Lesson

Training and Testing a Decision Tree Classifier Using the Iris Dataset with Visualization

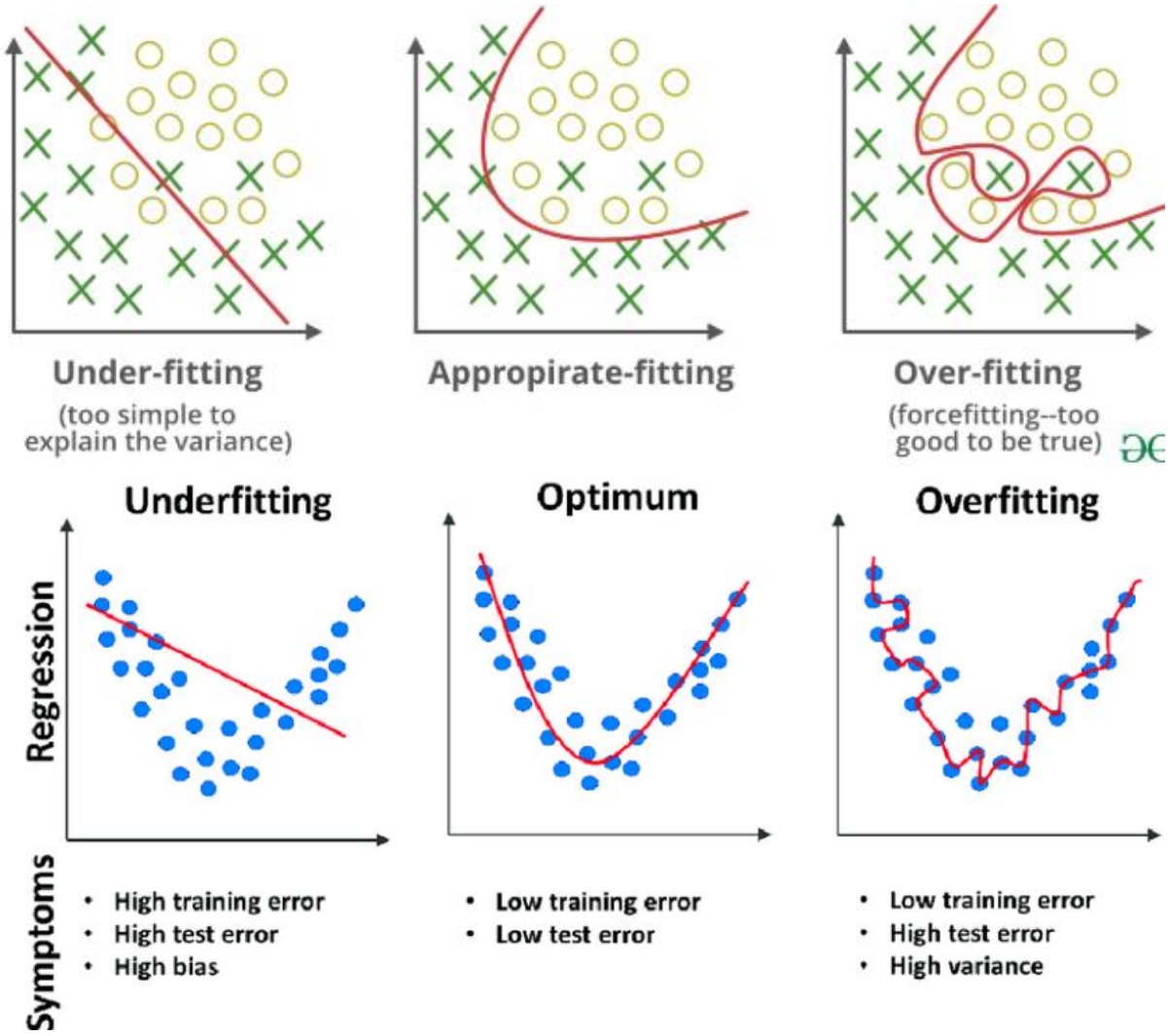
Introduction to Decision Trees

- **What is a Decision Tree?**
  - A decision tree is a flowchart-like structure where:
    - **Internal nodes** represent features (attributes).
    - **Branches** represent decision rules.
    - **Leaf nodes** represent outcomes (class labels).
  - It splits the dataset into subsets based on feature values, aiming to create homogeneous subsets.
- **Advantages**:
  - Easy to understand and interpret.
  - Handles both numerical and categorical data.
  - Requires little data preprocessing.
- **Disadvantages**:
  - Prone to overfitting (can be mitigated with pruning).
  - Sensitive to small changes in the data.

