Slide 1: Multi-Level Classification

Multi-level classification, also known as hierarchical classification, is a machine learning approach where labels are organized in a hierarchical structure, often resembling a tree or a directed acyclic graph (DAG). In contrast to flat classification, where classes are independent of each other, multi-level classification captures relationships between labels, such as parent-child or sibling relationships, allowing for more complex decision-making processes.

The goal of multi-level classification is to predict labels at multiple levels of the hierarchy. For example, in an e-commerce product categorization system, a product like a smartphone may belong to categories like "Electronics" (parent) and "Mobile Phones" (child).

Types of Multi-Level Classification

1. Local Classifiers per Parent Node: A separate classifier is trained for each parent node. For example, once a parent category like "Electronics" is predicted, a new classifier will determine which child class (like "Mobile Phones" or "Laptops") to choose.
2. Local Classifiers per Level: A classifier is trained for each level of the hierarchy. For example, the first classifier will determine the high-level category (e.g., "Electronics" or "Home Appliances"), and subsequent classifiers predict more granular categories (like "Mobile Phones" or "Washing Machines").
3. Global Classifier: A single model is used to predict the entire path from the root to the leaf node in the hierarchy.

Please give pictorial representation as well

Slide 2: Decision Trees Algorithm

A Decision Tree is a tree-like structure where each internal node represents a decision on a feature, each branch represents the outcome of the decision, and each leaf node represents a class label. The model recursively splits the data based on the features that provide the highest information gain (or lowest Gini impurity).

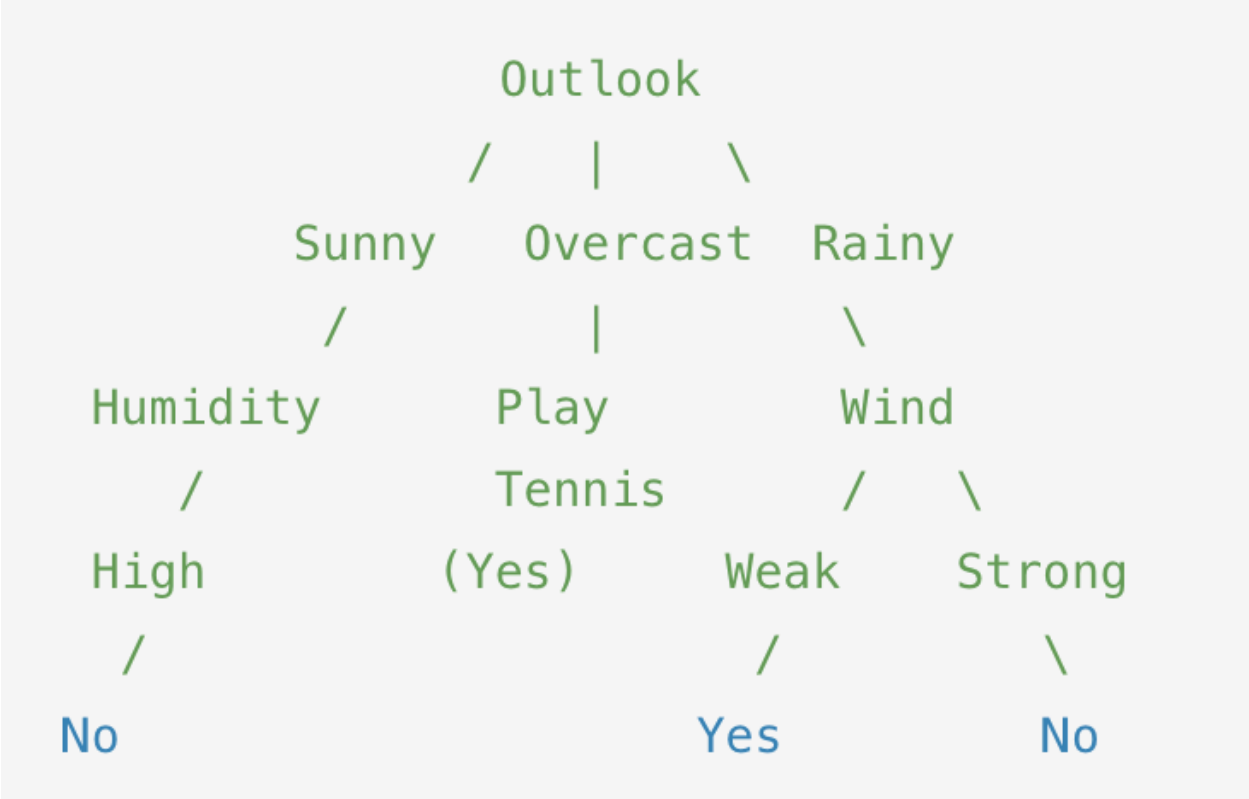
simple representation of a Decision Tree for the "Play Tennis" example based on the features: Outlook, Temperature, Humidity, and Wind.

Dataset

Outlook	Temperature	Humidity	Wind	Play Tennis?
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes

Decision Tree Representation

For simplicity, let’s assume that Outlook is the feature with the highest information gain for our first split. The tree could look something like this:



Slide 3: Gini Impurity

Before Split:

Probability of "Yes" $p_{\text{Yes}} = \frac{3}{5} = 0.6$
Probability of "No" $p_{\text{No}} = \frac{2}{5} = 0.4$

Using the Gini formula:

$$\text{Gini Impurity (before split)} = 1 - (p_{\text{Yes}}^2 + p_{\text{No}}^2)$$

$$= 1 - (0.6^2 + 0.4^2) = 1 - (0.36 + 0.16) = 1 - 0.52 = 0.48$$

After Split:

Split Details

1. **Outlook = Sunny**: 2 samples (both are "No").
2. **Outlook = Overcast**: 1 sample (label is "Yes").
3. **Outlook = Rainy**: 2 samples (both are "Yes").

