Universidad de Buenos Aires

Facultad de Ingeniería

1er Cuatrimestre de 2021

Organización de Datos

Tp2: Richter Predictor

Grupo Hakuna Madata

Github: Trabajo Práctico 2

Video Presentación

Integrantes:	*Padrón:*
Tizziana Mazza Reta <tmazzar@fi.uba.ar>	101715
Lucía Pardo <lpardo@fi.uba.ar>	99999
Mauricio Rodríguez Bertella <mrodriguezb@fi.uba.ar>	100624
Nicolás Sanchez <nsanchez@fi.uba.ar>	98960

Observaciones:

---

29 de julio de 2021

Índice

1. Introducción **2**
1.1. Pipeline 2

2. Feature engineering **4**
2.1. Encodings 4
2.2. Seleccion de features 4
2.3. Creacion de nuevos features 4

3. Modelos **6**
3.1. KNN 6
3.2. Arboles de decisión 7
3.2.1. Random Forest 7
3.2.2. XGBoost 7
3.2.3. CatBoost 8
3.2.4. LightGBM 8
3.3. Redes neuronales 9
3.3.1. Perceptrón multicapa 9

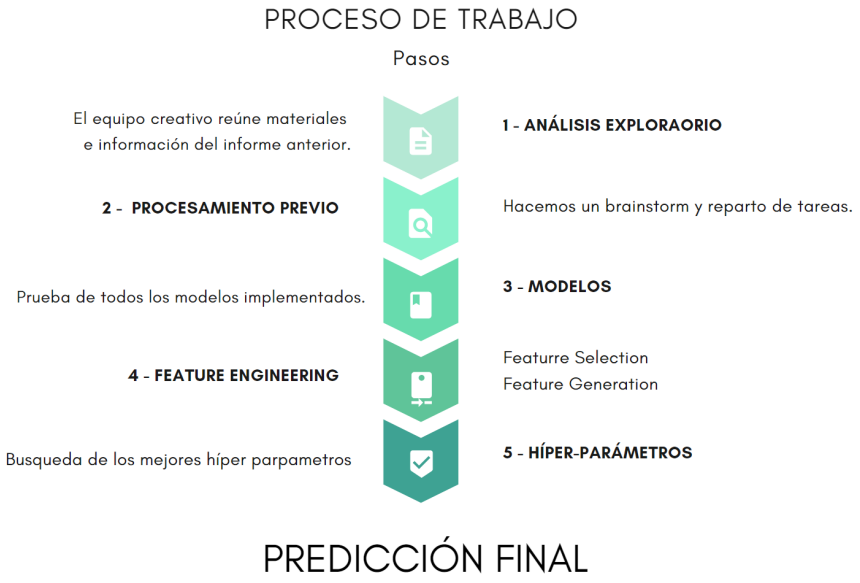
4. Submission Format **10**
4.1. F1 Score 10
4.2. Búsqueda del mejor Submit 10

5. Conclusión **11**

article graphicx float

1. Introducción

El objetivo del presente trabajo práctico es determinar con los features brindados por el dataset el tipo de daño que un edificio puede sufrir. Estas predicciones se deben lograr a partir de algoritmos de Machine Learning. En base a un cierto algoritmos se usan de distintas maneras los datos. Previo a utilizar estos algoritmos fue necesario realizar un filtrado y una investigación, para realizar feature engineering y así poder utilizar de mejor manera estos datos.

