**Machine Learning Algorithms Part2**

**🔍 What is Support Vector Regression (SVR)?**

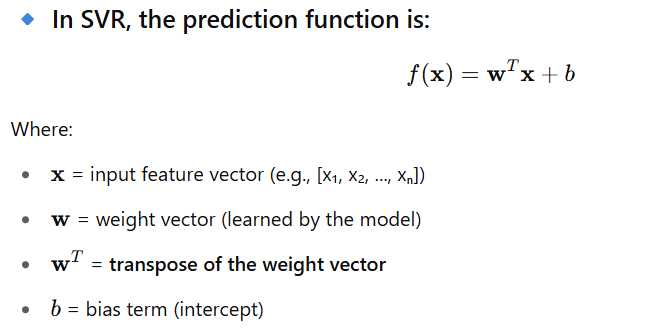
* + 1.1.1.2 Support Vector Regression (SVR)

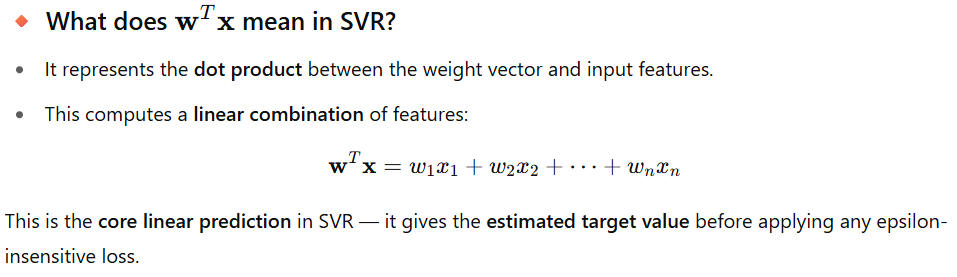
**ε (epsilon) – Epsilon-Insensitive Tube**

* **Name:** Epsilon (Greek letter)
* **Meaning:** Defines a **margin of tolerance** where no penalty is given.
* **Role:** The model does not care (i.e., is *insensitive*) to errors within ±ε range from the actual value.
* **Also called:** Epsilon-insensitive loss margin or epsilon-tube.

👉 Think of it as a "safe zone" — predictions within this margin are not penalized.

**🔹 w – Weight Vector**

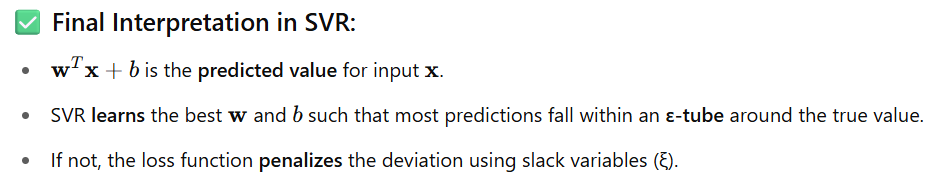




**Why the transpose?**

* Vectors w and x are typically column vectors.
* To compute the dot product, we need one to be a **row vector** — so we use the **transpose** of





**🔹 ξ and ξ\* (xi and xi-star) – Slack Variables**

* **Name:** Slack variables (Greek letter xi: ξ)
* **Meaning:** Measure how far the predictions are **outside** the epsilon margin.
* **Role:**
  + **ξi** : deviation above ε
  + **ξ\***​: deviation below -ε
* **Also called:** Loss variables, error distances, or constraint violations

👉 They represent how much a prediction **violates** the margin — these values are **penalized** during training.

**Support Vector Regression (SVR) – A Deep Dive**

**✅ Category:**

* Supervised Learning
* Regression Algorithm (predicts continuous values)
* Based on **Support Vector Machines (SVM)**

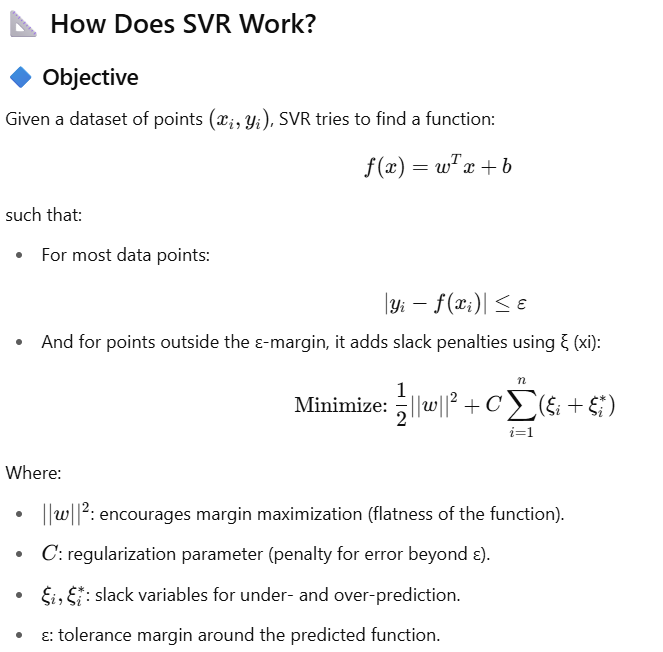
**What is Support Vector Regression (SVR)?**

SVR is a type of regression algorithm that tries to find a function that approximates the data within a certain acceptable error margin (ε).

