Facultad de ingeniería de la UBA

75.06/95.58 Organización de Datos

Trabajo práctico N°2
Machine Learning

Primer cuatrimestre de 2021

Predicción del nivel de daños causado por el terremoto de Nepal en 2015

Integrante	Padrón	Correo electrónico
Gutiérrez, Matías	92172	magutierrez@fi.uba.ar
Julián Rodrigo Cropano Miranda	103960	jcropano@fi.uba.ar
Alexis Brian, Herrera Aguilar	104639	abherrera@fi.uba.ar
Tomás Yavicoli	97617	tyavicoli@fi.uba.ar

Nombre del grupo: *DataStalkers*
Link al repositorio:
git@github.com:gutierrezmatias/Organizacion-de-Datos.git

Índice

1. Introducción **1**

2. Feature Engineering **2**
2.1. Introducción 2
2.2. Feature Importance inicial 2
2.3. Dropeo de columnas 3
2.4. Ajustar valores de columnas 4
2.5. Media de daños por localización 5
2.6. Nuevo análisis tras transformaciones iniciales 7
2.7. Superestructuras 8
2.7.1. Superestructuras mean damage 8
2.7.2. Superestructuras y age 10
2.8. Age 12
2.9. Features restantes 13
2.9.1. Altura 13
2.9.2. Area 14
2.10. Dataset Final 14

3. Tuneo de hiperparámetros **16**
3.1. Optimización Bayesiana 16
3.2. Random Search 20

4. Análisis de predicciones **23**
4.1. Predicciones correctas e incorrectas 23
4.2. Planteo de mejoras al modelo 24

1. Introducción

El siguiente trabajo se dividirá en 3 partes

- Feature Engineering: Se mejora el set de entrenamiento, para que el algoritmo de Machine Learning pueda predecir con mayor precisión el daño en los edificios.
- Tuneo de hiperparámetros: Se evalúa que hiperparámetros mejoran el aprendizaje y se realiza una búsqueda automatizada de los valores que lo optimizan.
- Análisis de predicciones: Se realiza un análisis de los resultados predichos por el algortimo. Se plantean posibles mejoras.

2. Feature Engineering

2.1. Introducción

En esta sección se explicarán las transformaciones que se aplicaron al dataset de entrenamiento original para mejorar las predicciones.

Se debe tener en cuenta que para esta sección se utilizó XGBoost con los hiperparámetros por defecto como baseline. De esta forma podemos ir probando y analizando si nuestras transformaciones son útiles o no.

También cabe aclarar que las modificaciones son consecutivas. Es decir, los cambios se van aplicando en el orden que se exponen en este informe, y permanecen hasta el final en caso de que mejoren nuestro modelo.

2.2. Feature Importance inicial

Para poner un punto de partida, se entrenó al algoritmo con el dataset original y One Hot Encoding para las columnas categóricas.

Con este dataset inicial, el score de training es de 0.7452 y de test 0.7295. Es un buen score para un modelo con los hiperparámetros no optimizados, por lo que podremos enfocarnos por ahora solo en el Feature Engineering .

El beneficio adicional de hacer esto, es que tenemos la importancia de cada columna [1]. Esto nos va a permitir entender que features son posibles columnas redudantes, o cuales podemos mejorar.

De este gráfico podemos sacar las siguientes conclusiones rápidas:

- Hay columnas que son sumamente utilizadas, como las de geo_level_id, area_percentage y age.
- También columnas que no lo son tanto como plan_configuration y las variantes de has_secondary_use.
- La importancia está poco distribuida. Es decir, hay columnas que se utilizan muchísimo más que otras.

[1] Inicialmente se utilizó la frecuencia para evaluar la importancia

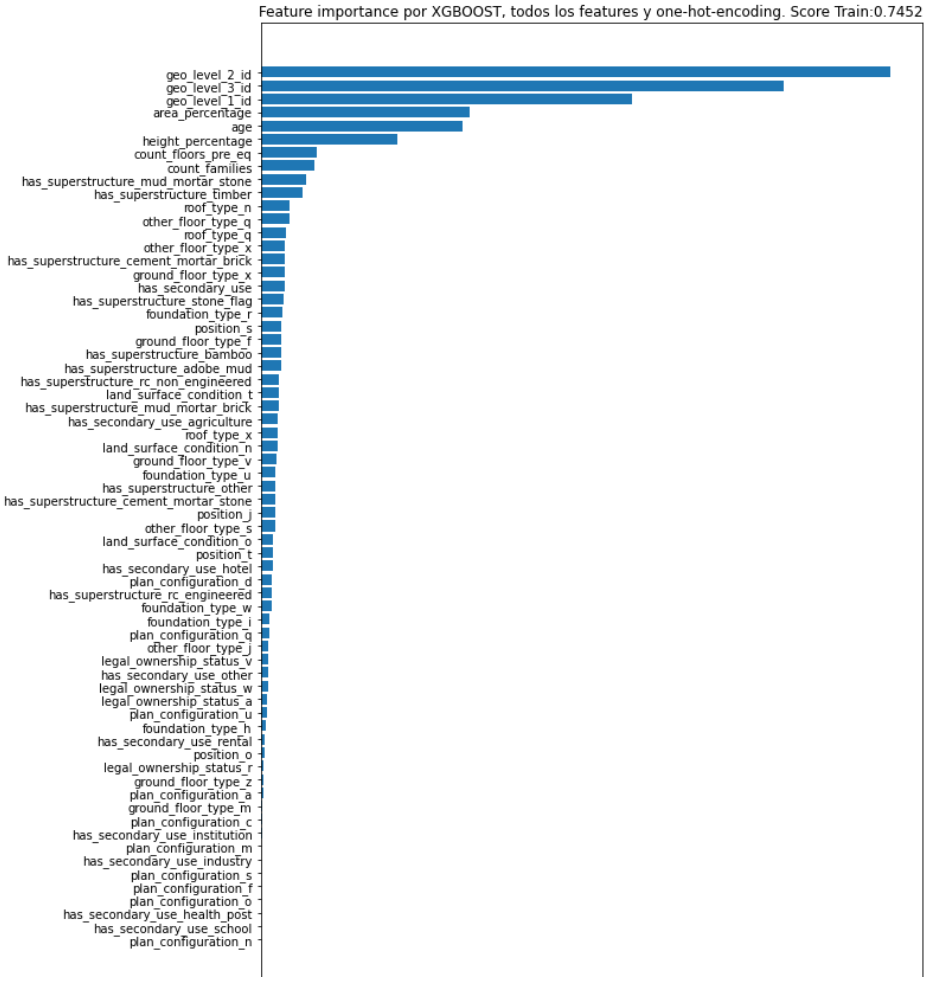

Figura 1: Feature Importance Inicial

2.3. Dropeo de columnas

Como se mencionó antes, hay columnas que casi no se utilizan para predecir. En general, porque hay muy pocos valores. Por ejemplo las variantes de has_secondary_use y plan_configuration.

Se realizaron las pruebas sin estas columnas (permutando) y en todas mejora el score de entrenamiento si no están.

Es decir, se logró mejorar el score de entrenamiento hasta 0.7467 (de 0.7452) sin incluir las variantes de has_secondary_use (solo dejamos la columna booleana has_secondary_use) y sin incluir plan_configuration. Respecto al score de Test mejoró a 0.7304 (de 0.7295).

