Grado en Ingenieria Informatica - UPC

Proyecto Aprendizaje Autónomo (APA)

KKBox's Music Recommendation Challenge

Raquel Leandra Pérez Arnal
Adrián Sánchez Albanell

Contents

1 Descripción del trabajo

1.1 Introducción

Este trabajo consiste en elegir un problema de regresión o clasificación y generar un modelo para resolverlo. Para ello usaremos algunos de los métodos, lineales y no lineales, vistos en clase durante el curso de Aprendizaje Autónomo.

Hemos elegido un problema de kaggle competitions sobre recomendación de música llamado WSDM - KKBox's Music Recommendation Challenge. Este consiste en un conjunto de datos, proporcionado por KKBOX - servició de streaming de música asiático - con información sobre diferentes canciones, usuarios y como ha sido el acceso de los usuarios a dichas canciones.

Nuestro objetivo es predecir si un usuario que ha escuchado una canción lo volverá a hacer en un periodo de tiempo determinado, por lo tanto se trata de un problema de clasificación binaria: si el usuario volverá a escuchar o no una canción que ya ha oído anteriormente.

1.2 Conjunto de datos disponible

Kaggle nos ha proporcionado los datos en seis ficheros con formato CSV, de los cuales usaremos cuatro para la práctica. Los dos restantes són un conjunto de datos de muestra sobre como enviar los datos para el concurso y los datos de test para el concurso (que no nos sirven ya que no tienen la variable target).

train.csv

Contiene la información de las reproducciones de canciones por parte del usuario. Tiene las siguientes variables:

msno	identificador del usuario.
song_id	identificador de la canción.
source_system_tab	nombre de la pestaña donde se selecciono el evento. Ejemplos: *my library*, *search*, etc.
source_screen_name	nombre de la pantalla que ve el usuario.
source_type	des de donde se ha reproducido la canción. Ejemplos: *album*, *online-playlist*, *song*, etc.
target	variable de target. Si el usuario ha escuchado la canción más de una vez en un intervalo de un mes target es 1, si no es 0.

members.csv

Contiene información de los usuarios. Tiene las siguientes variables:

msno	identificador del usuario.
city	identificador de ciudad.
bd	edad del usuario. Contiene valores outlier.
gender	genero del usuario. Puede ser *female* o *male*.
registered_via	identificador del método de registro de usuario.
registration_init_time	día del registro de usuario, en formato *%Y%m%d*.
expiration_date	día de expiración del registro de usuario, en formato *%Y%m%d*.

songs.csv

Contiene información de las canciones. Tiene las siguientes variables:

song_id identificador de la canción.

song_length duración de la canción en milisegundos.

genre_ids género musical de la canción. Hay canciones con más de un genero, donde el carácter | hace de separador.

artist_name nombre del artista.

composer nombre del compositor o compositores. Si hay más de uno el carácter | hace de separador.

lyricist nombre del escritor o escritores de la canción. Si hay más de uno el carácter | hace de separador.

language identificador del lenguaje de la canción.

song_extra_info.csv

Contiene información extra de las canciones. Tiene las siguientes variables:

song_id identificador de la canción.

song_name nombre de la canción.

isrc International Standard Recording Code. En teoría se puede usar como identificador de la canción, pero hay codigos ISRC sin verificar. Contiene información de la canción aunque puede ser erronea o confusa como el country code, que no se refiere a la canción si no a la agencia que proporciona el codigo ISRC.

1.3 Notas sobre el lenguaje de programación escogido, Python

El lenguaje de programación usado para realizar esta práctica tenía que ser R. Sin embargo pronto nos dimos cuenta de que, debido al tamaño de los datos de muestra, nuestro desconocimiento de como usarlo para trabajar con grandes cantidades de datos eficientemente y el material de que disponemos, nos iba a ser imposible trabajar con este lenguaje.

Debido a esto, hemos decidido usar Python, lenguaje muy usado en Machine Learning y con muchos recursos para trabajar comodamente en este ambito. Python es bastante más eficiente que R gestionando memória y más rápido en cuanto a tiempo de ejecución.

