CSCE509 - Homework 4 - Decision Tree Classifier

C00580017

RAKSHYA PANDEY

Introduction and Objective

This report explores the learning behavior of a **Decision Tree Classifier** applied to non-linear synthetic datasets using make_moons() form sklearn.datasets under controlled noise and data size conditions. With this study I try to understand how different training set sizes and noise levels influence model test accuracy and overfitting behavior (measured via accuracy gap).

Then I build predictive models (linear and polynomial regression) for estimating the required training size based on noise, test accuracy, and overfitting.

Github Link:
https://github.com/rakshyaaa/DecisionTreeImplementation/blob/main/DecisionTreeImplementation.ipynb

Experimental Setup

Data Generation

Synthetic datasets were created using make_moons() with varying training sizes as [500, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000] and noise levels as [0.1, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5].

A fixed test set (1,500 samples, noise = 0.3) was used across all experiments to allow consistent performance comparisons.

Model Configuration

A **DecisionTreeClassifier** was used with the following hyperparameters:

max_depth = 10

min_samples_split = 30

min_samples_leaf = 10

criterion='entropy'

max_features='sqrt'

class_weight='balanced' and a fixed random_state = 42

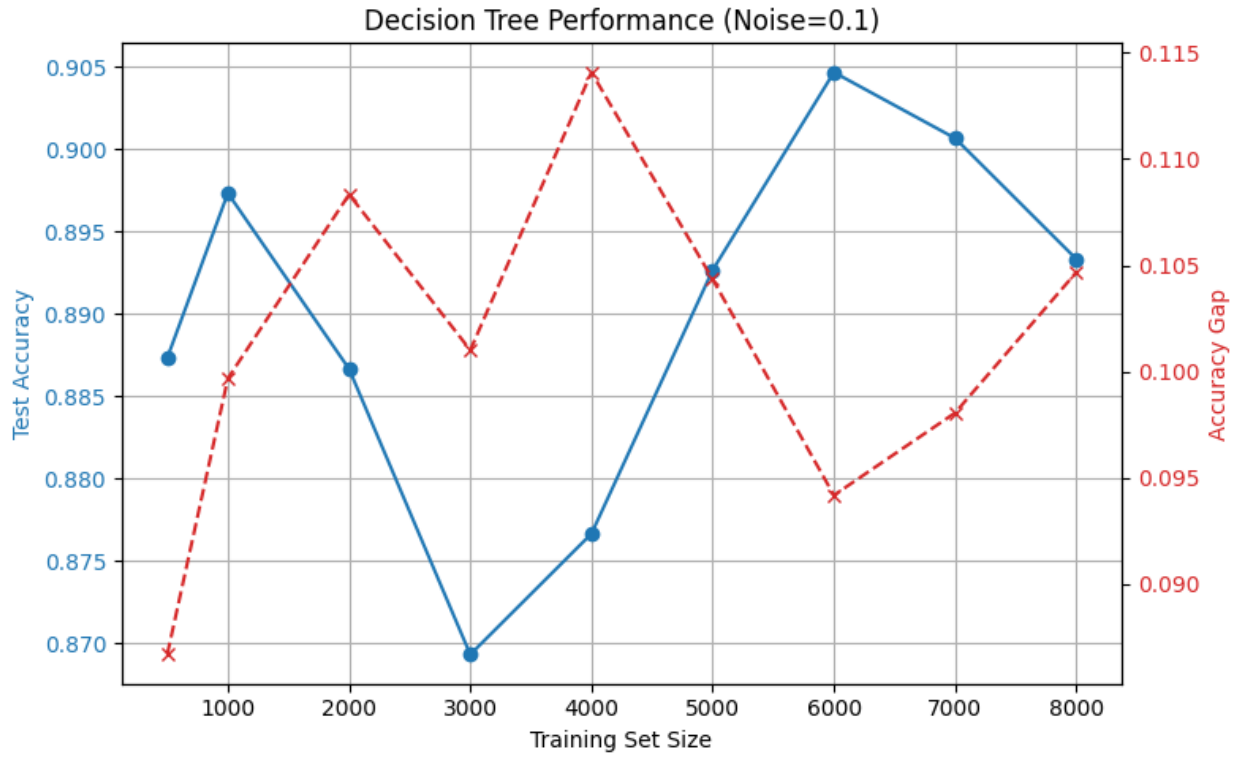
These hyperparameters were tuned to control model complexity and regularization.

