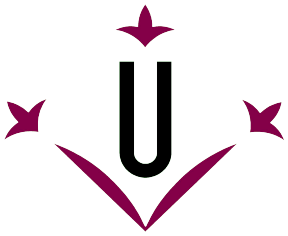

Universitat de Lleida
Escola Politècnica Superior

Inteligencia Artificial

Práctica 2: *Machine Learning*

Pere Rollón Baiges
Jordi Rafael Lazo Florensa

24 de enero de 2021

1 Árboles de decisión

1.1 T9 - Construcción del árbol de forma recursiva

Para la resolución de esta pregunta se ha implementado una función principal llamada *buildtree* y dos funciones auxiliares llamdas *split_dataset* y *get_column_values*.

- *buildtree*: esta es la función principal que se encarga de construir el árbol según el criterio de β.
- *split_dataset*: divide el conjunto de datos en dos particiones y devuelve la mejor de estas dos.
- *get_column_values*: esta función recoge los valores de un cierto atributo que se le pasa por parámetro, estos valores se usan como criterio de división de los conjuntos.

La idea principal de la construcción del árbol de forma recursiva se ha basado en que cada vez que se pasa por parámetro un *dataset* este se divide en dos ramas con la función *split_dataset*. Esta función crea dos conjuntos los cuales estarán divididos si pertenecen o no a los atributos del *header* del *dataset* y devuelve los conjuntos, su puntuación y el criterio de división.
Entonces comparando el parámetro β con la puntuación de cada par de conjuntos se decide si será tratado como un subárbol o como una hoja.

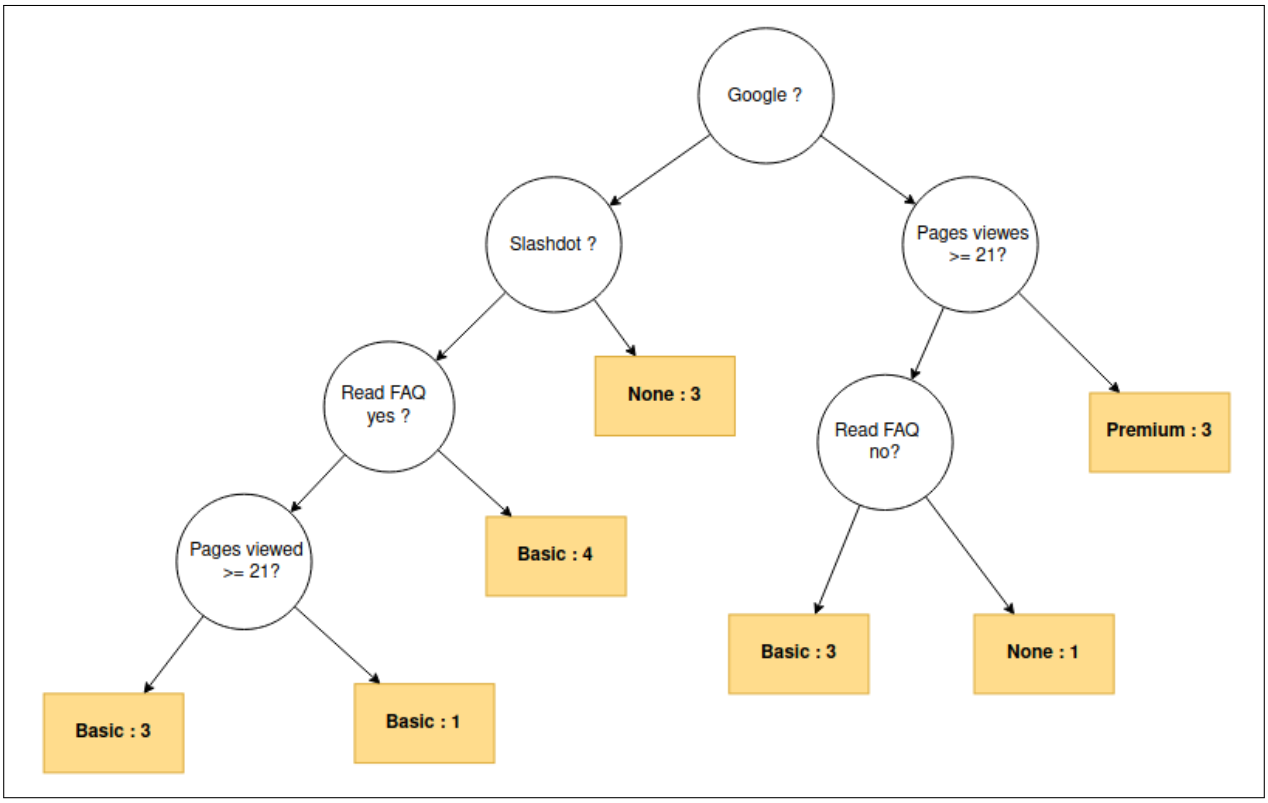

Figura 1: Representación gráfica del árbol usando el *dataset decision_tree_example.txt*.

