
Universidad de los Andes

Sergio Alberto Mora Pardo *

[github: sergiomora03](#)s.morap@uniandes.edu.co

Bogotá D.C.

Jahir Stevens Rodriguez Riveros

[github: jarorid](#)js.rodriguezr@uniandes.edu.co

Bogotá D.C.

Cindy Zulima Alzate Roman

[github: czalzate](#)c.alzate@uniandes.edu.co

Bogotá D.C.

13 de diciembre de 2020

1. Introducción

Una aplicación de música quiere actualizar su aplicación online para que genere recomendaciones a sus usuarios de nuevos artistas para escuchar. El sistema de recomendación debe tomar en cuenta las preferencias de cada usuario, con el fin de ofrecer recomendaciones automáticas y personalizadas.

Por ello se le pide, desarrollar un algoritmo de recomendación de artistas para cada usuario.

1.1. Definición del problema

Actualmente, la compañía desea actualizar la aplicación online para que genere recomendaciones a nuevos usuarios de artistas para escuchar. La información se encuentra en Celma [2010](#) en la página de [lastfm-360K](#).

1.2. Pregunta de Investigación

¿Cómo generar recomendaciones automáticas y personalizadas teniendo en cuenta las preferencias de cada usuario?

2. Metodología

Usaremos la librería [Surprise](#) es un [scikit](#)¹ de Python para construir y analizar sistemas de recomendación que traten con datos explícitos de acuerdo con Hug [2020](#). En ella se enmarcan diversos modelos para construir sistemas de recomendación inspirados² y basados en [skit-learn](#) una librería de Python para machine learning según Pedregosa y col. [2011](#).

La librería [Surprise](#) provee diversos algoritmos *listos para usar* como lo son: Métodos de vecindades o Factorización matricial. Así mismo, provee herramientas para evaluar, analizar y

* Administrador del repositorio [sergiomora03/deep-learning-intermediate](#)

¹SciKits (abreviatura de SciPy Toolkits) son paquetes complementarios para SciPy, alojados y desarrollados por separado e independientemente de la distribución principal de SciPy. Todos los SciKits tienen licencia aprobada por OSI.

²De esto se conversará más adelante, en la sección de presentación de modelos.

comparar el desempeño de los algoritmos. De acuerdo con Hug 2020 muchas de estas herramientas han sido basadas en *skit-learn* como lo son la validación cruzada y la búsqueda exhaustiva sobre un conjunto de parámetros.

Para el desarrollo y actualización del problema. Se hace necesario crear una API³. No obstante, el tiempo es limitado. Generando que nos concentremos en el desarrollo de los modelos y la mejora de los mismos. En este sentido, nuestra metodología es la siguiente:

1. Análisis Exploratorio de Datos.

- Número de usuario, número de artistas, análisis de Pareto sobre las reproducciones.

- Dinámica de usuarios con artistas.

- Análisis de la distribución de reproducciones por usuario.

- Distribución de reproducciones por artista.

- Análisis de distribución por percentiles.

- Análisis de asimetría y curtosis en la distribución de las reproducciones.

2. Benchmark de modelos.

- Selección de métricas.

- Selección de base de datos.

- Análisis de una muestra.

3. Selección del modelo.

- Train test split

4. Hiperparámetros del mejor modelo.

- Mejora de desempeño del modelo con mejor ajuste.

2.1. Identificación y Presentación de los modelos

Para la identificación de los modelos, se implementó un *benchmark* con en el cual se corrieron los siguientes modelos:

- **Algoritmos basados en factorización matricial**

SVD⁴ Este modelo es equivalente a la Factorización Matricial Probabilística por Salakhutdinov y Mnih 2007

Non-negative Matrix Factorization. Algoritmo de filtro colaborativo basado en una factorización matricial no negativa, equivalente al de Wang y Zhang 2013 pero en su forma no regularizada.

- **Pronósticos Aleatorias**

Normal Predictor. Supone una distribución normal en la distribución del conjunto de entrenamiento y pronóstica una clasificación aleatoria.

- **Baseline**

Baseline Only. Algoritmo que predice la estimación de la línea de base para un usuario y artículo determinados siguiendo a Koren, Bell y Volinsky 2009.