Let's calculate the Gini Impurity for each child node:

- **Node 1 (Sunny)**:
  - Both samples are "No".
  - Gini Impurity for this node:
$$= 1 - (1^2 + 0^2) = 1 - 1 = 0$$
- **Node 2 (Overcast)**:
  - The sample is "Yes".
  - Gini Impurity for this node:
$$= 1 - (1^2 + 0^2) = 1 - 1 = 0$$
- **Node 3 (Rainy)**:
  - Both samples are "Yes".
  - Gini Impurity for this node:
$$= 1 - (1^2 + 0^2) = 1 - 1 = 0$$

1. **Sunny**: $\frac{2}{5} \times 0 = 0$
2. **Overcast**: $\frac{1}{5} \times 0 = 0$
3. **Rainy**: $\frac{2}{5} \times 0 = 0$

The total Gini Impurity after the split:

$$\text{Gini Impurity (after split)} = 0 + 0 + 0 = 0$$

Slide 4: Entropy

Before split:

The formula for entropy is:

$$\text{Entropy} = -\sum_{i=1}^{K} p_i \log_2(p_i)$$

where p_i is the probability of each class i.

Probability of "Yes" $p_{\text{Yes}} = \frac{3}{5} = 0.6$
Probability of "No" $p_{\text{No}} = \frac{2}{5} = 0.4$

Now we substitute these values into the formula:

$$\text{Entropy (before split)} = -(0.6 \cdot \log_2(0.6) + 0.4 \cdot \log_2(0.4))$$

Calculating each term:

1. $0.6 \cdot \log_2(0.6) \approx -0.442$
2. $0.4 \cdot \log_2(0.4) \approx -0.528$

So,

$$\text{Entropy (before split)} = 0.442 + 0.528 = 0.970$$

After split:

Split Details

1. **Outlook = Sunny**: 2 samples, both labeled "No."
2. **Outlook = Overcast**: 1 sample, labeled "Yes."
3. **Outlook = Rainy**: 2 samples, both labeled "Yes."

Let's calculate the Entropy for each of these child nodes.

- **Node 1 (Sunny)**:
  - Both samples are "No" (pure node).
  - Entropy:
$$\text{Entropy} = -(1 \cdot \log_2(1) + 0 \cdot \log_2(0)) = 0$$
- **Node 2 (Overcast)**:
  - The single sample is "Yes" (pure node).
  - Entropy:
$$\text{Entropy} = -(1 \cdot \log_2(1) + 0 \cdot \log_2(0)) = 0$$
- **Node 3 (Rainy)**:
  - Both samples are "Yes" (pure node).
  - Entropy:
$$\text{Entropy} = -(1 \cdot \log_2(1) + 0 \cdot \log_2(0)) = 0$$

1. **Sunny**: $\frac{2}{5} \times 0 = 0$
2. **Overcast**: $\frac{1}{5} \times 0 = 0$
3. **Rainy**: $\frac{2}{5} \times 0 = 0$

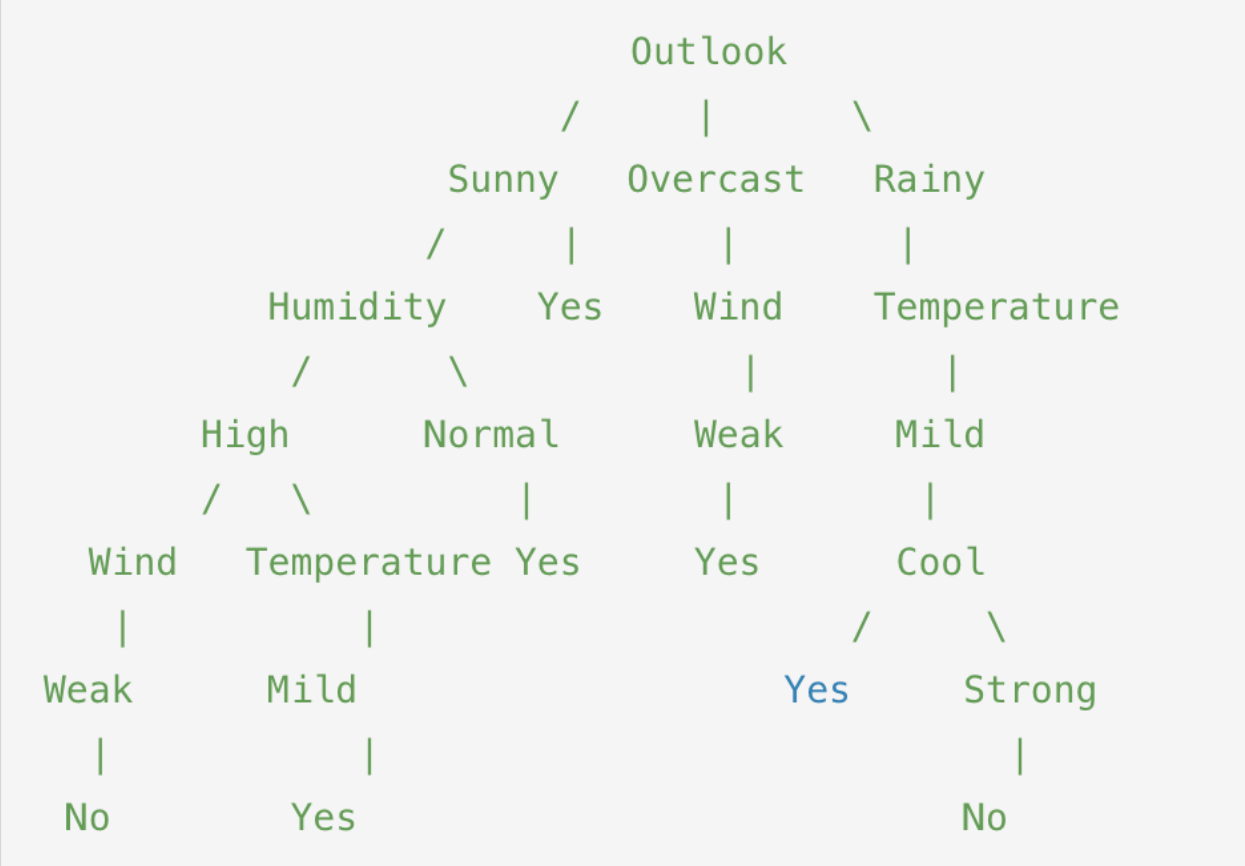
The total Entropy after the split is:

$$\text{Entropy (after split)} = 0 + 0 + 0 = 0$$

Slide 5: Overfitting

Overfitting in a Decision Tree occurs when the tree learns not just the general patterns in the data but also the noise or outliers, leading to a model that performs very well on

the training data but poorly on new, unseen data. Overfitting typically happens when the tree grows too deep, capturing every detail of the training data.



