FACULTAD DE INGENIERIA
Universidad de Buenos Aires

75.06/95.58 Organización de Datos

Primer Cuatrimestre 2021

Trabajo Práctico 2

Grupo:

En busca del dato perdido

Apellido/s	Nombre/s	Padrón	E-mail
Inneo Veiga	Sebastian Bento	100998	sinneo@fi.uba.ar
Massone	Mario Bernardo	102141	mbmassone@fi.uba.ar

Resumen

El objetivo del presente trabajo práctico es elaborar un modelo de machine learning capaz de predecir el daño, en una escala de 1 al 3, que sufrieron las edificaciones Nepal frente al terremoto Gorkha en 2015.

El dataset que se utiliza tanto para el entrenamiento como para el testeo son datos que fueron recopilados a través de encuestas realizadas por Kathmandu Living Labs y la Oficina Central de Estadísticas, que trabaja bajo la Secretaría de la Comisión Nacional de Planificación de Nepal.

Las predicciones realizadas para el set de test fueron subidas a la página DrivenData, más específicamente a la competencia "Richter's Predictor: Modeling Earthquake Damage".

Durante la realización del trabajo se probaron distintos algoritmos estudiados en la materia con distintos features que se extrajeron del set de datos. También se pensaron qué nuevos features se podrían extraer del set, estos fueron utilizados para llevar a cabo la clasificación y, al igual que con los algoritmos, se conservaron aquellos que exhiben resultados positivos, dejando de lado al resto.

El modelo que ha mostrado mejores resultados al momento de la presentación de este trabajo es RandomForest.

El F1 score micro obtenido con el mismo en la competencia de DrivenData es de 0.7429.

Introducción

El 25 de abril la Tierra tembló durante 40 segundos bajo el Himalaya. Un terremoto de 7,8 de magnitud sacudió Gorkha, un distrito de Nepal no lejos de la capital, Katmandú. En ese minuto escaso, mientras los ciudadanos buscaban refugio en la calle, edificios históricos de cientos de año de antigüedad se derrumbaban y, en cambio, casas desvencijadas aguantaban en pie.

En el trabajo práctico anterior se analizó las distintas características del dataset de Richter's Predictor: Modeling Earthquake Damage de DataDriven, el cual brindó aspectos de ubicación y de construcción de edificios que sufrieron daños causados por el terremoto de Gorkha de 2015 en Nepal.
Estos fueron los datos posteriores a un desastre más grandes jamás recopilados, que contiene información valiosa sobre los impactos del terremoto, las condiciones de los hogares y las estadísticas socioeconómicas y demográficas.

Para la predicción de daños ante un nuevo fenómeno, en esta oportunidad se utilizaron los archivos *train_values.csv*, *train_labels.csv*, *test_values.csv* y *submission_format.csv* para entrenar el modelo.

En el siguiente informe se detallarán los pasos seguidos y algoritmos utilizados para resolver el problema de predicción de Richter's Predictor: Modeling Earthquake Damage.

Link al repositorio: https://github.com/SBen-IV/TP-OrgaDeDatos

Link al video:
https://drive.google.com/file/d/1rvjBs7fijnf7NzsIWxY5fmgz6tI6CBbs/view?usp=sharing

Organización

Para realizar el trabajo práctico de manera efectiva y tomando el tiempo como un factor importante, decidimos que inicialmente lo mejor era que cada integrante del grupo hiciera sus propias pruebas sobre el set de datos. De esta forma podríamos probar distintos algoritmos y combinación de columnas que quizás a otro integrante no se le ocurriría. Al mismo tiempo que hacíamos pruebas, nos informábamos de los resultados obtenidos y así teníamos un mejor panorama de si apuntábamos para el camino correcto o no.
Se hizo uso de herramientas como Git, Github para mantener un control de versiones sobre los notebooks hechos y Google Colab para correr algunos de los algoritmos.

Feature Engineering

A continuación se detallarán las columnas creadas a partir de las columnas originales.

Limpieza de datos

Como ya habíamos analizado en el trabajo práctico anterior, no era necesario aplicar limpieza de datos en las filas por lo que ninguna fue descartada.

building_id

Dado que esta columna representa un valor único por cada building, se la descartó.

geo_level_1_id, geo_level_2_id y geo_level_3_id

Se dejaron las 3 columnas originales y, basados en el feature importance de los primeros algoritmos probados, se crearon algunas nuevas en base a estas:

geo_level_1_id_sum, geo_level_2_id_sum y geo_level_3_id_sum

Estas columnas representan la cantidad de buildings con el mismo geo level id, respectivamente.

geo_level_sum

Representa el resultado de la suma de los valores de las 3 columnas.

geo_level_multiply

Representa el resultado de la multiplicación de los valores de las 3 columnas.

geo_level_1_per_mud_mortar_stone

Basándonos en cómo se comportaban las nuevas columnas en los distintos algoritmos probados y en el análisis hecho en el TP anterior probamos crear una columna que represente al mismo tiempo si una edificación estaba hecha con mud mortar stone y de ser así qué geo level 1 id tenía.

geo_level_1_sum_per_mud_mortar_stone

Dado el feature importance obtenido por geo level 1 sum se decidió aplicar la misma misma idea que con la columna anteriormente mencionada.

