

Universidad de Granada

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIONES

## P3 : COMPETICIÓN EN KAGGLE



















*Inteligencia de Negocio Viernes 12:30*

Autor:

Ignacio Garach Vélez [igarachv@correo.ugr.es](mailto:igarachv@correo.ugr.es)

Enero 2022

This competition has completed. This leaderboard reflects the final standings.

#	Team	Members	Score	Entries	Last	Solution
1	PabloNieto_UGR_IN		0.58821	27	5d	
2	DanielBarbero_UGR_IN		0.57015	9	4d	
3	CarlosRomero_UGR_IN		0.56960	11	5d	
4	IsabelJaldo_UGR_IN		0.56737	8	4d	
5	<b>IgnacioGarach_UGR_IN</b>		0.56237	38	4d	
6	PaulaHernandez_UGR_IN		0.56126	22	5d	
7	EnriqueSanguesa_UGR_CEUTA_IN		0.56126	26	4d	
8	CarlosLopez_UGR_IN		0.56071	18	5d	
9	Rodrigolbañez_UGR_IN		0.56043	22	4d	
10	PatriciaVillalba_UGR_IN		0.56015	12	4d	
11	LauraTorres_UGR_IN		0.55987	15	4d	
12	JoseAbela_UGR_IN		0.55959	29	5d	
13	PabloGarcia_UGR_IN		0.55932	10	5d	
14	JesusLeyva_UGR_IN		0.55709	11	5d	
15	SergioRamos_UGR_IN		0.55543	14	5d	
16	JoseAntonioLopez_UGR_IN		0.55432	7	5d	
17	JuanJoseMartinez_UGR_IN		0.55432	20	4d	
18	AlexLedesma_UGR_IN		0.55376	22	5d	

# COMPETICIÓN EN KAGGLE

9 de enero de 2023

## Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Tabla resumen de las subidas a Kaggle</b>	<b>3</b>
<b>3. Exploración de datos</b>	<b>6</b>
<b>4. Técnicas de preprocesado utilizadas</b>	<b>7</b>
4.1. Imputación de valores nulos . . . . .	7
4.1.1. Simple Imputer . . . . .	8
4.1.2. KNN Imputer . . . . .	8
4.2. Escalado de variables . . . . .	8
4.2.1. Escalado MinMax . . . . .	8
4.2.2. Escalado a la Normal Estándar . . . . .	8
4.2.3. Escalado Robusto . . . . .	8
4.2.4. PCA . . . . .	8
4.3. Eliminación de Outliers . . . . .	8
4.4. Selección de variables . . . . .	9
4.5. Creación y selección de instancias . . . . .	9
4.6. Detección de instancias duplicadas y etiquetas ruidosas . . . . .	9
<b>5. Estrategias y modelos utilizados</b>	<b>10</b>
5.1. Gradient Boosting . . . . .	10
5.2. Multilayer Perceptron . . . . .	10
5.3. Histogram Gradient Boosting . . . . .	10
5.4. Soluciones de AutoML . . . . .	11
5.4.1. AutoSKLearn . . . . .	11
5.4.2. AutoKeras . . . . .	11
5.4.3. Google AutoML VertexAI . . . . .	11
5.5. Postprocesado: Calibrado de Probabilidades . . . . .	11
5.6. Stacking . . . . .	12
5.7. Subidas erróneas . . . . .	12
<b>6. Modelos finales</b>	<b>13</b>
<b>7. Bibliografía</b>	<b>13</b>

## 1. Introducción

En esta práctica vamos a tratar de conseguir los mejores resultados de tasa de acierto en clasificación para un conjunto de datos de canciones, en el que las mayores dificultades son que la variable a predecir es el género con 11 clases posibles y el desbalanceo de las clases, además encontraremos datos faltantes. La competición se realiza en la popular plataforma Kaggle. Se llevaran a cabo todas las fases del aprendizaje, con especial atención al preprocesado de los datos, además intentaremos poner a prueba un importante abanico de modelos, centrandonos en principio en los ensembles basados en árboles que tienen la propiedad de funcionar de forma robusta pese a encontrarnos antes un dataset desbalanceado.

En primer lugar llevaremos a cabo un pequeño análisis exploratorio de los datos para ver si podemos extraer alguna conclusión previa a comenzar el entrenamiento. En estas primeras páginas además se incluye una tabla con las subidas realizadas a Kaggle, las horas de subida y posición del momento en la clasificación los he estimado durante la redacción de la memoria pues creía que Kaggle los guardaba, creo que son bastante aproximados a la realidad, aunque la estimación de la posición la he realizado de forma algo pesimista en torno a las subidas 15 a 30, donde hubo más movimientos en la clasificación.

## 2. Tabla resumen de las subidas a Kaggle

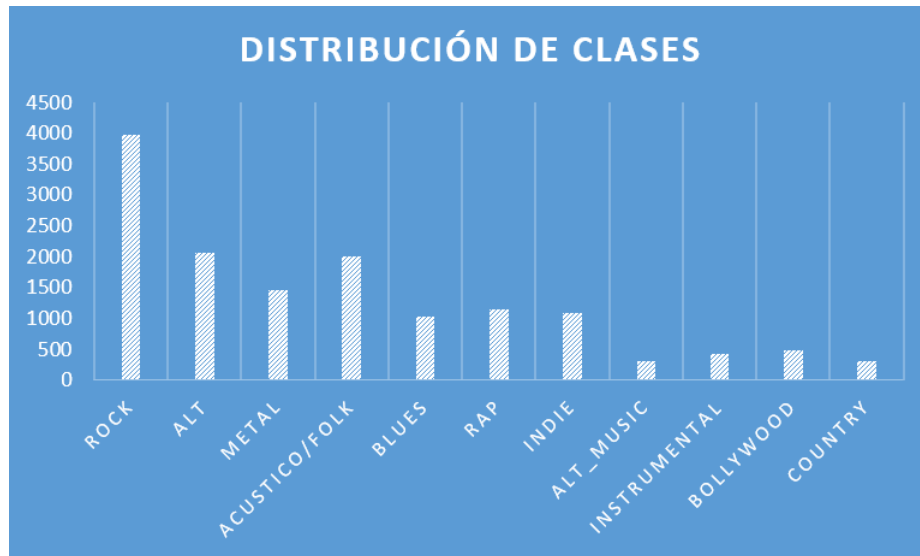
A continuación se incluye la tabla solicitada que incluye fecha y hora de la subida, posición del momento, scores en entrenamiento, validación y test, así como una breve descripción del preprocesado y modelo utilizado.

