4 Statistical Machine Learning

First steps into Data Science

4.1
Statistical Machine Learning.

Statistical Machine Learning

- Statistical Machine Learning emphasizes on the statistical properties of datasets
- This is most commonly used where predictions are paramount
  - Stock Market forecasting
  - Medical Diagnosis

Understanding the framework

- So far we have understood how to handle the data
- Now we use the dataset to map out a problem to solve
    - Example: Given data (X,Y) where X is input and Y is output, what function can be a good predictor of F(X)=Y ?
- This is where ML models come into play

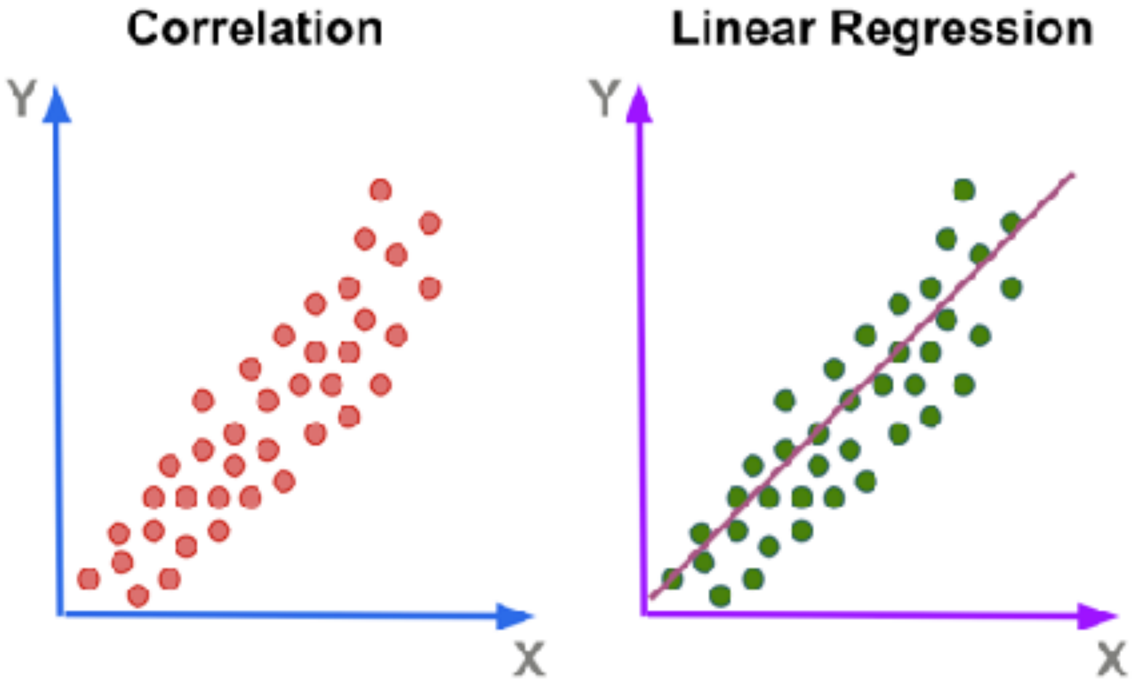
What roles to models play

- Models play the role of the function that is “fitted” onto the dataset
- The model takes in data X and predicts an output Y
- We evaluate the model with loss functions
- Lets learn a few of those models today

4.2
Linear Regression.

What is Linear Regression?

- Linear Regression is one of the simplest and most widely used statistical technique
- It’s goal is to model a relationship between a single dependent variable with one or more multiple independent variable



4.2. Linear Regression.

When to use Linear Regression

- **Nature of the Relationship**: When the relationship between the independent and dependent variable is believed to be linear.
- **Continuous Output**: When predicting values that are continuous (e.g., house prices, temperatures).
- **Interpretability**: When it's important to understand the influence of each feature on the output. Linear regression provides coefficients for each feature which indicate their relative importance.