count_floors_pre_eq

Esta columna que representa la cantidad de pisos que tenía un building antes del terremoto se dejó como estaba y se creó una en base a la misma:

count_floors_pre_eq_sum

Esta columna representa la cantidad de edificaciones que tenían la misma cantidad de pisos antes del terremoto.

age

La columna que contenía el dato de la antigüedad de cada building fue dejada como estaba y se creó otra en base a la misma:

age_sum

Representa la cantidad de edificaciones que tenían la misma antigüedad.

area_percentage y height_percentage

Se dejaron ambas columnas como estaban originalmente y se crearon 2 nuevas basadas en las mismas:

base_percentage

Representa el cociente entre area y height percentage.

area_per_height

Representa el producto entre area y height percentage.

land_surface_condition, foundation_type, roof_type, ground_floor_type, other_floor_type, position, plan_configuration y legal_ownership_status

Para el caso de las columnas categóricas todas fueron transformadas con el mismo encoding:

One hot encoding

Para cada una de las columnas y cada uno de sus valores asignarle una nueva columna binaria que indica si cumple o no con dicha condición.

Binary encoding

En este se le asigna un valor binario a cada uno de los valores de las distintas columnas, creando nuevas columnas binarias. La diferencia principal con one hot encoding es que de esta forma se crean menos columnas.

Berni encoding

Se reemplaza los caracteres categóricos por valores enteros numéricos comenzando de cero.

has_superstructure

En cuanto a las columnas binarias que representan el material con que estaban hechos los buildings, no se las modificó.

has_secondary_use

Por último, las columnas has secondary use tampoco fueron modificadas y en algunos modelos fueron descartadas.

Algoritmos de Machine Learning

Luego de analizar los datos y sus características se decidió poner a prueba modelos basados en árboles como *Random Forest*, *XGBoost*, *LightGBM*, *Catboost* y *Ensamble* con los anteriormente nombrados. Para la búsqueda de hiper parámetros se utilizó tanto grid y random search.

Hiper parámetros

Grid Search

Consiste en buscar un conjunto de hiper parámetros lo más óptimo posible para un determinado modelo, entrenando este para las distintas combinaciones posibles en un cierto rango de valores predefinidos. Su principal desventaja es que prueba todas las combinaciones, lo cual puede generar un aumento considerable de tiempo de ejecución.

Random Search

Este método busca los hiper parámetros que optimizan un modelo de manera aleatoria entre un rango de valores para cada hiper parámetro.
Su principal ventaja es que, como se definen la cantidad iteraciones limitadas de antemano, esto permite tener una idea del tiempo de ejecución que es *n* veces el tiempo que tarda el modelo en entrenarse.
Debido a esto, es común pasarle amplios rangos de posibles valores para cada hiper parámetro, pues el tiempo que tarda es constante, siempre y cuando, el tiempo de entrenamiento del modelo no depende de uno o más hiper parámetros, o en el peor de los casos está acotado. Esto permite encontrar valores para los hiper parámetros que, difícilmente, se probarían en un Grid Search.
Por otro lado, su principal desventaja es que al no probar absolutamente todas las combinaciones posibles de hiper parámetros como en Grid Search, puede que se llegue a un sub-óptimo entre el conjunto de valores posibles.

A pesar de utilizar estos métodos tan eficaces en general las pruebas manuales dieron mejores resultados.

Modelos

Random Forest

Random Forest es un ensamble de árboles de decisión que usa bagging, esto quiere decir que no todos los árboles ven la totalidad de los datos. Esto genera que cada árbol se entrene con un set de datos distinto, lo que al momento de combinar los resultados, hace que los errores se compensen.

XGBoost

XGBoost es una implementación especial de Gradient Boosting con la diferencia de que presenta mejoras en la velocidad de ejecución y en la performance de las predicciones.

LightGBM

LightGBM es un algoritmo de refuerzo y/o potenciación de gradientes (gradient boosting) basado en modelos de árboles de decisión. Este puede ser utilizado para la categorización, clasificación y muchas otras tareas de aprendizaje automático, en las que es necesario maximizar o minimizar una función objetivo mediante la técnica de gradient boosting, que consiste en combinar clasificadores sencillos, como por ejemplo árboles de decisión de profundidad limitada.
Se puede destacar como principales ventajas el aumento de velocidad de entrenamiento y mayor eficiencia, menor uso de memoria, mayor precisión, soporte de aprendizaje paralelo y soporte para GPUs, y capacidad para manejar datos a gran escala.

CatBoost

CatBoost es un algoritmo que utiliza también Gradient Boosting sobre árboles de decisiones.
Como principal diferencia a los otros algoritmos, este permite atributos no numéricos.

