Universidad de Sevilla  
Escuela Internacional de Posgrado

**Proyecto de la asignatura FID 2021/2022**



Máster en Ingeniería del Software: Cloud, Datos y Gestión TI  
Fundamentos de Ingeniería de Datos

Curso 2021 – 2022

<https://github.com/sersanleo/FID>

|  |  |
| --- | --- |
| **Fecha** | **Versión** |
| 10/01/2022 | 1.0 |

|  |  |
| --- | --- |
| **Grupo de prácticas** | **7** |
| **Autores** | |
| Blasco Cuadrado, Reyes | |
| Fernández Fernández, Juan | |
| Sánchez León, Sergio | |
| Sola Espinosa, Fernando Luis | |

**Índice de contenido**

[**1.** **Presentación de los datos** 2](#_Toc92651202)

[**2.** **Preprocesamiento de datos y visualización de datos** 2](#_Toc92651203)

[**3.** **Análisis supervisado** 4](#_Toc92651204)

[**3.1.** **Clasificación** 4](#_Toc92651205)

[**3.1.1.** **Objetivos** 4](#_Toc92651206)

[**3.1.2.** **Trabajo realizado** 4](#_Toc92651207)

[**3.1.3.** **Conclusiones** 6](#_Toc92651208)

[**3.2.** **Regresión** 7](#_Toc92651209)

[**3.2.1.** **Objetivos** 7](#_Toc92651210)

[**3.2.2.** **Trabajo realizado** 7](#_Toc92651211)

[**3.2.3.** **Conclusiones** 10](#_Toc92651212)

[**4.** **Análisis no supervisado** 11](#_Toc92651213)

[**4.1.** **Clustering** 11](#_Toc92651214)

[**4.1.1.** **Objetivos** 11](#_Toc92651215)

[**4.1.2.** **Trabajo realizado** 11](#_Toc92651216)

[**4.1.3.** **Conclusiones** 15](#_Toc92651217)

[**4.2.** **Reglas de asociación** 16](#_Toc92651218)

[**4.2.1.** **Objetivos** 16](#_Toc92651219)

[**4.2.2.** **Trabajo realizado** 16](#_Toc92651220)

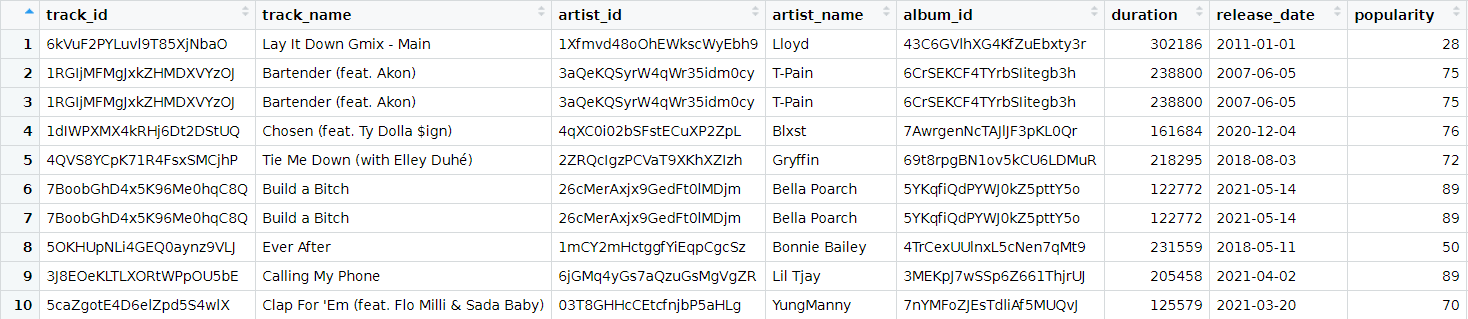
[**4.2.3.** **Conclusiones** 20](#_Toc92651221)

[**5.** **Reparto del trabajo** 21](#_Toc92651222)