Things to note

- Linear Regression works best when there is a linear relationship between the predictors and the response
- Regression tasks predict a value based on input data
- Works best when:
  - Residuals are normally distributed
  - Residuals have constant variance

Python Implementation

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_breast_cancer

# Load a sample dataset
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train the model
clf = LogisticRegression(max_iter=5000)
clf.fit(X_train, y_train)

# Make predictions on test data
predictions = clf.predict(X_test)
```

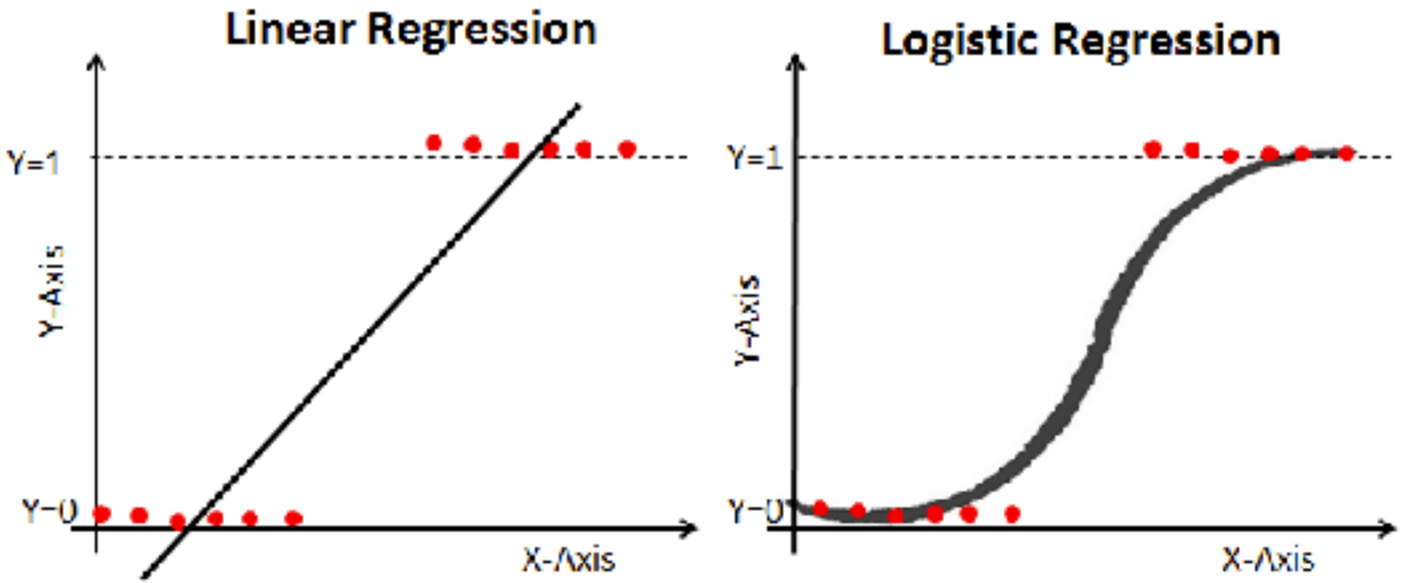
Cost Function in Linear Regression

- Mean Square Error (MSE): Average squared difference between actual and predicted values
- Adjust model parameters to minimize MSE

4.3
Logistic Regression.

What is Logistic Regression?

- Linear Regression is good for predicting continuous value but not much else
- Logistic Regression predicts the probability of occurrence of an event by fitting data to a logistic curve



4.3. Logistic Regression.

When to use Logistic Regression

- **Binary Outcome:** When the dependent variable is binary (e.g., spam or not spam, churn or not churn).
- **Probabilistic Results:** When you need to know the probability of your output. Logistic regression doesn’t just give a binary outcome, it gives the probability of that outcome.
- **Feature Importance:** Similar to linear regression, logistic regression provides coefficients that can help in understanding the influence of features.

Python Implementation

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_breast_cancer

# Load a sample dataset
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train the model
clf = LogisticRegression(max_iter=5000)
clf.fit(X_train, y_train)

# Make predictions on test data
predictions = clf.predict(X_test)
```

