Tarea 3 Tema 2: Arboles de decisión

Juan Manuel García Moyano
IABD
Informática y comunicaciones

Índice

1. Importación de librerías necesarias

He importado las librerías necesarias con el fin de entrenar un modelo con arboles de decisión. Además, todas las funcionalidades para realizar todos los apartados que no están directamente relacionado con el entrenamiento.

2. Preproceso

2.1 Importación de los datos del dataset

La importación de los datos lo he realizado desde el drive. Para ello subí el dataset, luego monté mi drive en colab y leí el csv.

2.2 Mostrar las primeras y últimas filas del dataframe importado

En colab, si ponemos en la última línea una variable muestra las primeras y últimas filas pero si se hace desde un entorno diferente el cual no sea jupyter, deberíamos utilizar los métodos head y tail.

2.3 Mostrar parámetros estadísticos de los datos (media, desviación típica, cuartiles, etc.)

Un dataframe tiene un método llamado describe que nos muestra los datos estadísticos de todos los atributos. Se debe de tener en cuenta que los datos estén en formato numérico.

En esta sección defino dos funciones para tratar con los outliers. Una de las funciones se utilizará más adelante.

2.4 Mostrar un mapa de calor que indique la correlación entre variables

Para realizar el mapa de calor he utilizado la librería seaborn. Para ello tuve de indicar la dimensión del mapa porque en un principio no se veía muy bien, ya que, los nombres de los atributos se juntaban. Por ello, primero lo redimensiono y después lo muestro con el método headmap(dataframe.corr(), square = True, annot =True). Vamos a desglosar los parámetros:

- dataframe.corr(): calcula la matriz de correlación entre las columnas numéricas del dataframe. La correlación mide la relación lineal entre dos variables, con valores que van desde:

  - 1: correlación positiva perfecta (ambas variables aumentas juntas).
  - -1: correlación negativa perfecta (una variable aumenta mientras la otra disminuye).
  - 0: no hay relación. Estas nos indican que deben ser eliminadas para el entrenamiento.
- square = True: fuerza que cada celda mapa de calor tenga forma cuadrada.
- Annot = True: muestra los valores numéricos de la correlación dentro de cada celda.

2.5 Seleccionar las características a tener en cuenta en el estudio

Para elegir las características a tener en cuenta en el estudio, va a ser conforme el mapa de calor. Como ya sabemos las mejores características para un estudio son las que más se acerquen a 1 o -1. En este dataset no existe alguna característica que tenga una fuerte relación entre las demás características y la objetivo. Las características serán:

- Alcohol
- Sulphates
- Density
- Total sulfur dioxide
- Citric acid
- Volatile acidity

2.6 Separar datos entre datos de entrada y etiquetas (resultados)

Primero elimino los outliers. El estudio de se va realizar con los atributos alcohol, sulphates, density, total sulfur dioxide, citric acid y volatile acidity. Para ello divido los datos en una variable (x) que va a tener los datos con los atributos anteriores y otra variable (y) con los datos del atributo objetivo.

2.7 Separar datos entre entrenamiento y prueba (usando un 80% para entrenamiento y 20% para test)

Para separar los datos utilizo la funcionalidad train_test_split. A este le paso dos variables una con los datos de la variable x y la otra variable es y. X tiene los datos sin los atributos anteriormente mencionados y la otra variable que es la que tiene todos los datos del atributo objetivo. Además, a esta función le paso los parámetros random_state (1) y test_size (0.20). Esto lo que hace que la división sea reproducible y representa la proporción del dataframe total asignada al conjunto de prueba, respectivamente.

Mediante el desempaquetado múltiple, lo que devuelve la función se lo asigno a 4 variables. Estas son:

- x_entrenamiento: tiene el 80% de los datos que van a servir para entrenar el modelo.
- x_test: tiene el otro 20% de los datos que van a servir hacer la predicción.
- y_entrenamiento: tiene el 80% de los datos que van a servir para entrenar el modelo.
- y_test: tiene el otro 20% de los datos que van a servir para calcular la probabilidad de acierto del modelo.

3. Entrenamiento y predicción

3.1 Elegir, instanciar y entrenar el modelo

Primero instancio el modelo deseado (clasificación). Después, lo entreno con las variables anteriores de entrenamiento.