# **Presentación de los datos**

Hemos elegido un *dataset* de canciones populares en **Tiktok** de Kaggle (<https://www.kaggle.com/yamqwe/tiktok-trending-tracks>), que contiene información acerca de unas **7.000 pistas de audio**. Algunos de los atributos que podemos saber de cada canción son su popularidad, energía, “bailabilidad”, alegría, duración, tempo, escala…

Dicha información es obtenida de la página <https://tunebat.com/>, que recoge datos de cerca de 70 millones de canciones; y otros, de Spotify (atributos como el id de la canción, id del artista o id de la lista de reproducción):



# **Preprocesamiento de datos y visualización de datos**

En cuanto al preprocesamiento de los datos, ha sido necesario realizar una serie de operaciones sobre el *dataset* inicial para hacerlo más usable. En primer lugar, nos percatamos de que había un gran número de canciones que estaban **repetidas**, por lo que lo primero que se tuvo que hacer es eliminar los duplicados, pasando el *dataset* de 6746 instancias a 4645.

Luego, había una serie de variables que no aportaban información relevante al ser **únicas** para cada fila o canción, o que eran **redundantes**, por lo que se decidió eliminarlas, como por ejemplo la de *playlist\_name*, *duration\_mins* o *track\_id*.

Una vez hecho esto, se procedió a formatear las **fechas**, ya que en algunas canciones sólo salía el año y en el resto la fecha completa, por lo que a estas canciones se decidió añadirle que por defecto su fecha de lanzamiento fue el 1 de enero del año que tuvieran en esa variable.

Por otro lado, se **normalizaron** las variables de loudness, popularity, acousticness y tempo.

Finalmente, se crearon 4 **variables dummy** a partir de la variable *genre*, ya que era categórica con 4 valores.

Previo al análisis y tratamiento de los datos con los distintos tipos de algoritmos y acercamientos, generamos los histogramas de varias de las variables para conocer de **forma visual** su distribución y ver a qué nos enfrentábamos:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

# **Análisis supervisado**

* 1. **Clasificación**
     1. **Objetivos**

El objetivo de la clasificación es la obtención de un modelo que permita predecir si una canción es **bailable** o no, en función de los valores de los demás atributos.

