Predictive Analysis with Decision Trees

1. Introduction

Decision Trees are supervised machine learning models widely used for predictive analysis because of their simplicity, interpretability, and ability to handle both classification and regression problems. A decision tree works by recursively splitting the dataset into smaller subsets based on feature values, forming a tree-like structure of decisions. Each internal node represents a decision rule, each branch represents an outcome of the rule, and each leaf node represents a final prediction.

Despite their advantages, decision trees are prone to **overfitting**, especially when the tree grows too deep and captures noise from the training data. To address this issue, **pruning techniques** are applied to improve model generalization.

This project focuses on implementing decision trees for classification and regression tasks, understanding impurity measures such as **Entropy** and **Gini Index**, and applying **pre-pruning** and **post-pruning** techniques to optimize model performance.

2. Objectives of the Project

The main objectives of this project are:

- To implement decision tree models for both classification and regression tasks.
- To understand and apply **Entropy** and **Gini Index** as criteria for splitting nodes.
- To reduce overfitting using **pre-pruning** (early stopping) and **post-pruning** (cost complexity pruning).
- To visualize the decision tree structure and analyze **feature importance**.

3. Decision Tree Concepts

3.1 Entropy

Entropy is a measure of randomness or impurity in a dataset. It is commonly used in decision trees to decide the best split at each node.

- Entropy is **0** when all samples belong to the same class (pure node).
- Entropy is **maximum** when classes are equally mixed.

The decision tree selects splits that **maximize information gain**, which is the reduction in entropy after a split.

3.2 Gini Index

The Gini Index measures how often a randomly chosen element would be incorrectly classified. It is computationally faster than entropy and commonly used in CART (Classification and Regression Trees).

- Gini Index = 0 indicates a pure node.
- Lower Gini values indicate better splits.

Both entropy and Gini index aim to create child nodes that are more homogeneous than the parent node.

4. Classification Using Decision Tree

4.1 Dataset Description

For the classification task, the **Iris dataset** is used. It contains:

- 150 samples
- 4 input features:
    - Sepal length
    - Sepal width
    - Petal length
    - Petal width
- 3 target classes representing different iris species

The dataset is split into training (80%) and testing (20%) sets.

4.2 Model Training

A Decision Tree Classifier is trained using the **Gini Index** as the splitting criterion. The model learns decision rules based on feature values to classify iris species.

Model performance is evaluated using **accuracy**, which measures the percentage of correct predictions on the test dataset.

4.3 Pre-Pruning (Early Stopping)

Pre-pruning is applied by limiting tree growth using parameters such as:

- max_depth: Maximum depth of the tree
- min_samples_leaf: Minimum number of samples required at a leaf node

By controlling these parameters, the tree avoids learning overly complex patterns. In this project, the pre-pruned classifier achieved **100% accuracy**, indicating excellent generalization on the test set.

4.4 Post-Pruning (Cost Complexity Pruning)

Post-pruning is performed after training the full tree using **Cost Complexity Pruning (CCP)**.

- The algorithm computes different values of ccp_alpha, which control how much pruning is applied.
- Multiple models are trained using different alpha values.
- The model with the best test accuracy is selected.

In this case, the best value of ccp_alpha was **0.0**, indicating that additional pruning did not improve performance.

4.5 Visualization and Feature Importance

The decision tree is visualized using Matplotlib, showing:

- Decision rules at each node
- Gini/Entropy values
- Class distribution

Feature importance analysis revealed that:

- **Petal length** is the most important feature
- **Petal width** has moderate importance
- **Sepal features** contribute very little to classification

This confirms that petal measurements are the strongest predictors for iris species.

5. Regression Using Decision Tree

5.1 Dataset Description

For regression, the **Diabetes dataset** is used. It consists of:

- 442 samples
- 10 numerical input features
- A continuous target variable representing disease progression

The data is split into training (80%) and testing (20%) sets.

5.2 Model Training

A Decision Tree Regressor is trained to predict continuous target values. Model performance is evaluated using **Mean Squared Error (MSE)**.

- Lower MSE indicates better prediction accuracy.
- The initial unpruned model produced a relatively high MSE, indicating overfitting.

5.3 Pre-Pruning in Regression

Pre-pruning is applied by setting:

- max_depth = 4
- min_samples_leaf = 10

This reduces model complexity and improves generalization. The pre-pruned regression model achieved a lower MSE compared to the unpruned model, demonstrating improved performance.