Cost Function in Logistic Regression

- Log-Loss: Measure the performance of a classification model whose output probability value is between 0 and 1

4.4
Decision Trees.

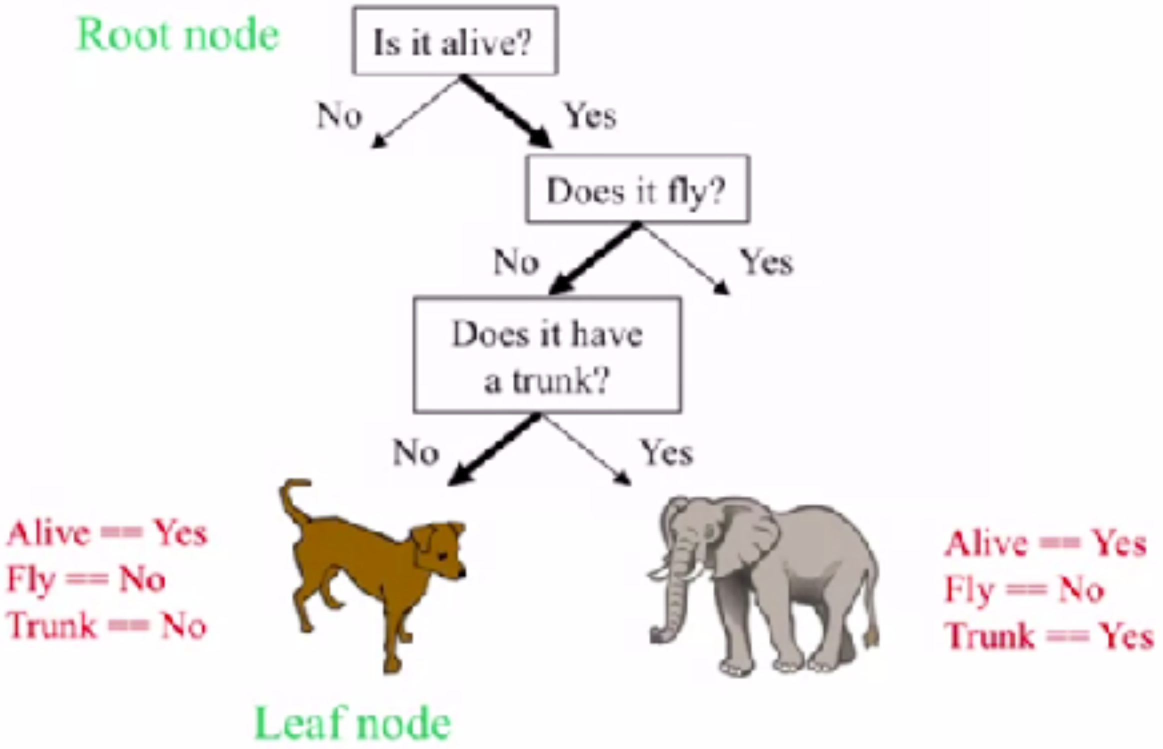
What are Decision Trees?

- Decision Trees split data into subsets
- This process is repeated recursively
- Results in a tree-like model of decisions

Components of Decision Trees

- Root Node: Represents the entire dataset, gets divided
- Decision Node: When a sub-node splits into further sub-nodes
- Leaf Node: Nodes that contain the decision to be taken

Decision Tree Example



Splitting Criteria

- **Impurity**: Measures how often a randomly chosen element would be incorrectly classified.
- **Entropy**: Measures randomness or unpredictability in the dataset.
- **Information Gain**: The entropy of the original dataset minus the weighted average entropy of the split datasets.

Overfitting Issue in Decision Trees

- **Overfitting:** When a model captures noise in the training data and performs poorly on new, unseen data.
- **Complex Trees:** Trees that are too deep can capture noise.
- **Pruning:** Process of reducing the size of a tree by turning some branch nodes into leaf nodes to reduce complexity.