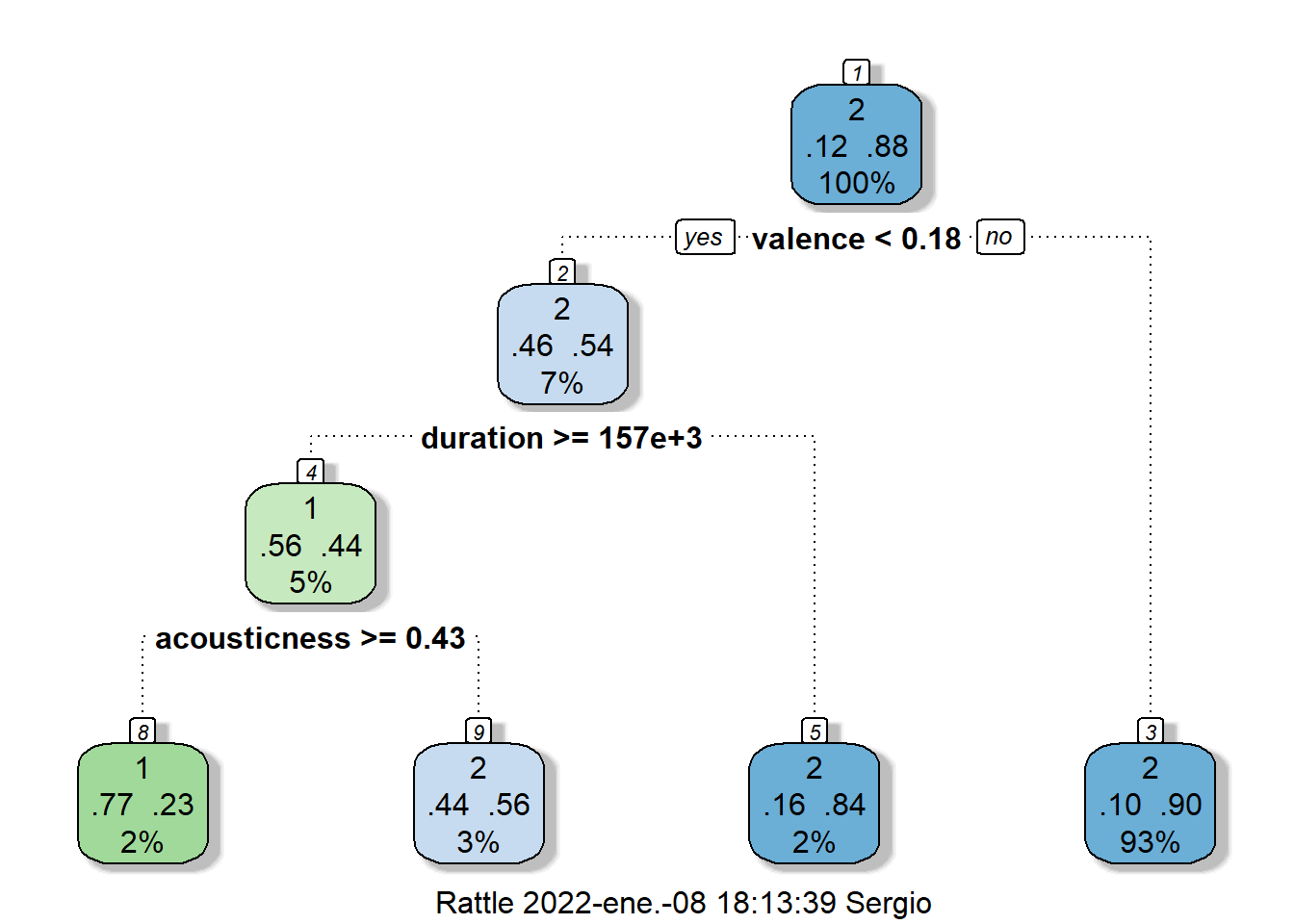
Al principio nuestro objetivo era el de realizar un modelo para predecir si una canción iba a ser o no **popular**, pero se encontró que no existía suficiente correlación con las demás variables como para realizar una predicción.

* + 1. **Trabajo realizado**

Se han analizado y comparado distintos modelos para obtener aquel que mejor realizase la tarea de clasificación.

Inicialmente, como la propiedad bailable como variable binaria no existía, ha sido creada mediante **discretización** de la variable numérica ***danceability*** en dos intervalos de igual anchura. Para los conjuntos de entrenamiento y prueba, se ha utilizado la función *createDataPartition*, dividiendo los datos originales en dos: 70% de datos para el conjunto de entrenamiento y 30% restante para el conjunto de prueba.

Se han construido tres **árboles de decisión** con la función *rpart*: uno con la heurística por defecto, otro usando como heurística la ganancia de información, y otro resultado de podar aquel con mejores métricas (para mejorar su interpretabilidad). Este último árbol es el siguiente:



Las métricas utilizadas han sido *accuracy*, *sensibility*, *specificity* y análisis ROC:

|  |  |  |
| --- | --- | --- |
| Sin ganancia de información | Con ganancia de información | Con ganancia de información (podado) |
|  |  |  |
| Accuracy: 0.8916009  Sensibilidad: 0.9722675  Especificidad: 0.2994012  AUC: 0.7476067 | Accuracy: 0.8987796  Sensibilidad: 0.9771615  Especificidad: 0.3233533  AUC: 0.7335574 | Accuracy: 0.8923187  Sensibilidad: 0.9918434  Especificidad: 0.1616766  AUC: 0.6171670 |