Code Snippet:

```
results = []
all_results = []

for noise in noise_levels:

    noise_results = []
    for size in train_sizes:

        X_train, y_train = generate_data(size, noise)


        clf = DecisionTreeClassifier(max_depth = 10, min_samples_split=30,
min_samples_leaf=10,
criterion='entropy',max_features='sqrt',class_weight='balanced',random_state=
random_state)
        clf.fit(X_train, y_train)


        y_train_pred = clf.predict(X_train)
        y_test_pred = clf.predict(X_test)

        train_acc = accuracy_score(y_train, y_train_pred)
        test_acc = accuracy_score(y_test, y_test_pred)
        gap = train_acc - test_acc



        noise_results.append((size, train_acc, test_acc, gap))
        all_results.append({
            'noise': noise,
            'train_size': size,
            'train_acc': train_acc,
            'test_acc': test_acc,
            'gap': gap
        })
    results.append((noise, noise_results))

results_df = pd.DataFrame(all_results)
results_df
```

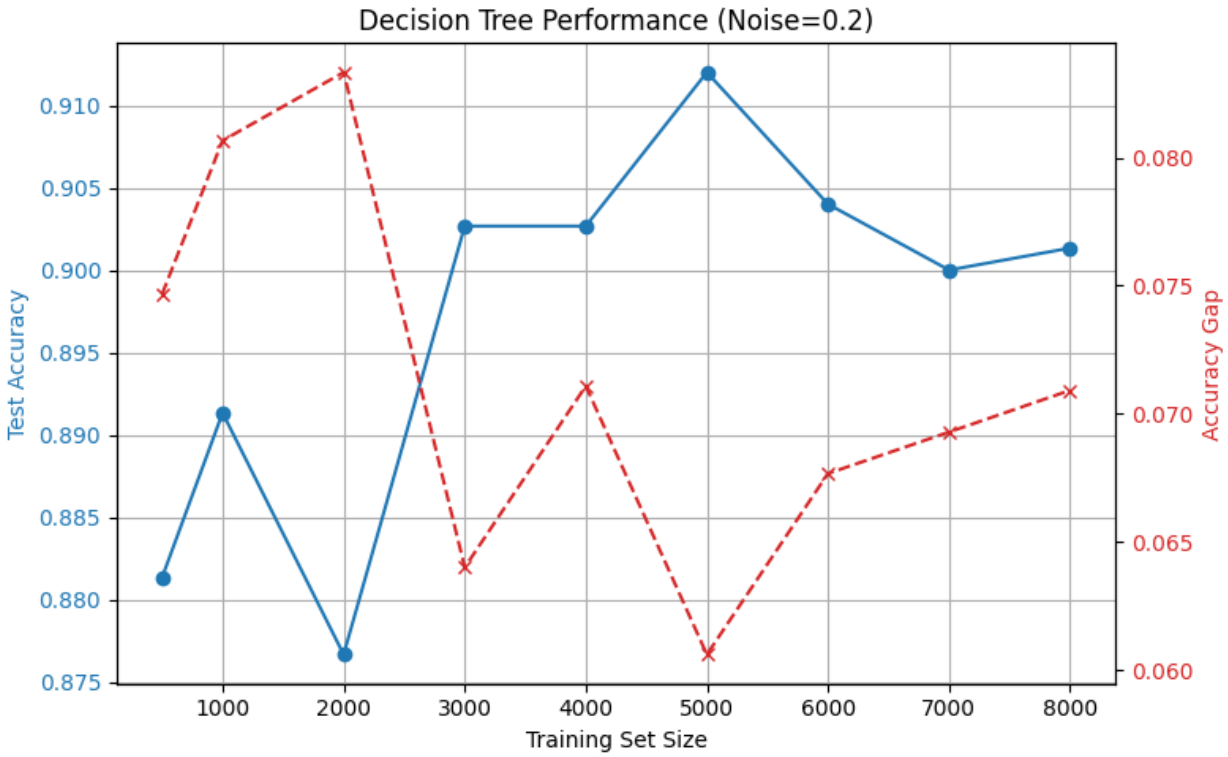
Classifier Performance Analysis

Noise = 0.1



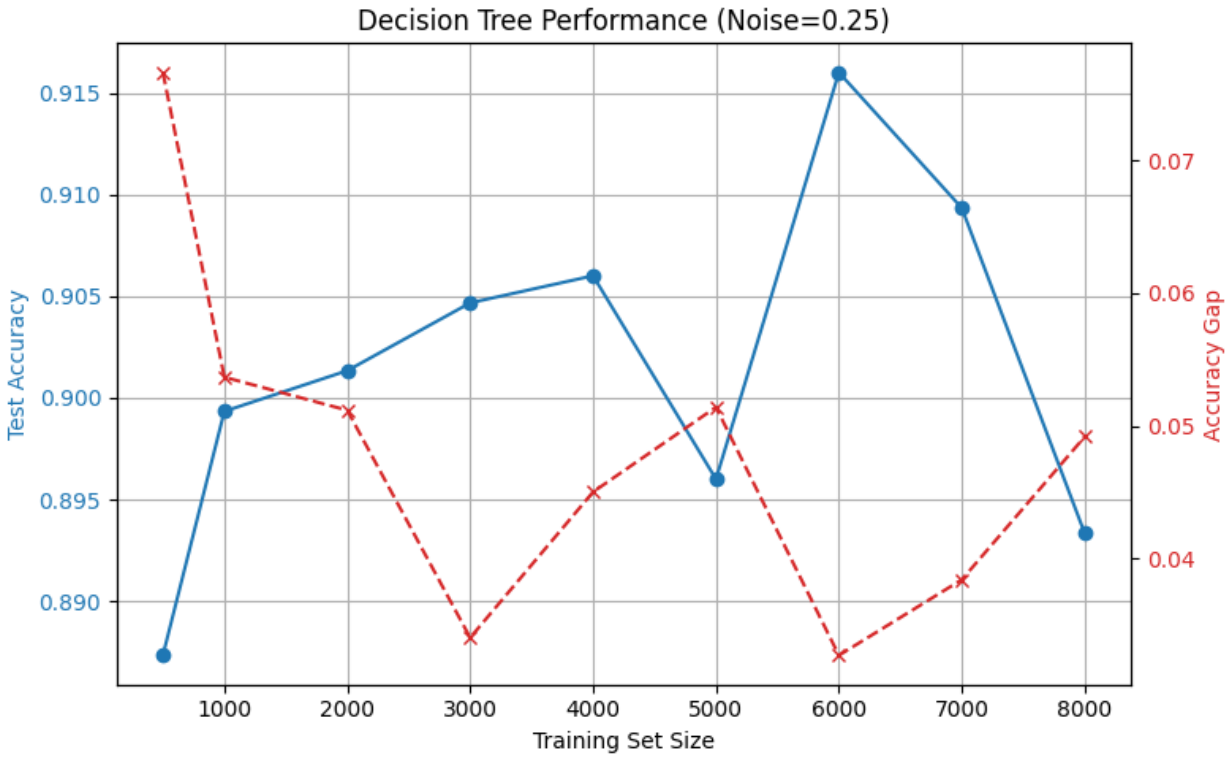
There are slight fluctuations in test accuracy, peaking at nearly 6000 samples. The accuracy gap is relatively high (~0.11) at smaller sizes but stabilizes at higher data volumes. With minimal noise, the model performs well but still needs enough data to avoid overfitting.

