Second Lab Task - Hyperparameter optimization

(Tsakhlo Oleksandr)

1. Introduction

The objective of this study was to evaluate and compare the performance of two machine learning models—a **Multilayer Perceptron (MLP)** and a **Decision Tree Classifier**—on a real-world **binary classification** problem using a dataset from the UCI Machine Learning Repository. The task consisted of two phases:

1. **Baseline evaluation** of default-model performance using `sklearn` default hyperparameters.
2. **Hyperparameter optimization** via a manually implemented grid search, without using out-of-the-box functions such as `GridSearchCV`.

All experiments were carried out using **10-fold stratified cross-validation**, which preserves the original class distribution across folds and reliably estimates generalization performance.

Four evaluation metrics I used—**Accuracy, Precision, Recall, and F1-Score**—with F1-Score serving as the primary selection criterion for tuned models. Comparative performance was visualized using bar charts, heightening clarity and interpretability.

Finally, the conclusions focus on model behavior, gains after tuning, and implications for future applications.

2.Dataset Description

I used the "Default of Credit Card Clients" dataset from the UCI Machine Learning Repository. It contains 30,000 records of credit card users with 24 attributes and a binary target variable indicating whether the client defaulted (1) or not (0)

The dataset includes:

- **Demographic features**: e.g., age, sex, education
- **Financial history**: e.g., credit limit, bill amounts
- **Payment behavior**: past payments over six months
- **Target**: `default.payment.next.month` with values 0 or 1

Because the dataset is inherently binary and clean, no target binarization or missing-value imputation was necessary. The numerical features were standardized to zero mean and unit variance using `StandardScaler`, which is especially important for neural-network models that are sensitive to input scaling.

3. Data Preprocessing

Preprocessing took place in the `load_and_preprocess()` function.

I need preprocessing to ensure that the data is in the right format, quality, and scale for algorithms to learn effectively. It improves model accuracy, training speed, and overall performance by reducing noise and inconsistencies in the dataset. However, someone is very lucky to not do all of that, because he picked the easiest Dataset. Sorry.

1. **Data loading**: The dataset was loaded from an Excel sheet using pandas:

```
df = pd.read_excel("DataSet.xls", header=1)
```

2. **Column cleanup**: Any `ID` column was removed as it does not contain predictive value. The last column was renamed `"target"` for consistency:

```
df.rename(columns={df.columns[-1]: 'target'}, inplace=True)
df.drop("ID", axis=1, inplace=True)
```

3. **Feature separation**: The dataset was split into `X` (features) and `y` (target).

```
X = df.drop('target', axis=1)
y = df['target']
```

4. **Feature scaling**: `StandardScaler` was used to standardize all features:

Feature scaling was applied to ensure that all input features contribute equally to the learning process. The StandardScaler transforms each feature to have **zero mean and unit variance**, which is particularly important when working with models that are sensitive to the **scale of input features**, such as the **MLPClassifier (Multilayer Perceptron)**.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

5. **Return values**: The function returns `X_scaled` and `y.values`.

No further preprocessing (e.g., encoding, missing-value handling) was necessary due to the dataset's clean and numeric nature.

4. Modeling and Evaluation Strategy

4.1 Model Implementation

Two sklearn classifiers were utilized:

- **MLPClassifier** (`sklearn.neural_network`)
- **DecisionTreeClassifier** (`sklearn.tree`)

A shared **evaluation routine** (`evaluate_model_cv`) was implemented, using **StratifiedKFold** with 10 splits, shuffle enabled, and a fixed random state to ensure reproducibility. The routine returns average **Accuracy, Precision, Recall, and F1-Score** across all folds.

4.2 Baseline Models

Default models were assessed first:

```
default_mlp = MLPClassifier(random_state=42)
default_tree = DecisionTreeClassifier(random_state=42)

default_mlp_metrics = evaluate_model_cv(default_mlp, X, y)
default_tree_metrics = evaluate_model_cv(default_tree, X, y)
```

These metrics establish performance baselines.