Para construir un **modelo de clasificación kNN**, basado en la distancia de la observación a clasificar en función de los vecinos más cercanos, se ha usado la función *knn* de R. Se han probado varios valores de k para conocer cuál es el que mejor *accuracy*, *sensitivity* y *specificity* devuelve.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| k: 12  **Accuracy**: 0.8937545  Sensibilidad: 0.9934747  Especificidad: 0.1616766 | k: 51  Accuracy: 0.8808327  **Sensibilidad**: 0.9983687  Especificidad: 0.0179641 | k: 1  Accuracy: 0.8758076  Sensibilidad: 0.9265905  **Especificidad**: 0.5029940 |

Gracias al paquete *caret*, también se han podido construir algunos modelos más utilizando la función *train* y modificando el valor del parámetro *method.* Se ha utilizado validación cruzada con 10 *folds*. Estos han sido las métricas resultantes:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | method | | | | |
| glm | glmnet | | glmnet tuning | xgbTree |
| ROC (AUC) | 0.7556501 | 0.7553013 | | 0.7567495 | **0.9501691** |
| Sensibilidad | 0.9906977 | 0.9906977 | | **0.9953488** | 0.9930761 |
| Especificidad | 0.140 | 0.140 | | 0.115 | **0.555** |
|  | | |  | | |

* + 1. **Conclusiones**

Viendo las métricas resultantes, de entre todos los modelos probados el que mejor predice si una canción es o no bailable es el devuelto por la función *train* de *caret* con el método ***xgbTree***.

* 1. **Regresión**
     1. **Objetivos**

El objetivo de la regresión es aprender a partir de los datos una función que nos permita aproximar lo mejor posible cómo de **bailable** es una canción dada (*danceability*) a partir del resto de atributos de la canción como por ejemplo la energía o la acústica, entre otros. Esto nos va a permitir saber a su vez si esta canción tendrá más o menos éxito en **TikTok**.

Cabe destacar que el objetivo inicial era aproximar la **popularidad**, no cómo de bailable es, pero al intentarlo obtuvimos unos malos resultados y llegamos a la conclusión de que dicha variable tenía poca correlación con el resto:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

* + 1. **Trabajo realizado**

Para llevar esto a cabo, se hizo uso del paquete *Caret* (Classification And REgression Training), y se realizaron **cuatro** **entrenamientos** distintos para compararlos y ver con cuál se extrae el mejor modelo.

Para ello, primero se ha realizado un preprocesado solo para esta parte ya que algunas columnas contienen *strings* que daban problemas a la hora de predecir una nueva instancia que tuviese en dichas variables algún valor diferente a los que se hubiesen visto en el entrenamiento.

Texto, Carta

Descripción generada automáticamente

Por tanto, lo que se ha hecho es eliminar la columna *track\_name* y se han sustituido los valores de la columna *release\_date* por el número de años que han pasado desde el 1 de enero de 1970 hasta dicha fecha, y de la columna *artista\_id* por un id numérico.

Después de este preprocesado, se ha dividido el *dataset* resultante de aplicarle el preprocesado en datos de **entrenamiento** y datos de **prueba**, conformando el conjunto de entrenamiento el 80% de los datos y el conjunto de pruebas el 20% restante. Estos porcentajes se han elegido porque proporcionan un rendimiento razonable y para que no haya sobreajuste.

Una vez tenemos el conjunto de entrenamiento, se realizan cuatro entrenamientos distintos usando la función *train* de *Caret* y *10-fold cross validation* para predecir la variable *danceability* a partir de las demás. Estos son:

El **primer** modelo se obtendrá utilizando el método *glmnet* y especificando un *tuneGrid*, que es el que se ha visto en las prácticas, en lugar del *tuneGrid* por defecto.

El **segundo** modelo se obtendrá usando también el método *glmnet* pero se usará en *tuneGrid* que viene por defecto, sin especificarse ninguno.

El **tercer** modelo se obtendrá usando el método *glm*.

Finalmente, el **cuarto** modelo se obtendrá usando uno de los algoritmos más populares en *Kaggle*, el *xgbTree*.