La conclusión que sacamos de esto, es que tal vez estas columnas al ser tan

pocas, obliguen a overfittear al modelo, siendo estos casos muy particulares.

Por lo tanto, decidimos retirar estas columnas en lo que resta del análisis ya que es un buen trade-off: 20 columnas menos mejorando (un poco) el score.

2.4. Ajustar valores de columnas

Como se mencionó en el TP1, las columnas age, area_percentage y height_percentage tienen una distribución inclinada hacia la izquierda.

Utilizamos winsorizing para ajustar estas columnas en los valores máximos. Este método reemplaza a todos los valores mayores a cierto percentil por el valor de ese percentil. Se utilizó para age y area_percentage 0.03 y 0.06 para height_percentage. En los gráficos se puede observar esto como un pico al final de cada columna.

Para decidir estos valores se corrio el algoritmo en los siguientes rangos, los cuales nos parecieron razonables gráficamente:

- age: [0.01,0.05]. 5 valores
- area_percentage: [0.03,0.06]. 4 valores
- height_percentage: [0.03,0.06]. 4 valores

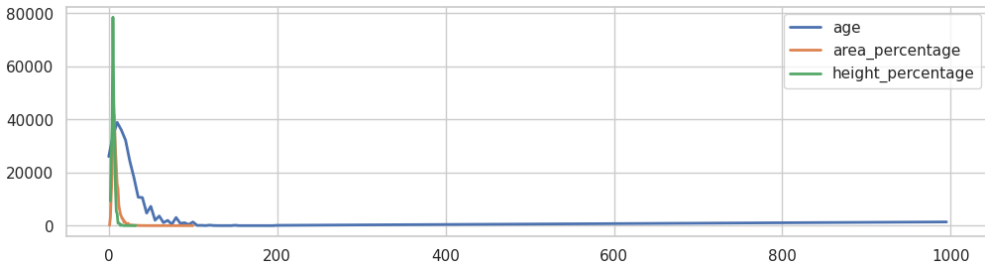

Figura 2: Distribución original age, area_percentage y height_percentage

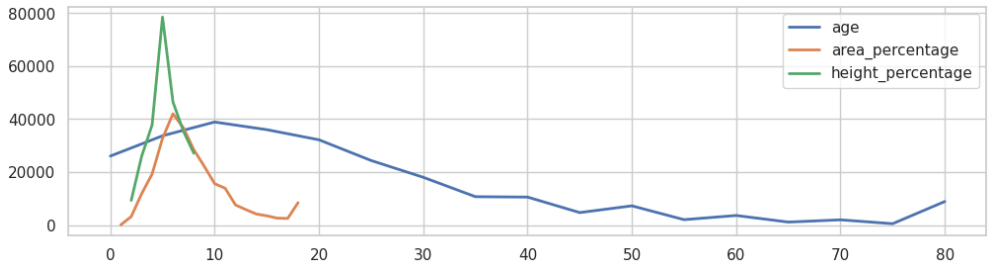

Figura 3: Distribución de age, area_percentage y height_percentage winsorized

Este cambió mejoró el score de entrenamiento a 0.7473 (anterior 0.7467). Respecto al score de Test mejoró a 0.7307 (de 0.7304).

Concluimos con lo siguiente:

1. Para llegar a este resultado, inicialmente se planteó la hipótesis de que —la mejor distribución de los valores de las columnas es la combinación del mejor resultado individual.— Sin embargo esto resultó ser erróneo. (Combinar los mejores scores individuales resultó en un score de 0.7454, peor a no hacer nada)

2. Como el height_percentage óptimo quedó en el borde del rango de pruebas, se podría pensar que ampliando el rango mejore. Se probó hasta 0.08 sin mejora alguna.

3. Si bien la mejora es ínfima, estos valores servirán para las transformaciones que utilicen estos valores. Evitando que los outliers se expandan por el dataset. Por ejemplo una media que utilice alguna de estas columnas.

2.5. Media de daños por localización

A continuación vamos a trabajar con uno de los features más importantes en términos de predicción de daños, la localización de los edificios.

Como vimos en el TP1, hubo zonas donde el terremoto afectó de manera más grave que otras. Esta información se puede incluir ya que los terremotos, en general, tienen un epicentro y ese no es un dato particular de nuestro dataset, por lo que en teoría no debería overfittear.

Entonces se hizo la primera prueba de nuestra hipótesis, se creó una nueva columna llamada mean_high_damage_geo_level_1_id, la cual tendrá la media de daños graves en esa localización de nivel 1.

Por ejemplo, en la localización geo_level_1_id número 17, la media de daños graves es del 81 % (una de las más afectadas).

Esta primera prueba nos dió un score de entrenamiento 0.7463 (de 0.7473) y 0.7308 (de 0.7307) en el test.

En principio este es un resultado aparentemente desfavorable. Sin embargo, como aprendimos anteriormente, es probable que para esta combinación de transformaciones justo empeore un poco el entrenamiento. Por lo tanto seguimos explotando este concepto.

Fuimos aplicando esto a distintos geo_level_id y pudimos notar que el score mejoraba cuanto más específico era el geo_level_id.

Luego notamos que combinando los geo_level lográbamos una aún mejor precisión.

Geo level id	Train Score
1	0.7463
2	0.7529
3	0.7689
1 y 2	0.7531
2 y 3	0.7691
1,2 y 3	0.7690

Tabla 1: Train score al agregar columnas solo del tipo mean_high_damage

Esto nos daba la pauta de que el algoritmo tenía bien en cuenta estas medias calculadas para decidir. Por lo tanto, por ahora solo tiene la probabilidad de que reciba un daño grave o no grave.

Entonces podemos darle las medias de daño leve y medio para que pueda tener aún más detalle. Es decir en total agregaremos 9 columnas (por los 3 tipos de daño y por los 3 niveles de geo_level_id).

Geo level id	Train Score
1	0.7468
2	0.7548
3	0.7755
1 y 2	0.7542
2 y 3	0.7761
1,2 y 3	0.7766

Tabla 2: Train score al agregar columnas mean_damage para los 3 tipos de daños

A continuación se puede ver un ejemplo de las columnas agregadas. En este caso en geo_level_1_id.

building_id	geo_level_1_id	mean_low_damage_geo_level_1_id	mean_medium_damage_geo_level_1_id	mean_high_damage_geo_level_1_id
802906	6	0.09	0.67	0.25
28830	8	0.03	0.45	0.52
94947	21	0.02	0.39	0.58
590882	22	0.13	0.74	0.13
201944	11	0.05	0.57	0.38
...	...	...	...	...
688636	25	0.08	0.78	0.14
669485	17	0.01	0.18	0.81
602512	17	0.01	0.18	0.81
151409	26	0.35	0.56	0.09
747594	21	0.02	0.39	0.58

Figura 4: Ejemplo de las nuevas columnas mean_geo_damage para geo_level_id_1

Como se puede ver en la segunda tabla, la que mejor score nos da es la que combina todas las columnas de mean_damage y geo_level_id. Por lo tanto ahora tenemos 9 nuevas columnas.

