#UpSkillWithKalpesh

Day 19

Data Science Unlocked

From Zero to Data Hero

Classification in ML Algorithm Theory

Kalpesh Pathade
@DataSimplified

ML Classification Algorithms Theory

Type	@DataSimplified

Classification Algorithms in Machine Learning

1. Introduction

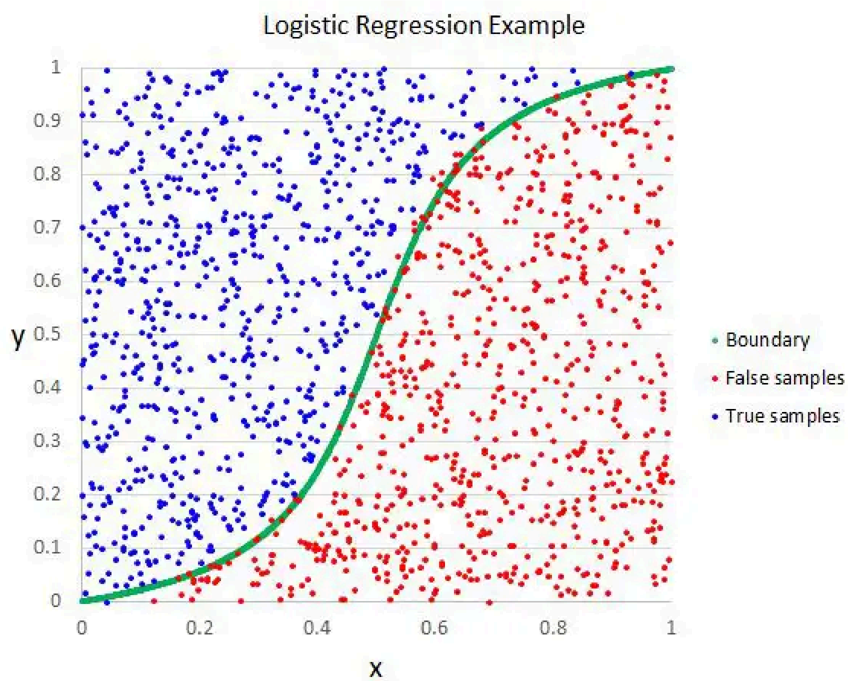
Classification is a supervised learning technique in machine learning where the goal is to predict the categorical label of a given input based on training data. It is widely used in applications such as spam detection, medical diagnosis, sentiment analysis, and image recognition.

1.1 Types of Classification

- **Binary Classification**: Two possible classes (e.g., spam vs. not spam)
- **Multi-Class Classification**: More than two classes (e.g., digit recognition: 0-9)
- **Multi-Label Classification**: Each instance can belong to multiple classes

2. Common Classification Algorithms

2.1 Logistic Regression



Theory:

Logistic regression is a linear model for binary classification that predicts probabilities using the **sigmoid function**:

$\sigma(z) = \frac{1}{1+e^{-z}}$

Where:

$z = w^T x + b.$

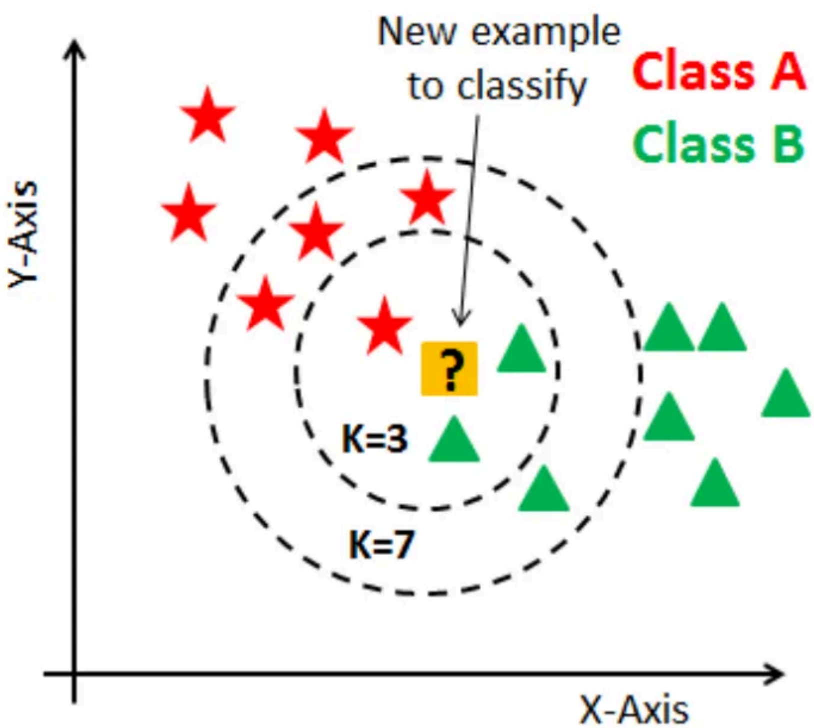
The decision boundary is determined by a threshold (e.g., 0.5).

