Grado en Ingenieria Informatica - UPC

Proyecto Aprendizaje Autónomo (APA)

KKBox's Music Recommendation Challenge

Raquel Leandra Pérez Arnal y 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 dataset, 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 Data exploration and Preprocessing

Partiendo de los datos iniciales (procedentes de los ficheros nombrados en el apartado 1.2) hemos generado un solo dataframe. Para hacerlo hemos unido la información de **members** y **train** por *msno*, y el resultado lo hemos unido a **songs** y **song_extra_info** por *song_id*. Al nuevo dataset con toda la información lo llamaremos **merged**.

Este dataframe, después de preprocesarlo, lo convertiremos en los dos conjuntos *train* y *test*, que guardaremos en los ficheros **clean_train.csv** y **clean_test.csv**.

3.1 Analísis de los datos

El nuevo dataset **merged** se compone de 20 columnas o variables y 7377418 filas diferentes. La siguiente tabla nos muestra algo de información que nos interesa sobre este:

Nombre	**Valores perdidos**	**Porcentage**	**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 dataset merged.

3.2 Tratamiento de valores perdidos

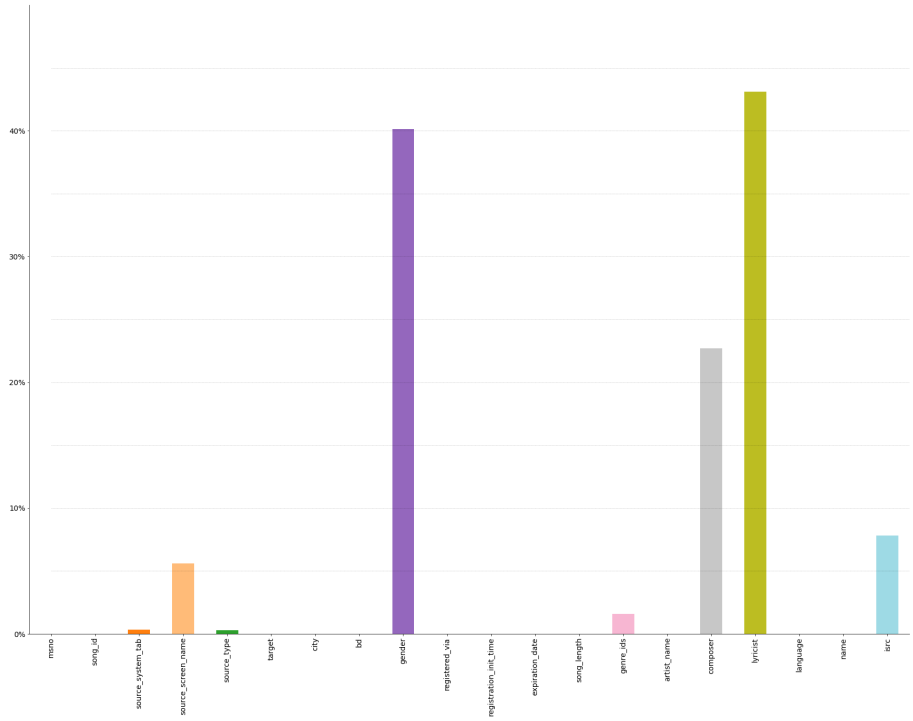

Figure 1: En esta gráfica podemos ver porcentage de valores perdidos en cada variable del conjunto final de datos.

Nombre	Porcentage
msno	0
song_id	0
source_system_tab	0.3368
source_screen_name	5.6226
source_type	0.2919
target	0
city	0
bd	0
gender	40.142
registered_via	0
registration_init_time	0
expiration_date	0
song_length	0.0015
genre_ids	1.605
artist_name	0.0015
composer	22.7139
lyricist	43.0882
language	0.0020
name	0.0197
isrc	7.8327

Table 2: Porcentage de valores perdidos de cada variable

Viendo esta información descartamos *lyricist* y *composer*

En el resto de variables eliminamos las muestras con missings salvo en *gender*.

Tratamiento de los valores perdidos en *gender*.

Los imputamos utilizando la clasificación que nos daría un knn.

3.3 Tratamiento de outliers

3.4 Tratamiento de valores incorrectos

3.5 Selección de features

Hay dos grandes motivos por los que hemos eliminado features:

- La feature en cuestión es un **identificador**, es decir, es una variable cuyo único objetivo es identificar la muestra o alguna de sus partes. Los hemos utilizado para unir los distintos conjuntos de datos, pero de cara al análisis no tienen utilidad.
- Algunas de las variables eran **computacionalmente intratatable**, esto se debe a que tienen tal cantidad de categorías, que al intentar procesarlas con los conocimientos y hardware que tenemos se vuelven intratables (tanto en python como en R).

Table 3: My caption

Nombre	Motivo
msno	Es solo un identificador
song_id	Es solo un identificadors
artist_name	Computacionalmente intratable
composer	Computacionalmente intratable
lyricist	Computacionalmente intratable
name	Computacionalmente intratable
isrc	Es solo un identificador

3.6 Codificación de variables categóricas

3.7 Creación de nuevas variables

3.8 Estandarización

3.9 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 conjuntos de datos, con objetivo de poder comparar bien los resultados. Obteniendo de esta forma:

Table 4: Partición de los datos

Subconjunto	**Proporción**	**Tamaño**
Entrenamiento	0.7	4,422,186
Test	0.3	1,895,223
Total	1	6,317,409

Hemos tomado un 30% de los datos para test con el objetivo de, por un lado obtener unos resultados más estables a la hora de comparar los distintos modelos, además de esta forma reducimos la cantidad de datos de entrenamiento a una cantidad ligeramente más tratable a nivel computacional. Puesto a que con tal cantidad de datos la potencia computacional y el tiempo de procesamiento es bastante considerable.

Además para algunos modelos hemos tenido que calcular hiper-parámetros. Para ello hemos utilizado *Random Search Cross-Validation*.

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, para 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, haciendo que de esta forma fuera significativamente más lento que otras opciones ya implementadas en la librería utilizada, como serían *sklearn.model_ selection.GridSearchCV sklearn.model_ selection.RandomizedSearchCV*. Entre estas dos opciones decidimos utilizar *RandomizedSearchCV*, por ser más rápida sin tener unos peores resultados por ello.

sklearn.model_selection.RandomizedSearchCV es un método de *sklearn* que aplica *3-fold stratified cross-validation* a una muestra aleatoria de los hiper-parámetros que queremos obtener. A diferencia con *Grid-Search CV*, este método no prueba todas las posibles opciones, lo que implica una menor cantidad de entrenamientos del modelo, y por tanto un coste muy inferior respecto al tiempo total, sin perder demasiada accuracy por el camino. Como se puede ver en [1] es especialmente efectivo con conjuntos de validación grandes, como es nuestro caso.

5 Resultados de los métodos lineales

logistic regression, multinomial regression (single-layer MLP), LDA, QDA, RDA, **Naive Bayes**, **nearest-neighbours**,**linear SVM**, quadratic SVM

6 Resultados de los métodos no lineales

one-hidden-layer MLP, the RBFNN, the SVM with RBF kernel, a Random Forest

7 Descripción y justificación del modelo escogido

8 Conclusiones

References

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