Las conclusiones son las siguientes:

1. Mejoramos el score de entrenamiento a 0.7766 (0.7473) y el score de test a 0.7444 (de 0.7307). La mejor hasta ahora.

2. De estos resultados, empezamos a notar una distancia mayor entre el score de entrenamiento y del test. Es probable que los edificios en el test estén mejor distribuidos geográficamente, generando una brecha entre estos scores. Sin embargo no implica que el modelo este overfitteando.

3. Aún es conveniente conservar las columnas de geo_level_id. Si las reemplazamos por estas columnas perdemos un poco de score (0.7747 train y 0.7441 test).

2.6. Nuevo análisis tras transformaciones iniciales

Para poder decidir que transformaciones faltan hacer, volvemos a analizar el feature importance. Pero esta vez, calculado a través del *gain*.

Podemos ver que la localización ahora es mucho más relevante que antes. Le dimos mayor valor a estos features.

Decidimos continuar con las columnas de superstructure, ya que hicimos un análisis en el TP1 y pudimos ver que hay superestructuras más resistentes que otras.

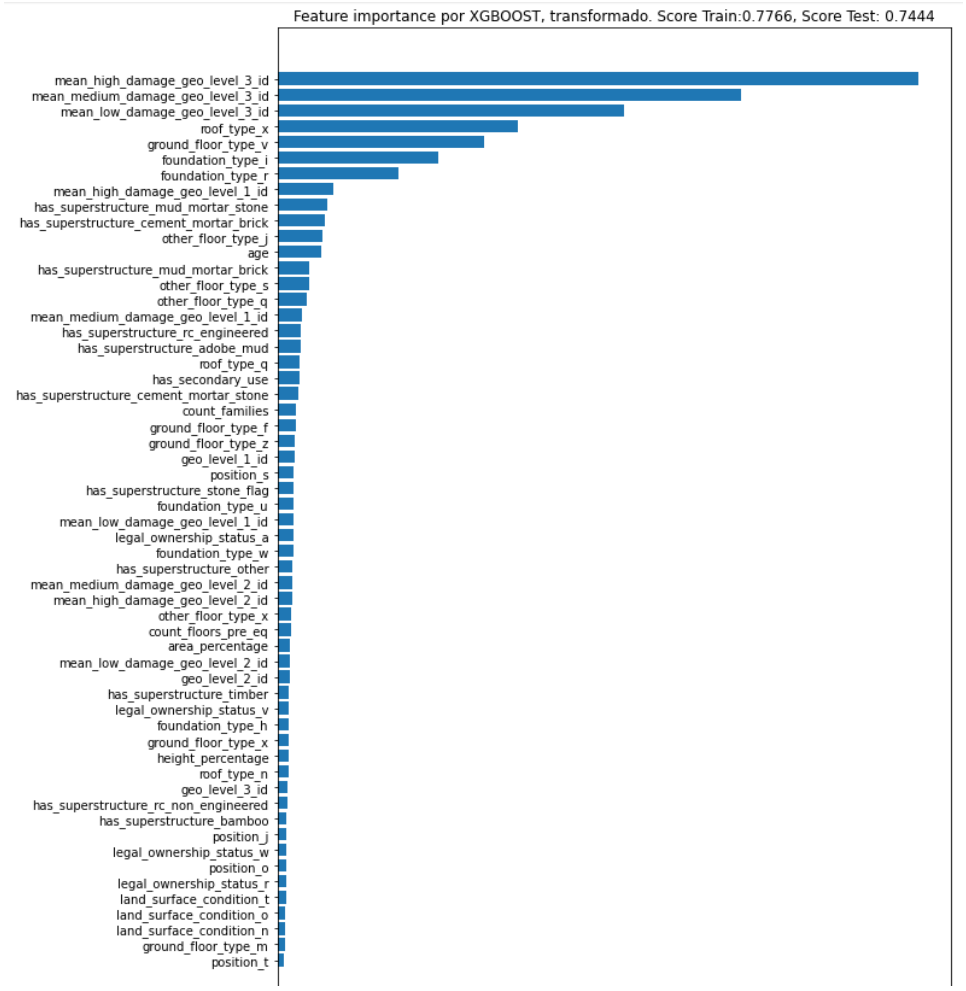

Figura 5: Feature Importance post mean_damage

2.7. Superestructuras

2.7.1. Superestructuras mean damage

Antes de aplicar alguna transformación a las superestructuras recordemos como resistieron a el terremoto. Además es importante ver la cantidad de cada tipo para ver si tendremos problemas de overfitting.

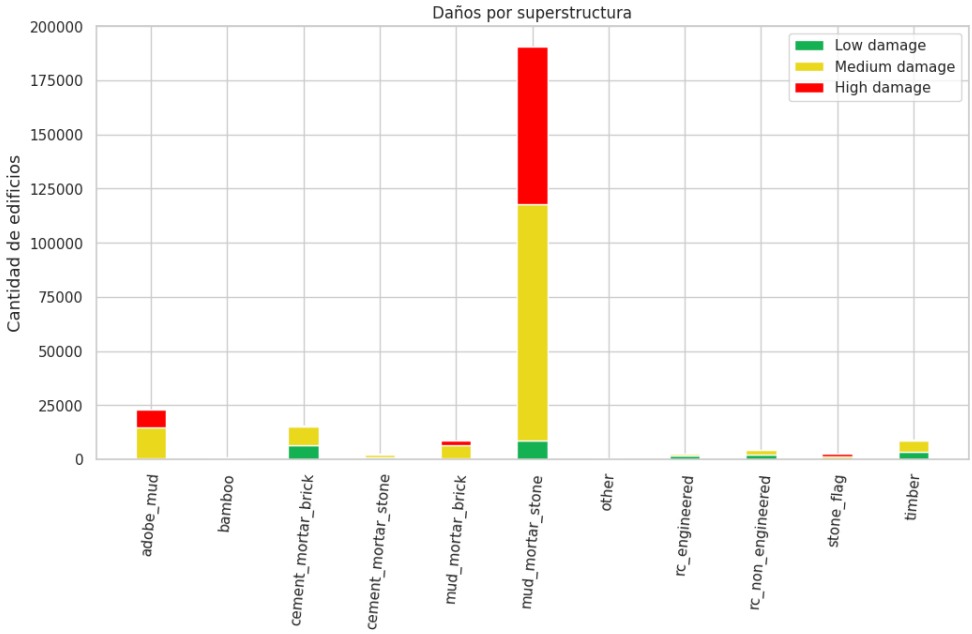

Figura 6: Cantidad de edificios de cada superestructura y la distribución del daño

Con este gráfico podemos decir que la superestructura más representativa es mud_mortar_stone. Esto nos generar un problema de overfitting para aquellas superestructuras que tengan poca representatividad.

Sin embargo todas las superestucturas tienen una distribución de los daños no uniforme. Por lo tanto, vale la pena intentar una transformación.

Como hicimos anteriormente, agregaremos las columnas mean_low_damage, mean_medium_damage y mean_high_damage. Estas columnas tendran el porcentaje de daño que tuvo esa superestructura. Por ejemplo: Un edificio que tenga superestructura mud_mortar_stone tendra 0.05 en mean_low_superstructure, 0.57 en mean_medium_damage y 0.38 en mean_high_damage.