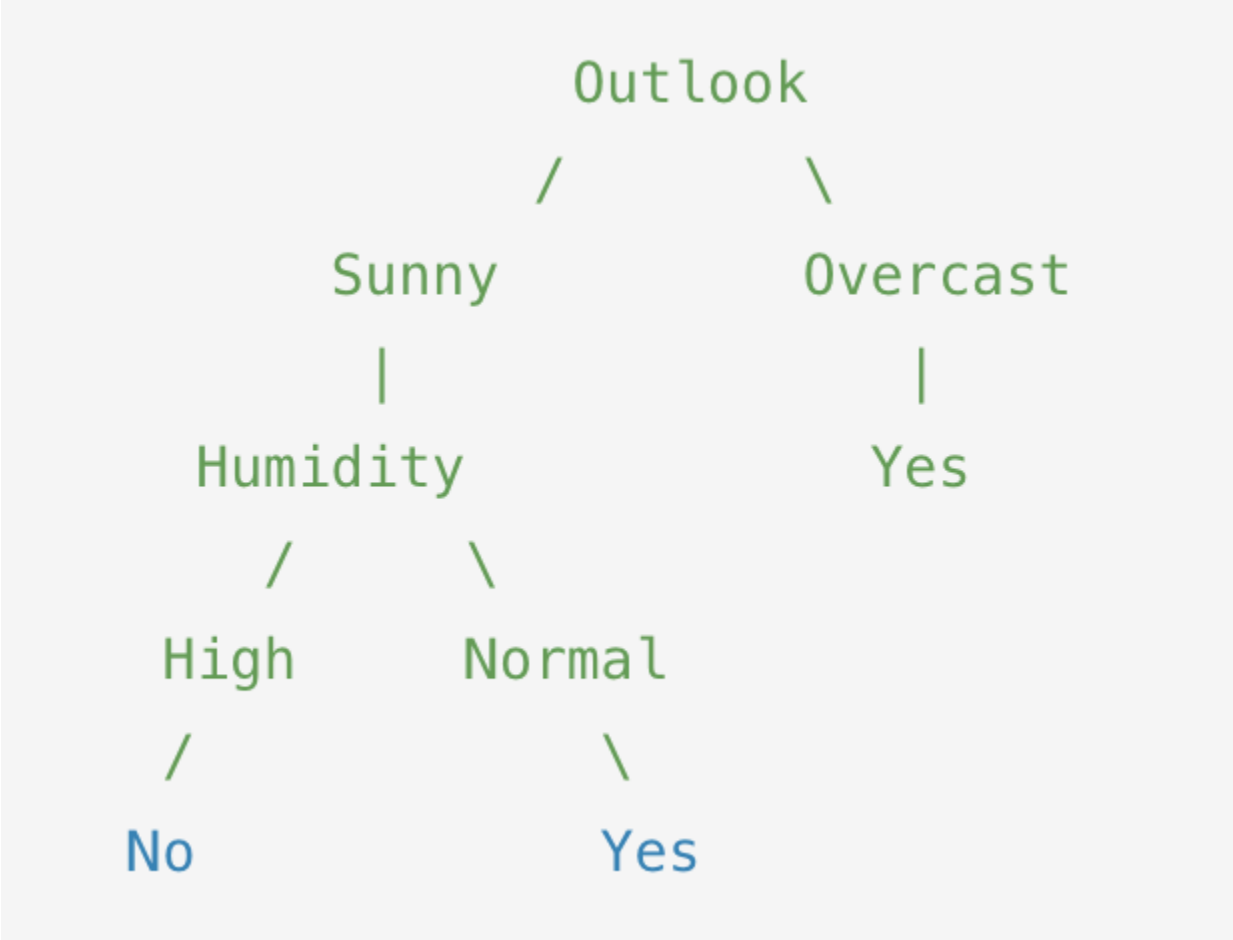
- Low Generalization: With this structure, the tree is very specific to this dataset. If we test it on new data, such as another combination of "Sunny" and "Humidity = High" but with different wind conditions, the model might misclassify due to the overly complex structure.

In practice, pruning (removing unnecessary branches) or limiting tree depth can help avoid overfitting, making the model simpler and better at generalizing to new data.

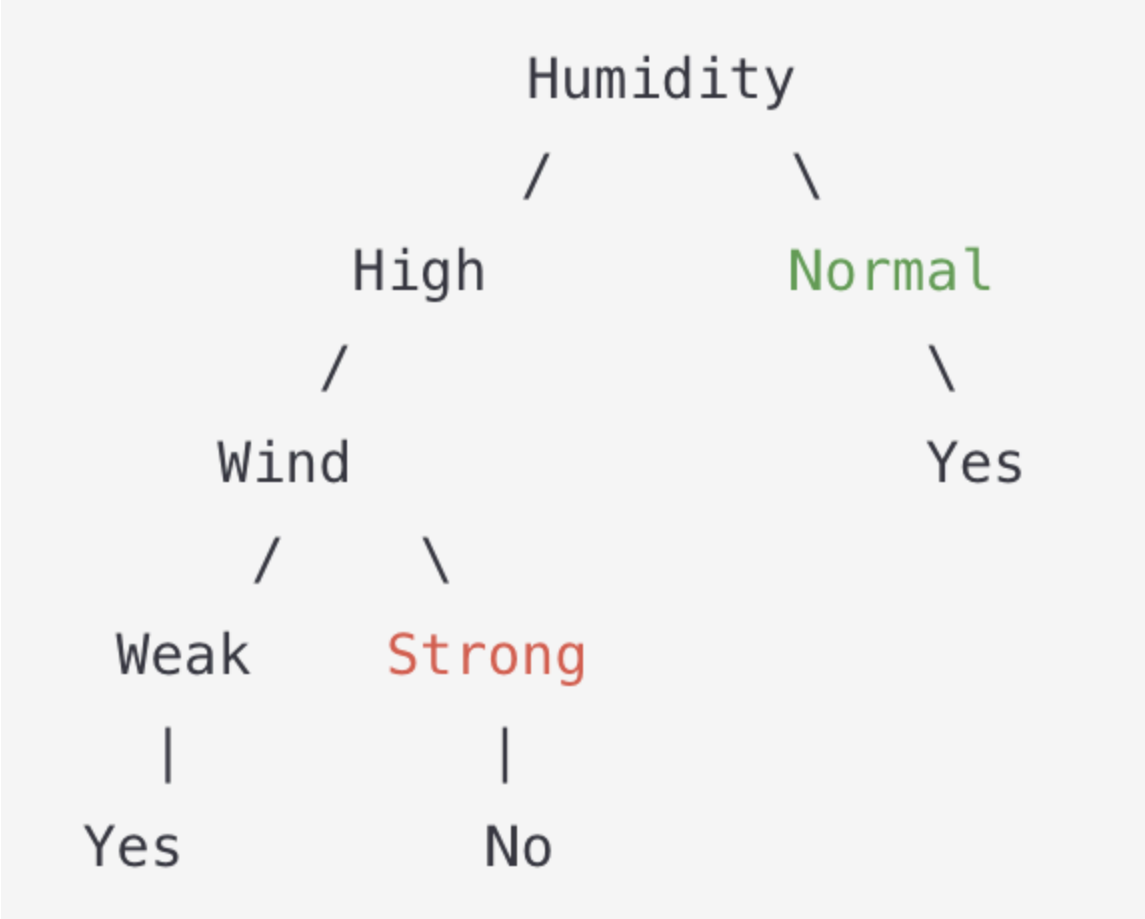
Slide 6: Random Forest Ensemble Averging