Unlike linear regression that tries to minimize the total error (like MSE), SVR:

* Tries to fit the best hyperplane (or curve) within an ε-insensitive tube.
* Only penalizes predictions that fall outside this ε margin.
* Focuses on support vectors: the data points that lie on the edge or outside the ε-tube.

[**https://www.geeksforgeeks.org/support-vector-regression-svr-using-linear-and-non-linear-kernels-in-scikit-learn/**](https://www.geeksforgeeks.org/support-vector-regression-svr-using-linear-and-non-linear-kernels-in-scikit-learn/)

****

**Graphical Intuition**

* SVR creates a tube of width 2ε around the regression line.
* Points inside the tube are considered correctly predicted.
* Only the points outside the tube influence the model — these are the support vectors.

**🧠 Key Concepts**

| **Term** | **Meaning** |
| --- | --- |
| **Epsilon (ε)** | ε-tube width: determines how much deviation from actual y is acceptable |
| **Support Vectors** | Data points that lie outside the ε margin |
| **Kernel Trick** | Maps input features into higher-dimensional space (for nonlinear regression) |
| **Regularization (C)** | Controls the trade-off between flatness and tolerance for outliers |

**🧮 Kernel Functions in SVR**

SVR can perform both **linear** and **non-linear** regression using kernels.

| **Kernel Type** | **Description** |
| --- | --- |
| Linear | Works well when data is linearly separable |
| Polynomial | Fits polynomial curves |
| RBF (Gaussian) | Popular for non-linear problems |
| Sigmoid | Behaves like a neural network |

**🎯 Use Cases of SVR**

| **Domain** | **Application** |
| --- | --- |
| Finance | Stock price forecasting |
| Healthcare | Predicting patient recovery time |
| Energy | Load forecasting in power grids |
| Engineering | Sensor signal estimation |
| Marketing | Forecasting ad campaign performance |

**⚙️ How SVR Works (Training Process)**

1. Choose a kernel (linear or nonlinear)
2. Define the ε margin
3. Use support vectors that fall outside the margin to define the model
4. Optimize using **Quadratic Programming** to find the best fit with minimum deviation

**📈 Assumptions of SVR**

* Data may or may not be linearly separable (kernels help with nonlinear data)
* It can tolerate some outliers (controlled via C parameter)
* Works best with **normalized or scaled data**

**Dataset: California Housing (predicting median house value)**

# 📌 Step 1: Import Libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.svm import SVR

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.pipeline import make\_pipeline

# 📌 Step 2: Load Dataset

data = fetch\_california\_housing()

X = data.data # Features

y = data.target # Target: Median house value

# Let's take only 1 feature for visualization (e.g., average rooms per household)

X = X[:, [3]] # 'AveRooms'

# 📌 Step 3: Train/Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 📌 Step 4: Define SVR Model Pipeline (with scaling)

svr\_model = make\_pipeline(

StandardScaler(), # Normalize data

SVR(kernel='rbf', C=100, epsilon=0.1) # RBF kernel

)

# 📌 Step 5: Train the Model

svr\_model.fit(X\_train, y\_train)

# 📌 Step 6: Make Predictions

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

# 📌 Step 7: Evaluate the Model

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

r2 = r2\_score(y\_test, y\_pred)

print("✅ Mean Squared Error (MSE):", mse)

print("✅ R^2 Score:", r2)

# 📌 Step 8: Visualize

plt.figure(figsize=(10,6))

plt.scatter(X\_test, y\_test, color='blue', label='Actual')

plt.scatter(X\_test, y\_pred, color='red', alpha=0.6, label='Predicted')

plt.xlabel('Average Rooms per Household')

plt.ylabel('Median House Value')

plt.title('SVR - California Housing')

plt.legend()

plt.grid(True)

plt.show()

🔍 Result Explanation

* Mean Squared Error (MSE): Measures average squared difference between actual and predicted values (lower is better)
* R² Score: Indicates how well the predictions match the actual values (1 is perfect)

**✅ Pros and Cons**

**✅ Pros:**

* Works for both linear and non-linear data
* Can model complex relationships with kernel tricks
* Robust to outliers (due to ε margin)
* High generalization capability

**❌ Cons:**

* Computationally expensive for large datasets
* Choosing the right kernel and tuning parameters (C, ε, gamma) is critical
* Difficult to interpret compared to linear regression

**🔎 When to Use SVR?**

* When the relationship between input and output is **non-linear**
* When **outliers** are present in your dataset
* When you want **flexible and smooth fitting**
* When dataset is **small to medium-sized**