Después de haber obtenido estos modelos, podemos hacernos una idea de cómo de buenos son, sin tener que observar las métricas que devuelve la ejecución de cada entrenamiento, representando en una gráfica los valores reales frente a los valores predichos al utilizar la función *predict.train* con el conjunto de prueba, y una recta y=x, de tal forma que los puntos estarán más cerca de la recta cuanto mejor sea el modelo:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Finalmente, se comparan gráficamente los 4 modelos que hemos obtenido.

Gráfico

Descripción generada automáticamente

Gráfico

Descripción generada automáticamente

* + 1. **Conclusiones**

Como vemos, el mejor algoritmo con diferencia es el *xgbTree*, ya que obtiene un menor error cuadrático que el resto de modelos lineales y también obtiene el *Rsquared* más cercano a 1, siendo 1 el mayor grado de asociación lineal entre variables predictoras y respuesta.

Por tanto, utilizando ***xgbTree***, se obtienen unas predicciones de lo bailable que es una canción, y por consiguiente si podrá ser popular en *TikTok*, con un margen de error bastante aceptable, ya que el RMSE es de 0,1 aproximadamente, como se puede ver en la siguiente imagen, en la que se ha utilizado un conjunto de prueba con aproximadamente 900 instancias:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

# **Análisis no supervisado**

* 1. **Clustering**
     1. **Objetivos**

El objetivo principal del *clustering* es dividir en grupos (*clusters*) de tal forma que los datos que pertenecen al mismo clúster son similares mientras que los datos que pertenecen a otros, son diferentes.

En el caso actual, vamos a emplear la variable **popularidad** para realizar la comparación entre los distintos algoritmos de *clustering*.

* + 1. **Trabajo realizado**

En primer lugar, se construye una matriz con los valores de las variables numéricas, quitando los datos correspondientes a la columna popularidad, la cual utilizaremos para medir la **eficiencia** de los algoritmos de *clustering* implementados. Posteriormente se escalan los datos, para que todos se encuentren en la misma dimensión, y se obtiene la matriz de distancias que se utilizará en *clustering* jerárquico.

Para utilizar este vector de popularidad, se ha realizado una **categorización** de este para reducir sus valores a 4 posibles. Esta acción, simplificará la visualización de los datos posteriormente:

* 1: 0%-25% Popular.
* 2: 26%-50% Popular.
* 3: 51%-75% Popular.
* 4: 76%-100% Popular.

Antes de realizar el *clustering*, calculamos el k óptimo mediante el **método del codo**. A continuación, se muestra el resultado obtenido:

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Posteriormente, se realizó el *clustering* jerárquico y mediante *k means* utilizando el valor de k calculado anteriormente, obteniendo los siguientes resultados mediante validación externa:

* Clustering jerárquico:

Texto

Descripción generada automáticamente con confianza mediaImagen que contiene Escala de tiempo

Descripción generada automáticamente

* *Clustering* mediante *k-means*:

Texto

Descripción generada automáticamente

* Tabla

  Descripción generada automáticamente con confianza mediaValidación externa de ambos:

Texto

Descripción generada automáticamente

Si nos fijamos en los valores de precisión, especificidad y sensibilidad, podemos ver que el *clustering* jerárquico muestra los mismos valores para las dos últimas métricas, pero el valor de la precisión es mucho mayor, por lo que se podría afirmar que este algoritmo ofrece **mejores resultados** que el implementado mediante *k-means*.

Si introducimos PCA (Análisis de componentes principales) a la implementación de ambos algoritmos, se puede observar que en PC11 se obtiene un 0.9 de variabilidad, por lo tanto, se ha utilizado este valor para reducir la dimensionalidad del conjunto de datos.