	Subida	Pos	Training	Validation	Test	Preprocesado
1	25/12 17:05	2	0.544	0.522	0.52236	Preprocesado 1 + Simple Imputer
2	25/12 17:45	1	0.617	0.531	0.54347	Preprocesado 1 + KNN Imputer + MinMaxScaler
3	25/12 18:27	1	0.573	0.524	0.52875	Preprocesado 1 + Simple Imputer + StandardScaler
4	26/12 13:30	1	0.721	0.533	0.54737	Preprocesado 1 Sin Extra
5	26/12 23:45	1	0.916	0.866	0.35982	Preprocesado 2 KNN Imputer + Feature Selection + SMOTE + ENN
6	27/12 23:55	1	0.558	0.526	0.25812	Preprocesado 1 + KNN Imputer + StandardScaler
7	28/12 00:10	1	0.632	0.529	0.54098	Preprocesado 1
8	30/12 13:37	1	0.627	0.539	0.55404	Preprocesado 1
9	30/12 14:20	2	0.619	0.536	0.55043	Preprocesado 3 Isolation Forest + Simple Imputer
10	30/12 15:00	2	0.637	0.529	0.54125	Preprocesado 1 + KNN Imputer + StandardScaler
11	30/12 17:33	2	0.591	0.521	0.50986	Preprocesado 1 + SimpleImputer + RobustScaler
12	30/12 19:38	2	0.573	0.524	0.26812	Preprocesado 1 + SimpleImputer + Standard Scaler
13	31/12 17:20	3	0.754	0.544	0.24312	Preprocesado 2 + Robust Scaler + SimpleImputer + SMOTETomek
14	31/12 17:45	3	0.592	0.534	0.21255	Preprocesado 1 + SimpleImputer + Robust Scaler
15	31/12 18:30	3	0.592	0.534	0.54569	Preprocesado 1 + SimpleImputer + Robust Scaler
16	31/12 20:12	3	0.592	0.534	0.54848	Preprocesado 1 + SimpleImputer + Robust Scaler
17	31/12 20:42	2	0.621	0.542	0.55237	Preprocesado 1 + SimpleImputer + Robust Scaler
18	01/01 20:00	4	0.751	0.714	0.50764	Preprocesado 2 + Robust Scaler + SimpleImputer + SMOTETomek
19	01/01 21:20	2	0.631	0.546	0.55598	Preprocesado 1
20	01/01 23:23	2	0.627	0.547	0.55626	Preprocesado 1
21	02/01 00:00	3	0.626	0.538	0.54626	Preprocesado 1
22	02/01 00:45	3	0.626	0.538	0.54626	Preprocesado 1
23	02/01 13:00	3	0.627	0.552	0.55237	Preprocesado 1 + Simple imputer
24	02/01 17:23	2	0.633	0.551	0.55682	Preprocesado 1 + Simple imputer
25	02/01 19:26	3	0.611	0.534	0.47818	Preprocesado 1 + Robust Scaler + PCA + Simple imputer
26	02/01 23:47	2	0.634	0.552	0.55987	Preprocesado 1 + Simple imputer
27	03/01 00:58	2	0.624	0.552	0.55487	Preprocesado 1 + Simple imputer
28	03/01 17:24	3	0.627	0.551	0.56015	Preprocesado 1 + Simple imputer
29	04/01 00:35	3	0.633	0.553	0.55904	Preprocesado 1 + Simple imputer
30	04/01 02:08	2	0.622	0.552	0.56071	Preprocesado 1 + Simple imputer
31	04/01 17:08	2	0.585	0.558	0.56237	Preprocesado 1 + Simple imputer
32	04/01 19:56	4	0.701	0.594	0.55765	Preprocesado 4 + Eliminación duplicados Ruidosos + Simple Imputer
33	04/01 20:26	4	0.673	0.573	0.55793	Preprocesado 5 + Eliminación duplicados Ruidosos manteniendo algunos + Simple Imputer
34	05/01 00:06	4	0.659	0.623	0.53152	Preprocesado 4 + Robust Scaler + PCA + Simple imputer
35	05/01 00:16	4	0.571	0.562	0.51708	Preprocesado 4 + Eliminación duplicados Ruidosos + Simple Imputer
36	05/01 21:08	4	0.727	0.600	0.56237	Preprocesado 4 + Simple Imputer en los que lo requieren
37	05/01 22:38	5	0.689	0.577	0.54459	Preprocesado 4 + Simple Imputer
38	05/01 23:55	5	—	0.588	0.22478	Preprocesado 4 + Simple Imputer en los que lo requieren