1.2 T10 - Construcción del árbol de forma iterativa

Este método, igual que el anterior, crea un árbol de decisión, pero esta vez se tiene que construir de forma iterativa. Para hacer-lo, se ha implementado dos colas, una apila *datasets* nombrada *queue* y la otra los resultados nombrada *tree_queue*, estos se diferencian con un booleano, y la otra los nodos del árbol. Une vez declaradas las dos colas mientras la cola principal *queue* que apila los *datasets* no este vacía se desencolan el nodo y el *dataset*, seguidamente se comprueba que este último no sea el resultado. En el caso de no ser una hoja se procede a la partición del *dataset*. A continuación se comprueba que la puntuación supere β y si es así entonces se le aplican los criterios de partición creando los subárboles y encolando estos y las nuevas particiones. En caso contrario solamente se encolará de nuevo en el nodo, puesto que no ha sido modificado. Finalmente se calculan los resultados y se marcan como que no son *datasets*. En el caso que al desencolar el *dataset* y detectar que se trata de una hoja simplemente se añaden los resultados al nodo desencolado.

1.3 T12 - Función de clasificación

La función *classify* se ha pensado como una función recursiva que partiendo del nodo raíz del árbol se va moviendo por las ramas hasta llegar a un nodo hoja en función a los valores que tenga el nuevo objeto a clasificar. Cuando se llega a un nodo hoja se retorna su resultado.

2 Clustering

2.1 T9 - Total distance

Para la resolución de esta pregunta se ha implementado la función *_calculate_inertia* y su función auxiliar llamada *_intracentroid_distance*.

- *_calculate_inertia* : es la función principal que se encarga de hacer la suma de las distancias internas entre los centroides.
- *_intracentroid_distance*: retorna la suma de las distancias entre un centroide y sus puntos asignadas.

2.2 T10 - Distancia en función de k.

Para la resolución de esta pregunta se ha implementado la función nombrada *total _distance_function*. Esta se encarga de ejecutar el algoritmo *Kmeans* en funcion de un rango de k mostrando su inercia asociada.

2.3 T11 - Restarting policies

Dado el factor de aleatoriedad al inicializar los centroides, tras las pruebas realizadas, es posible que se acaben consiguiendo clústers que no son realmente representativos, para ello se ha modificado la función ya implementada *fit*, se pasa un parámetro nombrado *restarting policies*. Primero que todo es necesario saber cómo de buena es una disposición de clústers dada, para ello se utilitza la función *_calculate_inertia*, con la que dados los ítems y los centroides (sus vectores), calculamos la suma de las distancias entre cada ítem y su centroide asociado. De esa forma podemos medir cuantitativamente cómo de bueno es una disposición de centroides respecto a otra. Una vez que tenemos el sistema de calificación, para buscar la mejor disposición repetimos el proceso de *k-means* i veces, guardándonos la mejor disposición hasta el momento.

Una vez el proceso ha acabado, obtenemos como resultado la mejor disposición de centroides con ítems asociados de entre todas las iteraciones hechas.

2.4 Escoger un valor de k

Un paso fundamental para cualquier algoritmo no supervisado es determinar el número óptimo de agrupaciones en las que se pueden agrupar los datos. El método del codo *(Elbow method)* es uno de los métodos más populares para determinar este valor óptimo de k.

Para hacer el gráfico: se trazan los valores de K en el eje horizontal X y la inercia en el eje Y (los valores calculados con la función de costo).

Esto resulta en:

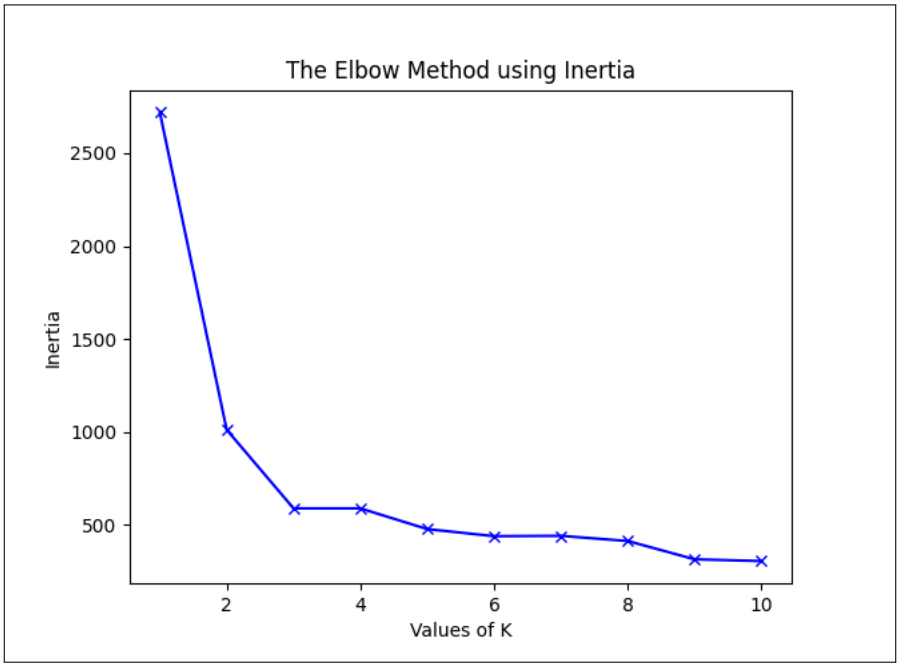

Figura 2: Gráfico de relación entre clústers e inercia usando el *dataset seeds.csv*.

El número k (# clústers) óptimo es el que forma el codo, en el caso de la Figura 2 el valor óptimo es k=*3*.