Ensamble

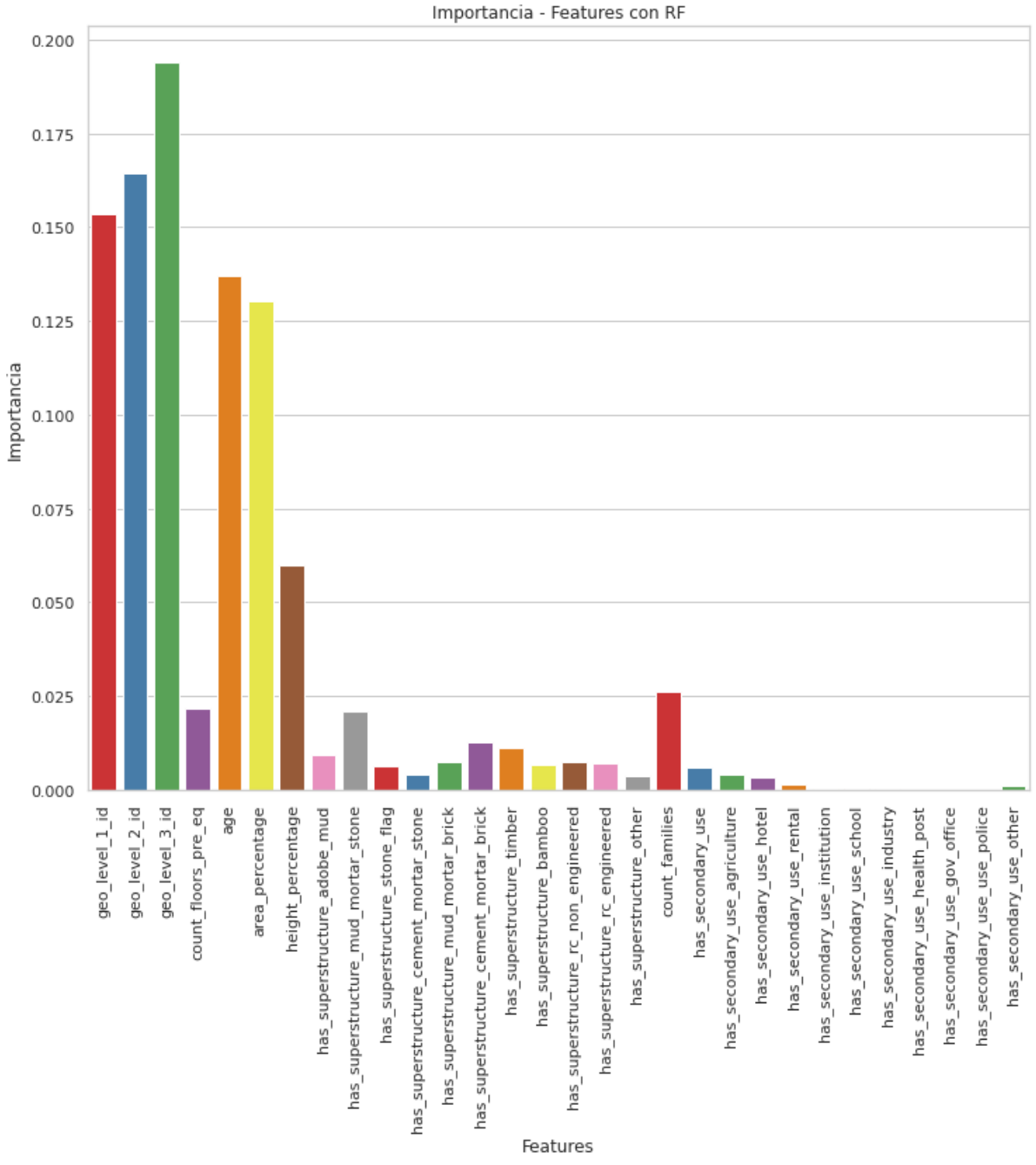
En general, es muy raro que un solo algoritmo de Machine Learning logre mejores resultados que un ensamble por eso es que se realiza la combinación de varios de estos.