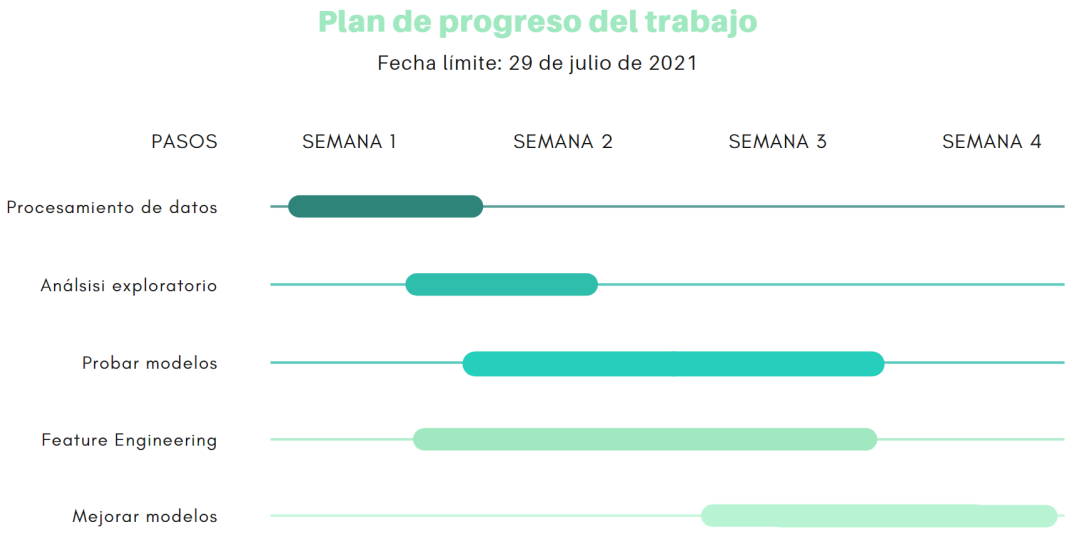


1.1. Pipeline

Para el trabajo fue propuesto un pipeline, una forma de trabajo para organizarse.

- Análisis explotatorio: Una recorrida por el dataset y etapa de reconocimiento y visualización, cubierta por el trabajo práctico 1. Además se realizó una etapa de pre procesamiento para intentar ahorrar memoria y convertir variables y facilitar la tarea posterior.
- Feature Engineering: Una etapa muy importante ya que es en lo que luego los modelos se van a basar y el resultado de estos se basan en lo que se genere en esta etapa. Se procede a realizar una selección de features en base a la importancia y luego a un feature generation para intentar obtener más información en base a la combinación de features.
- Modelar Se definen qué modelos utilizar y probar, decidiendo profundizar o cambiar de modelos en base a los resultados obtenidos.

- Parameter tuning: A partir de la elección y desarrollo de modelos es necesario profundizar en los resultados en base a los parámetros que se le pasan a estos modelos. Esto se puede realizar tanto manualmente como a través de algoritmos automatizados.
- Submission: A partir de definir una métrica y habiendo entrenado los modelos elegidos, seleccionamos las mejores predicciones y son subidas a la págia DrivenData



2. Feature engineering

Para esta etapa, se partio del set de datos original con 38 features, utilizando conceptos y conclusiones del TP1 y profundizando aun mas en la seleccion, creacion e interaccion de nuevos features, para lograr un modelo que generalice mejor y sea computacionalmente manejable.

2.1. Encodings

Para encodear nuestros features categoricos se utilizo en primer lugar One Hot Encoding, ya que por su facil implementacion fue el mas rapido para comenzar a trabajar.

2.2. Seleccion de features

Lo primero a tener en cuenta fue poder extraer de nuestro set de datos las features que le den mayor ganancia a nuestro modelo, para poder generar nuevas features a partir de ellas. Esto nos permite tener un modelo mas complejo y no tan costoso, ya que no vamos a utilizar todos los features sino solo los mas importantes para nuestro modelo.

Para realizar esta tarea, se siguio el siguiente algoritmo:

1. Entrenar el modelo en su totalidad, obteniendo asi la importancia de los features para el modelo.

2. Ordenar estas importancias, quedando asi las importancias de los features de menor a mayor

3. Con las importancias ordenadas (n importancias), se realiza una iteracion donde en cada paso (k pasos) se va generando un nuevo modelo y entrenandolo, con los (n-k) features mas importantes, analizando su score en cada uno de estos pasos.

4. Se analizan los resultados y, realizando un trade-off entre complejidad computacional (cantidad de features) y score del modelo, se elige una cantidad deseada de features.

Una vez extraidos los features mas importantes para nuestro modelo, se prosigue a la generacion de nuevos features a partir de estos.