Noise = 0.2



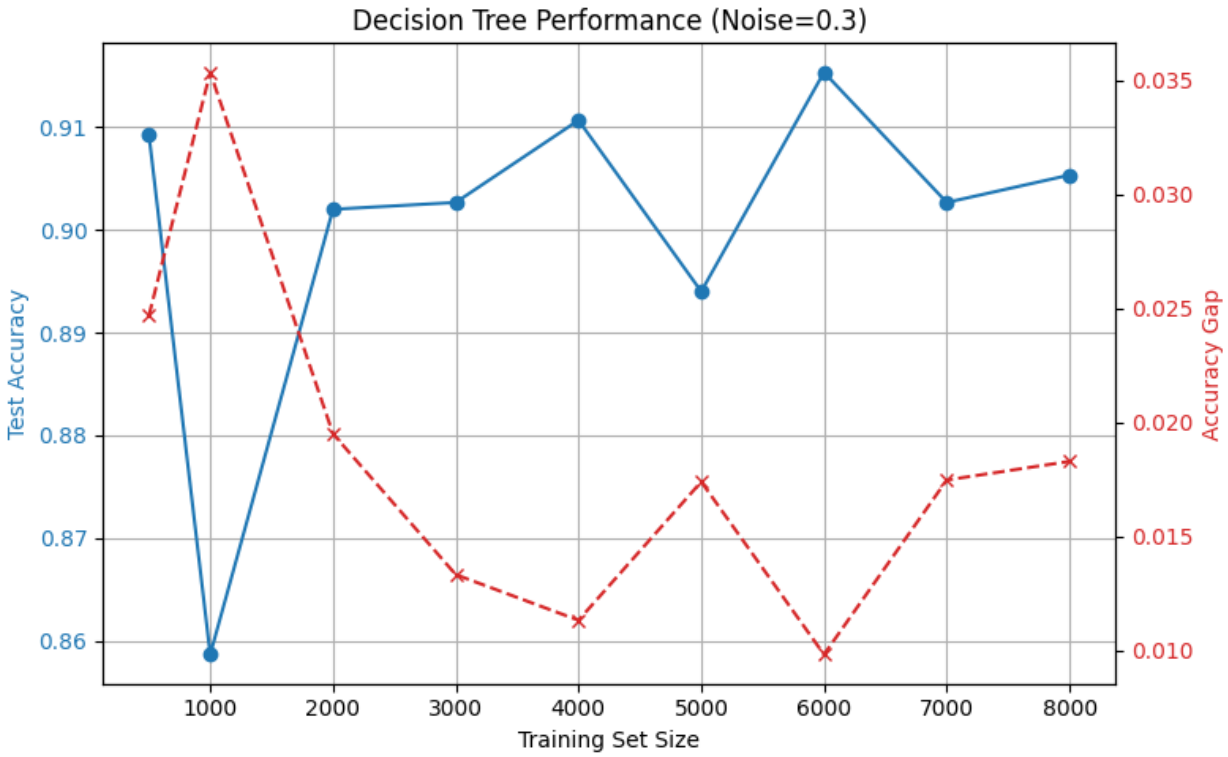
We can observe a better test accuracy, reaching 0.91+ with 5000 samples. The Gap decreases with training size, suggesting better generalization. A moderate level of noise actually helps avoid overfitting if enough data is present.

Noise = 0.25



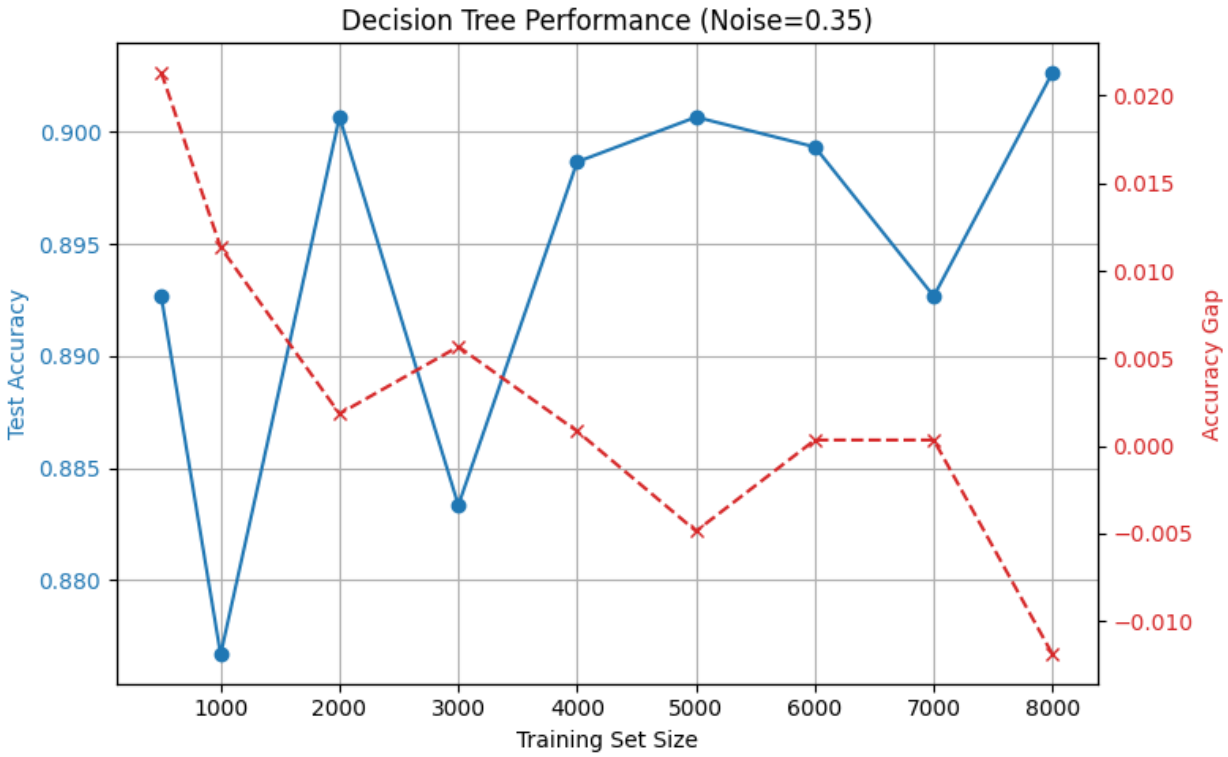
The test accuracy remains consistently high (≥0.89) across sizes.

