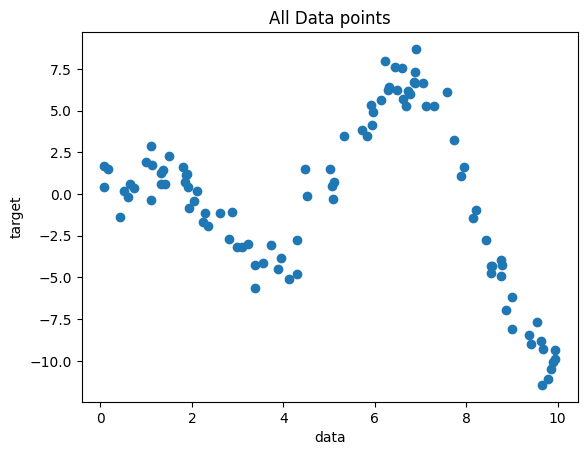
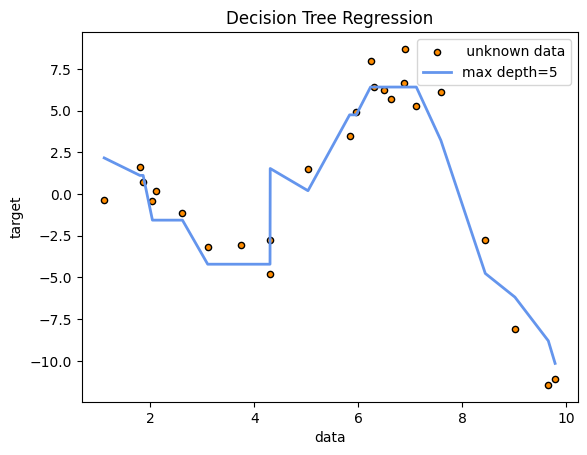
**Decision Tree and SVM Regression Models: Comparison and Hyperparameter Exploration**

