



Facultad de Estudios Superiores: Acatlán.

Diplomado en Ciencia de Datos

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

Valencia Martínez Jesús Adolfo

# 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:

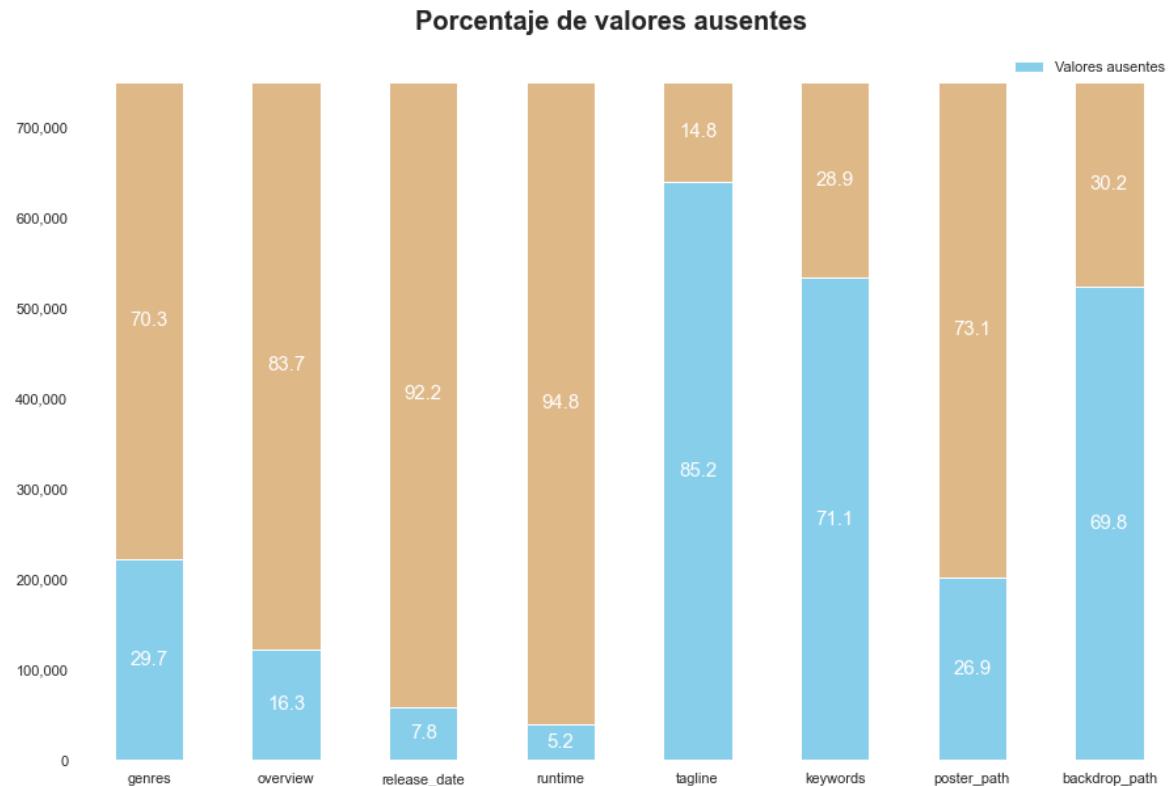
<i>Característica</i>	<i>Descripción</i>
<i>id</i>	Identificador único en la página de TMDb
<i>title</i>	Nombre de la película
<i>genres</i>	Géneros relacionados
<i>original_language</i>	Idioma original
<i>overview</i>	Resumen o sinopsis
<i>popularity</i>	Métrica de TMDb para medir la popularidad dentro 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

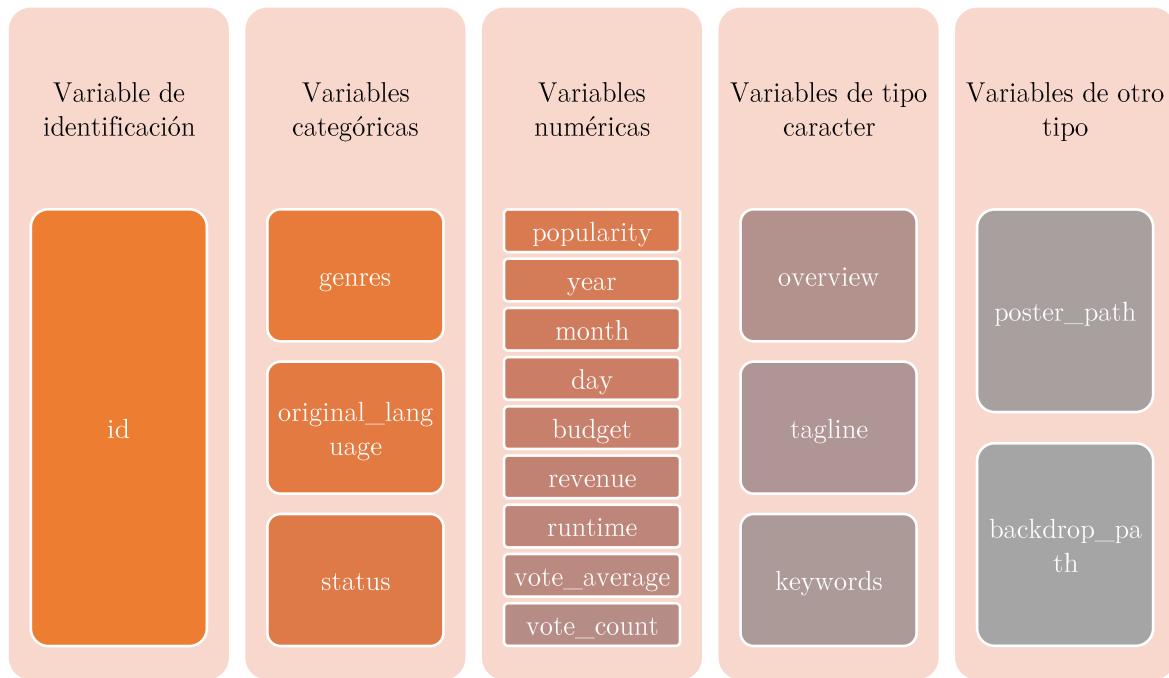
Realizando la exploración de nuestros datos, se pudo observar que varias películas no cuentan con información de algunas características. Las características que no cuentan con algunos registros se presentan en el siguiente gráfico, donde, a su vez, se detalla el porcentaje de valores ausentes respecto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

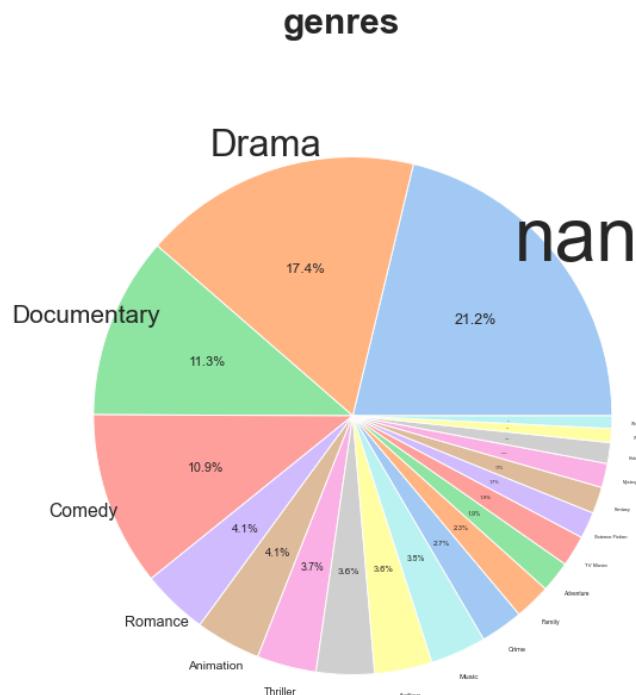
Una vez explicado el punto referente a los valores ausentes, las variables de nuestra base de datos se segmentaron de acuerdo a su tipo. La clasificación es la siguiente:



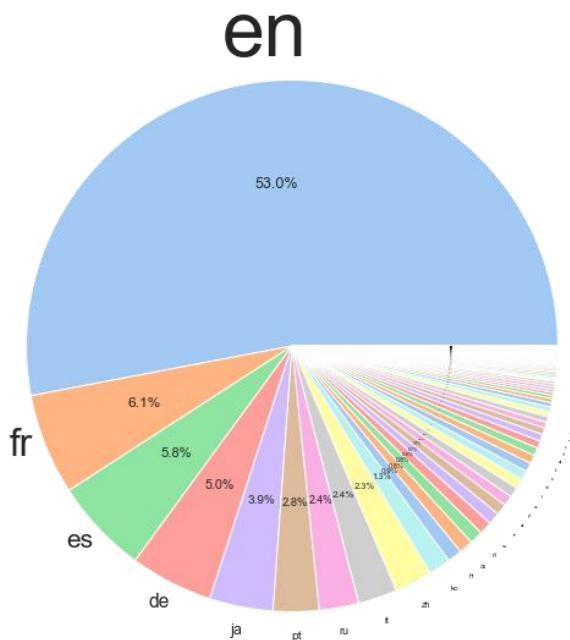
En los gráficos a continuación expuestos, se presenta un breve resumen de las características descritas

## 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.

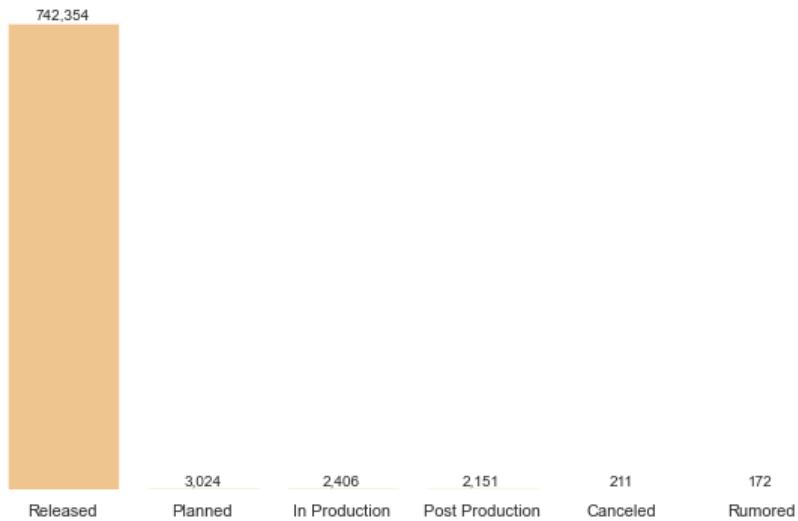


## original\_language



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.

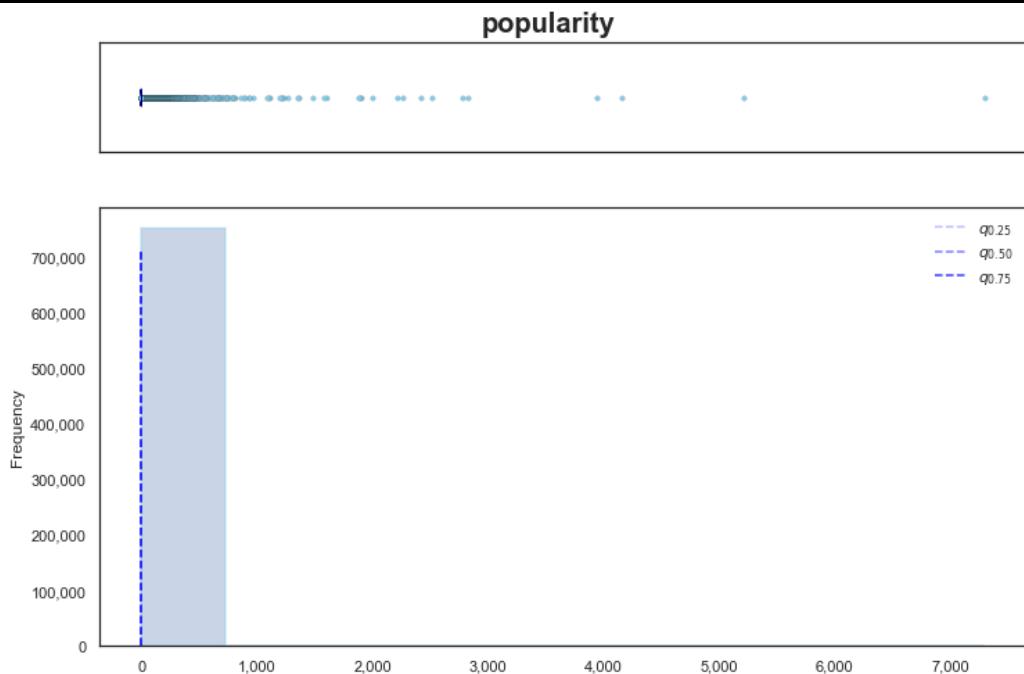
## status

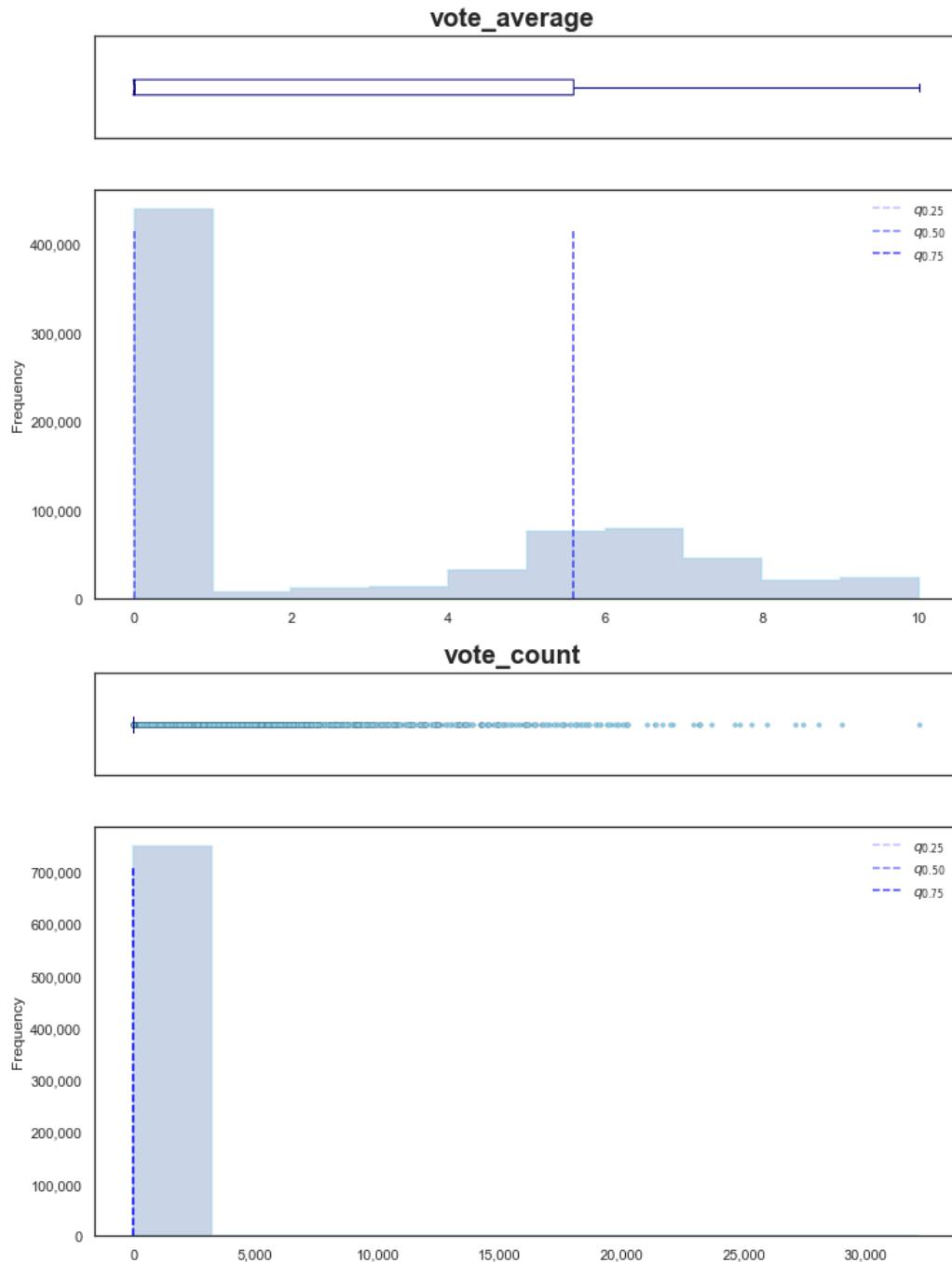


Finalmente, para la variable del estado de la película, es bastante evidente que la información prácticamente se basa en las películas que ya han sido estrenadas