Calendario

Descripción generada automáticamente con confianza media

Al repetir los algoritmos de *clustering* con este conjunto de datos, se obtiene los siguientes resultados:

* *Clustering* jerárquico:

Tabla

Descripción generada automáticamenteImagen que contiene Diagrama

Descripción generada automáticamente

* *Clustering* mediante *k-means*:

Texto

Descripción generada automáticamenteGráfico, Gráfico de dispersión

Descripción generada automáticamente

* Tabla

  Descripción generada automáticamenteValidación externa de ambos:

Texto

Descripción generada automáticamente

Como podemos observar, el algoritmo jerárquico sigue siendo más eficaz que el implementado mediante *k-means*, aunque se puede ver **cierta** **mejoría** en la implementación de *clustering* mediante *k-means* con PCA y sin ella.

* + 1. **Conclusiones**

Se puede concluir que el **número de *clusters* ideal** es cuatro. En base a este dato, tras realizar la implementación de los algoritmos, se puede observar, el algoritmo implementado que ofrece un **mejor resultado** es *clustering* jerárquico sin la introducción de PCA si nos fijamos en la precisión, especificidad y sensibilidad de los datos. En el caso de *clustering* mediante *k-means*, éste mejora al añadir PCA.

* 1. **Reglas de asociación**
     1. **Objetivos**

En esta sección pretendemos obtener información del *dataset* de forma que **encontremos patrones** del tipo si se da A, se da B, que nos permitan **estudiar factores de popularidad** de las canciones y características comunes de estas. Para ello, hemos trabajado los conceptos de **ítem**, ***itemsets*** comunes y por último, obtenido las **reglas de asociación**.

* + 1. **Trabajo realizado**

En la tarea de **extracción de reglas de asociación**, hemos encontrado un primer problema que nos ha obligado a realizar un preprocesamiento de los datos específico para esta tarea: las variables binarias generan reglas de tipo: X variable tiene un valor entre cero y uno, lo cual no aporta valor alguno.

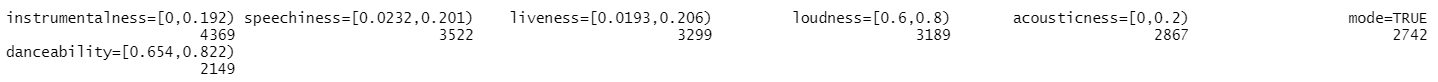
Por ello, nos hemos abstenido de crear la variable *dummy* a partir del *genre* de la canción y hemos tenido que traducir la variable binaria *mode* a valores *TRUE* o *FALSE* para que deje de tratarla como numérica.

Tras esto, el algoritmo *a priori* sólo es capaz de obtener una única regla, y la eliminación de otras variables no numéricas que no aportan mucho valor, tampoco mejoran la situación.

Finalmente, concluimos que se trata de los valores tan dispersos y **poco correlados** de las variables, todas normalizadas entre cero y uno, los que no permiten al algoritmo obtener algún tipo de conclusión, esto es, sacar alguna regla de asociación. Para solucionarlo, recurrimos a la **discretización de los valores de las variables numéricas** en varios intervalos, trabajando así el algoritmo *a priori* con menos valores y facilitándole un poco el trabajo.

De esta forma, observamos cómo para cuatro intervalos en los valores de las variables numéricas se generan unas 1500 reglas y para cinco intervalos, unas 700, por lo que la razón de ½ por cada unidad de intervalo afirma que a **mayor número de intervalos**, mayor diferencia de valores y **mayor dificultad** para el algoritmo *a priori* de **encontrar reglas**, como habíamos razonado previamente.

Una vez tenemos **preparados nuestros datos** para poderles aplicar algoritmos de reglas de asociación, comenzamos estudiándolos desde el concepto básico: el **ítem**. En nuestro caso cada canción tendrá un conjunto de ítems o variables con un valor concreto. Estudiando los ítems más frecuentes, tenemos:



Donde el ítem más frecuente es la variable *instrumentalness* con valor entre 0 y 0,192.