5.4 Post-Pruning in Regression

Cost complexity pruning is also applied to the regression model:

- Different ccp_alpha values are evaluated
- Models are compared using MSE
- The alpha value producing the minimum MSE is selected as the optimal pruning level

This approach helps balance bias and variance in the regression model.

6. Results and Observations

- Decision trees perform well for both classification and regression tasks.
- Entropy and Gini Index effectively measure node impurity.
- Pre-pruning significantly reduces overfitting while maintaining strong performance.
- Post-pruning helps fine-tune the model, though its impact depends on the dataset.
- Feature importance analysis improves interpretability and model understanding.

7. Conclusion

This project demonstrates the effectiveness of decision trees for predictive analysis. By applying entropy, Gini index, and pruning techniques, robust and interpretable models were developed for both classification and regression tasks. Pruning proved essential in controlling overfitting and improving generalization, making decision trees a powerful yet simple tool in machine learning.

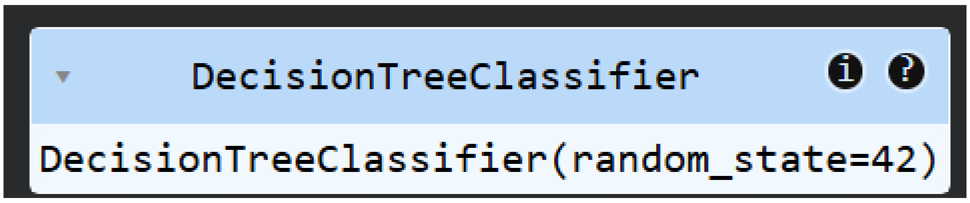
8. Tools and Libraries Used

- Python
- Scikit-learn
- Matplotlib

9. Future Scope

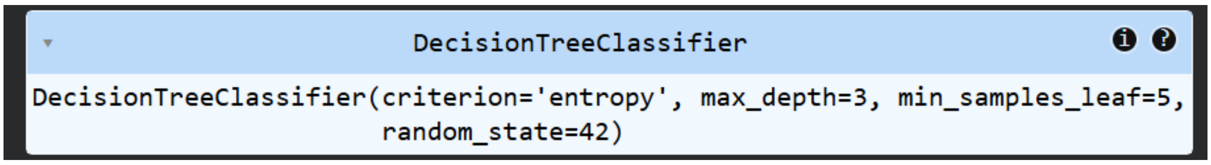
- Compare decision trees with ensemble methods like Random Forests and Gradient Boosting
- Perform hyperparameter tuning using cross-validation
- Apply the model to real-world datasets

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt


data = load_iris()

X = data.data

y = data.target


X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)


clf = DecisionTreeClassifier(

    criterion="gini",

    random_state=42

)

clf.fit(X_train, y_train)
```



```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
y_pred = clf.predict(X_test)