Subida	Test	Descripción
1	0.52236	GradientBoosting(n_estimators=100, learning_rate=0.5, max_depth=1)
2	0.54347	GradientBoosting(n_estimators=100, learning_rate=0.1, max_depth=3)
3	0.52875	MLPClassifier(iter = 1000, layer_sizes=(30, 30, 30, 11), alpha=0.0001, lr=0.0005)
4	0.54737	HistGradientBoostingClassifier(learning_rate=0.07571428571428572)
5	0.35982	GradientBoostingClassifier(n_estimators=100, learning_rate=0.325, max_depth=3)
6	0.25812	MLPClassifier(alpha=3.198481e-06, beta_1=0.999, beta_2=0.9, layer_sizes=(24, 24))
7	0.54098	AutoSklearnClassifier(time_left_for_this_task=1200)
8	0.55404	HistGradientBoostingClassifier(learning_rate=0.02424137, depth=30)
9	0.55043	HistGradientBoostingClassifier(learning_rate=0.023620689655172413, depth=20)
10	0.54125	MLPClassifier(max_iter = 1000, random_state=1, hidden_layer_sizes=(24, 24, 24, 11))
11	0.50986	autokeras.StructuredDataClassifier(overwrite=True, max_trials=20, seed=1)
12	0.26812	SVC(kernel='rbf', gamma=0.025, C=13.48)
13	0.24312	HistGradientBoostingClassifier(random_state=0, learning_rate=0.0242413793103, max_depth=30)
14	0.21255	MLPClassifier(alpha=0.001, hidden_layer_sizes=(50, 50, 11), max_iter=1024)
15	0.54569	MLPClassifier(alpha=0.001, hidden_layer_sizes=(50, 50, 11), max_iter=1024)
16	0.54848	Calibrated + MLPClassifier(alpha=0.001, layer_sizes=(50, 50, 11), max_iter=1024)
17	0.55237	Calibrated + HistGradientBoostingClassifier(12_regularization=0.10736842, learning_rate=0.02424137931, max_depth=30)
18	0.50764	HistGradientBoostingClassifier(random_state=0, learning_rate=0.02424137931, max_depth=30)
19	0.55598	HistGradientBoostingClassifier(12_regularization=0.0020612244897959186, learning_rate=0.026000000000000002, max_depth=22)
20	0.55626	Calibrated + HistGradientBoostingClassifier(learning_rate=0.024241379310344827, max_depth=30)
21	0.54626	Calibrated + HistGradientBoostingClassifier(12_regularization=0.00206122448, learning_rate=0.0260, max_depth=22) + Balanceo
22	0.54626	Calibrated + HistGradientBoostingClassifier(12_regularization=0.00206, learning_rate=0.026000) + Balanceo
23	0.55237	XGBClassifier(learning_rate=0.083, max_depth=5)
24	0.55682	Stacking( HistGradBoost + XGBBoost)
25	0.47818	autokeras.StructuredDataClassifier(overwrite=True, max_trials=20, seed=1)
26	0.55987	Stacking( HistGradBoost + XGBBoost + RandomForest)
27	0.55487	Calibrated + Stacking( HistGradBoost + XGBBoost + RandomForest)
28	0.56015	Stacking( HistGradBoost + XGBBoost + RandomForest + GradientBoosting)
29	0.55904	Stacking( HistGradBoost + XGBBoost + RandomForest + GradientBoosting) Optimized LR
30	0.56071	Stacking( HistGradBoost + XGBBoost + RandomForest + GradientBoosting + LightGBM)
31	0.56237	Google AutoML 1 hora de cómputo gratuita
32	0.55765	Stacking( HistGradBoost + XGBBoost + RandomForest + GradientBoosting + LightGBM)
33	0.55793	Stacking( HistGradBoost + XGBBoost + RandomForest + GradientBoosting + LightGBM)
34	0.53152	autokeras.StructuredDataClassifier(overwrite=True, max_trials=20, seed=1)
35	0.51708	AutoSklearnClassifier(time_left_for_this_task=1800, resampling_strategy=cv))"
36	0.56237	Stacking( HistGradBoost + XGBBoost + RandomForest + GradientBoosting + LightGBM)
37	0.54459	CatBoostClassifier(iterations=1000, learning_rate=0.005455775365712, depth=3)
38	0.22478	Stacking 2 mejores(Stacking 5 + Google AutoML) Compensación del desbalanceo

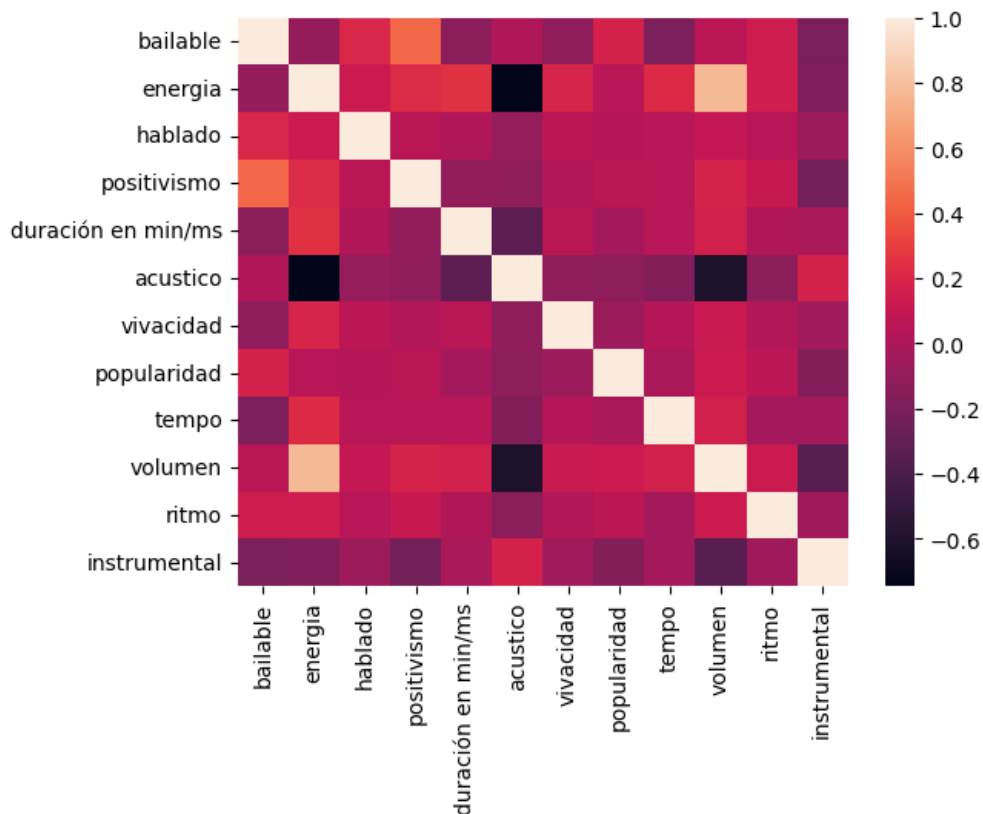
### 3. Exploración de datos

Comenzaremos observando la distribución de clases, para ello se ha construido un diagrama de barras:

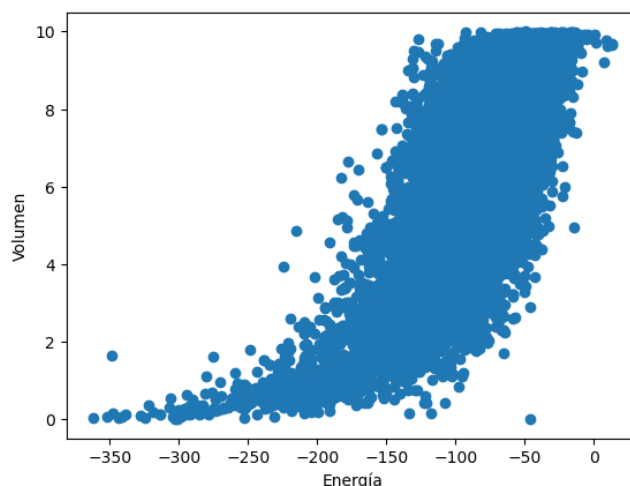


Se hace evidente que hay un desbalanceo alto entre las clases, abundando la música Rock y Alternativa, luego el resto de grupos comienzan a ser cada vez más minoritarios.

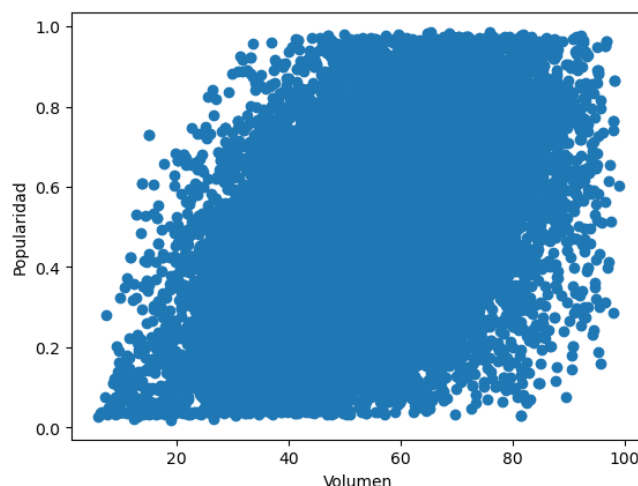
Por otro lado respecto a las variables podemos comprobar que tenemos algunas correlaciones altas, por lo que podríamos plantearnos realizar una selección de las mismas:



Por ejemplo, entre volumen y energía hay una correlación lineal bastante alta, si graficamos los valores con un scatter plot reafirmamos el comportamiento.



Por otro lado, entre volumen y popularidad, la correlación también es positiva pero más tenue, lo cual en la siguiente gráfica también se aprecia:



## 4. Técnicas de preprocesado utilizadas

Para poder comenzar a explorar distintos tipos de modelos de aprendizaje, debemos adecuar los datos a los mismos, dependiendo del modelo tendremos unos requerimientos u otros, en esta sección vamos a estudiar los distintos tipos de preprocesado que se efectúan.

En primer lugar, para todos los preprocesados, se eliminan las columnas de título o artista según se indica en las instrucciones de la práctica, aunque seguramente se podría utilizar algún enfoque para aprovecharlas, es razonable pensar que un artista está restringido a determinados géneros, igual que los títulos podrían contener palabras clave de algún género. Por otro lado, se separa la columna target, y se codifica con un LabelEncoder para poder utilizarla comodamente con los modelos de scikit-learn.

Además se van a transformar las variables categóricas a numéricas, en el caso de la binaria mayor/menor, simplemente se cambia a 0 y 1, además hay una columna con strings P1 A P11 que siguen una escala numérica, por lo cual se ha considerado razonable transformarla a ella.

Estos primeros pasos se han reflejado en la Tabla Resumen como preprocesado 1.

El paso siguiente pese a poder considerarse base, no lo incluimos, porque hay algunos modelos que lo incluyen durante el entrenamiento, se trata del procesamiento de datos nulos.

### 4.1. Imputación de valores nulos

Como hemos visto en el análisis exploratorio, existen 3 columnas en el dataset que tienen bastantes valores nulos; clave, popularidad e instrumental. Tenemos básicamente 4 opciones para tratarlos, la primera es eliminar



toda instancia que los tenga, algo que se presume inviable en este dataset, se pierde mucha información, por otro lado se pueden dejar tal cual porque muchos algoritmos como XGBoost o HistGradBoosting tratan estos valores faltantes de forma automática, normalmente basados en técnicas de separación con árboles. Finalmente, lo más habitual es la imputación mediante la media o la moda, o mediante técnicas basadas en los vecinos más cercanos. Es crucial que utilicemos solo los datos de entrenamiento para esta imputación, y una vez ajustados los objetos utilizarlos también en los datos de test para evitar filtrado de información que haga que sobreestimemos los resultados obtenidos en test.

#### 4.1.1. Simple Imputer

Se han testeado las opciones que ofrece el objeto de sklearn, donde las que mejores resultados ofrecen son mediante la media y la mediana, principalmente la primera, por lo que es la que se utilizará en un alto porcentaje de los modelos desarrollados.

