



Facultad de Estudios Superiores: Acatlán.

Diplomado en Ciencia de Datos

Proyecto:  
*La magia del cine (o de los datos)*

Valencia Martínez Jesús Adolfo

# TABLA DE CONTENIDO

<b>CONTEXTO.....</b>	<b>3</b>
<b>LA FUENTE .....</b>	<b>3</b>
<b>LOS DATOS.....</b>	<b>4</b>
VALORES AUSENTES.....	5
SEGMENTACIÓN DE VARIABLES .....	6
VARIABLES CATEGÓRICAS.....	7
VARIABLES NUMÉRICAS .....	8
VARIABLES DE TIPO CARÁCTER .....	13
<b>MODELAJE SUPERVISADO .....</b>	<b>16</b>
OBJETIVO .....	16
METODOLOGÍA .....	17
RESULTADOS .....	24
ELECCIÓN DE MODELO.....	31
REDES NEURONALES .....	31
CONCLUSIONES .....	34
<b>MODELAJE NO SUPERVISADO .....</b>	<b>34</b>
OBJETIVO .....	34
METODOLOGÍA .....	34
RESULTADOS .....	38
ELECCIÓN DE MODELO.....	42
PRUEBAS ESTADÍSTICAS.....	42
CONCLUSIONES .....	44
PERFILAMIENTO .....	45
<b>RESUMEN .....</b>	<b>47</b>
<b>COMENTARIOS FINALES.....</b>	<b>48</b>
<b>APÉNDICE .....</b>	<b>48</b>

# La magia del cine (o de los datos)

## Contexto

En el mundo, la información crece de manera constante y, con el tiempo, se han ido generando metodologías con el fin de convertir esta información en valiosos activos para su explotación. Hablando del arte, y más específicamente, en el llamado séptimo arte, este fenómeno no sería la excepción, la cantidad de producciones emerge y se va anexando a la base gigantesca de películas creadas por la humanidad.

El presente reporte expone la aplicación de diversas metodologías de aprendizaje de máquina para la obtención de información de valor haciendo uso de los datos generados por la industria del cine.

## La fuente

Los datos de insumo corresponden a una extracción dentro de un sitio web llamado Kaggle. Estos datos se actualizan diariamente y detallan la información de las películas que se encuentran en la página de reseñas de internet llamada TMDb. Esta página cuenta no solo con la información de películas, también de series, capítulos de series y aunque la misma página ofrece una API para realizar extracción de información, la base que se encuentra en Kaggle contiene únicamente la información de películas junto con otra información relevante, por lo cual, se opta por elegir esta última base como insumo de trabajo.

# Los datos

Los datos cuentan con la información de un total de 750,318 películas y 20 características:

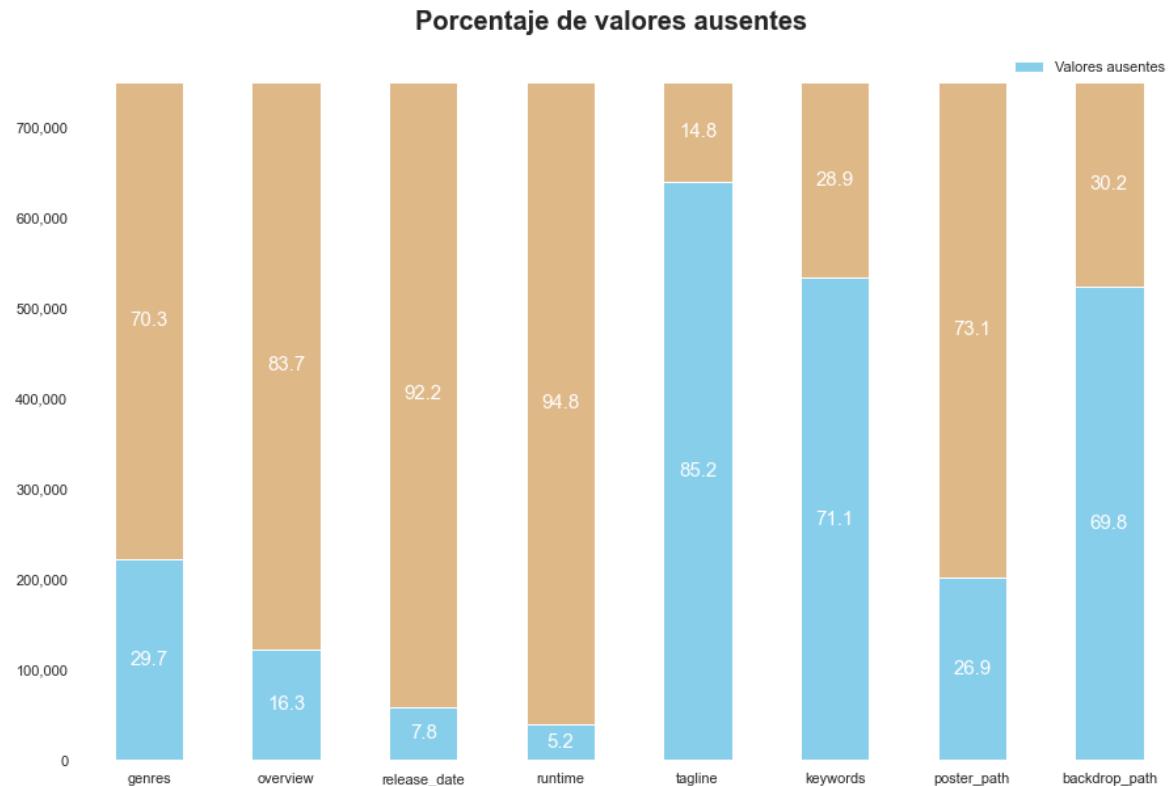
<i>Característica</i>	<i>Descripción</i>
<i>id</i>	Identificador único en la página de TMDb
<i>title</i>	Nombre de la película
<i>genres</i>	Géneros relacionados
<i>original_language</i>	Idioma original
<i>overview</i>	Resumen o sinopsis
<i>popularity</i>	Métrica de TMDb para medir la popularidad dentro del sitio web
<i>production_companies</i>	Compañías productoras
<i>release_date</i>	Fecha de estreno
<i>budget</i>	Presupuesto de producción
<i>revenue</i>	Ganancia reportada
<i>runtime</i>	Duración en minutos
<i>status</i>	Estado actual de la película (estrenada o no)
<i>tagline</i>	Breve frase alusiva a la trama de la película
<i>vote_average</i>	Calificación promedio que los usuarios le dan a la película
<i>vote_count</i>	Cantidad de calificaciones proporcionadas
<i>credits</i>	Nombre de los productores principales
<i>keywords</i>	Palabras clave sobre la película
<i>poster_path</i>	URL para obtener el poster de la película
<i>backdrop_path</i>	URL para obtener el backdrop de la película
<i>recommendations</i>	Recomendaciones a otras películas relacionadas

De las características anteriores, hay algunas que, para propósitos de aplicación, no son relevantes en su uso, por lo que fueron descartadas. Las variables que se descartaron fueron:

- 1** title
- 2** production\_companies
- 3** credits
- 4** recommendations

## Valores ausentes

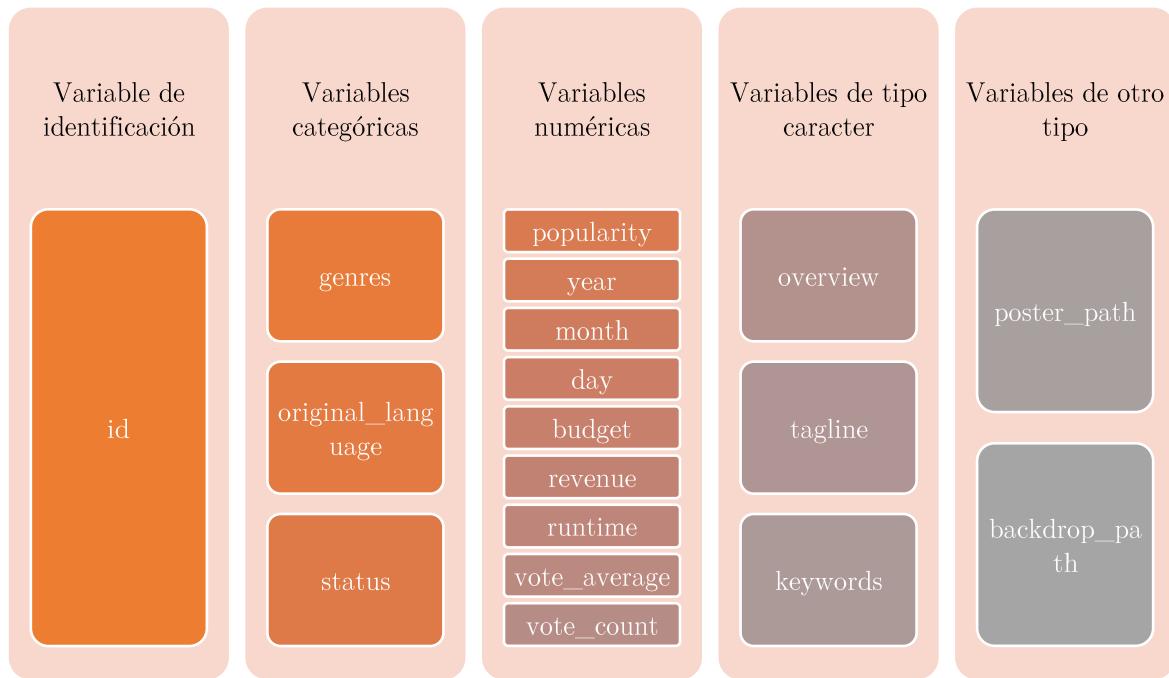
Realizando la exploración de nuestros datos, se pudo observar que varias películas no cuentan con información de algunas características. Las características que no cuentan con algunos registros se presentan en el siguiente gráfico, donde, a su vez, se detalla el porcentaje de valores ausentes respecto al total de registros:



Del gráfico anterior, podemos observar que existen variables que cuentan con una gran cantidad de valores ausentes, sin embargo, el tratamiento que se optó por realizar deriva de la naturaleza de su uso. En consecuencia, el tratamiento de estos valores no se realiza de manera general dentro de la primera exploración. Posteriormente, se describe el tratamiento de los mismos en cada caso de uso.

## Segmentación de variables

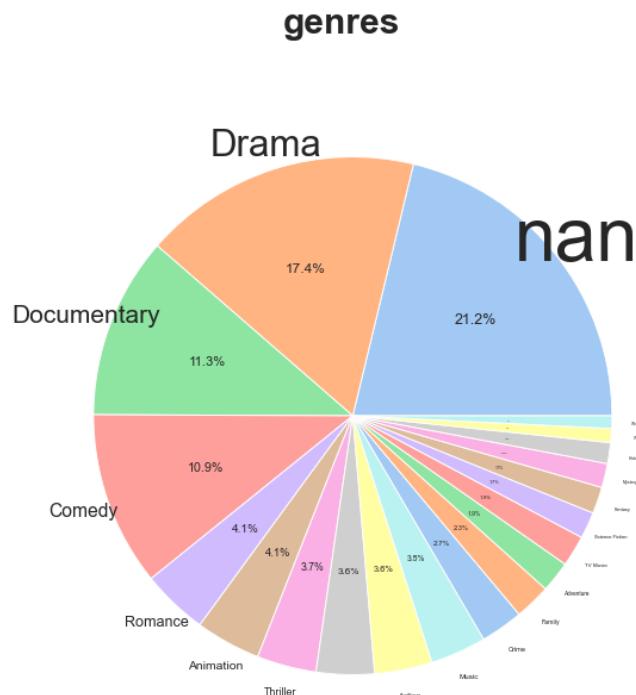
Una vez explicado el punto referente a los valores ausentes, las variables de nuestra base de datos se segmentaron de acuerdo a su tipo. La clasificación es la siguiente:



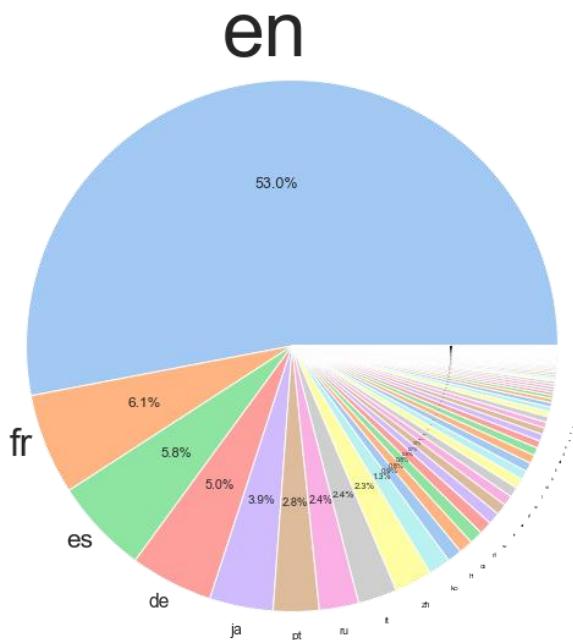
En los gráficos a continuación expuestos, se presenta un breve resumen de las características descritas

## Variables categóricas

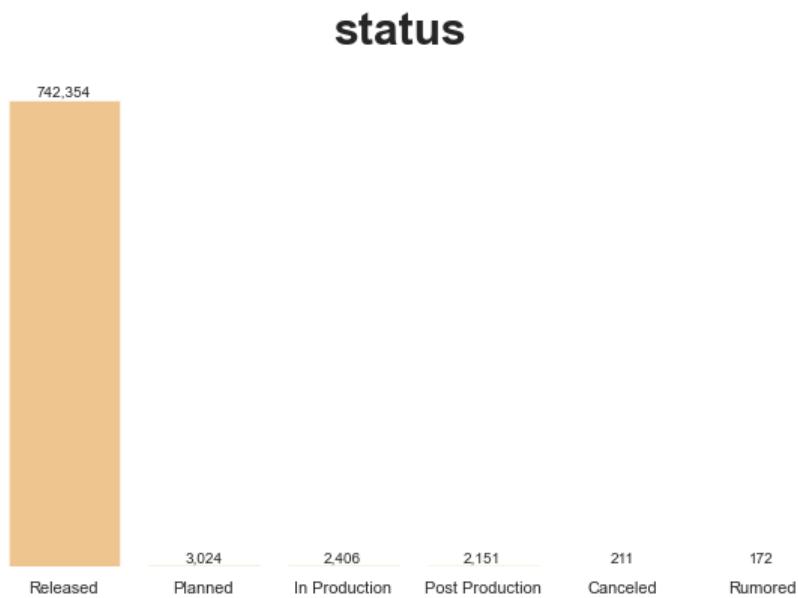
Analizando la gráfica de la derecha, la cantidad de valores ausentes juega un papel importante en la cantidad de géneros que se encuentran en nuestra información. Además, existen diversos géneros que si bien, existen en la base, su representación estadística es muy baja.



**original\_language**



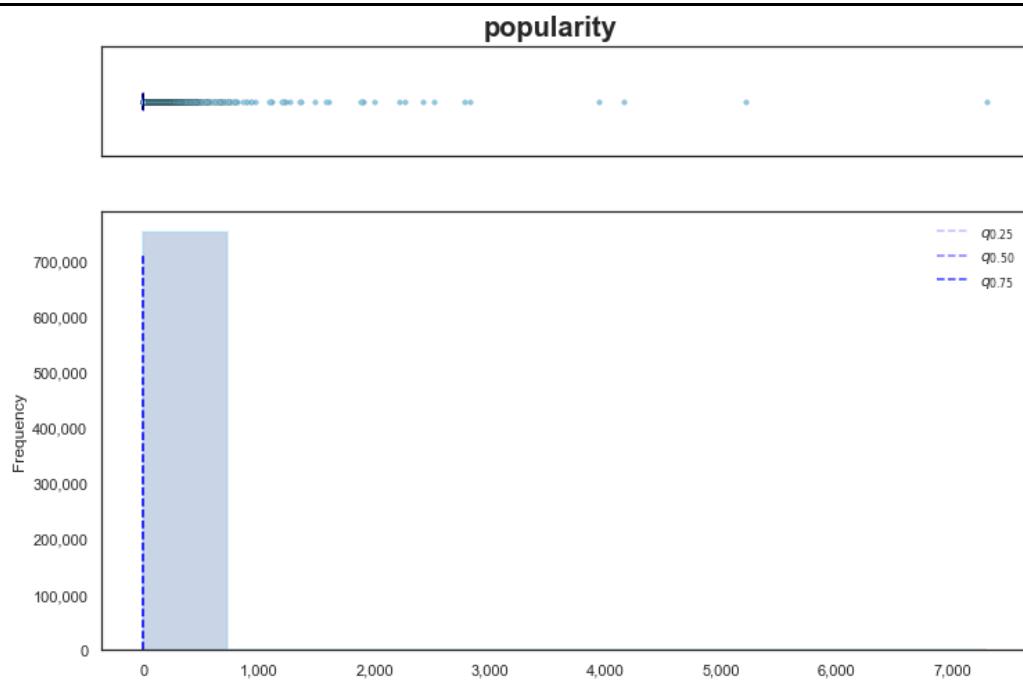
Para el caso del idioma original de las películas, aunque la variable no cuenta con valores ausentes, se puede apreciar la carga de información al idioma inglés, además de que varios idiomas no cuentan con una gran representación estadística.