## Variables numéricas

*Referentes a opiniones del usuario*

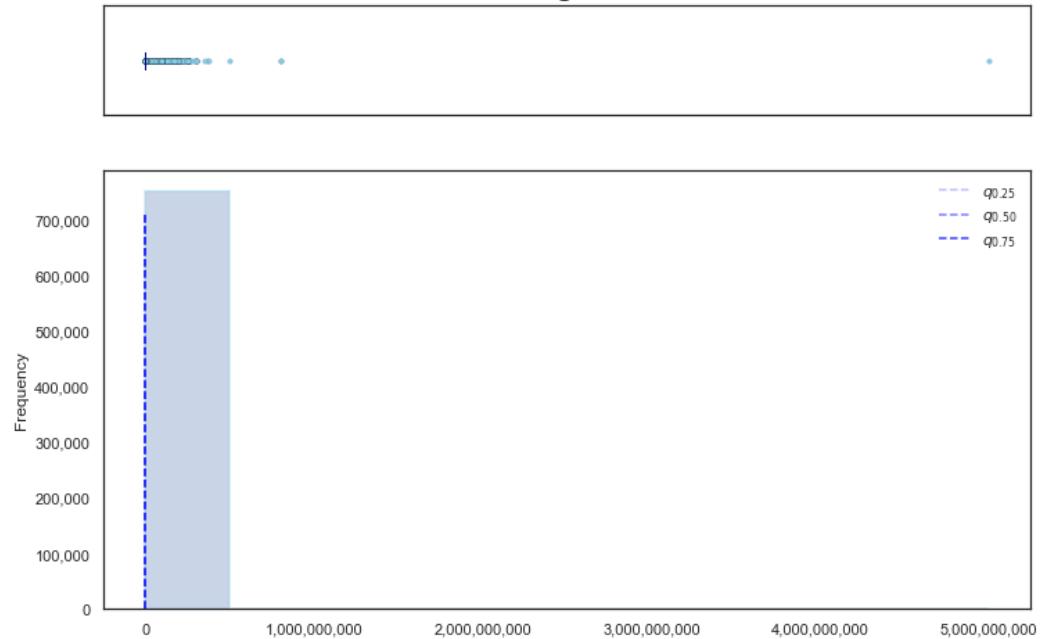




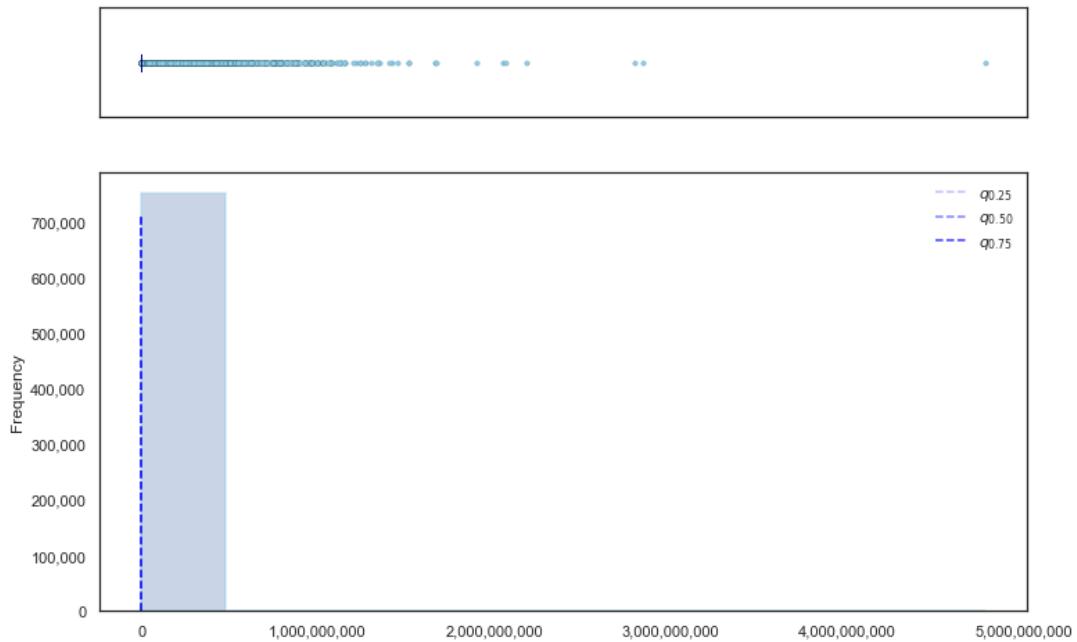
Las gráficas anteriores exhiben una carga de la distribución al valor 0. Esto es consecuencia a que existe una cantidad considerable de películas que, si bien, no cuentan con valores ausentes, los datos se registran como 0 y, por lo tanto, esto influye en la calificación promedio y en la métrica de popularidad.

## *Referentes cifras monetarias*

**budget**

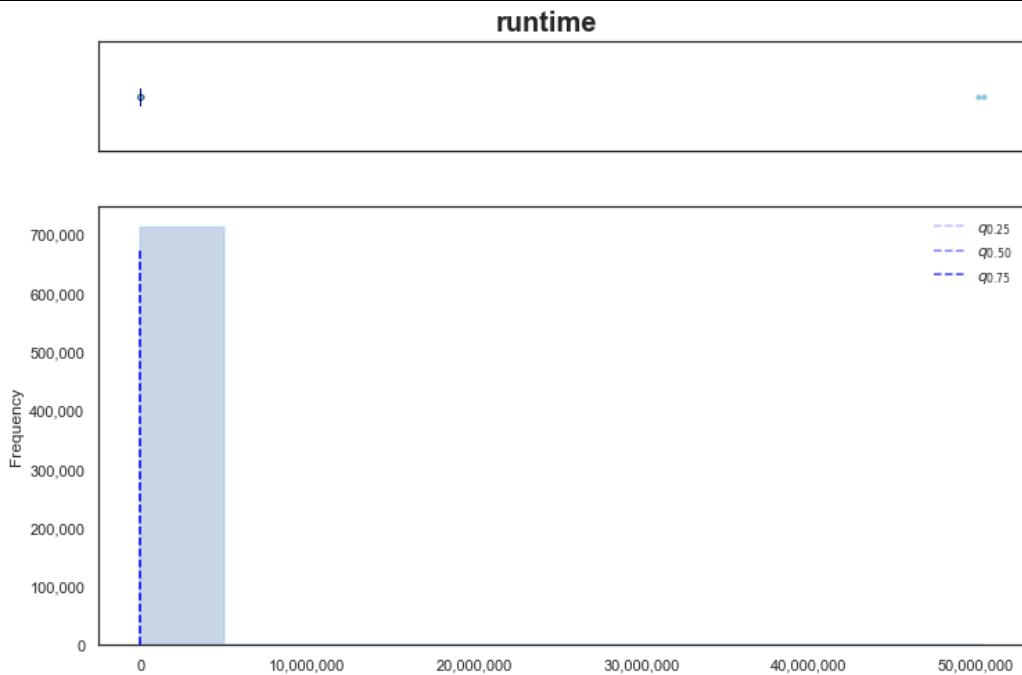


**revenue**



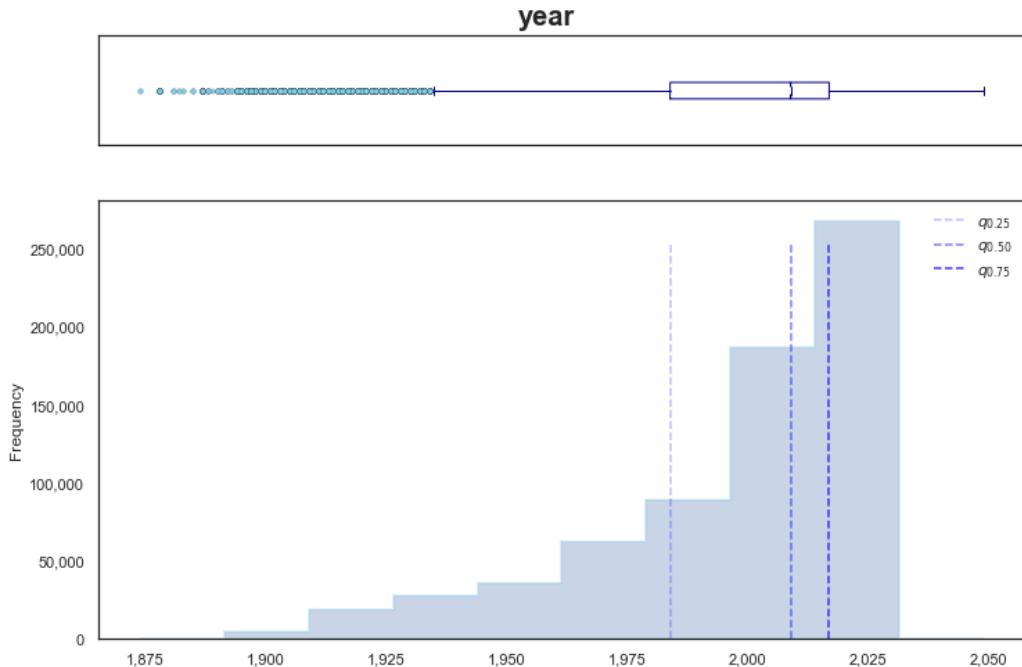
El comportamiento en estas variables es similar a las anteriores, si bien, no hay valores ausentes, su carga hacia el 0 se debe a que se registra la información como 0 al no contar con información

## *Referentes información de la película*

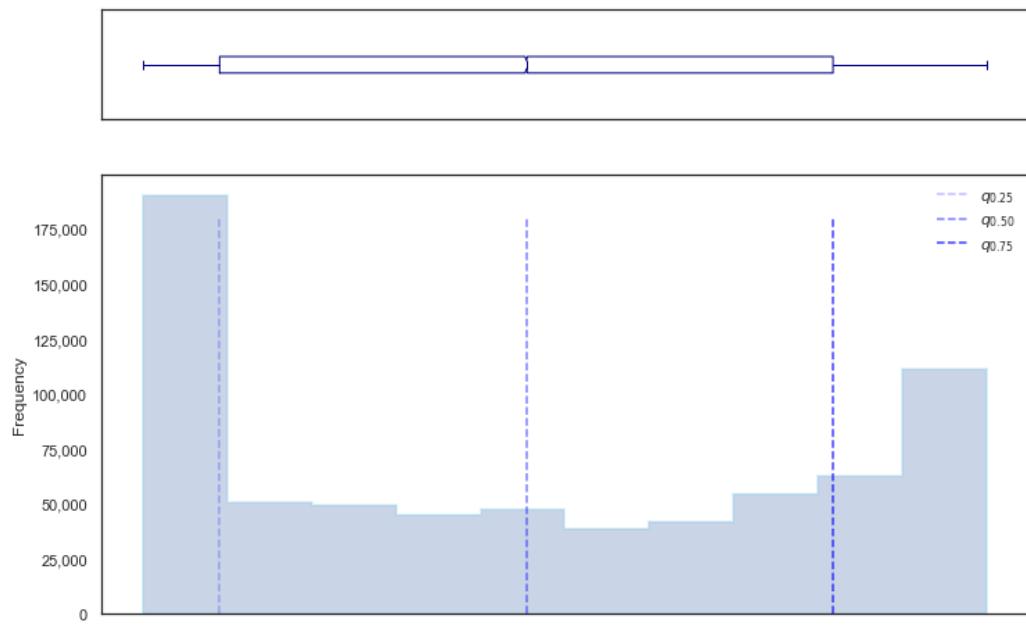


En este gráfico se puede apreciar la evidencia de un claro valor atípico o bien, una captura errónea que provoca un sesgo enorme en la distribución.

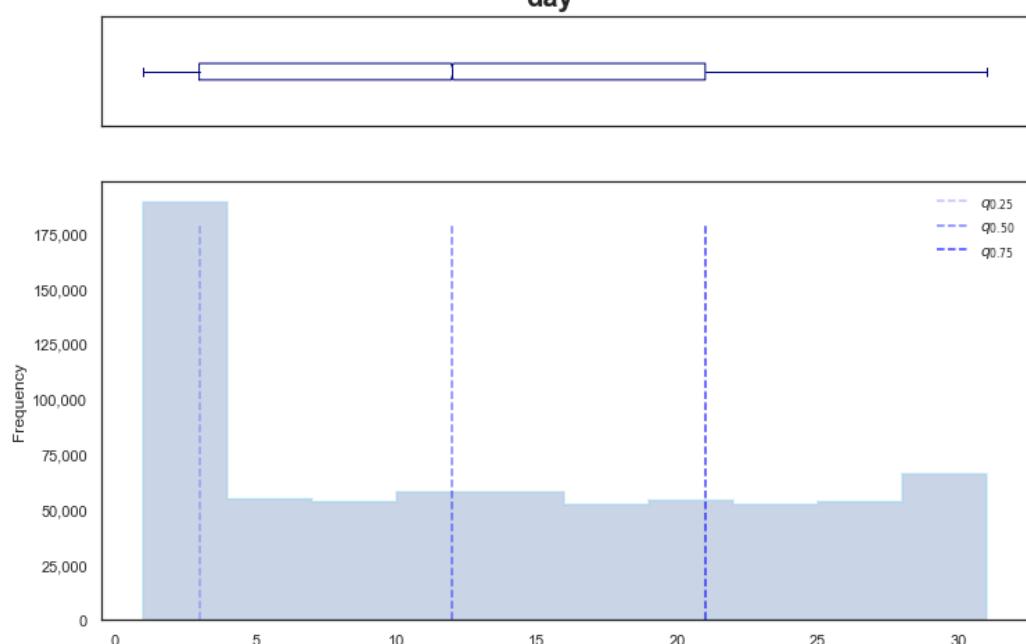
Las siguientes, son las distribuciones de la fecha de lanzamiento de las películas en la base:



**month**



**day**

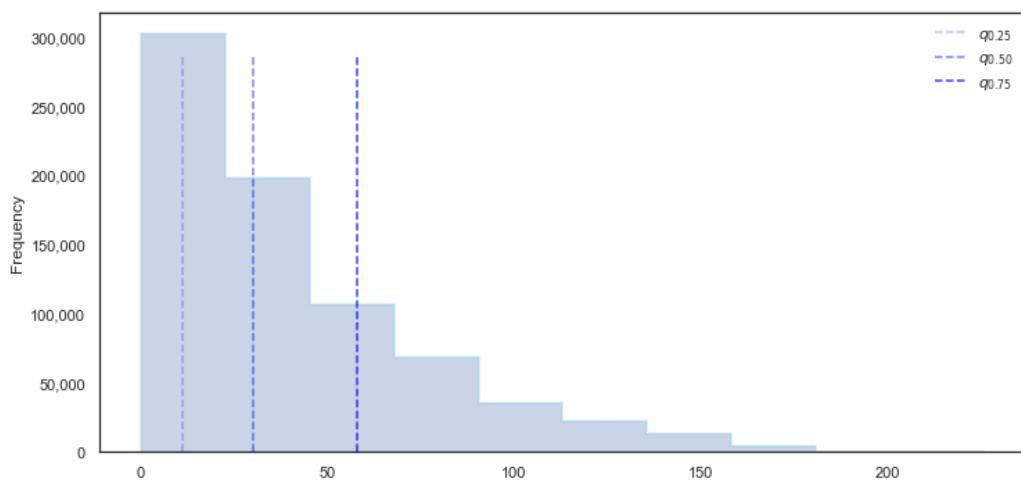
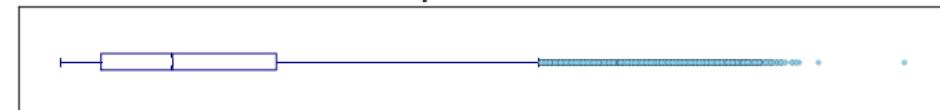


## Variables de tipo carácter

*overview*

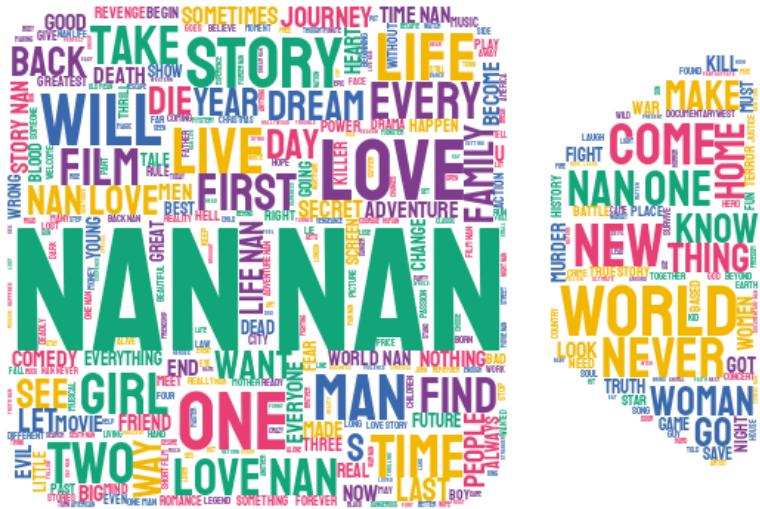


Cantidad de palabras : overview

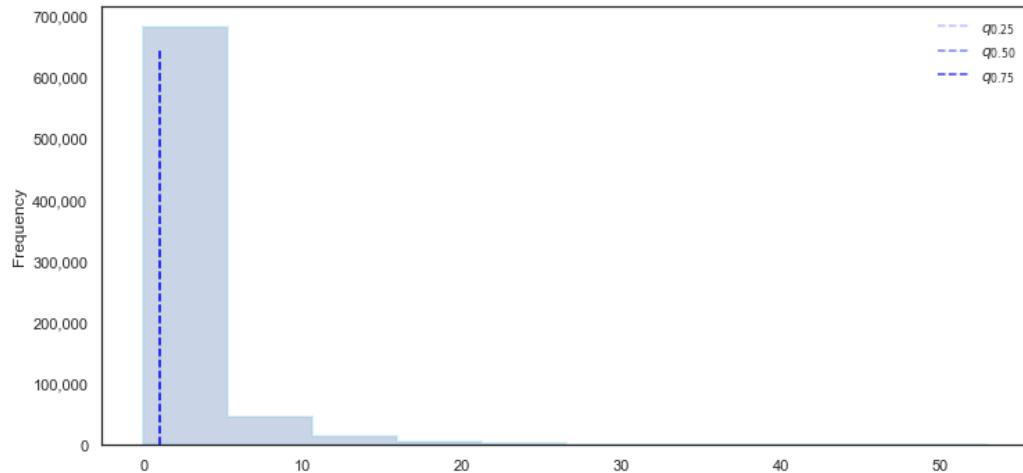


*tagline*

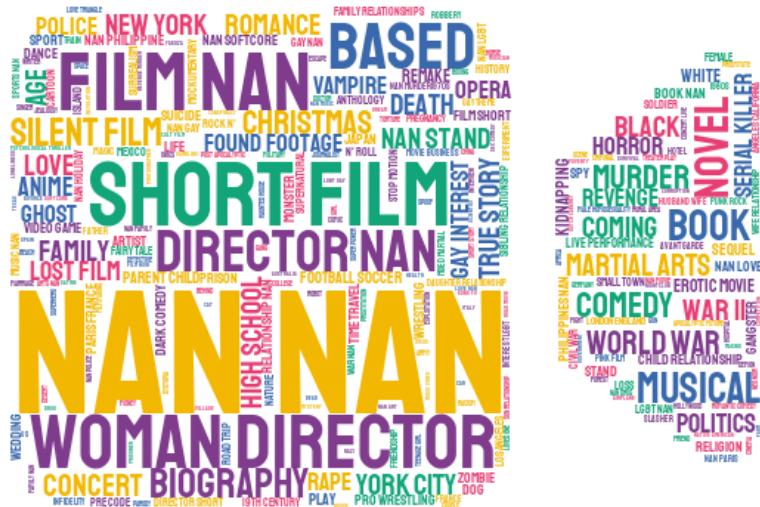
---



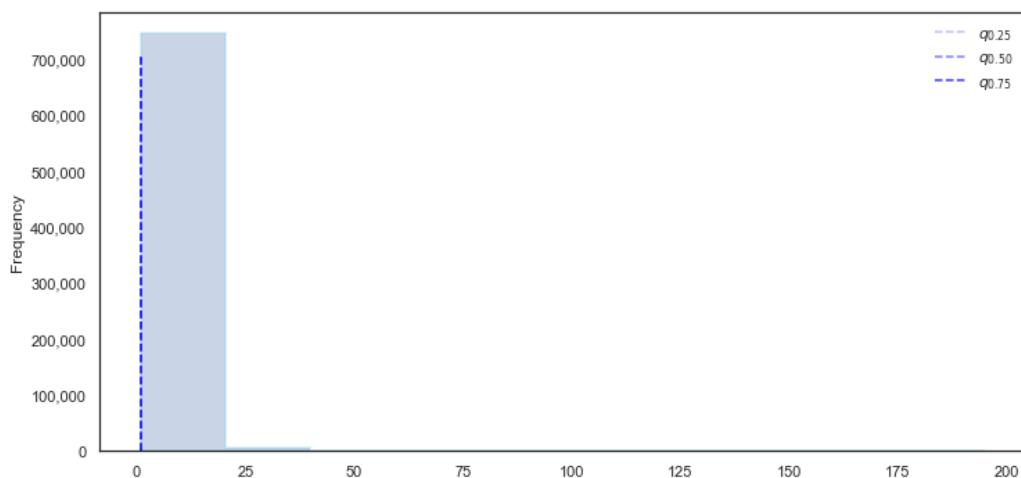
Cantidad de palabras : tagline



### *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.**

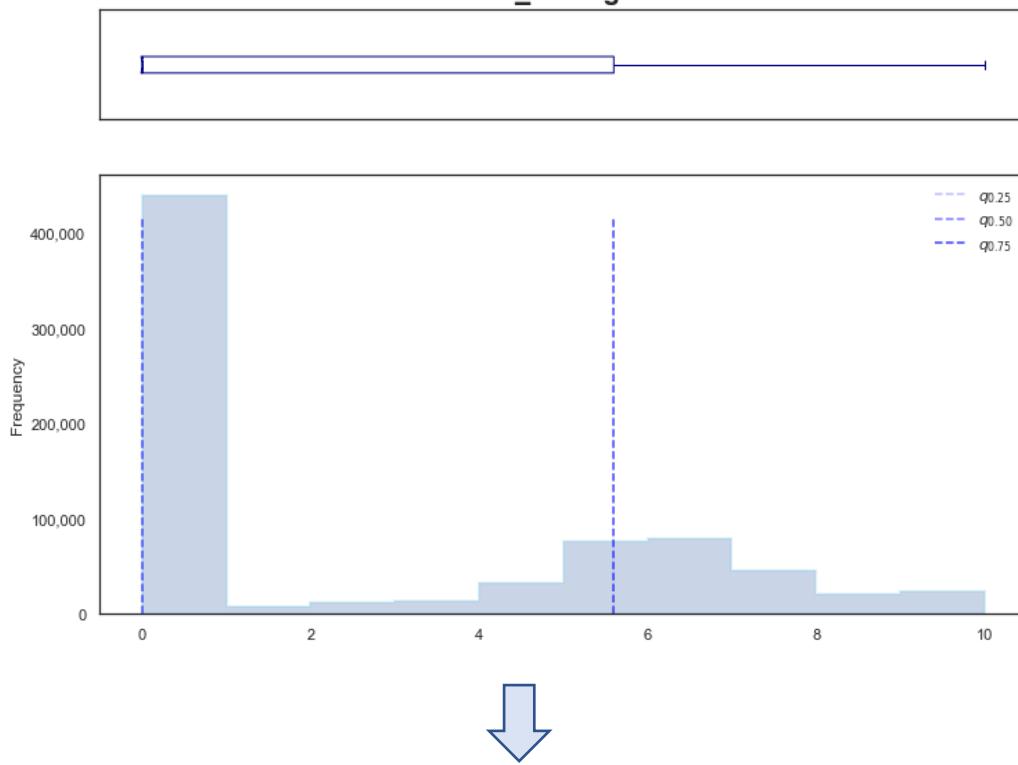
En el siguiente gráfico podemos ver la cantidad de películas, que, de acuerdo a nuestra segregación, se clasifican como una película buena o mala:



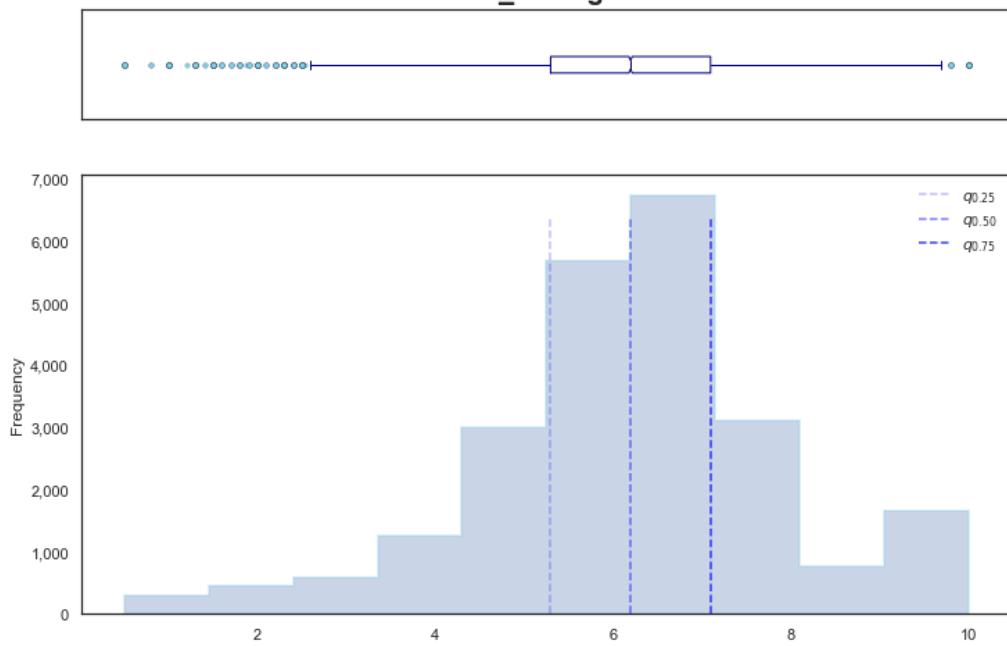
Adicional, se muestran los cambios significativos que sufren las distribuciones al realizar los filtrados.



**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:

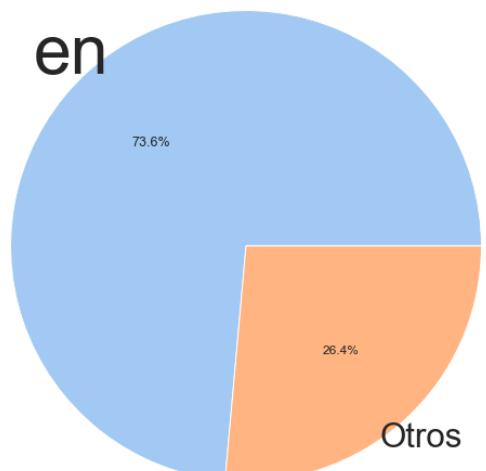
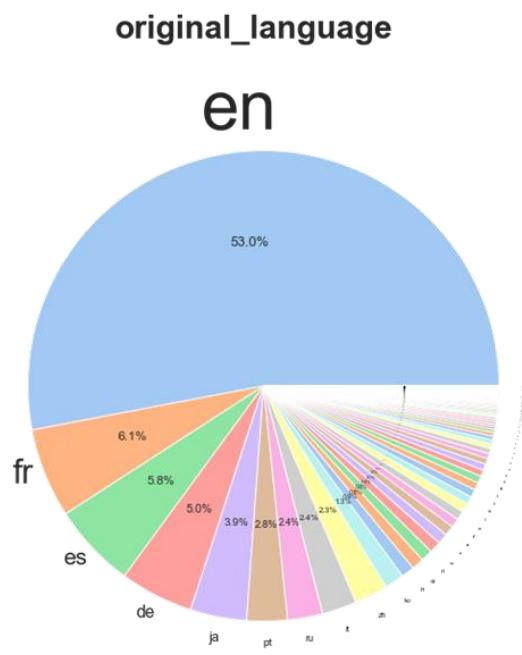
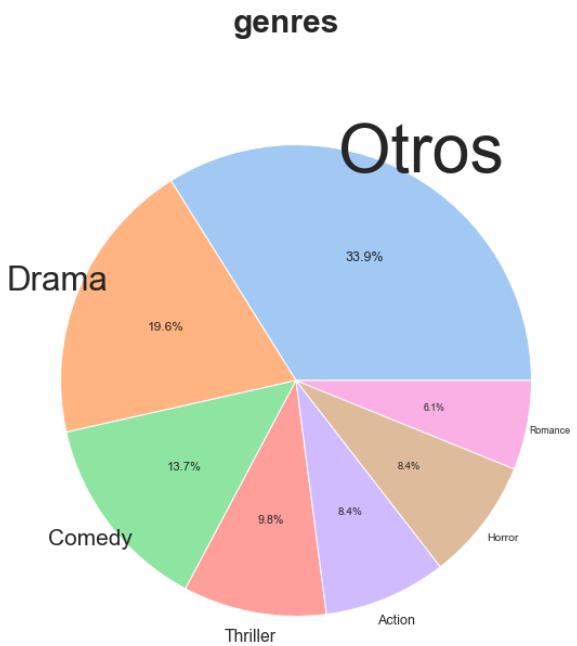
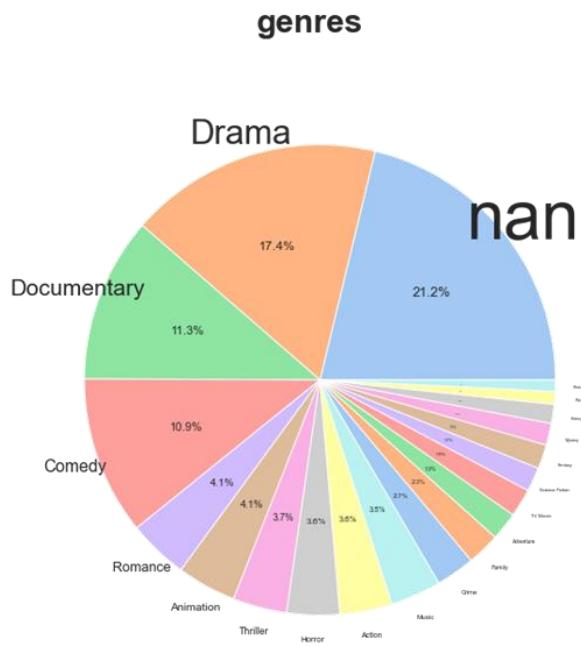