Noise = 0.3



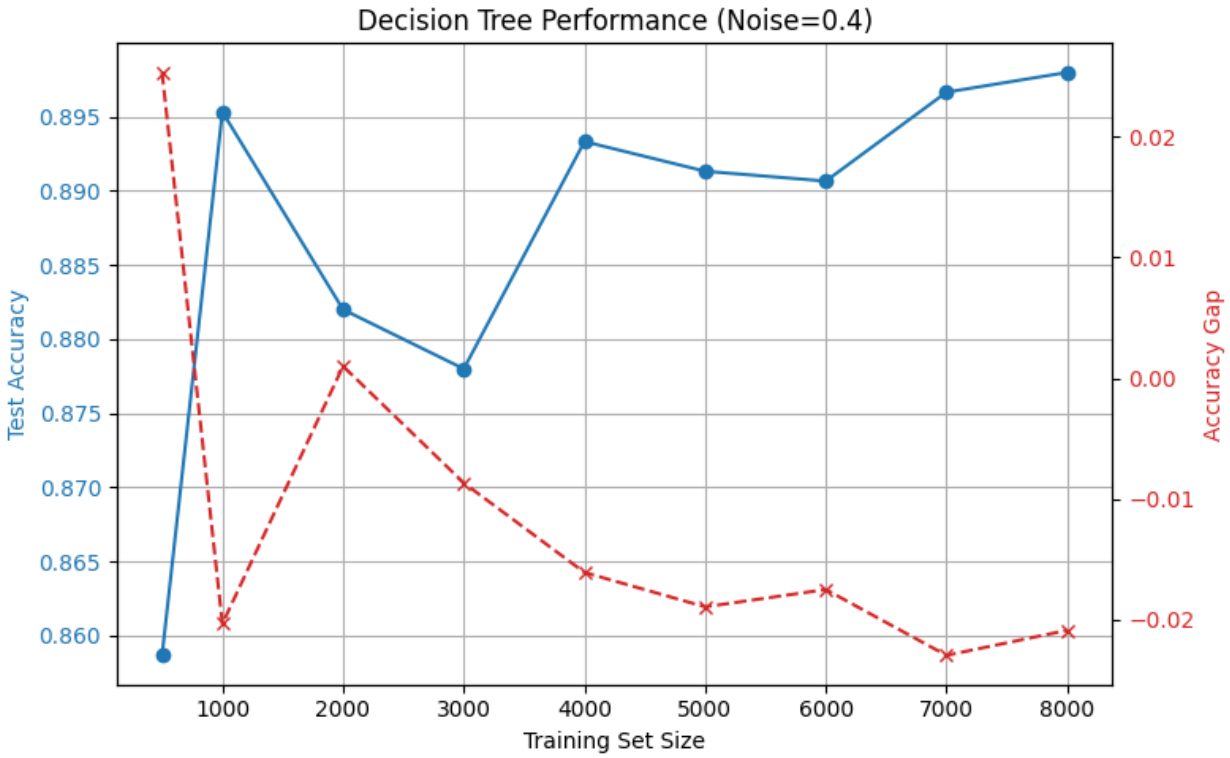
The accuracy remains strong, with a peak near 6000 samples. The gap declines to ~0.01 by 6000+ samples. This match expected performance since test set is also at noise = 0.3. This confirms low overfitting with medium noise and large datasets.