A continuación, estudiamos los **conjuntos de ítems** o ***itemsets*** más frecuentes, ordenados según su soporte o número de transacciones en que aparece, mediante la función *apriori* del paquete *arules*, la cual sirve también para la obtención de las propias reglas de asociación:

Tabla

Descripción generada automáticamente con confianza media

Como se puede observar, muchos de los *itemsets* contienen un único ítem. Si nos restringimos a aquellos que contienen un par al menos:

Tabla

Descripción generada automáticamente con confianza baja

Además del algoritmo *a priori*, hemos usado el algoritmo ***Eclat*** para la búsqueda de estos *itemsets* más frecuentes, aunque no hubo diferencia de resultados. La diferencia de funcionamiento entre ambos es que el primero distribuye y analiza las transacciones, en nuestro caso, las canciones, de forma horizontal, y el segundo lo hace de forma vertical.

Por último, pasamos al **estudio de las reglas de asociación** en sí, mediante la función *apriori*, que emplea el algoritmo homónimo. Con unos valores de soporte mínimo de 0.1 y confianza de 0.7, es capaz de encontrar alrededor de mil trescientas reglas que podemos ver representadas gráficamente según estas métricas:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Una vez tenemos todas estas reglas, podemos realizar tareas de **filtrado** para buscar aquellas que contengan cierto ítem o *itemset* que nos interese. Por ejemplo, podemos buscar aquellas con ítem *popularity* con un valor alto, de forma que encontremos alguna relación con otras variables que nos permitan discernir cuándo una canción va a ser popular. Este filtrado puede realizarse de dos formas:

* Directamente al ejecutar la función *apriori*.
* Obteniendo un *subset* filtrado a partir de las reglas ya obtenidas.

El problema encontrado en este punto es que, si se opta por la primera opción, sólo obtendremos reglas cuyos *itemset* antecedente o consecuente únicamente contengan el ítem filtrado, por lo que, si buscamos por un ítem de popularidad en el antecedente, por ejemplo, no vamos a obtener reglas que tengan otro ítem junto a él, descartando gran cantidad de información que puede ser interesante. Por ello, es preferible optar por la segunda opción, teniendo unos resultados del tipo:

Texto

Descripción generada automáticamente con confianza baja

De estos resultados podemos obtener información interesante como que, por ejemplo, para canciones con popularidad entre 0.6 y 0.8 se tendrá un grado de instrumentalidad de entre 0 y 0.192 con un soporte del 34% y una confianza del 95%. De la misma forma con canciones con esa popularidad se tiene que tendrán un *speechiness* de entre 0.0232 y 0.201 con un soporte del 27% y una confianza del 73%.

Así, podríamos seguir caracterizando estas canciones de alta popularidad, o podríamos filtrar por otra característica que nos fuera de interés y sacar otras conclusiones que nos aporten valor.

Podemos también, finalmente, **representar las reglas** en una representación de tipo grafo, la cual muestra la relación entre los distintos ítems:

Gráfico radial

Descripción generada automáticamente

* + 1. **Conclusiones**

Como hemos visto, las técnicas de aprendizaje no supervisado, en este caso las reglas de asociación, son una herramienta **muy útil** para obtener **patrones** y extraer **información** de interés de un conjunto de datos, como los factores que hacen popular una canción en TikTok, en nuestro caso, lo cual puede suponer una **ventaja competitiva** en el ámbito de un negocio o una empresa.

# **Reparto del trabajo**

Nuestro grupo de prácticas ha decidido ir a por el nivel 10, por lo que nos hemos dividido en 2 subequipos; uno de ellos se encargaría del análisis supervisado de los datos y el otro haría lo propio con el análisis no supervisado. Para repartir de forma equitativa la carga de trabajo, se ha acordado que cada persona de un subequipo se encargue de una técnica, quedando el reparto del siguiente modo:

* Análisis supervisado - clasificación: Sergio Sánchez León.
* Análisis supervisado - regresión: Juan Fernández Fernández.
* Análisis no supervisado – reglas de asociación: Fernando Luis Sola Espinosa.
* Análisis no supervisado - clustering: Reyes Blasco Cuadrado.