Sin embargo esta transformación da un score de train 0.7768 (de 0.7766) y un score de test 0.7433 (de 0.7444).

Podemos concluir sobre esta transformación en lo siguiente:

1. Pese a darle más información sobre el daño que recibe cada tipo de estructura, no mejoró la predicción.

2. Se analizó la opción de que sea a causa de la cantidad de columnas (59 columnas post transformación). Para descartar esto sacamos las de has_superstructure. Sin embargo el score de training es 0.7762 y score de test es 0.7428. Aún peor que con 59 columnas.

3. Otra posible causa es que la profundidad de los árboles es muy poca para aprovechar esos valores. Esto podremos probarlo cuando optimicemos los hiperparámetros.

4. No utilizaremos esta transformación en lo que resta de Feature Engineering ya que no es una mejora.

2.7.2. Superestructuras y age

Antes de abandonar las superestructuras, vamos a intentar un nuevo feature.

Nos basamos en el análisis del TP1, donde vemos que para cada material existe una tendencia a aumentar el daño cuanto más antiguo es el edificio.

Esto no es lo único, ya que cada superestructura tiene una media distinta, por lo que puede ayudar a comprender aún más a las superestructuras. Esto se puede observar en el siguiente gráfico.

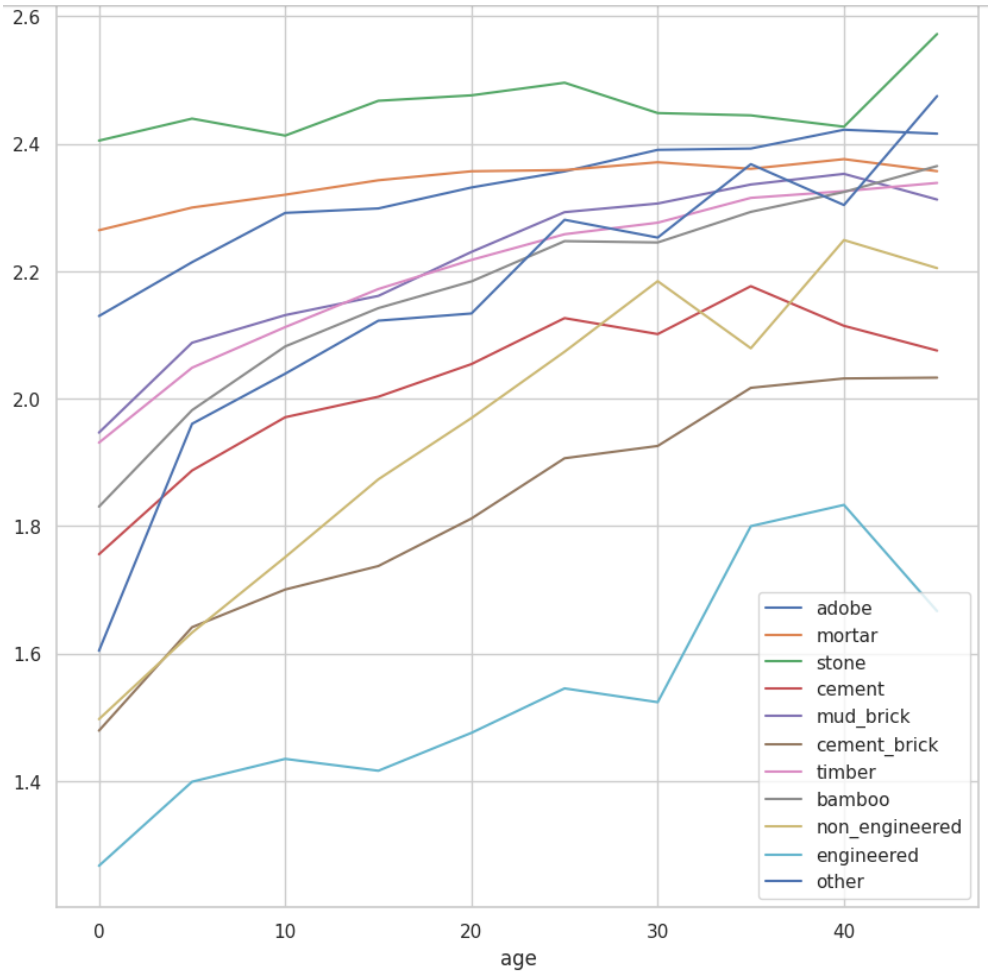

Figura 7: Media de daño de las superestructuras según la edad del edificio

Para probar este nuevo feature, crearemos una nueva columna llamada mean_damage_superstructure_age. El cual para cada edificio tendrá la media de daño para ese rango de edad.

Recordemos que la edad ahora se distribuye en 17 buckets empezando de 0 hasta 80 de 5 en 5 debido ajuste hecho anteriormente.

Para ejemplificar, un edificio de age 0 y superestructura mud_mortar_stone tendrá 2.42 de media en la nueva columna (como se puede observar en el gráfico anterior).

Esta transformación da un score de 0.7771 (de 0.7766) y score de test 0.7438 (de 0.7444).

Concluimos nuestro análisis de superestructuras con lo siguiente:

1. Similar al caso anterior, no mejora el score de test pero sí el de entre-

namiento. Podemos llegar a sospechar que la nueva información agrega ruido al data set, y que es suficiente con las columnas que ya tiene.

2. Nuevamente el factor que mud_mortar_stone sea el material más utilizado nos lleva inevitablemente al overfitting para el resto de materiales. Esta es la razón la cual estemos empeorando el score de test.

3. Un detalle, rellenamos los valores de la nueva columna en el set de test con la media global del set de entrenamiento. Esto pasa porque hay combinaciones de age/superstructure que aparecen en el set de test pero no en el de entrenamiento.

4. Tampoco utilizaremos esta transformación en lo que resta de Feature Engineering.

2.8. Age

La columna age aporta información valiosa respecto a la predicción de daños ya que las viviendas más antiguas son más propensas a daños graves.

Veamos el siguiente gráfico para ver como quedó distribuida tras aplicar *winsorize* a la columna age.

Aplicaremos la misma técnica de mean_damage para cada tipo de daño.

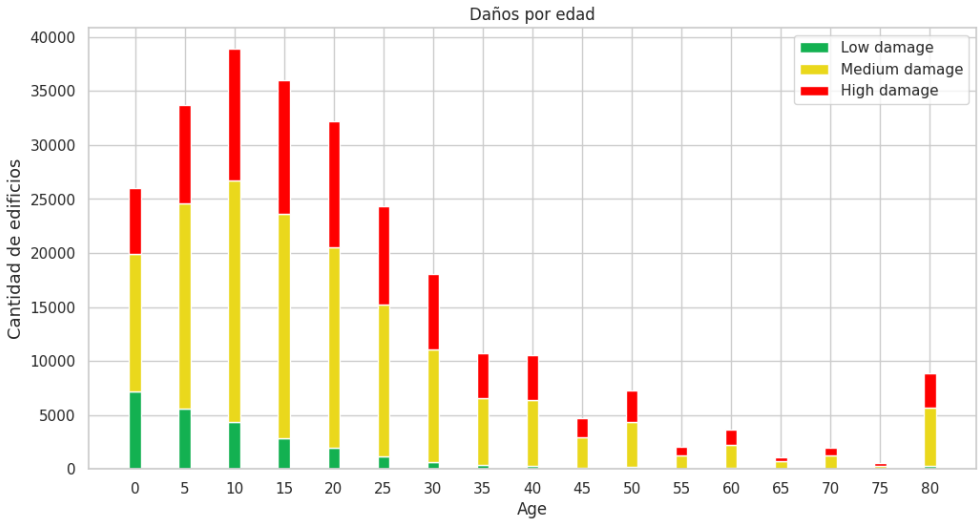

Figura 8: Daño por age winsorized y cantidad de edificios

Sin embargo el resultado no es el esperado. Se obtuvo un score de entrenamiento 0.7465 (de 0.7766) y un score de test de 0.7442 (de 0.7444)

Una conclusión que podemos sacar de esto es que la columna edad es suficiente para predecir bien el tipo de daño.

