FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2

[7506/9558] Organización de Datos
Curso 1
Primer cuatrimestre de 2021

Alumnos	Padrón
Reinaudo, Dante	102848
Lerer, Joaquín	105493
Elvis, Claros Castro	99879

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Organización de Datos, el cual consiste en realizar un modelo de *machine learning* utilizando el set de datos estudiado a lo largo del primer trabajo práctico. Para lograr esto, se hará uso de todos los conocimientos adquiridos en la materia. En particular, el modelo se realiza sobre un conjunto de datos con información acerca del impacto del terremoto Gorkha, ocurrido en Nepal durante el año 2015. Cabe aclarar que el modelo propuesto se lleva a cabo en el marco de una competencia de *DrivenData*, esta es una plataforma que hostea competencias de ciencia de datos y machine learning.

Para la realización del algoritmo de aprendizaje automática, la plataforma nos brinda tres archivos en formato csv[1] , estos son: Train Values, Train Label y Test Values. El primer archivo cuenta con el set de datos sobre el cual entrenaremos nuestro algoritmo, principalmente cuenta con información sobre la estructura de los edificios, las condiciones de las diferentes viviendas y su propiedad legal. Cada fila del conjunto de datos representa un edificio específico en la región que fue golpeada por el terremoto de Gorkha. El segundo archivo, cuenta con el grado de daño que sufrió cada uno de los edificios descriptos en el set de entrenamiento, esta variable será el target de nuestro algoritmo, es decir que nuestro objetivo será predecir este valor. Por último, el tercer archivo es el set de test con el que probaremos nuestro algoritmo, cuenta con los mismos features que el set de entrenamiento, pero no poseemos con su labels, es decir, desconocemos el grado de daño que sufrieron estos edificios. Haciendo uso de nuestro modelo de machine learning, trataremos de estimar el daño sufrido por dichas edificaciones, almacenaremos nuestros resultados en un archivo csv y lo subiremos a la plataforma, la cual le asignará un puntaje a nuestra entrega calculando el error de la solución con la métrica F1 Score, particularmente con la variación no binaria micro averaged F1 score. Cabe aclarar que con dicha métrica, una predicción es mejor cuanto el valor calculado por esta sea más cercano a uno.

$$F_{micro} = \frac{2 \cdot P_{micro} \cdot R_{micro}}{P_{micro} + R_{micro}}$$

$$P_{micro} = \frac{\sum_{k=1}^{3} TP_k}{\sum_{k=1}^{3} (TP_k + FP_k)}, \quad R_{micro} = \frac{\sum_{k=1}^{3} TP_k}{\sum_{k=1}^{3} (TP_k + FN_k)}$$

1 Comma-Separated Values (CSV): formato de archivo en el que los valores se encuentran separados por comas.

2. Desarrollo

En esta sección describiremos todo el proceso realizado en la construcción del modelo de machine learning, desde el preprocesamiento de los datos, el análisis de los features más significativos, la comparación de distintos modelos predictivos que encuadran el problema en cuestión, hasta la búsqueda de hiperparámetros óptimos mediante una búsqueda automatizada.

Antes de comenzar con el análisis del trabajo realizado, consideramos importante hacer una aclaración sobre el enfoque utilizado en la construcción de nuestro modelo. El objetivo que se buscó alcanzar con este proyecto, fue el de aumentar lo máximo posible nuestro puntaje en la competencia, por lo tanto el algoritmo aquí desarrollado sólo es útil en el marco de la competencia para la cual estamos suscritos y no sería de mucha utilidad para aplicaciones en la industria.

2.1. Preprocesamiento de los datos

A la hora de desarrollar un algoritmo de machine learning, debemos cerciorarnos que el set de datos con el que vayamos a trabajar se encuentre completo, es decir, vamos a estudiar si contamos columnas sin información, con datos nulos o con datos del tipo NaNs (not a number). En caso de haber, debemos definir que haremos con esta información faltante. Existen diversos tipos de soluciones para completar la data ausente, e incluso hay algoritmos que las completan automáticamente utilizando alguno de estos métodos, y es muy importante que definamos estas cuestiones antes de comenzar a modelar nuestro algoritmo.

En el análisis exploratorio realizado a lo largo del primer trabajo práctico, ya habíamos llevado a cabo un preprocesamiento de los datos. En primer lugar, buscamos si había valores nulos dentro de alguna de las columnas tanto del set de entrenamiento como del set de prueba, por suerte en ninguno de los DataFrame analizados se encontraron columnas sin información, ni tampoco columnas con valores nulos. Podríamos decir que los conjuntos de datos con los que trabajaremos están bastante limpios.

Por último, con el objetivo de administrar el espacio utilizado y reducir el tiempo de procesamiento de nuestros algoritmos, cada vez que se leyeron los set de datos a utilizar, se almacenaron con el menor tipo de dato suficiente para su representación. Por ejemplo, en vez de almacenar una columna booleana con un int64, tipo de dato por default que se le asigna al utilizar el read_csv de pandas, se le especifico a esa función que interprete el dato como un bool. De esta manera se pudo mejorar la performance de nuestro trabajo.

2.2. Análisis del target : Grado de Daño (Damage Grade)

Dentro de nuestro set de datos, contamos con la información de 260601 edificios afectados por el sismo Gorkha, de 7.8 grados de magnitud en la escala Richter. Una de las variables proporcionadas, es el grado de daño que sufrió cada edificio luego de la tragedia. Esta será nuestra variable de interés, y el target que buscamos predecir con nuestro algoritmo de aprendizaje automático, se estudiarán las características de cada construcción para tratar de encontrar alguna relación entre las propiedades de los edificios y su daño sufrido a causa del terremoto. Vamos a resaltar que se trata de una variable ordinal, es decir, que los valores posibles para el grado de daño guardan una cierta relación de orden entre sí, y no es simplemente una variable nominal. Deberemos tener esto en cuenta a la hora de desarrollar nuestro algoritmo.