Hipótesis

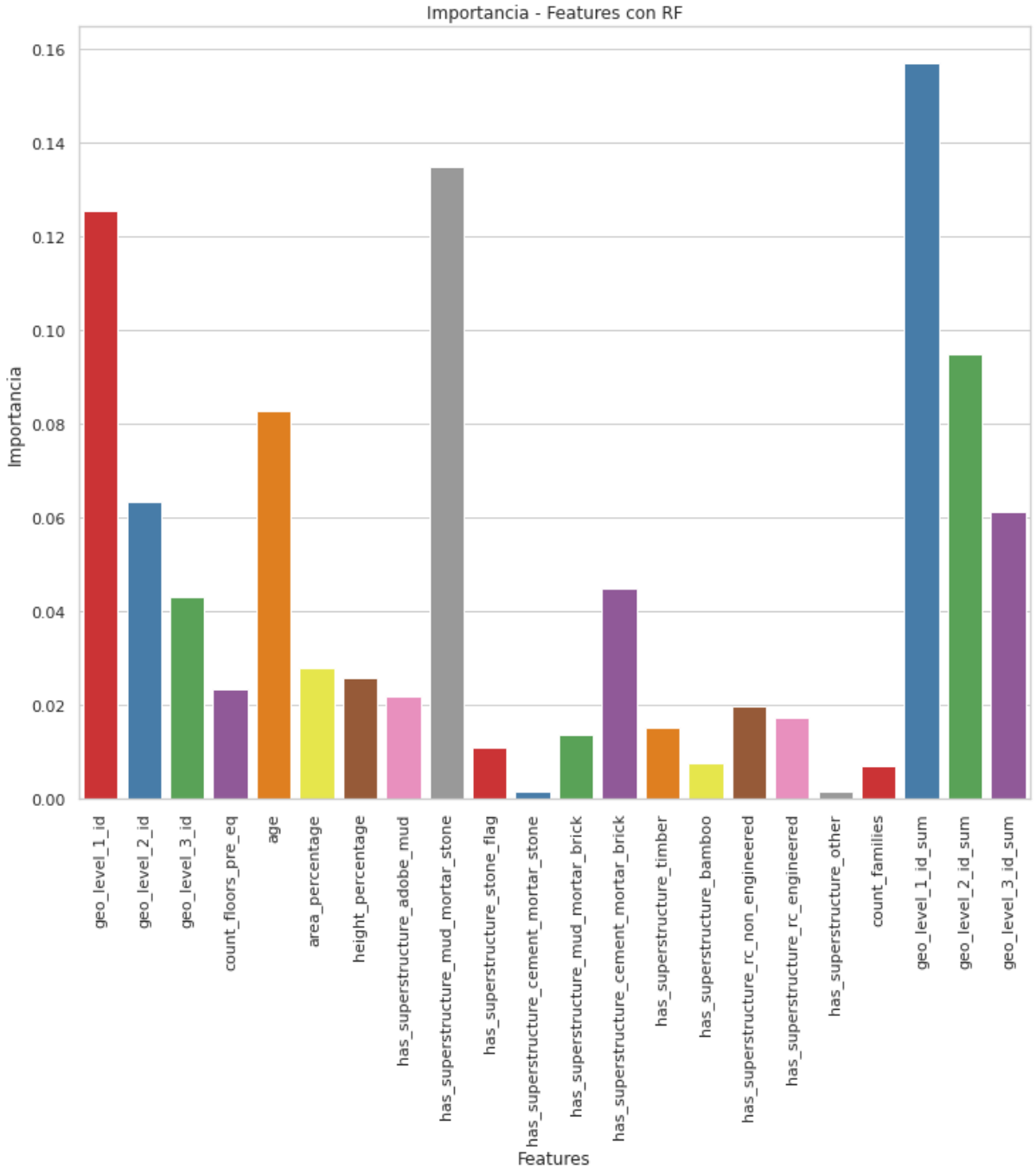
A continuación detallaremos las preguntas que nos planteamos y sus resultados.

¿Por dónde empezamos?

El primer algoritmo que probamos fue Random Forest dado que generalmente da un buen indicio de en qué columnas hay que enfocarse al ver el feature importance. Sin haber realizado un feature engineering (salvo por sacar las columnas categóricas para poder usar Random Forest) resultaron más importantes las columnas *geo_level_1_id*, *geo_level_2_id* y *geo_level_3_id*.



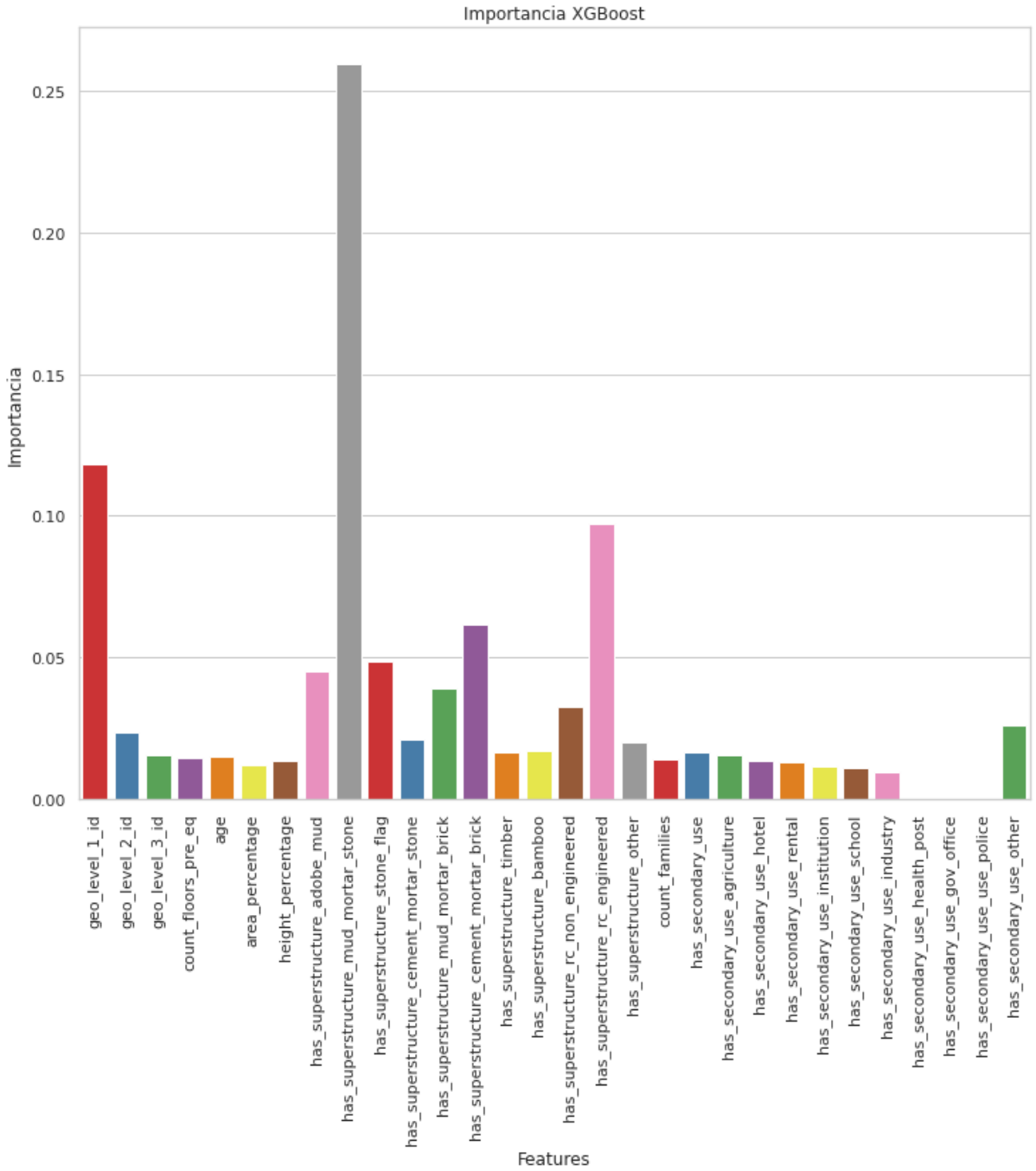
A su vez, probamos cómo se ajustaba con el set de test y, como inicio, obtuvimos un score de 0.6519. Dado estos resultados un primer planteo fue crear columnas que relacionaran a estas 3. De las columnas creadas, sólo *geo_level_1_id_sum*, *geo_level_2_id_sum* y *geo_level_3_id_sum* tomaron un mayor feature importance que las originales.