- **Co-Clustering**

Co-Clustering. Es un algoritmo de filtrado colaborativo basado en agrupación conjunta. Existe una implementación usada en la librería **Surprise** de George 2005.

- **Algoritmos basado en Vecindades**⁵.

- K-NN Básico.**

K-NN Baseline. Un algoritmo de filtrado colaborativo básico que tiene en cuenta una calificación de referencia.

³Buitinck y col. 2013, Algunas API basadas en experiencias con la librería de *skit-learn*.

⁴Popularizado por *Simon Funk* por el premio de Netflix

⁵Hug 2020, En la librería *Surprise* se construyó estos con algoritmos que se derivan directamente de un enfoque básico de vecinos más cercanos.

2.2. Estimación de Coeficientes

George y Merugu [2005](#) indica que La mayoría de los métodos de filtrado colaborativo existentes se basan en criterios de correlación, descomposición de valor singular (SVD) y factorización matricial no negativa. (NNMF) han demostrado proporcionar una alta precisión predicciones de calificaciones.

En la librería **Surprise** se enmarcan los modelos y algoritmos usados. En este sentido, se destina el apéndice para comentar sobre la estimación de los coeficientes⁶. Para la construcción de los coeficientes y comentarios, se destina el apéndice [A](#).

3. Resultados y Análisis

Antes de iniciar con el análisis de los modelos, es preciso indicar que se realizaron procesos sobre la base de datos. Los cuales, pueden ser consultados en el notebook **MicroProyecto2_MusicRecommender.ipynb** o de forma resumida en el apéndice [B](#).

Así mismo, el proceso de construcción de los modelos se hio con base en una muestra de 100.000 datos. Sin embargo, solo se mencionan resultados parciales de los modelos bajo los datos preprocesados y no sobre la muestra de 100.000 registros.

3.1. Análisis de los Modelos

3.1.1. Construcción de los modelos

Después del preprocesamiento de los datos, se construyo un benchamark de los modelos. Componiendo la siguiente estructura en los parámetros para los modelos:

```
bsl_options = {'method': 'als',
               'n_epochs': 20,
               'reg_u': 12,
               'reg_i': 5
              }

sim_options = {'name': 'cosine',
               'user_based': False
              }
```

Así mismo, dentro del Benchmark, se ejecuto una validación cruzada con la métrica: *RMSE*⁷. Este benchmark se corrió en un loop que contenía los siguientes algoritmos:

- SVD
- SVDpp
- NMF
- NormalPredictor
- BaselineOnly
- CoClustering

NOTA: Los algoritmos basados en vecindades no fueron incluidos en el benchmark, dado que requieren de un alto nivel de memoria.

Los algoritmos basados en vecindades, que fueron probados, pero no incluidos, son:

- KNNBaseline
- KNNBasic
- KNNWithMeans

⁶Por defecto se utiliza el RMSE como la métrica de error a minimizar para la predicción.

⁷Métrica por defecto de los modelos en la librería Surprise

- KNNWithZScore

Esto, nos arrojó los siguientes resultados:

Algoritmo	$test_rmse$	fit_{time}	$test_{time}$
SVD	446.24961	737.34729	147.18905
SVDpp	446.25344	4096.90589	2725.53427
BaselineOnly	446.25455	158.11567	7156.53749
CoClustering	446.29042	545.46788	7299.05748
NMF	446.46561	808.59790	5154.28341
NormalPredictor	446.95889	317.09584	7187.67404

Luego, usando los parámetros previamente definidos, se utilizó un `train_test_split()` dividiendo los datos entre 70 % para entrenamiento y 30 % para validación. Con esto, se definieron los siguientes algoritmos:

- `SVD_algo = SVD(random_state=0)`
- `Non_negative_algo = NMF()`
- `NormalPredictor_algo = NormalPredictor()`
- `Base_algo = BaselineOnly(bsl_options=bsl_options)`
- `CoClustering_algo = CoClustering(random_state=0)`
- `KNNBaseline_algo = KNNBaseline(bsl_options=bsl_options, sim_options=sim_options)`
- `KNNBasic_algo = KNNBasic(bsl_options=bsl_options, sim_options=sim_options)`

Si se evidencia, se puede notar como se logró incluir dos de los cuatro algoritmos basados en vecindades de la librería **Surprise**. Los incluidos son *KNNBaseline* y *KNNBasic*.