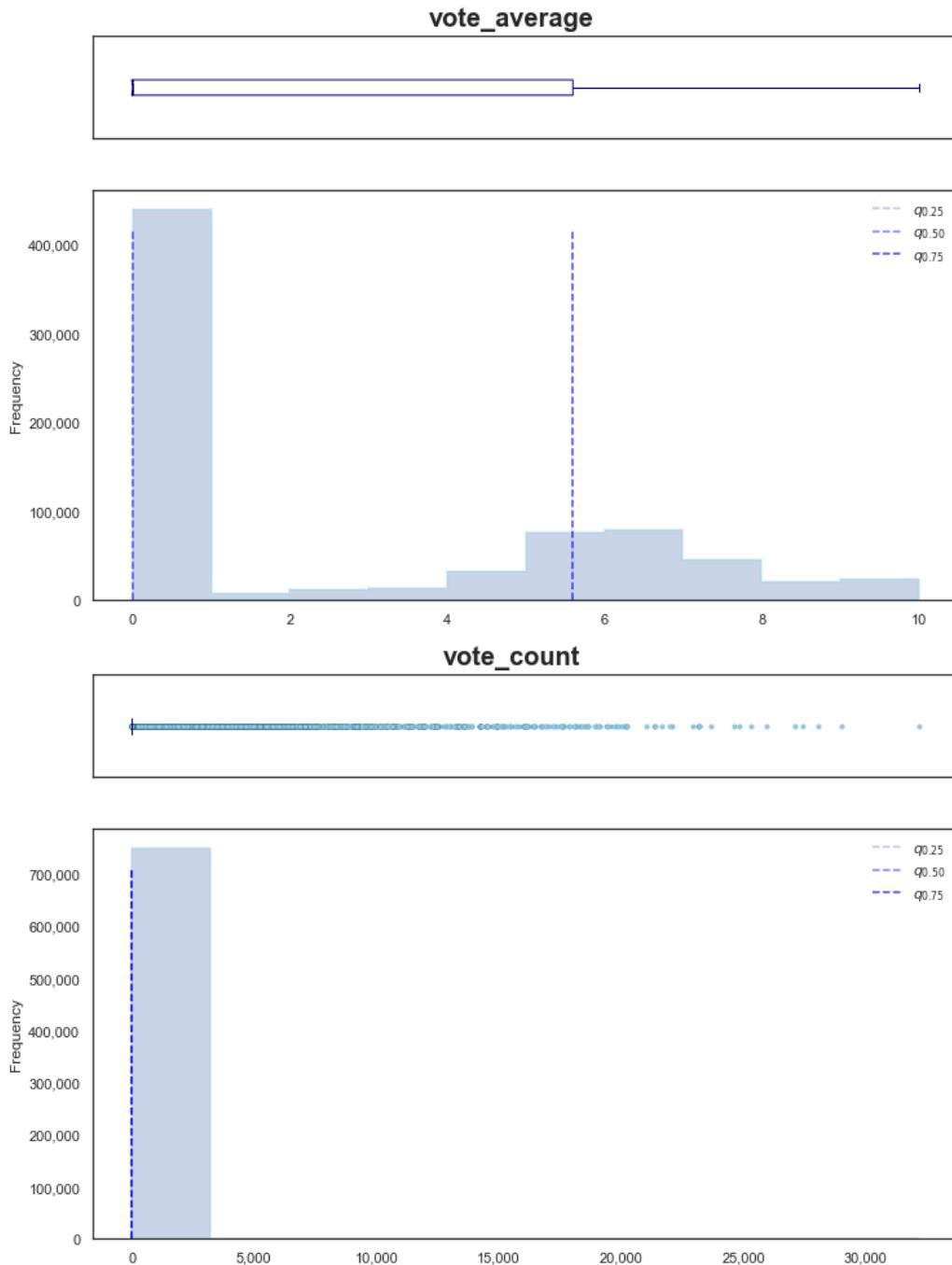


Finalmente, para la variable del estado de la película, es bastante evidente que la información prácticamente se basa en las películas que ya han sido estrenadas

## Variables numéricas

*Referentes a opiniones del usuario*

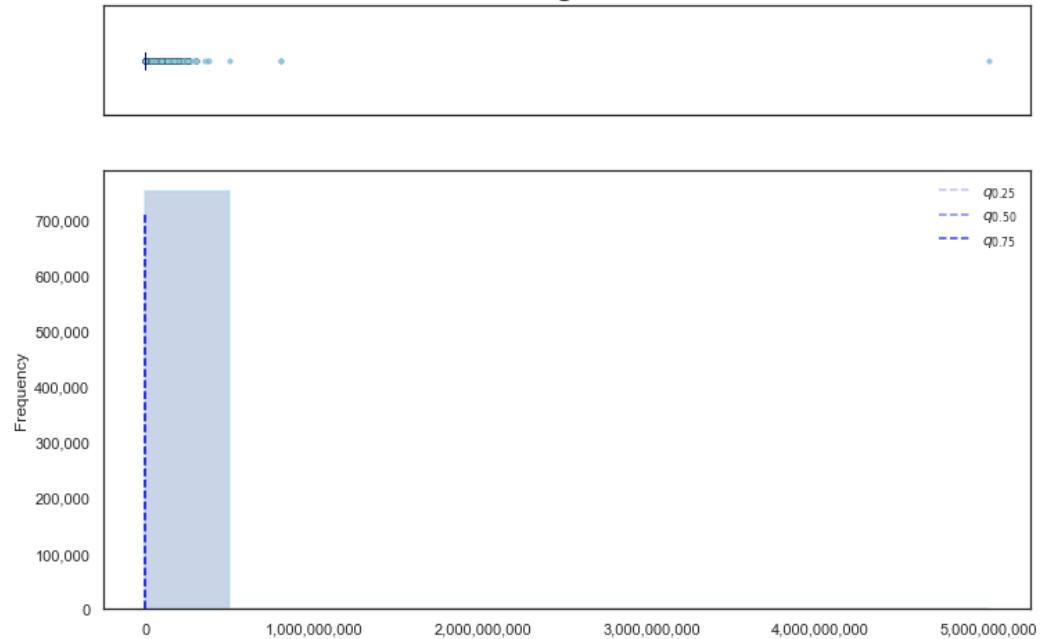




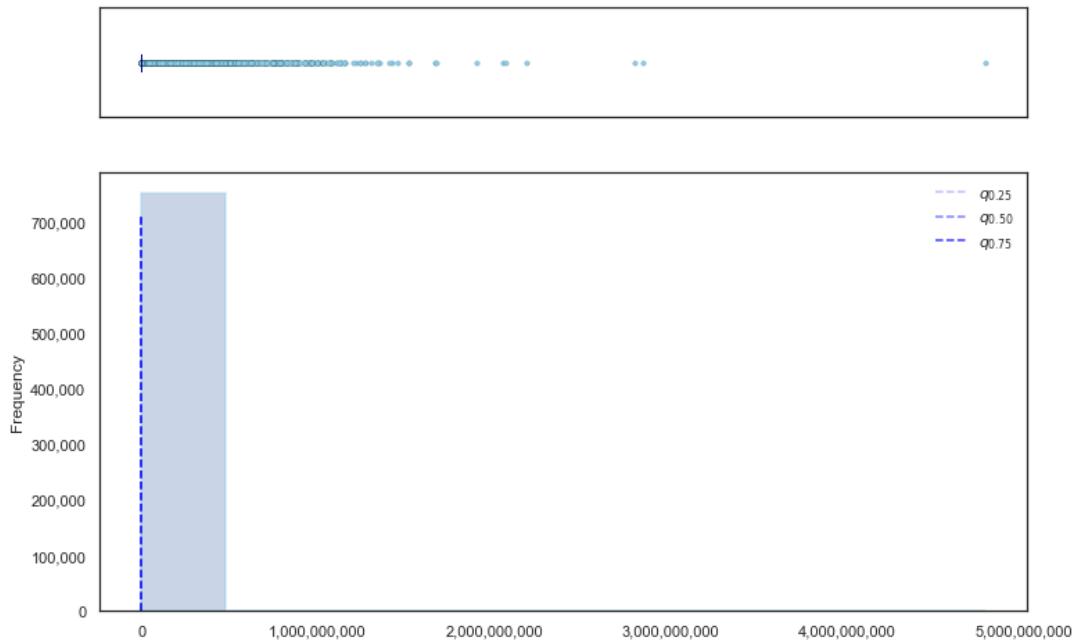
Las gráficas anteriores exhiben una carga de la distribución al valor 0. Esto es consecuencia a que existe una cantidad considerable de películas que, si bien, no cuentan con valores ausentes, los datos se registran como 0 y, por lo tanto, esto influye en la calificación promedio y en la métrica de popularidad.

## *Referentes cifras monetarias*

**budget**

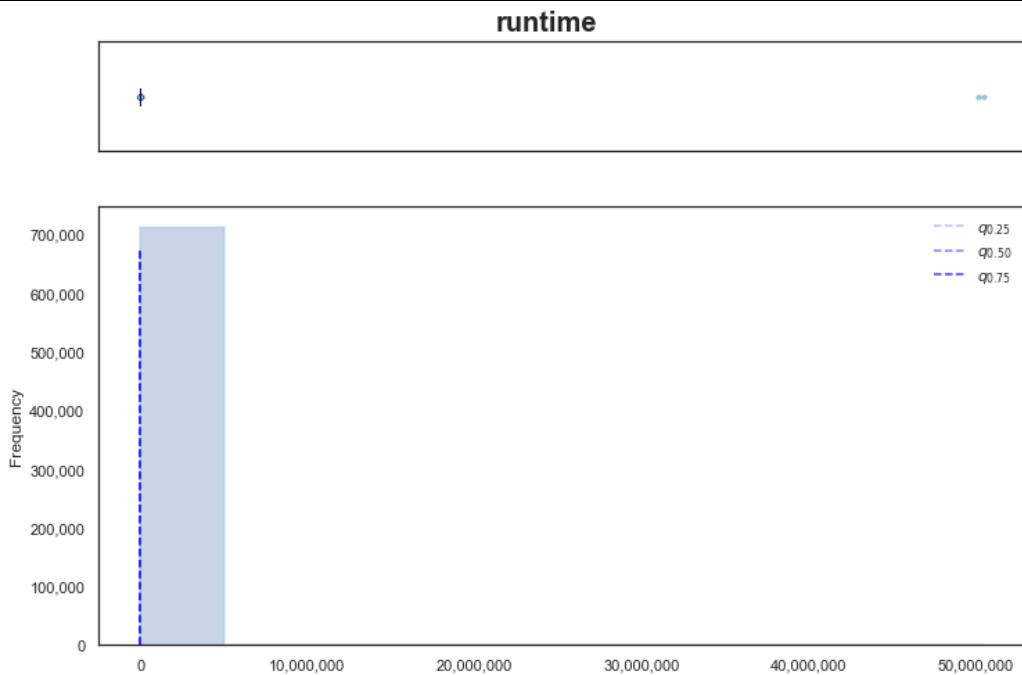


**revenue**



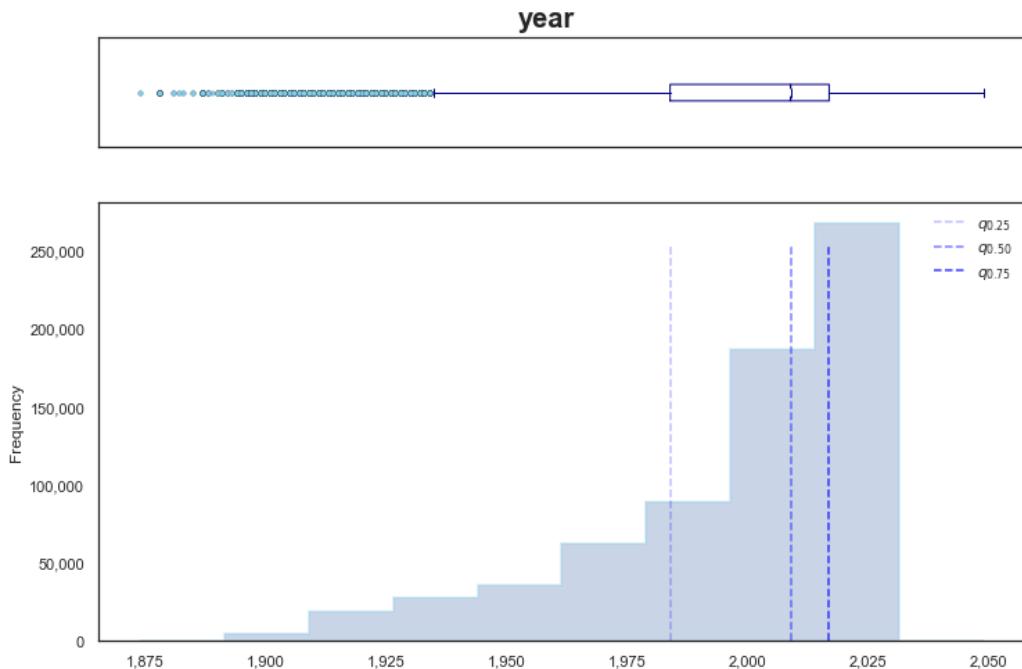
El comportamiento en estas variables es similar a las anteriores, si bien, no hay valores ausentes, su carga hacia el 0 se debe a que se registra la información como 0 al no contar con información

## *Referentes información de la película*

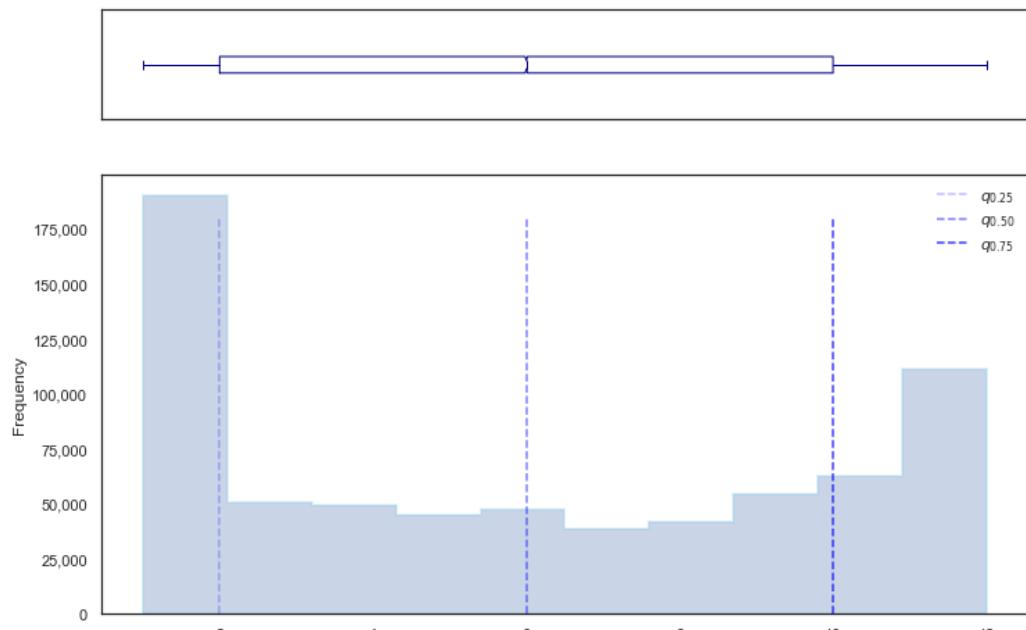


En este gráfico se puede apreciar la evidencia de un claro valor atípico o bien, una captura errónea que provoca un sesgo enorme en la distribución.