Sin embargo, los resultados para el set de test eran peores y Random Forest parecía no ser lo suficientemente expresivo para predecir bien los datos con estas columnas. A pesar de esto insistimos en que el camino era con esas 3 columnas.

Probando otros modelos

Algunas pruebas preliminares sobre algoritmos como XGBoost en el set de entrenamiento daban resultados un poco mejores a los obtenidos con Random Forest pero al momento de probarlo con los datos del set de test overfitteaba.
Al momento de ver a qué columnas le daba más importancia nos llamó la atención el gráfico:

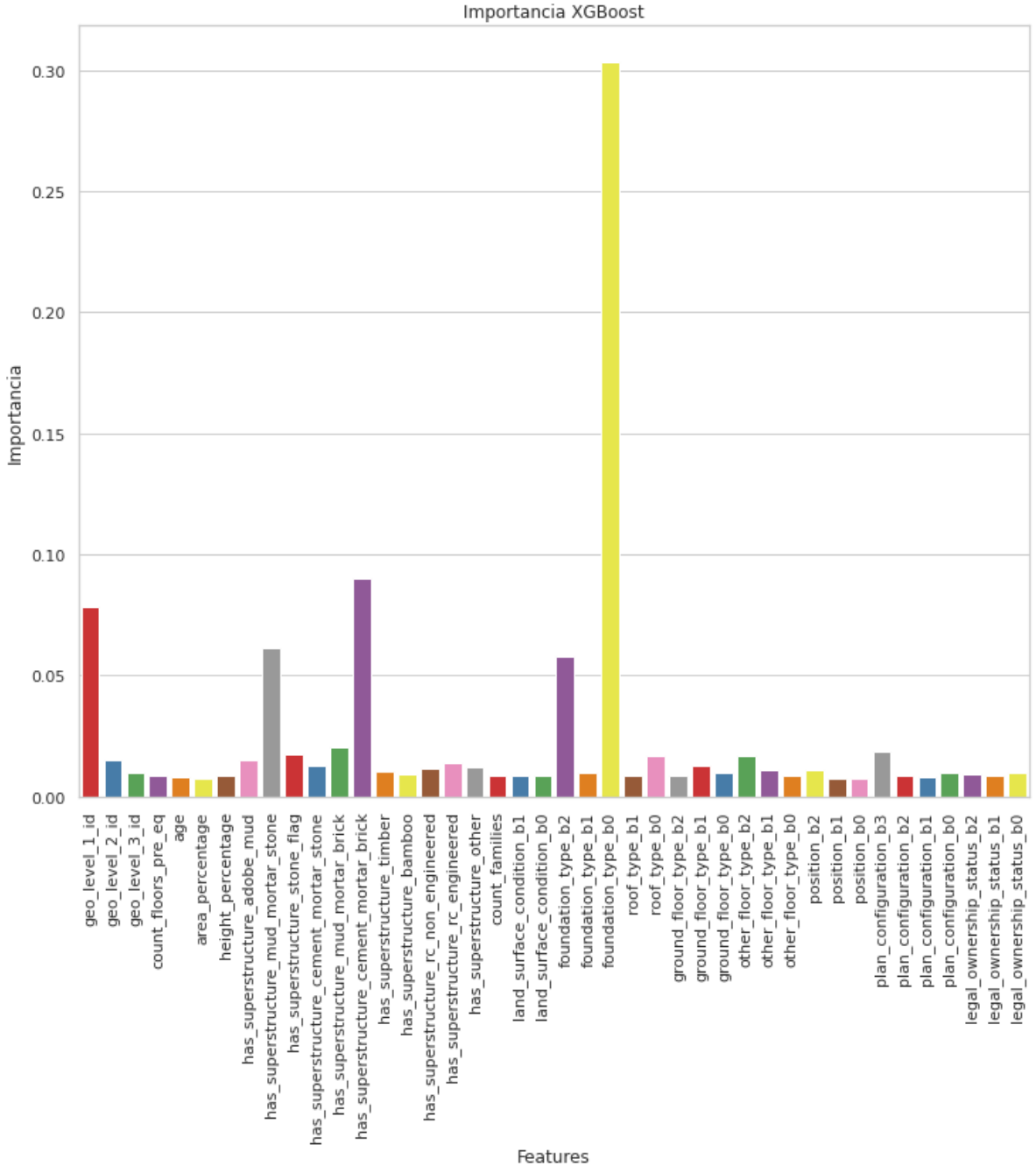


Esto nos recordó un poco al trabajo práctico anterior en el cual habíamos concluido que la columna *has_superstructure_mud_mortar_stone* podría tener relación con el *damage_grade*. En esta instancia, tanto LightGBM como Catboot daban resultados

parecidos o peores en el set de entrenamiento por lo que no se indagó más a fondo con esos algoritmos.