2.3. Creacion de nuevos features

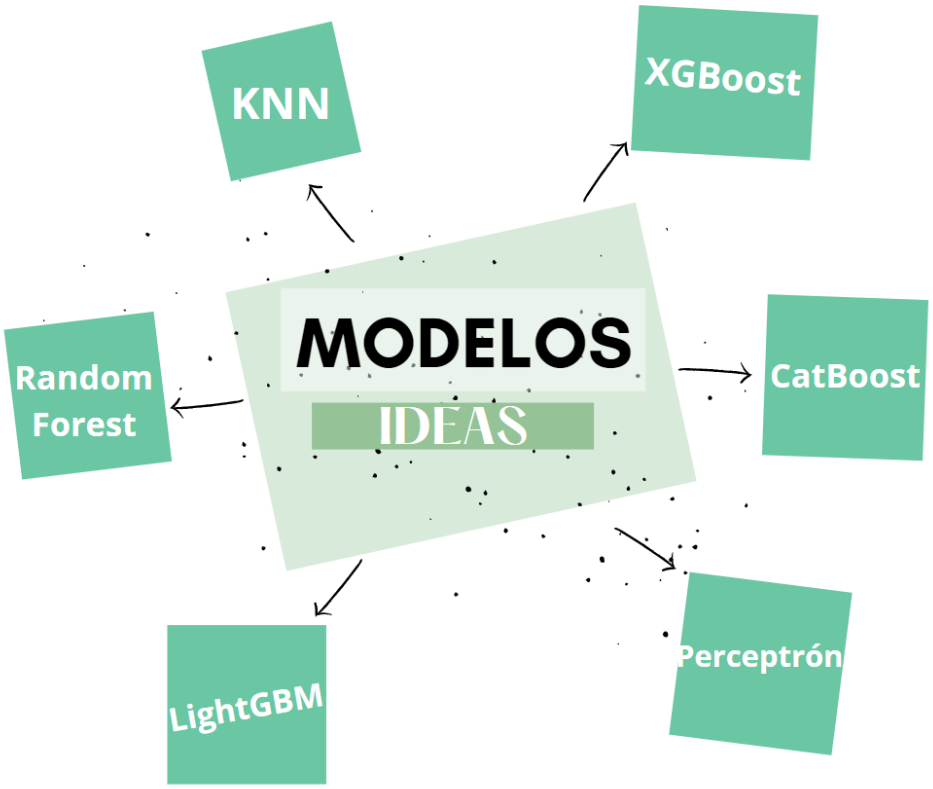
Para la generacion de nuevos features, primero se tuvo en cuenta el tipo de estos features (numerico, categorico, etc), para luego crear features que reflejen distintas interacciones entre los mismos.

Para los features numericos, se analizaron distintas estadisticas, como el promedio, maximo, desviacion estandar, etc., de ese feature respecto de otro, ya sea categorico o numerico. Otra relacion interesante para los modelos basados en arboles, fue realizar los cocientes (u otras operaciones) entre features numericos (ya que estos modelos no pueden hacer esta relacion), como por ejemplo el cociente entre la altura del edificio y la cantidad de pisos, para tener una idea aproximada de la altura de cada piso.

Para los features categoricos, una vez ya encodeados, se realizaron las concatenaciones de los mismos, para que nuestro modelo pueda analizar mas de un feature a la vez.

Para los features booleanos, se decidio concatenarlos como los categoricos, para que nuestro modelo sepa si cierto dato posee mas de una propiedad al mismo tiempo.

3. Modelos



3.1. KNN

K-Nearest Neighbors, o mejor conocido como KNN, es uno de los modelos mas básicos y sencillos que tenemos a la hora de predecir un determinado punto, basandose en la semejanza mayoritaria de los vecinos, justanmente, más cercanos.

Al utilizar mucha memoria, funciona mejor con pocas columnas de datos, lo que comprobamos a la hora de probar todas las columnas vs sólo algunas.

Una de sus desventajas es que es muy sensible a los features muy ruidosos, es decir, datos que estén definitivamente mal. Como por ejemplo, la edad de los edificios. Notamos, en la anterior entrega, que en la columna 'Edad' había datos que, al parecer como no se sabían que edad tenían, solo que eran muy antiguos, se les dió una edad de 900 años. Estos datos mal dados, son una de las cosas que hacen que KNN no sea el mejor de los modelos a utilizar para nuestro set.