Esto tal vez se deba a que la edad es númerica y está ordenada y por lo tanto sea suficiente información relevante para el algoritmo. Información adicional sobre la edad es irrelevante o ruido.

2.9. Features restantes

Finalizaremos el Feature Engineering con las columnas restantes.

Estas columnas no tenian una tendencia marcada respecto al daño cuando las trabajamos en el TP1. Sin embargo a veces no podemos ver cosas que el algoritmo sí.

2.9.1. Altura

Haremos algo similar a lo que hicimos con geo_level_id . Como se puede ver en la imagen, no existe una tendencia marcada sobre lo que se podría predecir.

Aunque tal vez haya una diferencia al *winsorize* esta columna para los valores extremos (realizado anteriormente).

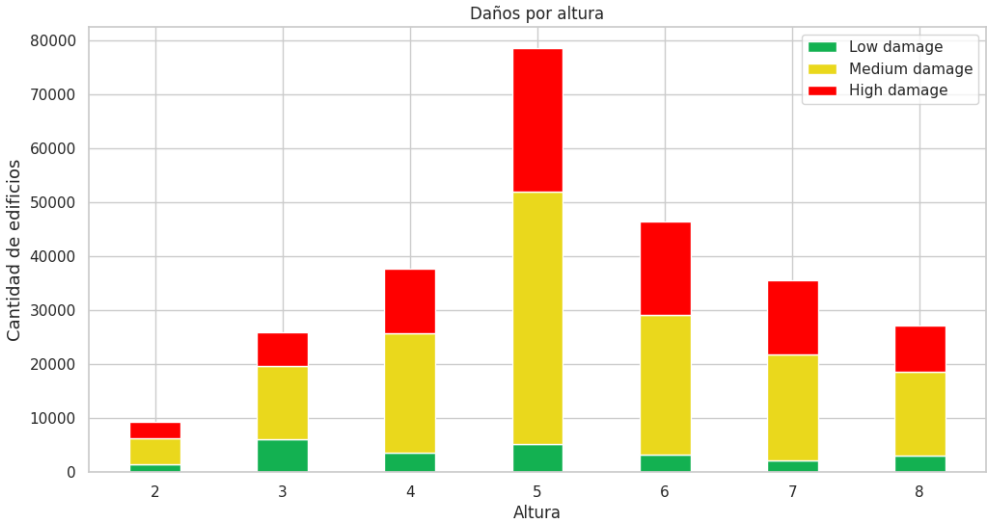

Figura 9: Daño por altura winsorized y cantidad de edificios

Al agregar estas tres columnas de mean_damage para cada tipo de daño, se obtuvieron los siguientes scores: train score 0.7765 (de 0.7766) y test score 0.7443 (de 0.7444).

No hubo mejora, pero tampoco empeoró demasiado. Decidimos no incluir esta transformación.

2.9.2. Area

Aplicaremos lo mismo que hicimos con la altura pero al area_percentage, ya que nuevamente al principio de este análisis aplicamos *winsorize* a los valores.

El gráfico es el que se muestra a continuación. En el mismo se puede observar como se agrupan todos los valores mayores a 18 en 18.

Decimos que no hay una tendencia marcada respecto al daño ya que –a la vista– las barras tienen la misma proporción de daño.

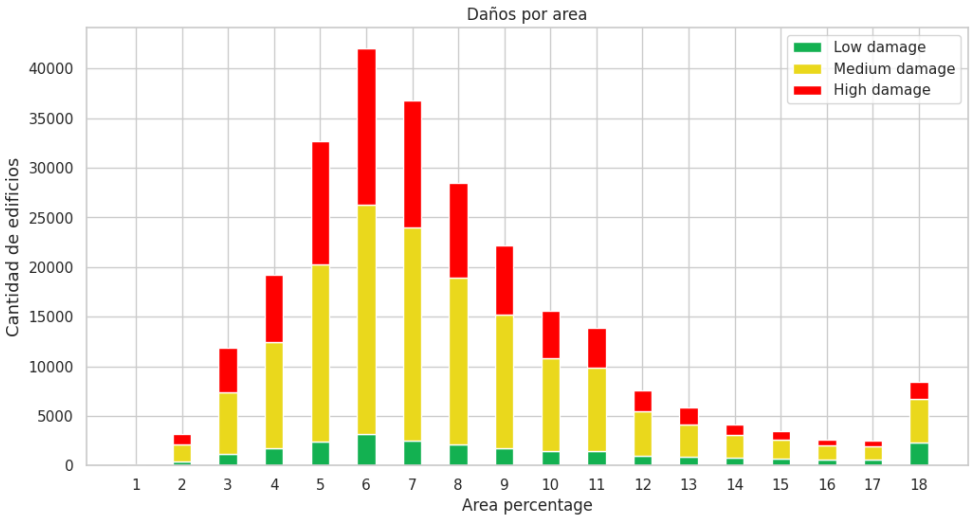

Figura 10: Daño por area percentage winsorized y cantidad de edificios

Se obtuvieron los siguientes scores: train score 0.7765 (de 0.7766) y test score 0.7445 (de 0.7444).

Es practicamente el mismo score que con la altura. Sin embargo mejora un poco el score de test. Conservaremos este cambio.

2.10. Dataset Final

Usando nuestro modelo baseline obtuvimos el mejor score de test con las siguientes transformaciones en el dataset:

- One Hot Encoding para las features categóricas.
- Dropeo de columnas: has_secondary_use (variantes) y plan_configuration.

- Ajuste de valores extremos para las columnas age, area_percentege y height_percentage.
- Nueva columna: Mean_geo_damage para cada geo level y para cada tipo de daño.
- Nueva columna: Mean area damage.

Cuenta con 60 columnas, un score de entrenamiento 0.7765 (empezamos con 0.7452) y score de test de 0.7445 (iniciamos con 0.7295).

Comentarios adicionales:

- Como XGBoost está basado en árboles de decisión, no necesita normalizar valores. Por lo tanto, no hicimos esa transformación.
- Utilizar como referencia de mejora un score no es lo mejor, ya que podremos overfitear y pensar que mejoramos. Sin embargo como se verá en la siguiente sección, el score de entrenamiento es muy buena referencia.

3. Tuneo de hiperparámetros

Para empezar, mostraremos los hiperparámetros más relevantes de nuestro baseline en XGBoost.