En primer lugar, se calculó la proporción del grado de daño del total de los 26061 edificios afectados, obteniendo la siguiente tabla:

Grado de Daño	Total de edificios afectados	Porcentaje del total (%)
Low Damage	25124	9.6
Medium Damage	148259	56.9
High Damage	87218	33.5

Tabla 1: Porcentaje del grado de daño de los edificios

Para obtener una mejor visualización de la información, se realizó el siguiente gráfico de tortas, también conocido como pie chart .

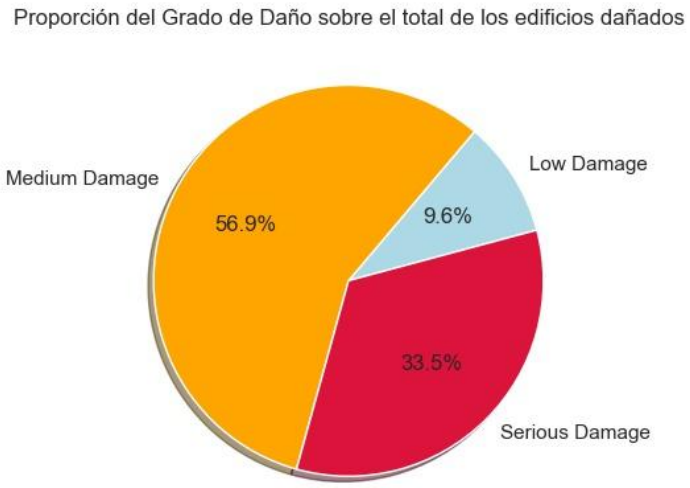

Figura 1: Proporción del Grado de Daño sobre el total de los edificios

Como puede verse en la figura anterior, la mayoría de los edificios sufrió un daño moderado, otro gran porcentaje sufrió un daño alto, mientras que tan solo una pequeña minoría sufrió un daño bajo. En las secciones posteriores vamos a investigar si existen características comunes entre los edificios que sufrieron un mismo grado de daño.

2.4. Correlación de los datos

La correlación es una medida estadística que expresa hasta qué punto dos variables están relacionadas linealmente. Es una herramienta común para describir relaciones simples sin hacer afirmaciones sobre causa y efecto.

Como una primera aproximación vamos a proceder a analizar la correlación que existe entre el grado de daño de los edificios con respecto al resto de las columnas de nuestro set de datos, para tener una primera noción de cuáles pudieran ser los features más significativos. Para realizar este análisis, se hizo uso de las herramientas brindadas por las librerías pandas y matplotlib, cabe aclarar que la librería de pandas no soporta el cálculo de correlaciones entre variables categóricas, por lo que previamente realizamos una codificación One-Hot sobre los features categóricos. Tomando los valores absolutos para la correlación se obtuvo la figura 2.

Al observar la figura 2, queda al descubierto cuales son las columnas que guardan más relación lineal con el grado de daño. Una primera hipótesis que podríamos plantearnos en nuestro análisis, es que los features con más correlación serán los que mayor influencia tendrán en los resultados a la hora de realizar predicciones. A medida que avancemos en nuestra investigación, buscaremos estimar la validez de esta hipótesis.