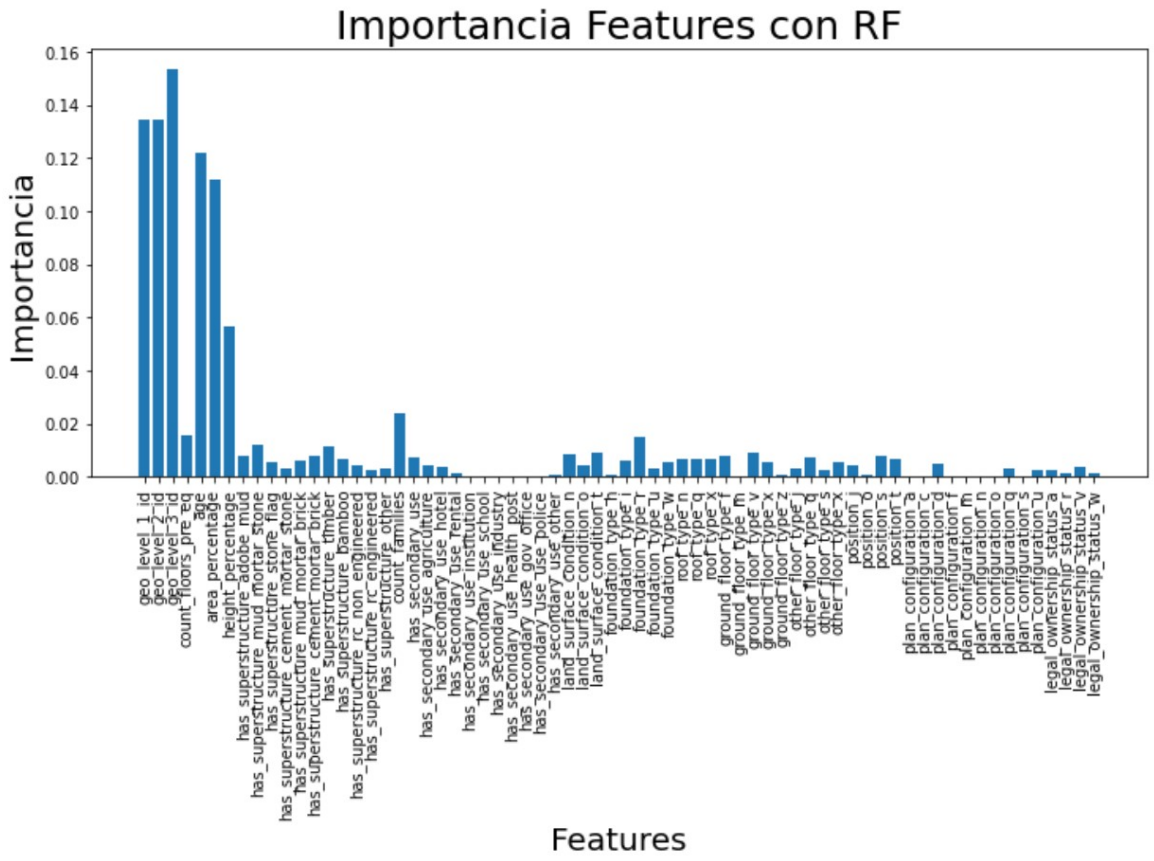
Sus resultados fueron de los mas bajos una vez subidos a la competencia, por lo que pudimos presenciar su tendencia a overfitear.

3.2. Arboles de decisión

3.2.1. Random Forest

Comenzamos con el algoritmo de Random Forest ya que suele producir buenos resultados y se intentó aprovechar ya que evita realizar overfitting con los datos. Este algoritmo aplica bagging a árboles de decisión usando un subset de los atributos del dataset.

Intentamos generar y seleccionar features, observando la incidencia en el modelo, probando así quitando features y evaluando nuevamente el modelo, sin embargo esto no logró una mejoría y terminó quedando un mejor puntaje con una mayoria de features o una selección por 'geo_level'



Además a través de gridSearch realizamos un tunning de hiperparámetros tratando de ajustar mejor los mismos y optimizar Random Forest pero el resultado no terminó obteniendo mucho mayor puntaje.