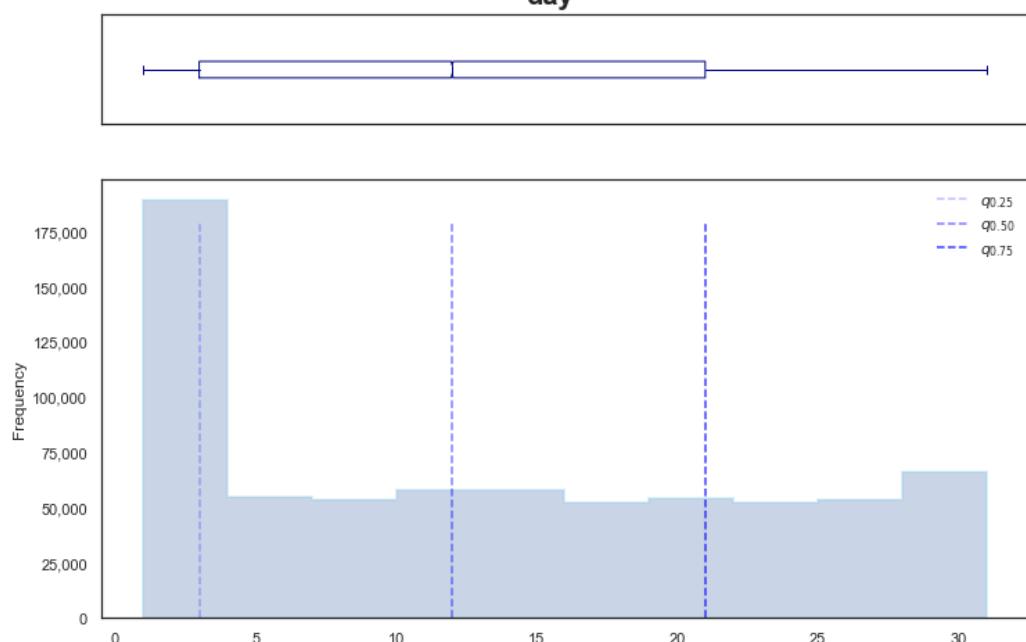
Las siguientes, son las distribuciones de la fecha de lanzamiento de las películas en la base:



**month**



**day**

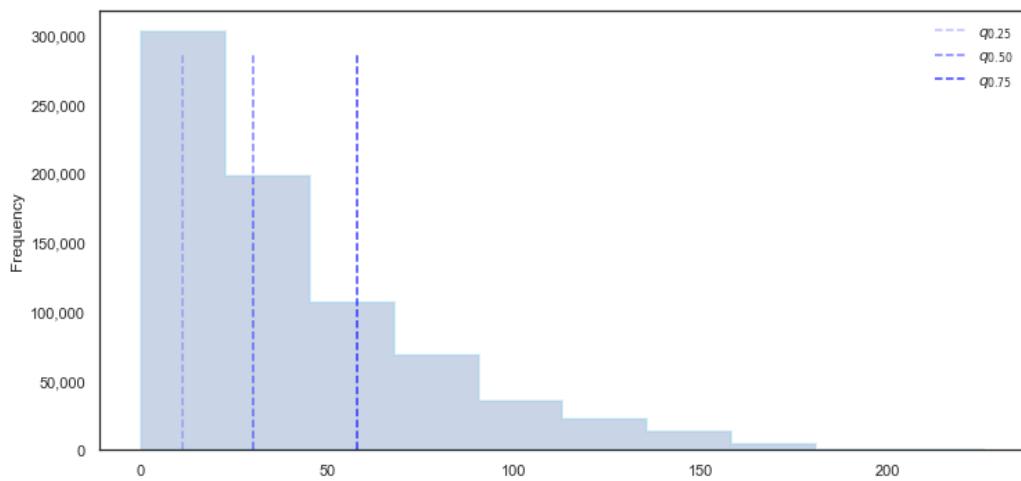
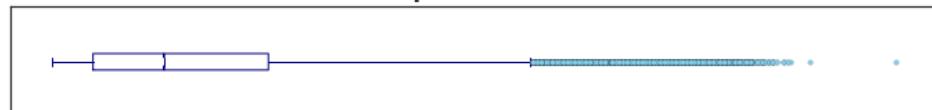


## Variables de tipo carácter

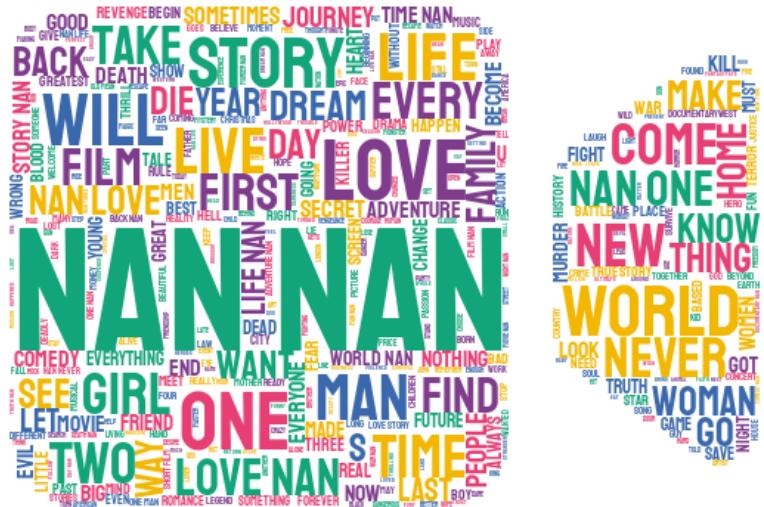
## *overview*



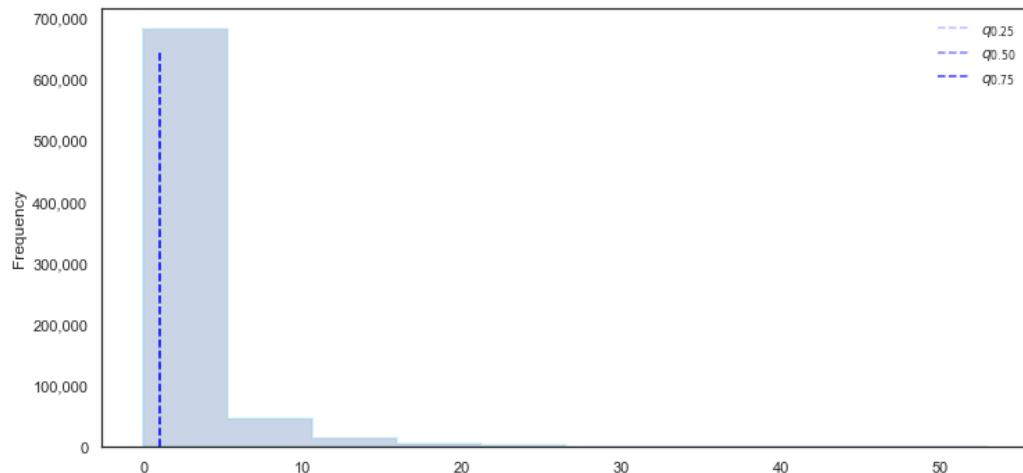
## Cantidad de palabras : overview



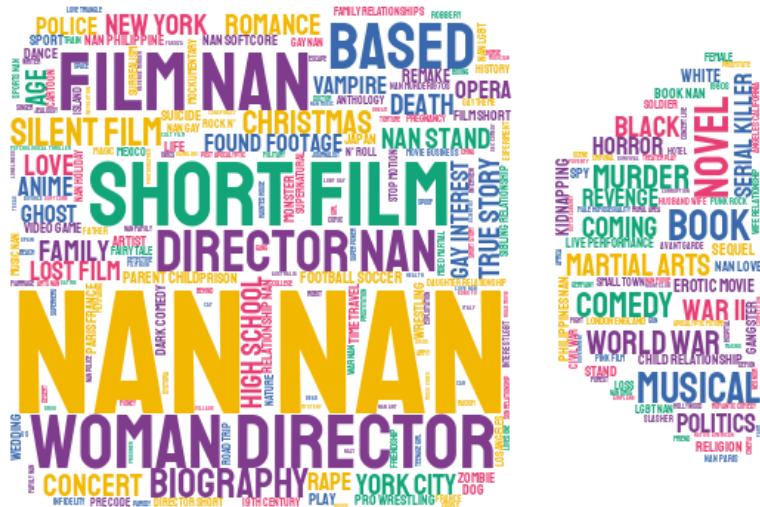
*tagline*



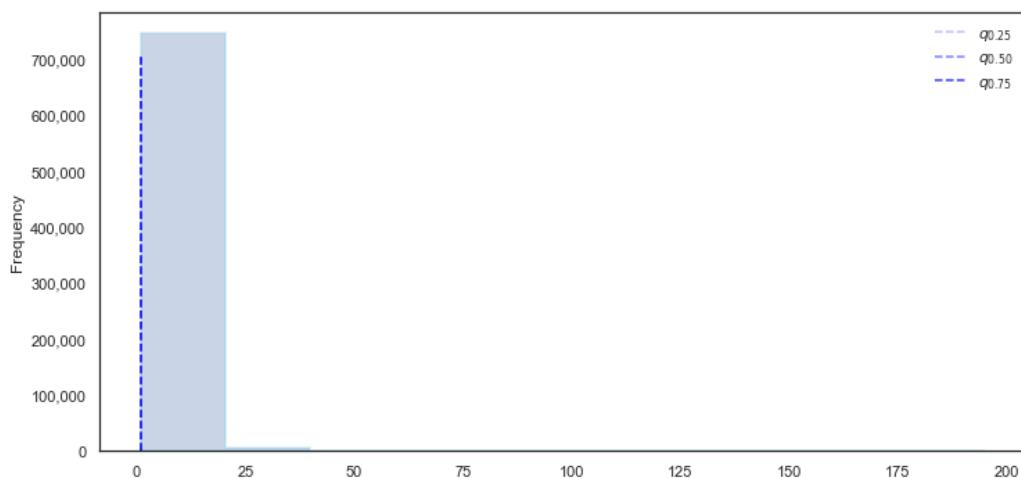
## Cantidad de palabras : tagline



### *Keywords*



## Cantidad de palabras : keywords



Finalmente, como conclusiones al análisis exploratorio, podemos mencionar que, aunque se cuenta con una gran cantidad de películas, muchas de ellas no cuentan con información relevante. Por esta razón, el uso de determinadas películas no será relevante para futuros análisis. A continuación, se presentan las metodologías aplicas con las consideraciones aplicadas para el correcto aprovechamiento de nuestros datos.

## Modelaje supervisado

### Objetivo

El incentivo principal del uso del modelaje supervisado se basa en la capacidad de predecir con ayuda de las variables que ya contamos. Por lo cual, el objetivo que se planteó y se trabajó para este apartado es el siguiente:

*“Predecir si una película es buena o mala”*

Este problema, al basarse en la predicción de alguna categoría, su naturaleza es de clasificación, por lo cual, se aplicaron diversos algoritmos de aprendizaje de máquina que atacan de manera eficaz nuestro objetivo:



## Metodología

### Variable objetivo

Para realizar nuestro modelo de clasificación es necesario definir cuando una película es buena y cuando es mala. Por lo cual, el criterio que se tomó para esta segmentación se basa en la intuición natural sobre esta estratificación:

- Una película es buena si sobrepasa el valor de **6.5** de calificación promedio.

### Variables predictoras

Por otro lado, la elección de las variables que determinarían el que una película sea buena o mala obedece a la idea de poder ocupar variables que se encuentren disponibles incluso para una película recién estrenada. Por ejemplo, el ocupar como predictora la ganancia de una película no podría ser tan viable ya que es un dato que no necesariamente se puede obtener o bien, el dato no podría ser tan verídico.

Teniendo en cuenta lo anterior, las variables que se consideran para la construcción del modelo son las siguientes:

Año de lanzamiento

Mes de estreno

Presupuesto

Duración

Géneros relacionados

Sinopsis

Palabras clave

Así, para tener insumos de valor en nuestros modelos, se procede a considerar los siguientes filtrados:

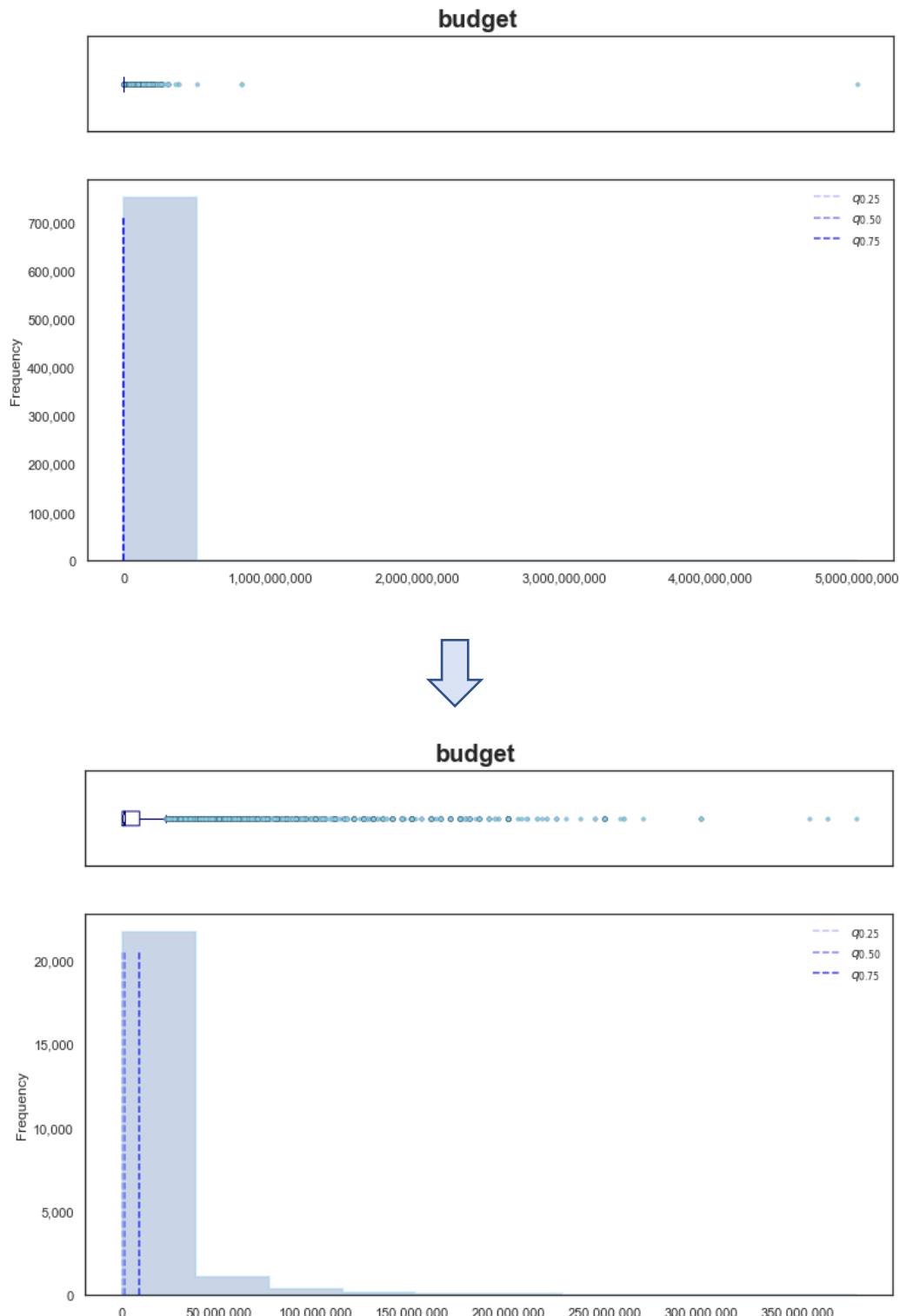
- Filtrado de películas con si cuentan con sinapsis
- Filtrado de películas que si cuentan con información del presupuesto
- Filtrado de películas que si cuentan con información de calificaciones
- Filtrado de películas que cuentan con información de géneros relacionados

Una vez realizado el filtrado, se obtiene un base de datos con una cantidad total de 22,880 películas.

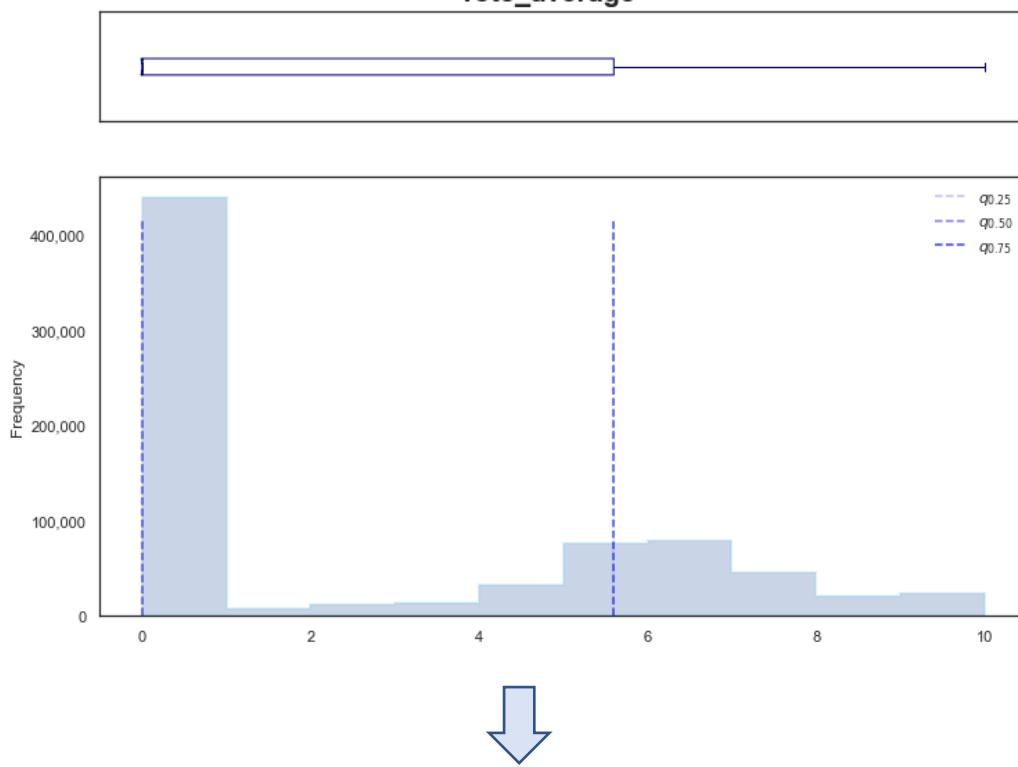
En el siguiente gráfico podemos ver la cantidad de películas, que, de acuerdo a nuestra segregación, se clasifican como una película buena o mala:



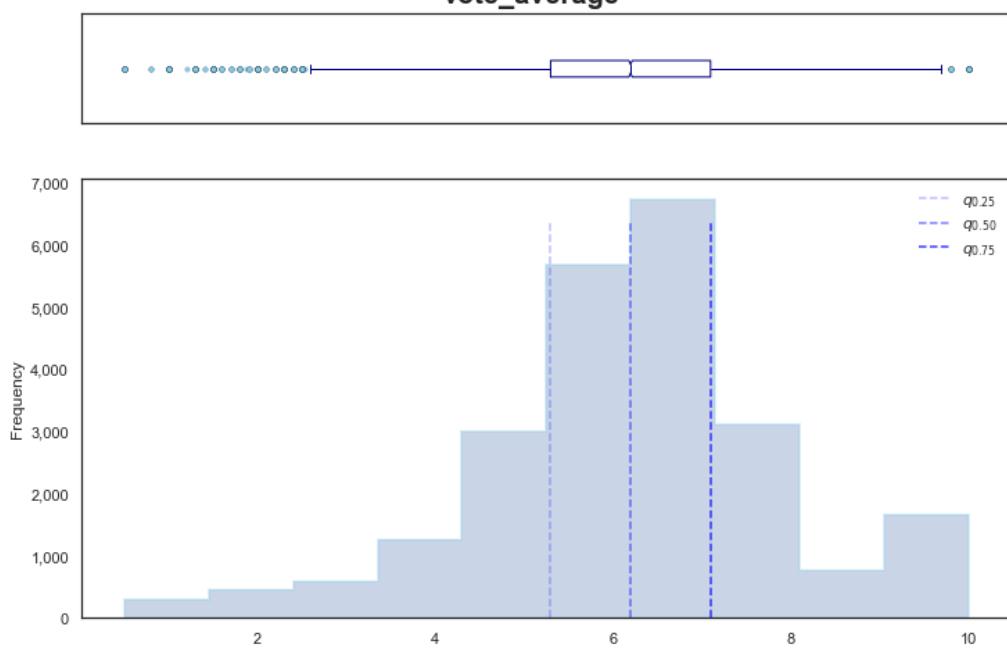
Adicional, se muestran los cambios significativos que sufren las distribuciones al realizar los filtrados.