Overfitting and **underfitting** are two common problems in machine learning and data analytics that affect the performance of predictive models. Understanding these concepts is crucial for building models that generalize well to unseen data. Below is a detailed explanation of overfitting and underfitting, along with examples in the context of data analytics.

understand some terms used in ML models.

1. **Noise**: Irrelevant data that negatively impacts the performance of ML models.
2. **Variance:** Occurs when the ML model performs great with the training data but poorly with the test data.
3. **Bias**: The difference between the actual value and the predicted value.



Underfitting:

- **What it is**: The model is **too simple** to learn the patterns in the data. It's like trying to fit a straight line through data that's clearly curved.
- **How it looks**: The model doesn't perform well on both **training** and **test** data.
- **Example**: Imagine you're trying to predict someone's age based only on whether they have gray hair or not. It's too simple and doesn't capture all the factors that influence age.

Signs of underfitting:

- Model makes **poor predictions** on both training and test data.
- The model is too simplistic (e.g., using a straight line for a curve).

2. Overfitting:

- **What it is**: The model is **too complex** and learns not just the real patterns, but also the random noise in the training data. It's like memorizing answers for a test without understanding the material.
- **How it looks**: The model performs **very well** on training data but poorly on new (test) data.

- **Example**: Imagine you're trying to predict someone's age based on a huge list of features (like height, weight, etc.), but you also start including irrelevant data like their favorite color or pet's name. The model gets too "caught up" in the training data and struggles to generalize to new situations.

Signs of overfitting:

- Model performs **really well** on training data but **poorly** on test data.
- The model is too complex, with many unnecessary features or too many parameters.

Simple Comparison:

Factor	Underfitting	Overfitting
Model Complexity	Too simple (e.g., linear model for complex data)	Too complex (e.g., a very detailed model for simple data)
Training Performance	Poor	Excellent
Test Performance	Poor	Poor
Example	Trying to fit a straight line through curvy data	Memorizing the training data, not learning general patterns

---

1. Overfitting

Definition

- Overfitting occurs when a model learns the training data too well, capturing noise and irrelevant details instead of the underlying patterns.
- As a result, the model performs exceptionally well on the training data but poorly on unseen data (test data).

Causes

- The model is too complex (e.g., too many parameters or features).
- The training dataset is too small or noisy.

Symptoms

- High accuracy on the training set but low accuracy on the test set.
- The model captures random fluctuations or outliers in the training data.

Example in Data Analytics

- **Scenario**: Predicting house prices based on features like size, location, and number of rooms.
- **Overfitting**: A model that considers irrelevant features (e.g., the color of the house) or fits the training data too closely (e.g., memorizing specific house prices instead of learning general trends).
- **Result**: The model predicts house prices accurately for the training data but fails to generalize to new data.

---

2. Underfitting

Definition

- Underfitting occurs when a model is too simple to capture the underlying patterns in the data.
- As a result, the model performs poorly on both the training data and unseen data.

Causes

- The model is too simple (e.g., a linear model for a nonlinear problem).
- Insufficient training or inadequate features.

Symptoms

- Low accuracy on both the training and test sets.
- The model fails to capture important trends in the data.

Example in Data Analytics

- **Scenario**: Predicting customer churn based on features like usage patterns and demographics.
- **Underfitting**: A model that uses only one feature (e.g., age) to predict churn, ignoring other important factors like usage frequency or customer support interactions.
- **Result**: The model performs poorly on both the training and test data because it fails to capture the complexity of the problem.

---

3. Visualizing Overfitting and Underfitting

Example: Polynomial Regression

- **Data**: A dataset with a nonlinear relationship between X and Y.
- **Models**:
    - **Underfitting**: A linear model (degree = 1) that fails to capture the curvature of the data.
    - **Good Fit**: A polynomial model (degree = 3) that captures the underlying trend.