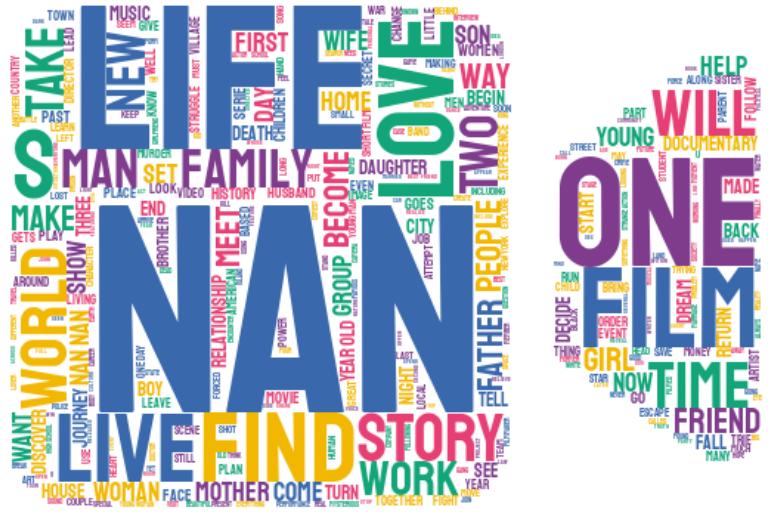
Finalmente, como consideraciones adicionales para poder realizar los entrenamientos se realizaron los siguientes pasos:

- 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.





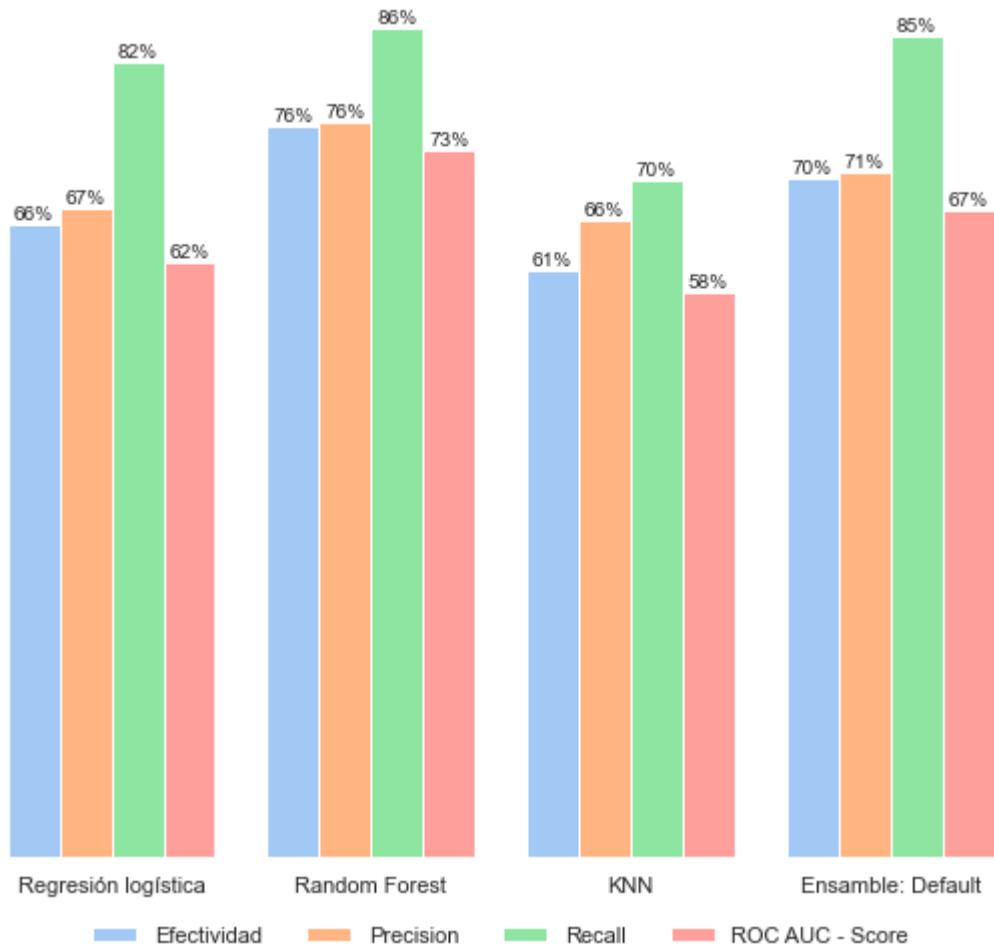
## Resultados

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.

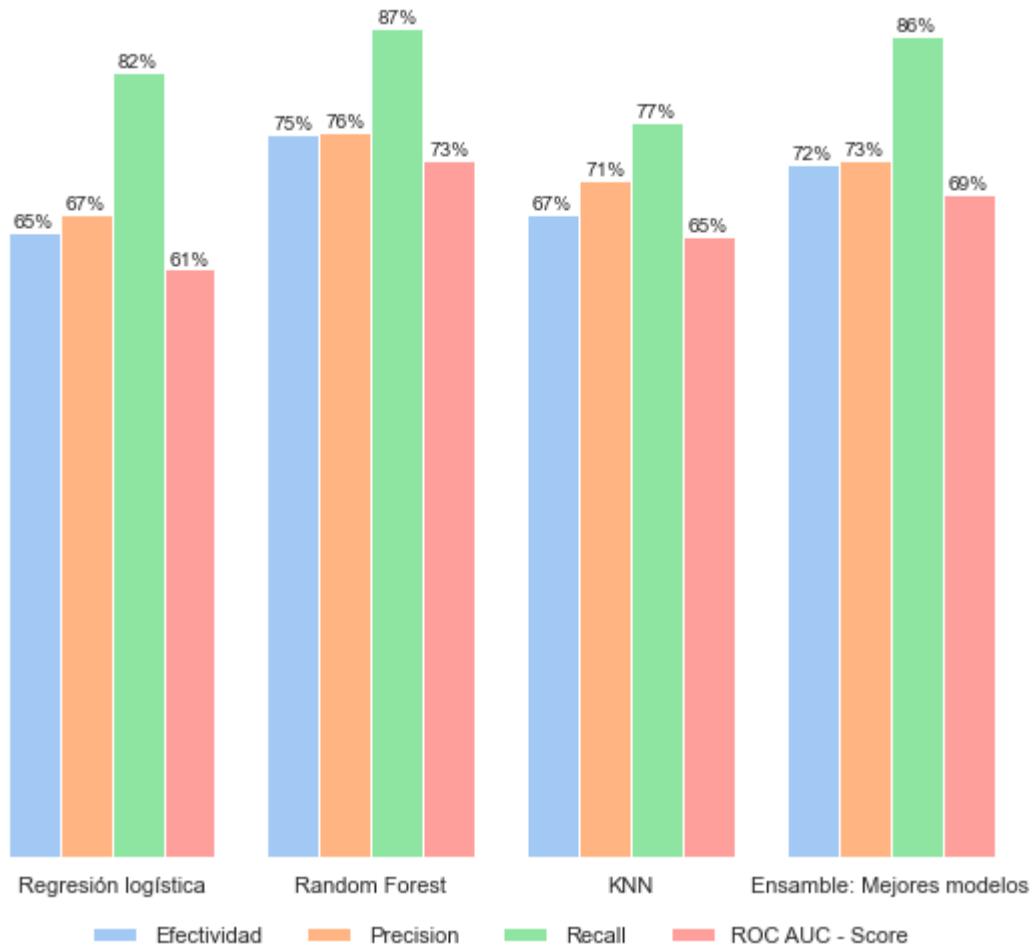
Es importante mencionar que se optó por elegir 100 features resultantes en cada word embedding y que las métricas expuestas fueron obtenidas evaluando los modelos en el conjunto de validación.

## 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

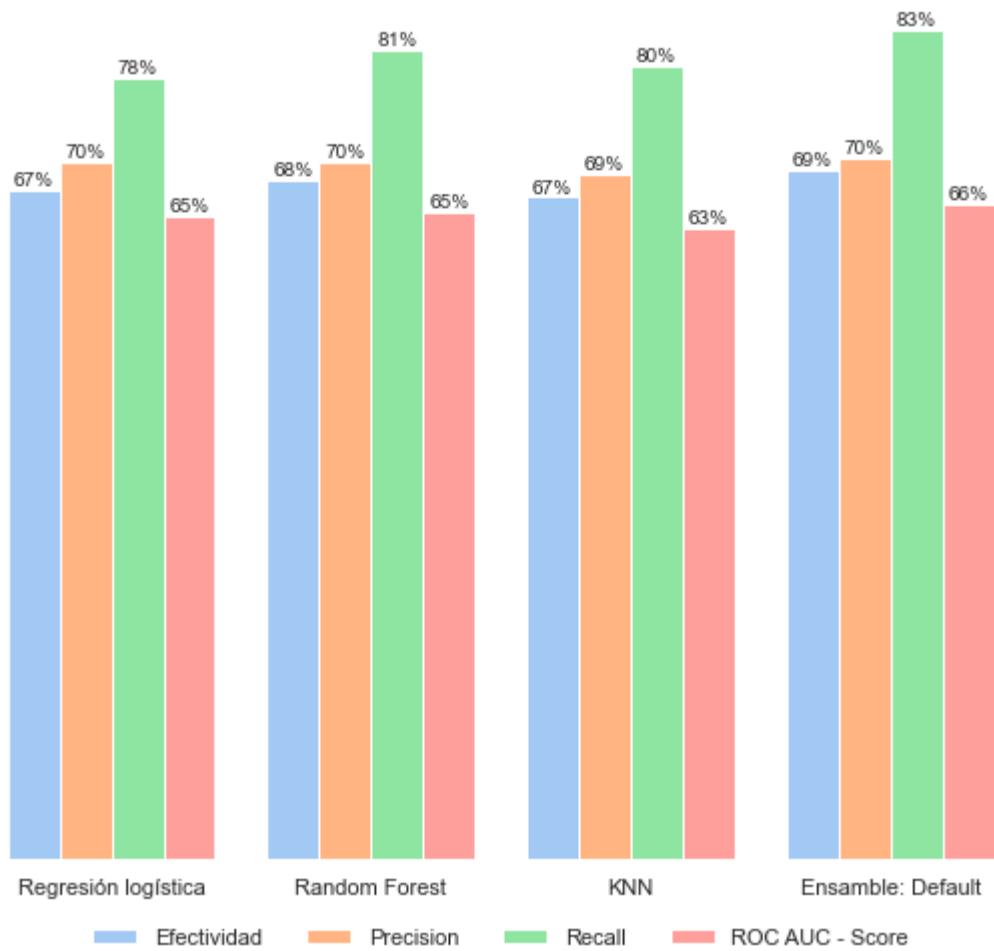


Mejores modelos				
	Regresión logística	Random Forest	KNN	Ensamble
Efectividad	0.6528	0.7540	0.6709	0.7229
Precision	0.6709	0.7564	0.7062	0.7267
Recall	0.8188	0.8658	0.7664	0.8571
ROC AUC	0.6135	0.7275	0.6483	0.6911
Parámetros	C = 3.4568	max_depth = 50 min_samples_split = 4 n_estimators = 200	leaf_size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