**vote\_average**



**vote\_average**



## División de los datos

Posterior, para poder entrenar nuestros modelos y generar métricas de desempeño, se realiza la división de nuestro set de datos en dos sets: el de entrenamiento y el de validación. Las proporciones se realizaron como sigue:

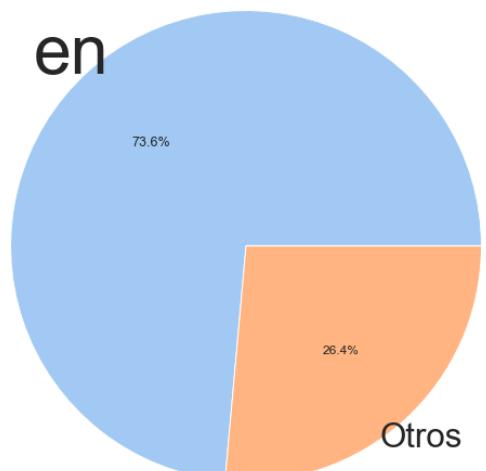
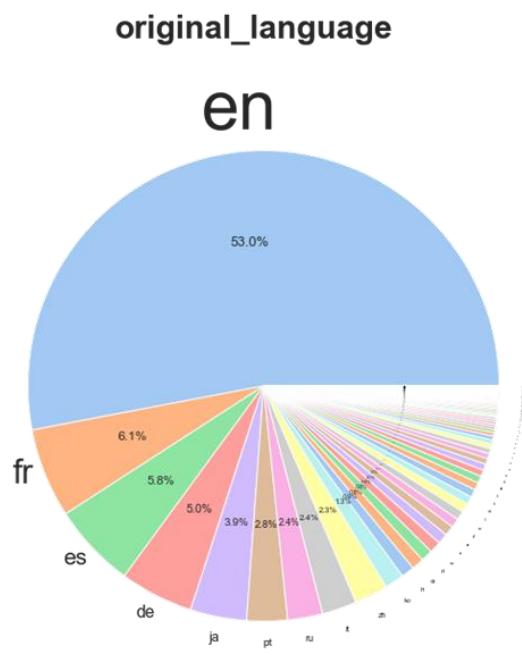
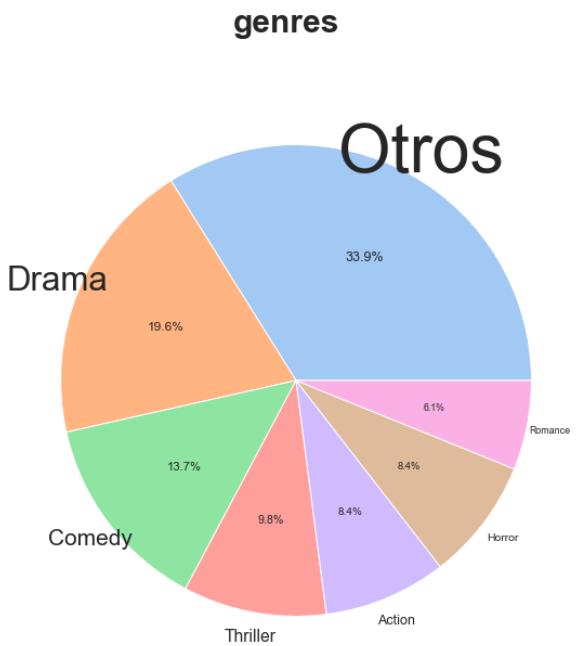
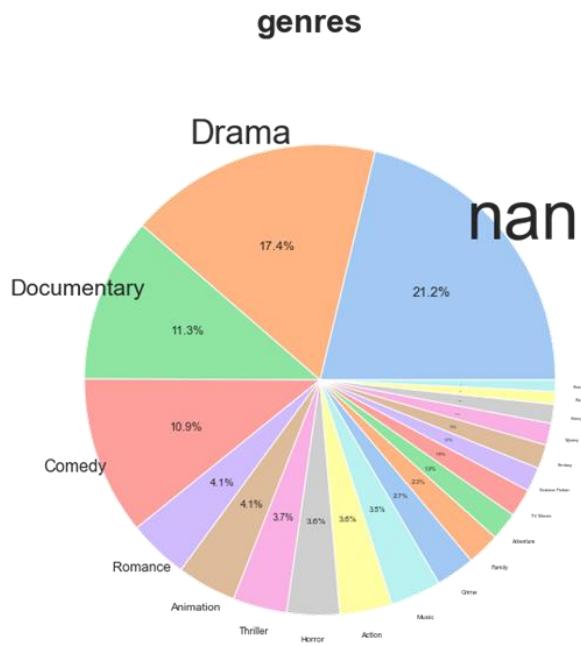


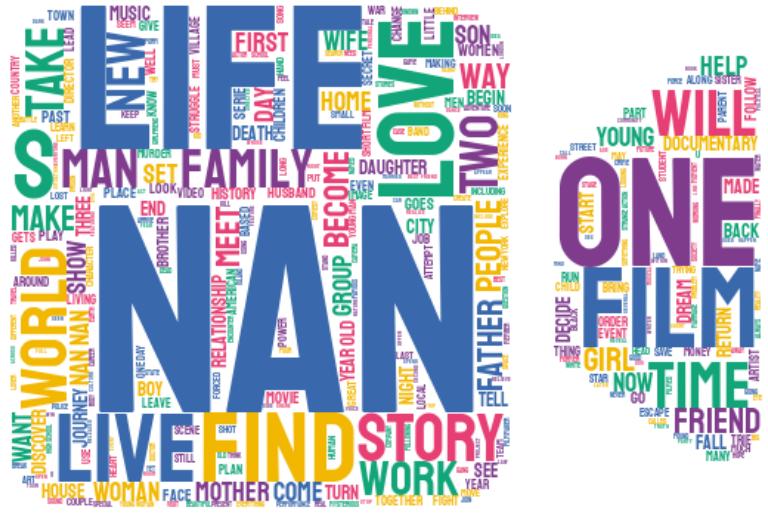
Finalmente, como consideraciones adicionales para poder realizar los entrenamientos se realizaron los siguientes pasos:

- Normalización de la variable de género
- One hot encoding de la variable de género
- Normalización de la variable de idioma
- One hot encoding de la variable de idioma
- Limpieza de las variables de tipo carácter
- Word embedding de las variables de tipo carácter
- Escalamiento de variables

Cabe mencionar que estos últimos pasos se realizan dentro del set de entrenamiento, ya que se busca que estos pasos preserven la estructura general y se apliquen de manera independiente al set de validación. Así, en caso de requerir generar una predicción fuera de la muestra, los pipelines de datos pueden funcionar de manera correcta y se puede generar la nueva predicción.

Como consecuencia, en los siguientes gráficos podemos ver algunos cambios que se suscitan al realizar estos últimos pasos y los filtrados mencionados.





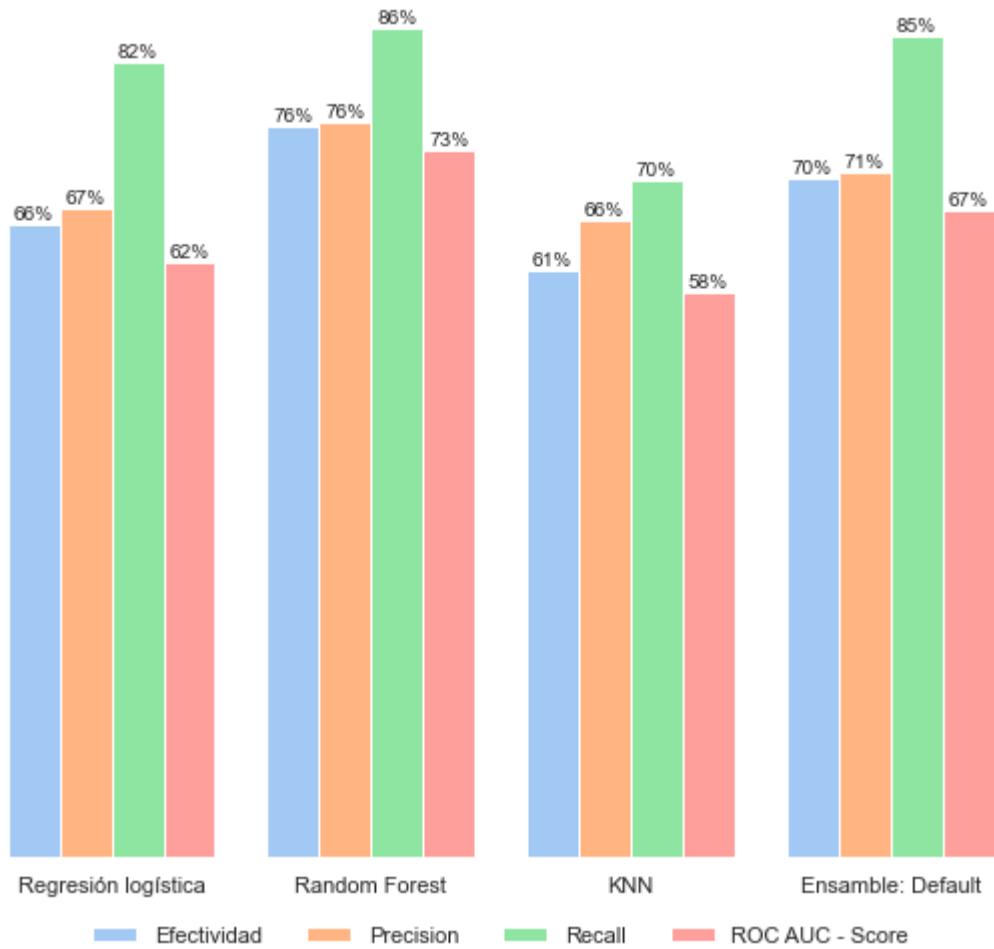
## Resultados

Teniendo los datos necesarios de insumo, se procedió a realizar los entrenamientos. Se entrenaron dos modelos para cada algoritmo, uno correspondiente a un modelo con parámetros default y otro obtenido al realizar el tuning de parámetros, eligiendo aquellos parámetros que lograron el mejor desempeño en el conjunto de validación usando Cross-Validation. Adicional, se consideraron dos métodos diferentes de word embedding, el método TF-IDF y el método Doc2Vec. Por lo cual, se tienen los resultados de los entrenamientos ocupando cada tipo de word embedding.

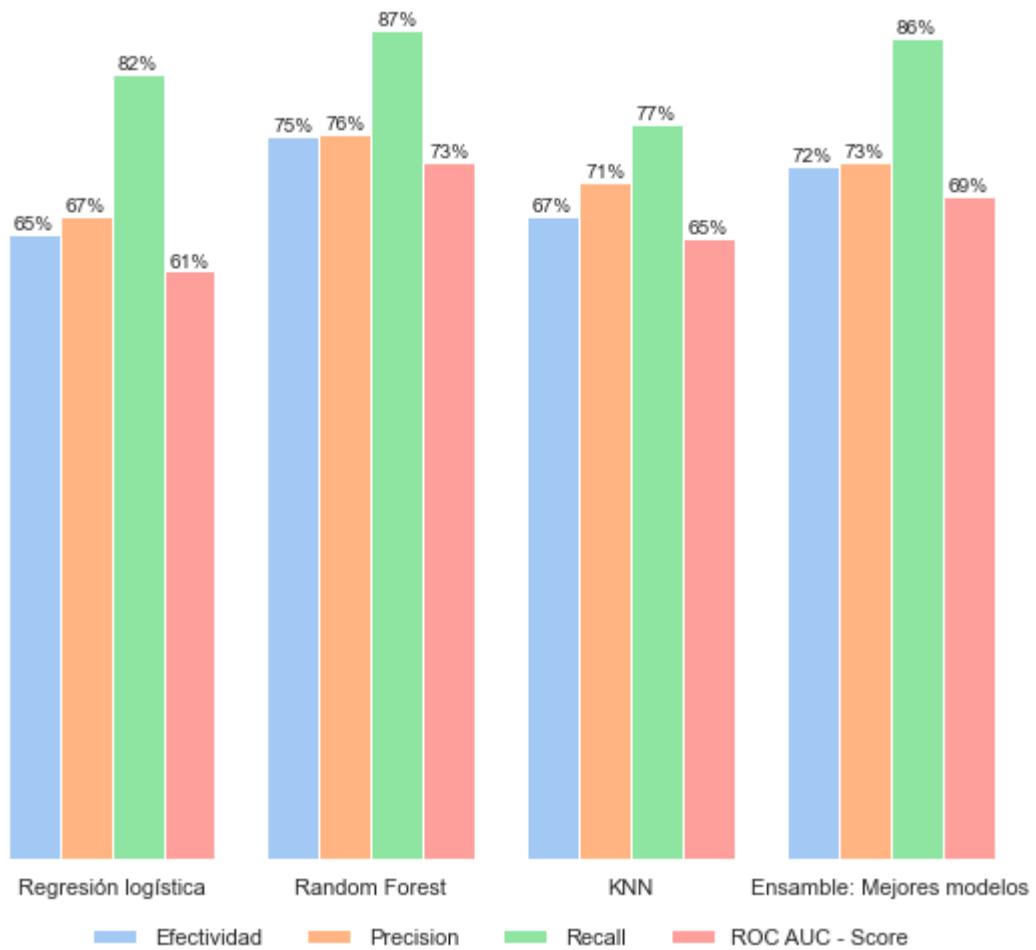
Es importante mencionar que se optó por elegir 100 features resultantes en cada word embedding y que las métricas expuestas fueron obtenidas evaluando los modelos en el conjunto de validación.

## Modelos ocupando TF-IDF.

		Modelos default			
		Regresión logística	Random Forest	KNN	Ensamble
Efectividad		0.6558	0.7563	0.6070	0.7041
Precision		0.6722	0.7621	0.6602	0.7102
Recall		0.8240	0.8591	0.7014	0.8500
ROC AUC		0.6159	0.7320	0.5847	0.6695

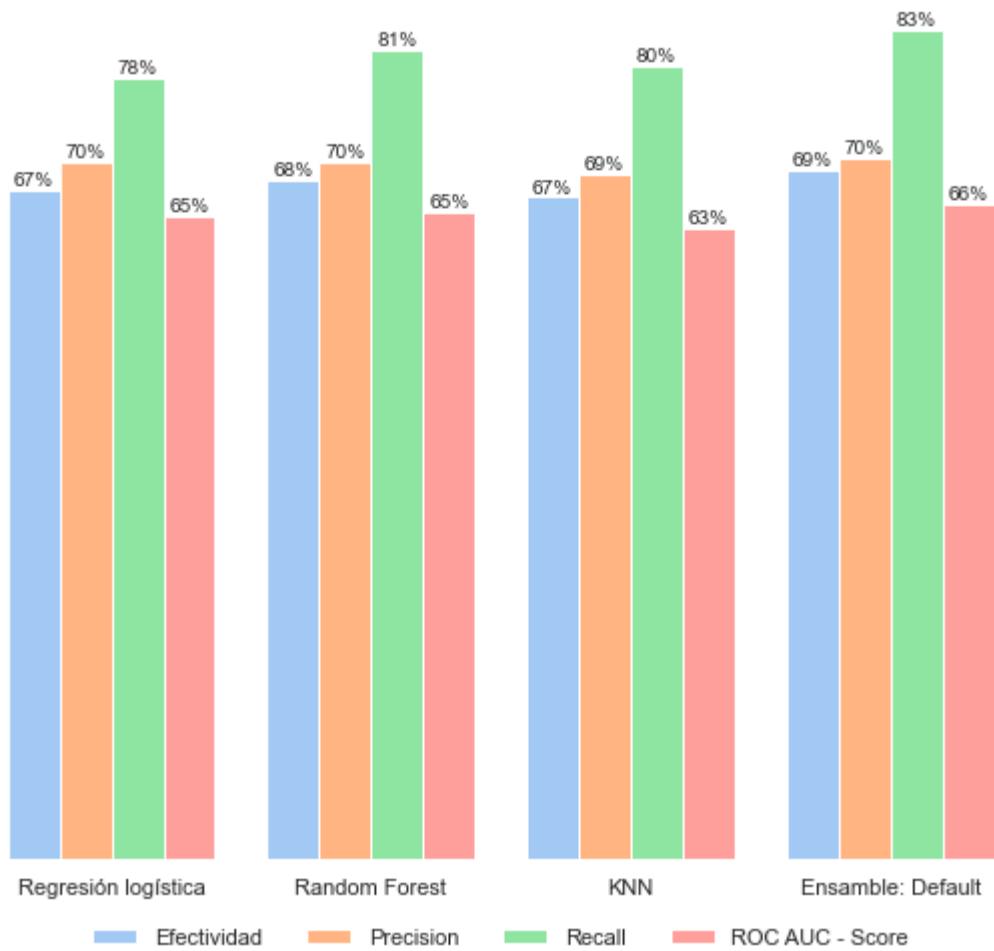


	Mejores modelos			
	Regresión logística	Random Forest	KNN	Ensamble
Efectividad	0.6528	0.7540	0.6709	0.7229
Precision	0.6709	0.7564	0.7062	0.7267
Recall	0.8188	0.8658	0.7664	0.8571
ROC AUC	0.6135	0.7275	0.6483	0.6911
Parámetros	C = 3.4568	max_depth = 50 min_samples_split = 4 n_estimators = 200	leaf_size = 50 n_neighbors = 50	