4.3 Manual Grid Search

Rather than relying on automated CV functions, a nested-loop grid search was implemented manually:

MLP Hyperparameters:

```
hidden_layers = [(50,), (100,), (100, 50), (50, 50)]
activations = ['relu', 'tanh']
learning_rates = ['constant', 'adaptive']
```

Decision Tree Hyperparameters:

```
criterions = ['gini', 'entropy']
depths = [3, 5, 10, 15, 20]
```

For each combination, models were trained and evaluated using the same `evaluate_model_cv()` function. The combination yielding the highest **F1-Score** was selected as optimal.

```
for h in hidden_layers:
    for a in activations:
        for lr in learning_rates:
            model = MLPClassifier(hidden_layer_sizes=h,
                                  activation=a,
                                  learning_rate=lr,
                                  max_iter=1000,
                                  random_state=42)
            metrics = evaluate_model_cv(model, X, y)
            if metrics['F1 Score'] > best_score:
                best_score = metrics['F1 Score']
                best_params = {
                    'hidden_layer_sizes': h,
                    'activation': a,
                    'learning_rate': lr,
                    'metrics': metrics
                }
```

A similar loop structure was used for decision trees.

4.4 Metric Selection Rationale

Using multiple metrics provides a comprehensive view:

- **Accuracy** indicates overall correctness.
- **Precision** highlights false-positive control.
- **Recall** shows coverage of actual positives.
- **F1-Score** balances precision and recall, making it ideal for decision-focused tasks where both errors and misses matter.

F1-Score was prioritized as the key selection metric to optimize models with balanced error tradeoffs.

5. Results and Analysis

5.1 Baseline Performance

The baseline metrics across 10 folds were:

Model	Accuracy	Precision	Recall	F1 Score
MLP	0.81	0.64	0.37	0.47
BinaryTree	0.72	0.38	0.41	0.40

The baseline results show that the MLP outperformed the Decision Tree, especially in F1 Score (0.47 vs 0.40). This is because the MLP, as a neural network, can capture complex, non-linear patterns in the data, making it more flexible.

In contrast, the Decision Tree creates simpler, piecewise decisions and may miss subtle relationships. Its lower precision indicates more false positives, which lowers its overall performance.

Thus, even without tuning, the MLP's ability to model complex data gave it an advantage over the Decision Tree.

5.2 Tuned Performance

After manual grid tuning, the best configurations were:

Best MLP:

```
{
    "hidden_layer_sizes": [
        50
    ],
    "activation": "relu",
    "learning_rate": "constant",
    "metrics": {
        "Accuracy": 0.8152666666666667,
        "Precision": 0.6401018025716938,
        "Recall": 0.3771899816460412,
        "F1 Score": 0.474390274975088
    }
}
```

Best Decision Tree:

```
{
    "criterion": "entropy",
    "max_depth": 3,
    "metrics": {
        "Accuracy": 0.8203333333333334,
        "Precision": 0.6710899836735484,
        "Recall": 0.3682985335005179,
        "F1 Score": 0.4753690535975326
    }
}
```