Random Forests reduce the likelihood of overfitting by creating an ensemble of multiple Decision Trees, each trained on different random subsets of the data and features. This randomness introduces diversity among the trees, so they capture general patterns rather than specific details or noise in the training data.

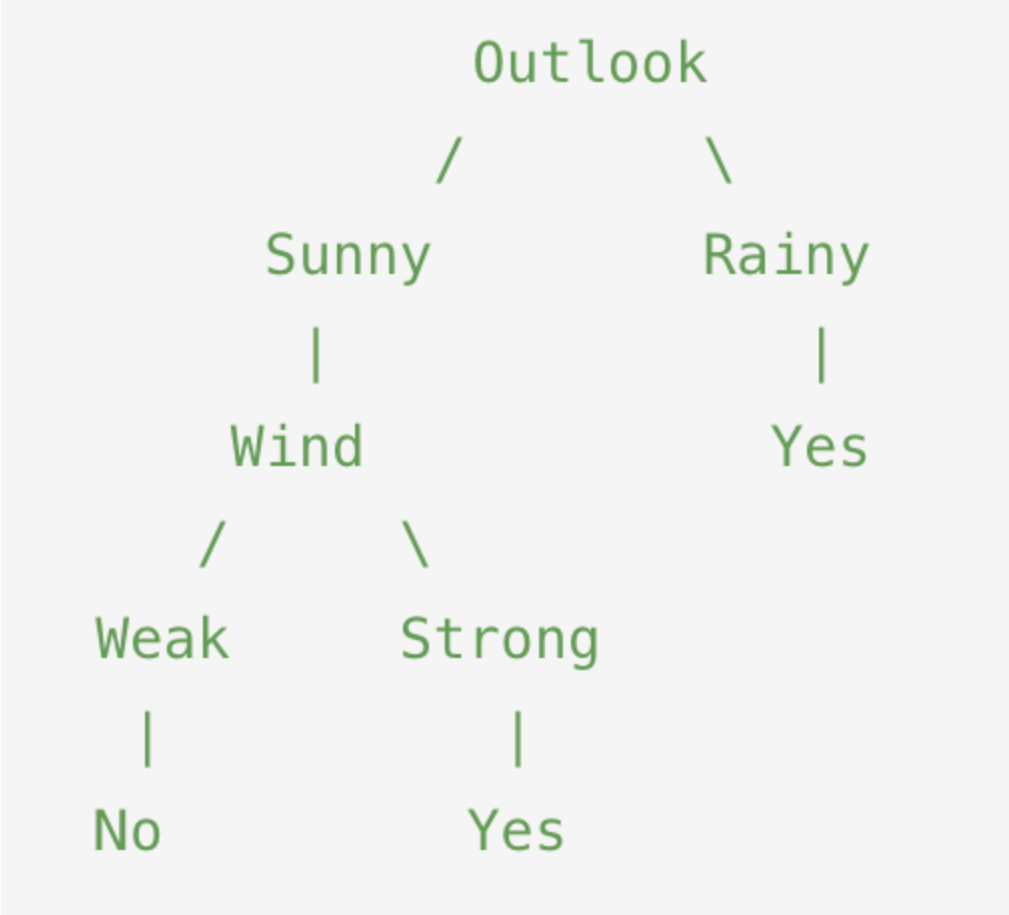
Tree 1:



Tree 2:



Tree 3:



Example Scenario:
To make a prediction for new data, say, `Outlook = Sunny`, `Humidity = High`, `Wind = Weak`:

1. Tree 1: Predicts No based on the path `Outlook = Sunny` → `Humidity = High`.
2. Tree 2: Predicts Yes based on the path `Humidity = High` → `Wind = Weak`.
3. Tree 3: Predicts No based on the path `Outlook = Sunny` → `Wind = Weak`.

With majority voting, the final prediction for this instance is No (since two out of three trees predict "No").

Slide 7: Evaluation Metrics

Accuracy

- **Definition**: The proportion of correctly classified instances out of the total instances.
- **Formula**:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Confusion Matrix:

Actual / Predicted	Fraud (Positive)	Legitimate (Negative)
Fraud	TP = 70	FN = 30
Legitimate	FP = 20	TN = 880

Step 1: Calculate Precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{70}{70 + 20} = \frac{70}{90} \approx 0.778$$

Precision of **0.778** means that about **77.8% of transactions predicted as fraud were actually fraud**.

Step 2: Calculate Recall

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{70}{70 + 30} = \frac{70}{100} = 0.7$$

Recall of **0.7** means that the model **correctly identified 70% of the actual fraudulent transactions**.

Step 3: Calculate F1 Score

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.778 \times 0.7}{0.778 + 0.7}$$

Calculating the numerator:

$$0.778 \times 0.7 = 0.5446$$

Calculating the denominator:

$$0.778 + 0.7 = 1.478$$

$$\text{F1 Score} = 2 \times \frac{0.5446}{1.478} \approx 2 \times 0.3685 = 0.737$$

Precision is important in situations where false positives are costly.
Recall is critical in situations where missing positives is costly.
F1 Score balances Precision and Recall, making it suitable when both false positives and false negatives carry significant costs.