**Summary Table**

| **Concept** | **Meaning** |
| --- | --- |
| ε-tube | The margin where predictions are not penalized (penalties) |
| C | Penalty parameter — controls tolerance to errors |
| Support Vectors | Points outside the ε margin — drive the model |
| Kernels | Functions to transform data into higher dimensions |
| Slack variables | Allow some points to be outside the ε margin |

* + **1.1.1.3 Decision Tree Regressor**

<https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/>

<https://www.youtube.com/watch?v=b_4nzmNK3tY&list=PLKnIA16_RmvYGY_n9PP8zN-0LG9MoZRjU&index=5>

entropy in decision tree

Information Gain

Gini impurity in depth intuition

Handling numerical/categorical values

Visualizing a decision tree

Overfitting and underfitting in decision tree

Decision tree hyperparameters

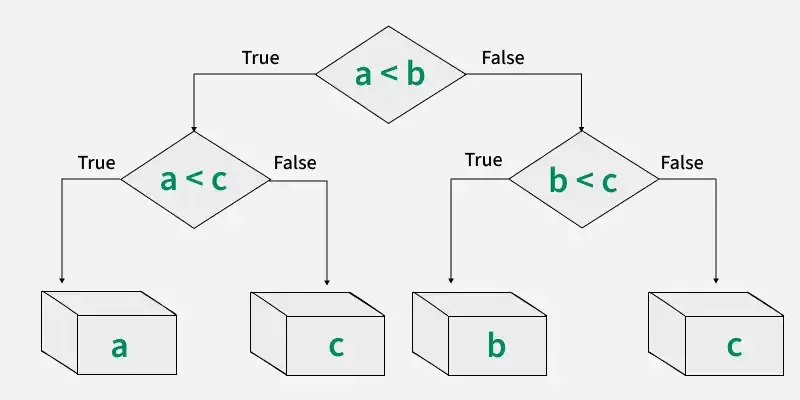
Hyper parameter tuning using grid search CV

**What is it?**

A **Decision Tree Regressor** splits the dataset into smaller subsets based on feature values and learns simple decision rules at each node to predict continuous values. It is non-linear and non-parametric.

Decision Tree Regression is a method used to predict continuous values like prices or scores by using a tree-like structure. It works by splitting the data into smaller parts based on simple rules taken from the input features. These splits help reduce errors in prediction. At the end of each branch, called a leaf node the model gives a prediction usually the average value of that group. In the tree:

* **Decision Nodes** (shown as diamonds) ask yes/no questions about the data, like “Is age greater than 50?”
* **Leaf Nodes** (shown as rectangles) give the final predicted number based on the data that reached that point.



Workflow of Decision Tree Regression

Branches connect nodes and represent the outcome of a decision. For example if the answer to a condition is "Yes," you follow one branch; if "No," you follow another. In below example it shows a decision tree that evaluates the smallest of three numbers:

**Entropy in decision trees:** represents the disorder or uncertainty within a dataset, specifically regarding the target variable. It's a measure of how "mixed" or "random" the classes are in a set of data. Decision trees use entropy to determine the best attribute for splitting the data, aiming to create subsets with lower entropy (higher purity) at each node. This process, known as information gain, helps the tree build a model that can accurately classify data by reducing uncertainty at each split.

Elaboration:

* **What is Entropy?**

In the context of decision trees, entropy is a measure of randomness or impurity in a dataset. A high entropy value indicates that the dataset is highly disordered, meaning the target variable (the variable you are trying to predict) is not easily predictable based on the features in the dataset. Conversely, a low entropy value indicates that the dataset is more homogeneous, meaning the target variable is more predictable.

* **How it's used in Decision Trees:**

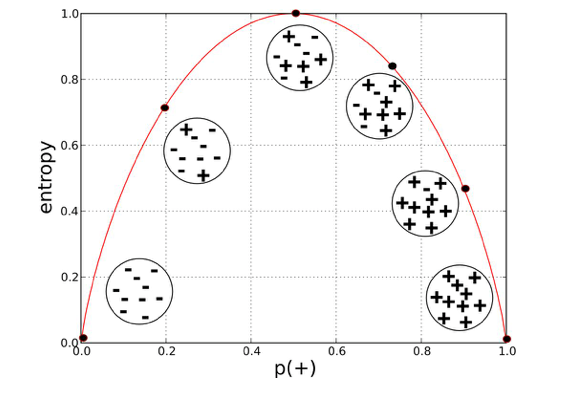
Decision tree algorithms use entropy as a key metric to decide how to split the data at each node. The goal is to split the data in a way that reduces entropy and creates subsets that are as pure as possible (meaning they are dominated by a single class or outcome).

* **Information Gain:**

The reduction in entropy achieved by a particular split is called information gain. Decision trees aim to maximize information gain, which means selecting the split that reduces entropy the most.

* **Example:**

Imagine you have a dataset of people, and you're trying to predict whether they will buy a product (yes or no). If the dataset is highly mixed (50% yes, 50% no), the entropy is high. If you split the data based on a feature like "age," and the resulting subsets are mostly yes or mostly no, the entropy is reduced, and you've gained information about the relationship between age and buying behavior.