- **Overfitting**: A high-degree polynomial model (degree = 10) that fits the training data perfectly but fails to generalize.

1. Precision

- **Definition**: Precision measures the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives + false positives).
- **Formula**:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

- **Interpretation**:
  - High precision means the model is good at avoiding false positives.
  - Useful in scenarios where false positives are costly (e.g., spam detection).

---

2. Recall (Sensitivity)

- **Definition**: Recall measures the proportion of correctly predicted positive instances (true positives) out of all actual positive instances (true positives + false negatives).
- **Formula**:

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

- **Interpretation**:
  - High recall means the model is good at identifying all positive instances.
  - Useful in scenarios where false negatives are costly (e.g., disease detection).

---

3. F1-Score

- **Definition**: The F1-score is the harmonic mean of precision and recall. It balances the two metrics, providing a single score that reflects both precision and recall.
- **Formula**:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Interpretation**:
  - High F1-score indicates a good balance between precision and recall.
  - Useful when you want to optimize both false positives and false negatives.

3. F1-Score

- **Definition**: The F1-score is the harmonic mean of precision and recall. It balances the two metrics, providing a single score that reflects both precision and recall.
- **Formula**:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Interpretation**:
  - High F1-score indicates a good balance between precision and recall.
  - Useful when you want to optimize both false positives and false negatives.

---

4. Support

- **Definition**: Support is the number of actual occurrences of each class in the dataset.
- **Interpretation**:
  - Indicates how many samples belong to each class.
  - Helps identify class imbalance (e.g., one class has significantly more samples than another).

---

5. Accuracy

- **Definition**: Accuracy measures the proportion of correctly predicted instances (both true positives and true negatives) out of all instances.
- **Formula**:

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{True Positives (TP)} + \text{True Negatives (TN)} + \text{False Positives (FP)} + \text{Fals}}$$

- **Precision** is concerned with the **accuracy of positive predictions**, focusing on minimizing false positives.
- **Recall** is concerned with **finding all positive instances**, aiming to minimize false negatives.
- **Precision**: How many of the predicted positives were correct?
  Recall: How many of the actual positives were identified?

F1 Score

- The **F1 Score** is used to **combine Precision and Recall** into one metric, balancing the trade-off between the two.

- It's particularly useful in cases where you care about both false positives and false negatives, and in situations with **imbalanced classes**.
- It helps you avoid situations where a model might seem "good" in accuracy but is actually performing poorly on the minority class.

When to Use Precision, Recall, and F1-Score

- **Precision:** Use when false positives are costly (e.g., spam detection).
- **Recall:** Use when false negatives are costly (e.g., disease detection).
- **F1-Score:** Use when you want a balance between precision and recall (e.g., imbalanced datasets).

True Positives (TP)

- **Definition**: True positives are cases where the model **correctly predicts the positive class**.
  - **Example**:
  - **Scenario**: A medical test predicts whether a patient has a disease.
  - **True Positive**: The test correctly identifies a patient as having the disease when they actually do.

False Positives (FP)

- **Definition**: False positives are cases where the model **incorrectly predicts the positive class** (i.e., predicts positive when the actual class is negative).
  - **Example**:
  - **Scenario**: A medical test predicts whether a patient has a disease.
  - **False Positive**: The test incorrectly identifies a healthy patient as having the disease.

Visualization in a Confusion Matrix

	Actual Positive	**Actual Negative**
Predicted Positive	True Positives (TP)	False Positives (FP)
Predicted Negative	False Negatives (FN)	True Negatives (TN)

Summary

- **Precision**: Measures how many predicted positives are actually positive.

- **Recall:** Measures how many actual positives are correctly predicted.
- **F1-Score:** Balances precision and recall.
- **Support:** Indicates the number of instances for each class.
- **Accuracy:** Measures overall correctness but can be misleading in imbalanced datasets.

---

2. The Iris Dataset

- **Description**:
  - The Iris dataset contains 150 samples of iris flowers, with 50 samples from each of three species:
    - Setosa
    - Versicolor
    - Virginica
  - Each sample has four features:
    - Sepal length
    - Sepal width
    - Petal length
    - Petal width
- **Objective**:
  - Train a decision tree classifier to predict the species of iris flowers based on their features.

