Classfication Algorithms

Algorithm	Purpose	Concept	How It Works
Logistic Regression	Binary classification problems	Estimates the probability of a binary outcome based on one or more predictor variables.	Uses a logistic (sigmoid) function to model the relationship between the input features and the binary outcome.
Naive Bayes	Classification	Based on Bayes' Theorem; assumes features are independent.	Calculates the probability of each class given the features and chooses the class with the highest probability.
Decision Tree	Classification and regression	Builds a model in the form of a tree structure, with decisions based on features.	Splits the data into subsets based on feature values, recursively creating branches until a stopping criterion is met.
k-Nearest Neighbors (k-NN)	Classification and regression	Classifies data points based on the majority class among its k nearest neighbors.	Calculates the distance between the point to be classified and all other points, then assigns the class of the majority.
Support Vector Machines (SVM)	Classification and regression	Finds the hyperplane that best separates the data into classes by maximizing the margin.	Transforms data into a higher-dimensional space (if needed) and finds the optimal hyperplane to separate the classes.

Classfication Algorithms

Logistic Regression

1. **Initialize Parameters**:
    - Start with initial weights and bias (often set to zero).
2. **Forward Propagation**:
    - Compute the linear combination of inputs: $z = \mathbf{w}^T \mathbf{x} + b$
    - Apply the logistic (sigmoid) function to get the probability: $\hat{y} = \frac{1}{1 + e^{-z}}$
3. **Calculate Loss**:
    - Use the binary cross-entropy loss function to measure the error between predicted probabilities and actual labels.
4. **Backward Propagation**:
    - Compute gradients of the loss function with respect to weights and bias.
    - Update weights and bias using gradient descent.

5. **Repeat**:
    - Iterate steps 2-4 until convergence (loss stabilizes or a maximum number of iterations is reached).
6. **Predict**:
    - For new data, use the learned weights and bias to compute probabilities and classify based on a threshold (typically 0.5).

Naive Bayes

1. **Initialize**:
    - Calculate prior probabilities for each class.
2. **Feature Probabilities**:
    - For each feature, compute the conditional probability of that feature given each class.
3. **Prediction**:
    - For a new instance, compute the posterior probability for each class using Bayes' Theorem: $$P(C_k | \mathbf{x}) = \frac{P(C_k) \cdot \prod_{i} P(x_i | C_k)}{P(\mathbf{x})}$$
    - Choose the class with the highest posterior probability.
4. **Update Model** (if needed):
    - Adjust probabilities based on new data or feedback.

Decision Tree

1. **Select the Best Feature**:
    - Compute the best feature to split the data based on criteria like Gini impurity or Information Gain.
2. **Split the Data**:
    - Partition the data into subsets based on the chosen feature.
3. **Recursively Apply**:
    - Apply steps 1 and 2 to each subset to create child nodes.
    - Repeat until stopping criteria are met (e.g., maximum depth, minimum samples per leaf, or pure nodes).
4. **Predict**:
    - For a new instance, traverse the tree from the root to a leaf node, making decisions based on feature values at each node.

k-Nearest Neighbors (k-NN)

1. **Choose k**:
    - Decide the number of nearest neighbors to consider.
2. **Distance Calculation**:
    - Compute the distance (e.g., Euclidean, Manhattan) between the new instance and all training instances.
3. **Find Nearest Neighbors**:
    - Identify the k nearest training instances to the new instance based on the calculated distances.
4. **Classify**:
    - For classification, take a majority vote among the k nearest neighbors.
    - For regression, calculate the average value of the k nearest neighbors.
5. **Predict**:
    - Assign the class label or predicted value based on the majority vote or average.

Support Vector Machines (SVM)

1. **Initialize**:
   - Select a kernel function (e.g., linear, polynomial, RBF).
2. **Transform Data** (if using a non-linear kernel):
   - Map the input features into a higher-dimensional space using the kernel function.
3. **Optimize**:
   - Solve the optimization problem to find the hyperplane that maximizes the margin between classes. This involves minimizing the cost function with regularization.
4. **Support Vectors**:
   - Identify the data points that lie on the margin (support vectors).
5. **Predict**:
   - For a new instance, use the learned hyperplane to classify the data or predict values. Compute the decision function and classify based on which side of the hyperplane the instance falls on.