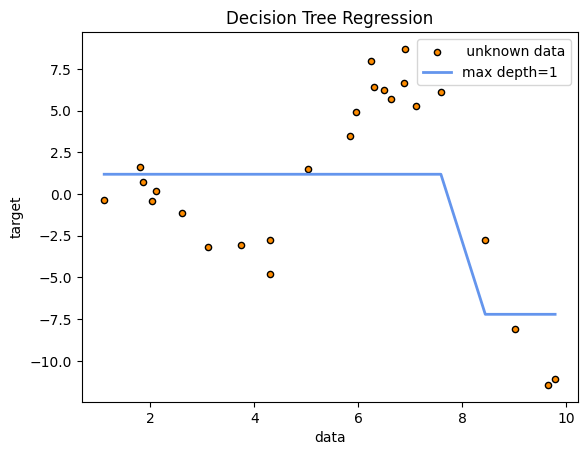
**Original plot:**



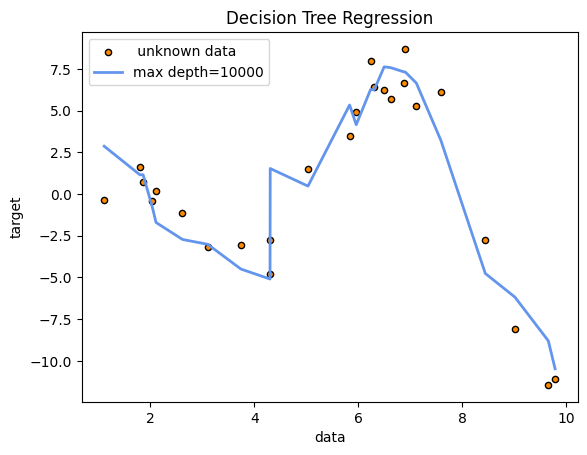
**Max depth = 5 (Good fit):**



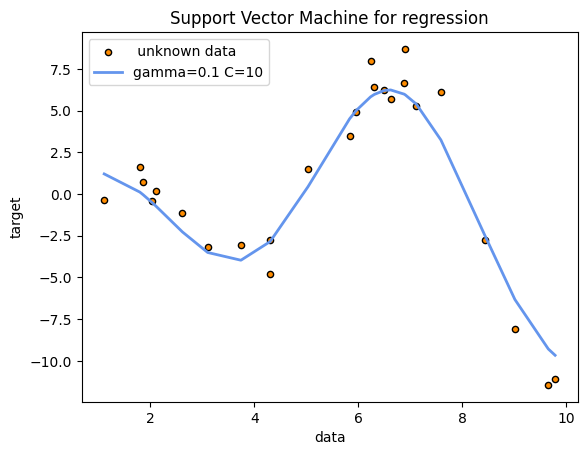
**Max depth = 1 (Under fit):**



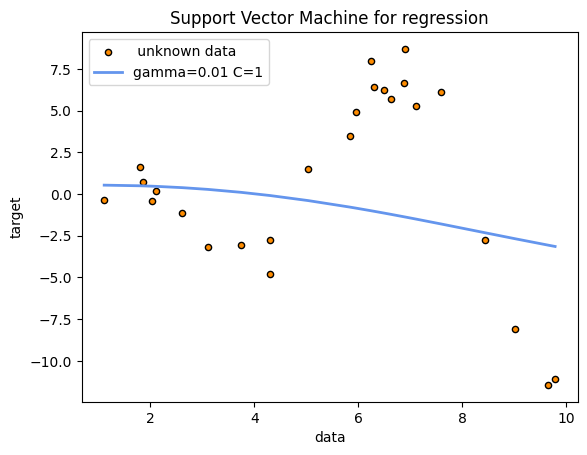
**Max depth = 10000 (Over fit):**



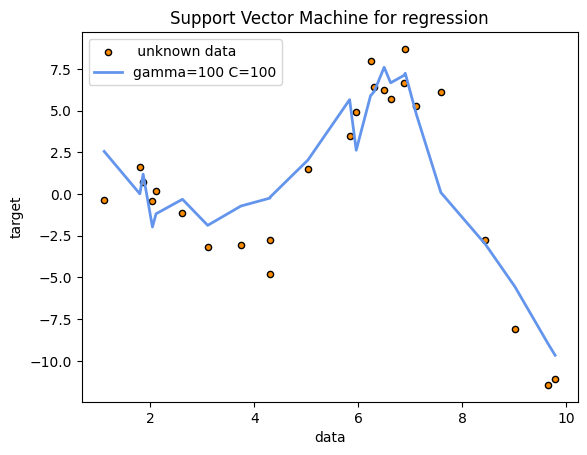
**Gamma = 0.1, C = 10 (Good fit):**



**Gamma = 0.01, C = 1 (Under fit):**



**Gamma = 100, C = 100 (Over fit):**



**Questions:**

1. How does changing the max\_depth hyperparameter affect the Decision Tree Regression model's performance?

2. What do you observe when varying the C and gamma hyperparameters of the SVM Regression model? How does each hyperparameter affect the model's flexibility and ability to fit the data?

3. If someone wants to work with this specific dataset, what model would you recommend him work with?

**Answers:**

1. The performance of the model changed radically with the altering of the max\_depth hyperparameter. If it was too low, like max\_depth = 1, the mean square error of the model was very high (~16.80), so the model was very inaccurate at predicting the target value. This can be seen at the plot with the model having only 2 average values for the data, which makes sense as the tree had only 1 node and 2 leafs bellow it. At a max\_depth =5, the results were much better, with the mean square error becoming the lowest (~2.75), so the prediction of unknown data was pretty good. The plot also followed the concentration of the data much better. At a higher max\_depth, the results started becoming worse though. For example, at an extreme max\_depth = 10000 the mean square error was (~3.12). There were also signs that the model was trained on the “noise” of the training data with sharp angles that led away from the main concentrations of the data.

2. With the change of the C and gamma hyperparameters of the SVM Regression model, there were profound consequences on the accuracy of the model. The gamma value change could be seen visually with the curves of the plot and how well it would follow the cluster of the data. Meanwhile, the C value change could be seen visually with how close the prediction would follow the cluster of the data, or in other words what tolerance it had for outliers in the data. With a smaller C value, there would be a larger margin, so better tolerance for errors. With gamma = 0.1 and C = 10, the results were fairly good (mse ~1.80). With a very small gamma = 0.01 and a small C = 1, the plot would barely even follow the cluster of data, with massive error (mse ~31.5). Finally, with a very high gamma = 100 and fairly high C = 100, the model was over-fitted on the training data, with signs that it “followed” specific training data points. It also displayed a higher error (mse ~4.85).