Usar Python no ha solucionado todos los problemas generados por tener tantos datos, pero nos ha permitido trabajar mejor y realizar muchas cosas que con R nos habrian sido o imposibles o muy dificiles.

2 Trabajo Relacionado

3 Análisis y preprocesamiento de los datos

3.1 Análisis inicial de los datos

Partiendo de los datos iniciales (descritos en el apartado *Conjunto de datos disponible*) hemos generado un solo data frame uniendo primero la información de **members** y **train** por *msno*, y luego uniendo el resultado a **songs** y **song_extra_info** por *song_id*. Al nuevo conjunto de datos con toda la información lo llamaremos **merged**. El nuevo conjunto de datos se compone de 20 variables y 7377418 muestras diferentes. La siguiente tabla nos muestra información sobre los valores perdidos en cada variable y su tipo:

Variable	Valores perdidos	Porcentaje	Tipo de datos
msno	0	0	categóricos (object)
song_id	0	0	categóricos (object)
source_system_tab	24849	0.3368	categóricos (object)
source_screen_name	414804	5.6226	categóricos (object)
source_type	21539	0.2919	categóricos (object)
target	0	0	categóricos (uint8)
city	0	0	categóricos (int64)
bd	0	0	numéricos (int64)
gender	2961479	40.142	categóricos (object)
registered_via	0	0	categóricos (int64)
registration_init_time	0	0	numéricos (int64) (*%Y%m%d*)
expiration_date	0	0	numéricos (int64) (*%Y%m%d*)
song_length	114	0.0015	numéricos (float64)
genre_ids	118455	1.605	categóricos (object)
artist_name	114	0.0015	categóricos (object)
composer	1675706	22.7139	categóricos (object)
lyricist	3178798	43.0882	categóricos (object)
language	150	0.0020	categóricos (float64)
name	1457	0.0197	categóricos (object)
isrc	577858	7.8327	categóricos (object)

Table 1: *Información general sobre el conjunto de datos merged.*

En la tabla siguiente vemos de cuantas categorías consta cada variable categórica. Esto es muy importante en nuestro caso porque contamos con un conjunto de datos muy grande y el número de categorías añade mucha complejidad a los datos cuando queramos diseñar los diferentes modelos:

Variable	Número de categorías
msno	30755
song_id	359966
source_system_tab	8
source_screen_name	20
source_type	12
target	2
city	21
gender	2
registered_via	5
genre_ids	166
artist_name	40582
composer	81566
lyricist	38473
language	10
name	234144
isrc	269760

Table 2: *Número de categorías en las variables categóricas del conjunto de datos.*

Con esta información pasaremos a tratar los datos para preprocesarlos.

