**Nombre del Algoritmo:** Alternating decision tree

**Descripción del Problema Computacional para la cual el algoritmo es una solución.**

**Delimitación del Problema:**

El algoritmo Alternating Decision Tree (ADTree) es una solución para problemas de clasificación supervisada. En lugar de construir un árbol de decisión estándar, ADTree alterna entre nodos de decisión y nodos de predicción (boosting), lo que permite manejar problemas complejos al combinar múltiples reglas simples. Es especialmente efectivo para datos no lineales y ruidosos.

**Comprensión del Problema:**

El problema se centra en aprender un modelo de clasificación que asigna etiquetas 𝑦 ∈ {−1,1} a puntos de entrada 𝑥∈𝑅𝑑 basado en un conjunto de datos etiquetados. ADTree utiliza boosting para mejorar la precisión de la clasificación.

**Clasificación del Problema:**

* Aprendizaje supervisado: El algoritmo se entrena en datos etiquetados.
* Clasificación binaria o multiclase: Aunque diseñado para binario, puede extenderse a múltiples clases.
* Boosting: Combinación de predictores débiles en un modelo fuerte.

**Viabilidad Computacional:**

ADTree es viable para conjuntos de datos medianos, pero puede volverse computacionalmente intensivo para datos de alta dimensionalidad o conjuntos grandes debido al costo acumulativo del boosting.

**Contexto del Problema:**

El algoritmo se aplica en:

* Medicina: Predicción de enfermedades basadas en síntomas.
* Finanzas: Clasificación de riesgos crediticios.
* Procesamiento de texto: Clasificación de correos como spam o no spam.

**Análisis de Datos y Estructura del Problema:**

1. Entrada:

* Un conjunto de datos etiquetados 𝐷 = {(𝑥1, 𝑦1),…,(𝑥𝑛,𝑦𝑛)}
* 𝑥i son vectores de características.
* Yi son etiquetas binarias.

1. Estructura:

* Nodos de decisión (pruebas sobre atributos).
* Nodos de predicción (ponderaciones acumuladas de predicción).

**Evaluación de Complejidad:**

* Entrenamiento:
  + Depende del número de iteraciones del boosting (𝑇) y el tamaño del conjunto de datos (𝑛).
  + Complejidad: 𝑂(𝑇⋅𝑛⋅𝑑) donde 𝑑 es la dimensionalidad.
* Predicción:
  + Complejidad lineal respecto al número de nodos del árbol: 𝑂 (𝑛𝑜𝑑𝑜𝑠).

**Análisis del Contexto del Problema para la cual el algoritmo es una solución**.

**Contexto general del Problema:**

ADTree aborda problemas de clasificación en los que los datos pueden ser no lineales y ruidosos. Su capacidad para combinar múltiples reglas simples mediante boosting lo hace robusto en condiciones adversas. Es útil cuando se busca interpretabilidad y precisión.

**Identificación de los conjuntos y estructuras de datos:**

1. Conjunto de datos:

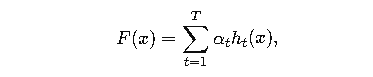
* Consta de características 𝑋 y etiquetas 𝑌.

1. Estructura de Datos:

* Árbol alternado con nodos de decisión (pruebas) y predicción (ponderaciones).

**Formalización del problema:**

El objetivo es aprender una función 𝐹(𝑥) que combine predictores débiles ℎ(𝑥) para predecir 𝑦:



donde 𝛼𝑡 es el peso asociado al predictor ℎ𝑡(𝑥) y 𝑇 es el número de iteraciones de boosting.

**Análisis de la Dinámica del Problema:**

1. Fase de Construcción:

* Generación iterativa de reglas de decisión (división de datos) y nodos de predicción (ponderaciones).

1. Predicción:

* Evaluación acumulativa de ponderaciones para cada ruta en el árbol.

**Evaluación de la Complejidad del Contexto:**

* Ventajas:
  + Buena capacidad para modelar datos no lineales.
  + Mejora en precisión con boosting.
* Desventajas:
  + Propenso a sobreajustarse si no se regulariza.
  + Complejidad computacional elevada para grandes conjuntos.

**Identificación de los casos límites y excepciones:**

**Casos límite:**

* Conjuntos de datos pequeños: Puede sobreajustarse.
* Altas dimensionalidades: Requiere técnicas para reducir el costo computacional.

**Excepciones:**

* Datos altamente correlacionados entre atributos pueden llevar a reglas redundantes.

**Interacción con otros sistemas:**

* Compatible con pipelines de aprendizaje automático, como selección de características o preprocesamiento.
* Puede integrarse con técnicas de validación cruzada para ajustar parámetros.

**Especificación Formal:**

**Definición de las entradas y salidas:**

Entradas:

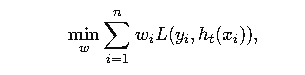
* D = {(x1​,y1​ ),…,(xn​,y n​)}: Datos de entrenamiento.
* T: Número de iteraciones (o nodos boosting).
* ϵ: Tolerancia al error.

Salida:

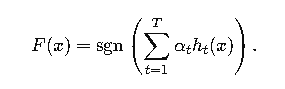
* Árbol alternado 𝐹(𝑥) para clasificar nuevos datos.

**Definición formal del problema mediante funciones:**

1. Generación del árbol:

* ****Iterativamente, para cada 𝑡 = 1,…,𝑇:

Donde L esa la función de perdida, y 𝑤𝑖, son pesos ajustados en cada iteración

1. Predicción:

​

**Relaciones y Restricciones:**

* **Relaciones:** 
  + Relación entre nodos de decisión y predicción: Los nodos de predicción dependen de las decisiones previas.
* **Restricciones:**
  + Las pruebas en nodos deben ser claras y no redundantes.
  + Los pesos 𝛼𝑡 deben ser no negativos para garantizar consistencia.