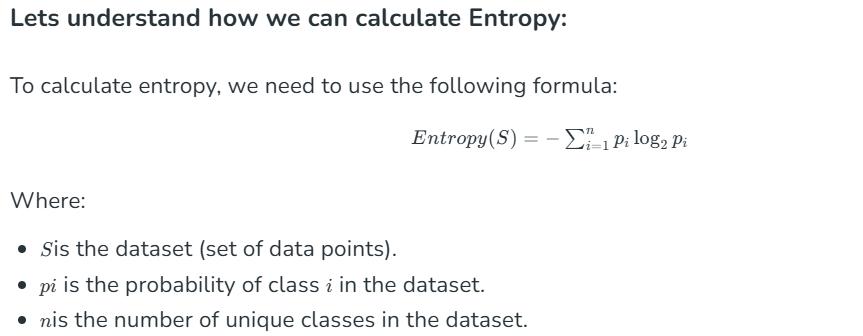
In [decision tree](https://www.geeksforgeeks.org/decision-tree/) algorithms, **entropy** is a critical measure used to evaluate the impurity or uncertainty within a dataset. By understanding and calculating entropy, you can determine how to split data into more homogenous subsets, ultimately building a better decision tree that leads to accurate predictions. Concept of entropy originates from information theory, where it quantifies the amount of "surprise" or unpredictability in a set of data.

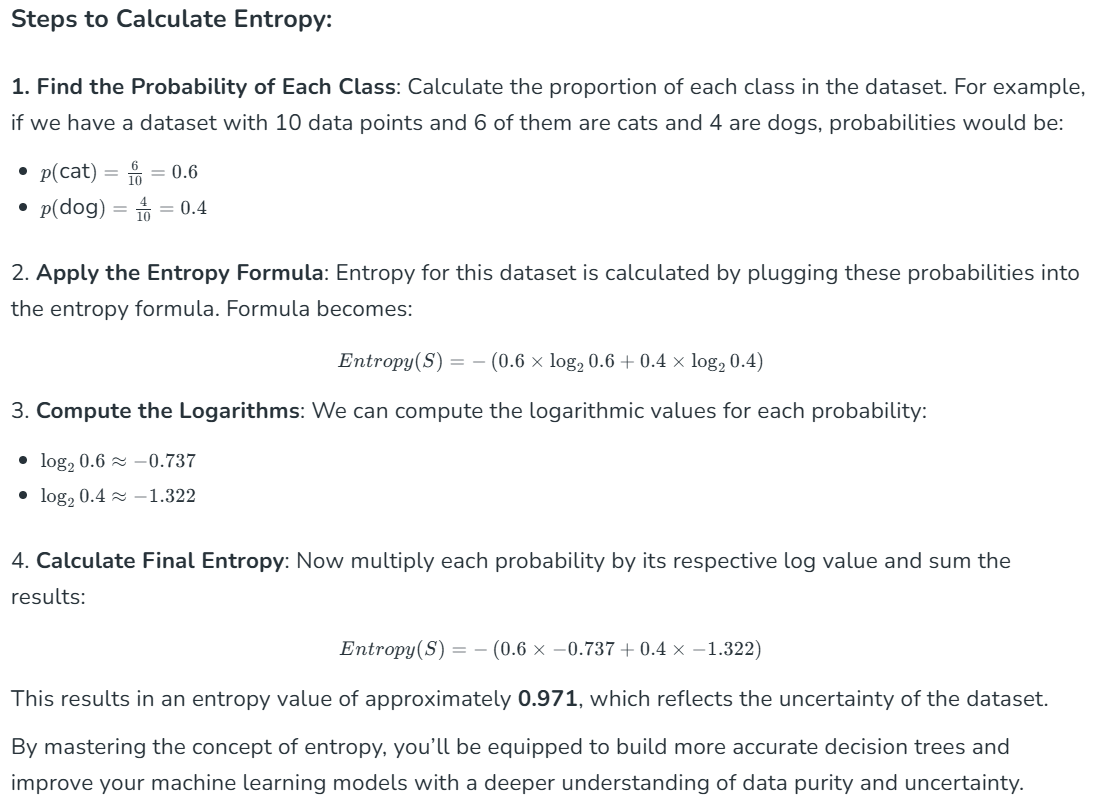
**Understanding Entropy**

Entropy is a measure of **uncertainty** or **disorder**. In the terms of decision trees, it helps us understand how mixed the data is. If all instances in a dataset belong to one class, entropy is **zero**, meaning the data is perfectly pure. On the other hand, when the data is evenly distributed across multiple classes, entropy is **maximum**, indicating high uncertainty.

* **High Entropy**: Dataset has a mix of classes, meaning it's uncertain and impure.
* **Low Entropy**: Dataset is homogeneous, with most of the data points belonging to one class.