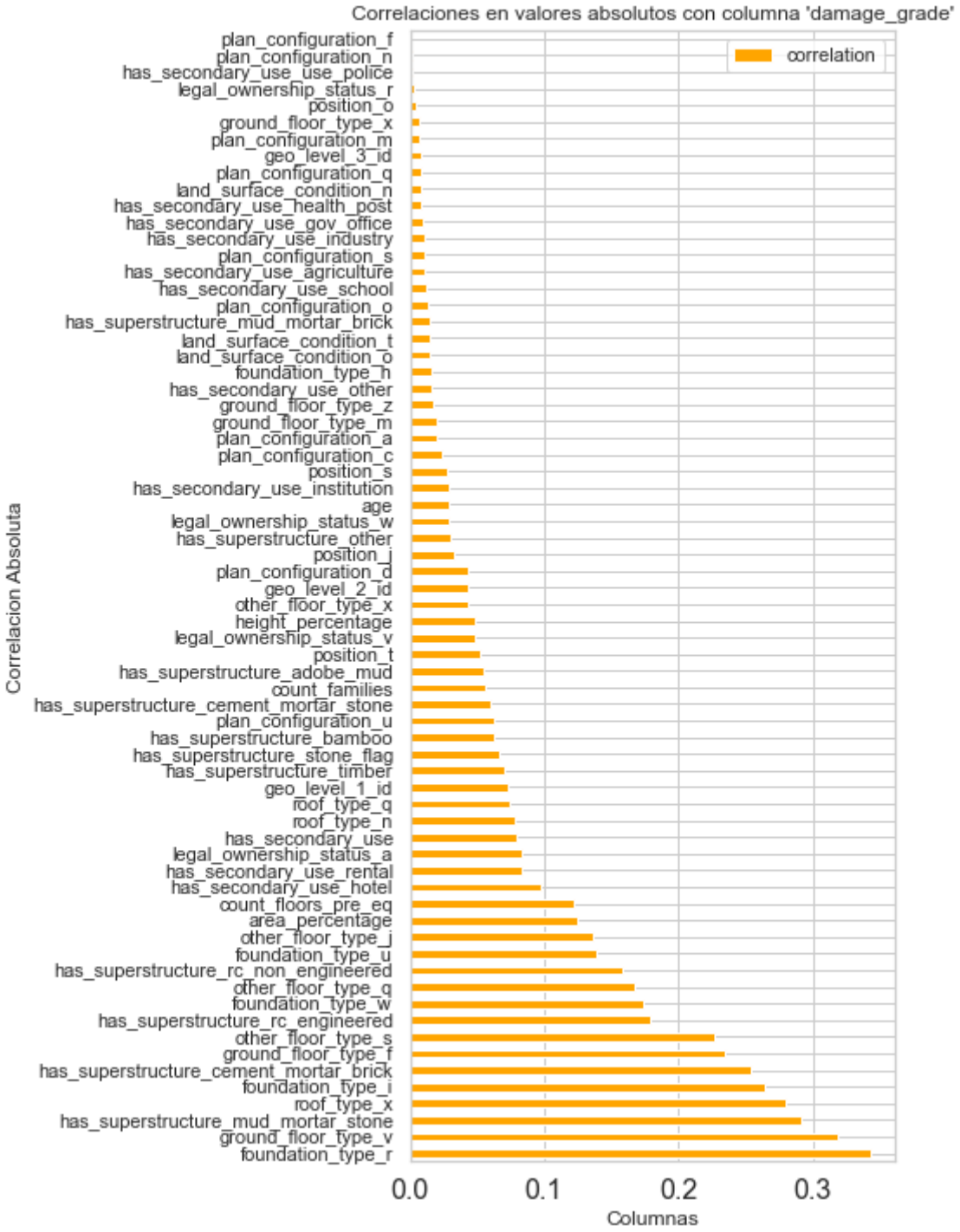

Figura 2: Correlación en valores absolutos de las columnas con Damage Grade

Por otro lado, calculamos cuales eran las 15 columnas que guardan mayor relación con el grado de daño, y se obtuvo la siguiente tabla:

colum_name	correlation
Cimientos Tipo r	0.34
Planta baja tipo v	0.32
has_superstructure_mud_mortar_stone	0.29
Techos Tipo x	0.28
Cimientos Tipo i	0.26
has_superstructure_cement_mortar_brick	0.25
Planta baja tipo f	0.23
Contruccion de otros pisos tipo s	0.23
has_superstructure_rc_engineered	0.18
Cimientos Tipo w	0.17
Contruccion de otros pisos tipo q	0.17
has_superstructure_rc_non_engineered	0.16
Cimientos Tipo u	0.14
Contruccion de otros pisos tipo j	0.14
area_percentage	0.13
count_floors_pre_eq	0.12

Figura 3: Los 15 features que guardan mayor relación con el grado de daño

Para una mejor visualización, procederemos a realizar un gráfico de barras horizontal de estos quince features con mayor correlación:

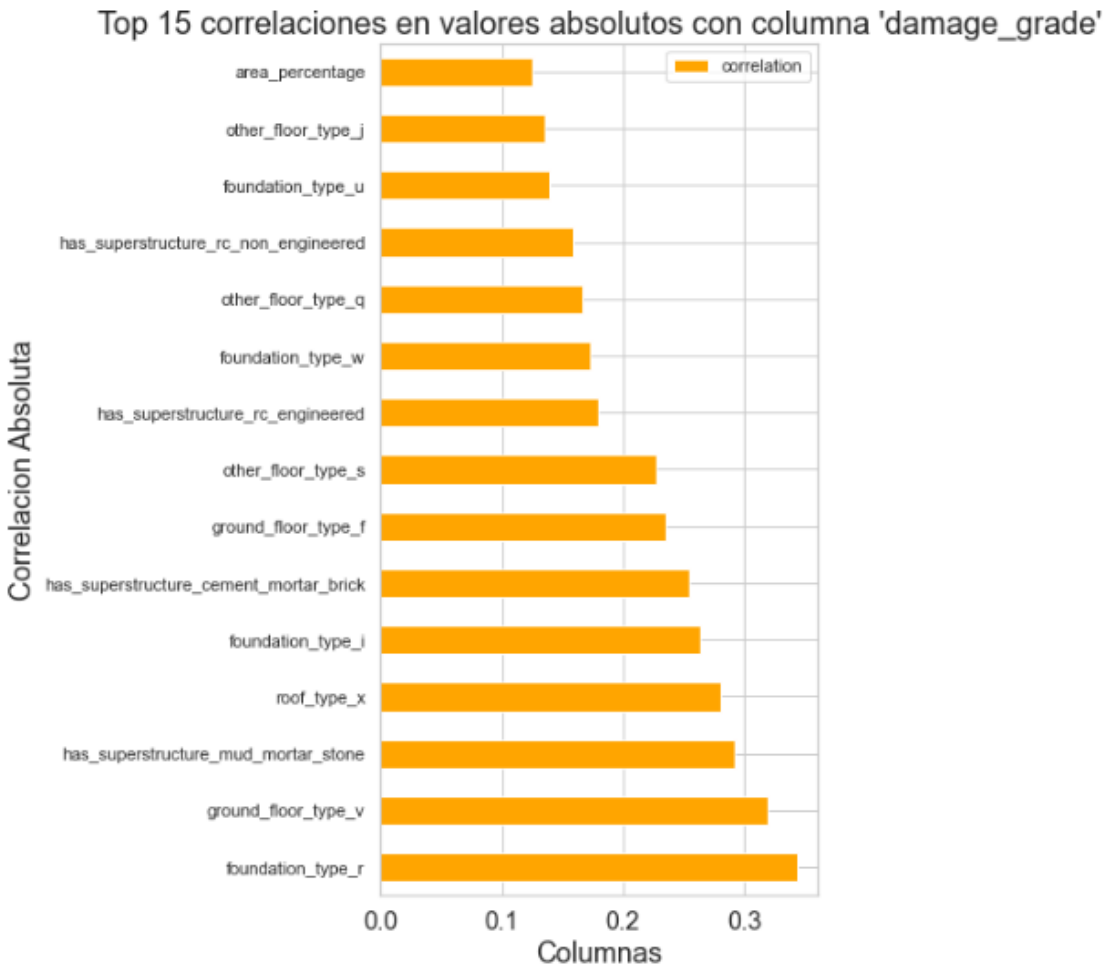

Figura 4: Gráfico de barras de los 15 features que guardan mayor relación con el grado de daño.

Tomando como referencia las correlaciones principales se puede observar una clara relación entre la selección de materiales, el grado de ingeniería, el tipo de cimientos y el tipo de techos seleccionados en relación con la gravedad del estado del edificio afectado por un terremoto. Teniendo en cuenta estos gráficos, otra hipótesis que podríamos plantearnos sería que dado que estas son los features con más correlación, deberían ser más importantes que el resto y deberíamos enfocar nuestro análisis ponderando principalmente dichas columnas por sobre las demás, mientras que los features que presentan baja correlación podrían ser descartados ya que serían de baja relevancia. A continuación para

nuestro modelo. A continuación, procederemos a desarrollar nuestro primer modelo de machine learning, estudiando la validez de las hipótesis planteadas hasta el momento.

