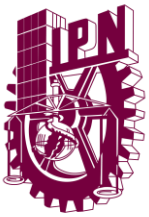

Proyecto integrador

Optimización de hiper parámetros de bosque aleatorio

De Hernández Jiménez Erick Yael

Para la materia de Algoritmos bio inspirados

Ingeniería en Inteligencia Artificial

Semestre 2025-1

Resumen

Como muestra de lo aprendido a lo largo del semestre, en este proyecto se implementa uno de los algoritmos bio inspirados para optimizar los hiper parámetros que regulan el comportamiento de un bosque aleatorio para clasificar los datos de cualquier conjunto sin datos faltantes y que puedan presentar una correlación entre sus atributos. Para el cometido anterior, se usó el algoritmo de minimización con abejas, el cual se practicó en clases anteriores y se desarrolla en un cuaderno Jupyter (archivo `ipynb`), junto con el mismo script necesario para ejecutar desde la terminal (archivo `py`).

Palabras clave: Python, *random forest*, bosque aleatorio, minimización, optimización, hiper parámetros, abejas, bio inspirado.

Planteamiento

Los bosques aleatorios son un algoritmo que combina el resultado de varios árboles de decisiones tanto para clasificar un conjunto de datos a partir de otros con los que se entrenan como para aplicar regresión a estos (IBM, s.f.).

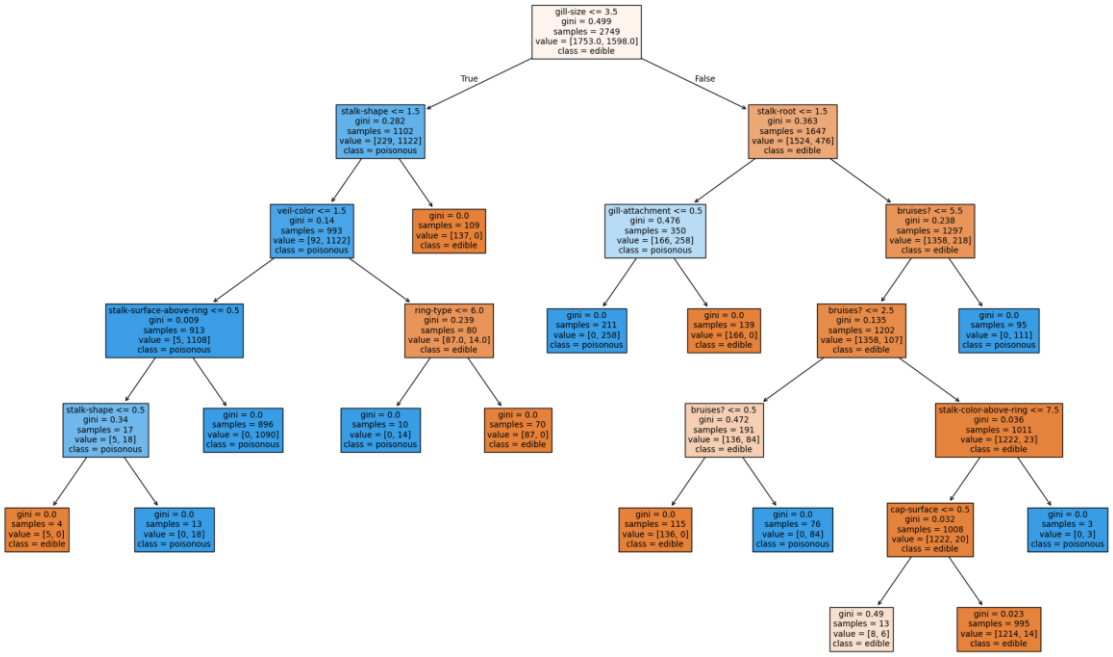

Ilustración 1. Ejemplo de árbol de decisión.

Gracias a la flexibilidad que este algoritmo presenta, es difícil que el afecte el fenómeno denominado *overfitting*, así como darle más peso a aquellas características con las que lo alimentamos, sin embargo, debido a la complejidad que implica computar varios árboles de decisión y aún más aquellos cuya extensión puede elevarse a cifras altas y que pueden sobrepasar por mucho el mínimo necesario para que este sea eficiente.

Por otro lado, como parte de los algoritmos bio inspirados vistos en clase, el algoritmo de abejas ofrece la posibilidad de explorar en el subespacio de soluciones mínimo para encontrar la configuración óptima para que el modelo se entrene y clasifique correctamente sin ocupar excesos de recursos computacionales.