Entropy helps in choosing which feature to split on at each decision node in the tree. Goal is to reduce entropy with each split to create subsets that are as pure as possible.

****



<https://www.geeksforgeeks.org/how-decision-tree-depth-impact-on-the-accuracy/>

**What is Max depth of decision tree:**

The maximum depth of the selection tree is a critical parameter that influences the version's complexity. When set to None, the tree continues to enlarge nodes till all leaves are pure or until each leaf consists of fewer samples than the specified `min\_samples\_split`. The depth of the tree is a key thing in controlling its complexity: growing intensity usually ends in extra complexity, which may bring about overfitting, even as decreasing intensity decreases complexity, probably causing underfitting. It is critical to strike a balance and choose an appropriate intensity to gain choicest version performance.

**What is the impact of depth on Accuracy?**

1. **Underfitting (Shallow Trees)**: When a decision tree is too shallow, it may not capture enough of the underlying patterns in the data. This can lead to high bias and low variance, resulting in poor accuracy on both the training and test sets. In such cases, increasing the tree depth can improve accuracy by allowing the tree to capture more complex patterns in the data.
2. **Overfitting (Deep Trees)**: On the other hand, when a decision tree is too deep, it may memorize the training data instead of learning general patterns. This can lead to high variance and low bias, resulting in high accuracy on the training set but poor performance on the test set. In this case, reducing the tree depth can improve generalization and hence improve accuracy on unseen data.

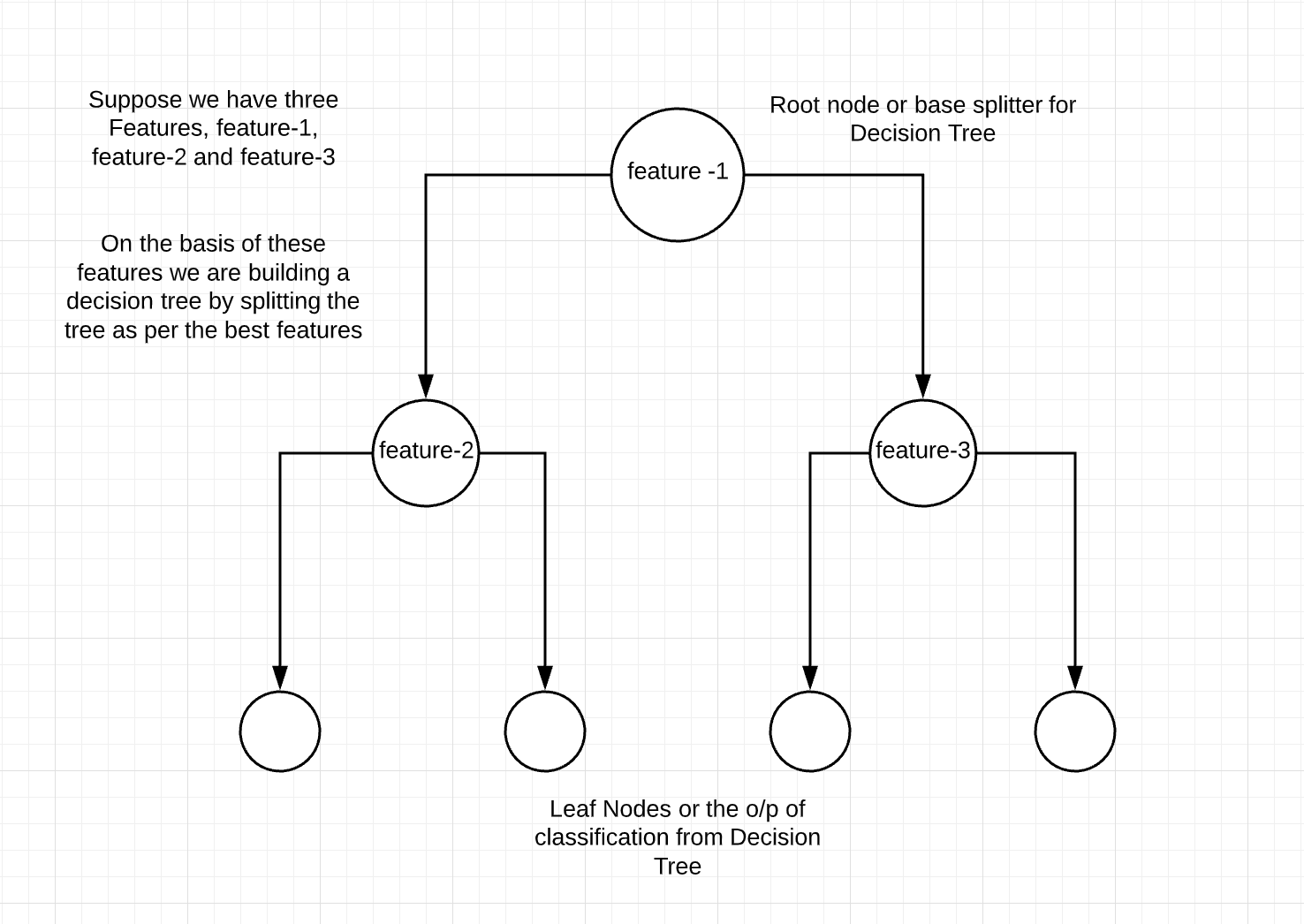
<https://www.geeksforgeeks.org/how-decision-tree-depth-impact-on-the-accuracy/>

**Gini Index**

* The **Gini Index** is the additional approach to dividing a decision tree.
* Purity and impurity in a junction are the primary focus of the Entropy and Information Gain framework.
* The Gini Index, also known as Impurity, calculates the likelihood that somehow a randomly picked instance would be erroneously cataloged.