NOTA: Se usaron los parámetros previamente definidos *bsl_options* y *sim_options* para correr estos modelos. Esto con el fin de evaluar su desempeño bajo los mismos parámetros (en lo posible).

Esto nos arrojó los siguientes resultados:

Algoritmo	RMSE	FCP	TIME
SVD	447.66120	0.00000	0 days 00:36:03.584442
KNNBaseline	447.66141	0.13813	0 days 00:26:27.350901
CoClustering	447.66123	0.15941	0 days 00:28:43.812763
Base	447.67855	0.18636	0 days 00:14:37.155037
KNNBasic	447.69664	0.18973	0 days 00:27:55.763893
NormalPredictor	448.36007	0.23288	0 days 00:14:13.305294
Non_negative	447.87067	0.34954	0 days 00:44:41.959739

Para este, si se logró incluir algoritmos basados en vecindades. Dada la partición, se incluyó *KNN Baseline* y también *KNN Basic*.

A continuación, se construyó una rutina para calibrar los hiperparámetros del modelo con mejor ajuste, pueden evidenciarse en el algoritmo 1. Escogiendo la métrica *FCP*; en el notebbok se podrá

evidenciar como se utilizó para la muestra de 100.000 y para la base de datos preprocesada.

Algorithm 1: Calibración de hiperparámetros

Input: $data \leftarrow Train, Test$; Un conjunto N
 $N = \{n_factors, n_epochs, biased, reg_pu, reg_qi, reg_bu, reg_bi, lr_bu, lr_bi\}$;
 $N_i = [var_1, var_2, \dots, var_j] \ j \in N_i$; $N_i \leftarrow list() \ \forall \ i \in N$; $algoritmo \leftarrow surprise.algo()$;
Output: Una lista B donde, $B_i \leftarrow best \ parameter$; $B_i \in N_{ij}$
Result: $list()$: Parámetros que maximizan el FCP.
 $current \leftarrow \emptyset$;
 $C \leftarrow \emptyset$;
for i **to** N **do**
 for j **to** N_i **do**
 $mod \leftarrow algoritmo$;
 $mod.args = N_{ij}$;
 $preds \leftarrow mod.fit.test$;
 $accuracy \leftarrow preds.FCP$;
 if $accuracy \geq current$ **then**
 $B \leftarrow [N_{ij} \ \forall \ i \in N]$;
 $current \leftarrow accuracy$
 else
 $next()$;
 end
 $C \leftarrow C + \{c_i\}$;
 end
end
return C, B ;

Algoritmo 1 contiene C como contador de iteraciones para representar el avance del loop.

Así mismo, se observará la preferencia sobre el algoritmo **Non-negative matrix factorization** debido a sus propiedades y desempeño en la métrica **FCP**. No obstante, se resalta que el algoritmo **BaselineOnly** tiene un mejor ajuste sobre la muestra que sobre la base de datos preprocesada.

En este sentido, la calibración de hiperparámetros fue sobre el algoritmo **Non-negative matrix factorization** dado el desempeño. Sin embargo, este mismo algoritmo presentó un alto nivel de demanda de memoria. Siendo el más demorado en ejecutarse.

3.2. Análisis de los Resultados

Luego de evaluar varias configuraciones para los algoritmos evaluados, se obtuvieron los resultados expuestos en la tabla, que reflejan los valores de las métricas para una muestra aleatoria de 100.000 registros, donde se definió en 30 % el tamaño para la muestra de prueba (test). Estos resultados permiten identificar al modelo BaselineOnly con los parámetros expuestos en el numeral 3.1.1., como aquel con mejor desempeño al presentar el menor valor en el RMSE, adicionalmente, no requiere un esfuerzo computacional importante al ejecutarlo.

ALGO.	RMSE	FCP	TIME
SVD	875.59809	0.48837	0 days 00:00:11.982058
KNN	875.59780	0.65664	0 days 00:00:33.900963
Base	875.59763	0.65664	0 days 00:00:01.985999

También se evidencia que la evaluación por k-vecinos cercanos es aquella con mayor capacidad de ejecución computacional necesita.