3.2 Tratamiento de valores perdidos

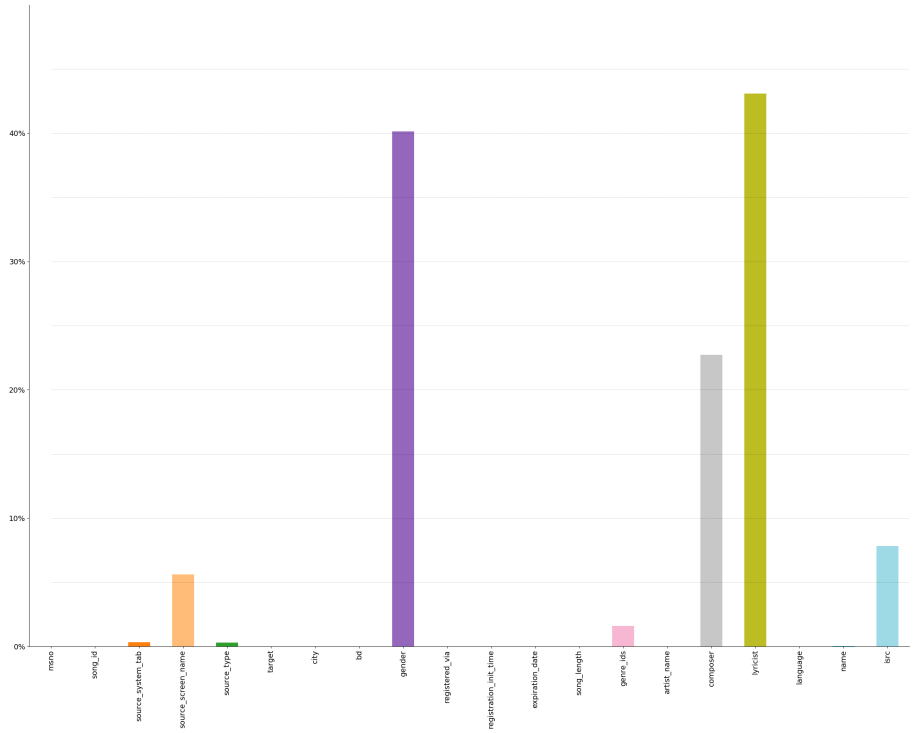

Figure 1: *En esta gráfica podemos ver porcentaje de valores perdidos en cada variable del conjunto final de datos.*

Como podemos ver en *Table 1* y *Figure 1* tenemos varias variables donde hay valores perdidos y que trataremos de diferentes maneras.

- Las variables *lyricist* y *gender* hemos decidido eliminarlas por tener un porcentaje muy alto de valores perdidos, más de un 40%.
- La variable *composer* también hemos decidido eliminarla. Aunque el porcentaje de valores perdidos en esta variable es un 22.71%, un valor alto pero que podriamos haber decidido imputar por ejemplo, cuenta con 81566 categorías diferentes. Teniendo en cuenta el tamaño de nuestro conjunto de datos, la complejidad que añade esta variable nos es intratable con el conocimiento y material del que disponemos.
- El resto de variables tienen menos de un 10% de valores perdidos. Podriamos haberlos imputado a partir del resto de datos, pero disponemos de un conjunto de datos lo bastante grande como para poder eliminar las muestras donde aparecen valores nulos sin problemas.

Después de tratar los valores perdidos en el conjunto de datos este consta de 17 variables y 6317407 muestras.

3.3 Tratamiento de outliers

La única variable que contiene outliers es *bd*, que representa la edad de un usuario, y tiene valores imposibles (como 0 años). Lo primero que hemos hecho ha sido eliminarlos, convirtiendo los valores menores a 16 y los mayores a 90 en valores nulos.

Habiamos pensado en imputar el valor de los outliers mediante KNN, pero al convertirlos en valores nulos hemos visto que un 39.68% de *bd* són outliers. Al ver esto hemos decidido que lo mejor era simplemente eliminar esta variable.

3.4 Creación de nuevas variables

La variable *ISRC* representa el International Standard Recording Code de la canción. Este codigo tiene un formato de 12 carácteres y cuatro partes de la forma CC-XXX-YY-NNNNN, donde:

CC	identificador del país del emisor del código ISRC.
XXX	codigo numérico identificador del emisor del código ISRC.
YY	dos últimos digitos del año en que el código ISRC fue asignado a la grabación.
NNNNN	identificador numérico de la grabación.

De esta variable crearemos una nueva variable con el año en que se registro el codigo de la grabación, ya que nos ha parecido un valor representativo para el conjunto, a la que llamaremos *song_year*. Para hacerlo cogeremos los dos dígitos YY del código ISRC, los valores mayores a 18 los convertimos en valores numéricos de la forma 19YY, el resto de la forma 20YY.

3.5 Selección de features

Hemos decidido eliminar varias de las variables. En esta sección explicaremos cuales y los motivos principales para hacerlo.