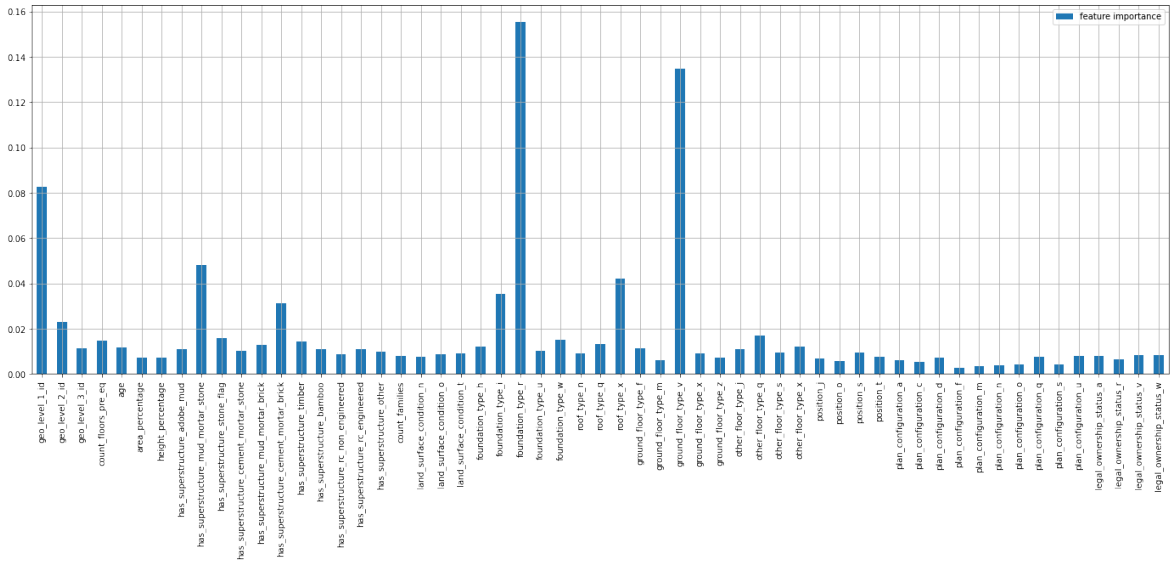
3.2.2. XGBoost

El siguiente algoritmo utilizado fue XGBoost también por su popularidad en las competencias de machine learning tanto de regresión como de clasificación. Este consiste en un boosting de árboles de decision, es decir, es un ensamble de arboles donde cada árbol intenta corregir lo que predijo el anterior.

En un principio se intentó predecir con un regressor, arrojando resultados intermedios entre los esperados, y luego redondeandolos para llegar a la categoría esperada pero esto nos daba un resultado no muy bueno. Esto se debe a que de todos los features

que tenemos solo 4 son númericos y el resto son categóricos. Entonces comenzamos a trabajar con un classifier y en base a eso fuimos obteniendo cada vez mejores resultados, hasta así lograr una de las dos mejores puntuaciones de nuestro grupo en la competencia (0.7441).

Para lograr esto se fue probando con distintas features, mirando el feature importance, y se realizaron sistemáticos tuneos de hiperpametro con gridsearch. Así se encontraron unos parametros y features especificos con los que se alcanzó el mejor resultado posible.



Algo a destacar es que siempre al ser los arboles de poca profundidad, cada ejecución del algoritmo era relativamente rápida, por lo que nos permitió realizar mayor cantidad de intentos.

3.2.3. CatBoost

CatBoost es reconocido por ser un modelo que funciona muy bien con variables categóricas, además de tener una facil implementación. Es mejor con conjuntos de datos relativamente más pequeños que otros métodos de aprendizaje automático.

En este caso, al probarlo con todas las columnas del dataset obtuvimos un mejor resultado que utilizandolo sólo con algunas de ellas, las cuales son consideradas de las más relevantes.

Tiene una buena función para hacer Tunning, que mejora indudablemente la presición de la predicción. Aún así, el resultado del mismo no fue de los mejores.

3.2.4. LightGBM

Este modelo dió uno de los mejores resultados, en cuanto a la presición buscada, pudiendo comprobar los bueno que es comparado con otros árboles ya que usa buckets basados en histogramas. Además, su manera de construir el árbol, que, a diferencia de los demás, lo hace de forma vertical, es decir, que intenta ser mas profundo.

