Trabajo Práctico II

Organización de Datos [75.06]

Primer cuatrimestre de 2021

Betz, Ghersi, Fontela, Francetich, Rosenblatt

104348, 104330, 103924, 104048, 104105

Índice

1. Introducción

Este informe busca presentar e informar sobre el predictor de daño de terremotos que programamos. Comentaremos los detalles de los diferentes modelos de machine learning que realizamos, feature engineering, etc.

El mismo, en resumidas cuentas, intentará categorizar el grado de daño que recibirá una estructura en función de la data correspondiente al terremoto de Gorkha en Nepal del año 2015.

1.1. Terremoto

El 25 de abril de 2015, en Nepal, ocurrió un sismo de magnitud de entre 7,8 a 8,1 puntos en la escala sismológica de magnitud. Se registró a las 6:11 GMT (11:57 hora local) del sábado 25 de abril de dicho año y que afectó a ese país del Asia del Sur. Tuvo su hipocentro a una profundidad de 15 km y su epicentro se localizó a 81 km al noroeste de la capital Katmandú, concretamente en el distrito de Lamjung.

Ha sido el terremoto más grave que ha sufrido Nepal desde el acontecido en 1934. Con casi 9000 víctimas mortales, aproximadamente 22.000 heridos y altos costos producto del daño material, entre ellos la plaza Basantapur Durbar de Katmandú y otros edificios emblemáticos, como la torre Dharahara y el templo Manakamana esta resulta ser una gran catástrofe.

1.2. Feature a Predecir

1.2.1. Información

El DataSet cuenta con 260601 edificaciones particulares de las cuales se tienen registro único. Para estas se hallan datos específicos que se utilizan

para el análisis de los desastres ocurridos.

Los dos DataFrames no cuentan con datos *null*. Se intentará predecir la columna de *Damage Grade* el cual está compuesto por los grados de daño $1, 2, 3$.

2. Desarrollo

2.1. Análisis exploratorio

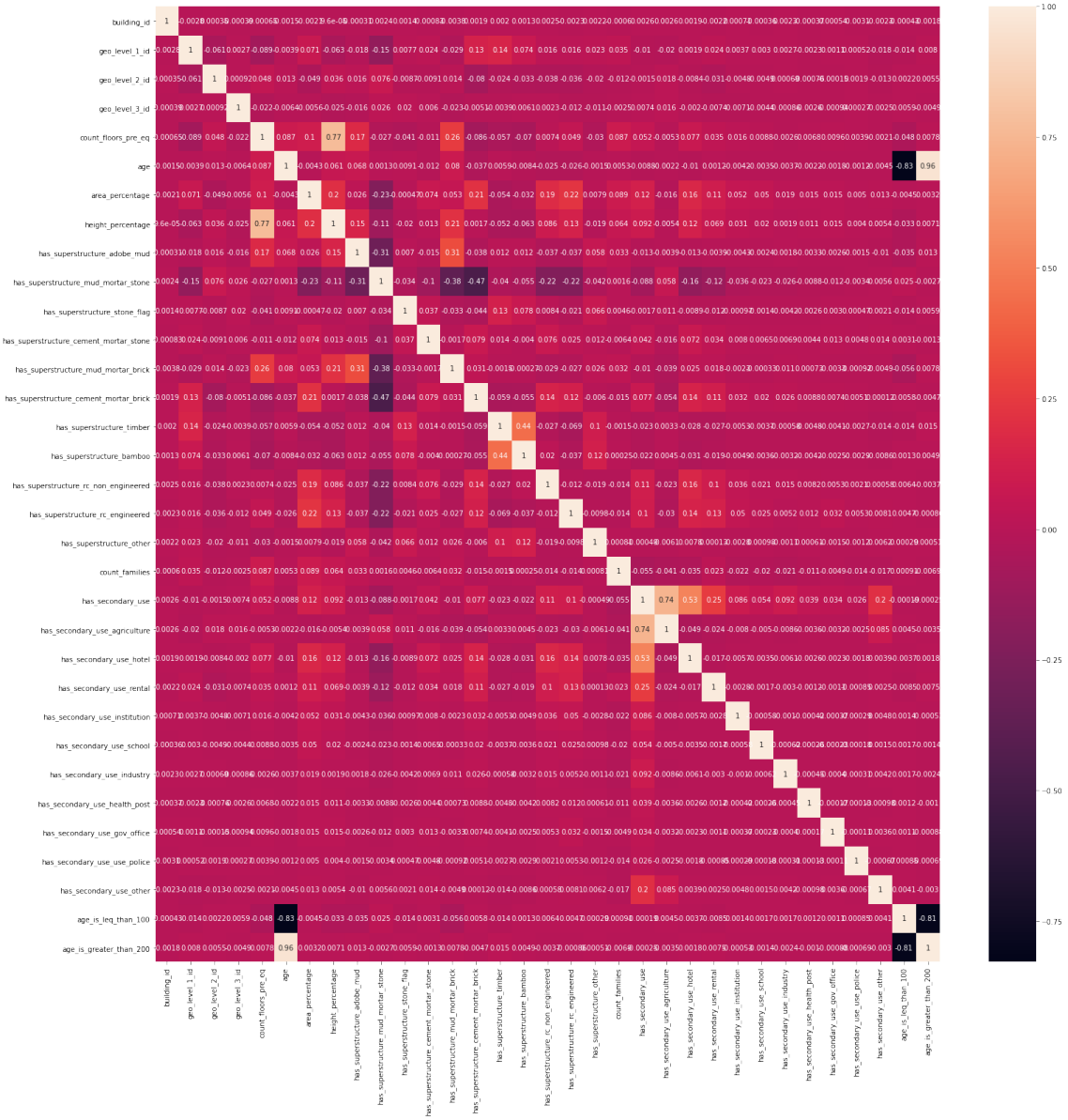
Primero y principal, lo que hicimos fue analizar como están conformados nuestros datos. De esta forma tenemos un conocimiento general sobre lo que vamos a trabajar.