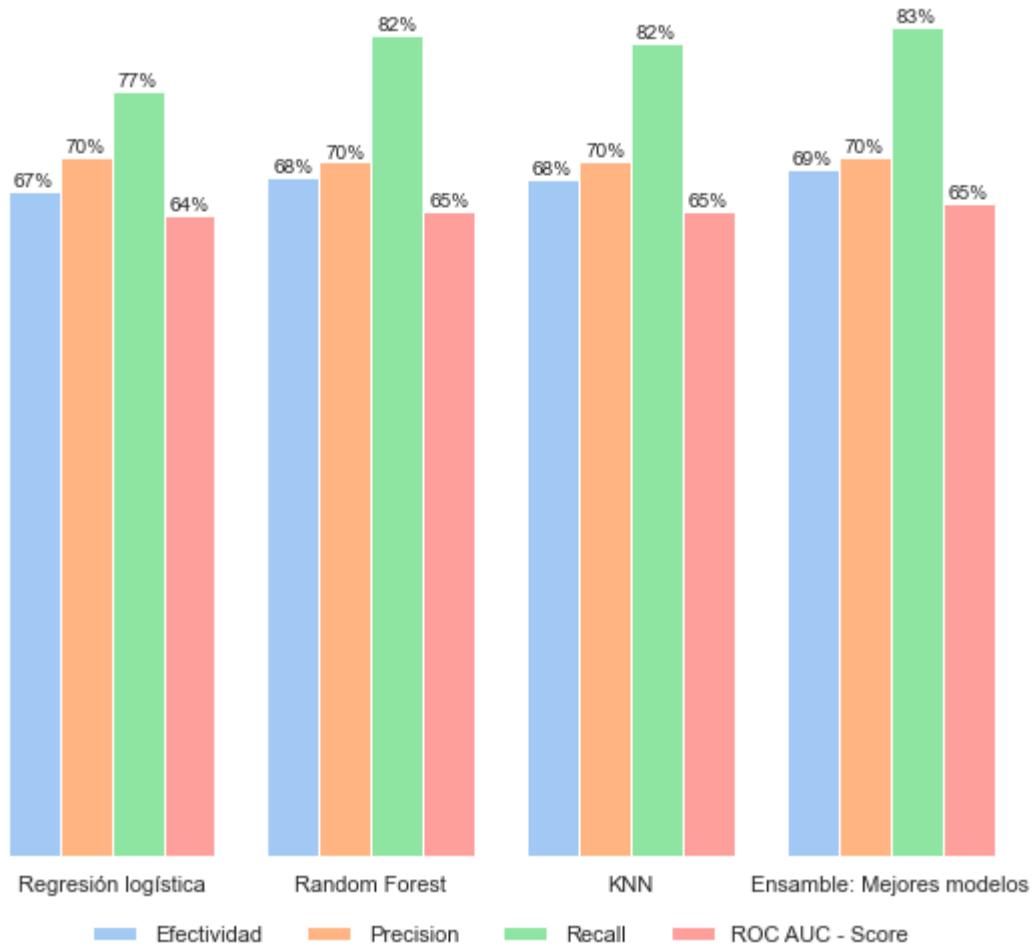


## Modelos ocupando Doc2Vec.

		Modelos default			
		Regresión logística	Random Forest	KNN	Ensamble
Efectividad		0.6727	0.6824	0.6658	0.6930
Precision		0.7013	0.7013	0.6897	0.7047
Recall		0.7850	0.8132	0.7978	0.8344
ROC AUC		0.6461	0.6515	0.6345	0.6596



	Mejores modelos			
	Regresión logística	Random Forest	KNN	Ensamble
Efectividad	0.6670	0.6802	0.6785	0.6885
Precision	0.7014	0.6958	0.6966	0.7010
Recall	0.7679	0.8230	0.8154	0.8317
ROC AUC	0.6431	0.6464	0.6460	0.6546
Parámetros	C = 8.2972	criterion = 'entropy' min_samples_leaf = 3 n_estimators = 50	leaf_size = 100	n_neighbors = 10

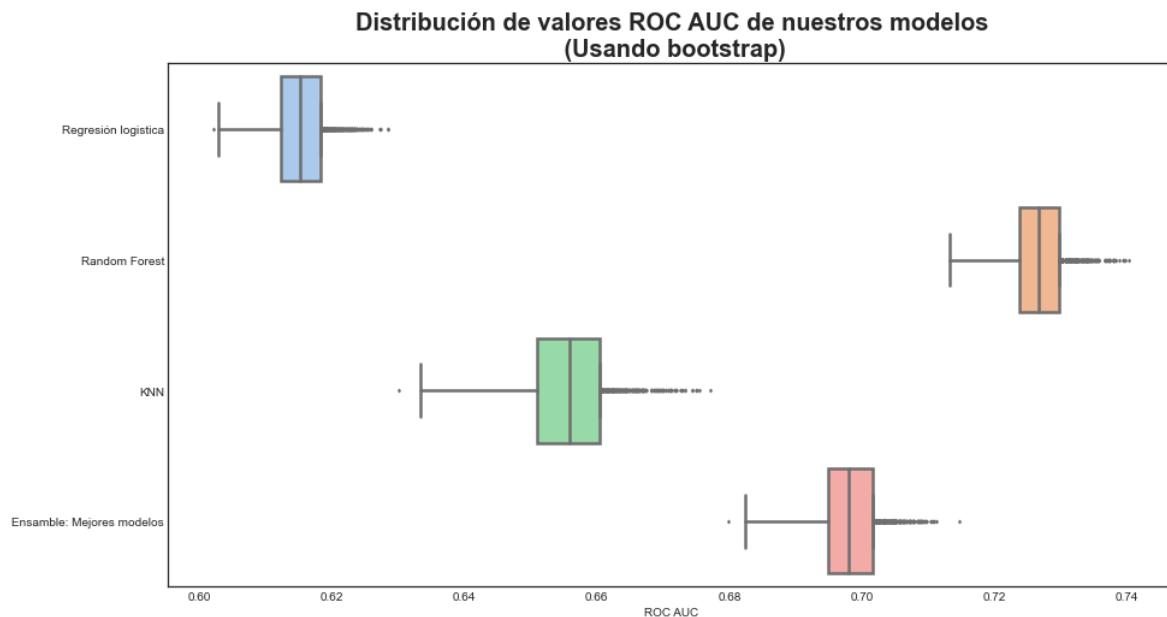


## Estabilidad

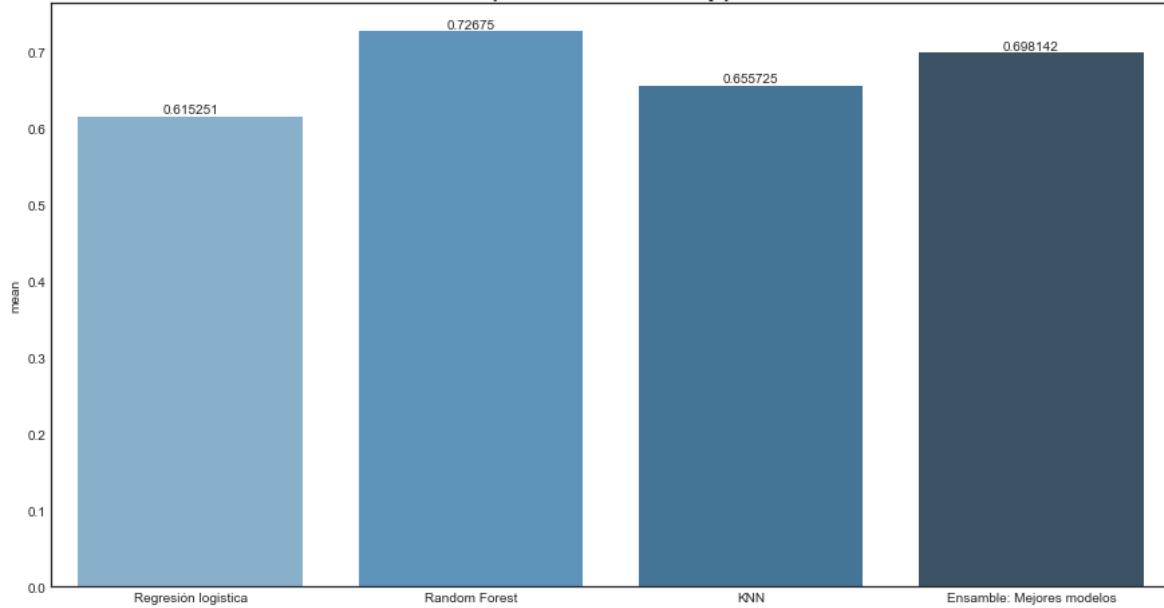
A manera de verificar que nuestros modelos estén brindando métricas confiables y poder generar una elección de modelo, se realizó un proceso de bootstrap con los modelos obtenidos de hacer el tuning de parámetros.

Para este proceso, solamente se hizo bootstrap con los mejores modelos usando el embedding de TF-IDF, ya que los modelos usando Doc2Vec aunado a que brindan peores resultados, el proceso de entrenamiento es considerablemente mayor al de TF-IDF, por lo cual, realizar bootstrap con este método de embedding resultaría en un tiempo excesivo de cómputo.

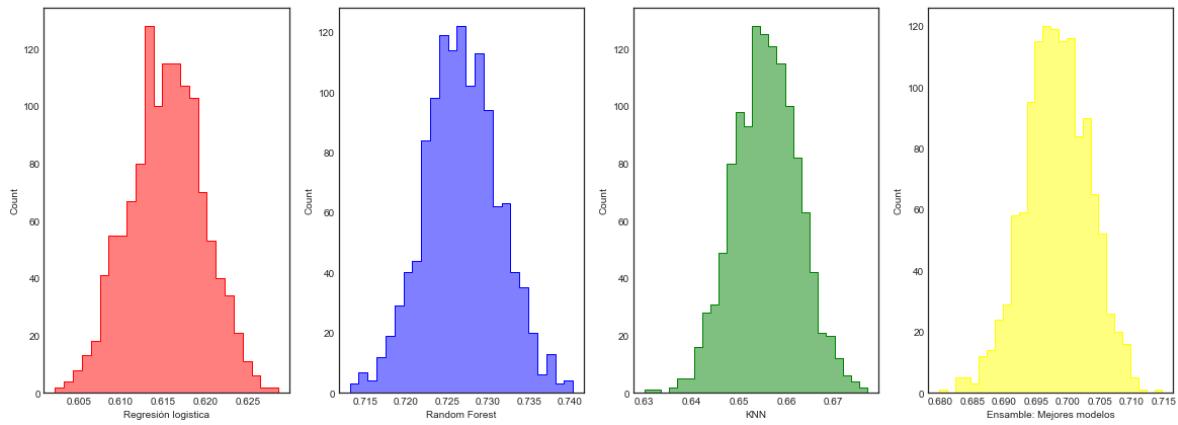
Los siguientes resultados recuperan el valor ROC AUC de cada modelo elegido realizando un total 1,250 iteraciones bootstrap que tomaron alrededor de 9 hrs de cómputo:



**Valor medio de los valores ROC AUC de nuestros modelos  
(usando bootstrap)**



**Distribución de valores ROC AUC de nuestros modelos  
(Usando bootstrap)**



Por lo tanto, de las gráficas anteriores, podemos notar estabilidad en las métricas de validación y es viable elegir nuestro modelo final.

## Elección de modelo

Las métricas obtenidas hacen evidente que los modelos ocupando Doc2Vec no brindan resultados que avalen la preferencia en su uso, por lo que, si nos enfocamos en los modelos usando TF-IDF, el modelo Random Forest demuestra tanto en métricas bootstrap como en entrenamientos sencillos, una eficacia notoria comparando contra los demás modelos.

Por lo tanto, la elección de nuestro modelo final es un Random Forest con los siguientes parámetros:

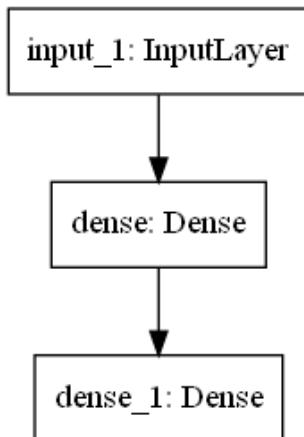
- max\_depth = 50
- min\_samples\_split = 4
- n\_estimators = 200

Haciendo uso de este modelo, podemos obtener una eficacia en nuestra clasificación de aproximadamente 76% y un valor de ROC AUC de aproximadamente 72%.

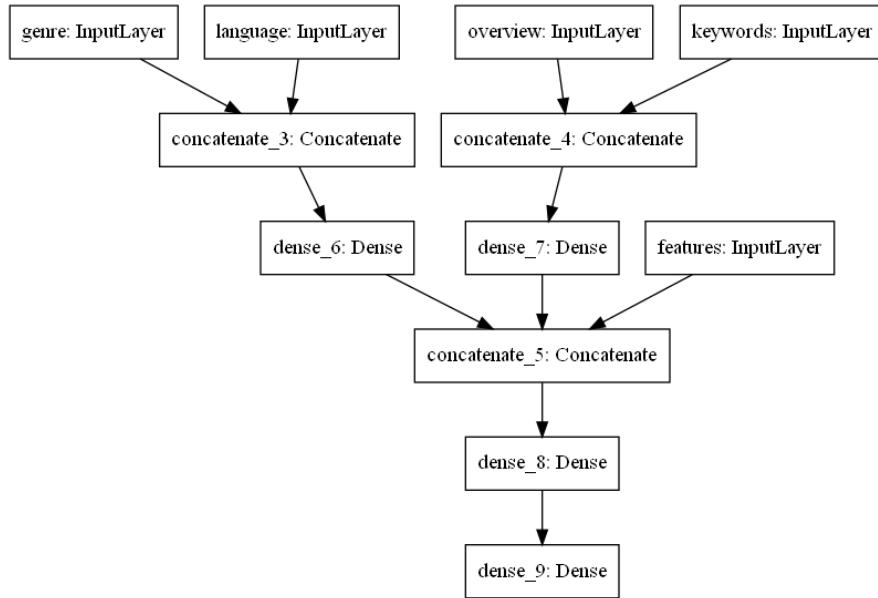
## Redes neuronales

A manera comparativa, también se entrenaron tres tipos de redes neuronales. Todas se entrenaron con un early stopping de 50 épocas sobre la efectividad en el set de validación y las estructuras son las siguientes:

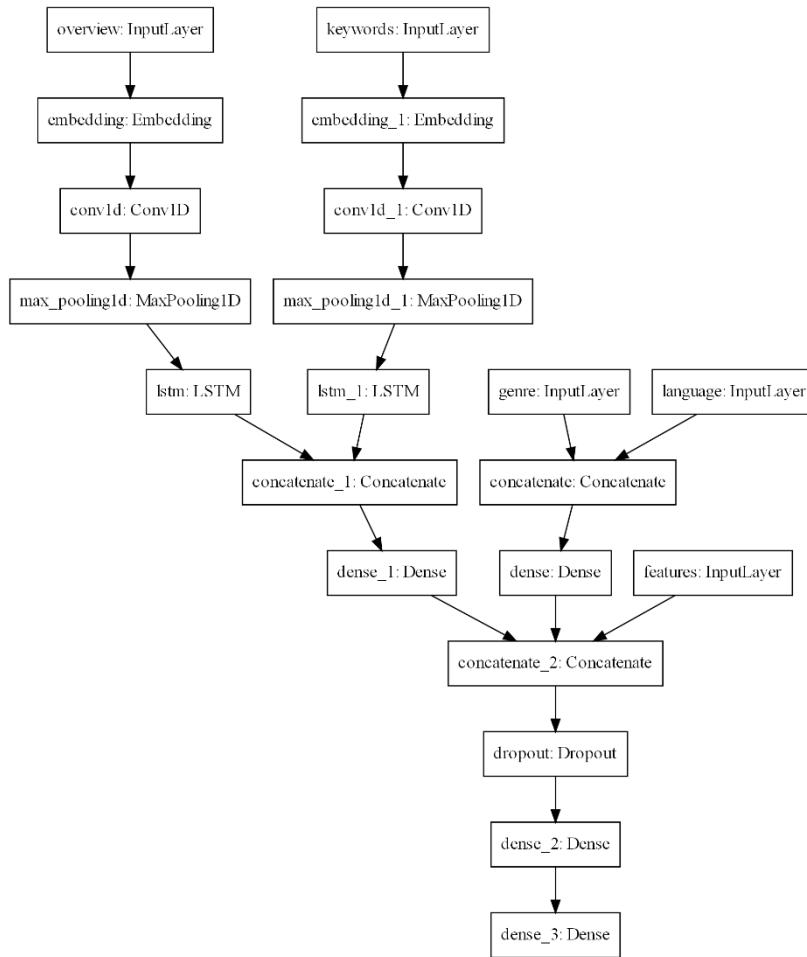
Estructura 1:



Estructura 2:



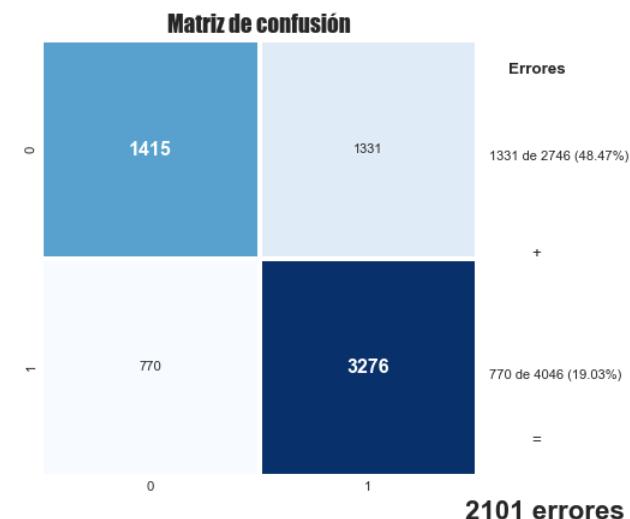
Estructura 3:



## Resultados

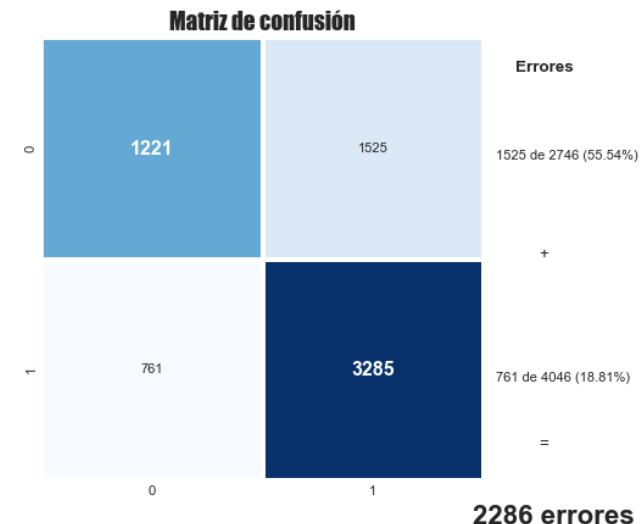
### Redes neuronales (Estructura 1)

Efectividad: 69.067%  
Precision: 71.109%  
Recall: 80.969%  
ROC AUC: 0.66249



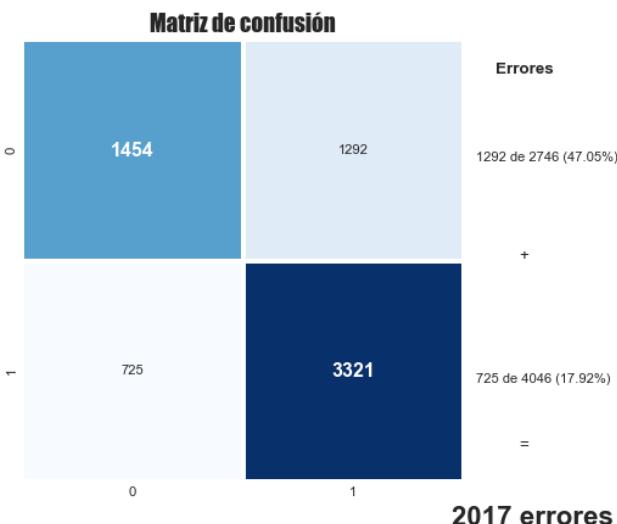
### Redes neuronales (Estructura 2)

Efectividad: 66.343%  
Precision: 68.295%  
Recall: 81.191%  
ROC AUC: 0.62828



### Redes neuronales (Estructura 3)

Efectividad: 70.303%  
Precision: 71.992%  
Recall: 82.081%  
ROC AUC: 0.67515



## Conclusiones

Los resultados de las redes anterior entrenadas muestran un buen desempeño (sobre todo la última estructura) dando pauta a un modelo incluso mejor que algunos modelos que anteriormente habíamos expuesto, sin embargo, las métricas obtenidas aun no son mejores que el modelo Random Forest que se había elegido.

A pesar de esto, las redes neuronales brindan posibilidades de establecer un sin fin de parámetros que podrían incluso superar las métricas del modelo que elegimos, sin embargo, para esto, se requieren hacer muchísimas pruebas exhaustivas en busca de estos parámetros.

Por lo tanto, aunque el uso de redes neuronales puede resultar beneficioso para ciertos tipos de problemas, en este caso, su uso no es tan eficiente si comparamos contra otros modelos menos complejos.

## Modelaje no supervisado

### Objetivo

El uso de modelaje no supervisado tiene diversas aplicaciones para la predicción de valores; de manera muy general, pretende generar agrupaciones de la información. En particular, para nuestro estudio, su incentivo de uso en nuestro proyecto surge de la siguiente idea

*“Segmentar los usuarios con gustos por el cine”*

### Metodología

Para lograr lo anterior, era necesario contar con una base de datos que proporcionara información de usuarios; nuestra base no contaba con dicha información. Así, realizando investigaciones, GroupLens cuenta con bases de usuarios de TMDb que proporcionan calificaciones a las películas del sitio.

## Base de datos de calificaciones de usuarios

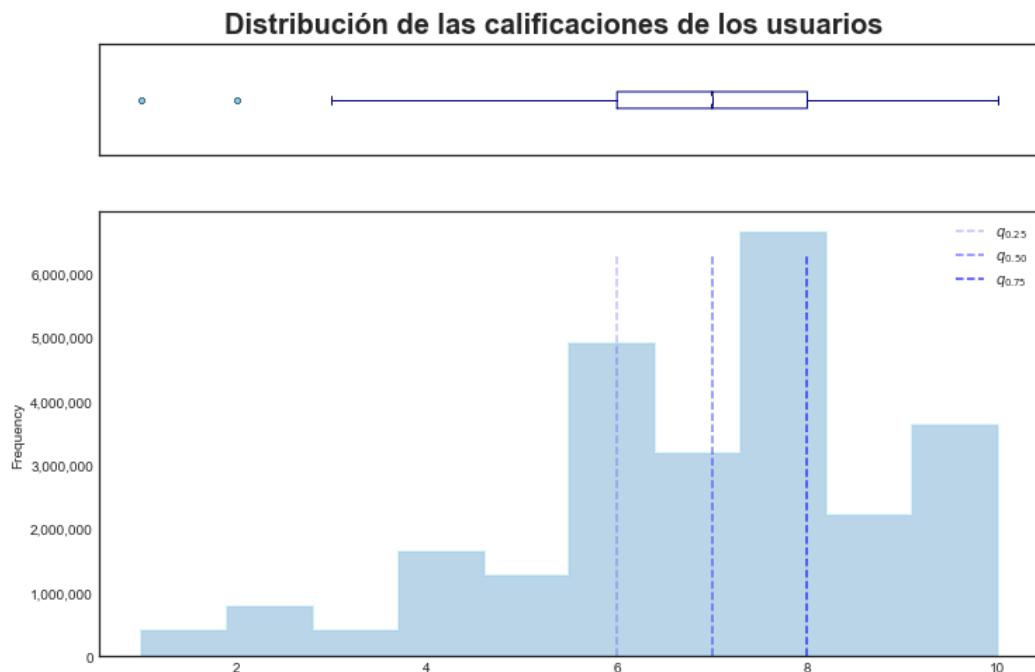
La información que contiene esta base de datos consolida la siguiente información:



En consecuencia, la base cuenta con:

- ID del usuario que califica
- ID de la película calificada
- Calificación brindada

El siguiente gráfico muestra la distribución de las calificaciones que los usuarios brindan



## Consideraciones

A pesar de ya contar con información de usuarios, nuestra base de películas contaba con información que no necesariamente sería relevante para nuestra segmentación de usuarios.

Como primera consideración, las variables categóricas no iban a figurar para el modelado. Posterior, al igual que en el modelaje supervisado, las variables elegidas obedecen a la idea de ser variables que se puedan obtener de una manera efectiva para todos los usuarios. En consecuencia, las variables conservadas para cada película fueron:

- Duración de película,
- Año de estreno
- Calificación promedio

Y, al igual que las variables categóricas, las variables de tipo carácter no fueron consideradas ya que, en el fondo, obedecen a la idea de generar categorías, en consecuencia, el entrenar modelos con estas variables no resultaría buenos modelos.

## Consolidación de la información

Teniendo en cuenta lo anterior, era necesario filtrar nuestra base de películas obedeciendo los criterios impuestos, por lo que se realizo:

- Filtrado de películas que tengan registrada duración
- Filtrado de películas que si cuenten con fecha de lanzamiento
- Filtrado de películas que tengan registrada una calificación promedio

Cabe mencionar que, aunque no figuran las variables categóricas no figuran en el modelado, estas se conversan para futuros análisis de perfilamiento.

Finalmente, teniendo las variables relevantes de nuestra base de películas, se procedió a realizar la consolidación de bases con ayuda de nuestra llave de unión, el ID de la película calificada.

## Generación de la TAD

Resultando un total de 11,839,138 calificaciones de usuarios al realizar el cruce, el siguiente paso para conseguir nuestro objetivo sería obtener la información a nivel usuario, sin embargo, hasta este punto la información se encuentra a nivel calificación a película, por lo cual se realizó la agrupación de información para cada usuario.

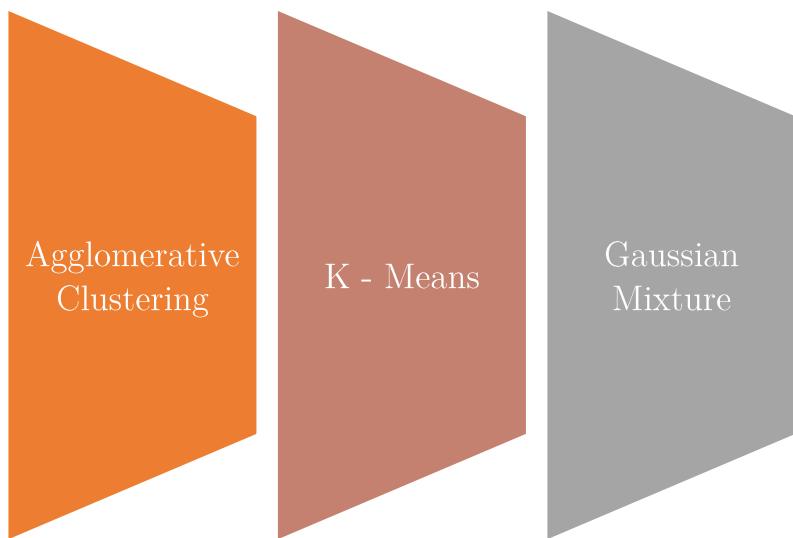
Como resultado, las variables finales de insumo de cada para generar nuestros modelos serían las siguientes:

- Cantidad de películas vistas/calificadas
- Mediana de la popularidad de las películas calificadas
- Mediana del año de las películas calificadas
- Mediana de la duración de las películas
- Mediana de la calificación promedio de las películas calificadas

Finalmente, obteniendo la información de **162,541** usuarios.

## Modelos a entrenar

Los modelos a entrenar que nos ayudan a generar segmentaciones y lograr nuestro objetivo son los siguientes:

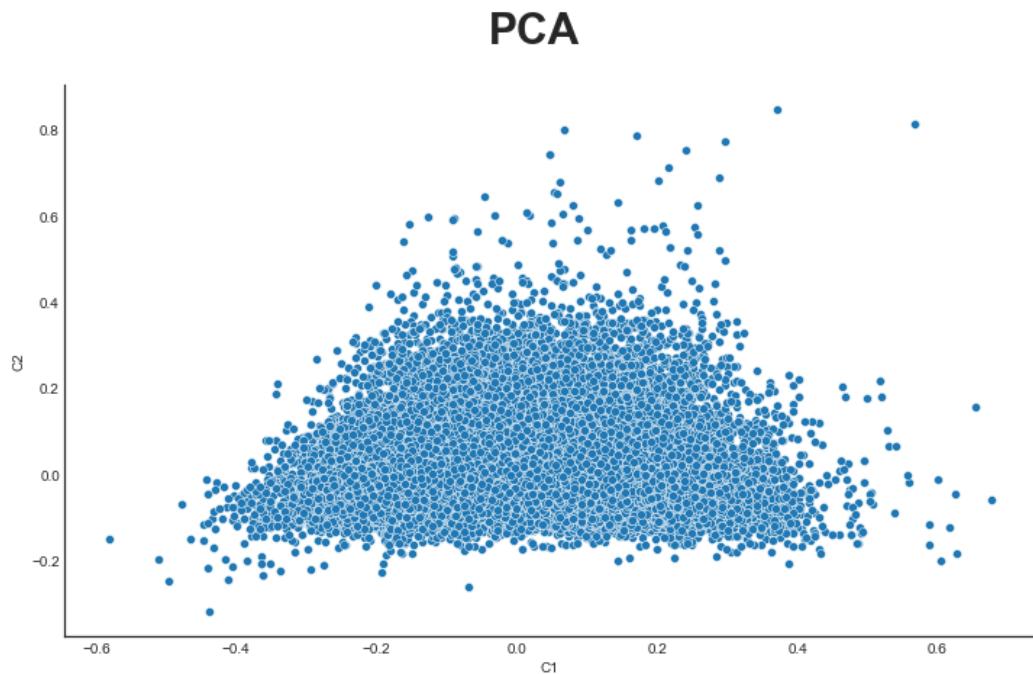


## Resultados

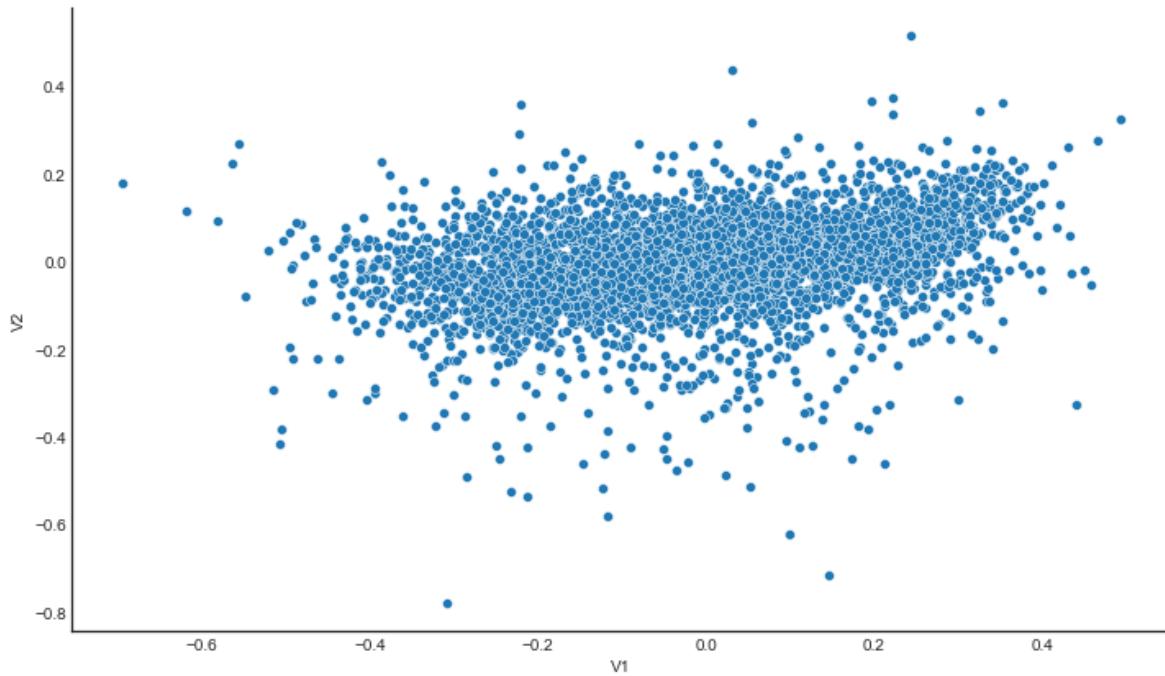
A continuación, se presentan los resultados de los entrenamientos de los modelos anterior propuestos.

### Visualización en dimensión reducida

Primeramente, se realizó una visualización de nuestras características ocupando dos algoritmos de reducción de dimensión. De esta manera, con ayuda de esta visualización se pretendía llegar a inferir una evidente segmentación de información.



## MDS



Como nota a esta última visualización, corresponde a una visualización de una muestra de solo 5,000 usuarios, ya que, al intentar ajustar el algoritmo con la totalidad de la información, este requería una cantidad enorme de recursos computacionales.

Así, de las visualizaciones en dimensión reducida, no marcan una pauta clara para la elección para una cantidad óptima de segmentaciones.

## Modelos entrenados

El modelo óptimo que se consideró fue aquel que presentara una estructura de segmentación adecuada y además tuviera un score de silueta lo más próximo a uno.

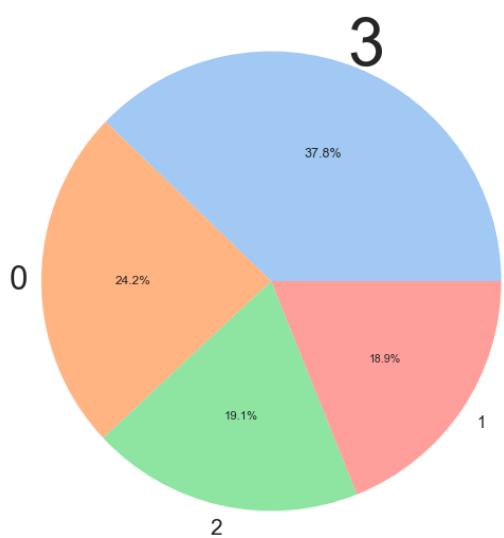
De esta manera, se realizaron diversos entrenamientos, probando cantidades diferentes del número de segmentaciones haciendo uso de los algoritmos de K-Means y Gaussian Mixture. El algoritmo agglomerative clustering presentaba fallos al requerir un tamaño de memoria excesivo, por lo que las métricas correspondientes a este algoritmo, fueron obtenidas entrenando los modelos con una muestra de 5,000 observaciones.