#### 4.1.2. KNN Imputer

Esta técnica de imputación más compleja, en principio debería ser más correcta, tomando en cuenta los grupos de instancias más semejantes al dato con valor nulo, sin embargo en nuestro caso con clases tan altamente desbalanceadas, es complicado escoger el número de vecinos a considerar, y en general la abundancia de ejemplos de la clase mayoritaria hace que se suelen imputar valores semejantes a las mismas. Probablemente por ello los resultados han resultado peores en las pruebas que se han efectuado.

### 4.2. Escalado de variables

Puesto que los valores de las variables están en distintas escalas, puede que algunos algoritmos den mayor peso a aquellas con escala mayor (esto no suele ocurrir por ejemplo con los algoritmos que trabajan con árboles, que suelen hacer separaciones independientes a las escalas). Para solventarlo en el caso de otros modelos como redes neuronales, modelos lineales, máquinas de soporte vectorial, etc... vamos a utilizar distintos métodos para escalar: escalado uniforme, estandarización y escalado robusto. Aunque como los datos están desbalanceados y se utilizan mayoritariamente algoritmos ensemble de árboles, no es necesario utilizarlos en estos modelos, aunque teóricamente por ejemplo el escalado uniforme no debería afectarlos.

#### 4.2.1. Escalado MinMax

Se trata de un escalado que conserva distancias dentro de cada variable, aunque las transforma todas al rango  $[0, 1]$ . Cumple su objetivo mediante una transformación afín que no distorsiona apenas los datos.

#### 4.2.2. Escalado a la Normal Estándar

En este caso, adecuamos los datos para que sigan una distribución normal con media 0 y varianza 1. Esto puede ser útil por ejemplo para el entrenamiento de redes neuronales, pues además de no otorgar mayor importancia a algunas variables, acelera el entrenamiento.

#### 4.2.3. Escalado Robusto

Finalmente este método se utiliza como técnica que además de escalar y centrar los datos, es robusta a los outliers presentes en los mismos, utilizando el rango intercuartílico entre el primer y el tercer cuartil. Este método ha resultado ser el mejor cuando ha sido necesario al no utilizarse técnicas basadas en árboles.

#### 4.2.4. PCA

Aunque no se trata de un escalado como tal, nosotros además de para visualización en 2D lo vamos a considerar como método de eliminación de correlación entre las variables, algo que también puede ser beneficioso para el entrenamiento de las redes neuronales.

### 4.3. Eliminación de Outliers

Además del escalado robusto, se ha probado la técnica de Isolation Forest. Este método está inspirado en el algoritmo de RandomForest visto en teoría, se genera un nodo raíz con los datos de entrenamiento, se selecciona un atributo aleatorio y un valor dentro de su rango de valores, se separan 2 nodos separando según dicho valor y se repiten los pasos anteriores hasta llegar a los nodos hoja. Esto se repite varias veces para tener un bosque en lugar de un solo árbol. El valor predicho para cada observación es el número de separaciones en promedio que

se han necesitado para aislar dicha observación en el conjunto de árboles. Cuanto menor es el valor, mayor es la probabilidad de que se trate de una anomalía o outlier.

Hemos usado la función asociada de la librería con la recomendación de tamaño del bosque (200) del autor para nuestro tamaño de datos, igualmente se probó con distintos tamaños y resultado similar a partir del umbral de 100. Se añadió una probabilidad de existencia de outliers de un 5 por ciento. Trás esto, eliminamos los datos que el algoritmo ha considerado como outliers.

```
from sklearn.ensemble import IsolationForest
out = IsolationForest(n_estimators = 200, contamination=0.05, random_state=1)
out.fit(train_df)
liers = out.predict(train_df)
(liers==1).sum()

index=[]
for i in range(len(liers)):
    if liers[i]==-1:
        index.append(i)
train_df = np.delete(train_df, index, axis=0)
train_target = np.delete(train_target, index, axis=0)
```

#### 4.4. Selección de variables

Se ha tratado de utilizar una búsqueda de las variables que mejor resultado obtienen, mediante una técnica de wrapping como la del notebook del ejemplo, aunque los resultados no han sido satisfactorios, puede ser razonable porque no tenemos una dimensionalidad muy alta y todas las variables parecen aportar algo. Se ha utilizado la técnica de búsqueda hacia delante y hacia atrás.

#### 4.5. Creación y selección de instancias

Se ha tratado de utilizar la técnica SMOTE para crear instancias sintéticas que lidien con el alto desbalanceo de las clases, aunque por separado los resultados son muy malos, es probable que se produjera el fenómeno de sobregeneralización visto en teoría. Por otro lado al utilizar validación cruzada, se sobreestima el desempeño del modelo, porque se esta mezclando la información en las particiones, obteniendo tasas de acierto en validación mucho mas altas de los posteriormente obtenido en test.

Finalmente, se ha utilizado la hibridación de SMOTE con métodos de selección de instancias como ENN y Tomek Links, que en general no han sido demasiado útiles, pese a combatir el desbalanceo, los resultados en el accuracy que se pretende optimizar no son muy buenos, el mejor de ellos con SMOTE+Tomek, obtiene un 0.507 en test.