Slide 8: Decision Tree

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
# Sample dataset
data = {
    'weight': [1.5, 2.0, 0.5, 0.3, 5.0, 0.7, 2.2],
    'dimensions': [15, 20, 5, 3, 50, 7, 25],
    'category': ['Electronics', 'Electronics', 'Accessories', 'Accessories', 'Furniture',
'Accessories', 'Electronics']
}

# Create DataFrame
df = pd.DataFrame(data)

# Feature matrix and target variable
X = df[['weight', 'dimensions']]
y = df['category']

# Split data 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)

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(max_depth=3, random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Slide 9: Random Forest

```
# Import necessary libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

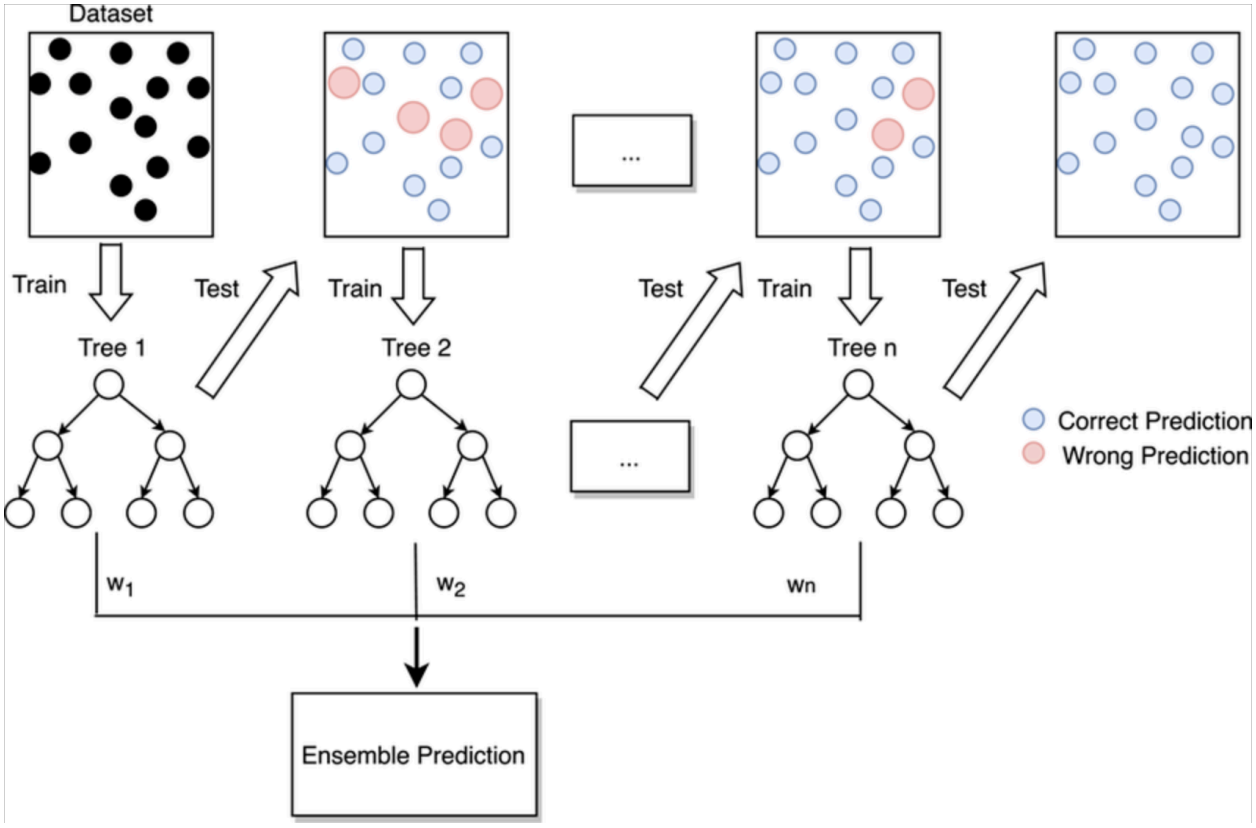
```
# Sample text dataset
text_data = {
    'review': [
        'This product is amazing!',
        'Terrible, not worth the money.',
        'Very satisfied with the quality.',
        'I would not recommend this product.',
        'Exceeded my expectations!',
        'Waste of money, very disappointing.',
        'I love it, great quality and value.'
    ],
    'sentiment': ['Positive', 'Negative', 'Positive', 'Negative', 'Positive', 'Negative', 'Positive']
}

# Create DataFrame
df = pd.DataFrame(text_data)

# Features and target
X = df['review']
y = df['sentiment']

# Convert text data to TF-IDF features
tfidf = TfidfVectorizer(max_features=100)
X_tfidf = tfidf.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.3,
random_state=42)

# Initialize the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf.fit(X_train, y_train)

# Make predictions
y_pred = rf.predict(X_test)

# Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Formula for Cross-Entropy

For a true distribution P and a predicted distribution Q, cross-entropy $H(P, Q)$ is calculated as:

$$H(P, Q) = -\sum_{c} p(c) \log_2(q(c))$$



Aspect	Bagging (Bootstrap Aggregating)	Boosting
Objective	Reduce variance by averaging multiple models trained on different subsets of the data.	Reduce bias by sequentially training models, each focusing on correcting errors of its predecessor.
Training Process	Trains base models independently and in parallel on random subsets of the data.	Trains base models sequentially, with each new model emphasizing the misclassified instances from previous ones.
Data Sampling	Utilizes bootstrapping: each model is trained on a random subset of the data sampled with replacement.	Uses the entire dataset for each model but adjusts the weights of instances based on previous classification errors.
Model Combination	Aggregates predictions by averaging (for regression) or majority voting (for classification).	Combines predictions through a weighted sum, giving more importance to models with better performance.
Overfitting Tendency	Less prone to overfitting due to the averaging of multiple models.	More susceptible to overfitting, especially with noisy data, as it focuses on hard-to-classify instances.
Parallelization	Easily parallelizable since models are trained independently.	Challenging to parallelize due to its sequential training nature.
Common Algorithms	Random Forests, Bagged Decision Trees.	AdaBoost, Gradient Boosting Machines (GBM), XGBoost.