El algoritmo BaselineOnly requiere el ajuste de 4 parámetros⁸, a saber:

- **method**: Puede tomar los valores ALS (Alternating Least Squares) o SGD (Stochastic Gradient Descent)
- **n_epochs**: Corresponde al número de iteraciones del procedimiento ALS
- **reg_u**: Parámetro de regularización para usuarios
- **reg_i**: Parámetro de regularización para ítems

⁸https://surprise.readthedocs.io/en/stable/prediction_algorithms.html#baseline-estimates-configuration

4. Conclusiones

1. Hacer uso del algoritmo BaselineOnly para la recomendación a generar para un nuevo usuario.
2. Buscar ampliar la base de datos con variables que enriquezcan la característica del conteo de reproducciones.
3. Ejecutar el algoritmo con una muestra de usuarios, teniendo en cuenta los artistas y reproducciones por usuario.

Referencias

- [1] Lars Buitinck y col. “API design for machine learning software: experiences from the scikit-learn project”. En: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, págs. 108-122.
- [2] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.
- [3] Thomas George. “A scalable collaborative filtering framework based on co-clustering”. En: *Fifth IEEE International Conference on Data Mining*. 2005, págs. 625-628.
- [4] Thomas George y Srujana Merugu. “A Scalable Collaborative Filtering Framework based on Co-clustering”. En: *Department of Computer Science, Texas A M University* (2005). URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.6458&rep=rep1&type=pdf>.
- [5] Nicolas Hug. “Surprise: A Python library for recommender systems”. En: *Journal of Open Source Software* 5.52 (2020), pág. 2174. DOI: [10.21105/joss.02174](https://doi.org/10.21105/joss.02174). URL: <https://doi.org/10.21105/joss.02174>.
- [6] Y. Koren, R. Bell y C. Volinsky. “Matrix Factorization Techniques for Recommender Systems”. En: *Computer* 42.8 (2009), págs. 30-37. DOI: [10.1109/MC.2009.263](https://doi.org/10.1109/MC.2009.263).
- [7] F. Pedregosa y col. “Scikit-learn: Machine Learning in Python”. En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.
- [8] Ruslan Salakhutdinov y Andriy Mnih. “Probabilistic Matrix Factorization”. En: *Department of Computer Science, University of Toronto* (2007). URL: <https://papers.nips.cc/paper/2007/file/d7322ed717dedf1eb4e6e52a37ea7bcd-Paper.pdf>.
- [9] Yu-Xiong Wang y Yu-Jin Zhang. “Nonnegative Matrix Factorization: A Comprehensive Review”. En: *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING* 25.6 (2013). URL: <http://oa.ee.tsinghua.edu.cn/~zhangyujin/Download-Paper/E224%3DTKDE-13.pdf>.

A. Algoritmos en Surprise

A.1. Notaciones

R : the set of all ratings.

R_{train} , R_{test} and \hat{R} denote the training set, the test set, and the set of predicted ratings.

U : the set of all users. u and v denotes users.

I : the set of all items. i and j denotes items.

U_i : the set of all users that have rated item i .

U_{ij} : the set of all users that have rated both items i and j .

I_u : the set of all items rated by user u .

I_{uv} : the set of all items rated by both users u and v .

r_{ui} : the true rating of user u for item i .

\hat{r}_{ui} : the estimated rating of user u for item i .

b_{ui} : the baseline rating of user u for item i .

μ : the mean of all ratings.

μ_u : the mean of all ratings given by user u .

μ_i : the mean of all ratings given to item i .

σ_u : the standard deviation of all ratings given by user u .

σ_i : the standard deviation of all ratings given to item i .

$N_i^k(u)$: the k nearest neighbors of user u that have rated item i . This set is computed using a [similarity metric](#).

$N_u^k(i)$: the k nearest neighbors of item i that are rated by user u . This set is computed using a [similarity metric](#).

NOTA: Estas notaciones son extridas directamente de la librería **Surprise** al igual que las ecuaciones descritas a continuación.