When to use Decision Trees

- **Non-linear Relationships:** Decision trees can capture nonlinear relationships between features and the target variable
- **Interpretability:** They are easy to visualize and understand, making them great for deriving insights and rules
- **Categorical Input Features:** They handle categorical variables easily
- **Feature Interactions:** Decision trees can inherently capture interactions between features

Python Implementation

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris

# Load a sample dataset
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train the model
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

# Make predictions on test data
predictions = tree.predict(X_test)
```

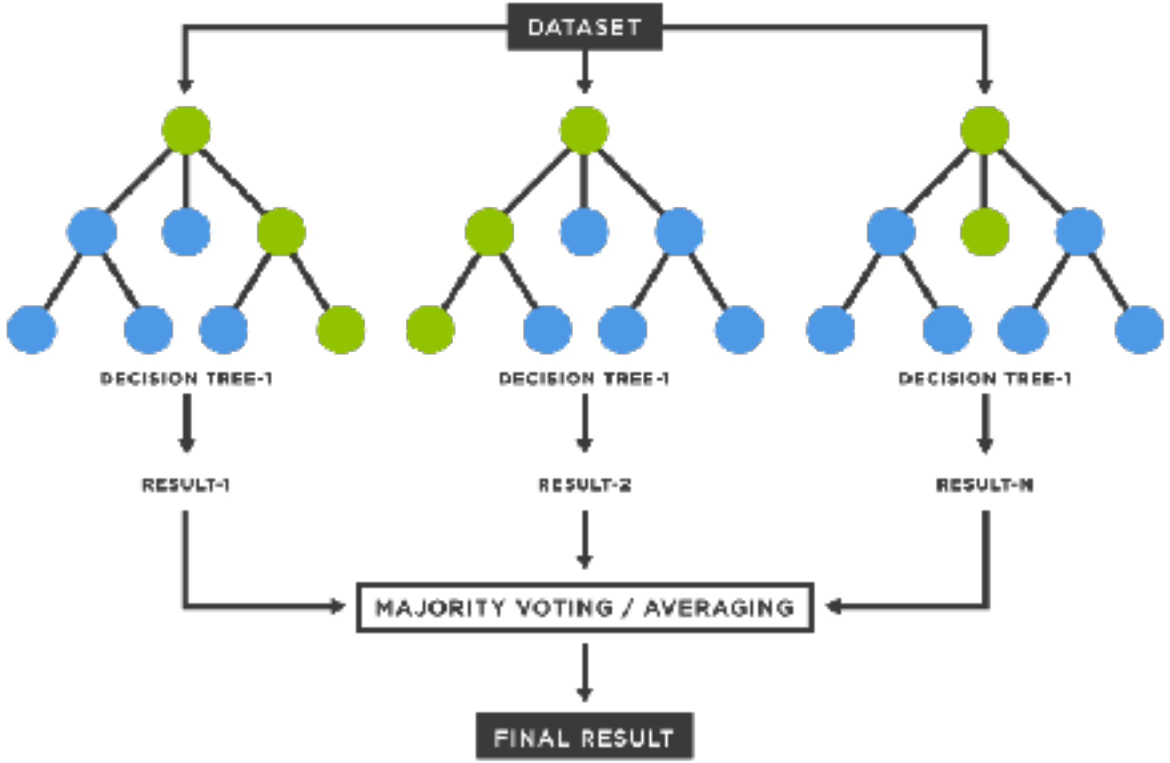
4.5 Random Forest.

What are Random Forests?

- Random Forests are an ensemble of decision trees
- Each tree in a Random Forest is trained on a random subset of data
- Bagging: A feature of Random Forests that aggregates decision of individual trees and reduces variance

Advantages of Random Forests

- **Reduction in Overfitting**: Diversity among trees reduces chances of overfitting.
- **Feature Importance**: Ability to rank features based on their importance in making predictions.
- **Handling Missing Values**: Can handle missing data without explicit imputation.