Ocupando 4 clústers

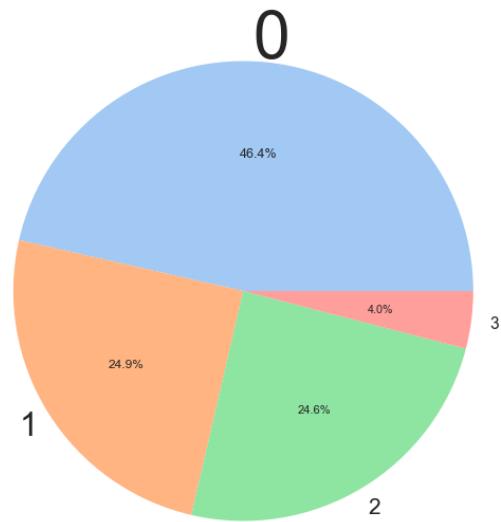
Silhouette score		
4 clústers		
Agglomerative clustering*	K-Means	Gaussian Mixture
0.2362	0.3591	-0.0852

\* Métrica obtenida usando una muestra de 5,000 usuarios

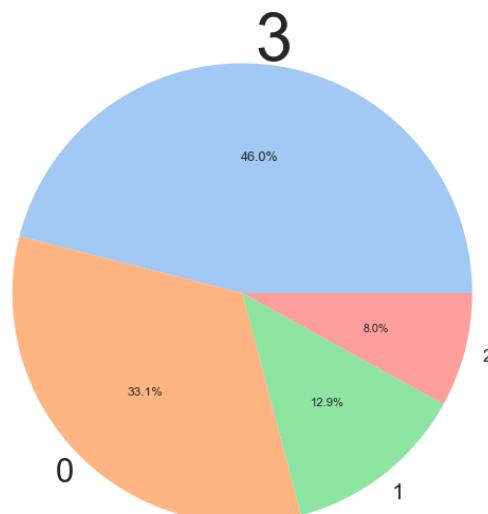
**Método: Ward**



**Método: KMeans**



**Método: GM**

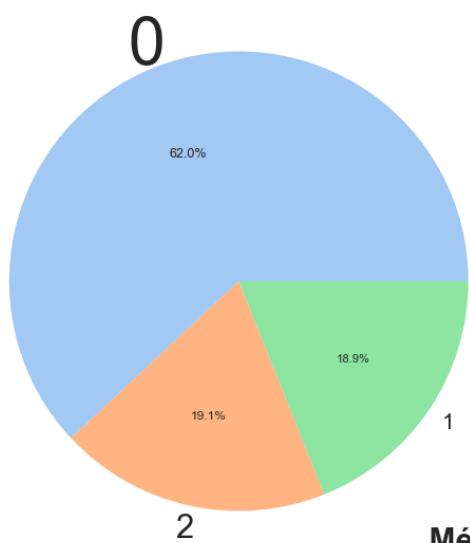


Ocupando 3 clústers

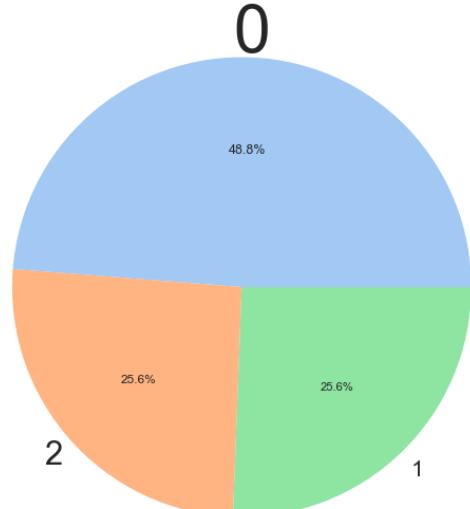
Silhouette score		
3 clústers		
Agglomerative clustering*	K-Means	Gaussian Mixture
0.2970	0.3476	-0.0642

\* Métrica obtenida usando una muestra de 5,000 usuarios

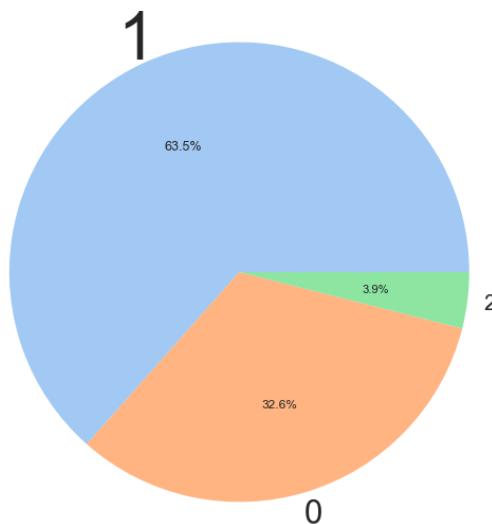
**Método: Ward**



**Método: KMeans**



**Método: GM**



## Elección de modelo

De los resultados anterior expuestos, podemos observar que el modelo con mejor score de silueta es el modelo K-Means con 4 clusters, sin embargo, si observamos la segmentación que realiza, el cuarto cluster prácticamente no cuenta con usuarios. Así, analizando el segundo modelo con mejor score de silueta también resulta ser un modelo entrenado con K-Means con la diferencia de estar ajustado con una cantidad total de 3 cluster pero en esta ocasión ahora este modelo cuenta con una clara segmentación de usuarios.

**Por lo tanto, el modelo que se elige para realizar la correcta segmentación de usuarios de TMDb es el modelo K – Means ajustado con un total de 3 cluster, logrando una eficacia en el score de silueta de 0.3476.**

## Pruebas estadísticas.

A pesar ya haber elegido el modelo, es necesario verificar que el modelo elegido en efecto es confiable, por lo cual, se procede a realizar las siguientes pruebas estadísticas para poder validar que las variables usadas cuentan con suficiente evidencia estadística para determinar un buen modelo.

Las pruebas realizadas son las siguientes:

Prueba	Uso
<i>Kruskal - Wallis</i>	Comparar variables entre los clusters
<i>Kruskal - Wallis</i>	Comparar variables de cada clúster contra la TAD
<i>Tuckey</i>	Comparar variables entre pares de clusters

## Kruskal – Wallis (Entre clusters)

Variable	statistic	p - value	Decisión
Cantidad de películas	2,986.74	0.00	Distribuciones diferentes
Mediana de popularidad	119,323.74	0.00	Distribuciones diferentes
Mediana del año de estreno	12,830.47	0.00	Distribuciones diferentes
Mediana de la duración	85,663.37	0.00	Distribuciones diferentes
Mediana de la calificación promedio	130,090.82	0.00	Distribuciones diferentes

## Kruskal – Wallis (Contra la TAD)

TAD vs Cluster 0		
Variable	p - value	decision
Cantidad de películas	0.0000	Distribución diferente
Mediana de popularidad	0.4793	<b>Distribución parecida</b>
Mediana del año de estreno	0.0000	Distribución diferente
Mediana de la duración	0.1656	<b>Distribución parecida</b>
Mediana de la calificación promedio	0.0001	Distribución diferente

TAD vs Cluster 1		
Variable	p - value	decision
Cantidad de películas	0.0000	Distribución diferente
Mediana de popularidad	0.0000	Distribución diferente
Mediana del año de estreno	0.0000	Distribución diferente
Mediana de la duración	0.0000	Distribución diferente
Mediana de la calificación promedio	0.0000	Distribución diferente

TAD vs Cluster 2		
p_value	decision	
Cantidad de películas	0.0000	Distribución diferente
Mediana de popularidad	0.0000	Distribución diferente
Mediana del año de estreno	0.0000	Distribución diferente
Mediana de la duración	0.0000	Distribución diferente
Mediana de la calificación promedio	0.0000	Distribución diferente

## Tuckey

Variable	Cantidad de veces que la distribución fue diferente entre clusters	Decisión
Cantidad de películas	3/3	Conservar la variable
Mediana de popularidad	3/3	Conservar la variable
Mediana del año de estreno	3/3	Conservar la variable
Mediana de la duración	3/3	Conservar la variable
Mediana de la calificación promedio	3/3	Conservar la variable

## Conclusiones

La evidencia estadística nos brinda la posibilidad de concluir que el modelo elegido cuenta con un buen desempeño y que además las variables usadas para su entrenamiento han sido representativas.

**Finalmente, podemos afirmar que el mejor modelo para solventar nuestro objetivo de segmentación de usuarios es el modelo K – Means ajustado con 3 clusters.**

## Perfilamiento

Elegido el modelo, a continuación, se presenta el análisis de los segmentos encontrados. Tres segmentos, cada uno con características diferentes.

Cabe mencionar que, para el presente perfilamiento, se opta por usar, además de las variables con las que se entreno el modelo, las variables categóricas. Estas variables consideran una ponderación por gusto, es decir, si el usuario proporciona una calificación mayor a 7 a la película, las variables categóricas correspondiente tomarán el valor de 1

A continuación, se presentan las estadísticas relevantes de los clústers:

Estadísticas del cluster 0 : 79,242 usuarios

		Promedio	std	min	0.25	0.5	0.75	max
Variables numéricas	Cantidad de películas vistas/calificadas	85	113	2	21	44	103	2,367
	Popularidad	11	2	1	10	11	13	41
	Año de estreno	1,995	5	1,931	1,993	1,995	1,998	2,011
	Duración	104	4	74	102	105	107	147
	Calificación promedio	7	0	6	7	7	7	8
Variable género	Action	8	10	-	2	5	11	168
	Comedy	13	16	-	3	8	16	295
	Documentary	1	1	-	-	-	1	28
	Drama	24	29	-	7	14	30	487
	Horror	4	6	-	1	2	5	107
	Otros géneros	35	44	-	9	20	44	732
	Romance	8	11	-	2	5	11	186
	Thriller	11	13	-	3	7	14	212
Variable idioma original	Otros lenguajes	7	9	-	2	4	9	172
	en	32	38	-	9	18	40	656
	es	1	1	-	-	-	1	29
	fr	4	5	-	1	2	5	86

Estadísticas del cluster 1 : 41,636 usuarios

		Promedio	std	min	0.25	0.5	0.75	max
Variables numéricas	Cantidad de películas vistas/calificadas	77	151	1	16	32	75	##
	Popularidad	6	2	1	4	6	7	20
	Año de estreno	1,997	5	1,930	1,994	1,997	2,000	2,013
	Duración	98	5	7	96	98	101	130
	Calificación promedio	6	0	4	6	6	7	7
Variable género	Action	7	10	-	2	4	8	257
	Comedy	11	17	-	3	6	13	527
	Documentary	1	3	-	-	-	2	150
	Drama	19	28	-	5	10	21	791
	Horror	4	6	-	1	2	5	174
	Otros géneros	26	42	-	6	13	29	1,203
	Romance	6	10	-	1	3	7	258
	Thriller	8	12	-	2	5	9	319
Variable idioma original	Otros lenguajes	8	12	-	2	4	9	496
	en	25	38	-	7	13	27	1,111
	es	1	2	-	-	-	1	70
	fr	4	5	-	1	2	4	136

Estadísticas del cluster 2 : 41,663 usuarios

		Promedio	std	min	0.25	0.5	0.75	max
Variables numéricas	Cantidad de películas vistas/calificadas	46	40	1	21	33	56	842
	Popularidad	16	3	7	14	15	18	54
	Año de estreno	1,994	4	1,946	1,993	1,994	1,997	2,007
	Duración	110	4	86	108	110	113	145
	Calificación promedio	7	0	6	7	7	7	8
Variable género	Action	5	5	-	2	3	6	83
	Comedy	6	6	-	2	5	8	118
	Documentary	1	1	-	-	-	1	9
	Drama	14	13	-	6	10	18	237
	Horror	2	2	-	-	1	3	38
	Otros géneros	20	19	-	9	15	25	371
	Romance	4	5	-	1	3	6	87
	Thriller	6	6	-	3	5	8	113
Variable idioma original	Otros lenguajes	4	4	-	2	3	5	74
	en	17	15	-	7	12	21	298
	es	0	1	-	-	-	-	9
	fr	2	2	-	1	1	3	25

De estas métricas, se propuso el siguiente perfilamiento para futuras predicciones y estrategias de mercado aplicadas a cada segmento:

Perfilamiento						
Nombre	Número de cluster	Cantidad de películas calificadas / observadas	Popularidad	Duración	Calificación	Lenguaje
Fans	0	85	De todo	Estandar	Regulares	En su mayoría inglés
Críticos	1	77	Underground	Cortas y estandar	Malas y buenas	Varios idiomas
General	2	46	Cartelera	Largas	Buenas	Solo inglés

## Resumen

---

El mejor modelo para atacar el problema propuesto en la modelación supervisada:

“Predecir si una película es buena o mala”

Es un Random Forest que brinda una eficacia promedio de 76%

---

El mejor modelo para atacar el problema propuesto en la modelación no supervisada:

“Segmentar los usuarios con gustos por el cine”

Es un modelo K – Means, ajustando 3 segmentaciones que resultan en el siguiente perfilamiento:

Perfilamiento						
Nombre	Número de cluster	Cantidad de películas calificadas / observadas	Popularidad	Duración	Calificación	Lenguaje
Fans	0	85	De todo	Estandar	Regulares	En su mayoría inglés
Críticos	1	77	Underground	Cortas y estandar	Malas y buenas	Varios idiomas
General	2	46	Cartelera	Largas	Buenas	Solo inglés

## Comentarios finales

El uso de algoritmos de aprendizaje brinda posibilidades increíbles. Este proyecto es un ejemplo del aprovechamiento de estos algoritmos, logrando atacar de manera eficaz problemas que pueden surgir en el día a día.

Finalmente, vale la pena mencionar que, aunque los modelos anteriores expuestos han tenido un buen desempeño, aún queda trabajo por realizar en harás de encontrar un mejor desempeño y poder llevar estos modelos a un entorno meramente productivo.

