PRÁCTICA 2 SISTEMAS INTELIGENTES
Alejandro García Gil 77042008N

TAREA 1A: IMPLEMENTA LAS CLASES ADABOOST Y DECISIONSTUMP

El código implementa estas dos clases: Adaboost y DecisionStump, que son parte de un algoritmo de aprendizaje automático conocido como Adaptive Boosting y sirve para mejorar el rendimiento de otros modelos de aprendizaje.

La clase “DecisionStump” está formada por:

-Constructor ‘__init__’ Esta función inicializa un tocon de decisión (DecisionStump) con características aleatorias.

-self.umbral: Se establece como un valor aleatorio multiplicado por 255. Este umbral se utiliza para tomar decisiones de clasificación basadas en una característica específica de los datos.

-self.carac: Se selecciona un índice de característica al azar, determinado por n_features (el número total de características disponibles).

-self.polaridad: Se elige aleatoriamente entre -1 y 1. Esto determina cómo se clasificarán los datos basándose en el umbral: si los datos están por encima o por debajo del umbral.

Método predict: Predice las etiquetas para un conjunto de datos X. Crea un vector de predicciones inicializado con 1s. Basándose en la polaridad y el umbral, modifica este vector para asignar -1 a ciertas observaciones, lo que indica su clasificación.

Clase “Adaboost” está formada por:

-Constructor ‘__init__’: Inicializa el algoritmo AdaBoost.

-self.T: Número de clasificadores débiles (tocónes de decisión) a entrenar.

-self.A: Número de intentos para encontrar el mejor clasificador débil en cada iteración.

-self.clasificadores: Una lista para almacenar los clasificadores entrenados

Método fit: Entrena el modelo AdaBoost.

Inicializa los pesos de las observaciones de manera uniforme. En cada una de las T iteraciones, intenta encontrar el mejor DecisionStump (clasificador débil) entre A candidatos. Calcula el error de cada clasificador y selecciona el que tiene el menor error. Actualiza los pesos de las observaciones basándose en el rendimiento del clasificador seleccionado. Guarda el clasificador seleccionado junto con su peso (alpha), que es calculado en función del error.

Método predict: Realiza predicciones en un conjunto de datos X. Combina las predicciones de todos los clasificadores débiles almacenados, ponderados por sus respectivos alpha. Puede devolver la confianza de la clasificación si return_confidence es verdadero.

TAREA 1B: MOSTRAR RESULTADOS DE TU CLASIFICADOR ADABOOST.

La funcion ‘train_and_evaluate_my_adaboost’ contiene lo siguiente parámetros, preparación de datos y la inicialización y entrenamiento del Adaboost.

-Parámetros:
digit: El dígito específico que se desea identificar frente a los demás.
T: Número de clasificadores débiles (tocónes de decisión) a utilizar AdaBoost.
A: Número de intentos para encontrar el mejor clasificador débil en cada iteración de AdaBoost.
X_train, Y_train: Conjuntos de datos de entrenamiento (características y etiquetas).
X_test, Y_test: Conjuntos de datos de prueba (características y etiquetas).
verbose: Un parámetro booleano que, si es verdadero, activará la impresión de información adicional durante el entrenamiento.

-Preparación de los datos:
Y_train_binary y Y_test_binary: Convierte las etiquetas del conjunto de datos en una forma binaria donde el dígito específico es 1 y todos los demás dígitos son -1. Esto es crucial para realizar una clasificación binaria.

-Inicialización y entrenamiento del modelo AdaBoost:
Crea una instancia de la clase Adaboost con los parámetros T y A. Registra el tiempo de inicio. Entrena el modelo AdaBoost en los datos de entrenamiento (X_train,Y_train_binary) y, si verbose es verdadero, imprime información sobre el proceso de entrenamiento. Registra el tiempo de finalización.

-Evaluación del modelo:
Utiliza el modelo entrenado para hacer predicciones en el conjunto de datos de prueba (X_test).
Calcula la precisión de las predicciones (test_accuracy) comparándolas con las etiquetas binarias del conjunto de datos de prueba (Y_test_binary).
Finalmente, la función devuelve la precisión de la prueba y el tiempo total de entrenamiento (calculado como la diferencia entre end_time y start_time).

TAREA 1C: AJUSTE ÓPTIMO DE T Y A

La función está diseñada para mostrar dos conjuntos de información relacionados con el rendimiento de los clasificadores AdaBoost entrenados con diferentes números de estimadores (T) y tamaños de muestra de entrenamiento (A):

-La línea roja representa la precisión de la clasificación en el conjunto de pruebas para cada clasificador. La precisión se calcula como el porcentaje de etiquetas correctamente predichas y se traza en función del número de estimadores T. Cada punto en esta línea corresponde a un clasificador AdaBoost entrenado con un T específico. El eje vertical izquierdo (con etiquetas en rojo) indica la escala de precisión, típicamente entre 0 y 1 (o 0% y 100%).