Notebook:

pip install pandas numpy scikit-learn matplotlib

To create a Jupyter Notebook that trains and evaluates Decision Tree, Random Forest, and Support Vector Machine (SVM) classifiers on the Drug Classification dataset, follow these steps:

Import Necessary Libraries: Ensure all required libraries are imported:
python
Copy code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

1.

Load and Preprocess the Dataset: Load the dataset and preprocess it as previously described:
python
Copy code

```
# Load the dataset
df = pd.read_csv('drug200.csv')

# One-hot encode categorical variables
df = pd.get_dummies(df, columns=['Sex', 'BP', 'Cholesterol'],
drop_first=True)

# Define features and target
X = df.drop('Drug', axis=1)
y = df['Drug']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=1)
```

2.
3. **Train and Evaluate Models**: Train each classifier and evaluate their performance:

Decision Tree Classifier:
python
Copy code

```
# Initialize and train the Decision Tree classifier
dt_clf = DecisionTreeClassifier(random_state=1)
dt_clf.fit(X_train, y_train)
```

```
# Predict and evaluate
dt_pred = dt_clf.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_pred)
print(f"Decision Tree Accuracy: {dt_accuracy:.2f}")
```

-

Random Forest Classifier:
python
Copy code

```
# Initialize and train the Random Forest classifier
rf_clf = RandomForestClassifier(random_state=1)
rf_clf.fit(X_train, y_train)

# Predict and evaluate
rf_pred = rf_clf.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)
print(f"Random Forest Accuracy: {rf_accuracy:.2f}")
```

-

Support Vector Machine (SVM) Classifier:
python
Copy code

```
# Initialize and train the SVM classifier
svm_clf = SVC(random_state=1)
svm_clf.fit(X_train, y_train)

# Predict and evaluate
svm_pred = svm_clf.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_pred)
print(f"SVM Accuracy: {svm_accuracy:.2f}")
```

-

Compare Model Accuracies: Compile and display the accuracies of all models:
python

Copy code

```
# Create a DataFrame to compare accuracies
accuracy_df = pd.DataFrame({
    'Model': ['Decision Tree', 'Random Forest', 'SVM'],
    'Accuracy': [dt_accuracy, rf_accuracy, svm_accuracy]
})

print(accuracy_df)
```

4.

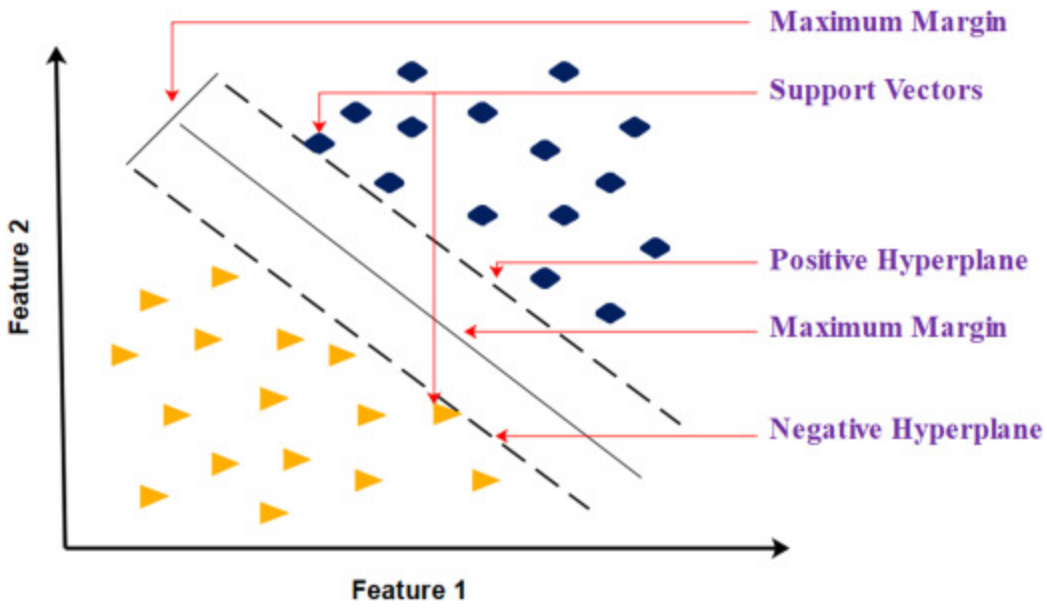
Visualize Confusion Matrices: Plot confusion matrices for each model to assess their performance in detail:
python
Copy code

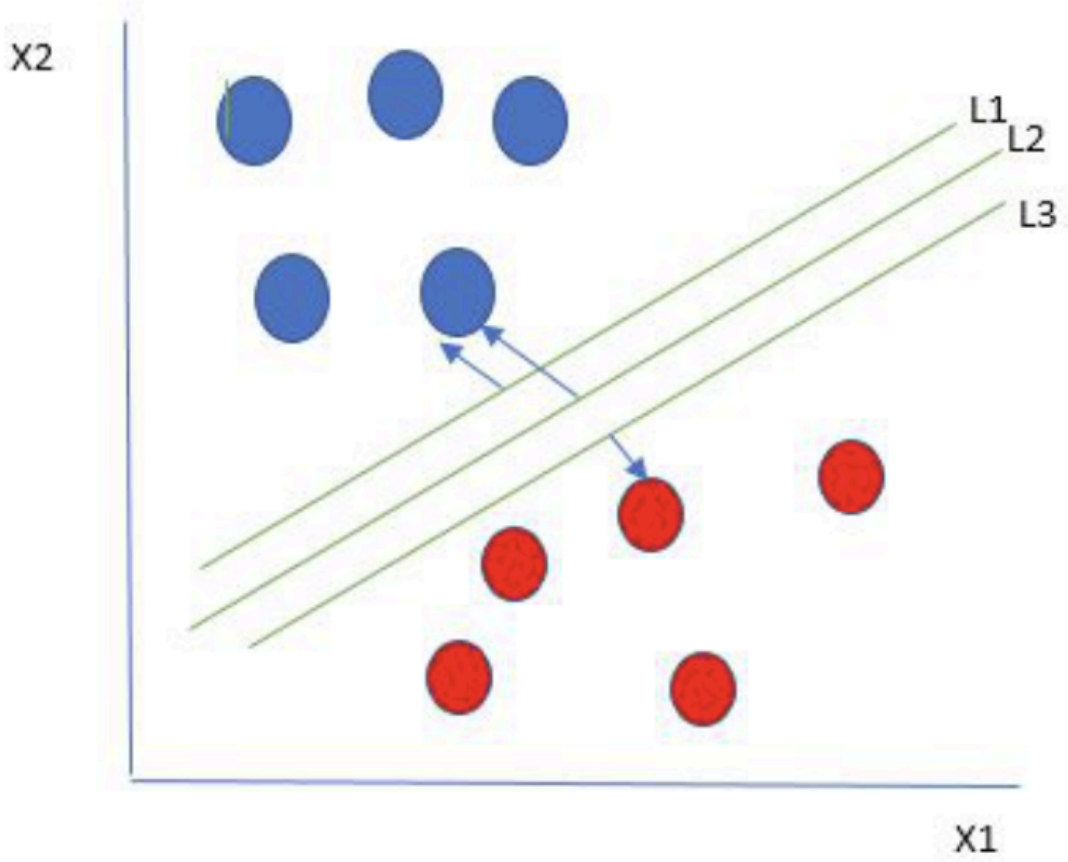
```
# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred, labels=np.unique(y))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(y))
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f'Confusion Matrix for {model_name}')
    plt.show()

# Plot confusion matrices
plot_confusion_matrix(y_test, dt_pred, 'Decision Tree')
plot_confusion_matrix(y_test, rf_pred, 'Random Forest')
plot_confusion_matrix(y_test, svm_pred, 'SVM')
```