## Apéndice

Código ocupado a partir de la siguiente página.

```

1 import matplotlib.ticker as tkv
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 import numpy as np
5
6 def formatter(str_number):
7     try:
8         str_number = float(str_number)
9         if str_number.is_integer():
10             return "{:,}.".format(str_number, 0)
11         else:
12             return "{:,.(1f)}.".format(str_number, 1)
13     except:
14         return str_number
15
16 def bar(db,variable,title="barlabs=None,barlabelrot = 'horizontal',rot=0,topNone,ax=None,scaled = None,format=None,yvisible=False,palette='Wistia_r'):

17     if top is None:
18         counts=db[variable].value_counts().head(top)
19     else:
20         counts=db[variable].value_counts()
21
22     if scaled:
23         counts = counts/scaled
24
25     if ax is not None:
26         g=sns.barplot(x=counts.index,y=counts,palette=palette,ex=ax,edgecolor = 'white')
27         ax.set_xticks([x for x in range(len(counts))],fontsize=30,fontweight='bold')
28         if barlabelrot:
29             if format:
30                 g.bar_label(g.containers[0],labels=[formatter(x) for x in g.containers[0].datavalue],rotation=barlabelrot)
31             else:
32                 g.bar_label(g.containers[0],rotation=barlabelrot)
33
34         else:
35             g.bar_label(g.containers[0],rotation=barlabelrot)
36
37         pass
38         ax.tick_params(labelrotation=rot)
39         ax.set_yticks([])
40     else:
41         plt.figure(figsize=(10,6))
42
43     if yvisible:
44         g=sns.barplot(x=[formatter(c) for c in counts.index],y=counts,palette=palette,ex=ax,edgecolor = 'white', alpha = 0.5)
45         if barlabel==True:
46             if format:
47                 g.bar_label(g.containers[0],labels=[formatter(x) for x in g.containers[0].datavalue],rotation=barlabelrot)
48             else:
49                 g.bar_label(g.containers[0],rotation=barlabelrot)
50         else:
51             pass
52         plt.xticks(rotation=rot)
53
54         plt.gca().get_yaxis().set_visible(False)
55
56     else:
57         g=sns.barplot(x=[formatter(c) for c in counts.index],y=counts,palette=palette,ex=ax,edgecolor = 'white',alpha=0.5)
58         sns.despine(bottom = True, left = True)
59         if barlabel==True:
60             if format:
61                 g.bar_label(g.containers[0],labels=[formatter(x) for x in g.containers[0].datavalue],rotation=barlabelrot)
62             else:
63                 g.bar_label(g.containers[0],rotation=barlabelrot)
64         else:
65             pass
66         plt.xticks(rotation=rot)
67         plt.gca().set_yticks([])
68
69     plt.suptitle(title,size=30,fontweight='bold') if title else plt.suptitle(variable,size=30,fontweight='bold')

70 def pie(db,variable,title=None,labels = True,legend = True,threshold = 0):
71     counts=db[variable].value_counts(normalize=True)
72     lns = counts.index.values
73     n = len(lns)
74
75     def my_level_list(data,threshold):
76         list = []
77         for i in range(len(data)):
78             if (data[i]*100/np.sum(data)) > threshold : #2%
79                 list.append(db[i])
80             else:
81                 list.append(0)
82         return list
83
84     def my_autopct(pct):
85         return formatter(pct) + "% if pct > threshold else ''"
86
87     plt.figure(figsize=(10,10))
88
89     if labels:
90         patches, labeltext, pcts = plt.pie(x = counts,labels = my_level_list(counts,threshold),colors = sns.color_palette('pastel'),n,autopct=my_autopct, textprops = dict(rotation_mode = 'default', va='center', ha='center'))
91
92         for i,l in enumerate(labeltext):
93             l.set_fontsize(20*(1/(i*(i+1))))
94
95         for i,l in enumerate(pcts):
96             l.set_fontsize(1.25*l.get_fontsize()*(1-0.05*i))
97
98     else:
99         plt.pie(x = counts,colors = sns.color_palette('pastel'),len(lns))
100
101     if legend:
102         plt.legend(lns)
103
104     plt.suptitle(title,size=30,fontweight='bold') if title else plt.suptitle(variable,size=30,fontweight='bold')

105     hist(db,variable,ctitle=None,min=10,logx=False,logy=False):

106     plt.figure(figsize=(8,9))
107
108     db[variable].plot(kind="hist", logx=logx, logy=logy,bins=nbins, histtype="stepfilled",alpha=0.3, ec="k",edgecolor="skyblue", linewidth=2 ,label="nolegend_")
109
110     plt.gca().xaxis.set_major_formatter(tkr.FuncFormatter(lambda v, p: format(sum(roundy,0), ',')))
111
112     plt.gca().yaxis.set_major_formatter(tkr.FuncFormatter(lambda v, p: format(int(roundy,0)), ','))

113     plt.grid(False)

114     plt.axvline(x=db[variable].quantile(0.25),color='blue',alpha=0.25, ymax=0.00,ls="--")
115     plt.axvline(x=db[variable].quantile(0.5),color='blue',alpha=0.5, ymax=0.00,ls="--")
116     plt.axvline(x=db[variable].quantile(0.75),color='blue',alpha=0.75, ymax=0.00,ls="--")
117
118     plt.legend(['q1(0.25)', 'q2(0.50)', 'q3(0.75)'])
119
120     plt.suptitle(title,size=30,fontweight='bold') if title else plt.suptitle(variable,size=30,fontweight='bold',y=0.02)
121
122     def hist_box(db, variable,ctitle=None,min=10,logx=False,logy=False):
123         f, (ex_box, ex_hist) = plt.subplots(2,figsize=(10,8), sharex=True, gridspec_kw={'height_ratios': [1.2, .88]})

124         db[variable].plot(kind="hist", axes=ex_hist, logx=logx, logy=logy,bins=nbins, histtype="stepfilled",alpha=0.3, ec="k",edgecolor="skyblue", linewidth=2 ,label="nolegend_")
125
126         plt[ex_box].xaxis.set_major_formatter(tkr.FuncFormatter(lambda v, p: format(int(roundy,0)), ',')))
127
128         plt[ex_box].yaxis.set_major_formatter(tkr.FuncFormatter(lambda v, p: format(int(roundy,0)), ',')))

129         plt[ex_hist].grid(False)

130         plt[ex_hist].axvline(x=db[variable].quantile(0.25),color='blue',alpha=0.25, ymax=0.00,ls="--")
131         plt[ex_hist].axvline(x=db[variable].quantile(0.5),color='blue',alpha=0.5, ymax=0.00,ls="--")
132         plt[ex_hist].axvline(x=db[variable].quantile(0.75),color='blue',alpha=0.75, ymax=0.00,ls="--")
133
134         plt[ex_hist].legend(['q1(0.25)', 'q2(0.50)', 'q3(0.75)'])

135         f.suptitle(title,size=30,fontweight='bold',y=0.02) if title else f.suptitle(variable,size=30,fontweight='bold',y=0.02)
136
137     def hist_by_var(db,var,by, normalize = False):
138         if var == by:
139             pass
140         else:
141             if normalize:
142                 try:
143                     aux = pd.DataFrame(pd.cut(db[var],bins = 30)).set_axis([var],axis=1)
144                     aux[by] = db[by]
145                     aux = aux.value_counts().reset_index().pivot_table(index=var,columns = by,value=0,fill_value=0)
146                     aux = aux.T
147                     aux.columns.map(lambda x: x.left)
148                     aux[TOTAL] = aux.apply(sum, axis=1)
149                     aux = aux.apply(lambda x: x / sum(x), axis=1)
150                     aux = aux.T.plot(kind="bar",stacked=True, width=0.9)
151                     plt.locator_params(axis="x", nbins=6)
152                     plt.xticks(rotation=90)
153
154                     plt.show()
155                 except Exception as e:
156                     print(e)
157                     aux = db[[var]]
158                     aux[by] = db[by]
159                     aux = aux.value_counts().reset_index().pivot_table(index=var,columns = by,value=0,fill_value=0)
160                     aux = aux.T
161                     aux.columns.map(lambda x: x.left)
162                     aux[TOTAL] = aux.apply(sum, axis=1)
163                     aux = aux.apply(lambda x: x / sum(x), axis=1)
164                     aux = aux.T.plot(kind="bar",stacked=True, width=0.9)
165
166                     for c in ax.containers:
167                         labels = [v.get_height() if v.get_height() > 0 else 0 for v in c]
168                         ax.bar_label(c, labels=labels, label_type="center")
169
170                     plt.show()
171
172                 else:
173                     try:
174                         aux = pd.DataFrame(pd.cut(db[var],bins = 30)).set_axis([var],axis=1)
175                         aux[by] = db[by]
176                         aux = aux.value_counts().reset_index().pivot_table(index=var,columns = by,value=0,fill_value=0)
177                         aux = aux.T
178                         aux.columns.map(lambda x: x.left)
179                         aux[TOTAL] = aux.apply(sum, axis=1)
180                         aux = aux.apply(lambda x: x / sum(x), axis=1)
181                         aux = aux.T.plot(kind="bar",stacked=True, width=0.9)
182
183                         for c in ax.containers:
184                             labels = [v.get_height() if v.get_height() > 0 else 0 for v in c]
185                             ax.bar_label(c, labels=labels, label_type="center")
186
187                         plt.show()
188
189                     except Exception as e:
190                         print(e)
191                         aux = db[[var]]
192                         aux[by] = db[by]
193                         aux = aux.value_counts().reset_index().pivot_table(index=var,columns = by,value=0,fill_value=0)
194                         aux = aux.T
195                         aux.columns.map(lambda x: x.left)
196                         aux[TOTAL] = aux.apply(sum, axis=1)
197                         aux = aux.apply(lambda x: x / sum(x), axis=1)
198                         aux = aux.T.plot(kind="bar",stacked=True, width=0.9)
199
200                         for c in ax.containers:
201                             labels = [v.get_height() if v.get_height() > 0 else 0 for v in c]
202                             ax.bar_label(c, labels=labels, label_type="center")
203
204                         plt.show()

```

```

1 import pandas as pd
2 import re
3 from nltk.corpus import stopwords
4 from nltk.sentiment.vader import SentimentIntensityAnalyzer
5 from unidecode import unidecode
6 from sklearn.feature_extraction.text import TfidfVectorizer
7 from sklearn.feature_extraction.text import CountVectorizer
8 from tensorflow.keras.preprocessing.sequence import pad_sequences
9 from keras.preprocessing.text import Tokenizer
10
11
12 def formatter(str_number):
13     try:
14         str_number = float(str_number)
15         if str_number.is_integer():
16             return '{:.1f}'.format(str_number, 0)
17         else:
18             return '{:.1f}'.format(str_number, 1)
19     except:
20         return str_number
21
22 def remove_accents(a):
23     return list(map(unidecode,a))
24
25 def clean_text():
26     def clean_text(c):
27         txt = c.text.lower()
28         txt = re.sub(r'[^a-zA-Z]', ' ',txt)
29         txt = ' '.join([w for w in txt.split() if len(w)>=3])
30         txt = re.sub(r'\d+', ' ', txt)
31         return txt
32     return list(map(clean_text,c))
33
34 def remove_stopwords(txt,stopwords):
35     return list(map(lambda x : ' '.join([item for item in x.lower().split() if item not in stopwords]),txt))
36
37 def get_string_stats(df,y1,y2,y3):
38     df[y1] = y1 + 'len' + y2 + 'len' + y3 + 'len'
39     df[y1] = y1 + 'len' + y2 + 'len' + y3 + 'len' / df[y1].str.len()
40     df[y1] = y1 + 'len' + y2 + 'len' + y3 + 'len' / df[y1].str.len() / df[y1].str.len()
41     sid = SentimentIntensityAnalyzer()
42     vader_df[StringVar].apply(sid.polarity_scores)
43     vader_wd = DataFrame(map(lambda x: vader_,x))
44     vader_.columns = StringVar + ' ' + vader_.columns
45     return (df.join(vader_))
46
47 class Vocabulary:
48     def __init__(self,df,var,splitter=' '):
49         self.str_values = df[var]
50         self.var_index = {x: i for i,x in enumerate(df[var].str.split(splitter)) for x in y}
51         self.counter = getSeries(self.var_index,vocabulary).value_counts()
52         self.vocab_len = len(self.counter)
53         self.len_sentences = df[y1].str.split().str.len()
54         self.longest_sentence = max(self.len_sentences)
55         print(f'El tamaño de vocabulario es de {self.vocab_len} palabras únicas')
56         print(f'La oración más larga es de {self.longest_sentence} palabras')
57         print(f'Las 5 palabras que más se repiten son: {(tuple(self.counter.index[0:5]))}')
58
59 class VectorizeVariable:
60     def __init__(self,tokenizer , X , max_features):
61         self.max_features = max_features
62         self.tokenizer = tokenizer
63         self.X = X
64         self.X_word_index = self.tokenizer.word_index
65         self.X_pad = self.tokenizer.texts_to_sequences(X)
66         self.X_pad = pad_sequences(self.X_pad, maxlen=max_features)
67
68     def transform(self,X_test):
69         X_test = self.tokenizer.texts_to_sequences(X_test)
70         X_test = pad_sequences(X_test,maxlen=self.max_features)
71         return(X_test)
72
73 class StringCleaner():
74     def __init__(self,**kwargs):
75         # pass
76         self.__dict__.update(kwargs)
77
78     def clean(self,x,**kwargs):
79         temp = ''
80         for function in list(self.__dict__.values()):
81             try:
82                 temp = function(temp,kwargs=function.__code__.co_varnames[1:])
83             except:
84                 temp = function(temp)
85         return(temp)

```

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 from sklearn.preprocessing import LabelEncoder
6
7 from sklearn.metrics import mean_squared_error, roc_curve, confusion_matrix, roc_auc_score
8 from sklearn.model_selection import train_test_split
9
10 from sklearn.model_selection import validation_curve
11
12
13 from sklearn.linear_model import LogisticRegression
14 from sklearn.neighbors import KNeighborsClassifier
15 from sklearn.ensemble import RandomForestClassifier
16 from sklearn.model_selection import StratifiedKFold
17
18 from sklearn.experimental import enable_halving_search_cv
19 from sklearn.model_selection import HalvingGridSearchCV
20
21
22 def plot_cf(cf_matrix,title=True,ax=None):
23
24     off_diag_mask = np.eye(cf_matrix.shape, dtype=bool)
25     vmean = np.mean(cf_matrix[off_diag_mask])
26     vmax = np.max(cf_matrix[off_diag_mask])+np.mean(cf_matrix[~off_diag_mask])
27     vmin = np.min(cf_matrix[off_diag_mask])-np.mean(cf_matrix[~off_diag_mask])
28     vmax = np.max(cf_matrix)
29
30     sns.heatmap(cf_matrix, mask=off_diag_mask, cmap='Blues', vmin=vmin, vmax=vmax, char=False,square=True,linewidths=2,annot_kws={'fontsize': 'x-large','weight': 'bold'},ax=ax)
31     sns.heatmap(cf_matrix, annot=True, yfmt='g', mask=off_diag_mask, cmap='Blues', vmin=vmin, vmax=vmax, char=False,square=True,linewidths=2,ex=.05)
32
33     if title is not None:
34         ax.set_title('Matriz de confusión', fontsize=20,fontfamily='fantasy')
35
36     def grafica_curva_roc(y, y_pred, title='Curva ROC'):
37
38         for tpr, _ in roc_curve(y, y_pred)
39             roc_score=roc_auc_score(y, y_pred)
40
41             plt.figure(figsize(15,6))
42             plt.plot([0, 1], [0, 1], 'k--')
43             plt.xlabel('Falsos positivos')
44             plt.ylabel('Tasa de verdaderos positivos')
45             plt.title(title)
46             plt.text(0.6, 0.2, f'ROC AUC: {round(roc_score,4)}', size=15)
47             plt.show()
48
49
50 def model_cf(label,y,y_pred):
51
52     cf_matrix = confusion_matrix(y, y_pred)
53
54     n_i = list(np.sum(cf_matrix))
55     precision=(cf_matrix.diagonal()/list(map(sum,cf_matrix)))
56     errors = n_i*errors
57     xx=[1]*len(errors)
58
59     recall = round(cf_matrix[1][1]/(cf_matrix[1][1] + cf_matrix[1][0]),5)
60     precision=round(cf_matrix[1][1]/((cf_matrix[1][1] + cf_matrix[0][1])),5)
61     roc_score=roc_auc_score(y, y_pred)
62
63     efectividad = round(1-np.sum(errors)/len(y)),5
64
65     plt.figure(figsize(15,6), dpi=80)
66
67     ax1 = plt.subplot2grid((1, 20), (0, 0), colspan=8)
68     ax2 = plt.subplot2grid((1, 20), (0, 8), colspan=8)
69     ax3 = plt.subplot2grid((1, 20), (0, 16))
70
71     ax1.axis('off')
72     ax1.set_title('Recall', fontstyle='italic', fontsize=20, weight='bold', ha='center')
73     ax1.text(0.05,0.35, f'{recall}', fontstyle='italic', fontsize=15, ha='center', fontweight='bold')
74     ax1.text(0.05,0.28, f'({format(efectividad, ".1f")})', fontstyle='italic', fontsize=15, ha='center', fontweight='bold')
75     ax1.text(0.05,0.30, f'Precision', fontstyle='italic', fontsize=15, ha='center', fontweight='bold')
76     ax1.text(0.05,0.28, f'({format(precision, ".1f")})', fontstyle='italic', fontsize=15, ha='center', fontweight='bold')
77     ax1.text(0.05,0.18, f'Recall', fontstyle='italic', fontsize=15, ha='center', fontweight='bold')
78     ax1.text(0.05,0.15, f'({format(recall, ".1f")})', fontstyle='italic', fontsize=15, ha='center', fontweight='bold')
79     ax1.text(0.05,0.08, f'Precision', fontstyle='italic', fontsize=15, ha='center', fontweight='bold')
80     ax1.text(0.05,0.05, f'({format(precision, ".1f")})', fontstyle='italic', fontsize=15, ha='center', fontweight='bold')
81
82     plot_cf(cf_matrix, True,ax2)
83     ax3.sharey(ax2)
84     ax3.xaxis('off')
85
86     for ind, i in enumerate(errors):
87         ax3.text(i,ind*0.35, f'{int(round(i,0))} ({int(round(i,0))}) ({round(errors[ind]*100,2)})', ha='left', fontsize=10)
88
89         ax3.text(0.05,0.15, f'({round(errors[ind]*100,2)})', fontstyle='italic', fontweight='bold')
90         ax3.text(0.1,0.15, f'({int(round(errors[ind]*100,2))})', fontstyle='italic', fontweight='bold')
91         ax3.text(0.1,0.15, f'({int(round(errors[ind]*100,2))})', fontstyle='italic', fontweight='bold')
92         ax3.text(1.1,0.15, f'{int(round(np.sum(errors),0))} errores', fontstyle=20, fontweight='bold', ha='center')
93
94     return efectividad, precision, recall, roc_score

```

```
1 import pandas as pd
2
3 from sklearn.decomposition import PCA
4 from sklearn.manifold import MDS
5 from sklearn.cluster import AgglomerativeClustering
6 from sklearn.cluster import KMeans
7 from sklearn.mixture import GaussianMixture
8
9 from scipy.stats import kruskal
10 from sklearn.metrics import silhouette_score
11 from statsmodels.stats.multicomp import pairwise_tukeyhsd
12
13
14 def get_random_string(length):
15
16     letters = string.ascii_lowercase
17     result_str = ''.join(random.choice(letters) for i in range(length))
18     result_str = 'trybase.' + result_str
19     return result_str
20
21 def plot_method(df,var1,var2,tad,method,title=None):
22     fig = px.scatter(x=df[var1],y=df[var2],color=df[method].astype(str), symbol=tad[method].astype(str))
23     fig.update_traces(marker=dict(size=5,
24                                     line=dict(width=0.2,
25                                               color="white"),
26                                     selector=dict(mode='markers')))
27
28     if title:
29         fig.update_layout(height=500, width=750, title=title)
30         fig.show()
31     else:
32         fig.update_layout(height=500, width=750)
33         fig.show()
34
35 def inertia(df, min_, max_):
36
37     x = [i for i in range(min_, max_)]
38     inertia_ = []
39
40     for k in x:
41         km = KMeans(n_clusters=k)
42         km.fit(df)
43         inertia_.append(km.inertia_)
44
45     sns.lineplot(
46         x=x, y=inertia_,
47         marker="o"
48     )
49
50 def pruebas_hipotesis(df1, df2, col_list):
51     p_values = []
52     dec = []
53
54     for col in col_list:
55         stat, p_value = kruskal(df1[col], df2[col])
56         p_values.append(p_value)
57
58         if p_value <= 0.05:
59             decision = 'Distribución diferente'
60         else:
61             decision = 'Distribución parecida'
62         dec.append(decision)
63
64     return pd.DataFrame(data = {'variables': col_list, 'p_value': p_values, 'decision': dec})
```