- **Generalization**: Often generalizes better to new data than individual trees.



4.5. Random Forests.

Disadvantages of Random Forests

- **Interpretability:** Harder to interpret than a single decision tree.
- **Computation:** Requires more computational resources.
- **Forest Size:** Need to choose the number of trees (more isn't always better).

When to use Random Forests

- **High Accuracy:** When performance is a primary concern. Random forests generally yield better accuracy than individual decision trees.
- **Feature Importance:** Random forests can rank features based on their importance in making accurate predictions.
- **Handling Overfitting:** Random Forests, through bagging, tend to reduce the overfitting that can be observed with individual decision trees.
- **Handling Large Data:** They can handle datasets with a higher dimensionality and can manage missing values.
- **Non-linear Data:** They can capture non-linear feature interactions.

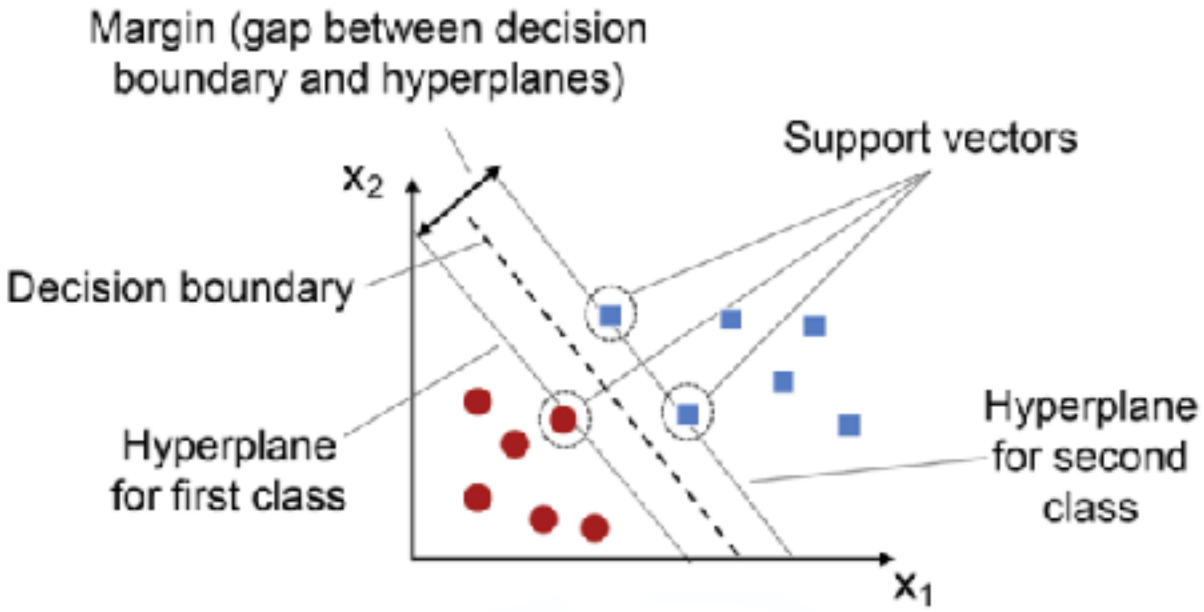
Python Implementation

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_digits

# Load a sample dataset
X, y = load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train the model
forest = RandomForestClassifier(n_estimators=100)
forest.fit(X_train, y_train)

# Make predictions on test data
predictions = forest.predict(X_test)
```