Noise = 0.35



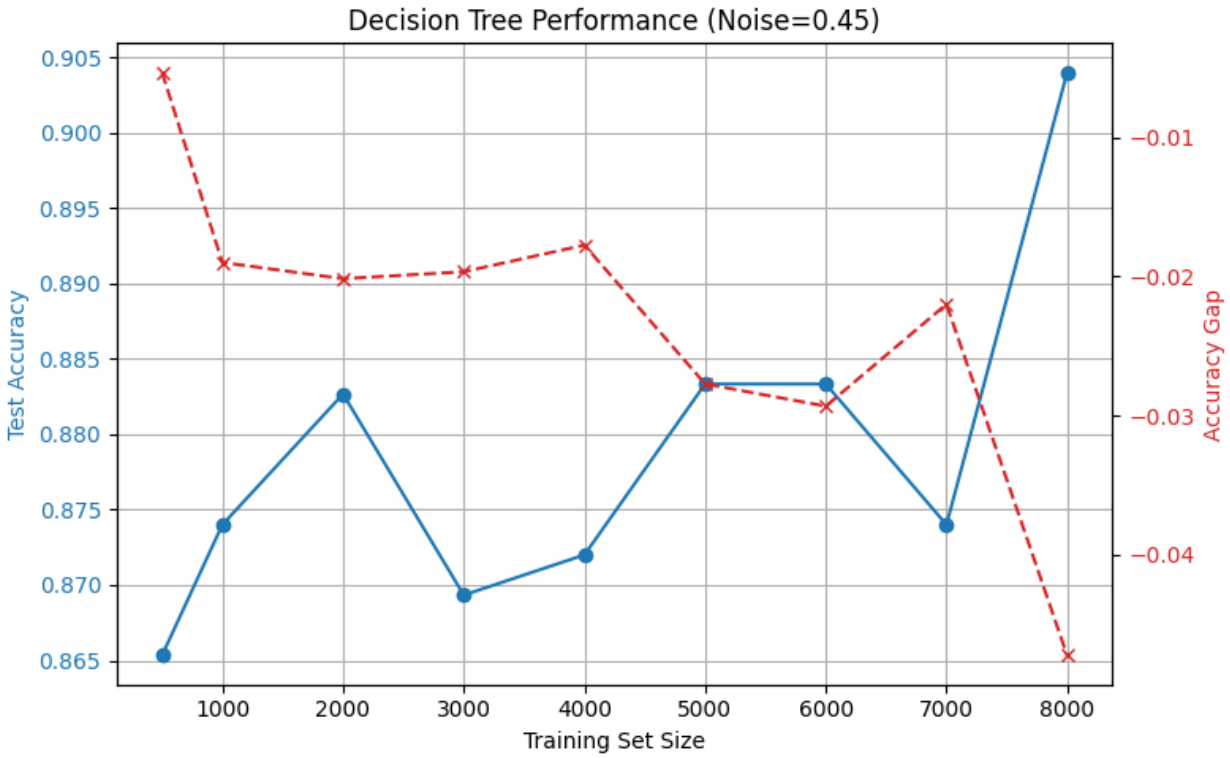
Here, the performance remains stable. The gap shrinks and even becomes negative, suggesting slight underfitting. We can conclude at higher noise, overfitting disappears, possibly due to model regularization.

Noise = 0.4



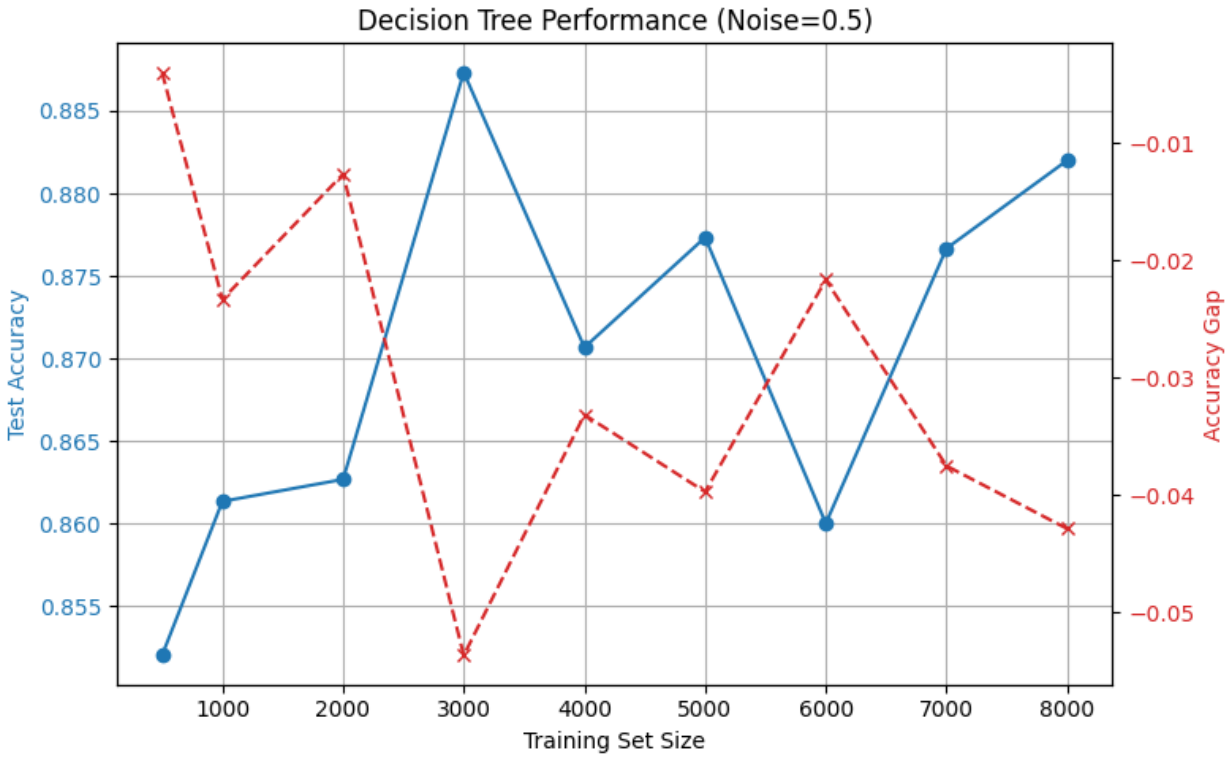
The test accuracy starts dipping but improves with data size. The gap is mostly negative, reinforcing the model's shift from overfitting to slight underfitting. We can conclude, decision trees handle noisy data better with larger datasets but suffer a performance plateau.