Nombre del hiperparámetro	Valor
n_estimators	100
max_depth	6
learning_rate	0.3
gamma	0
subsample	1
max_delta_step	0
colsample_bytree	1
min_child_weight	1

En esta sección utilizaremos 5-Fold Cross Validation para evitar overfitting al buscar nuestros hiperparámetros.

Con este modelo, obtenemos una media de score: 0.761904. Si encontramos mejores hiperparámetros, deben mejorar esta media al realizar 5-Fold Cross Validation.

El problema que tenemos es el siguiente: existen demasiados hiperparámetros que podemos mejorar. Por lo tanto encontrar la combinación óptima puede tomar demasiado tiempo si no tomamos las decisiones correctas.

3.1. Optimización Bayesiana

Resumidamente, lo que hace este algoritmo es buscar el óptimo global mediante una técnica basada en el Teorema de Bayes. Es decir, utiliza los resultados de iteraciones anteriores para mejorar la función objetivo (score).

Esto nos va permitir descartar rangos que no son convenientes y acotar aún más nuestro rango de búsqueda.

Iniciaremos la búsqueda en los siguientes rangos:

Tras 46 iteraciones, siendo la función objetivo la media de score de 5-Fold Cross Validation, se observan los siguientes resultados:

Nombre del hiperparámetro	Inicio	Fin	Step
n_estimators	100	400	10
max_depth	6	14	1
learning_rate	0.1	1	0.1
gamma	0	5	0.5
subsample	0.5	1	0.1
max_delta_step	0	5	1
colsample_bytree	0.5	1	0.1
min_child_weight	0.5	3	0.5

Tabla 3: Rangos de búsqueda por Optimización Bayesiana

trial_number	score	max_depth	n_estimators	learning_rate	gamma	subsample	max_delta_step	colsample_bytree	min_child_weight
21	-0.7630	13.0000	330.0000	0.1000	5.0000	0.9000	1.0000	0.5000	1.0000
36	-0.7629	9.0000	340.0000	0.1000	3.0000	0.9000	0.0000	0.9000	2.0000
3	-0.7627	9.0000	330.0000	0.1000	2.5000	0.9000	4.0000	1.0000	2.0000
31	-0.7626	13.0000	130.0000	0.1000	4.5000	0.8000	1.0000	0.8000	3.0000
22	-0.7626	10.0000	330.0000	0.1000	5.0000	0.9000	1.0000	0.8000	1.0000
37	-0.7622	13.0000	340.0000	0.1000	3.5000	0.9000	0.0000	0.9000	0.0000
42	-0.7622	12.0000	290.0000	0.2000	5.0000	0.9000	1.0000	1.0000	0.0000
32	-0.7621	10.0000	150.0000	0.3000	5.0000	0.9000	3.0000	0.6000	1.0000
26	-0.7620	9.0000	100.0000	0.1000	4.5000	0.9000	4.0000	0.6000	1.0000
34	-0.7620	8.0000	370.0000	0.2000	2.5000	1.0000	1.0000	0.9000	2.0000

Figura 11: Configuración y score de los mejores 10 resultados.

En los siguientes gráficos se puede observar:

- La mejor combinación de hiperparámetros se obtuvo en la iteración 21 (empezando desde 0).
- Se puede ver como el score empieza a converger a 0.76 en cada iteración
- Se puede observar una correlacion negativa del score con el learning_rate y positiva con gamma y subsample.
- Se puede ver que los mejores scores se obtuvieron con un learning_rate de 0.1
- Valores superiores a 300 en n_estimators obtuvieron los mejores resultados
- El valor de subsample que dio mejor resultado es 0.9

- Se obtuvo el mejor test score (0.7473) con el trial_number 36. El resto fue inferior a 0.7465.

  Uilizaremos este análisis para definir los nuevos rangos cuando hagamos búsqueda por GridSearch. Fijando los menos definitorios y variando los que más mueven la aguja.