#### 4.6. Detección de instancias duplicadas y etiquetas ruidosas

En los últimos días de realización de la práctica, concretamente trás la subida en la que se utilizó la solución de AutoML de Google, se detectó que existían instancias duplicadas tanto en test como en entrenamiento. Siendo además muchas de las de entrenamiento probablemente ruidosas en el sentido de que pese a tener los mismos valores de variable, tenían una clase distinta como etiqueta, esto salvo que sea intencional en los datos por algún motivo de diseño ya que sean ambos géneros correctos, es un error en la toma de la muestra que ya impide que se lleve a cabo un entrenamiento correcto (ni siquiera estamos seguro que el conjunto de test sea correcto, que impide que las instancias duplicadas de test en Kaggle tengan tambien clases distintas...). A partir de ahí y con el poco tiempo disponible, se utilizaron los preprocesados 4 y 5, en el el primero se eliminan todas las instancias duplicadas con clase errónea y en el segundo se conserva sólo la primera aparición de la instancia. Con el 4 se han obtenido mejores resultados aunque no se ha tenido mucho tiempo para trabajar con estos nuevos datos, es probable que se hubiera conseguido un mejor accuracy. Finalmente el mejor modelo ha resultado ser uno de los que usaba estos nuevos datos, igualado con otro que usaba todos, por lo que consideramos que el que no contenía duplicados es bastante más fiable para un uso futuro.

Como propuesta final no realizada por falta de tiempo, podría utilizarse algún método basado en los vecinos más cercanos o en árboles para poder elegir con que clase de los duplicados quedarnos, evitando así perder datos de entrenamiento, más de un 10 por ciento de las mismas en este caso.

## 5. Estrategias y modelos utilizados

La exploración de datos ha dejado como principal conclusión que una de las mayores dificultades de este dataset es su desbalanceo, la clase Rock por ejemplo es muy mayoritaria. Por ello en el preprocesado ya se han documentado los intentos que se han hecho para tratar este desbalanceo, sin gran éxito. La conclusión ha sido sencilla, trabajar mayoritariamente con algoritmos ensembles basados en árboles, que hemos estudiado en teoría, que tienen como ventaja un manejo bastante bueno para estos conjuntos de datos. Por otro lado, recientemente había leído acerca del paradigma del AutoML y me ha parecido una buena ocasión para ponerlo a prueba, los resultados en general son bastante buenos, sobre todo teniendo en cuenta que no se cuenta con gran capacidad de cómputo y que el esfuerzo para ponerlos en marcha es casi nulo. En el caso de la versión de prueba de la solución de Google, el tema de ingestión de datos y predicción por batches hace algo complicado el pre y postprocesamiento, pero obtiene uno de los 2 mejores modelos que se han conseguido en esta práctica para nuestros datos, además de ayudarnos a detectar la duplicidad, probablemente errónea de algunas instancias.

En esta sección se relatan los intentos llevados a cabo con estas técnicas, como algunas de ellas ya se describieron en la práctica 1, simplemente se comenta su optimización de hiperparámetros(normalmente mediante el objeto GridSearchCV) o su postprocesado.

### 5.1. Gradient Boosting

Para comenzar la práctica, los primeros 2 intentos se utilizó un modelo de Boosting de Scikit-Learn, para el cuál ya se apreciaba, sólo con 100 árboles y con la profundidad mínima, que la capacidad predictiva ya es mucho mayor a la de un sólo árbol de decisión, además en este caso no estamos sobreajustando, ya que los errores en entrenamiento, validación y test son muy similares, por otro lado, la capacidad predictiva y de ajuste es mejorable.

El segundo intento, aumenta la profundidad a 3, algo que sí será habitual en los modelos de más adelante, además se optimiza el learning rate, obteniendo un resultado bastante mejor en validación y test, aunque comienza a apreciarse que estamos sobreajustando los datos, porque el error en entrenamiento es bastantes más pequeño que en validación y test. A pesar de ello dada la naturaleza del dataset parece que será algo a lo que tengamos que acostumbrarnos, el desbalanceo de clases, hace que el ajuste de los datos de entrenamiento en detalle sea muy importante para poder detectar los nichos del espacio en los que se encuentren las instancias de las clases minoritarias.

Lo negativo de este modelo es que es costoso de entrenar, escoger hiperparámetros y validar con todos los datos que tenemos y lo será aun más cuando aumentemos el número de estimadores, esto lo solucionaremos utilizando una "discretización" del mismo que además trata los valores nulos de forma automática.

### 5.2. Multilayer Perceptron

Por otro lado, a lo largo de la práctica, se ha tratado sin mucho éxito de utilizar redes neuronales para la predicción. Se han probado todos los métodos de preprocesamiento comentados, tanto estandarización como eliminación de correlaciones con el método de componentes principales. La elección de la arquitectura de la red fue complicada, porque los tiempos de entrenamiento son también costosos, en muchas ocasiones se hacen pruebas con grandes diferencias entre sí, dejando sin explorar grandes zonas del espacio paramétrico del modelo. Se ha utilizado el perceptrón multicapa, normalmente con 2 o 3 capas ocultas de entre 20 y 50 neuronas, logrando como mejor resultado un 0.545 en test, pese a que estas redes normalmente al sobreajuste, en este caso no es tan exagerado como por ejemplo ha ocurrido con los ensembles de árboles, quizá no hemos conseguido encontrar una arquitectura que sea capaz de ajustar bien los datos.

### 5.3. Histogram Gradient Boosting

Este modelo implementa el algoritmo Gradient Boosting de una forma curiosa y bastante eficiente y útil, discretiza en 255 grupos(por defecto, puede disminuirse) los valores de cada variable de entrada y utiliza histogramas para almacenar los datos durante la construcción de cada árbol a la hora de generar las divisiones. Esto ayuda mucho en la elección de hiperparámetros y en el entrenamiento, que es una de las mayores taras del método, a diferencia de técnicas como Random Forest que pueden aprovechar en alto grado el paralelismo, los árboles de boosting se construyen secuencialmente. Evidentemente, estamos haciendo una aproximación bastante grande, pero en general se ha comprobado que los resultados no empeoran, en nuestro caso incluso mejoran, de hecho es el algoritmo que más hemos utilizados en la práctica, tratando de llevar a cabo optimizaciones de los distintos parámetros que ofrece en distinto orden como regularización L2, tasa de aprendizaje, profundidad máxima, mínimo número de instancias por nodo hoja, etc...