A.2. Algoritmos base:

A.2.1. NormalPredictor

Este algoritmo predice un rating aleatorio asumiendo que la muestra de entrenamiento proviene de una distribución Normal:

$$\hat{r}_{ui} \sim Normal(\hat{\mu}, \hat{\sigma}) \quad (1)$$

donde,

$$\hat{\mu} = \frac{1}{|R_{entrenamiento}|} \sum_{r_{ui} \in R_{entrenamiento}} r_{ui} \quad (2)$$

$$\hat{\sigma} = \sqrt{\sum_{r_{ui} \in R_{entrenamiento}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{entrenamiento}|}} \quad (3)$$

A.2.2. BaselineOnly

El algoritmo obtiene su estimación a partir del rating medio y las desviaciones observadas para la película i y el usuario u :

$$\hat{r}_{ui} = \mu + b_i + b_u \quad (4)$$

donde μ es el rating promedio de los datos de entrenamiento, b_i que es el rating promedio del item i menos μ y b_u que es el rating promedio del usuario u menos μ .

A.3. Algoritmos de vecindades:

A.3.1. KNNBasic

El **KNNBasic** es un modelo que estima los ratings de acuerdo con los k vecinos más cercanos, ya sea por usuario o por ítem, de acuerdo con:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)} \quad (5)$$

ó

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)} \quad (6)$$

donde sim es la función de [similitud](#)

A.3.2. KNNWithMeans

Este algoritmo modifica el **KNNBasic** tomando además en cuenta los ratings promedios de los usuarios:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)} \quad (7)$$

ó

$$\hat{r}_{ui} = \mu_i + \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot (r_{uj} - \mu_j)}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)} \quad (8)$$

A.3.3. KNNWithZScore

Este algoritmo toma en cuenta las similitudes y los ratings estandarizados:

$$\hat{r}_{ui} = \mu_u + \sigma_u \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - \mu_v) / \sigma_v}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)} \quad (9)$$

ó

$$\hat{r}_{ui} = \mu_i + \sigma_i \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot (r_{uj} - \mu_j) / \sigma_j}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)} \quad (10)$$

A.3.4. KNNBaseline

Este algoritmo toma en cuenta el rating medio y las desviaciones observadas para la película i y el usuario u :

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)} \quad (11)$$

ó

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot (r_{uj} - b_{uj})}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)} \quad (12)$$

A.4. Algoritmos de factores latentes:

A.4.1. SVD

Este algoritmo corresponde con la factorización de la matriz de ratings (visto en la actividad anterior):

$$\hat{r}_{ui} = q_i' p_u + \mu + b_i + b_u \quad (13)$$

A.4.2. SVDpp

Este algoritmo extiende el SVD al tomar en cuenta los ratings implícitos, ó la cantidad de **feedback** implícito:

$$\hat{r}_{ui} = \mu + b_i + b_u + q'_i \left(p_u + |I_u|^{-1/2} \sum_{j \in I_u} y_j \right) \quad (14)$$

donde y_j es un valor binario que captura el hecho de que el usuario u haya calificado o revelado su rating para j (sin importar el valor del rating).

A.4.3. NMF

Este algoritmo corresponde con la factorización no-negativa de la matriz de ratings, y sigue la misma formulación del SVD. Solo que se garantiza que los factores sean no-negativos.

A.5. Algoritmo de clustering:

A.5.1. Co-clustering

En este algoritmo, los usuarios y los items son asignados a los clusters C_u , C_i y C_{ui} :

$$\hat{r}_{ui} = \bar{C}_{ui} + (\mu_u - \bar{C}_u) + (\mu_i - \bar{C}_i) \quad (15)$$

donde $\bar{C}_{ui}, \bar{C}_u, \bar{C}_i$ son respectivamente los rating promedio de los clusters C_{ui}, C_u y C_i . Los clusters se asignan de acuerdo con K-medias.

B. Análisis Exploratorio de Datos

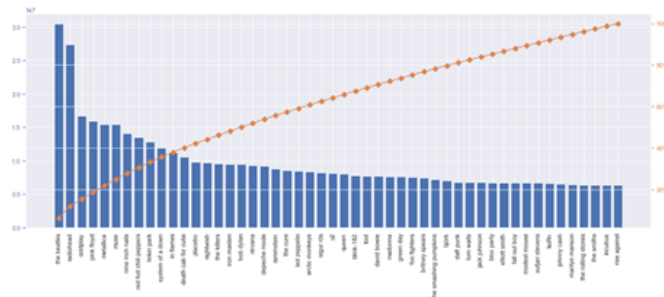


Figura 1: Pareto por artistas.