3. If someone wanted to work with this specific dataset, I would recommend them to work with the SVM. It had lower errors, as well as the ability to adapt very well with the specific requirements of the person who is using it. Its hyperparameters allow far more fine-tuning than that of the Decision Tree.

**Code:**

The code was written in a .ipybn file for ease of use.

import numpy as np

# Function to approximate

a = 6 # Last number of your AEM

f = lambda x: x \* np. cos(x + a)

# Generate 100 random points with distribution f and noise factor of 1

x = np.random.uniform(0, 10, 100)

y = f(x) + np.random.normal(scale=1, size=100)

X = x[:, np.newaxis]

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

from sklearn.svm import SVR

def regression\_tree(X\_train, y\_train, X\_test, y\_test, max\_depth):

    decision\_tee\_rgr = DecisionTreeRegressor(max\_depth=max\_depth)

    decision\_tee\_rgr.fit(X\_train, y\_train)

    y\_pred\_rt = decision\_tee\_rgr.predict(X\_test)

    max\_depth =str(max\_depth)

    label = "max depth="

    label += max\_depth

    plt.figure()

    plt.scatter(X\_test, y\_test, s=20, edgecolor="black", c="darkorange", label=" unknown data")

    plt.plot(X\_test, y\_pred\_rt, color="cornflowerblue", label=label, linewidth=2)

    plt.xlabel("data")

    plt.ylabel("target")

    plt.title("Decision Tree Regression")

    plt.legend()

    plt.show()

    mse = mean\_squared\_error(y\_test, y\_pred\_rt)

    print(f"Mean Squared Error: {mse:.2f}")

def svm\_regression(X\_train, y\_train, X\_test, y\_test, gamma, C):

    svr = SVR(kernel='rbf', gamma=gamma, C=C)

    svr.fit(X\_train, y\_train)

    y\_pred\_svr = svr.predict(X\_test)

    gamma =str(gamma)

    C =str(C)

    label = "gamma="

    label += gamma

    label += " C="

    label += C

    plt.figure()

    plt.scatter(X\_test, y\_test, s=20, edgecolor="black", c="darkorange", label=" unknown data")

    plt.plot(X\_test, y\_pred\_svr, color="cornflowerblue", label=label, linewidth=2)

    plt.xlabel("data")

    plt.ylabel("target")

    plt.title("Support Vector Machine for regression")

    plt.legend()

    plt.show()

    mse = mean\_squared\_error(y\_test, y\_pred\_svr)

    print(f"Mean Squared Error: {mse:.2f}")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, shuffle=True)

data\_test = np.column\_stack((X\_test,y\_test))

data\_test = data\_test[data\_test[:, 0].argsort()]

X\_test=data\_test[:, 0]

y\_test=data\_test[:,1]

X\_test=X\_test[:, np.newaxis]

y\_test=y\_test[:, np.newaxis]

plt.scatter(X, y)

plt.title("All Data points")

plt.xlabel("data")

plt.ylabel("target")

plt.show()

plt.scatter(X\_test, y\_test)

plt.title("Test Data points")

plt.xlabel("data")

plt.ylabel("target")

plt.show()

#Good fit

regression\_tree(X\_train, y\_train, X\_test, y\_test, 5)

#Under fit

regression\_tree(X\_train, y\_train, X\_test, y\_test, 1)

#Overfit

regression\_tree(X\_train, y\_train, X\_test, y\_test, 10000)

#########SVM#########

#Good fit

svm\_regression(X\_train, y\_train, X\_test, y\_test, 0.1, 10)

#Underfit

svm\_regression(X\_train, y\_train, X\_test, y\_test, 0.01, 1)

#Overfit

svm\_regression(X\_train, y\_train, X\_test, y\_test, 100, 100)