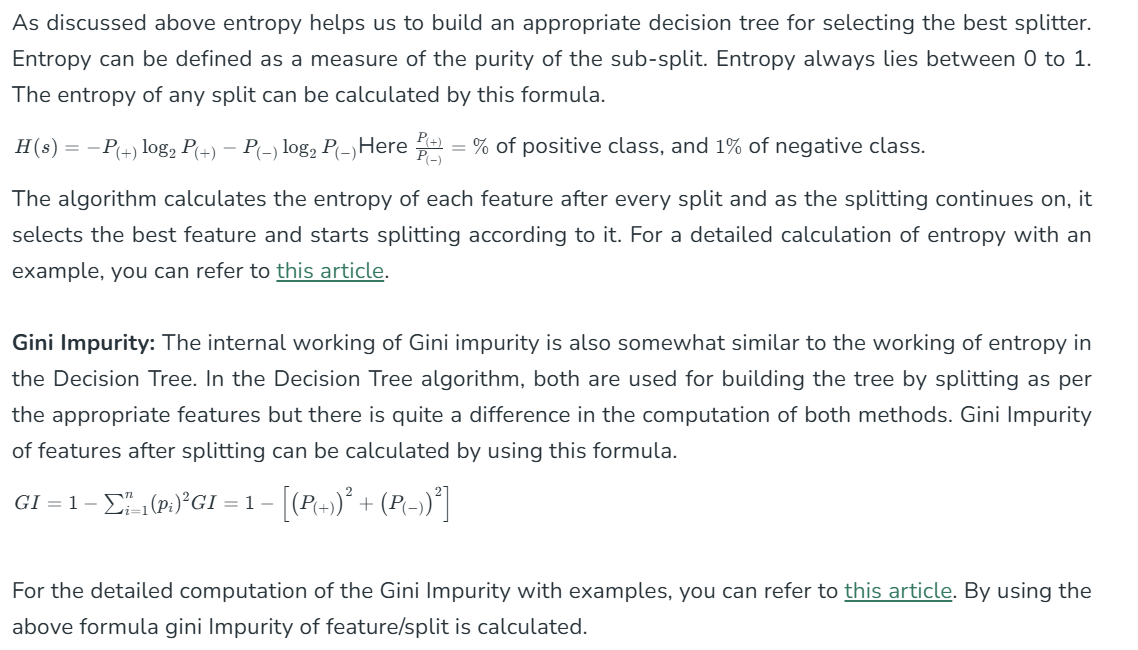
Machine Learning is a Computer Science domain that provides the ability for computers to learn without being explicitly programmed. Machine Learning is one of the most highly demanded technologies that everybody wants to learn and most companies require highly skilled Machine Learning engineers. In this domain, there are various machine learning algorithms developed to solve complex problems with ease. These algorithms are highly automated and self-modifying, as they continue to improve over time with the addition of an increased amount of data and with minimum human intervention required. To learn about top Machine Learning algorithms that every ML engineer should know [click here](https://www.geeksforgeeks.org/top-10-algorithms-every-machine-learning-engineer-should-know/).