Después de revisar el Pareto en la figura 1, se evidencia que 35 artistas representan el 80 % de las reproducciones.

plays	
count	292363.00000
mean	12907.02249
std	185981.63087
min	1.00000
25%	53.00000
50%	208.00000
75%	1048.00000
max	30466827.00000

Figura 2: Distribución de reproducciones por artista.

Como vemos en la figura 2, existe una varianza de 18.598 reproducciones con un promedio de 12.907 sin embargo, se evidencia que la media esta sobre 208 reproducciones. Es decir, que existe un sesgo revisando la varianza, dado que los datos tienen tanta 'diversidad'. Ahora, revisaremos ¿cuántas reproducciones tiene cada usuario? con el fin de verificar esto.

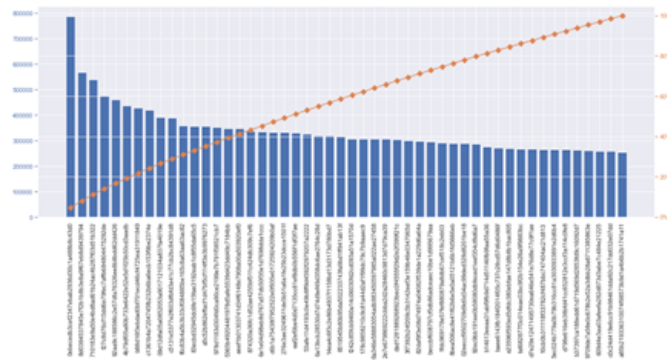


Figura 3: Pareto por usuario.

Con el Pareto de la figura 3, evidenciamos que 37 usuarios concentran el 80 % de las reproducciones.

Vemos que de los 358.868 existe uno que ha reproducido 787.884 canciones, que representa el 20 % de los datos. Esto nos lleva a preguntarnos ¿Cuánto tiempo le habría tomado escuchar esas canciones? En [Spotify](#) se cuenta una canción como reproducida después de que el usuario la escucha por más de 30 segundos. Si suponemos que escucho todas las canciones al menos 30 segundos, este usuario tuvo que haber invertido en tiempo:

Días:

273 días reproduciendo sin parar.

820 días reproduciendo mínimo 8 horas por día.

Años:

0.7583333333333333 años reproduciendo sin parar.

2 años reproduciendo mínimo 8 horas por día.

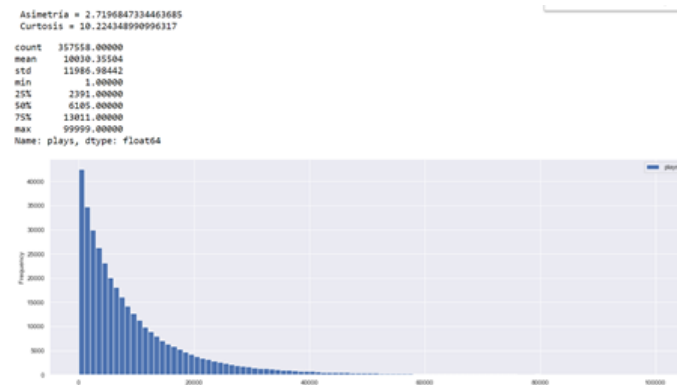


Figura 4: Distribución de reproducciones por usuario.

Vemos en la figura 4 que al concentrarse en menos de 100.000 reproducciones por usuario disminuye considerablemente la curtosis, incluso más que en la concentración de más de 100.000 reproducciones por usuario. Así mismo, vemos que la asimetría también disminuye considerablemente. De manera que, podemos concentrarnos mejor en esta distribución de reproducciones. No obstante, intentaremos disminuir a 60.000 reproducciones por usuario y observaremos si la distribución disminuye la asimetría y la curtosis. Dado que aún continua siendo leptocúrtica.