-La línea azul muestra el tiempo que tardó en entrenarse cada clasificador AdaBoost. Este tiempo se mide desde el inicio hasta el final del entrenamiento del clasificador y se traza también en función del número de estimadores T. Cada punto en esta línea indica cuánto tiempo se tardó en entrenar para un valor de T específico. El eje vertical derecho (con etiquetas en azul) muestra la escala del tiempo en segundos.

Además, se muestra el eje horizontal (T) común para ambas líneas refleja el número de estimadores utilizados para entrenar cada modelo AdaBoost. Este eje va desde el valor más pequeño de T hasta el más grande en los conjuntos de datos proporcionados.

Como conclusión y/o resumen de esta tarea, pienso que la gráfica está diseñada para ayudar a visualizar cómo la variación en el número de estimadores afecta tanto la precisión como el tiempo de entrenamiento, permitiendo al usuario encontrar un equilibrio entre el rendimiento del clasificador y la eficiencia computacional.

TAREA 1D: CLASIFICADOR MULTICLASE

La función evaluate_multiclass, tiene propósito que es evaluar clasificadores multiclase basados en AdaBoost para el reconocimiento de dígitos. Esta función es parte de un enfoque más amplio para clasificar los diez dígitos (0-9) utilizando el conjunto de datos MNIST, un conjunto de datos que consisten en imágenes de dígitos manuscritos.

La función ‘evaluate_multiclass’ contiene lo siguiente:

-Creación de Clasificadores Individuales: para cada dígito (0 a 9), se entrena un clasificador AdaBoost específico. Cada clasificador se especializa en distinguir un dígito en particular del resto. Por ejemplo, un clasificador se entrenará para identificar el dígito '0', considerando '0' como la clase positiva (1) y todos los demás dígitos como la clase negativa (-1).

-Entrenamiento de Clasificadores: utiliza la función train_and_evaluate_my_adaboost, previamente definida, para entrenar un clasificador AdaBoost para cada dígito. Los parámetros T (número de clasificadores débiles) y A (número de intentos para encontrar el mejor clasificador débil) son utilizados en el proceso de entrenamiento. Los conjuntos de datos de entrenamiento y prueba (X_train, Y_train, X_test, Y_test) se pasan a la función junto con el dígito actual.

-Estructura de Almacenamiento: los clasificadores entrenados se almacenan en un diccionario classifiers_manual, con las claves siendo los dígitos (0 a 9) y los valores siendo los clasificadores correspondientes.

-Clasificación Multiclase: en la práctica, para clasificar una nueva imagen de dígito, se ejecutarían los diez clasificadores. Cada clasificador daría una predicción de si la imagen representa su dígito específico o no. La clase final asignada a la imagen podría ser la del clasificador que tiene la mayor confianza (o un criterio similar) en su predicción.

TAREA 2A: MODELA EL CLASIFICADOR ADABOOST CON SCIKIT-LEARN

La función ‘train_and_evaluate_adaboost_sklearn’ está diseñada para entrenar y evaluar un clasificador AdaBoost utilizando la librería scikit-learn en Python. La función se enfoca

en la clasificación binaria de dígitos manuscritos, particularmente distinguir un dígito específico de todos los demás en un conjunto de datos. A continuación, se describe el propósito y la funcionalidad de cada parte del código:

-Definición de la función:
digit: El dígito específico que queremos clasificar contra todos los demás.
T: El número de clasificadores débiles (estimadores) que se utilizarán en el algoritmo AdaBoost.
A: No se utiliza en esta función y podría ser removido para evitar confusiones.
X_train, Y_train: Los conjuntos de datos de entrenamiento, donde X_train son las características y Y_train son las etiquetas.
X_test, Y_test: Los conjuntos de datos de prueba.
verbose: Un indicador para imprimir mensajes adicionales durante el entrenamiento (no se utiliza en esta función).

-Preparación de los datos:
Y_train_binary y Y_test_binary: Se convierten las etiquetas de entrenamiento y prueba (Y_train y Y_test) a una forma binaria para clasificación binaria, donde el dígito de interés se etiqueta como 1 y todos los demás dígitos como -1.

-Inicialización del clasificador AdaBoost:
ada_clf: Se crea una instancia de AdaBoostClassifier con DecisionTreeClassifier como el clasificador base y profundidad máxima de 1 (tocón de decisión). Se especifica el número de estimadores con n_estimators=T.