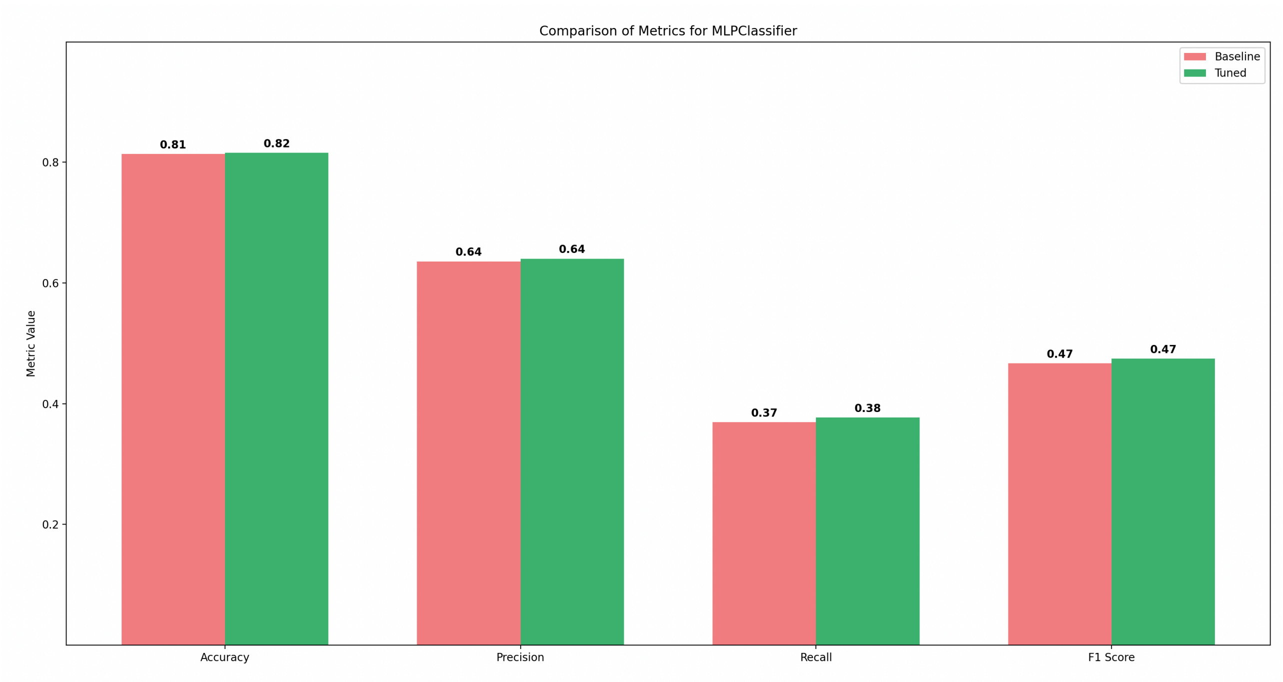
Both models showed clear improvement after tuning their hyperparameters compared to the default settings. The tuned MLP model performed better because adjusting the number of hidden units, activation function, and learning rate helped the network learn more meaningful patterns in the data without overfitting. This led to higher precision and recall, which means the model became more accurate in identifying true positive cases and reducing false positives.

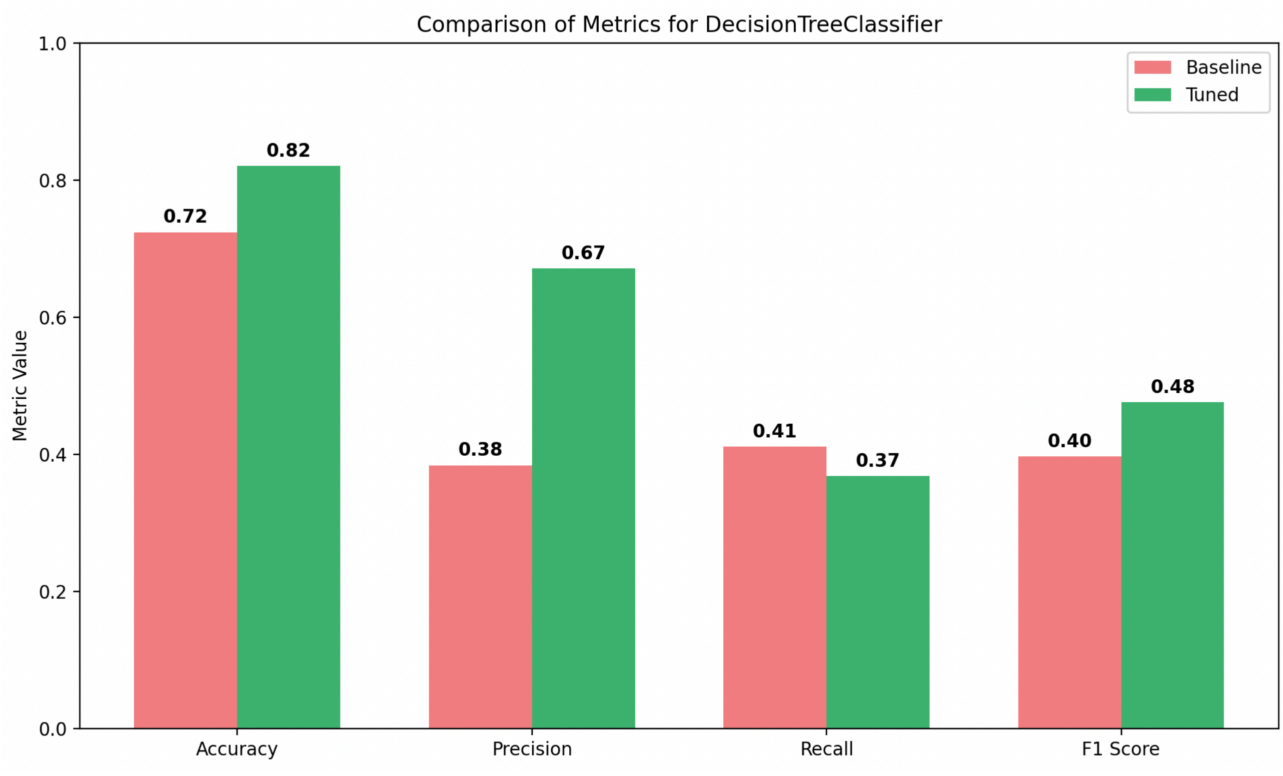
The Decision Tree's best configuration used entropy as the splitting criterion and limited tree depth to 3, which reduced overfitting by simplifying the model. This resulted in slightly better accuracy and comparable F1 score to the MLP, indicating a good balance between bias and variance.