---

3. Steps to Train and Test a Decision Tree Classifier

Step 1: Load the Dataset

- Use the sklearn.datasets module to load the Iris dataset.
- Split the dataset into **training** and **testing** sets.

python

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
```

```
y = iris.target  # Labels

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

Step 2: Train the Decision Tree Classifier

- Use the DecisionTreeClassifier class from sklearn.tree to train the model.

python

```
from sklearn.tree import DecisionTreeClassifier

# Initialize the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
clf.fit(X_train, y_train)
```

Step 3: Test the Decision Tree Classifier

- Use the trained model to make predictions on the test set.
- Evaluate the model's performance using metrics like **accuracy**.

python

```
from sklearn.metrics import accuracy_score

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Step 4: Visualize the Decision Tree

- Use the plot_tree function from sklearn.tree to visualize the decision tree.

python

```
import matplotlib.pyplot as plt
```

```
from sklearn.tree import plot_tree

# Plot the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names)
plt.title("Decision Tree for Iris Dataset")
plt.show()
```

---

4. Key Concepts in Decision Tree Visualization

- **Nodes:**
  - **Root Node:** The topmost node representing the entire dataset.
  - **Internal Nodes:** Nodes that split the data based on feature values.
  - **Leaf Nodes:** Nodes that represent the final decision (class label).
- **Splitting Criteria:**
  - The decision tree uses metrics like **Gini impurity** or **entropy** to decide how to split the data.
- **Feature Importance:**
  - The decision tree assigns importance to features based on how well they split the data.

---

5. Example Output

Accuracy:

