Tarea 3 Tema 2: Arboles de decisión

Juan Manuel García Moyano
IABD
Informática y comunicaciones

Índice

1. Importación de librerías necesarias....3
2. Preproceso....3
2.1 Importación de los datos del dataset....3
2.2 Mostrar las primeras y últimas filas del dataframe importado....3
2.3 Mostrar parámetros estadísticos de los datos (media, desviación típica, cuartiles, etc.)....3
2.4 Mostrar un mapa de calor que indique la correlación entre variables....3
2.5 Separar datos entre datos de entrada y etiquetas (resultados)....4
2.6 Separar datos entre entrenamiento y prueba (usando un 80% para entrenamiento y 20% para test)....4
3. Entrenamiento y predicción....5
3.1 Elegir, instanciar y entrenar el modelo....5
3.2 Realizar una predicción con los datos de prueba....5
3.3 Mostrar el árbol de decisión resultante....5
3.4 Mostrar la importancia de cada atributo en el árbol resultante....6
3.5 Intenta guardar el modelo de predicción ya entrenado usando dump (https://scikit-learn.org/stable/model_persistence.html)....6
4. Evaluación....6
4.1 Mostrar el error cuadrático medio (mean_squered_error)....6
4.2 Mostrar el error absoluto medio (mean_absolute_error)....6
4.3 Representar gráficamente los valores predichos con los valores reales....6
5. Optimización de hiperparámetros....7
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....7
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....8
5.3 Finalmente los parámetros elegidos serán los que den mejor media de esas medidas anteriormente nombradas....8
5.4 Una vez obtenidos esos parámetros óptimos los aplicaremos al problema en cuestión y mostraremos los resultados....8

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 los csv. Uní ambos csv a partir del atributo dtaday que tienes en común.

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.

Como en este dataset tuve que unir dos en uno, aparecen demasiadas características. Para solucionar este problema primero realicé el corr eliminando el atributo dteday. El resultado lo filtré por los valores absolutos fuesen mayor al 0.5 y mostré un nuevo mapa de calor.

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

El estudio de se va realizar con los atributos registered_day, casual_day", atemp_day, temp_day, yr_day, instant_day, atemp_hour, temp_hour, yr_hour y instant_hour. 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. Más adelante veremos que la gran mayoría no los utiliza el modelo.

2.6 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 (regresión). 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. tree.export_graphviz(...)
    - Se exporta la estructura del árbol de decisión en formato DOT, con nombres de características y clases.
3. graph = pydot.graph_from_dot_data(dot_data.getvalue())
    - Se convierte el código DOT en un objeto gráfico usando `pydot`.
4. 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 regresión, por eso realizo estos apartados.

4.1 Mostrar el error cuadrático medio (mean_squered_error)

Con la ayuda de la función mean_squered_error, que me sirve para comprobar el error cuadrático medio 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 MSE.

4.2 Mostrar el error absoluto medio (mean_absolute_error)

Con la ayuda de la función mean_squered_error, que me sirve para comprobar el error cuadrático medio 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 MSE.

4.3 Representar gráficamente los valores predichos con los valores reales

- xx = np.stack([i for i in range(y_test.shape[0])])
    - Se crea un array xx con índices que representan la posición de cada muestra en el conjunto de prueba. Básicamente, es un array [0, 1, 2, ..., n] donde n es el número de muestras en y_test.
- plt.plot(xx, y_test, c='r', label='data')
    - Se traza la curva de los valores reales (y_test) en color rojo (c='r').

    - Se le asigna la etiqueta 'data', que aparecerá en la leyenda.
- plt.plot(xx, y_prediccion, c='g', label='prediccion')
    - Se traza la curva de los valores predichos (y_prediccion) en color verde (c='g').
    - La etiqueta 'prediccion' se usará en la leyenda.
- plt.axis('tight')
    - Ajusta los límites de los ejes automáticamente para que la gráfica se vea bien.
- plt.legend()
    - Muestra la leyenda con las etiquetas definidas en `label`.
- plt.show()
    - Muestra la gráfica en pantalla.

5. Optimización de hiperparámetros

En este apartado he eliminado todos los atributos cuya importancia sea 0.00. Los atributos con los que se ha quedado ha sido registered_day y casual_day.

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.

Primero separo los datos de entrenamiento de los datos objetivos. Me creo un dataframe que va a contener los datos de la profundidad máxima, el error cuadrático medio y el error absoluto medio. Separo los datos de entrenamiento y los de test. Realizo un bucle con para indicar la máxima profundidad del modelo. Dentro del bucle declaro la profundidad del bucle, entreno el modelo, realizo la predicción, calculo el error cuadrático medio y el error absoluto medio y añado los datos anteriores al dataframe definido anteriormente. Para finalizar muestro el dataframe.

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.

Si observamos los errores_medios de ambos, se puede observar que hasta el paso 12 se nota una mejoría notable, pero a partir de este puede ser que modelo se esté sobreajustando a sobreentrenando.

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.