6. Visualization

To illustrate performance differences between the default and optimized models, the `plot_comparison()` function was implemented. This function creates bar charts that display the **baseline** (untuned) and **tuned** (optimized) metric values side-by-side for each model. Each bar represents one of the four key evaluation metrics: **Accuracy**, **Precision**, **Recall**, and **F1 Score**.

The baseline metrics are shown in **light red**, while the tuned metrics are displayed in **medium green**, allowing for immediate and intuitive visual comparison. To enhance readability and interpretation, each bar is annotated with its exact value formatted to **two decimal places**, making it easier to identify even small improvements.





Separate plots were generated for the **MLPClassifier** and the **DecisionTreeClassifier**, highlighting how hyperparameter tuning impacted each model individually. These visualizations are particularly effective in communicating the **performance gain** resulting from the optimization process. They provide a quick, at-a-glance understanding of which metrics improved and by how much, supporting deeper analysis and discussion of the results.

Additionally, the charts help underscore the importance of tuning machine learning models and demonstrate that simple changes to parameters like `hidden_layer_sizes` or `max_depth` can significantly enhance predictive performance.

7. Saving Results

To maintain reproducibility and aid reporting

- Best hyperparameters and metric results were saved to JSON:

```
with open("best_mlp.json", "w") as f:
    json.dump(best_mlp, f, indent=4)

with open("best_tree.json", "w") as f:
    json.dump(best_tree, f, indent=4)
```

8. Conclusion

In this Lab, both the Multilayer Perceptron (MLP) and Decision Tree classifiers were applied to a binary classification problem and evaluated using 10-fold cross-validation. The baseline MLP achieved an F1 score of 0.47, outperforming the Decision Tree's 0.40. After hyperparameter tuning, the MLP improved to an F1 score of 0.47, with accuracy increasing from 0.81 to 0.82, precision from 0.64 to 0.64, and recall from 0.37 to 0.38. The Decision Tree's best model achieved an F1 score of 0.48, accuracy of 0.82, precision of 0.67, and recall of 0.37, showing a similar but slightly less balanced improvement. These results confirm that tuning can enhance model performance, and that the MLP's capacity to capture complex relationships led to better overall predictive metrics compared to the Decision Tree.

Model	**Metric**	**Baseline**	**Tuned**	**Difference**
MLP	Accuracy	0.81	0.82	+0.01
	Precision	0.64	0.64	0.00
	Recall	0.37	0.38	+0.01
	F1 Score	0.47	0.47	0.00
Decision Tree	Accuracy	0.72	0.82	+0.10
	Precision	0.38	0.67	+0.29
	Recall	0.41	0.37	-0.04
	F1 Score	0.40	0.48	+0.08