- *msno* y *song_id* han sido eliminadas por tratarse de simples identificadores, sin otro objetivo que identificar la muestra o una parte de ella. Se han usado para unir los diferentes conjuntos de datos de que disponiamos inicialmente, pero de cara al análisis no tienen utilidad alguna.
- El *isrc* es otro identificador como las variables mencionadas anteriormente. Sin embargo contiene más información que podría parecer útil.
  De esa información ya hemos elegido una parte que se encuentra en la variable *song_year*, la otra información que podemos encontrar en el ISRC o bien són identificadores (y no nos aportan información relevante) o puede ser erronea (el CC, en la pagina del concurso de kaggle, cuando explican la información de los diferentes ficheros, se avisa de que esta parte del identificador puede resultar confusa o incluso erronea).
  Por estas razones de esta variable no aprovechamos nada más de lo que pueda aportar y la descartamos.
- *artist_name* y *name* las rechazamos porque nos són computacionalmente intratables, tanto en R como en Python. La variable *artist_name* tiene 40582 clases diferentes y *name* otras 234144.
- *registered_via*, *registration_init_time* y *expiration_date* nos proporcionan información irrelevante para determinar si un usuario escuchara o no una canción repetidas veces.

Después del proceso de selección de features nos quedan 6317407 muestras y 9 variables, 8 de ellas categóricas y 2 numéricas:

Variable	Número de categorías
source_system_tab	8
source_screen_name	20
source_type	12
target	2
city	21
genre_ids	166
language	10
song_length	(no categórica)
song_year	(no categórica)

Table 3: *Número de categorías en las variables categóricas del conjunto de datos.*

3.6 Codificación de variables categóricas

La mayoría de variables categóricas las dejamos como estan, pero *genre_ids* la preprocesamos para hacer un poco menos complejo el conjunto de datos con el que trataremos luego. La variable *genre_ids* permite varias categorías diferentes por muestra (como se explica en la sección *Conjunto de datos disponible*). Hemos tratado los datos para que cada muestra tenga una sola categoría para esta variable.

Para hacerlo hemos calculado la freqüencia de cada categoría diferente y hemos tratado cada muestra para que conserve solo la más freqüente de sus categorías.

A parte de simplificar las muestras que tenian multiples categorías, después de tratar los datos el número de categorías diferentes en *genre_ids* se ha reducido de 166 a 148.

3.7 Estandarización

Escalar los datos es una parte importante del preproceso del conjunto de datos. Hay muy pocos métodos, como el Random Forest, donde estandarizar los datos no sea importante para encontrar mejores resultados.

Para hacerlo lo más freqüente es onormalizar o estandarizar los datos. Hemos decidido estandarizarlos ya que hemos encontrado que era la mejor opción para la mayoría de métodos, como regresión logística o SVM[?], en cambio no hemos encontrado ningun caso que nos fuera útil en que fuera mejor la normalización de los datos.

Para hacerlo hemos usado el método *sklearn.preprocessing.StandardScaler* con la opción *copy=False*, para evitar que copie datos innecesarios y de problemas de memoria.

3.8 Transformación de variables

4 Protocolo de re-muestreo

Una vez preprocesados los datos, a la hora de entrenar los métodos hemos separado en dos conjuntos el conjunto de datos con el objetivo de poder comparar bien los resultados, obteniendo:

Subconjunto	Proporción	Tamaño
Entrenamiento	0.7	4,422,185
Test	0.3	1,895,222
Total	1	6,317,407

Table 4: *Particines de entrenamiento y test del conjunto de datos.*

Con tal cantidad de datos podriamos haber tomado una partición de datos con más entrenamiento y menos test (90/10 o incluso más por ejemplo), pero hemos elejido particionar los datos en 70% entrenamiento y 30% test por dos razones:

- Obtener unos resultados más estables a la hora de comparar los distintos modelos.
- Reducir la cantidad de datos de entrenamiento a una cantidad ligeramente más tratable a nivel computacional. Ya que con tal cantidad de datos la potencia computacional y el tiempo de procesamiento necesarios són bastante considerables.

Además, para algunos modelos hemos tenido que calcular hiper-parámetros. Hemos usado *Random Search Cross-Validation* para ello.