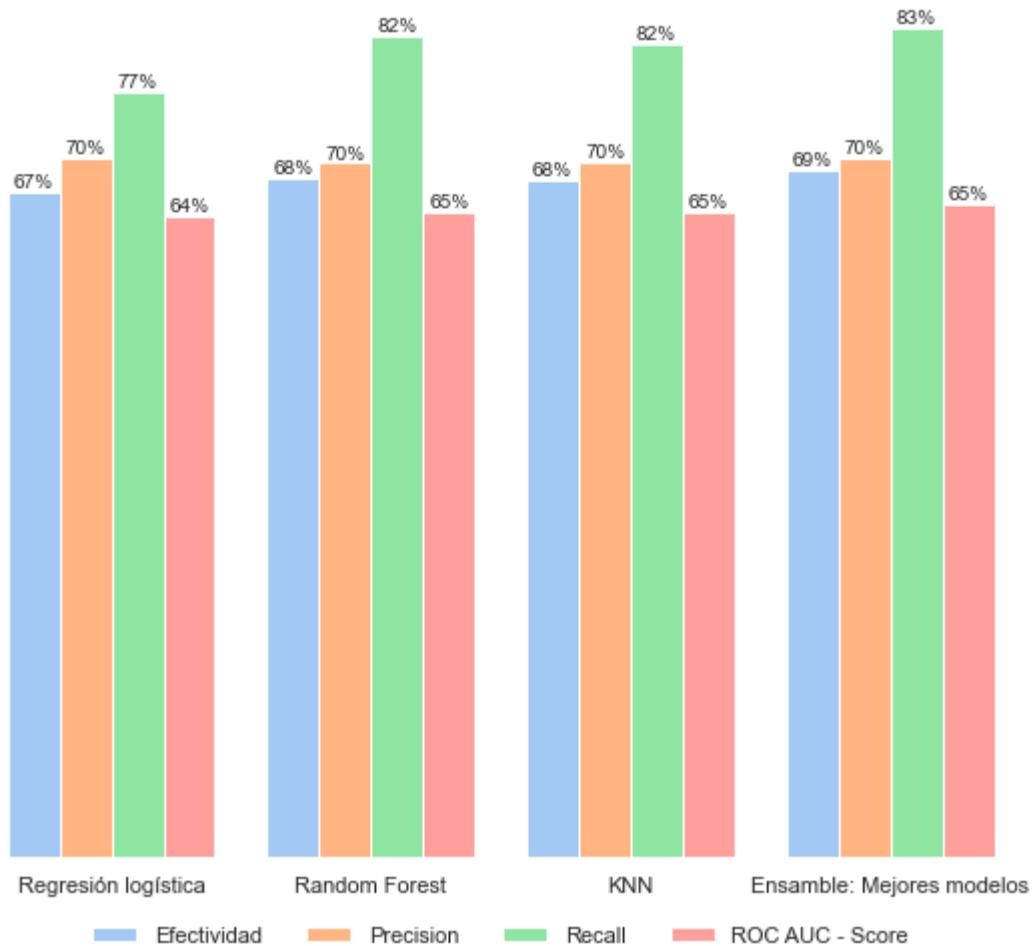


## Modelos ocupando Doc2Vec.

		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	

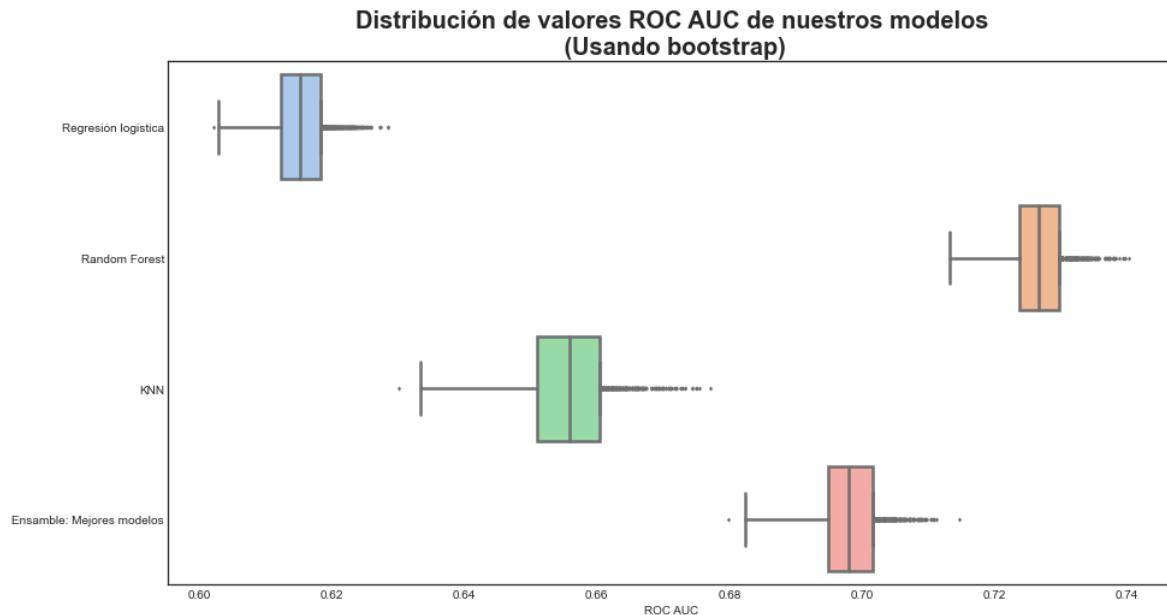


## Estabilidad

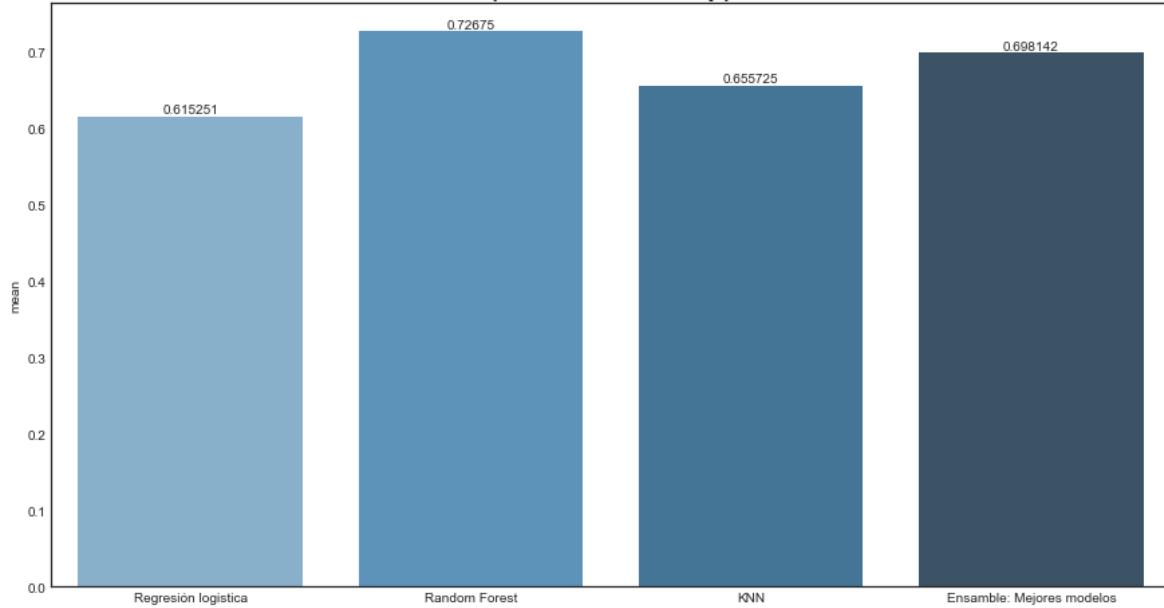
A manera de verificar que nuestros modelos estén brindando métricas confiables y poder generar una elección de modelo, se realizó un proceso de bootstrap con los modelos obtenidos de hacer el tuning de parámetros.

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.

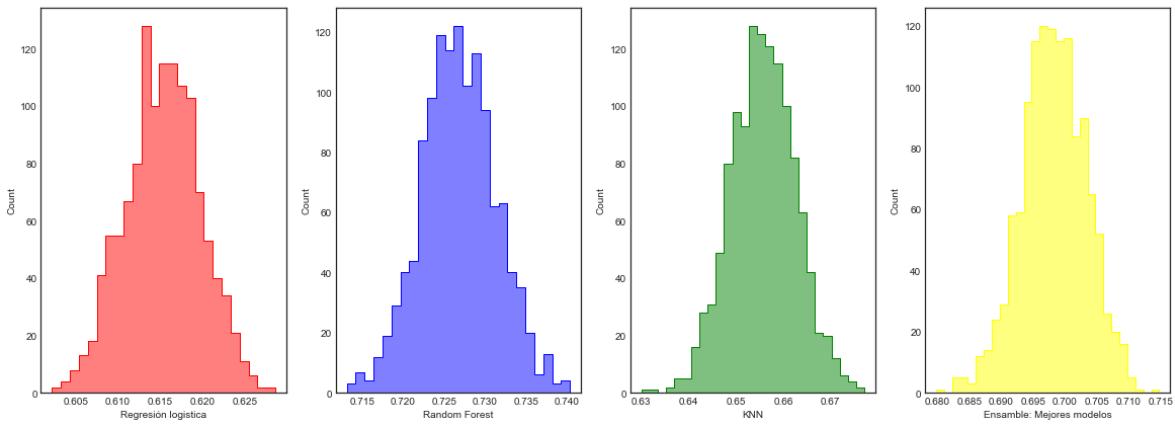
Los siguientes resultados recuperan el valor ROC AUC de cada modelo elegido realizando un total 1,250 iteraciones bootstrap que tomaron alrededor de 9 hrs de cómputo:



### 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:

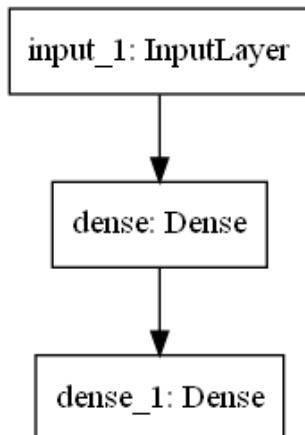
- max\_depth = 50
- min\_samples\_split = 4
- n\_estimators = 200

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

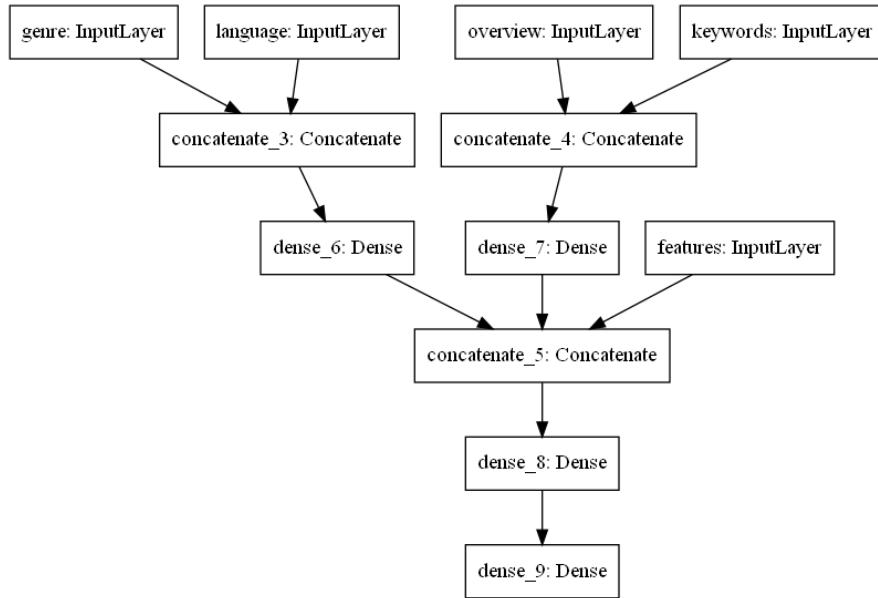
## Redes neuronales

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:

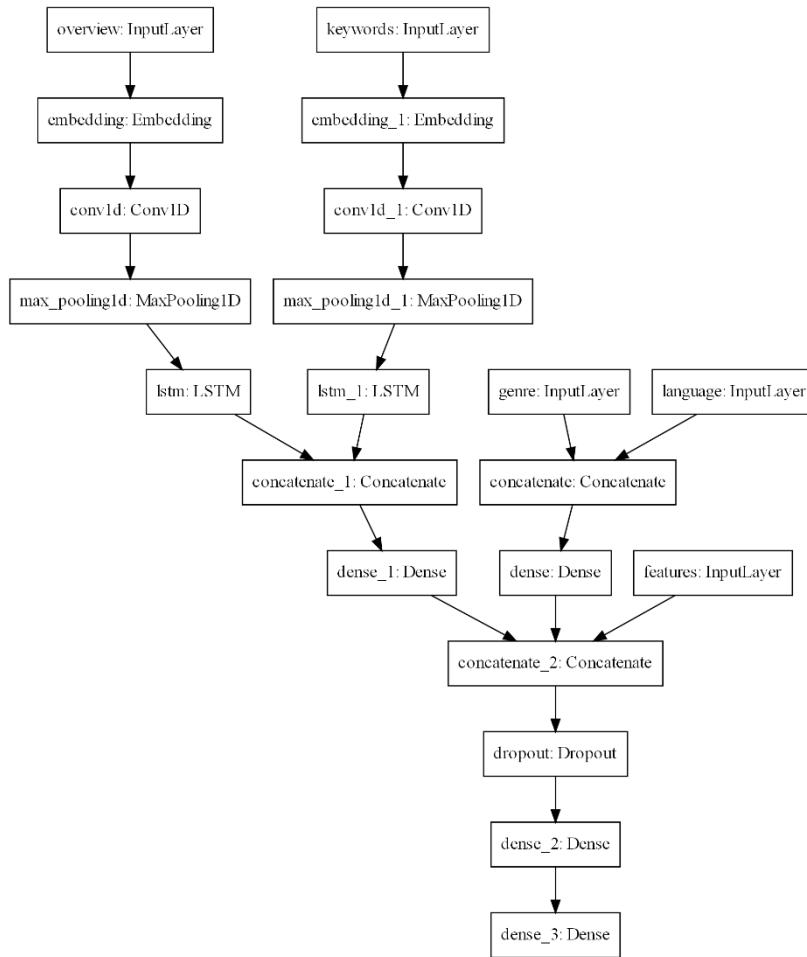
Estructura 1:



Estructura 2:



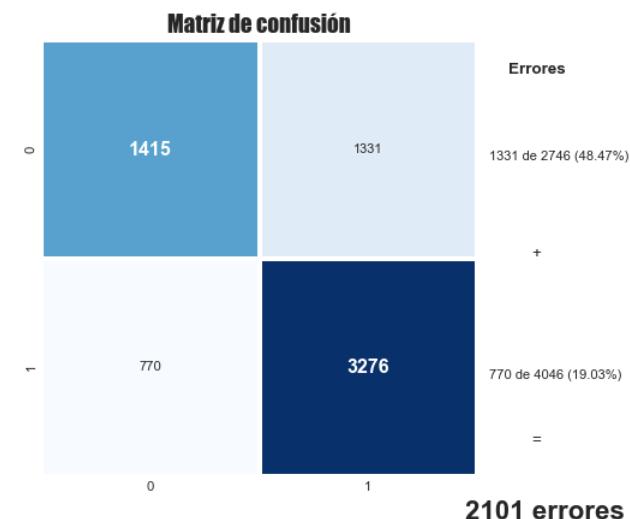
Estructura 3:



## Resultados

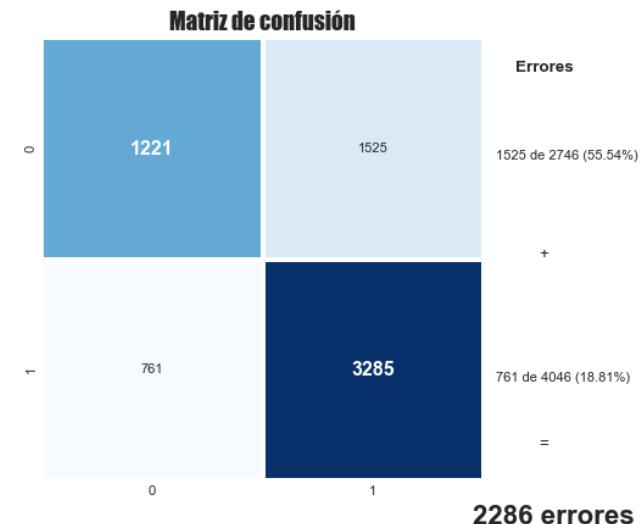
### Redes neuronales (Estructura 1)

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



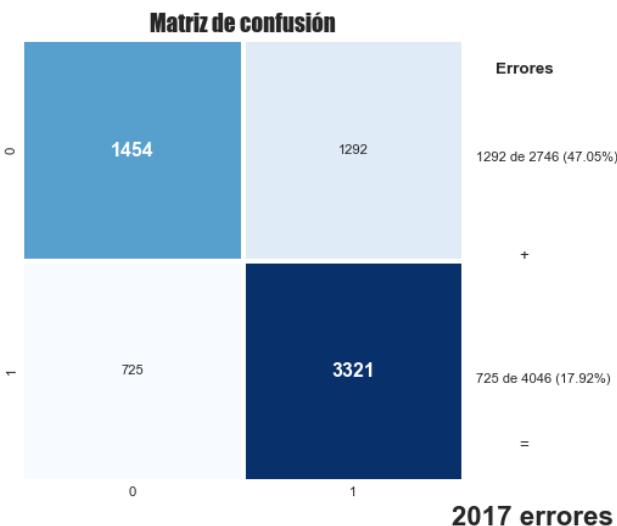
### Redes neuronales (Estructura 2)

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

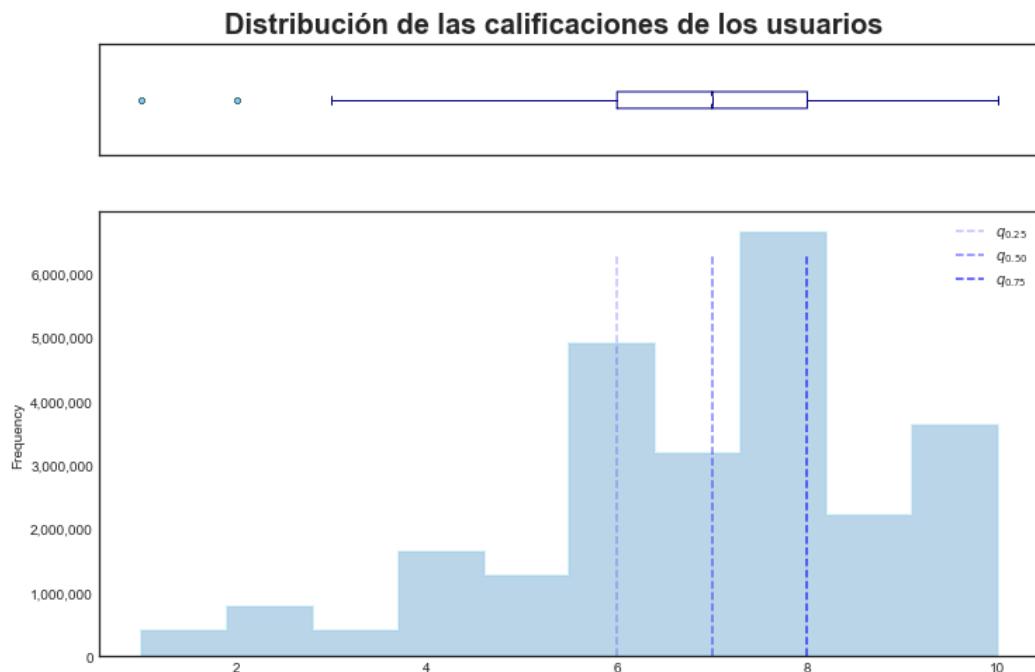
La información que contiene esta base de datos consolida la siguiente información:



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.