Noise = 0.45



The accuracy hovers in the 0.87–0.89 range. The gap is consistently negative. Hence, when the decision boundary becomes conservative, models resist fitting too closely due to regularization with high noise.

Noise = 0.5



The accuracy is more volatile but generally consistent. The gap is significantly negative (~-0.04), implying a regularized decision boundary. With 50% noise, further increasing training size has limited benefits. Decision trees reach their effective capacity.

Regression-Based Prediction of Training Size (N = f(Q, P) using Linear Regression)

After analyzing how model performance varies with noise and training size, the second part of this report focuses on building predictive models to estimate the required training set size based on observed noise levels and classifier performance.

Feature Engineering for Regression

To build the regression models, a feature set was constructed as follows for each (noise, training size) combination:

Noise: The level of noise used to generate the data.

Test Accuracy: How well the model performed on the fixed test set.

Accuracy Gap: The difference between training accuracy and test accuracy, a measure of overfitting.

Noise × Test Accuracy: An interaction term that captures how the joint effect of noise and accuracy influences the required training size.

Noise^2: A non-linear transformation of noise to capture any quadratic relationship.

Test Accuracy^2: A non-linear transformation of test accuracy to capture diminishing returns at higher accuracies.

These features were stored in a features array, and the corresponding training sizes were stored in a targets array.
Essentially, the machine learning task is framed as a supervised regression problem:

Inputs (features): Information about the noise and classifier performance.

Output (target): The training set size used.

Code Snippet:

```
features = []   # (noise, test accuracy, gap, noise*test_acc, noise^2, test_acc^2)

targets = []   # training set size

for noise, noise_results in results:

    for size, train_acc, test_acc, gap in noise_results:

        features.append([

            noise,               # noise

            test_acc,             # test accuracy

            gap,                  # train-test gap

            noise * test_acc,     # interaction term

            noise ** 2,           # noise squared

            test_acc ** 2         # test accuracy squared

            ])

        targets.append(size)

features = np.array(features)
```