Columnas categóricas, ¿necesarias?

Mientras por un lado se insistía por crear columnas en base a *geo_level_x_id*, por el otro, se probó convirtiendo las columnas categóricas con One Hot Encoding y Binary Encoding. En una de las primeras pruebas hechas se notó que con Binary Encoding los modelos le daban más importancia que a los *geo_level_x_id* como antes:



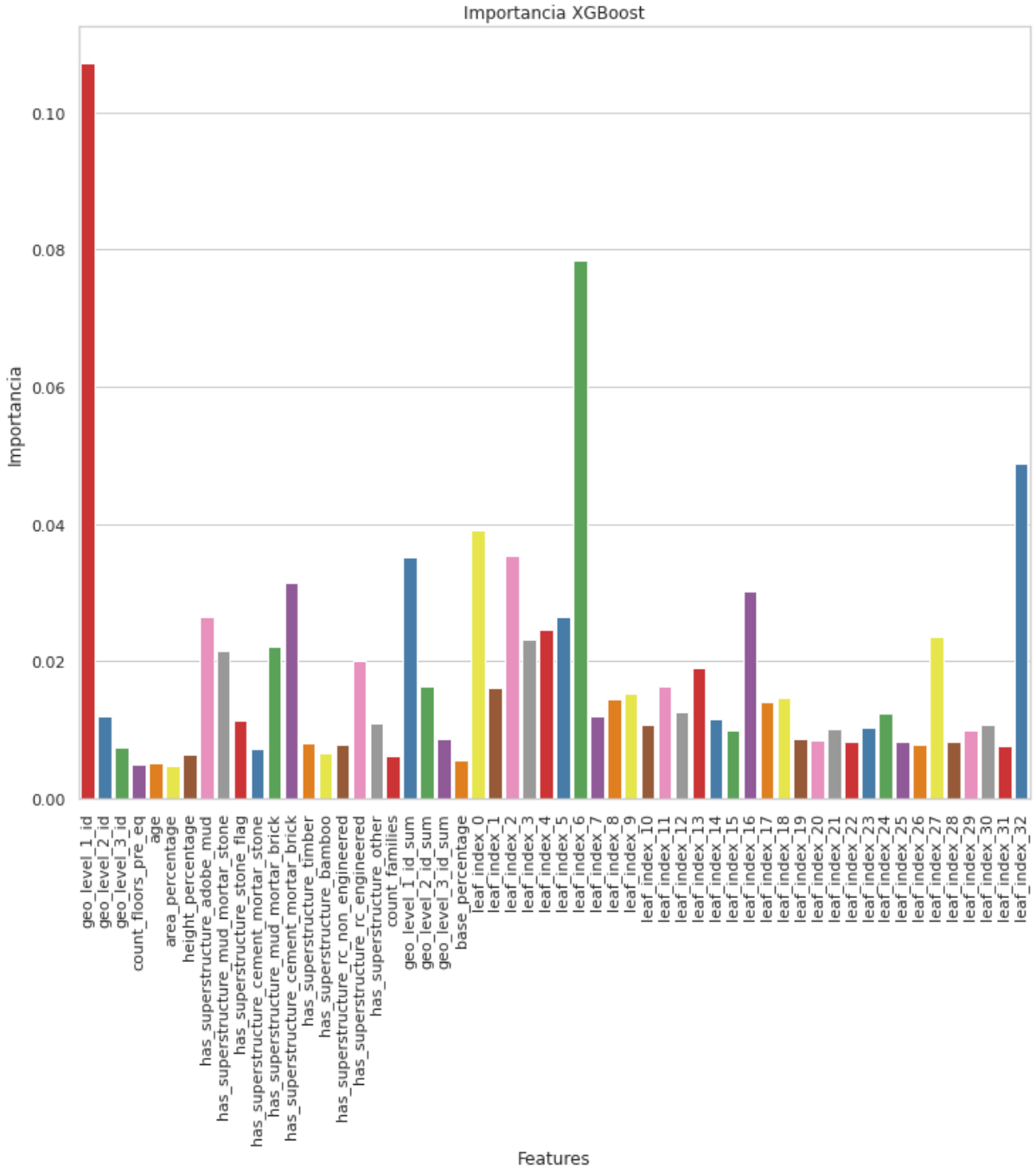
¿Más features, mejores resultados?

Creyendo que el problema seguía siendo la falta de columnas se crearon nuevas usando los mismos modelos con los mismos parámetros y otros probados con Grid o Random Search. Entre las nuevas columnas se crearon las básicas siguiendo la idea de *geo_level_x_id_sum* que fueron *age_sum* y *count_floors_pre_eq_sum*.