Con tal cantidad de datos nos podríamos haber limitado a tomar una partición de validación para buscar los mejores hiper-parámetros, pero de esta forma teníamos que entrenar iterativamente cada modelo con los diferentes hiper-parámetros a comprobar y luego analizar los resultados obtenidos con la partición de validación. El problema en este caso es que cuando había que buscar varios hiper-parámetros para un mismo modelo no sabíamos paralelizar el proceso, y hacerlo

de esta forma era significativamente más lento que otras opciones ya implementadas en librerías como la que hemos usado, *sklearn*, como por ejemplo *sklearn.model_selection.GridSearchCV* o *sklearn.model_selection.RandomizedSearchCV*. De estas dos opciones hemos decidido usar *RandomizedSearchCV*.

El método *RandomizedSearchCV* de *sklearn* aplica *3-fold stratified cross-validation* a una muestra aleatoria de los hiper-parámetros que queremos obtener. A diferencia del *Grid-Search CV*, este método no prueba todas las posibles opciones, lo que implica una menor cantidad de entrenamientos del modelo y por tanto va más rápido. Esto implica un coste en cuanto a la accuracy respecto al *Grid-Search CV*, sin embargo este coste es negligible. Como se puede ver randomSearch[1] es especialmente efectivo con conjuntos de validación grandes, como es nuestro caso, y es la mejor opción que hemos encontrado.

5 Resultados de los métodos lineales

5.1 Linear Discriminant Analysis

Hemos elegido este modelo por ser rápido y sencillo. Al no tener hiper-parámetros ha bastado con entrenarlo con la partición de train al completo y probarlo con la partición de test. De esta forma nos ha dado los siguientes resultados:

Clase	Precision	Recall	F1-Score	Support
0	0.62	0.63	0.63	933345
1	0.64	0.63	0.63	961878
Average/Total	0.63	0.63	0.63	1895223

Table 5: *Particines de entrenamiento y test del conjunto de datos.*

Viendo los resultados obtenidos, es probable que los datos no cumplan la hipótesis de homocedasticidad.

5.2 Quadratic Discriminant Analysis

Hemos elegido este modelo por ser rápido y sencillo. Al no tener hiper-parámetros ha bastado con entrenarlo con la partición de train al completo y probarlo con la partición de test. De esta forma nos ha dado los siguientes resultados:

Clase	Precision	Recall	F1-Score	Support
0	0.62	0.54	0.57	933345
1	0.60	0.67	0.64	961878
Average/Total	0.61	0.61	0.61	1895223

Table 6: *Particines de entrenamiento y test del conjunto de datos.*

** Escribir motivos por los que va peor que el LDA

5.3 Regularized Discriminant Analysis

Hemos elegido este modelo por ser rápido y sencillo.

Hemos probado hiperparámetros dentro de una distribución uiniforme en 0,1

Hemos acabado eligiendo: 0.7010416905212281

De esta forma nos ha dado los siguientes resultados:

Clase	Precision	Recall	F1-Score	Support
0	0.57	0.60	0.59	933345
1	0.59	0.56	0.58	961878
Average/Total	0.58	0.58	0.58	1895223

Table 7: *Particines de entrenamiento y test del conjunto de datos.*

Va mejor que el LDA pero no significativamente.

6 Resultados de los métodos no lineales

6.1 Multi Layer Perceptron

Clase	Precision	Recall	F1-Score	Support
0	0.56	0.89	0.69	933345
1	0.75	0.31	0.44	961878
Average/Total	0.66	0.60	0.56	1895223

Table 8: *Particines de entrenamiento y test del conjunto de datos.*

Va mejor que el LDA pero no significativamente.

6.2 Random Forest

Clase	Precision	Recall	F1-Score	Support
0	0.67	0.66	0.66	933345
1	0.67	0.69	0.68	961878
Average/Total	0.67	0.67	0.67	1895223

Table 9: *Particines de entrenamiento y test del conjunto de datos.*

Va mejor que el LDA pero no significativamente.

7 Descripción y justificación del modelo escogido

8 Conclusiones

9 Bibliografía

References

[1] Sebastian Raschka. Python machine learning. *Python Machine Learning*, 2015.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Random Search for Hyper-Parameter Optimization*, 2012.