-Entrenamiento del clasificador:
start_time y end_time: Se registra el tiempo antes y después del entrenamiento para calcular la duración del entrenamiento.
ada_clf.fit: Se entrena el clasificador AdaBoost en el conjunto de datos de entrenamiento (X_train) con las etiquetas binarias (Y_train_binary).

-Evaluación del clasificador:
y_pred: Se realizan predicciones en el conjunto de datos de prueba (X_test).
accuracy: Se calcula la precisión de las predicciones comparándolas con las etiquetas binarias del conjunto de datos de prueba (Y_test_binary).

Como conclusión y/o resumen de esta tarea la función devuelve la precisión del clasificador en el conjunto de datos de prueba y el tiempo que tardó en entrenarse.

TAREA 2B: COMPARA TU VERSIÓN DE ADABOOST CON LA SCIKIT-LEARN Y OPTIMIZA LA CONFIGURACIÓN DEL CLASIFICADOR DÉBIL POR DEFECTO

El scikit-learn me ha resultado más preciso que mi versión de Adaboost pero, la velocidad de ejecución no tiene nada que ver, el scikit-learn ha resultado ser bastante más lento. ‘La función plot_accuracy_and_time’ contiene y consta de lo siguiente:
El objetivo de esta función tiene como objetivo visualizar la precisión (tasa de acierto) y el tiempo de entrenamiento de dos implementaciones de Adaboost: una manual y otra utilizando scikit-learn.

-Entradas:
T_values, A_values: Listas de valores para los parámetros T (número de clasificadores) y A (número de características) usados en Adaboost.
‘results_manual’ y ‘results_sklearn’ son diccionarios que contienen los resultados de precisión y tiempo de las implementaciones manual y de scikit-learn, respectivamente.

-Proceso:
Se itera sobre todas las combinaciones de T y A, recopilando los resultados de precisión y tiempo de entrenamiento de ambas implementaciones.

TAREA 2D: MODELA UN CLASIFICADOR MLP PARA MNIST CON KERAS

La función 'train_mlp_classifier', esta función configura y entrena un modelo MLP. Los parámetros incluyen epochs (número de ciclos completos de entrenamiento), batch_size (número de muestras procesadas antes de actualizar el modelo), learning_rate (tasa a la que el modelo aprende durante el entrenamiento), hidden_layers (lista que define el número y tamaño de las capas ocultas) y activation (función de activación para las capas ocultas).
El conjunto de datos MNIST, compuesto por imágenes de dígitos escritos a mano, se carga y prepara para el entrenamiento. Las imágenes se redimensionan y normalizan, y las etiquetas se convierten a formato categórico.
En cuánto a la compilación y entrenamiento, el modelo se compila utilizando el optimizador Adam y se entrena con los datos de MNIST. El tiempo de entrenamiento se registra para analizar la eficiencia del proceso.
Se evalúa el modelo con un conjunto de prueba para determinar su precisión.
Y en cuánto a la experimentación con configuraciones, la función probarMLP sirve como un ejemplo práctico de cómo utilizar 'train_mlp_classifier' para experimentar con diferentes configuraciones del modelo, como variar el número de neuronas, capas ocultas y el 'learning_rate'.
Estas experimentaciones son esenciales para entender cómo los distintos parámetros afectan la tasa de acierto y el tiempo de cómputo del clasificador MLP en el problema de reconocimiento de dígitos.

TAREA 2F: REALIZA UNA COMPARATIVA DE LOS MODELOS IMPLEMENTADOS.

La función 'comparar_clasificadores' está diseñada para evaluar y comparar el rendimiento de varios clasificadores multiclase. Recibe como entrada un diccionario denominado resultados, donde cada clave corresponde al nombre de un clasificador y su valor asociado es otro diccionario que contiene métricas específicas como 'precision' y 'tiempo_entrenamiento'.

La función realiza las siguientes operaciones:

-Imprime un resumen detallado de cada clasificador, enumerando sus métricas de rendimiento como la precisión y el tiempo de entrenamiento. Esto proporciona una visión clara del desempeño individual de cada clasificador.

-Realiza un análisis comparativo para identificar cuál de los clasificadores tiene la mejor precisión y cuál es el más rápido en términos de tiempo de entrenamiento. Esto se logra mediante el uso de funciones de agregación (max y min) aplicadas sobre el diccionario resultados.

La función está estructurada para permitir la adición de una discusión más detallada sobre la facilidad de uso y la configuración de cada clasificador, aunque estas partes están pendientes de implementación.

También se contempla un análisis de sobreentrenamiento para cada clasificador, aunque esta sección también está pendiente de implementación.