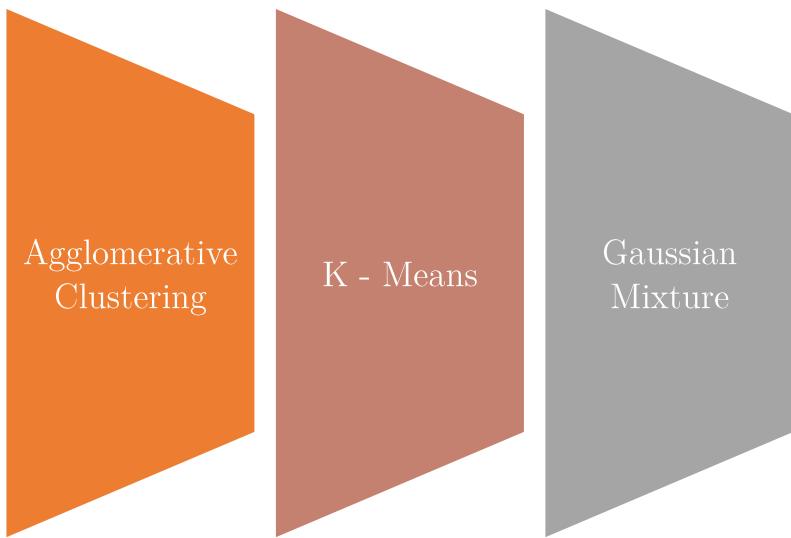
Como resultado, las variables finales de insumo de cada para generar nuestros modelos serían las siguientes:

- 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:

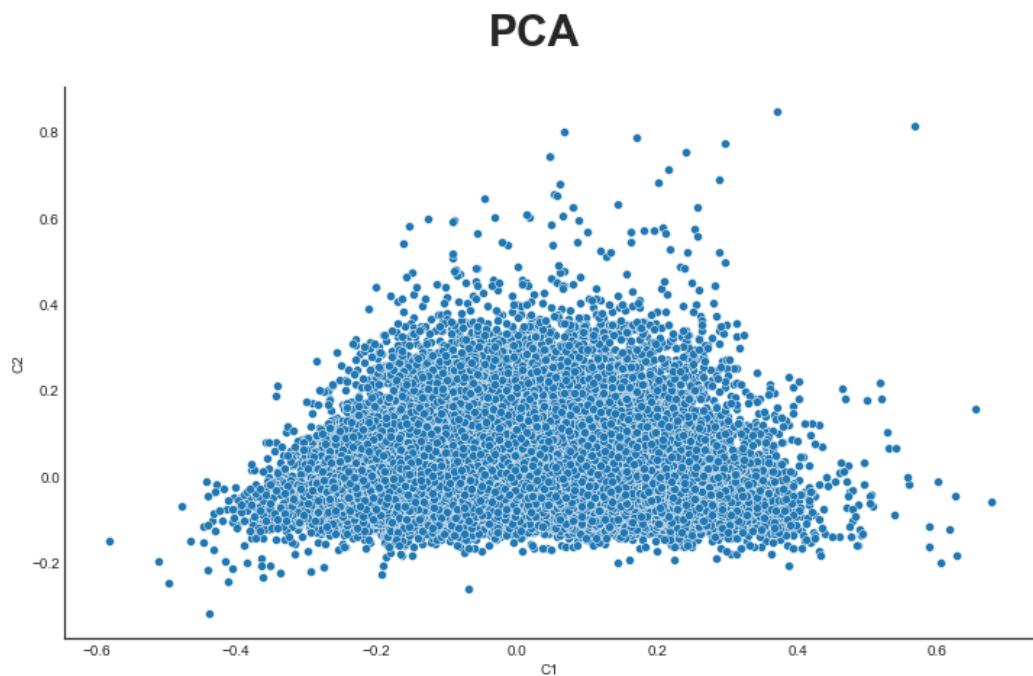


## Resultados

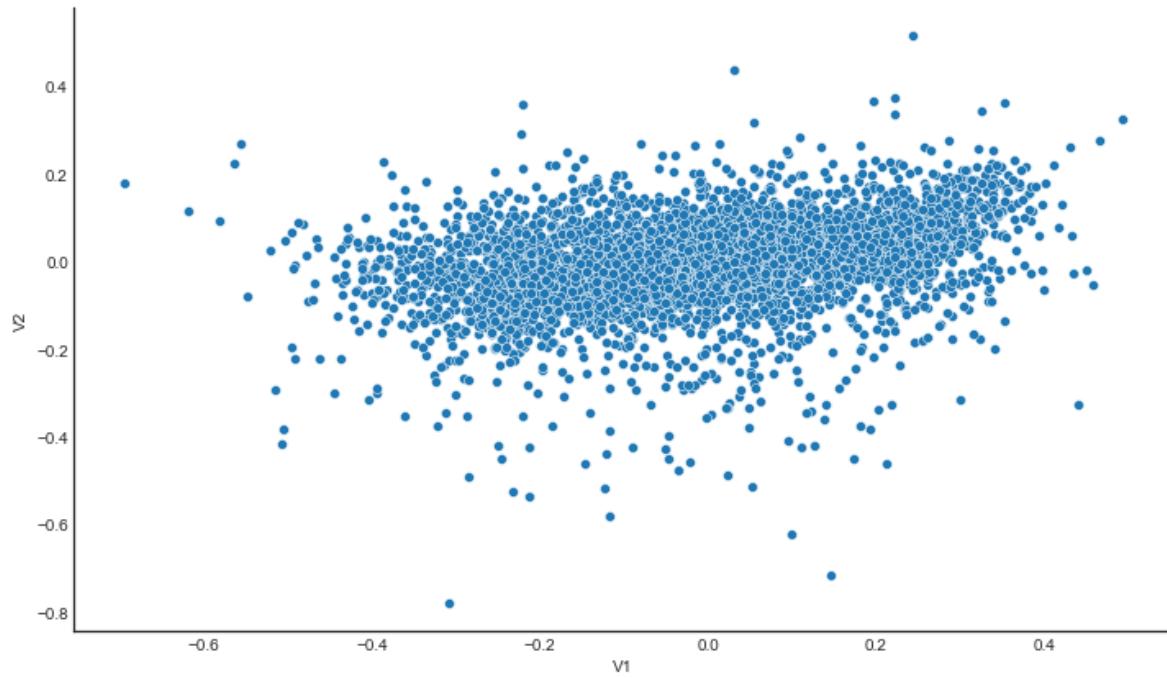
A continuación, se presentan los resultados de los entrenamientos de los modelos anterior propuestos.

### 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.

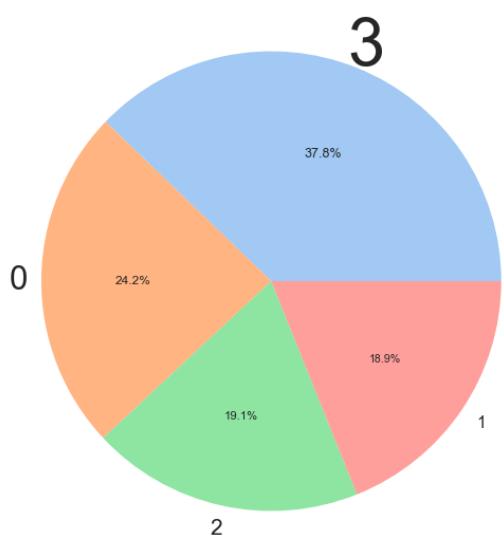
De esta manera, se realizaron diversos entrenamientos, probando cantidades diferentes del número de segmentaciones haciendo uso de los algoritmos de K-Means y Gaussian Mixture. El algoritmo agglomerative clustering presentaba fallos al requerir un tamaño de memoria excesivo, por lo que las métricas correspondientes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

Ocupando 4 clústers

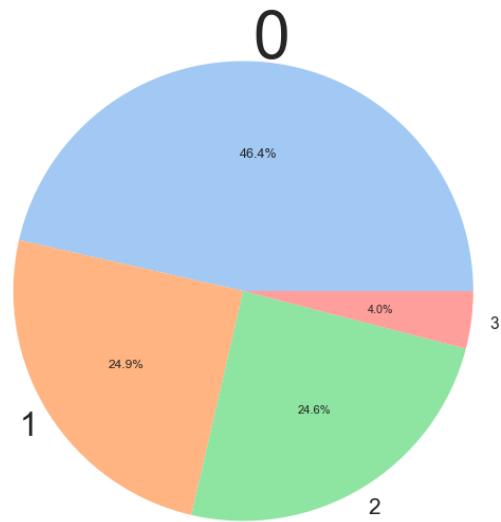
Silhouette score		
4 clústers		
Agglomerative clustering*	K-Means	Gaussian Mixture
0.2362	0.3591	-0.0852

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

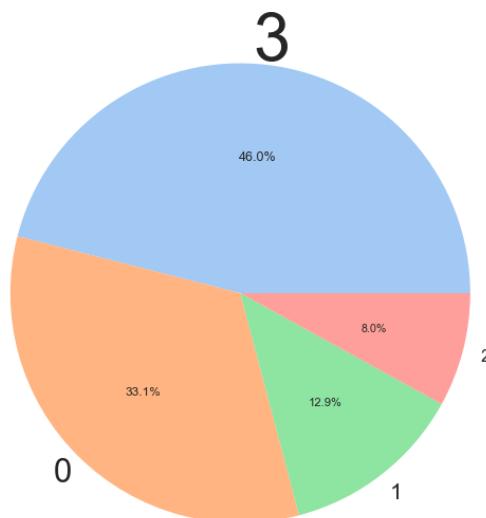
**Método: Ward**



**Método: KMeans**



**Método: GM**

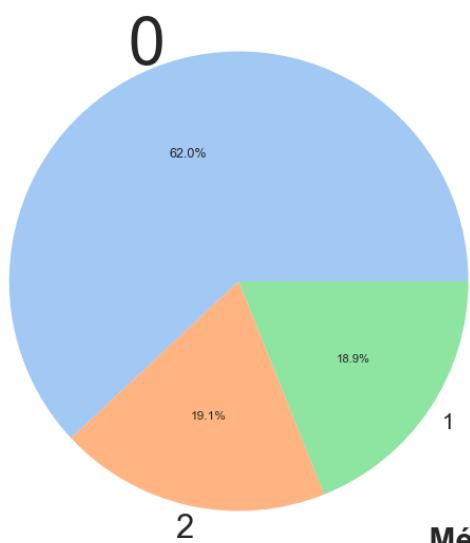


Ocupando 3 clústers

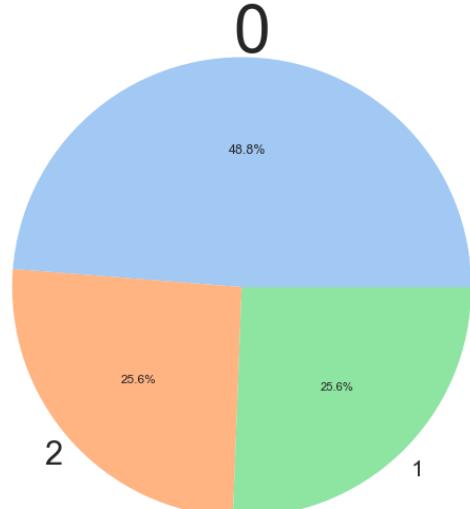
Silhouette score		
3 clústers		
Agglomerative clustering*	K-Means	Gaussian Mixture
0.2970	0.3476	-0.0642

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

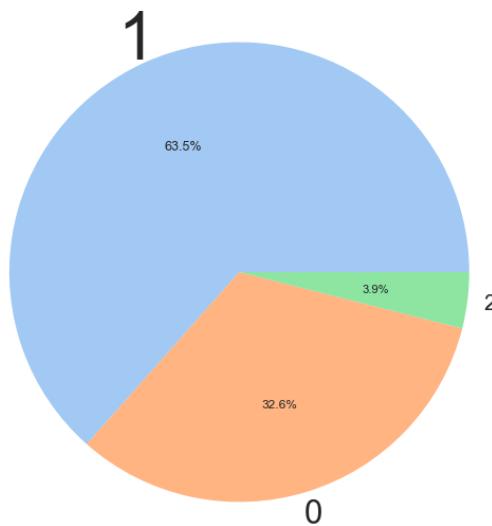
**Método: Ward**



**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:

<i>Prueba</i>	<i>Uso</i>
<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.