2.5. Modelo utilizando Random Forest

Random Forest es uno de los algoritmos más populares y más utilizados para resolver problemas de clasificación, entregando buenos resultados para la mayoría de los set de datos. Un Random Forest es un conjunto de árboles de decisión combinados con bagging. Al usar bagging, distintos árboles ven distintas porciones de los datos y ningún árbol ve todos los datos de entrenamiento. Esto ocasiona que cada árbol se entrene con distintas muestras de datos para un mismo problema. De esta forma, al combinar sus resultados, se logra reducir el overfitting y tenemos una predicción que generaliza mejor.

Optamos por desarrollar nuestro primer modelo utilizando esta técnica, ya que es muy práctica y simple de interpretar. En primer lugar, para realizar nuestro primer submit en la competencia, seguimos paso a paso las instrucciones descriptas en el blog proporcionado por la plataforma, *Richter's Predictor - Benchmark*, el cual podrán encontrar en el apartado de referencias. En dicho tutorial, se describe un poco más en detalle el problema sobre el cual estamos trabajando, y se desarrolla un modelo basado en Random Forest seleccionando ciertos features numéricos específicos. Luego de entrenar este modelo, realiza las predicciones correspondiente sobre el set de test y realiza un submit obteniendo un score de 0,5815. En informática, un Benchmark o prueba de rendimiento es una técnica utilizada para medir el rendimiento de uno o varios programas. En este caso, sirve para que el competidor tenga un puntaje base de referencia con el que pueda comparar su trabajo.

Siguiendo la misma idea del modelo desarrollado en el Benchmark, se procedió a realizar un primer modelo basado en Random Forest, pero esta vez seleccionando los 15 features con mayor correlación con nuestro target, los cuales estudiamos en la sección anterior. Para llevar a cabo este modelo, utilizamos el Random Forest que nos provee la librería de Sklearn, una de las bibliotecas de aprendizaje automático más utilizadas en la industria. Dado que este modelo no puede trabajar con variables categóricas, se procedió a codificarlas con One-Hot-Encoding, y se seleccionó para el entrenamiento solo las columnas booleanas de las variables categóricas que figuran en el ranking de correlaciones. Por ejemplo, en lugar de utilizar todas las columnas producidas por el One-Hot encoding del feature “roof_type”, se utilizó solo la columna binaria “roof_type_x” la cual figura segunda en el ranking de correlaciones, dejando de lado el resto de los diferentes tipos de techos. Luego de entrenar nuestro primer modelo utilizando estos features, realizamos la predicción correspondiente sobre el set de test y efectuamos nuestro primer submit en la competencia. Para nuestra sorpresa, el score obtenido fue incluso más bajo que el del Benchmark, dándonos un puntaje para nuestra primera predicción de tan solo 0,5795. Este resultado, ya empieza a poner en evidencia que nuestras primeras hipótesis planteadas estaban equivocadas, ya que un modelo que utilizó features que nosotros considerábamos que tenían menor relevancia para las predicciones, pudo estimar mejor el target para el set de test que el nuestro.

Luego de este hallazgo, se probó crear un segundo modelo basado en Random Forest, pero esta vez seleccionando absolutamente todos los features, encodeando con One-Hot las variables categóricas, con el objetivo de estudiar cómo afectaba esto en la predicción final. Luego de entrenar nuestro nuevo modelo, el puntaje obtenido fue 0,5765, incluso peor que el del modelo anterior. A diferencia de lo que pasó con nuestro primer modelo, este resultado no fue para nada sorprendente, dado que era esperable que este nuevo modelo realizara una peor predicción, ya que al seleccionar todos los features, estamos complejizando demasiado el modelo y favorecemos el Overfitting.

Por último, realizamos un tercer modelo basado en Random Forest, utilizando los mismos features que el primer modelo, pero esta vez incluyendo además todas las columnas booleanas producidas por la codificación de los features categóricos presentes en el ranking. De esta manera, tras entrenar, predecir y subir nuestros resultados a la plataforma una vez más, obtuvimos un score de 0,5802, el cual es más alto que el obtenido por los otros dos modelos planteados hasta el momento. Este resultado, desestima nuestra hipótesis de que las columnas con baja correlación no son relevantes para nuestro modelo, ya que al incluirlas, se pudo obtener un mejor valor en la predicción del grado de daño. Si bien este fue el mejor de los tres modelos propuestos en esta sección, sigue estimando el *damage grade* de peor manera que el algoritmo del Benchmark, por esta razón vamos a proceder a utilizar otras técnicas diferentes para el entrenamiento de nuestro algoritmo de machine learning, particularmente nos centraremos en las que utilizan algún tipo de Boosting .

Antes de finalizar esta sección, vamos a destacar la versatilidad que presenta la técnica de Random Forest, siendo muy útil para resolver problemas tanto de clasificación como de regresión. Además, considero que es una gran herramienta para comenzar a desarrollar un algoritmo de aprendizaje automático, obteniendo como resultado una muy buena primera aproximación de nuestro modelo.

2.6. Modelo utilizando XGBoost

El algoritmo XGBoost (Extreme Gradient Boosting) es una técnica de aprendizaje supervisado, también basada en árboles de decisión diseñada para ser altamente eficiente y flexible. Hoy en día, es considerada por muchos profesionales como el estado del arte dentro de la evolución de los algoritmos que utilizan árboles de decisión, ya que puede resolver muchos de los problemas de la ciencia de datos de manera rápida y precisa. Para la realización de los modelos descriptos en esta sección, se utilizó la librería de XGBoost.