La mejor solución obtenida con este modelo, que también se incluyó en los algoritmos de stacking fue con regularización l2 0.00206122448, tasa de aprendizaje 0.0260, máxima profundidad 22 y mínimo de instancias en

hoja 18, obteniendo 0.556 en test, de nuevo el sobreajuste ocurre, obteniéndose en torno a un 0.65 en training para todos los modelos probados a lo largo de la práctica. Además de este modelo, se han probado otras implementaciones que usan el mismo principio además de otras mejoras como XGBoost o LightGBM.

## 5.4. Soluciones de AutoML

En general, se entiende por *aprendizaje automático* aquel software que es capaz de llevar a cabo la labor de aprendizaje de principio a fin a partir de los datos, en nuestro caso tabulares simples. Es decir, lleva a cabo todo el preprocesado, imputación, balanceo, escalado, eliminación de outliers, ajuste de modelos, elección de hiperparámetros óptimos y validación. Construye un modelo desde cero a partir de los datos sin intervención humana. Esto evidentemente tiene grandes ventajas en cuanto a comodidad y sobre todo simplicidad para personas ajenas al campo, además para lo poco avanzado que está el campo, los resultados suelen ser buenos. Por otro lado, suelen tener en cuenta conjuntos de datos anteriormente procesados y pueden orientar la búsqueda en base a aquellos modelos para los que obtuvieron buenos resultados. Pero como desventaja principal, tenemos el aumento de los tiempos de cómputo y la falta de interpretabilidad en algunos casos. Como la mayoría de soluciones de este tipo utilizan técnicas de búsqueda bayesiana o mediante algoritmos genéticos, los tiempos de ejecución para obtener buenos resultados se hacen bastante grandes, sobre todo si se permite un espacio de búsqueda muy grande. En general, las librerías que los implementan tienden a ofrecer como parámetro el tiempo máximo de cómputo y adaptan el espacio de búsqueda en función del mismo. Comentaremos alguno de los que se han utilizado.

### 5.4.1. AutoSKLearn

Esta librería de código abierto proporciona un método de automl mediante búsqueda bayesiana, utilizando los modelos y métodos de scikit-learn, además construye el modelo final como un ensemble ponderado de los modelos que ha optimizado. La ventaja del uso de scikit-learn y de su transparencia con el modelo final, es poder tomarla como punto de partida para encontrar unos modelos de base con los que seguir trabajando. En mi caso, me fue útil para descubrir e investigar más en el modelo de Boosting basado en histogramas. Además los resultados obtenidos en uno de los intentos fueron bastante positivos, obteniendo un 0.541 en test.

### 5.4.2. AutoKeras

Esta librería, trata de buscar automáticamente arquitecturas de redes neuronales, introduciendo variaciones en el número de neuronas, en el número de capas, en la tasa de aprendizaje, en el optimizador o en la inclusión de técnicas de Dropout o BatchNormalization. Los resultados obtenidos son similares a los obtenidos por optimización manual del perceptrón multicapa de SKLearn. Además los resultados de validación son algo engañosos porque proporcionan una estimación del error de test superior a lo que realmente se obtiene, esto podría ser a causa de técnicas como Dropout que desactivan algunas neuronas durante el entrenamiento y los resultados de la red final, deben entenderse como los de la arquitectura promedio de apagar o encender dichas neuronas.

### 5.4.3. Google AutoML VertexAI

Finalmente se ha optado por utilizar la versión de prueba del sistema de automl de Google, en general la ingestión de datos ha sido algo más complicada, pues necesita de un formato de CSV específico, debe configurarse un bucket de almacenamiento en la nube otorgando unos permisos de acceso al mismo al sistema. Además la optimización se ha prolongado bastante más que la hora que se había solicitado. Por otro lado, para la predicción, la salida era un conjunto de probabilidades de pertenencia a cada clase que se han devuelto en csvs de forma desordenada tanto las columnas como las filas, teniendo que realizar un proceso de búsqueda y unión de los datos con sus predicciones. Además al no estar integrado en una librería no disponemos del modelo de forma fácil, debemos interactuar siempre con la aplicación web y no podemos realizar un postprocesado de forma fácil o integrar el modelo en otro de Stacking o un ensemble.

Curiosamente estas dificultades nos hicieron detectar la duplicación de instancias que hemos comentado en la sección de preprocesado.

Los resultados, sin embargo, son muy buenos, de hecho los resultados en test son los mejores empatados con otros que obtuvimos manualmente con otro método que se explicará en siguientes secciones.

## 5.5. Postprocesado: Calibrado de Probabilidades

El calibrado de probabilidades es un método de preprocesado que puede utilizarse para adaptar las probabilidades de pertenencia a una clase obtenidas por un clasificador, puede ser útil en casos como el nuestro porque nuestros datos están desbalanceados y de entre dos probabilidades muy cercanas es posible que se decida incorrectamente la clase minoritaria.