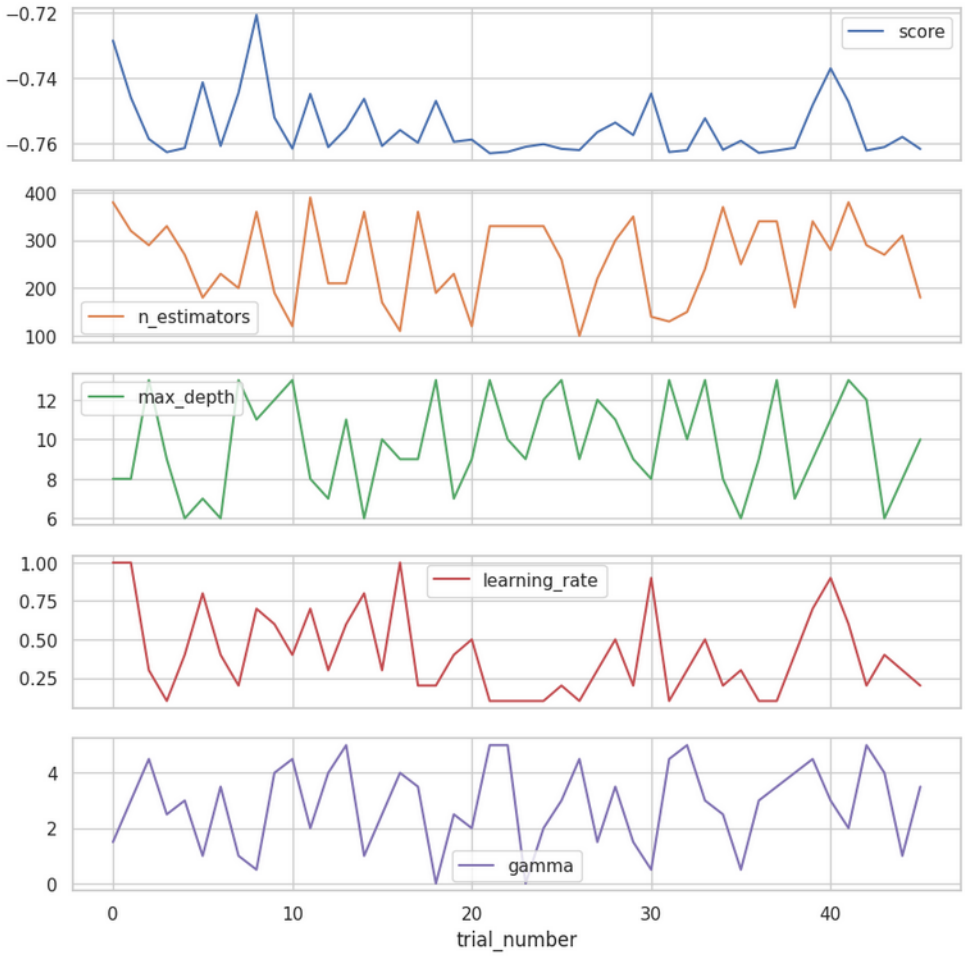

Figura 12: Variación del score y de los 4 primeros hiperparámetros

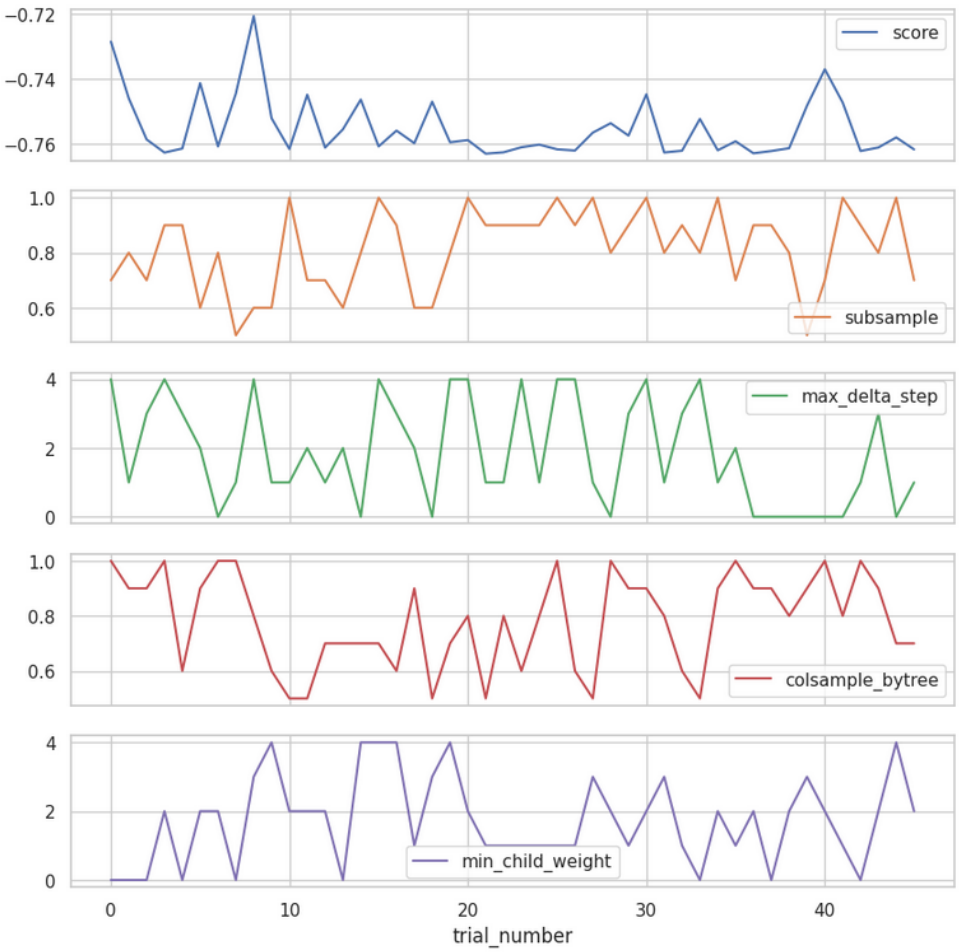

Figura 13: Variación del score y de los 5 últimos hiperparámetros

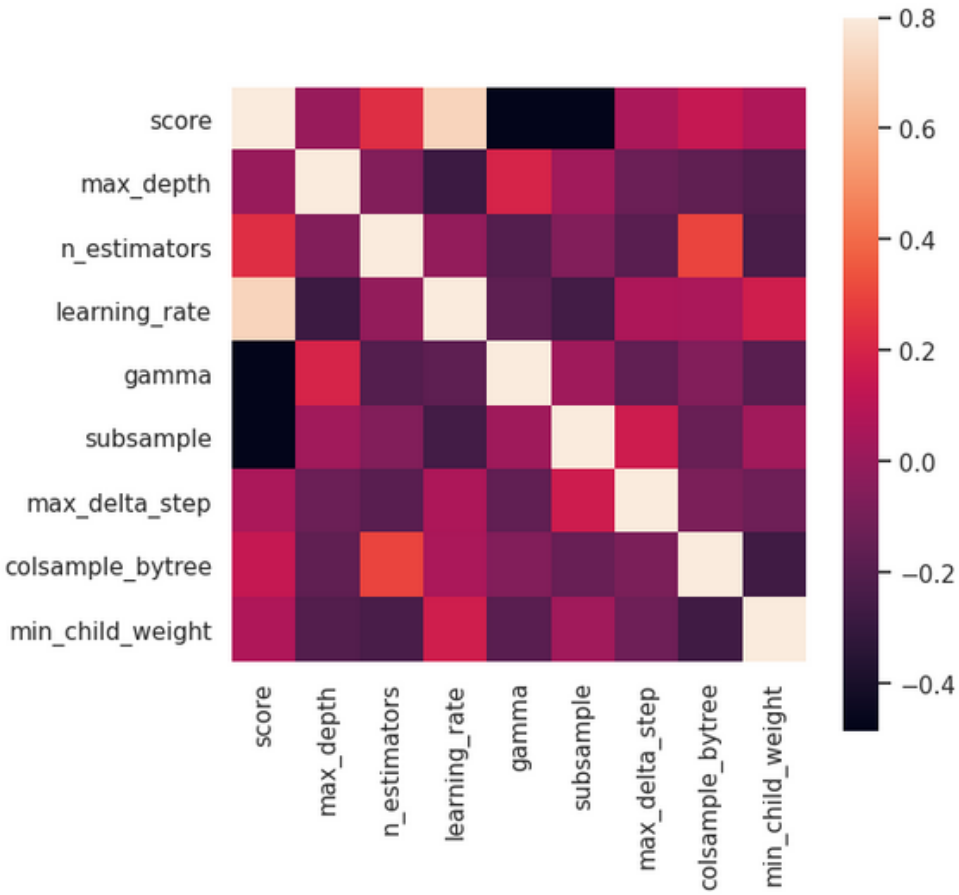

Figura 14: Correlación entre los hiperparámetros

3.2. Random Search

Para finalizar haremos una búsqueda aleatoria en un rango más acotado de los hiperparámetros que teníamos inicialmente.

Si bien, con este método, mejorar el óptimo depende de la suerte, es más probable que mejoremos en un rango que sabemos que es bueno a encontrarlo en uno muy amplio.

Entonces, se realizará la búsqueda en los siguientes rangos:

Se hicieron un total de 30 búsquedas con los siguientes resultados:

Nombre del hiperparámetro	Inicio	Fin	Step
n_estimators	290	350	5
max_depth	9	14	1
learning_rate	0.1	0.1	-
gamma	3	6	0.5
subsample	0.9	1	0.1
max_delta_step	0	3	1
colsample_bytree	0.9	1	0.1
min_child_weight	1	3	0.5