5.

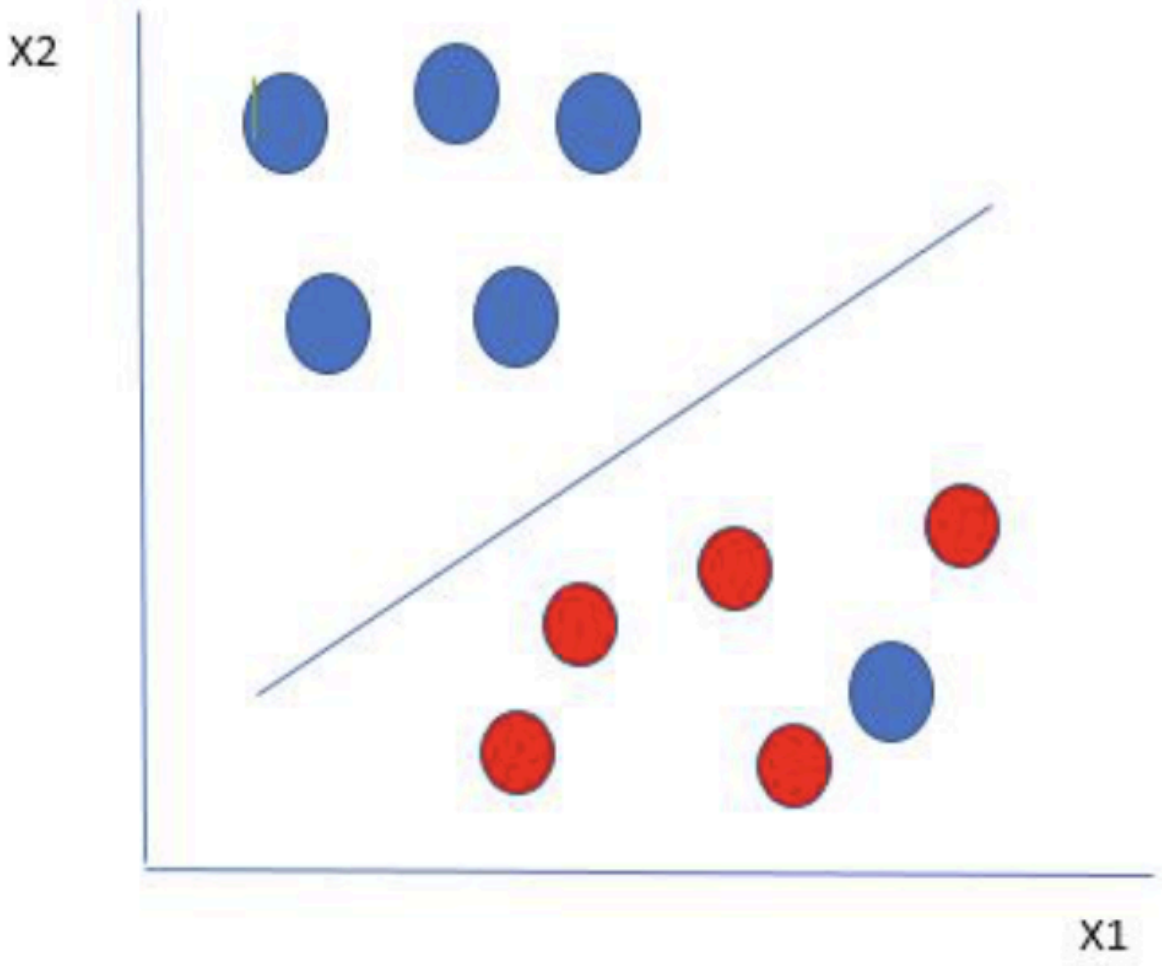
This enhanced notebook trains and evaluates three classifiers—Decision Tree, Random Forest, and SVM—on the Drug Classification dataset. It compares their accuracies and visualizes their performance using confusion matrices.

Note: Ensure that the dataset is in your working directory and that all necessary libraries are installed.

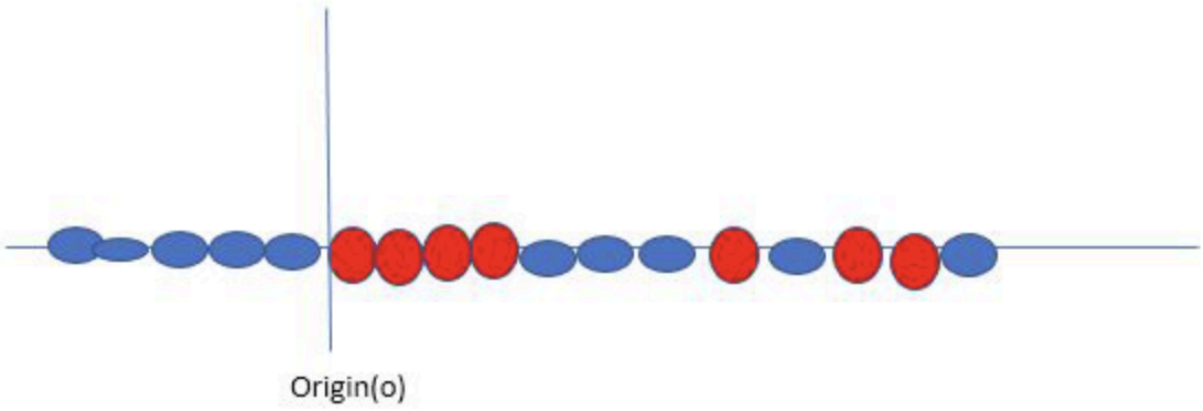

Maximum Margin
Support Vectors
Positive Hyperplane
Maximum Margin
Negative Hyperplane
Feature 2
Feature 1