En busca de los datos ¿perdidos?

Ante la falta de mejoras en los modelos, se hicieron algunas pruebas con columnas creadas a partir del índice de hoja que tenían con un modelo dado, en este caso se hicieron con 2 modelos que aparentaban dar buenos resultados en el set de entrenamiento: Random Forest y XGBoost.

Primero se realizó la prueba con el modelo de XGBoost que hasta el momento *parecía* tener buen rendimiento.

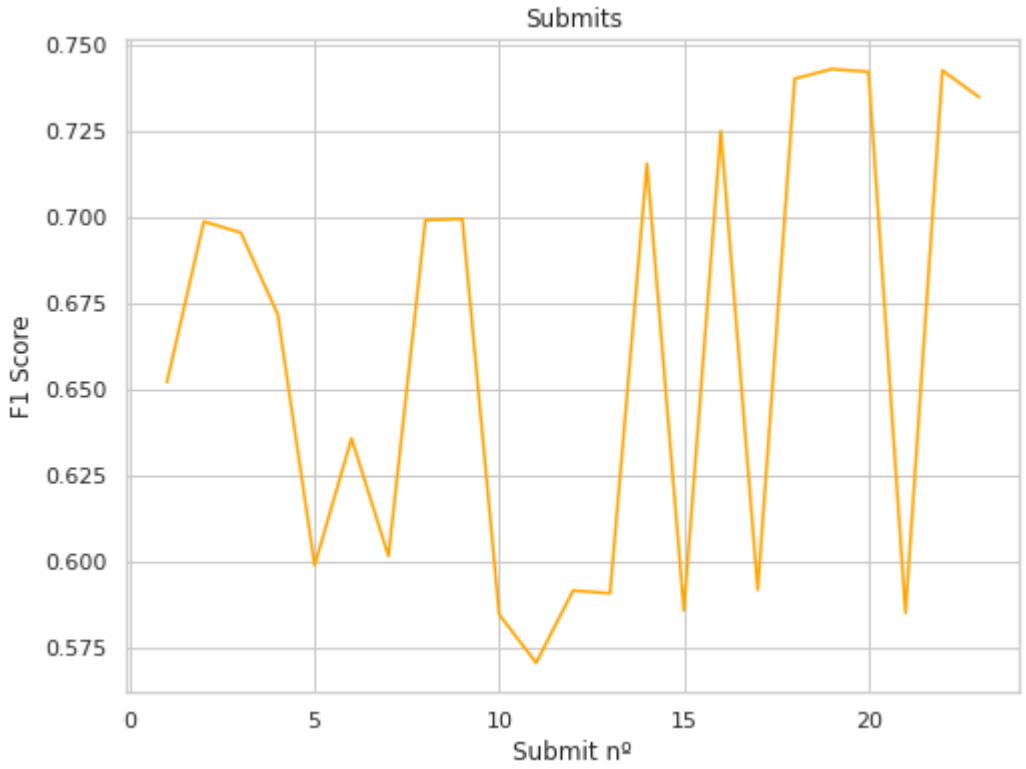


Como se puede observar, el feature importance es bastante más bajo que el de los probados anteriormente. A pesar de esto, se predijo sobre el set de test, ya que en el set de entrenamiento se ajustaba mejor que otros modelos en ese momento pero el resultado fue de 10 puntos por debajo.

Random Forest da mejores resultados

Hasta este momento el algoritmo que mejor había ajustado era Random Forest con algunas pocas columnas nuevas por lo que se insistió en seguir probando hasta conseguir un resultado decente. El gráfico muestra cómo fue la evolución del score