pre_pruned_clf = DecisionTreeClassifier(
    criterion="entropy",
    max_depth=3,
    min_samples_leaf=5,
    random_state=42
)
pre_pruned_clf.fit(X_train, y_train)
```



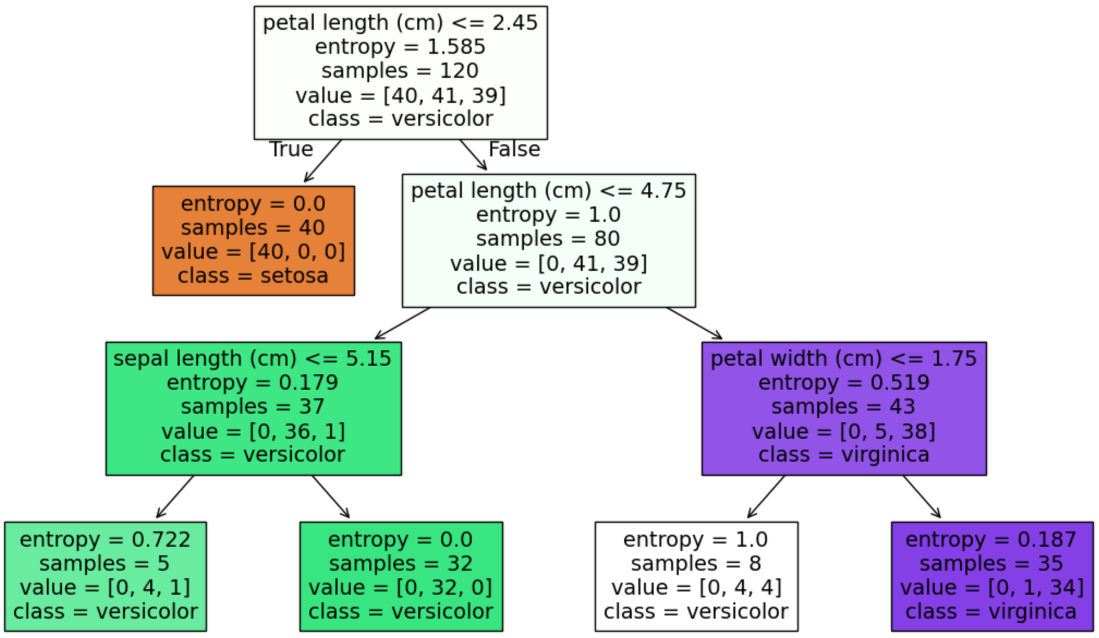
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=5,
                       random_state=42)
```

```
print(
    "Pre-Pruned Accuracy:",
    accuracy_score(y_test, pre_pruned_clf.predict(X_test))
)
Pre-Pruned Accuracy: 1.0
```

```
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas


pruned_models = []
accuracies = []


for alpha in ccp_alphas:
    model = DecisionTreeClassifier(
        ccp_alpha=alpha,
        random_state=42
    )
    model.fit(X_train, y_train)
    pruned_models.append(model)
    accuracies.append(
        accuracy_score(y_test, model.predict(X_test))
    )

best_alpha = ccp_alphas[accuracies.index(max(accuracies))]
print("Best ccp_alpha:", best_alpha)
```

```
Best ccp_alpha: 0.0
```

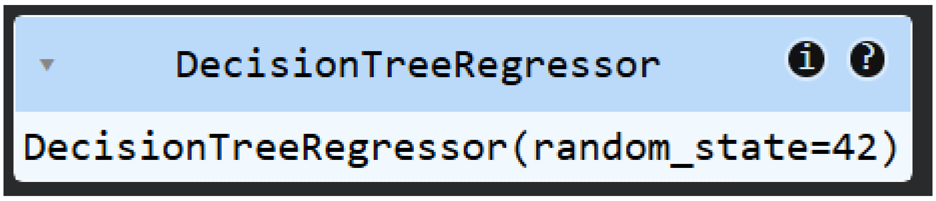
```
plt.figure(figsize=(14, 8))
plot_tree(
    pre_pruned_clf,
    feature_names=data.feature_names,
    class_names=data.target_names,
    filled=True
)
plt.show()
for name, importance in zip(
    data.feature_names,
    pre_pruned_clf.feature_importances_
):
    print(name, ":", round(importance, 4))
```



```
sepal length (cm) : 0.0176

sepal width (cm) : 0.0

petal length (cm) : 0.9374

petal width (cm) : 0.045
```

```
from sklearn.datasets import load_diabetes

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean_squared_error


data = load_diabetes()

X = data.data

y = data.target


X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)
```

```
reg = DecisionTreeRegressor(random_state=42)
reg.fit(X_train, y_train)
```



```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

```
y_pred = reg.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
```

```
Mean Squared Error: 4976.797752808989
```

```
pre_pruned_reg = DecisionTreeRegressor(
    max_depth=4,
    min_samples_leaf=10,
    random_state=42
)
pre_pruned_reg.fit(X_train, y_train)

print(
    "Pre-Pruned MSE:",
    mean_squared_error(y_test, pre_pruned_reg.predict(X_test))
)

pre_pruned_reg = DecisionTreeRegressor(
    max_depth=4,
    min_samples_leaf=10,
    random_state=42
)
```

```
pre_pruned_reg.fit(X_train, y_train)


print(
    "Pre-Pruned MSE:",
    mean_squared_error(y_test, pre_pruned_reg.predict(X_test))
)


path = reg.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas


pruned_models = []
mses = []


for alpha in ccp_alphas:
    model = DecisionTreeRegressor(
        ccp_alpha=alpha,
        random_state=42
    )
    model.fit(X_train, y_train)
    pruned_models.append(model)
    mses.append(
        mean_squared_error(y_test, model.predict(X_test))
    )


best_alpha = ccp_alphas[mses.index(min(mses))]
print("Best ccp_alpha:", best_alpha)
```