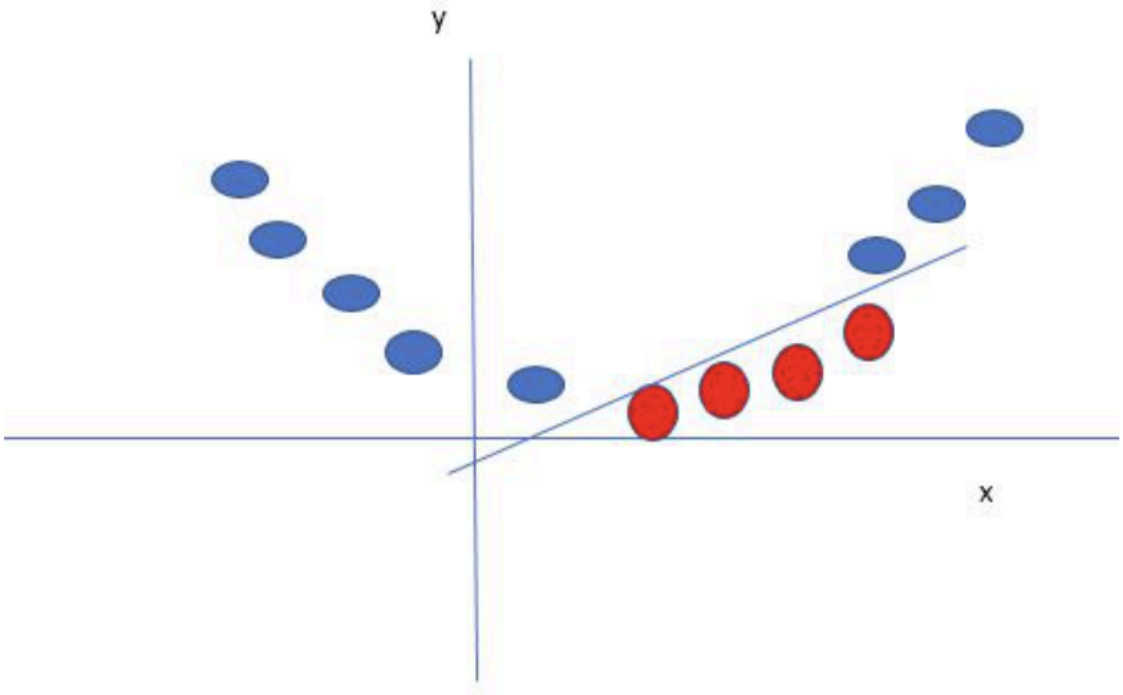
Multiple hyperplanes separate the data from two classes



Hyperplane which is the most optimized one



Original 1D dataset for classification

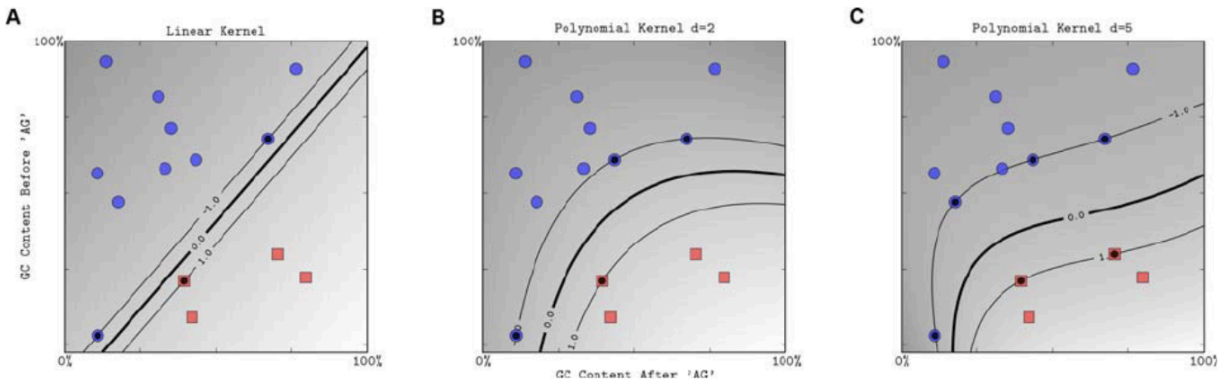


Mapping 1D data to 2D to become able to separate the two classes

$$K(x, x') = (x \cdot x' + c)^d$$

where:

- x and x' are input feature vectors,
- $x \cdot x'$ represents the dot product between x and x',
- c is a constant that controls the influence of higher-order terms (typically a small positive number or 0),
- d is the degree of the polynomial, which defines the complexity of the decision boundary.



1. **Define input vectors**:

$$x = [x_1, x_2] \quad \text{and} \quad x' = [x'_1, x'_2]$$

2. **Map input vectors to higher-dimensional space (degree-2 example)**:

$$\phi(x) = [x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1]$$

$$\phi(x') = [{x'_1}^2, {x'_2}^2, \sqrt{2}x'_1x'_2, \sqrt{2}x'_1, \sqrt{2}x'_2, 1]$$

3. **Compute dot product in the transformed space**:

$$\phi(x) \cdot \phi(x') = x_1^2{x'_1}^2 + x_2^2{x'_2}^2 + 2x_1x_2x'_1x'_2 + 2x_1x'_1 + 2x_2x'_2 + 1$$

4. **Rewrite using original dot product**:

$$\phi(x) \cdot \phi(x') = (x \cdot x' + 1)^2$$

5. **Generalize to degree d**:

$$K(x, x') = (x \cdot x' + c)^d$$