Python Implementation:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Sample Data
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([0, 0, 1, 1, 1])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

# Model Training
model = LogisticRegression()
model.fit(X_train, y_train)

# Prediction
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

2.2 k-Nearest Neighbors (k-NN)



Theory:

k-NN is a non-parametric algorithm that classifies data based on the majority class among its k-nearest neighbors.

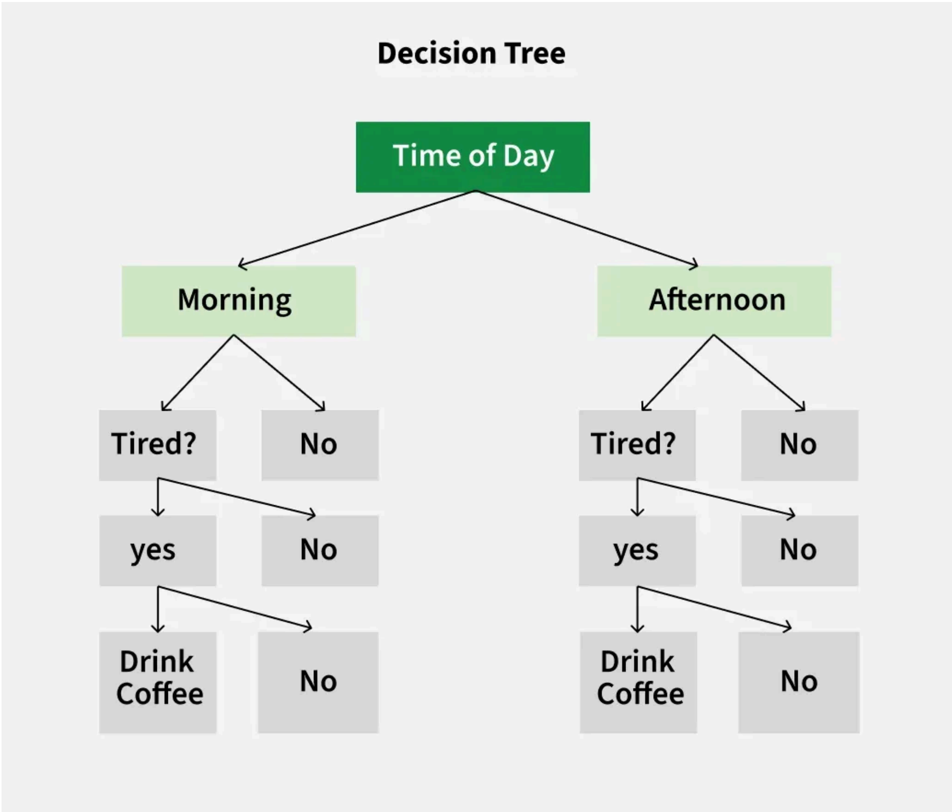
Steps:

1. Choose the number of neighbors k.
2. Compute the distance between the query point and all training samples (e.g., Euclidean distance).
3. Select the k-nearest samples and assign the majority class.