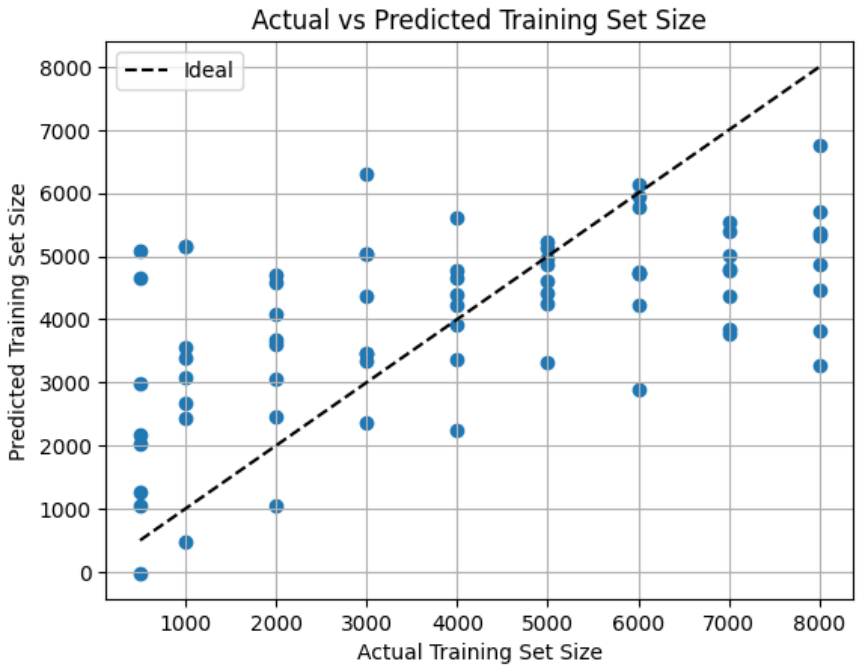
```
targets = np.array(targets)
```

Linear Regression Results

A simple linear regression model was first fitted using the constructed features.

I received a R^2 Score of 0.304. The points are moderately scattered around the ideal line, indicating a poor-to-moderate fit. We can see that a linear model struggles to capture the complex relationship between noise, test performance, and required data size, confirming that the relationship is non-linear.

```
Fitted Linear Model: N = a*Q + b*P + c
Coefficients (a, b): [  75283.77438843  672883.11995098  -65760.78477148 -138109.53657966
   45353.52681003 -318287.71154206]
Intercept (c): -331397.5120988165
Linear R²: 0.304
```

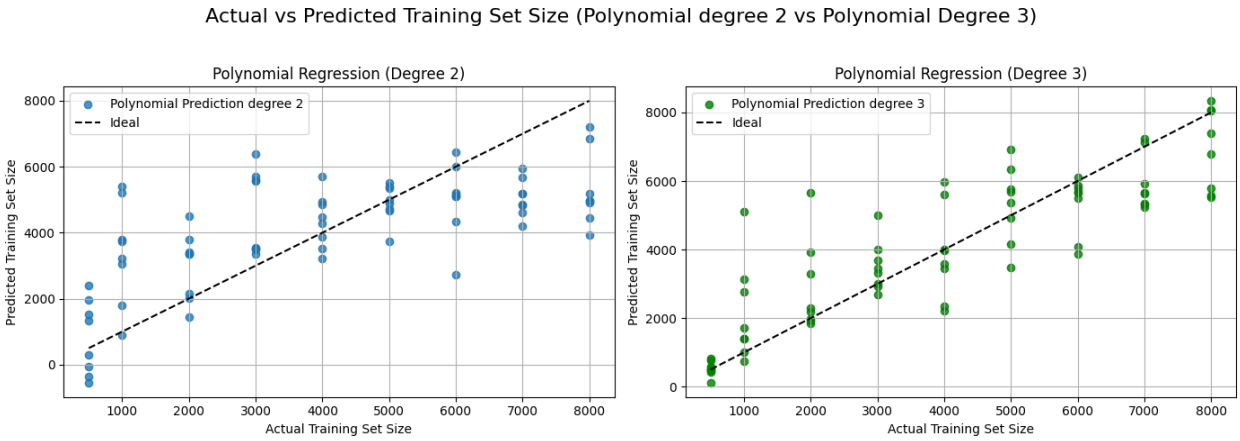


Polynomial Regression Results

Recognizing the limitations of the linear model, polynomial regression models of degree 2 and 3 were trained on the same features. With Degree 2 Polynomial model I observed a R^2 Score as 0.456 and with the Degree 3 Polynomial model, I observed a R^2 Score as 0.745.

For Degree 2 Polynomial model, the prediction line fits closer than the linear model but still has considerable errors.

For Degree 3 Polynomial model, the predictions align much closer to the ideal line, significantly reducing the error.



The degree 3 model demonstrated a strong capability to predict how much training data is necessary for a given noise and test accuracy level.

Conclusion

With this, I successfully explored the effects of noise levels and training set sizes on the generalization performance of a Decision Tree Classifier using synthetic two-class datasets. It was observed that:

1. Increasing training set size consistently improved test accuracy and reduced the overfitting gap across different noise conditions.
2. Higher noise levels made the learning task harder but naturally acted as a form of regularization, reducing overfitting tendencies.
3. Large enough datasets (greater than 4000 samples) were crucial in achieving stable performance, especially as noise levels increased.

In the second part, predictive models were built to estimate the required training size based on noise and performance metrics.

1. A linear regression model provided only moderate predictive ability ($R^2 = 0.304$).
2. Polynomial regression models (particularly degree 3) captured the complex patterns much more effectively, achieving a strong R^2 score of 0.75.

The combination of careful feature engineering and non-linear modeling proved essential in accurately predicting the training data requirements.