```
Accuracy: 95.56%
```

Decision Tree Visualization:

- The tree will show:
  - Feature names (e.g., petal length, sepal width).
  - Decision rules (e.g., petal length <= 2.45).
  - Class labels (e.g., setosa, versicolor, virginica).

---

6. Practical Tips

- **Hyperparameter Tuning:**
  - Use GridSearchCV or RandomizedSearchCV to find the best hyperparameters (e.g., max_depth, min_samples_split).
- **Overfitting:**

- Limit the depth of the tree (max_depth) or use pruning techniques to avoid overfitting.

- **Feature Importance:**
  - Use clf.feature_importances_ to check which features are most important for making predictions.

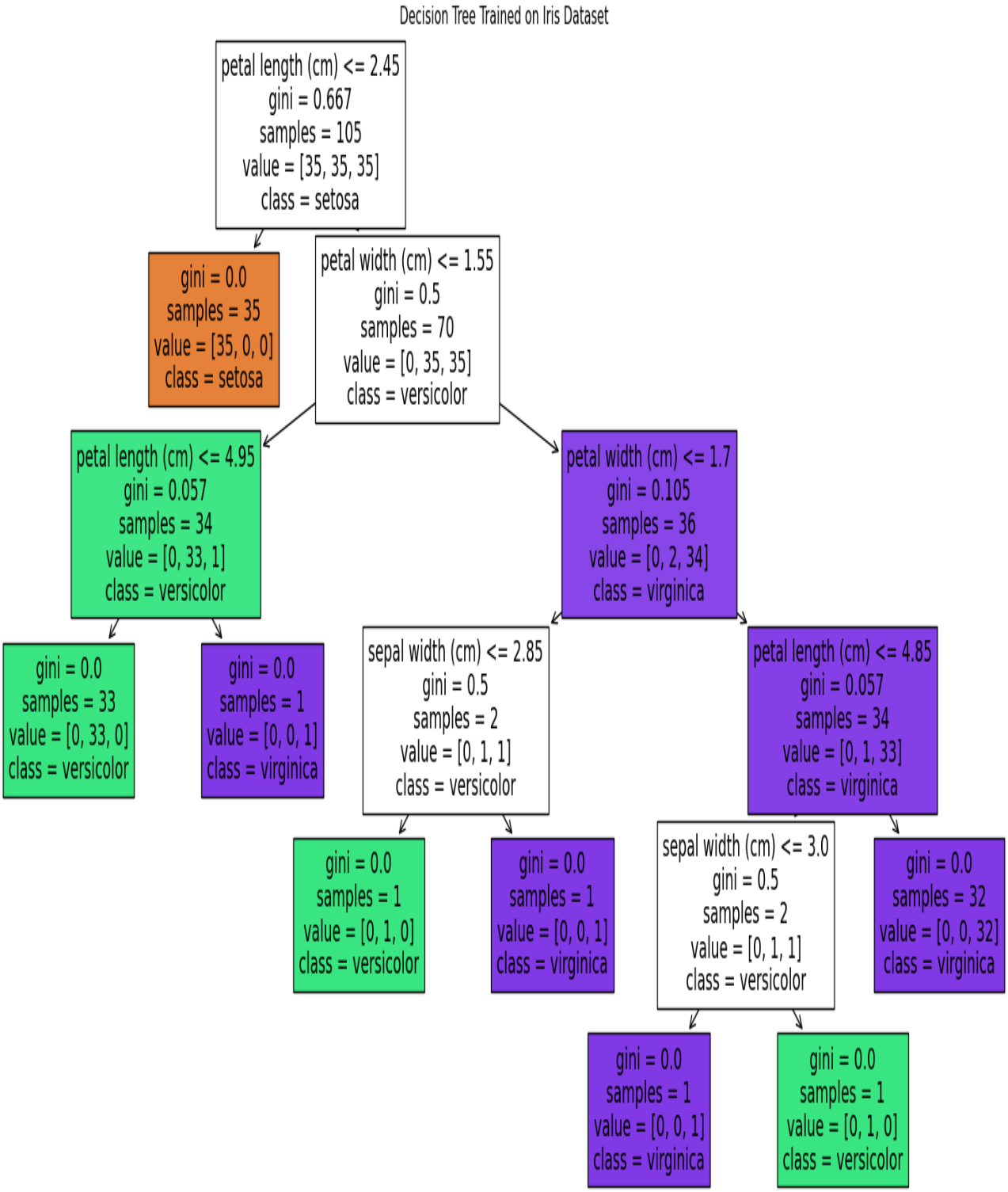
Training and Testing a Decision Tree Classifier Using the Iris Dataset with Visualization: Demo

---

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
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,
classification_report

# Load the iris dataset
iris = load_iris()
X = iris.data  # Features (sepal length, sepal width, petal
length, petal width)
y = iris.target  # Target labels (species of iris)

# Create a DataFrame for better visualization (optional)
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Split the dataset into training and testing sets (70/30
split)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y  #
Stratify to maintain class distribution
)

# Initialize and train the decision tree classifier
```

```
clf = DecisionTreeClassifier(random_state=42)  # Set a random
state for reproducibility
clf.fit(X_train, y_train)  # Fit the model on the training
data

# Make predictions on the test set
y_pred = clf.predict(X_test)  # Predict the labels for the
test set

# Calculate the accuracy
acc = accuracy_score(y_test, y_pred)  # Calculate the accuracy
score
print("Accuracy:", acc)  # Print the accuracy

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))  # Print detailed
classification metrics

# Plot the decision tree
plt.figure(figsize=(20,10))  # Set the figure size for the
plot
plot_tree(clf, feature_names=iris.feature_names,
class_names=iris.target_names, filled=True)  # Plot the
decision tree
plt.title("Decision Tree Trained on Iris Dataset")  # Title
for the plot
plt.show()  # Display the plot

print("Training and testing completed successfully!")  #
Confirmation message
Accuracy: 0.9333333333333333
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      0.80      0.89        15
           2       0.83      1.00      0.91        15

    accuracy                           0.93        45
   macro avg       0.94      0.93      0.93        45
```

```
weighted avg       0.94      0.93      0.93         45
```




```
Training and testing completed successfully!
```

Reference materials

Understanding Decision Trees using Python (scikit-learn)

https://www.youtube.com/watch?v=yi7KsXtaOCo&t=940s

How Do Decision Trees Work (Simple Explanation) - Learning and Training Process
https://www.youtube.com/watch?v=xDWZzD4TPO0

Machine Learning Tutorial Python - 9 Decision Tree
https://www.youtube.com/watch?v=PHxYNGo8NcI