Se desarrollaron dos modelos basados en XGBoost, el primero interpretando el trabajo como un problema de regresión, y el segundo tomando a éste como un problema de clasificación. Cabe aclarar, que como vimos en sección Análisis de target, la variable a predecir es ordinal, por lo que el problema a resolver seria un hibrido entre uno de clasificacion y otro de regresión, a menudo se suele referir a estos problemas como de Clasificación Ordinal o de Regresión Ordinal. Por esta razón, ninguno de los dos modelos planteados en esta sección serán del todo correctos, y no podemos esperar que su predicción sea completamente acertada.

El primer modelo que se desarrolló fue utilizando un XGBRegressor, proporcionado por la librería de XGBoost, en donde se seleccionaron algunos hiperparametros al azar. Para llevar a cabo esto, definimos a nuestro target como una variable numérica en lugar de categórica. Por otro lado, dado que este modelo no soporta trabajar con variables categóricas, se procedió a codificarlas con One Hot Encoding, tanto para el dataframe de entrenamiento como para el de test. Luego se dividió tanto el dataset de entrenamiento como el dataset de labels en dos partes de proporciones diferentes, dejando por un lado el set y label de entrenamiento y por otro lado el set y label de validación. Luego de entrenar nuestro modelo con ambos sets, utilizando todos los features disponibles, se predijo el grado de daño para los edificios del set de test. Esto nos dio como resultado un vector con números flotantes con un rango de valores entre 1 y 3. Los valores para dicha predicción fueron redondeados, almacenados en un csv y cargados en la plataforma. El score para esta estimación fue de 0.5552, valor mucho más bajo que los obtenidos hasta el momento. Sin embargo, esto no quiere decir que XGBoost ande mal y deje mucho que desear, sino que simplemente el modelo desarrollado fue muy pobre, no se hizo ningún tipo de proceso de selección de los features más importantes, los hiperparametros fueron elegidos bajo un concepto totalmente arbitrario y el hecho de redondear la estimación final, de por sí conlleva una incerteza muy grande en la estimación.

Para desarrollar el segundo modelo esta vez utilizamos un XGBClassifier, interpretando el problema como uno de clasificación multiclase, donde las distintas clases a predecir son 1, 2 o 3, según el grado de daño del edificio. En cuanto a los hiperparametros, en este caso definimos un learning rate de 0,1, con una función objetivo softmax para una clasificación multiclase de tres tipos de clases, y se dejaron al resto de los hiperparametros posibles con su valor por defecto, ya que como vimos para el modelo anterior, no es una buena práctica para el modelo asignarle valores completamente aleatorio y sin criterio. Igual que en el caso anterior y por las mismas razones, se le aplicó una codificación One Hot a los features categóricos de los dataframes de entrenamiento y test. A su vez, se volvieron a separar los datos en set de entrenamiento y set de validación, ya que esta práctica ayuda a reducir el overfitting. Bajo estas condiciones, se entrenó el modelo con el set de entrenamiento y validación, se estimó el grado de daño para los edificios del test y se cargaron los resultados en la plataforma. Con este modelo se obtuvo un score de 0,7033, el más alto de todos los obtenidos hasta el momento. Si bien la clasificación multiclase ignora la ordinalidad de nuestra variable objetivo, teniendo en cuenta este resultado, podemos afirmar que aproximar el problema de esta manera, es una mejor estimación que la utilizada en el modelo anterior. A su vez, con este resultado queda evidenciada la potencia que presentan los algoritmos de Boosting y el porqué de su popularidad en la industria.

En la próxima sección se procederá a utilizar otro modelo que aplica Boosting de árboles de decisión con el objetivo de comparar el rendimiento entre ambos. Consideramos que los modelos aquí explicados son suficientes y sirven a modo de descripción, por lo que no serán profundizados en detalle. Si quisiéramos mejorar estos modelos, podríamos optar por realizar un proceso de *feature engineering* para obtener features interesantes junto con una búsqueda automatizada de los hiperparametros más óptimos.

2.7. Modelo utilizando CatBoost

CatBoost es una librería de *machine learning* que, al igual que XGBoost, aplica un Boosting de gradiente sobre árboles de decisión. Está desarrollado por investigadores e ingenieros de Yandex, y se utiliza para búsquedas, sistemas de recomendación, asistente personal, autos sin conductor, predicción del clima y muchas otras tareas. Se destaca por su gran performance sin ajuste de hiperparametros, utilizando solo sus valores por defecto. Además, a diferencia de los modelos vistos anteriormente, CatBoost brinda un soporte para variables categóricas, sin necesidad de tener que codificarlas previamente. Por las razones antes descriptas, dada su gran eficiencia, su funcionamiento interno novedoso que reduce el overfitting, su predicción rápida y mejorada, decidimos construir nuestro modelo centrándonos principalmente en esta herramienta.

El primer modelo que construimos utilizando Catboost, fue un clasificador multiclase, dado que hasta el momento fue la forma de encarar el problema que mejores resultados nos brindó. Como hicimos anteriormente, dividimos el dataset de entrenamiento juntos con sus labels, en uno de entrenamiento y otro de validación, gracias a la herramienta train_test_split que nos brinda la librería sklearn. La segmentación se realizó en un 90% para el set de entrenamiento y el otro 10% para el de validación. Dado que CatBoost, permite trabajar con variables categóricas, no fue necesario codificarlas con One Hot, es más, la misma librería nos aconseja no hacer esto ya que su propio algoritmo interno realiza codificaciones mucho más eficientes , aplicando cálculos estadísticos y numéricos. Si bien no es necesario codificar las variables categóricas, si es necesario al momento de entrenar al algoritmo especificarle cuales son estas variables, esto se logra mediante un parámetro denominado cat_features, en donde se le debe pasar un arreglo con el índice de las columnas con variables categóricas. La primera vez que entrene mi modelo, no era consciente de la funcionalidad de este parámetro, entonces le pase un vector con todos los índices de mis columnas, por lo que el algoritmo interpretó a todas las variables como categóricas. Dejando los parámetros por defecto y utilizando todos los features, se entrenó este primer modelo, el cual luego de 1000 iteraciones ya estaba listo para estimar nuestro target. Se realizó la predicción correspondiente sobre el set de test y se cargaron los resultados en la plataforma. Para mi sorpresa, el score obtenido fue 0,7501, considerablemente más alto que el obtenido con XGBoost y en ambos casos utilizando los parámetros por defecto. Con este resultado, ya se empieza a ver la potencia de utilizar Catboost, y la eficiencia en su análisis de variables categóricas. A continuación, se adjunta una tabla con la importancia de los features en el desarrollo para este primer modelo:

	features_names	features_importance
0	geo_level_1_id	7.49
1	geo_level_2_id	26.35
2	geo_level_3_id	16.25
3	count_floors_pre_eq	2.63
4	age	7.60
5	area_percentage	3.57
6	height_percentage	3.02
7	land_surface_condition	1.27
8	foundation_type	5.98
9	roof_type	3.45
10	ground_floor_type	5.05
11	other_floor_type	3.94
12	position	2.17
13	plan_configuration	0.87
14	has_superstructure_adobe_mud	0.25
15	has_superstructure_mud_mortar_stone	1.80
16	has_superstructure_stone_flag	0.27
17	has_superstructure_cement_mortar_stone	0.40
18	has_superstructure_mud_mortar_brick	0.73
19	has_superstructure_cement_mortar_brick	1.22
20	has_superstructure_timber	0.61
21	has_superstructure_bamboo	0.21
22	has_superstructure_rc_non_engineered	0.05
23	has_superstructure_rc_engineered	0.20
24	has_superstructure_other	0.20
25	legal_ownership_status	1.02
26	count_families	2.34
27	has_secondary_use	0.81
28	has_secondary_use_agriculture	0.10
29	has_secondary_use_hotel	0.02
30	has_secondary_use_rental	0.04
31	has_secondary_use_institution	0.02
32	has_secondary_use_school	0.00
33	has_secondary_use_industry	0.02
34	has_secondary_use_health_post	0.02
35	has_secondary_use_gov_office	0.00
36	has_secondary_use_use_police	0.00
37	has_secondary_use_other	0.04

Figura 5: Features importantes del primer modelo desarrollado con CatBoost.

De la figura 5, podemos observar la importancia que le da el modelo a los features de geo localización en comparación con el resto, teniendo un peso mucho mayor en la prediccion que la mayoría de los otros features. Si nos ponemos a analizar este resultado, es más que lógico debido a que la distancia del edificio al epicentro del terremoto es un claro indicador del grado de daño sufrido por este. Dado que el enfoque con el que se desarrolló este trabajo fue el de maximizar el puntaje en la competencia, no existe ningún problema con utilizar estos features, pero si por el contrario nuestro objetivo fuera predecir el grado de daño de las construcciones a causa de desastres naturales en general, nuestro modelo no sería muy útil ya que no podríamos usar la localización del terremoto como indicador. En este último caso, la ubicación del terremoto estaría generando un leak al target. Por otro lado, esta imagen termina de desestimar completamente ambas de las hipótesis planteadas en las secciones anteriores, ya que el modelo le está dando mucha importancia a features que siguiendo el criterio de la hipótesis no deberían ser relevantes, como la ubicación y la edad, mientras que a otros features que guardaban mucha correlación lineal con la variable grado de daño, como por ejemplo has_superstructed_rc_engineered, casi no son tenidos en cuenta.

Siendo fiel a la filosofía de que el equipo que gana no se toca, los siguientes dos modelos que se entrenaron fueron idénticos al primero, aumentando únicamente la cantidad de iteraciones en el entrenamiento, primero a 5000 y luego a 15000. Automáticamente, al aumentar la cantidad de iteraciones, el mismo algoritmo disminuye el learning rate o tasa de aprendizaje. Las predicciones para estos nuevos modelos fueron 0,7515 y 0,7520 respectivamente, estos puntajes son más que aceptables y nos permitieron entrar en el top 50 global de la competencia. Además, ambos modelos obtuvieron un feature importances bastante similar a la mostrada en la figura número 5.

Luego de obtener dichos puntajes, comencé a probar distintos métodos para ver si podía lograr mejorar aún más el score. Uno de estos métodos fue probar entrenar el modelo quitando las columnas con feature_importances muy bajos, pero la predicción bajo bastante de calidad. De esta prueba podemos afirmar que por poco que aporte una columna, debemos considerarla y no es una buena práctica descartarla per se.

Por otro lado, también probamos que sucedía si entrenamos el modelo pasando por el parámetro cat_features solo solo las verdaderas variables categóricas y no todas como lo habíamos hecho antes. Para nuestra sorpresa, el score obtenido fue 0,7254, más bajo que el obtenido por el primer modelo de esta sección. Esto nos llamó notablemente la atención, ya que todo pareciera indicar que el algoritmo de procesamiento de variables categóricas que utiliza CatBoost es tan bueno que incluso puede trabajar con variables numéricas.

También probé utilizando distintas funcionalidades que nos provee el módulo de CatBoost para el preprocesamiento de los datos con el que luego entrenaría los algoritmos. Desarrollé varios modelos utilizando las clases FeatureData y Pool de la librería para procesar los datos, e incluso probé distintas proporciones para dividir el set de entrenamiento y validación pero no logré hacer cambios sustanciales en la predicción de los modelos.