Tabla 4: Rangos de búsqueda por Random Search

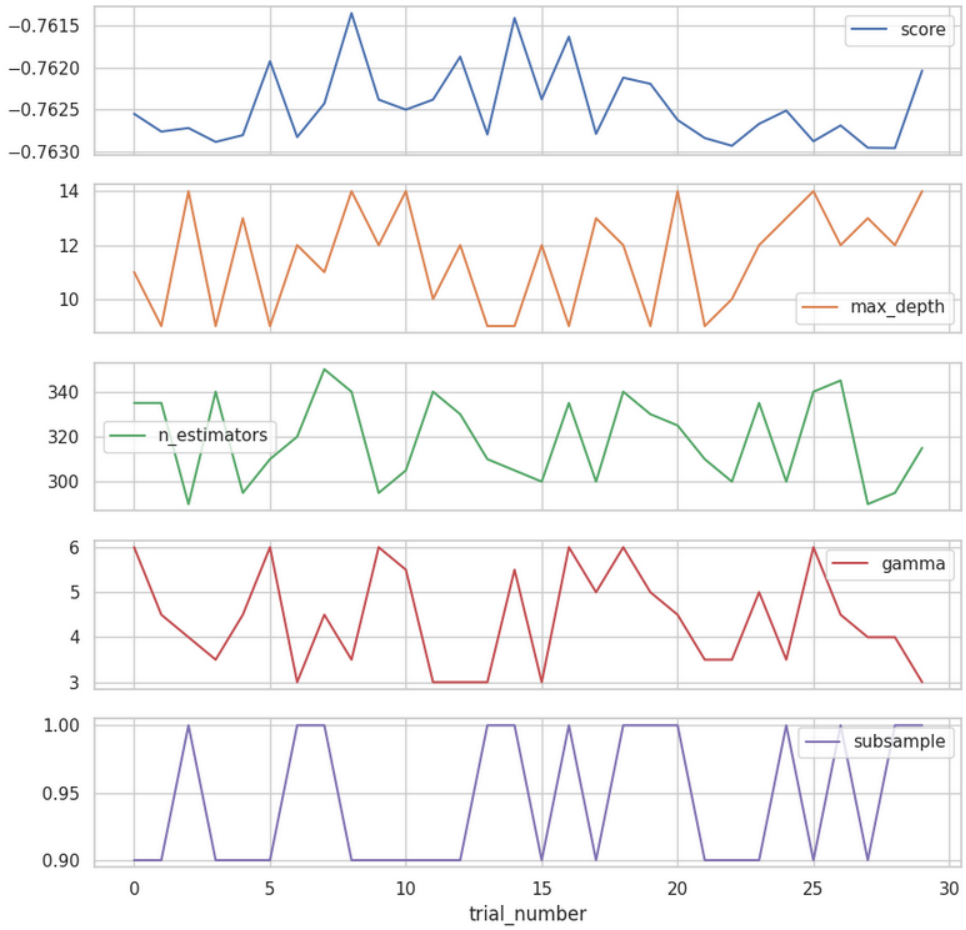

Figura 15: Variación de los hiperparámetros que más influyeron en Random Search

trial_number	score	max_depth	n_estimators	learning_rate	gamma	subsample	max_delta_step	colsample_bytree	min_child_weight
28	-0.7630	12	295	0.1000	4.0000	1.0000	3	0.9000	3.0000
27	-0.7630	13	290	0.1000	4.0000	0.9000	1	0.9000	2.0000
22	-0.7629	10	300	0.1000	3.5000	0.9000	4	1.0000	1.5000
3	-0.7629	9	340	0.1000	3.5000	0.9000	0	0.9000	3.0000
25	-0.7629	14	340	0.1000	6.0000	0.9000	4	0.9000	2.5000
21	-0.7628	9	310	0.1000	3.5000	0.9000	3	0.9000	2.5000
6	-0.7628	12	320	0.1000	3.0000	1.0000	1	0.9000	2.5000
4	-0.7628	13	295	0.1000	4.5000	0.9000	1	1.0000	2.0000
13	-0.7628	9	310	0.1000	3.0000	1.0000	2	0.9000	1.0000
17	-0.7628	13	300	0.1000	5.0000	0.9000	2	1.0000	2.5000

Figura 16: Ranking de los 10 mejores resultados y sus configuraciones

De los cuales obtenemos las siguientes conclusiones:

- Si bien no pudimos mejorar de manera gran manera el score de entrenamiento, encontramos un piso estable de 0.7610.
- Aunque los mejores scores son similares, pueden tener resultados distintos en el test ya que los tabulados son una media del CV.
- Como se mencionaba en la sección de Feature Engineering, los hiperparámetros por defecto de XGBoost fueron suficientemente buenos como para ignorarlos en esa etapa.
- Es probable que no hayamos encontrado mejoras en el score a causa de fijar el valor de learning_rate.
- El mejor resultado de test encontrado por este método es el trial_number 28 con un score de 0.7468.

4. Análisis de predicciones

En esta sección analizaremos brevemente si estamos prediciendo correctamente. También analizaremos que alternativas podremos explorar para mejorar nuestras predicciones.

4.1. Predicciones correctas e incorrectas

Para esta sección utilizaremos el modelo que mejor score de test arrojó hasta el momento (0.7473 encontrado por Optimización Bayesiana).

De 260.601 edificios se predijeron correctamente 206.686 (79.31 %) y de manera incorrecta 53915 (20.69 %).

Analizando los resultados de manera global se notó lo siguiente:

- El error de predicción por edad, area y height están bien distribuidos. Es decir no hubo edades, areas o alturas que se predijeron mucho mejor o peor que otras.
- En las ubicaciones, también el error se distribuyó uniformemente. El error porcentual más grande de cada nivel es: 25.93 %, 42.86 y 87.50 %, respectivamente pero en estos últimos dos casos la cantidad de edificios era menor a 10 en su respectivo bloque.

Para las superestructuras y el resto de features de este tipo sucede algo similar, en general se distribuyen uniformemente.

Podemos concluir que el modelo predijo correctamente de manera global.

Ahora veamos si existe alguna tendencia al predecir, como por ejemplo equivocarse más por dar un daño leve cuando era grave. Se puede ver estos casos en la siguiente tabla.

Real/Predicho	Leve	Medio	Grave
Leve	16657	8097	370
Medio	4372	129284	14603
Grave	552	25921	60745

Tabla 5: Cantidad de edificios predichos por cada tipo de daño

Los casos donde se predice más grave de lo que es suma el 8.85 % de los casos. Su contraparte suma el 11.84 %. La diferencia es del 3 % a favor de daños leves.

El caso donde más se equivoca es cuando predice medio cuando es grave y el caso donde menos se equivoca es cuando predice grave pero es leve (0.14 %).

Real/Predicho	Leve	Medio	Grave
Leve	6.40 %	3.10 %	0.14 %
Medio	1.68 %	49.60 %	5.61 %
Grave	0.22 %	9.94 %	23.31 %

Tabla 6: Porcentaje de edificios predichos por cada tipo de daño

4.2. Planteo de mejoras al modelo

Podemos concluir el análisis con lo siguiente:

- Como vimos recién, estamos prediciendo un 3 % a favor de daños leves. Por lo tanto podríamos ajustarlo con un modelo más propenso a predecir grave. Esto se podría lograr con un ensamble, con un modelo entrenado con mean damage a favor de daños graves.

- Es importante buscar en los casos donde el error es mayor (por ejemplo: daño medio cuando es grave), para ver si se encuentra alguna tendencia que haga cometer este error. Si se encuentra, por ejemplo una edad particular, ajustar los valores a la media (para que no sea tan determinante).

- Es posible que por la naturaleza del problema sea difícil obtener una mejora mayor. Ya que los features que tenemos no reflejen de manera directa lo necesario para obtenerlo. Por ejemplo sería útil saber si las viviendas recibían mantenimiento o no. También el azar es un factor importante.

Como último comentario, creemos que es importante dejar un marco de mejora, ya que como todo trabajo en ciencia de datos, es muy difícil que a la primera iteración hayamos tomado las mejores decisiones. Por lo que debemos trabajar con lo realizado hasta que el resultado sea satisfactorio.