4.6
Support Vector Machines **(SVM)**.

What are Support Vector Machines (SVM)?

- SVM is a supervised ML algorithm which can be used for both classification or regression
- It performs classification by finding the hyperplane that best divides dataset into classes
    - Hyperplane: a generalized plane in different dimensions


Margin (gap between decision boundary and hyperplanes)
Support vectors
x_2
Decision boundary
Hyperplane for first class
Hyperplane for second class
x_1

4.6. Support Vector Machines.

Ideal Situations

- Text Classification problems
- When there is a need of margin separation
- Complex datasets where linear separation is not obvious

Drawbacks

- Inefficient on large datasets
- Sensitive to noise
- Requires fine-tuning using parameters such as the kernel

When to avoid

- Large datasets
- Datasets with a lot of noise
- When there is no clear margin or separation

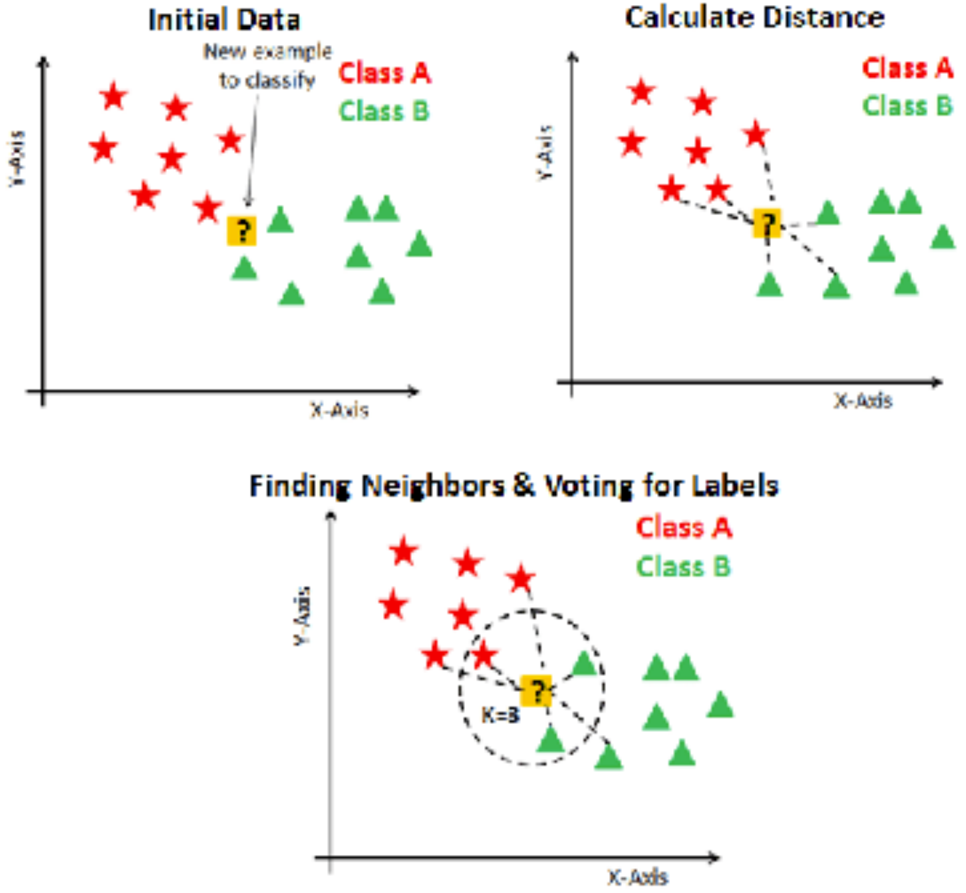
Python Implementation

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Training SVM
svm = SVC(kernel='linear', C=1)
svm.fit(X_train, y_train)