Luego de realizar varios intentos para mejorar la predicción del modelo y no poder lograrlo, llegue a la conclusión que esto podría deberse a que siempre estaba utilizando los mismos features. Por lo tanto, comencé una etapa de *feature engineering* sobre el set de datos en busca de features interesantes que ayudarán a mejorar las estimaciones realizadas. Debido a la importancia que le adjudicaba el modelo a las columnas de ubicación geográfica, y dado que cada una hacía alusión a un nivel de la región, opte por generar una nueva columna llamada geo_level_absolute la cual es la sumatoria de las otras tres columnas. Siguiendo la misma estructura que el primer modelo, entrene un nuevo algoritmo de aprendizaje automático pero esta vez utilizando el set de datos con esta nueva columna. Al realizar el submit con esta nueva predicción se obtuvo un score de 0,7503, mejor que la obtenida con el primer modelo de CatBoost. Con este resultado podemos notar la importancia e incidencia que tiene el proceso de feature engineering en la predicción final de nuestro modelo, ya que tan solo agregando una columna pudimos aumentar un poco la calidad en la estimacion. A continuación se adjunta una tabla con la importancia de los features para este nuevo modelo.

	features_names	features_importance
0	geo_level_1_id	7.90
1	geo_level_2_id	20.56
2	geo_level_3_id	20.04
3	count_floors_pre_eq	2.07
4	age	7.64
5	area_percentage	3.14
6	height_percentage	2.75
7	land_surface_condition	1.15
8	foundation_type	6.06
9	roof_type	3.12
10	ground_floor_type	5.95
11	other_floor_type	4.04
12	position	2.19
13	plan_configuration	0.81
14	has_superstructure_adobe_mud	0.32
15	has_superstructure_mud_mortar_stone	1.66
16	has_superstructure_stone_flag	0.19
17	has_superstructure_cement_mortar_stone	0.26
18	has_superstructure_mud_mortar_brick	0.61
19	has_superstructure_cement_mortar_brick	1.03

	features_names	features_importance
20	has_superstructure_timber	0.53
21	has_superstructure_bamboo	0.22
22	has_superstructure_rc_non_engineered	0.06
23	has_superstructure_rc_engineered	0.12
24	has_superstructure_other	0.19
25	legal_ownership_status	1.01
26	count_families	2.52
27	has_secondary_use	0.89
28	has_secondary_use_agriculture	0.16
29	has_secondary_use_hotel	0.00
30	has_secondary_use_rental	0.02
31	has_secondary_use_institution	0.02
32	has_secondary_use_school	0.01
33	has_secondary_use_industry	0.03
34	has_secondary_use_health_post	0.02
35	has_secondary_use_gov_office	0.00
36	has_secondary_use_use_police	0.00
37	has_secondary_use_other	0.06
38	geo_level_absolute	2.63

Figura 6: Features importantes del modelo con *feature engineering* desarrollado con CatBoost.

Como se observa en la tabla, la nueva columna agregada tiene bastante importancia en el modelo, siendo más considerada por este que otras columnas como por ejemplo la cantidad de pisos del edificio.

Al igual que lo que hicimos anteriormente, vamos a utilizar el mismo algoritmo con el mismo set de datos pero esta vez aumentaremos la cantidad de iteraciones a 15000. Luego de varias horas de entrenamiento, nuestra predicción final obtuvo un score 0,7528, siendo este nuestro puntaje más alto en la competencia y el que nos permitió entrar en el top 15 global. A continuación adjuntamos una tabla con la importancia de los features de este último modelo.

	features_names	features_importance
0	geo_level_1_id	7.40
1	geo_level_2_id	18.83
2	geo_level_3_id	10.65
3	count_floors_pre_eq	3.25
4	age	8.02
5	area_percentage	5.66
6	height_percentage	5.19
7	land_surface_condition	2.41
8	foundation_type	3.68
9	roof_type	3.48
10	ground_floor_type	3.57
11	other_floor_type	3.35
12	position	2.90
13	plan_configuration	1.20
14	has_superstructure_adobe_mud	0.52
15	has_superstructure_mud_mortar_stone	1.21
16	has_superstructure_stone_flag	0.22
17	has_superstructure_cement_mortar_stone	0.24
18	has_superstructure_mud_mortar_brick	0.60
19	has_superstructure_cement_mortar_brick	0.62

	features_names	features_importance
20	has_superstructure_timber	0.64
21	has_superstructure_bamboo	0.29
22	has_superstructure_rc_non_engineered	0.10
23	has_superstructure_rc_engineered	0.13
24	has_superstructure_other	0.24
25	legal_ownership_status	1.55
26	count_families	2.66
27	has_secondary_use	0.70
28	has_secondary_use_agriculture	0.29
29	has_secondary_use_hotel	0.04
30	has_secondary_use_rental	0.04
31	has_secondary_use_institution	0.01
32	has_secondary_use_school	0.01
33	has_secondary_use_industry	0.03
34	has_secondary_use_health_post	0.01
35	has_secondary_use_gov_office	0.01
36	has_secondary_use_use_police	0.02
37	has_secondary_use_other	0.09
38	geo_level_absolute	10.17

Figura 6: Features importantes del último modelo desarrollado con CatBoost.

En esta última imagen se aprecia claramente la importancia e incidencia que tuvo en nuestro modelo agregar la columna geo_level_absolute, siendo el tercer feature con más relevancia en la estimación del grado de daño.

Por último, realizamos una búsqueda automatizada de hiperparametros, proceso crucial para el desarrollo de algoritmos de aprendizaje automático conocido como *tuning*. Para esto realizamos un grid search de hiperparametros, utilizando una función de la misma librería de Catboost. Algunos de los hiperparametros a optimizar fueron el learning rate y la profundidad de los árboles de decisión (*depth*), entre otros. Debido a que entrenar un modelo con 1000 iteraciones, tomaba un tiempo aproximado de dos horas, al realizar la búsqueda de hiperparametros se hizo sobre un Catboost Classifier de 10 iteraciones cada uno para que no tardará demasiado. Luego de buscar sobre la grilla mostrada en la siguiente figura, se obtuvieron que los mejores hiperparametros para este modelo fueron: learning_ rate = 0,15 , depth = 15 y l2_leaf_reg = 1 .

```
grid = {'learning_rate': [0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.15],
        'depth': [None,4, 6, 10,12,15],
        'l2_leaf_reg': [None,1, 3, 5, 7, 9,12,15,20]}
```

Figura 7: Grilla de valores para la búsqueda de hiperparametros.