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

## Apéndice

```

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([0],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         else:
34             g.bar_label(g.containers[0],rotation=barlabelrot)
35
36     else:
37         pass
38     ax.tick_params(labelrotation=rot)
39     ax.set_yticks([])
40
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 barlabelrot:
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 barlabelrot:
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
106
107 def hist(db,variable,ctitle=None,min=10,logx=False,logy=False):
108
109     plt.figure(figsize=(8,9))
110
111     db[variable].plot(kind="hist", logx=logx, logy=logy,bins=min, histtype="stepfilled",alpha=0.3, ec="k",edgecolor="skyblue", linewidth=2 ,label="nolegend_")
112
113     plt.gca().xaxis.set_major_formatter(tkr.FuncFormatter(lambda v, p: format(sum(roundy,0), ',')))
114
115     plt.gca().yaxis.set_major_formatter(tkr.FuncFormatter(lambda v, p: format(int(roundy,0)), ','))

116     plt.grid(False)
117
118     plt.axvline(x=db[variable].quantile(0.25),color='blue',alpha=0.25, ymax=0.00,ls="--")
119     plt.axvline(x=db[variable].quantile(0.5),color='blue',alpha=0.5, ymax=0.00,ls="--")
120     plt.axvline(x=db[variable].quantile(0.75),color='blue',alpha=0.75, ymax=0.00,ls="--")
121
122     plt.legend(['q1(0.25)', 'q2(0.50)', 'q3(0.75)'])
123
124     plt.suptitle(title,size=30,fontweight='bold') if cttitle else plt.suptitle(variable,size=30,fontweight='bold',y=0.02)
125
126
127 def hist_box(db, variable,ctitle=None,min=10,logx=False,logy=False):
128     f, (ex_box, ex_hist) = plt.subplots(2,figsize=(10,8), sharex=True, gridspec_kw={'height_ratios': [1.2, .88]})

129     db[variable].plot(kind="hist", logx=logx, logy=logy,bins=min, histtype="stepfilled",alpha=0.3, ec="k",edgecolor="skyblue", linewidth=2 ,label="nolegend_")
130     db[variable].plot(kind="box", logx=logx, logy=logy,notch=True,log=logy,whiskerprops = dict(marker='.', markerfacecolor='yellow', markersize=7,linestyle = ':',markeredgecolor='black',markerewidth=0.1))

131     plt.gca().xaxis.set_major_formatter(tkr.FuncFormatter(lambda v, p: format(int(roundy,0)), ','))

132     plt.gca().yaxis.set_major_formatter(tkr.FuncFormatter(lambda v, p: format(int(roundy,0)), ','))

133     plt.grid(False)
134
135     plt.axvline(x=db[variable].quantile(0.25),color='blue',alpha=0.25, ymax=0.00,ls="--")
136     plt.axvline(x=db[variable].quantile(0.5),color='blue',alpha=0.5, ymax=0.00,ls="--")
137     plt.axvline(x=db[variable].quantile(0.75),color='blue',alpha=0.75, ymax=0.00,ls="--")
138
139     plt.legend(['q1(0.25)', 'q2(0.50)', 'q3(0.75)'])
140
141     f.suptitle(ctitle,size=30,fontweight='bold',y=0.02) if cttitle else f.suptitle(variable,size=30,fontweight='bold',y=0.02)
142
143
144 def hist_by_var(db,var,by,normalize = False):
145     if var == by:
146         pass
147     else:
148         if normalize:
149             try:
150                 aux = pd.DataFrame(pd.cut(db[var],bins = 30)).set_axis([var],axis=1)
151                 aux[by] = db[by]
152                 aux = aux.value_counts().reset_index().pivot_table(index=var,columns = by,value=0,fill_value=0)
153                 aux = aux.T
154                 aux.columns.map(lambda x: x.left)
155                 aux[TOTAL] = aux.apply(sum, axis=1)
156                 aux = aux.apply(lambda x: x / sum(x), axis=1)
157                 aux = aux.T.plot(kind="bar",stacked=True,width=0.9)
158                 plt.locator_params(axis='x', nbins=6)
159                 plt.xticks(rotation=90)
160
161             plt.show()
162         except Exception as e:
163             print(e)
164             aux = db[[var]]
165             aux[by] = db[by]
166             aux = aux.value_counts().reset_index().pivot_table(index=var,columns = by,value=0,fill_value=0)
167             aux = aux.T
168             aux.columns.map(lambda x: x.left)
169             aux = aux.T.plot(kind="bar",stacked=True,width=0.9)
170
171             for c in aux.containers:
172                 labels = [v.get_height() if v.get_height() > 0 else 0 for v in c]
173                 ax_bar_label(c, labels=labels, label_type='center')
174
175             plt.show()
176
177         else:
178             try:
179                 aux = pd.DataFrame(pd.cut(db[var],bins = 30)).set_axis([var],axis=1)
180                 aux[by] = db[by]
181                 aux = aux.value_counts().reset_index().pivot_table(index=var,columns = by,value=0,fill_value=0)
182                 aux = aux.T
183                 aux.columns.map(lambda x: x.left)
184                 aux = aux.T.plot(kind="bar",stacked=True,width=0.9)
185
186                 for c in aux.containers:
187                     labels = [v.get_height() if v.get_height() > 0 else 0 for v in c]
188                     ax_bar_label(c, labels=labels, label_type='center')
189
190             plt.show()
191
192         except Exception as e:
193             print(e)
194             aux = db[[var]]
195             aux[by] = db[by]
196             aux = aux.value_counts().reset_index().pivot_table(index=var,columns = by,value=0,fill_value=0)
197             aux = aux.T
198             aux.columns.map(lambda x: x.left)
199             aux = aux.T.plot(kind="bar",stacked=True,width=0.9)
200
201             for c in aux.containers:
202                 labels = [v.get_height() if v.get_height() > 0 else 0 for v in c]
203                 ax_bar_label(c, labels=labels, label_type='center')
204
205             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 '{:,}{}'.format(str_number, 0)
17         else:
18             return '{:,.1f}'.format(str_number, 1)
19     except:
20         return str_number
21
22 def remove_accents(s):
23     return list(map(unidecode,s))
24
25 def clean_text():
26     clean_text = []
27     txt = txt.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,s))
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,y):
38     df['stringvar'] = y
39     df['stringvar'] = df['stringvar'].str.split().str.len()
40     df['stringvar'] = ' - ' + df['stringvar'].str.len() + df['stringvar'].str.len() / df['stringvar'] + ' - ' + 'n_words'
41     sid = SentimentIntensityAnalyzer()
42     vader_df=df[["stringvar"]].apply(sid.polarity_scores)
43     vader=pd.DataFrame(map(lambda x: vader_,x,vader_))
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 = pd.Series(self.vocab_.value_counts())
52         self.vocab_len = len(self.counter)
53         self.len_sentences = df[y].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 mas 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 = x
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(0.05,0.05, f'Precision', fontstyle='italic', fontsize=15, ha='center', fontweight='bold')
80     ax1.text(0.05,0.02, 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'Errors', fontstyle='italic', fontweight='bold')
90         ax3.text(0.1,0.15, f'({format(errors, ".1f")})', fontstyle='italic', fontweight='bold')
91         ax3.text(0.1,0.05, f'Errors', fontstyle='italic', fontweight='bold')
92         ax3.text(0.1,0.02, f'({format(errors, ".1f")})', fontstyle='italic', fontweight='bold')
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})
```



```

1 import os
2 import pickle
3 import imputils
4
5 import pandas as pd
6 import numpy as np
7
8 import confutils as cf
9 import imputilslib.clicker as tkr
10 import imputilslib.pickle as pkl
11 import imputilslib as pit
12
13 #import random as rd
14
15 #all_styling = cf.all_styling()
16 #pd.set_option('display.max_colwidth', 500)
17 pd.set_option('display.max_rows', 100)
18 cf.set_config(cf.file_name='white', dimension=(650,550))
19 cf.set_offline()
20
21 sys.path.insert(1, '../../Datasets')
22
23 String
24 imputilslib.reload(String)
25
26 imputilslib.reload(Supervised)
27 imputilslib.reload(Plotutils)
28
29 imputilslib.reload(Supervisedutils)
30 Supervisedlib.reload(Supervisedutils)
31
32 imputilslib.reload(Plotutils)
33 Supervisedlib.reload(Plotutils)
34
35 def save_object(obj, name):
36     with open(name + '.pkl', 'wb') as file:
37         pickle.dump(obj, file, pickle.HIGHEST_PROTOCOL)
38
39 def load_object(name):
40     with open(name + '.pkl') as file:
41         return(pickle.load(file))
42
43 TAD = pd.read_feather('../ML/0001')
44 TAD.head()
45
46 ratings = pd.read_feather('../..../Datasets/Movielens/rating')
47 ratings.drop(columns=['timestamp'], inplace=True)
48 ratings.drop(columns=['timestamp'], inplace=True)
49 ratings['user_id'] = ratings['user_id'].apply(lambda x: str(x))
50
51 print('Hay un total de {} usuarios'.format(ratings['user_id'].value_counts().sum()))
52 print('Hay un total de {} películas'.format(ratings['movie_id'].value_counts().sum()))
53 print('Distribución de las calificaciones de los usuarios: ')
54
55 movies = TAD.copy()
56
57 movies.dropna(inplace=True, axis=0, how='any')
58 movies.reset_index(inplace=True, level=0)
59 movies['imdb'] = movies['imdb'].apply(lambda x: float(x))
60 movies['imdb'] = movies['imdb'].apply(lambda x: float(x))
61 movies['imdb'] = movies['imdb'].apply(lambda x: float(x))
62 movies['imdb'] = movies['imdb'].apply(lambda x: float(x))
63
64 movies['imdb'].mean()
65 movies['imdb'].median()
66 movies['imdb'].min()
67 movies['imdb'].max()
68
69 count = movies['genres'].str.split('|').explode().value_counts(True)
70
71 genres_mask = count[(count['genres'] > 1) & (count['genres'] < 100)]
72 genres = movies[['user_id', 'movie_id', 'rating', 'genres', 'imdb', 'vote_count', 'keywords', 'poster_path', 'backdrop_path']].loc[genres_mask].reset_index(drop=True)
73
74 movies = movies.loc[movies['imdb'] > 1].loc[(movies['imdb'] > 1).reset_index(drop=True)]
75
76 movies = movies.loc[movies['imdb'] < 1].loc[(movies['imdb'] < 1).reset_index(drop=True)]
77
78 movies = movies.loc[movies['imdb'] < 1].loc[(movies['imdb'] < 1).reset_index(drop=True)]
79
80 movies = movies.loc[movies['imdb'] > 1].loc[(movies['imdb'] > 1).reset_index(drop=True)]
81
82 movies = movies[['user_id', 'movie_id', 'rating', 'genres', 'imdb', 'vote_count', 'keywords', 'poster_path', 'backdrop_path']].reset_index(drop=True)
83
84 movies['imdb'].mean()
85 movies['imdb'].median()
86 movies['imdb'].min()
87 movies['imdb'].max()
88
89 TAD = TAD.merge(movies, on='user_id')
90
91 TAD.shape
92
93 genres_cols = [col for col in TAD.columns if 'genre' in col]
94 language_cols = [col for col in TAD.columns if 'original_language' in col]
95
96 for v in genres_cols:
97     TAD[v] = np.multiply(TAD['rating'],TAD[v])
98
99 for v in language_cols:
100     TAD[v] = np.multiply(TAD['rating'],TAD[v])
101
102 TAD = TAD.groupby('user_id').agg(reest_op)
103
104 TAD.columns = [key + str(i) for key in TAD.op.keys() for i in reest_op.get(key)]
105
106 TAD.columns = 1.0*map(lambda x: x.replace('id', 'n'), TAD.columns)
107
108 TAD.columns = TAD.columns
109
110 TAD.head()
111
112 TAD['rating'] = ratings['rating'].apply(lambda x: 1 if x>0.7 else 0)
113 TAD['rating'] = ratings['rating'].apply(lambda x: 1 if x>0.7 else 0)
114 TAD['rating'] = ratings['rating'].apply(lambda x: 1 if x>0.7 else 0)
115 TAD['rating'] = ratings['rating'].apply(lambda x: 1 if x>0.7 else 0)
116
117 TAD = TAD.groupby('user_id').agg(reest_op)
118
119 X = TAD.copy()
120 X
121
122 from sklearn.preprocessing import MinMaxScaler
123 scaler = MinMaxScaler()
124 scaler.fit(X)
125
126 X = pd.DataFrame(scaler.transform(X))
127
128 X.to_csv('X.csv')
129
130 X
131
132 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
133
134 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
135
136 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
137
138 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
139
140 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
141
142 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
143
144 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
145
146 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
147
148 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
149
150 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
151
152 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
153
154 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
155
156 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
157
158 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
159
160 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
161
162 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
163
164 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
165
166 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
167
168 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
169
170 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
171
172 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
173
174 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
175
176 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
177
178 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
179
180 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
181
182 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
183
184 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
185
186 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
187
188 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
189
190 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
191
192 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
193
194 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
195
196 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
197
198 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
199
200 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
201
202 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
203
204 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
205
206 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
207
208 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
209
210 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