Desarrollo

Bosque aleatorio de clasificación

Debido a que nos enfocaremos en la optimización en vez del desarrollo de un bosque aleatorio, usaremos la herramienta de clasificación desarrolla por *scikit-learn* con su clasificador *RandomForestClassifier*, misma que cuenta con los siguientes parámetros de interés (scikit learn, 2024):

- `max_depth`: número entero que describe la máxima profundidad que pueden tener los árboles cuyo rango semi estricto es de 1 a infinito o puede no estar definido.
- `max_features`: número entero o decimal o palabra que describe la cantidad u operación para seleccionar el número de características a considerar por cada árbol. En caso de ser un número decimal, el rango estricto es de 0 a 1, representando el porcentaje de atributos de los que se dispone en el conjunto de datos.
- `max_leaf_nodes`: número entero que describe el máximo de nodos hoja, podando los árboles en caso de que sobrepasen este límite y, así, evitando *overfitting*. El rango semi estricto dependerá de la profundidad máxima del árbol, por lo que es 1 a 2 a la potencia del número de profundidad.
- `max_samples`: número entero o decimal que describe el máximo de muestras que se deben tomar para entrenar cada árbol del total de instancias. En caso de ser decimal, se toma el rango estricto de 0 a 1 que corresponde al porcentaje del total de muestras de las que se dispone.
- `n_estimators`: número entero que describe el número árboles que contendrá el bosque. A este parámetro le corresponde el rango semi estricto de 1 a infinito, pero ya que pocos árboles no otorgan un resultado óptimo y evitar que la herramienta muestre un mensaje de advertencia, se cambiará el rango de búsqueda al rango de 60 a infinito.

En el ejemplo en el que se basó este proyecto para implementar el clasificador sin optimización (Gutiérrez García, 2021), únicamente se tomaron en cuenta los atributos `n_estimators` con 100 árboles, `max_features` con la configuración de la raíz cuadrada del total de atributos (abreviado como “sqrt”) y `max_samples` con el valor de $^2/_3$ o $66.\overline{6}\%$ del total de muestras del que se dispondrá. Con estos parámetros se consigue una precisión del 100% tanto en la validación como en la predicción de nuevos vectores.

Se usaron estos parámetros para guiar el espacio de búsqueda a valores cercanos a este agregando los parámetros `max_depth` y `max_leaf_nodes` con un máximo de 10.

Algoritmo de abejas

Anteriormente, el algoritmo de abejas se desarrollo para encontrar una solución óptima al problema de la mochila mismo problema que contaba con una función de costo (determina la eficiencia de la solución dentro del espacio de búsqueda considerado) y una función de ganancia (determina la eficacia de la solución).

Para este problema no se consideró una función de costo ya que no se definió una limitante para las soluciones más allá de los límites del espacio de búsqueda. Por otro lado, se definió a la función de ganancia como el promedio de las siguientes evaluaciones:

- score: porcentaje de la precisión de la validación entre los datos usados en el entrenamiento y las predicciones realizadas con el modelo ya entrenado.
- OOB score: porcentaje de la precisión de las predicciones hechas por el modelo ya entrenado con los datos no usados durante el entrenamiento (datos “fuera de la bolsa” de entrenamiento o *Out Of the Bag* en inglés).

Ya que ambas métricas son porcentuales, el valor necesario para que sea considerado que la optimización fue un éxito es de 1 o 100%.

Conjunto de datos

El código fue hecho para que pueda encontrarse el vector de valores mínimos necesarios para que el bosque pueda clasificar con precisión los datos de un conjunto sin valores nulos o irregulares en tipo de dato y con valores numéricos, textuales o categóricos.

Concretamente se usó el conjunto de datos Agaricus-Lepiota o Mushroom alojado en el repositorio de *datasets* de la UC Irvine (UC Irvine, s.f.) ya que este tiene un récord de precisión de clasificación con bosques aleatorios del 100%, por lo que aseguramos al menos una solución exitosa con este método de clasificación, permitiendo enfocarnos en optimizar el modelo.

Índice de contenidos

Índice de ilustraciones

Referencias

Gutiérrez García, J. O. (22 de 11 de 2021). Random Forest (Bosque Aleatorio) para Clasificación con Python. Recuperado el 26 de 12 de 2024, de https://youtu.be/yOCJQLf_YFI?si=fXBO-2WRFvA6s_e0

IBM. (s.f.). *IBM*. Recuperado el 26 de 12 de 2024, de What is random forest?: https://www.ibm.com/think/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,both%20classification%20and%20regression%20problems.

scikit learn. (2024). *RandomForestClassifier*. Recuperado el 26 de 12 de 2024, de scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

UC Irvine. (s.f.). *Mushroom*. Recuperado el 27 de 12 de 2024, de UC Irvine Machine Learning Repository: https://archive.ics.uci.edu/dataset/73/mushroom