Python Implementation:

```
from sklearn.neighbors import KNeighborsClassifier

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

# Prediction
y_pred = knn.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

2.3 Decision Trees



Theory:

A decision tree splits the feature space recursively using the **Gini index** or **entropy**.

- **Entropy**:

  $$H(X) = -\sum p(x)\log_2 p(x)$$

- **Gini Impurity**:

  $$Gini = 1 - \sum p(x)^2$$

Python Implementation:

```
from sklearn.tree import DecisionTreeClassifier

# Model Training
dt = DecisionTreeClassifier(criterion='gini')
dt.fit(X_train, y_train)
```

```
# Prediction
y_pred = dt.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

2.4 Support Vector Machines (SVM)

Theory:

SVM finds the **optimal hyperplane** that maximizes the margin between classes using:

$\max \frac{2}{||w||}$

where w is the weight vector.

1. Definition of Support Vectors

Support vectors are the **data points that lie closest** to the decision boundary (or hyperplane). These points **directly influence the position and orientation** of the optimal hyperplane.

2. Role of Support Vectors

- **Margin Maximization**: The SVM aims to maximize the margin (distance) between the hyperplane and the closest points from both classes. The margin is defined by the support vectors.
- **Hyperplane Calculation**: The decision boundary is fully determined by these support vectors, meaning that **only these points matter**, and other training points do not influence the decision surface.
- **Robustness**: Since SVM only relies on support vectors, it is less sensitive to outliers that are far from the margin.

3. Mathematical Perspective

The optimal hyperplane is defined as:

$w^T x + b = 0$

where:

- w is the weight vector,

- b is the bias term.

The margin is given by:

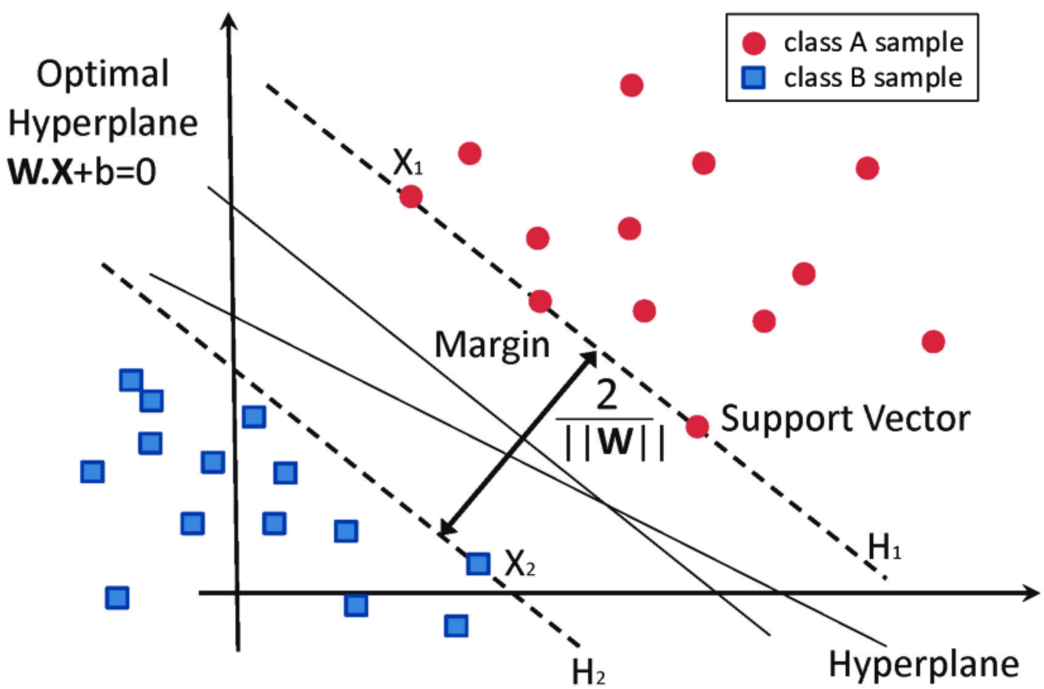
$$\frac{1}{\|w\|}$$

Support vectors satisfy the constraint:

$$yi(w^T x_i + b) = 1$$

where yiy_iyi is the class label (+1 or −1) and xi are the support vectors.