Solo se tomaron estos tres hiperparametros dado que en la bibliografía no encontré el significado de los otros y no quise asignarles cualquier valor arbitrario ya que el proceso de búsqueda es muy costoso. Una vez obtenidos estos hiperparametros intenté entrenar un nuevo modelo utilizándolos, pero el tiempo de entrenamiento requerido era demasiado alto, de aproximadamente varios días, así que opté por abortar esta operación y dejar el modelo tal cual estaba. Si bien el proceso de *tuning* es crucial en el desarrollo de algoritmos de machine learning, los valores predeterminados de las funciones de CatBoost tienen una buena performance, por lo que considero que el trabajo realizado hasta el momento es más que satisfactorio.

3. Conclusiones

A modo de conclusión, en primer lugar se cree que con el presente informe se ha logrado describir claramente la metodología de desarrollo, fundamentos y resultados que guiaron la estructura de nuestro proyecto. Se pudo enunciar paso a paso las decisiones que se fueron tomando para el perfeccionamiento del algoritmo, la motivación que nos llevó a realizar dichas optimizaciones y el porqué de su determinación.

Por otro lado, si bien el modelo final desarrollado no es el más óptimo ni el mejor de todos, considero que su labor fue más que satisfactoria logrando un score en su predicción bastante alto, el cual colocó a nuestro equipo entre los primeros puestos del ranking de la UBA y dejándonos en el top 15 global de la competencia. Este resultado es más que provechoso dado que esta fue nuestra primera competencia de machine learning, en donde por primera vez pudimos aplicar los conceptos vistos en clase, además de ser la primera vez que estamos cursando la materia. En particular, todo el trabajo aquí descripto cobra más valor si tenemos en cuenta que tanto todos los modelos realizados como el extenso informe aquí escrito fue el resultado del trabajo de un único alumno, un trabajo el cual está pensado para que lo realicen cuatro personas. En relación a esto último, si bien estoy más que satisfecho con los resultados obtenidos, considero que el modelo podría haber sido mucho más provechoso si se hubiera trabajado en equipo, ya que esto es una herramienta fundamental en cualquier trabajo científico, en especial en los trabajos de aprendizaje automático.

Asimismo, si en un futuro quisiéramos optimizar nuestro modelo, en primer lugar pondría mucha atención y le daría gran relevancia a la etapa de *feature engineering.* Como comprobamos experimentalmente, este proceso es de lo más provechoso y en algunos casos puede incluso ser más crucial que la etapa de *tuning*. Además, por la enorme cantidad de tiempo que conlleva la búsqueda de hiperparametros, no hice tanto hincapié en este proceso, pero si quisiéramos mejorar nuestro modelo predictivo no podemos dejar de lado esta etapa. Por otra parte, considero que si bien el Clasificador Multiclase nos brindó grandes resultados en la competencia, no hay que olvidar que este método ignora la ordinalidad de nuestra variable *damage grade*. La ordinalidad de nuestro target nos está brindando información importante y no debe ser descartada, probablemente desarrollando otro modelo que tuviera en cuenta este factor se podrían alcanzar mejores resultados.

1. Código Fuente

El código fuente se encuentra disponible en:
https://github.com/DatosOrga2021/orga_datos_1c_2021

2. Video Explicativo

El video explicativo se encuentra disponible en:

https://youtu.be/oSQDqmLlTIk

3. Librerías Externas

3.1. Pandas

Pandas es una librería de Python especializada en el manejo y análisis de estructuras de datos.

3.2. Numpy

Numpy es una librería para análisis numérico muy conocida. Dado su practicalidad estuvo involucrada de fondo en muchas operaciones que realizamos, así como también forma parte de otras de las librerías usadas.

3.3. Matplotlib

Matplotlib es una librería para creación de visualizaciones estáticas y animadas.

3.4. SeaBorn

Seaborn es una librería para la visualización de datos estadísticos basada en Matplotlib.

3.5. Sklearn

Scikit-learn, o también llamada Sklearn, es una biblioteca para aprendizaje automático de software libre para el lenguaje de programación Python. Incluye varios algoritmos de clasificación, regresión y análisis de grupos entre los cuales destacaremos, Random Forest, Gradient boosting, K-means y DBSCAN.

3.6. Pickle

Pickle es un módulo muy práctico que nos permite almacenar fácilmente colecciones y objetos en ficheros binarios abstrayendo toda la parte de escritura y lectura binaria. Fue utilizado para almacenar los modelos de machine learning entrenados.

3.7. XGBoost

XGBoost es una biblioteca optimizada de aumento de gradiente diseñada para ser altamente eficiente , flexible y portátil . Implementa algoritmos de aprendizaje automático bajo el marco Gradient Boosting .

3.8. CatBoost

CatBoost es un algoritmo de aprendizaje automático que utiliza un aumento de gradiente en árboles de decisión. Está disponible como una biblioteca de código abierto. Debido a su gran performance con sus parámetros por defectos, fue la herramienta principal con la cual se llevó a cabo nuestro modelo

4. Referencias

[1] Federal Democratic Republic of Nepal

https://en.wikipedia.org/wiki/Nepal

[2] Hunter, J. (2007). Matplotlib: A 2D graphics environment. Matplotlib: Python plotting. Retrieved from

https://app.flourish.studio/visualisation/4347256/edit.

[3] NumPy. Numpy.org. (2019). Retrieved from

https://numpy.org/.

[4] The pandas development team. (2020). pandas - Python Data Analysis Library. Pandas.pydata.org. Retrieved from

https://pandas.pydata.org/.

[5] Competencia de *DrivenData*:

https://www.drivendata.org/competitions/57/nepal-earthquake/

[6] Terremoto de Nepal de abril del 2015
https://es.wikipedia.org/wiki/Terremoto_de_Nepal_de_abril_de_2015

[7] Richters Predictors - Benchmark

https://www.drivendata.co/blog/richters-predictor-benchmark/

[8] Informe del primer trabajo práctico de la materia:

Primer Trabajo Práctico: Análisis Exploratorio

[9] Enunciado del trabajo Práctico:

Enunciado del Trabajo Práctico 2

[10] Manual de librería CatBoost:

https://catboost.ai/

[11] Manual de librería XGBoost

https://xgboost.readthedocs.io/en/latest/