Su gran variedad de hiper-parámetros fue lo que nos permitió abarcar de mejor forma la predicción. La decisión que nos llevó a utilizarlo fue que, como dice su nombre, es un modelo rápido, que además utiliza menos espacio de memoria.

LightGBM agrega un límite de profundidad máxima por encima de las hojas para el overfiting y garantizar una alta eficiencia. Pero aún así corrimos con la desventaja de tropezamos con un overfit muy 'pequeño' en comparación con el de otros modelos, teniendo un valor de Accuracy de 0.7513 a la hora de probarlo con los datos que nosotros poseíamos, y luego obtuviendo un 0.7441 como resultado final, cuando se subió a la competencia.

3.3. Redes neuronales

3.3.1. Perceptrón multicapa

Para probar otro enfoque, nos preguntamos si el dataset se podría ajustar mejor utilizando un perceptrón multicapa, intentando aprovechar que pueda resolver problemas que no son linealmente separables.

Inicialmente se separó los features analizados anteriormente como los más importantes y se entrenó el modelo. Resultó ser muy sensible a los parámetros ingresados, pero siempre sin lograr un resultados destacable y resultando mejor con los valores por default del modelo.

4. Submission Format

4.1. F1 Score

Para evaluar la métrica y los modelos utilizados utilizamos el F1 score que resulta un balance entre la presición y el recall. Esto quiere decir que toma en cuenta falsos positivos y falsos negativos y resulta útil cuando no hay una distribución uniforme de los datos. En este caso para la fórmula se utilizó el promedio 'micro' que calcula la métrica globalmene a partir de contar el total de falsos positivos y negativos.

Esta métrica es útil para obtener una estimación de una clasificación que tiene multi clases como este dataset, con muchos datos categóricos.

4.2. Búsqueda del mejor Submit

Primero es necesario presentar un modelo y plantearnos qué features queremos utilizar para entrenar el modelo propuesto. A partir de que esté entrenado, se predicen los datos con el set de entrenamiento. Para obtener un puntaje real, se obtienen los datos de un archivo dado por driven data el cual presenta un nuevo set de datos y se procede a realizar una nueva predicción con un split de los datos y una nueva predicción para corroborar que no sea haya realizado overfitting. Una vez calculado el f1_score localmente y corroborado que es un buen puntaje procedimos a subirlo a la página, quedándonos únicamente con los building_id y las predicciones obtenidas del damage grade. A partir de este formato se guarda el data frame en un csv para ser submitteado en la página.

Submittear los resultados obtenidos es una gran herramienta para corroborar el trabajo realizado, ya que en caso de estar overfitteando se va a obtener un resultado mucho menor al obtenido localmente. Esto puede suceder en caso de estar pasando demasiados features y generando que el modelo se aprenda el dataset y es necesario revisar las columnas utilizadas, o revisar que no se esté filtrando el label.

5. Conclusión

En cuanto al set de datos, los features que más relevantes resultaron fueron los geo_levels y los features numéricos (count_floors pre_eq, age, area_percentage y height_percentage) ya que son los que están relacionados más directamente con el damage_grade de los edificios. De la misma forma, los datos categóricos sobre el uso secundario del edificio no nos aportó mayor información. Sin embargo algo que aprendimos es que en la mayoría de los casos quedarnos con los features importantes y descartas los que menos feature_importance tenían no nos beneficiaba en lo absoluto. Además, cabe mencionar que no tuvimos mayor inconveniente con el manejo de memoria en nuestras pruebas, pudimos trabajar con el dataset completo sin tener mayores complicaciones.

En cuanto a los mejores algoritmos, los arboles de decisión como XGBoost o LightGBM sin duda resultaron siendo los mejores predictores, sin embargo ambos poseen una gran cantidad de hiper-parámetros los cuales hubo que investigar para conocer cómo afectan al algoritmo, y además buscar cuáles son los mejores. Esto conlleva mucho tiempo de procesamiento, pero es necesario para poder mejorar el score final en la competencia. Sin embargo, notamos que horas de tunning tan solo mejoraban un par de puntos el score comparado con las pruebas manuales.

A modo de conclusión personal, coincidimos en que el machine learning es un trabajo que implica mucho más que solo el aplicado de un algoritmo a un set de datos, y que en su totalidad con un poco de estudio se pueden lograr predicciones muy interesantes.