3.2 Realizar una predicción con los datos de prueba

Al modelo ya entrenado hago que haga una predicción con los datos de x_test.

3.3 Mostrar el árbol de decisión resultante

Este código genera un gráfico visual de un árbol de decisión utilizando `graphviz` y `pydot`. Te lo explico en detalle:

1. dot_data = StringIO()
    - Se crea un objeto `StringIO` para almacenar la representación en formato DOT del árbol.

2. clases = [str(clase) for clase in modelo.classes_]
    - Se extraen las clases del modelo y se convierten en cadenas de texto.
3. tree.export_graphviz(...)
    - Se exporta la estructura del árbol de decisión en formato DOT, con nombres de características y clases.
4. graph = pydot.graph_from_dot_data(dot_data.getvalue())
    - Se convierte el código DOT en un objeto gráfico usando `pydot`.
5. Image(graph[0].create_png())
    - Se genera una imagen PNG del gráfico y se muestra.

3.4 Mostrar la importancia de cada atributo en el árbol resultante

Para cada característica elegida para el entrenamiento, muestro el nombre de la característica y su importancia. Podemos ver que algunas de ellas no tienen importancia en la decisión del modelo, esto hace que el modelo sea más complejo sin necesidad.

3.5 Intenta guardar el modelo de predicción ya entrenado usando dump (https://scikit-learn.org/stable/model_persistence.html)

Guardo el modelo con la librería joblib y su función dump. A esta le paso como argumentos, el modelo y el nombre del fichero.

4. Evaluación

Este ejercicio es un problema de clasificación, por eso realizo estos apartados.

4.1 Mostrar el porcentaje de elementos correctamente clasificados

Con la ayuda de la función accuracy_score, que me sirve para comprobar la probabilidad de acertar que tiene el modelo. A esta función le paso los datos y_test (datos del atributo calidad) y la predicción realizada en el anterior apartado. Muestro la probabilidad de 0 a 1.

4.2 Mostrar la predicción realizada (imprimir la variable con la predicción)

Muestro los datos que he realizado con la predicción.

4.3 Representar gráficamente la clasificación obtenida (matriz de confusión)

Aquí utilizo el método de ConfusionMatrixDisplay que es from_predictions. Se va a encargar de mostrarnos el mapa de confusión comparando los datos de test (y_test) y los datos de la predicción (y_prediccion).

5. Optimización de hiperparámetros

En estos apartados voy a desacerme del atributo volatile acidity porque en la importancia de los atributos del modelo tiene un valor de 0.00.

5.1 Calcula la combinación de parámetros óptima (profundidad de árbol y criterio). Para ello realiza ejecuciones con cada uno de los valores del criterio para los valores de la profundidad de árbol de 1 a 15.

Realizo los mismo pasos lo único que cambia son los criterios (gini y entropy). Primero separo los datos de entrenamiento de los datos objetivos. Me creo un dataframe que va a contener los datos de la predicción. Separo los datos de entrenamiento y los de test. Realizo un bucle con para indicar la máxima profundidad del modelo. Dentro del bucle declaro el modelo con el criterio que toque y la profundidad del bucle, entreno el modelo, realizo la predicción y la añado al dataframe definido anteriormente. Para finalizar muestro el dataframe con la profundidad máxima y su procentaje de acierto para cada criterio.

5.2 Cada ejecución anterior se deberá hacer usando validación cruzada (por ejemplo n_splits = 5). Con ello obtendremos una medida de bondad del modelo (accuracy_score o mean_absolute_error), como lo ejecutaremos 5 veces, calcularemos la media de esas 5 ejecuciones.

Realizo prácticamente lo mismo que en el apartado anterior pero en este apartado utilizo la función cross_val_score para realizar la validación cruzada.

5.3 Finalmente los parámetros elegidos serán los que den mejor media de esas medidas anteriormente nombradas.

El mejor resultado obtenido es con entropy y una profundidad máxima de 6, aunque en la profundidad 4 y 5 disminuye lo que puede ser que a partir de esos pasos empiece el sobreajuste.

5.4 Una vez obtenidos esos parámetros óptimos los aplicaremos al problema en cuestión y mostraremos los resultados.

Realizo un modelo con los mejores hiperparámetros que he mencionado en el anterior apartado.