211
212 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
213
214 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
215
216 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
217
218 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
219
220 TAD['user_id'] = np.multiply(TAD['rating'],TAD['user_id'])
221
222 clusters = []
223
224 for i in range(0,clust):
225     clusters.append(TAD.loc[TAD['user_id'] == i])
226
227 test_col = clusters[0].columns[0]
228
229 test_col
230
231 WkrsKruskal = np.loadtxt('WkrsKruskal.dat',delimiter=',')
232 WkrsKruskal.insert(0, 'user_id',test_col)
233 WkrsKruskal
234
235 KMeans = KMeans()
236 KMeans.fit(WkrsKruskal)
237 KMeans.labels_
238 KMeans.labels_
239 KMeans.labels_
240 KMeans.labels_
241 KMeans.labels_
242 KMeans.labels_
243 KMeans.labels_
244 KMeans.labels_
245 KMeans.labels_
246 KMeans.labels_
247 KMeans.labels_
248 KMeans.labels_
249 KMeans.labels_
250 KMeans.labels_
251 KMeans.labels_
252 KMeans.labels_
253 KMeans.labels_
254 KMeans.labels_
255 KMeans.labels_
256 KMeans.labels_
257 KMeans.labels_
258 KMeans.labels_
259 KMeans.labels_
260 KMeans.labels_
261 KMeans.labels_
262 KMeans.labels_
263 KMeans.labels_
264 KMeans.labels_
265 KMeans.labels_
266 KMeans.labels_
267 KMeans.labels_
268 KMeans.labels_
269 KMeans.labels_
270 KMeans.labels_
271 KMeans.labels_
272 KMeans.labels_
273 KMeans.labels_
274 KMeans.labels_
275 KMeans.labels_
276 KMeans.labels_
277 KMeans.labels_
278 KMeans.labels_
279 KMeans.labels_
280 KMeans.labels_
281 KMeans.labels_
282 KMeans.labels_
283 KMeans.labels_
284 KMeans.labels_
285 KMeans.labels_
286 KMeans.labels_
287 KMeans.labels_
288 KMeans.labels_
289 KMeans.labels_
290 KMeans.labels_
291 KMeans.labels_
292 KMeans.labels_
293 KMeans.labels_
294 KMeans.labels_
295 KMeans.labels_
296 KMeans.labels_
297 KMeans.labels_
298 KMeans.labels_
299 KMeans.labels_
300 KMeans.labels_
301 KMeans.labels_
302 KMeans.labels_
303 KMeans.labels_
304 KMeans.labels_
305 KMeans.labels_
306 KMeans.labels_
307 KMeans.labels_
308 KMeans.labels_
309 KMeans.labels_
310 KMeans.labels_
311 KMeans.labels_
312 KMeans.labels_
313 KMeans.labels_
314 KMeans.labels_
315 KMeans.labels_
316 KMeans.labels_
317 KMeans.labels_
318 KMeans.labels_
319 KMeans.labels_
320 KMeans.labels_
321 KMeans.labels_
322 KMeans.labels_
323 KMeans.labels_
324 KMeans.labels_
325 KMeans.labels_
326 KMeans.labels_
327 KMeans.labels_
328 KMeans.labels_
329 KMeans.labels_
330 KMeans.labels_
331 KMeans.labels_
332 KMeans.labels_
333 KMeans.labels_
334 KMeans.labels_
335 KMeans.labels_
336 KMeans.labels_
337 KMeans.labels_
338 KMeans.labels_
339 KMeans.labels_
340 KMeans.labels_
341 KMeans.labels_
342 KMeans.labels_
343 KMeans.labels_
344 KMeans.labels_
345 KMeans.labels_
346 KMeans.labels_
347 KMeans.labels_
348 KMeans.labels_
349 KMeans.labels_
350 KMeans.labels_
351 KMeans.labels_
352 KMeans.labels_
353 KMeans.labels_
354 KMeans.labels_
355 KMeans.labels_
356 KMeans.labels_
357 KMeans.labels_
358 KMeans.labels_
359 KMeans.labels_
360 KMeans.labels_
361 KMeans.labels_
362 KMeans.labels_
363 KMeans.labels_
364 KMeans.labels_
365 KMeans.labels_
366 KMeans.labels_
367 KMeans.labels_
368 KMeans.labels_
369 KMeans.labels_
370 KMeans.labels_
371 KMeans.labels_
372 KMeans.labels_
373 KMeans.labels_
374 KMeans.labels_
375 KMeans.labels_
376 KMeans.labels_
377 KMeans.labels_
378 KMeans.labels_
379 KMeans.labels_
380 KMeans.labels_
381 KMeans.labels_
382 KMeans.labels_
383 KMeans.labels_
384 KMeans.labels_
385 KMeans.labels_
386 KMeans.labels_
387 KMeans.labels_
388 KMeans.labels_
389 KMeans.labels_
390 KMeans.labels_
391 KMeans.labels_
392 KMeans.labels_
393 KMeans.labels_
394 KMeans.labels_
395 KMeans.labels_
396 KMeans.labels_
397 KMeans.labels_
398 KMeans.labels_
399 KMeans.labels_
400 KMeans.labels_
401 KMeans.labels_
402 KMeans.labels_
403 KMeans.labels_
404 KMeans.labels_
405 KMeans.labels_
406 KMeans.labels_
407 KMeans.labels_
408 KMeans.labels_
409 KMeans.labels_
410 KMeans.labels_
411 KMeans.labels_
412 KMeans.labels_
413 KMeans.labels_
414 KMeans.labels_
415 KMeans.labels_
416 KMeans.labels_
417 KMeans.labels_
418 KMeans.labels_
419 KMeans.labels_
420 KMeans.labels_
421 KMeans.labels_
422 KMeans.labels_
423 KMeans.labels_
424 KMeans.labels_
425 KMeans.labels_
426 KMeans.labels_
427 KMeans.labels_
428 KMeans.labels_
429 KMeans.labels_
430 KMeans.labels_
431 KMeans.labels_
432 KMeans.labels_
433 KMeans.labels_
434 KMeans.labels_
435 KMeans.labels_
436 KMeans.labels_
437 KMeans.labels_
438 KMeans.labels_
439 KMeans.labels_
440 KMeans.labels_
441 KMeans.labels_
442 KMeans.labels_
443 KMeans.labels_
444 KMeans.labels_
445 KMeans.labels_
446 KMeans.labels_
447 KMeans.labels_
448 KMeans.labels_
449 KMeans.labels_
450 KMeans.labels_
451 KMeans.labels_
452 KMeans.labels_
453 KMeans.labels_
454 KMeans.labels_
455 KMeans.labels_
456 KMeans.labels_
457 KMeans.labels_
458 KMeans.labels_
459 KMeans.labels_
460 KMeans.labels_
461 KMeans.labels_
462 KMeans.labels_
463 KMeans.labels_
464 KMeans.labels_
465 KMeans.labels_
466 KMeans.labels_
467 KMeans.labels_
468 KMeans.labels_
469 KMeans.labels_
470 KMeans.labels_
471 KMeans.labels_
472 KMeans.labels_
473 KMeans.labels_
474 KMeans.labels_
475 KMeans.labels_
476 KMeans.labels_
477 KMeans.labels_
478 KMeans.labels_
479 KMeans.labels_
480 KMeans.labels_
481 KMeans.labels_
482 KMeans.labels_
483 KMeans.labels_
484 KMeans.labels_
485 KMeans.labels_
486 KMeans.labels_
487 KMeans.labels_
488 KMeans.labels_
489 KMeans.labels_
490 KMeans.labels_
491 KMeans.labels_
492 KMeans.labels_
493 KMeans.labels_
494 KMeans.labels_
495 KMeans.labels_
496 KMeans.labels_
497 KMeans.labels_
498 KMeans.labels_
499 KMeans.labels_
500 KMeans.labels_
501 KMeans.labels_
502 KMeans.labels_
503 KMeans.labels_
504 KMeans.labels_
505 KMeans.labels_
506 KMeans.labels_
507 KMeans.labels_
508 KMeans.labels_
509 KMeans.labels_
510 KMeans.labels_
511 KMeans.labels_
512 KMeans.labels_
513 KMeans.labels_
514 KMeans.labels_
515 KMeans.labels_
516 KMeans.labels_
517 KMeans.labels_
518 KMeans.labels_
519 KMeans.labels_
520 KMeans.labels_
521 KMeans.labels_
522 KMeans.labels_
523 KMeans.labels_
524 KMeans.labels_
525 KMeans.labels_
526 KMeans.labels_
527 KMeans.labels_
528 KMeans.labels_
529 KMeans.labels_
530 KMeans.labels_
531 KMeans.labels_
532 KMeans.labels_
533 KMeans.labels_
534 KMeans.labels_
535 KMeans.labels_
536 KMeans.labels_
537 KMeans.labels_
538 KMeans.labels_
539 KMeans.labels_
540 KMeans.labels_
541 KMeans.labels_
542 KMeans.labels_
543 KMeans.labels_
544 KMeans.labels_
545 KMeans.labels_
546 KMeans.labels_
547 KMeans.labels_
548 KMeans.labels_
549 KMeans.labels_
550 KMeans.labels_
551 KMeans.labels_
552 KMeans.labels_
553 KMeans.labels_
554 KMeans.labels_
555 KMeans.labels_
556 KMeans.labels_
557 KMeans.labels_
558 KMeans.labels_
559 KMeans.labels_
560 KMeans.labels_
561 KMeans.labels_
562 KMeans.labels_
563 KMeans.labels_
564 KMeans.labels_
565 KMeans.labels_
566 KMeans.labels_
567 KMeans.labels_
568 KMeans.labels_
569 KMeans.labels_
570 KMeans.labels_
571 KMeans.labels_
572 KMeans.labels_
573 KMeans.labels_
574 KMeans.labels_
575 KMeans.labels_
576 KMeans.labels_
577 KMeans.labels_
578 KMeans.labels_
579 KMeans.labels_
580 KMeans.labels_
581 KMeans.labels_
582 KMeans.labels_
583 KMeans.labels_
584 KMeans.labels_
585 KMeans.labels_
586 KMeans.labels_
587 KMeans.labels_
588 KMeans.labels_
589 KMeans.labels_
590 KMeans.labels_
591 KMeans.labels_
592 KMeans.labels_
593 KMeans.labels_
594 KMeans.labels_
595 KMeans.labels_
596 KMeans.labels_
597 KMeans.labels_
598 KMeans.labels_
599 KMeans.labels_
600 KMeans.labels_
601 KMeans.labels_
602 KMeans.labels_
603 KMeans.labels_
604 KMeans.labels_
605 KMeans.labels_
606 KMeans.labels_
607 KMeans.labels_
608 KMeans.labels_
609 KMeans.labels_
610 KMeans.labels_
611 KMeans.labels_
612 KMeans.labels_
613 KMeans.labels_
614 KMeans.labels_
615 KMeans.labels_
616 KMeans.labels_
617 KMeans.labels_
618 KMeans.labels_
619 KMeans.labels_
620 KMeans.labels_
621 KMeans.labels_
622 KMeans.labels_
623 KMeans.labels_
624 KMeans.labels_
625 KMeans.labels_
626 KMeans.labels_
627 KMeans.labels_
628 KMeans.labels_
629 KMeans.labels_
630 KMeans.labels_
631 KMeans.labels_
632 KMeans.labels_
633 KMeans.labels_
634 KMeans.labels_
635 KMeans.labels_
636 KMeans.labels_
637 KMeans.labels_
638 KMeans.labels_
639 KMeans.labels_
640 KMeans.labels_
641 KMeans.labels_
642 KMeans.labels_
643 KMeans.labels_
644 KMeans.labels_
645 KMeans.labels_
646 KMeans.labels_
647 KMeans.labels_
648 KMeans.labels_
649 KMeans.labels_
650 KMeans.labels_
651 KMeans.labels_
652 KMeans.labels_
653 KMeans.labels_
654 KMeans.labels_
655 KMeans.labels_
656 KMeans.labels_
657 KMeans.labels_
658 KMeans.labels_
659 KMeans.labels_
660 KMeans.labels_
661 KMeans.labels_
662 KMeans.labels_
663 KMeans.labels_
664 KMeans.labels_
665 KMeans.labels_
666 KMeans.labels_
667 KMeans.labels_
668 KMeans.labels_
669 KMeans.labels_
670 KMeans.labels_
671 KMeans.labels_
672 KMeans.labels_
673 KMeans.labels_
674 KMeans.labels_
675 KMeans.labels_
676 KMeans.labels_
677 KMeans.labels_
678 KMeans.labels_
679 KMeans.labels_
680 KMeans.labels_
681 KMeans.labels_
682 KMeans.labels_
683 KMeans.labels_
684 KMeans.labels_
685 KMeans.labels_
686 KMeans.labels_
687 KMeans.labels_
688 KMeans.labels_
689 KMeans.labels_
690 KMeans.labels_
691 KMeans.labels_
692 KMeans.labels_
693 KMeans.labels_
694 KMeans.labels_
695 KMeans.labels_
696 KMeans.labels_
697 KMeans.labels_
698 KMeans.labels_
699 KMeans.labels_
700 KMeans.labels_
701 KMeans.labels_
702 KMeans.labels_
703 KMeans.labels_
704 KMeans.labels_
705 KMeans.labels_
706 KMeans.labels_
707 KMeans.labels_
708 KMeans.labels_
709 KMeans.labels_
710 KMeans.labels_
711 KMeans.labels_
712 KMeans.labels_
713 KMeans.labels_
714 KMeans.labels_
715 KMeans.labels_
716 KMeans.labels_
717 KMeans.labels_
718 KMeans.labels_
719 KMeans.labels_
720 KMeans.labels_
721 KMeans.labels_
722 KMeans.labels_
723 KMeans.labels_
724 KMeans.labels_
725 KMeans.labels_
726 KMeans.labels_
727 KMeans.labels_
728 KMeans.labels_
729 KMeans.labels_
730 KMeans.labels_
731 KMeans.labels_
732 KMeans.labels_
733 KMeans.labels_
734 KMeans.labels_
735 KMeans.labels_
736 KMeans.labels_
737 KMeans.labels_
738 KMeans.labels_
739 KMeans.labels_
740 KMeans.labels_
741 KMeans.labels_
742 KMeans.labels_
743 KMeans.labels_
744 KMeans.labels_
745 KMeans.labels_
746 KMeans.labels_
747 KMeans.labels_
748 KMeans.labels_
749 KMeans.labels_
750 KMeans.labels_
751 KMeans.labels_
752 KMeans.labels_
753 KMeans.labels_
754 KMeans.labels_
755 KMeans.labels_
756 KMeans.labels_
757 KMeans.labels_
758 KMeans.labels_
759 KMeans.labels_
760 KMeans.labels_
761 KMeans.labels_
762 KMeans.labels_
763 KMeans.labels_
764 KMeans.labels_
765 KMeans.labels_
766 KMeans.labels_
767 KMeans.labels_
768 KMeans.labels_
769 KMeans.labels_
770 KMeans.labels_
771 KMeans.labels_
772 KMeans.labels_
773 KMeans.labels_
774 KMeans.labels_
775 KMeans.labels_
776 KMeans.labels_
777 KMeans.labels_
778 KMeans.labels_
779 KMeans.labels_
780 KMeans.labels_
781 KMeans.labels_
782 KMeans.labels_
783 KMeans.labels_
784 KMeans.labels_
785 KMeans.labels_
786 KMeans.labels_
787 KMeans.labels_
788 KMeans.labels_
789 KMeans.labels_
790 KMeans.labels_
791 KMeans.labels_
792 KMeans.labels_
793 KMeans.labels_
794 KMeans.labels_
795 KMeans.labels_
796 KMeans.labels_
797 KMeans.labels_
798 KMeans.labels_
799 KMeans.labels_
800 KMeans.labels_
801 KMeans.labels_
802 KMeans.labels_
803 KMeans.labels_
804 KMeans.labels_
805 KMeans.labels_
806 KMeans.labels_
807 KMeans.labels_
808 KMeans.labels_
809 KMeans.labels_
810 KMeans.labels_
811 KMeans.labels_
812 KMeans.labels_
813 KMeans.labels_
814 KMeans.labels_
815 KMeans.labels_
816 KMeans.labels_
817 KMeans.labels_
818 KMeans.labels_
819 KMeans.labels_
820 KMeans.labels_
821 KMeans.labels_
822 KMeans.labels_
823 KMeans.labels_
824 KMeans.labels_
825 KMeans.labels_
826 KMeans.labels_
827 KMeans.labels_
828 KMeans.labels_
829 KMeans.labels_
830 KMeans.labels_
831 KMeans.labels_
832 KMeans.labels_
833 KMeans.labels_
834 KMeans.labels_
835 KMeans.labels_
836 KMeans.labels_
837 KMeans.labels_
838 KMeans.labels_
839 KMeans.labels_
840 KMeans.labels_
841 KMeans.labels_
842 KMeans.labels_
843 KMeans.labels_
844 KMeans.labels_
845 KMeans.labels_
846 KMeans.labels_
847 KMeans.labels_
848 KMeans.labels_
849 KMeans.labels_
850 KMeans.labels_
851 KMeans.labels_
852 KMeans.labels_
853 KMeans.labels_
854 KMeans.labels_
855 KMeans.labels_
856 KMeans.labels_
857 KMeans.labels_
858 KMeans.labels_
859 KMeans.labels_
860 KMeans.labels_
861 KMeans.labels_
862 KMeans.labels_
863 KMeans.labels_
864 KMeans.labels_
865 KMeans.labels_
866 KMeans.labels_
867 KMeans.labels_
868 KMeans.labels_
869 KMeans.labels_
870 KMeans.labels_
871 KMeans.labels_
872 KMeans.labels_
873 KMeans.labels_
874 KMeans.labels_
875 KMeans.labels_
876 KMeans.labels_
877 KMeans.labels_
878 KMeans.labels_
879 KMeans.labels_
880 KMeans.labels_
881 KMeans.labels_
882 KMeans.labels_
883 KMeans.labels_
884 KMeans.labels_
885 KMeans.labels_
886 KMeans.labels_
887 KMeans.labels_
888 KMeans.labels_
889 KMeans.labels_
890 KMeans.labels_
891 KMeans.labels_
892 KMeans.labels_
893 KMeans.labels_
894 KMeans.labels_
895 KMeans.labels_
896 KMeans.labels_
897 KMeans.labels_
898 KMeans.labels_
899 KMeans.labels_
900 KMeans.labels_
901 KMeans.labels_
902 KMeans.labels_
903 KMeans.labels_
904 KMeans.labels_
905 KMeans.labels_
906 KMeans.labels_
907 KMeans.labels_
908 KMeans.labels_
909 KMeans.labels_
910 KMeans.labels_
911 KMeans.labels_
912 KMeans.labels_
913 KMeans.labels_
914 KMeans.labels_
915 KMeans.labels_
916 KMeans.labels_
917 KMeans.labels_
918 KMeans.labels_
919 KMeans.labels_
920 KMeans.labels_
921 KMeans.labels_
922 KMeans.labels_
923 KMeans.labels_
924 KMeans.labels_
925 KMeans.labels_
926 KMeans.labels_
927 KMeans.labels_
928 KMeans.labels_
929 KMeans.labels_
930 KMeans.labels_
931 KMeans.labels_
932 KMeans.labels_
933
```