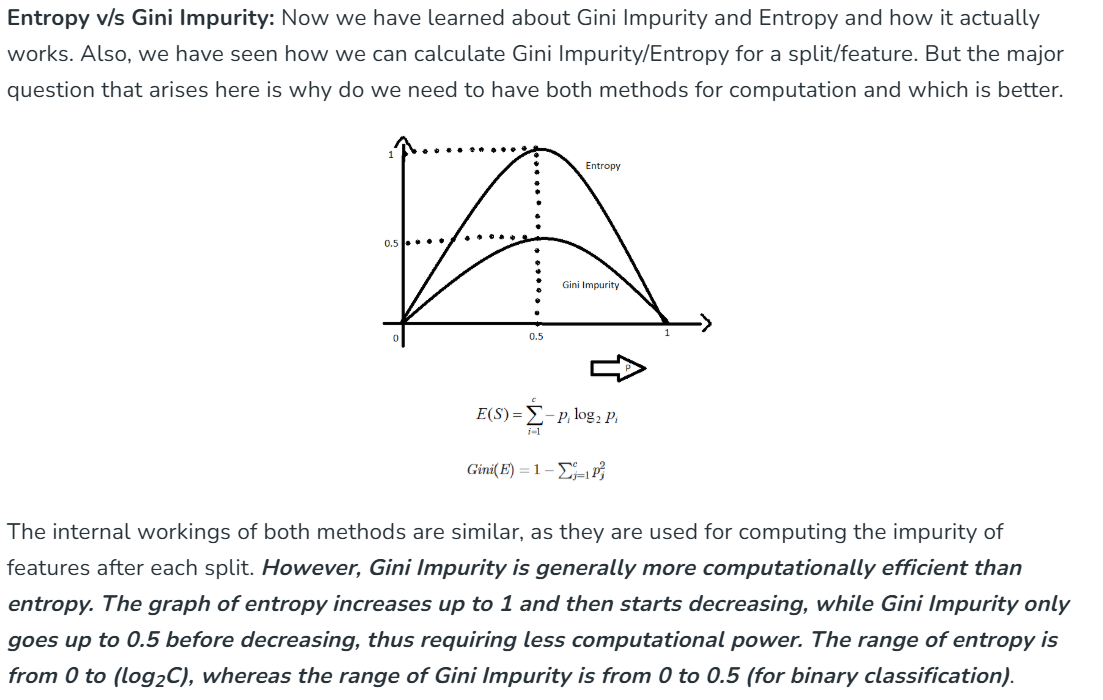
In this article, we will be focusing more on Gini Impurity and Entropy methods in the Decision Tree algorithm and which is better among them.

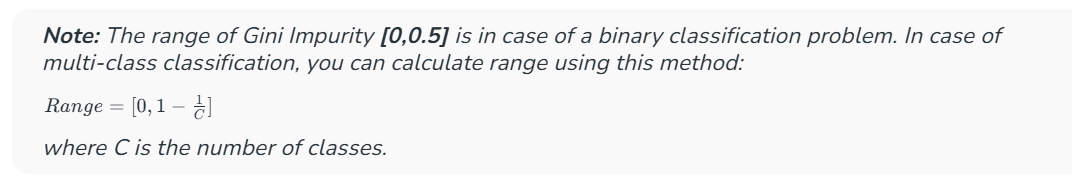


**Decision Tree** is one of the most popular and powerful classification algorithms that we use in machine learning. The decision tree from the name itself signifies that it is used for making decisions from the given dataset. The concept behind the decision tree is that it helps to select appropriate features for splitting the tree into subparts and the algorithm used behind the splitting is ID3. If the decision tree build is appropriate then the depth of the tree will be less or else the depth will be more. To build the decision tree in an efficient way we use the concept of **Entropy**. To learn more about the Decision Tree [click here](https://www.geeksforgeeks.org/decision-tree/). In this article, we will be more focused on the difference between Gini Impurity and Entropy.   
  
**Entropy:**

* The word "entropy," is hails from physics, and refers to an indicator of the disorder. The expected volume of "information," "surprise," or "uncertainty" associated with a randomly chosen variable's potential outcomes is characterized as the entropy of the variable in information theory.
* Entropy is a quantifiable and measurable physical attribute and a scientific notion that is frequently associated with a circumstance of disorder, unpredictability, or uncertainty.
* From classical thermodynamics, where it was originally identified, through the macroscopic portrayal of existence in statistical physics, to the principles of information theory, the terminology, and notion are widely used in a variety of fields of study



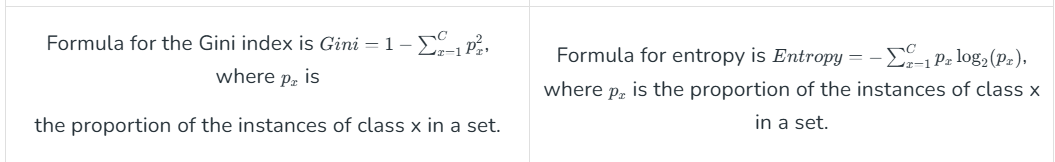




However, the main reason for Gini Impurity's computational advantage is that it does not involve logarithmic functions, which are more computationally intensive. Therefore, Gini Impurity is often considered more efficient compared to entropy for selecting the best features.

**Difference between Gini Index and Entropy**

| It is the probability of misclassifying a randomly chosen element in a set. | While entropy measures the amount of uncertainty or randomness in a set. |
| --- | --- |
| The range of the Gini index is **[0, 0.5]**, where 0 indicates perfect purity and 0.5 indicates maximum impurity. | The range of entropy is **[0, log2(C)]**, where c is the number of classes. The range becomes **[0, 1]**for binary classification. |
| Gini index is a linear measure. | Entropy is a logarithmic measure. |
| It can be interpreted as the expected error rate in a classifier. | It can be interpreted as the average amount of information needed to specify the class of an instance. |
| It is sensitive to the distribution of classes in a set. | It is sensitive to the number of classes. |
| The computational complexity of the Gini index is O(c). | Computational complexity of entropy is O(c \* log(c)). |
| It is less robust than entropy. | It is more robust than Gini index. |
| It is sensitive. | It is comparatively less sensitive. |
| It has a bias toward selecting splits that result in a more balanced distribution of classes. | It has a bias toward selecting splits that result in a higher reduction of uncertainty. |
| Gini index is typically used in CART (Classification and Regression Trees) algorithms | Entropy is typically used in ID3 and C4.5 algorithms |