# Prediction
y_pred = svm.predict(X_test)
```

4.7
K-Nearest Neighbors **(KNN)**.

What is KNN?

- KNN is a non-parametric, lazy learning algorithm
- It assumes the similarity between the new data input with the available data
- Then assigns the new data into the category that is most similar to the available data categories



4.7. K-Nearest Neighbours.

Ideal Situations

- When the dataset is relatively small
- The data has little noise
- The data has decision boundaries are very irregular

Drawbacks

- KNNs becomes significantly slower as the number of examples grows
- It is sensitive to irrelevant or redundant features as all features contribute to the similarity

When to avoid

- When the dataset is relatively large
- The data has high number of dimensions
- Dataset has a ton of noise

Python Implementation

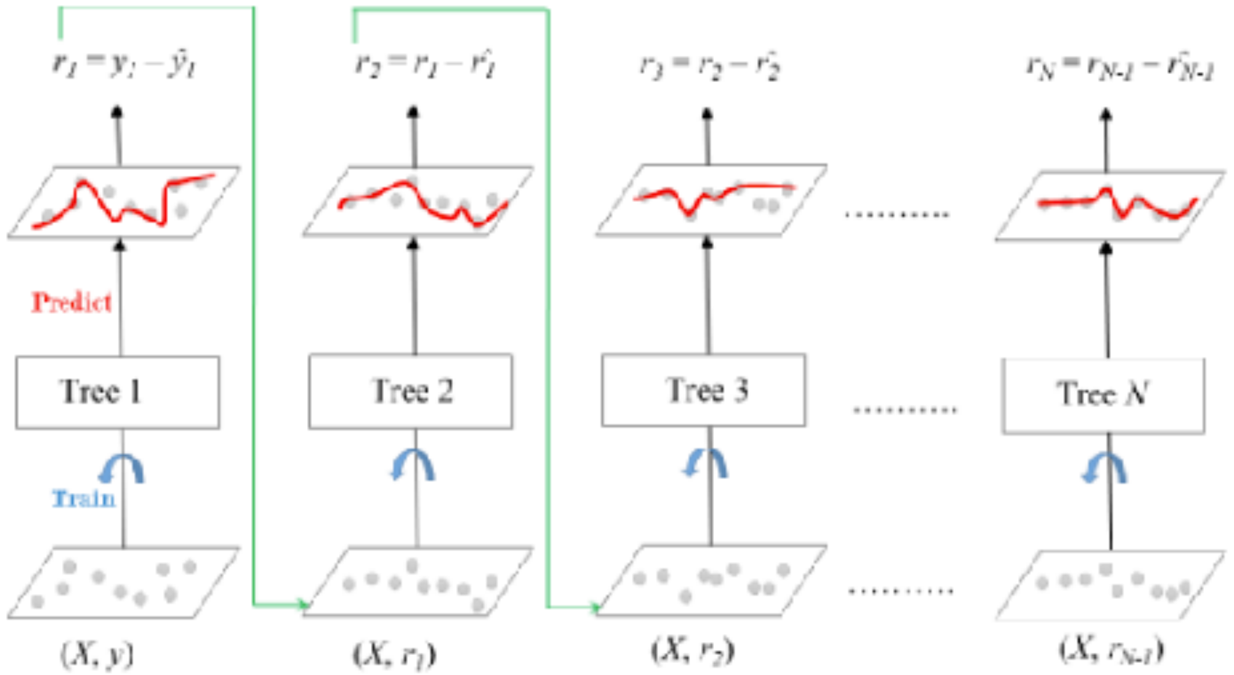
```
from sklearn.neighbors import KNeighborsClassifier

# Train/test split remains the same as the previous code

# Training KNN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Prediction
y_pred = knn.predict(X_test)

# Print accuracy
print("KNN Accuracy:", sum(y_test == y_pred) / len(y_test))
```

4.8
Gradient Boosting Algorithms.

What are Gradient Boosting Algorithms?

- Gradient Boosting is a boosting algorithm that combines several weak learners into strong learners
- It is an ensemble method
- Initialization: Begins with a simple model
- Residuals: It builds a sequence of trees where each new tree corrects the errors of its predecessors

Working of Gradient Boosting Algorithms



4.8. Gradient Boosting Algorithm.

Ideal Situations

- When there is an unbalanced dataset
- When model performance is the primary concern

Drawbacks

- Can overfit on noisy data
- Requires careful tuning of parameters
- Longer training time as trees are built sequentially

When to avoid

- When training time is a constraint
- Simple tasks where simple models can suffice

Python Implementation

```
from sklearn.ensemble import GradientBoostingClassifier

# Train/test split remains the same as the previous code

# Training Gradient Boosting Classifier
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=42)
gb.fit(X_train, y_train)

# Prediction
y_pred = gb.predict(X_test)

# Print accuracy
print("Gradient Boosting Accuracy:", sum(y_test == y_pred) / len(y_test))
```

Somethings to note

In practice, model selection often requires evaluating multiple models on the data and comparing their performance using some evaluation metric (e.g., RMSE for regression, accuracy/precision/recall for classification).

It's also essential to consider the domain-specific context, as practical considerations might dictate which model to choose even if it's not the absolute best performer.

END.