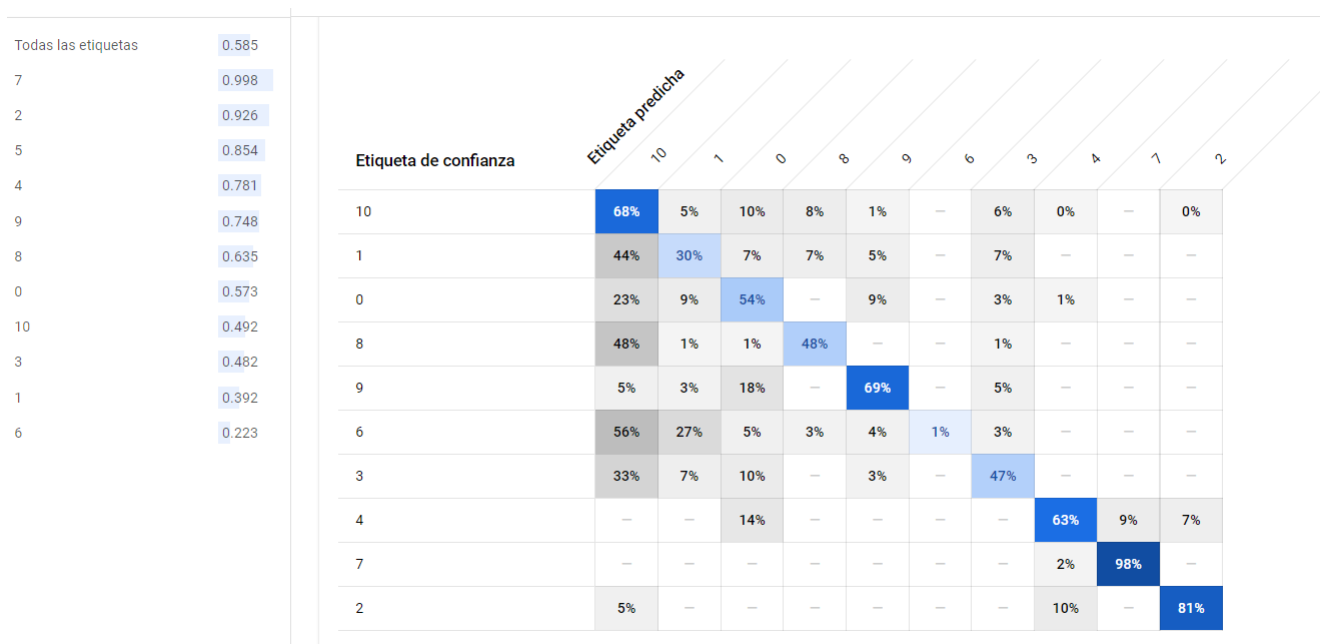


Figura 1: Resultados de Google AUTOML en validación

Usaremos el objeto Calibrated CV para llevar a cabo este calibrado, se utiliza el método de Platt que a grandes rasgos lleva a cabo una optimización de máxima verosimilitud. Se ha utilizado junto con algunos modelos, sobre todo de árboles, cuyas probabilidades se calculan por aproximación, obteniendo resultados buenos en algunos casos, elevando el acierto en validación y test.

## 5.6. Stacking

Finalmente, una vez se habían hecho pruebas con una gran variedad de modelos se optó por utilizar la técnica de Stacking para agrupar la potencia estimadora de los modelos ya optimizados. El método es sencillo, se trata de aprovechar las salidas de varios clasificadores para que, a través de otro clasificador final que las tenga como entrada, se calcule como salida una predicción conjunta, podemos considerarlo como otro método de ensemble. En nuestro caso, se probaron varias combinaciones, entre las cuales destaca la que mejor resultado ha obtenido de la práctica, que unificaba 5 ensembles de árboles: Random Forest, GradientBoosting, XGBoost, LightGBM e HistGradientBoosting.

Como clasificador final se suele usar una regresión logística, para la cual hemos necesitado aumentar el número de iteraciones por defecto porque obteníamos un warning de no convergencia.

## 5.7. Subidas erróneas

Como comentario de la tabla de subidas, hemos detectado a posteriori, el motivo de las subidas que tienen un score en test menor a 0.35. Por error en el orden de ejecución en el notebook, se olvidó cambiar una variable y no se habían aplicado las transformaciones de training a los datos de test, por ello el clasificador no conseguía buenos resultados. Además la máquina de soporte vectorial, también ha obtenido resultados extrañamente muy bajos. Por otro lado, la última subida, trataba de aunar los resultados de los dos mejores modelos, tratando de aumentar el valor de una predicción frente a otra en las diferentes, cuando se trataba de una clase mayoritaria frente a la otra bajo la asunción de que las distribuciones de clases de entrenamiento y test serían similares, sin embargo tras analizar los datos se observó que se realizaba una predicción muy excesiva de las 2 clases mayoritarias, un Stacking habría sido mejor solución pero como hemos comentado antes, el modelo de Google, no permite su exportación a Python.

## 6. Modelos finales

Como conclusión incluimos una pequeña tabla con los 2 mejores modelos obtenidos y sus métricas, considerando que el modelo Stacking es mas fiable pues se construyó eliminando las 2000 instancias con etiquetas probablemente ruidosas.

31	04/01 17:08	2	-	0.558	0.56237	Google AutoML 1 hora de cómputo gratuita
36	05/01 21:08	4	0.727	0.600	0.56237	Stacking( HistGB + XGB + RandomForest + GB + LightGBM)

Es de reseñar, que en todos los modelos en los que se eliminaron los duplicados, la fiabilidad del score de validación disminuyó, anteriormente solía ser una cota inferior, pero en estos casos no lo es, de nuevo nos quedamos con la duda de si los propios datos de test puedan tener etiquetas ruidosas.

Como conclusión, se ha obtenido un modelo quizá un poco complejo, aunque no muy lento de entrenar dada la cantidad de datos, pero bastante bueno en comparación con el resto. Por otro lado, lamentamos haber observado las etiquetas ruidosas tan tarde y no haber tenido tiempo de encontrar la mejor forma de procesarlas que no fuera eliminarlas.

## 7. Bibliografía

- Documentación de scikit-learn y XGBoost
- Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning <https://arxiv.org/abs/2007.04074>
- <https://machinelearningmastery.com/calibrated-classification-model-in-scikit-learn/>
- [https://autokeras.com/tutorial/structured\\_data\\_classification/](https://autokeras.com/tutorial/structured_data_classification/)
- <https://towardsdatascience.com/a-guide-to-xgboost-hyperparameters-87980c7f44a9>
- <https://towardsdatascience.com/how-to-deal-with-imbalanced-multiclass-datasets-in-python-fe0bb3f2b669>