**Invariantes y propiedades:**

* Invariante principal: El peso acumulado 𝐹(𝑥) siempre refleja el balance entre predicciones correctas e incorrectas.
* Propiedad: El algoritmo reduce iterativamente la función de pérdida.

**Control de Flujo:**

1. Preprocesamiento:

* Normalización de los datos.
* Selección de características relevantes.

1. Entrenamiento:

* Inicialización de presos iguales
* Iteración:
  + Construcción de nodo de decisión.
  + Ajuste de nodo de predicción.
  + Actualización de pesos 𝑤𝑖.

1. Predicción

* Sumar ponderaciones a lo largo del árbol
* Tomar el signo del resultado acumulado.

**Pseudocodigo del Algoritmo:**

Algoritmo ArbolDecisionAlternante

// Entradas:

// datos\_entrenamiento: Matriz[n][m], donde n es el número de instancias y m es el número de atributos.

// etiquetas: Vector[n], valores esperados (clases).

// max\_iter: Número máximo de iteraciones para construir el árbol.

Funcion ConstruirADTree(datos\_entrenamiento, etiquetas, max\_iter)

Definir arbol como Lista // Representa el ADTree

Definir pesos como Vector[n] // Pesos iniciales de cada instancia

Inicializar pesos[i] <- 1/tamaño(datos\_entrenamiento) para todo i

// Inicializar nodo raíz con una predicción base

prediccion\_inicial <- CalcularPrediccionBase(etiquetas, pesos)

AñadirNodo(arbol, "Raíz", prediccion\_inicial)

// Iterativamente construir el árbol

Para iter <- 1 Hasta max\_iter

MejorCaracteristica <- SeleccionarMejorCaracteristica(datos\_entrenamiento, etiquetas, pesos)

MejorUmbral <- EncontrarUmbral(MejorCaracteristica, datos\_entrenamiento, etiquetas, pesos)

Confianza <- CalcularConfianza(MejorCaracteristica, MejorUmbral, datos\_entrenamiento, etiquetas, pesos)

// Agregar nodo de decisión

AñadirNodo(arbol, "Decisión", MejorCaracteristica, MejorUmbral)

// Agregar nodos de predicción asociados

AñadirNodo(arbol, "Predicción", Confianza, "Izquierda") // Si el criterio se cumple

AñadirNodo(arbol, "Predicción", -Confianza, "Derecha") // Si el criterio no se cumple

// Actualizar pesos

pesos <- ActualizarPesos(pesos, datos\_entrenamiento, etiquetas, MejorCaracteristica, MejorUmbral, Confianza)

FinPara

Retornar arbol

FinFuncion

Funcion CalcularPrediccionBase(etiquetas, pesos)

// Predicción inicial basada en la distribución ponderada de etiquetas

Definir suma\_positivos como 0, suma\_negativos como 0

Para i <- 1 Hasta tamaño(etiquetas)

Si etiquetas[i] == 1 Entonces

suma\_positivos <- suma\_positivos + pesos[i]

Sino

suma\_negativos <- suma\_negativos + pesos[i]

FinSi

FinPara

Retornar log(suma\_positivos / suma\_negativos)

FinFuncion

Funcion SeleccionarMejorCaracteristica(datos\_entrenamiento, etiquetas, pesos)

// Selección de la mejor característica según un criterio como ganancia de información

MejorCaracteristica <- -1

MejorPuntaje <- -Infinito

Para j <- 1 Hasta tamaño(datos\_entrenamiento[1])

Puntaje <- EvaluarCaracteristica(j, datos\_entrenamiento, etiquetas, pesos)

Si Puntaje > MejorPuntaje Entonces

MejorPuntaje <- Puntaje

MejorCaracteristica <- j

FinSi

FinPara

Retornar MejorCaracteristica

FinFuncion

Funcion ActualizarPesos(pesos, datos\_entrenamiento, etiquetas, caracteristica, umbral, confianza)

// Actualizar pesos basados en el error

Para i <- 1 Hasta tamaño(pesos)

Si datos\_entrenamiento[i][caracteristica] <= umbral Entonces

prediccion <- confianza

Sino

prediccion <- -confianza

FinSi

pesos[i] <- pesos[i] \* exp(-etiquetas[i] \* prediccion)

FinPara

Normalizar(pesos)

Retornar pesos

FinFuncion

// Proceso principal

Escribir "Iniciando construcción del ADTree..."

Definir datos\_entrenamiento como Matriz[n][m] // Cargar datos

Definir etiquetas como Vector[n] // Cargar etiquetas

Definir max\_iter como Entero // Número de iteraciones

arbol <- ConstruirADTree(datos\_entrenamiento, etiquetas, max\_iter)

Escribir "Árbol de Decisión Alternante construido:"

MostrarArbol(arbol)

FinAlgoritmo

**Casos de usos Posible:**

* Clasificación Binaria:
  + Predecir si un correo es "spam" o "no spam" basándose en características textuales.
* Diagnóstico Médico:
  + Clasificar enfermedades en base a síntomas y pruebas diagnósticas.
* Detección de Fraude:
  + Identificar transacciones fraudulentas en base a características de la operación.
* Análisis de Sentimientos:
  + Determinar si una reseña es "positiva" o "negativa" basándose en palabras clave.
* Predicción de Comportamiento del Cliente:
  + Clasificar si un cliente comprará un producto o no, basado en su historial de compras.
* Reconocimiento de Patrones en Datos Científicos:
  + Clasificar galaxias en diferentes tipos en estudios astronómicos.