

'n_estimators': [50, 100, 200],
'max_depth': [None, 10, 20],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'max_features': ['auto', 'sqrt']

RandomForest Custom:

'num_trees': [5, 10, 50],
'max_depth': [None, 10, 20],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'num_features': [None, 'auto', 'sqrt']

Comparison runtime: Housing Prices

Model	Time for Training (seconds)	Time for Test Prediction (seconds)
RandomForestRegressor	0.3447	0.0114
RandomForestRegressor Custom	5.8951	0.0050
Linear Regression	0.0106	0.0013
XGBoost	0.0504	0.0030

Comparison with other algorithms: Life Expectancy

Metrics before hyperparameters tuning:

Model	RMSE	MSE	R^2	Train MSE	Test MSE	Test/Train MSE Ratio
Random Forest (sklearn)	0.2321	0.0539	0.9488	0.0083	0.0539	6.5185
Custom Random Forest	0.2571	0.0661	0.9372	0.0186	0.0661	3.5503
Linear Regression	0.4545	0.2065	0.8037	0.1878	0.2065	1.1001
XGBoost	0.2602	0.0677	0.9357	0.0372	0.0677	1.8197

- Random Forest (sklearn) performed best in accuracy, with a low RMSE (0.2321) and high R^2 (0.9488)
- Custom Random Forest: RMSE (0.2571) and R^2 (0.9372) are similar to XGBoost, but slightly worse
- Test/Train MSE Ratio for both models are higher, indicating more overfitting compared to other models

Experiments with hyperparameters: Life Expectancy

Before hyperparameters tuning:

Model	RMSE	MSE	R^2	Train MSE	Test MSE	Test/Train MSE Ratio
Random Forest (sklearn)	0.2321	0.0539	0.9488	0.0083	0.0539	6.5185
Custom Random Forest	0.2436	0.0594	0.9435	0.0186	0.0593	3.1712

Metrics after hyperparameters tuning:

Model	RMSE	MSE	R^2	Train MSE	Test MSE	Test/Train MSE Ratio
Random Forest (sklearn)	0.2111	0.0446	0.9576	0.0152	0.0446	2.9198
Custom Random Forest	0.2182	0.0476	0.9547	0.0128	0.0476	3.7035

RandomForest sklearn:

'n_estimators': [5, 10, 20],
'max_depth': [5, 10],
'min_samples_split': [2, 5, 10, 20, 50],
'min_samples_leaf': [1, 2, 4],
'max_features': [None, 'auto', 'sqrt']

RandomForest Custom:

'num_trees': [5, 10, 20],
'max_depth': [5, 10],
'min_samples_split': [2, 5, 10, 20, 50],
'min_samples_leaf': [1, 2, 4],
'num_features': [None, 'auto', 'sqrt']

Comparison runtime: Life Expectancy

Model	Time for Training (seconds)	Time for Test Prediction (seconds)
Random Forest (Sklearn)	0.136	0.0025
Custom Random Forest	16.4025	0.0025
XGBoost	0.0992	0.0022
Linear Regression	0.0027	0.0009

Conclusion

Prediction accuracy:

- On both datasets, sklearn Random Forest demonstrated the best results in RMSE, MSE and R^2 metrics, indicating high prediction accuracy

Hyperparameters:

- After tuning the hyperparameters for both sklearn Random Forest and Custom Random Forest, the results of both models were improved. However, sklearn Random Forest was still more accurate and efficient

Runtime:

- The training time of the models showed a significant difference, with Custom Random Forest taking much longer to train than sklearn Random Forest or XGBoost
- Despite this, the prediction times for both models were close, showing that Custom Random Forest is mostly lagging in time during the training phase

Conclusion

During the current task, we

- learned how to implement random forest and decision trees algorithms from scratch, which helped us understand their inner workings
- compared custom and existing models and learned that there are many details in the training settings
- learned data preprocessing and hyperparameter optimization to improve model accuracy
- understood that **Random forest from scratch** helps for better understanding the algorithm, but requires significant optimization efforts and may not always be competitive where speed and efficiency are important