***Conclusion:*** *It ought to be emphasized that there is no one appropriate approach for evaluating unpredictability or impurities, and that the decision between the Gini index and entropy varies significantly on the particular circumstance and methodology being employed.*

**Implementation of Decision Tree Regression**

For example we want to predict house prices based on factors like size, location and age. A Decision Tree Regressor can split the data based on these features such as checking the location first, then the size and finally the age. This way it can accurately predicts the price by considering the most impactful factors first making it useful and easy to interpret.

**📌 Use Case: Predict California House Prices from AveRooms**

We’ll use the same feature as before for consistency (AveRooms) and compare the result visually.

**🛠️**

**Full Code for California Housing Price Prediction (using AveRooms only)**

python

CopyEdit

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.tree import DecisionTreeRegressor, plot\_tree, export\_text

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

from sklearn.metrics import mean\_squared\_error, r2\_score

# 1. Load California Housing Dataset

housing = fetch\_california\_housing(as\_frame=True)

df = housing.frame

print("✅ Dataset loaded successfully!")

print("Dataset shape:", df.shape)

print("Features:", df.columns.tolist())

# Use only one feature: AveRooms

X = df[["AveRooms"]] # needs to be 2D for sklearn

y = df["MedHouseVal"]

# 2. Split dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print(f"Train size: {len(X\_train)}, Test size: {len(X\_test)}")

# 3. Train Decision Tree Regressor

regressor = DecisionTreeRegressor(max\_depth=4, random\_state=0)

regressor.fit(X\_train, y\_train)

print("\n🌳 Tree trained.")

print("Tree depth:", regressor.get\_depth())

print("Leaves:", regressor.get\_n\_leaves())

# 4. Predict and evaluate

y\_pred = regressor.predict(X\_test)

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

r2 = r2\_score(y\_test, y\_pred)

print("\n🔍 Model Evaluation:")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"R² Score: {r2:.4f}")

# 5. Visualize predictions vs actual

X\_grid = np.linspace(X["AveRooms"].min(), X["AveRooms"].max(), 1000).reshape(-1, 1)

y\_grid\_pred = regressor.predict(X\_grid)

plt.figure(figsize=(10, 6))

plt.scatter(X\_test, y\_test, color='red', label='Actual Prices (Test)')

plt.plot(X\_grid, y\_grid\_pred, color='blue', linewidth=2, label='Decision Tree Prediction')

plt.title("Decision Tree Regression: MedHouseVal vs AveRooms")

plt.xlabel("Average Rooms (AveRooms)")

plt.ylabel("Median House Value (in 100,000s)")

plt.legend()

plt.grid(True)

plt.show()

# 6. Show textual structure of tree

print("\n📄 Tree Structure (Text View):")

tree\_text = export\_text(regressor, feature\_names=["AveRooms"])

print(tree\_text)

# 7. Plot decision tree

plt.figure(figsize=(16, 8))

plot\_tree(regressor, feature\_names=["AveRooms"], filled=True, rounded=True)

plt.title("Decision Tree Structure")

plt.show()

**📊 Analysis of Results**

| **Metric** | **Interpretation** |
| --- | --- |
| **MSE** | Measures the average squared error — lower is better. |
| **R²** | Tells how much variance in target is explained — closer to 1 is better. |

You will likely find that using only **one feature (AveRooms)** doesn’t yield a very high R² — but it's great for learning.

**🧠 Insights**

* **Decision Tree Depth** (max\_depth=4) controls complexity. Try increasing it for better fit (but beware overfitting).
* **AveRooms** alone cannot perfectly predict MedHouseVal — in real scenarios, multiple features should be used.
* You can explore feature importance and try adding more features like AveOccup, HouseAge, etc.

**📊 Comparison: Regression Techniques**

| **Metric** | **Linear Regression** | **SVR** | **Decision Tree** |
| --- | --- | --- | --- |
| Assumes linearity | ✅ Yes | ❌ No | ❌ No |
| Feature scaling needed | ✅ Yes | ✅ Yes | ❌ No |
| Handles non-linearity | ❌ No | ✅ Yes | ✅ Yes |
| Interpretability | ✅ High | ❌ Low | ✅ High |
| Sensitive to outliers | ✅ High | ⚠️ Medium | ⚠️ Medium |
| Training time | ⚡ Fast | 🐢 Slower | ⚡ Fast (small depth) |

**✅ Next Step**

Would you like to continue with:

* 🌲 **4) Random Forest Regressor** (ensemble of decision trees)?
* 🧪 Try **multiple features** instead of one?
* 📁 Load your own dataset?

Let me know your learning path!