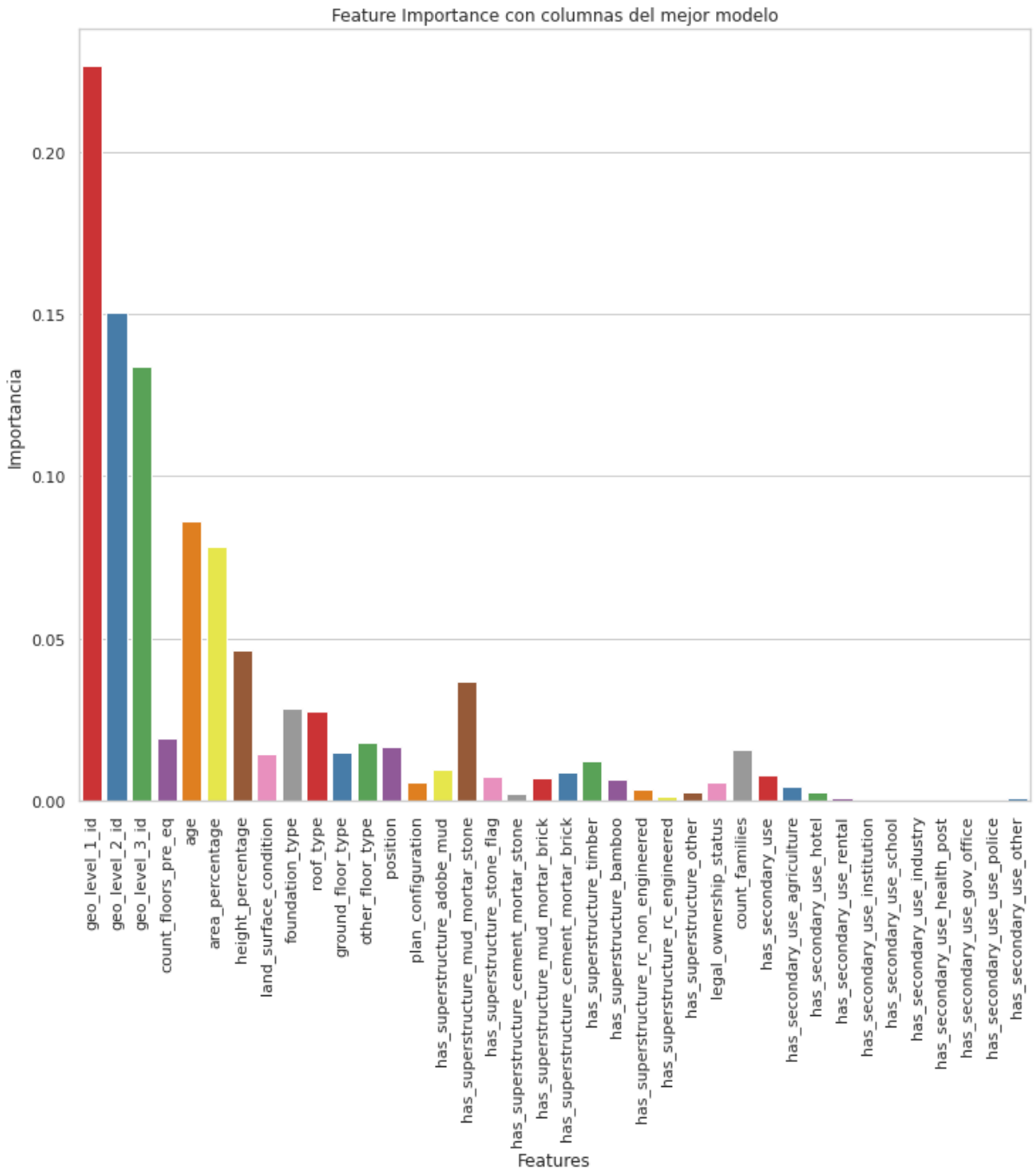
siendo los picos, en general, los conseguidos con Random Forest mientras que los valores más bajos con otros algoritmos:



Luego de una exhaustiva búsqueda de parámetros con Grid Search y manualmente encontramos una combinación de los mismos que nos dio el mejor resultado hasta el momento, 0.7429.

Columnas categóricas, ¡claves!

Llegando al final de la competencia nos dimos cuenta que no era necesario, para este set de datos, crear nuevas columnas sobre las columnas numéricas sino sobre las categóricas. Dada la falta de información sobre el significado de los valores en las mismas no habían demasiados encodings posibles o que se nos ocurrieran. Dicho esto, y haciendo unas últimas pruebas para mejorar unos centésimos en el score, probamos utilizar el mismo modelo que nos dio mejor score hasta el momento (0.7429) pero sin las columnas categóricas (dado que el feature importance daba la impresión de que no eran necesarias) y agregando algunas de las columnas que antes probamos y no dieron resultado.



Quitar las columnas categóricas fue un error porque si bien el modelo ajustaba bien en el set de entrenamiento (0.7459), no rendía nada bien en el set de test (0.5849). Por esto, probamos nuevamente pero con otro encoding a las columnas, en este caso, binary encoding. Resultó que en una prueba con el set de entrenamiento, al dar valores tan cercanos a los que teníamos con Random Forest, afinando los parámetros, llegamos a un score que nos convenció por falta de tiempo (0.7333) y, al hacer un submit, nos encontramos con la sorpresa de que nos dió 0.7347. Esto

nos llevó a una conclusión: la clave para mejorar el score estaba en cómo encodear las columnas categóricas.

Mejor algoritmo y resultado

Al momento de escribir este informe el algoritmo que mejor resultado nos dio fue **Random Forest** con un resultado de **0.7429**.

Los parámetros personalizados utilizados para Random Forest fueron:

- ➔ *n_estimators* = 300
- ➔ *min_samples_leaf* = 2
- ➔ *max_features* = 23
- ➔ *max_leaf_nodes* = 10000

Conclusión

El trabajo fue interesante y muy bueno para familiarizarse con los diferentes algoritmos de clasificación, crear nuevos features, entrenar distintos algoritmos y resultados entre ellos.

Podemos asegurar que el aprendizaje automático es mucho más que correr distintos algoritmos de predicción, hay que tener mucha práctica y conocer las distintas tácticas para poder mejorar la predicción.

Una observación que nos resultó interesante y llamativa es que con el agregado de features que a nuestro parecer eran de importancia para mejorar la predicción, el score no mejoró notablemente.
También la búsqueda y pruebas manuales de hiper parámetros dio mejores resultados en comparación con los métodos recomendados y mencionados en el grid y random search.
Notamos que la ausencia de los datos categóricos, a pesar de tener un feature importance bajo, empeoraban los resultados, por lo que la clave estaba en cómo codificarlos.

Consideramos que aún quedarían distintas opciones que nos hubiese gustado explorar y realizar que no hemos podido, ya sea por falta de tiempo estudiar en profundidad o porque lo hayamos intentado erróneamente.