4. Visualization



- The **hyperplane** is positioned such that it maximizes the distance from the support vectors.
- The **margin** is the gap between the support vectors of the two classes.
- Support vectors **lie exactly on the margin boundaries**.

5. Impact on SVM Performance

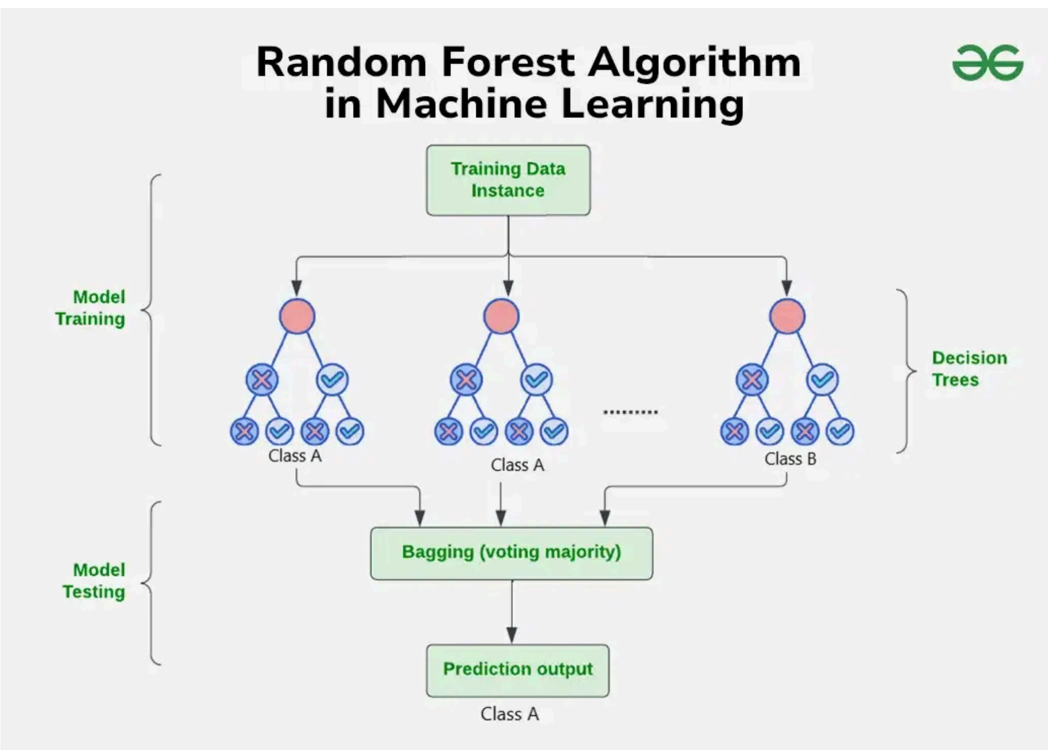
- **Fewer Support Vectors → Simpler Model**: If fewer points define the hyperplane, the model is computationally efficient.

- **More Support Vectors → Better Generalization**: More support vectors can make the model robust but may lead to higher complexity.

Python Implementation:

```
from sklearn.svm import SVC

# Model Training
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)

# Prediction
y_pred = svm.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

2.5 Random Forest



Theory:

Random Forest is an ensemble of multiple decision trees, where each tree votes for the final classification.

Python Implementation:

```
from sklearn.ensemble import RandomForestClassifier

# Model Training
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)

# Prediction
y_pred = rf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

3. Model Evaluation Metrics

3.1 Confusion Matrix

Actual \ Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

3.2 Performance Metrics

- **Accuracy**:

  $$(TP + TN)/(TP + TN + FP + FN)$$

- **Precision**:

  TP / (TP + FP)

- **Recall**:

  TP / (TP + FN)

- **F1 Score**:

  $$2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Python Code for Metrics:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```