Cada fila representa un valor, cada valor consiste en una descripción sobre la fila misma y cada columna es una representación de los valores del dataset.

Se utilizó el dataset train para entrenar y testear los distintos algoritmos utilizados durante el trabajo. Dicho dataset fue modificado a lo largo del tiempo, quitando aquellos datos que no fueran útiles; para lograr eso hemos recurrido al Feature Engineering y Encoding. A continuación detallaremos como fueron utilizados.

2.2. Feature Engineering

Sirve para limpiar el dataset, ayuda mucho a saber cuales columnas (features) aportan mejor información que otras y si hay que eliminarlas o crear nuevas para la optimización del algoritmo de predicción.



2.3. Encodings

El encoding es el proceso a través del cual se transforma información textual humana (caracteres alfabéticos y no alfabéticos) en un conjunto más reducido, para ser almacenado o transmitido. Se utilizó para encodear las columnas de tipo categórico y transformarlas a numérico.

Este es un proceso que debe ser hecho con distintos algoritmos de encoding, ya que si se hacen cambiando cualquier variable categórica a números, el algoritmo de Machine Learning puede encontrar patrones en esos números que son completamente inventados y sin ningún tipo de sentido, perjudicando de esta forma, al performance del algoritmo.

2.3.1. One Hot Encoding

El primer OneHotEncoder que probamos para las features categóricas, importado de la biblioteca SKlearn.

Esta herramienta nos dio una seguridad mucho mayor con respecto a la features que estábamos utilizando.

```
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
    res = res.drop([feature_to_encode], axis=1)
    return(res)

features_to_encode = ["geo_level_1_id","geo_level_2_id",
                      "land_surface_condition", "foundation_type",
                      "roof_type", "position",
                      "ground_floor_type", "other_floor_type",
                      "plan_configuration", "legal_ownership_status"]

for feature in features_to_encode:
    X_train = encode_and_bind(X_train, feature)
    X_test = encode_and_bind(X_test, feature)
```

2.3.2. Mean Encoding

La idea de este es codificar las variables categóricas en a base a los promedio de las apariciones de los labels.

Sin embargo, a pesar de que esta codificación puede resultar muy útil a veces, hay que tener bastante cuidado, ya que podría llegar a entorpecer fácilmente al modelo ocasionando que aprende correlaciones inexistentes.

3. Modelos

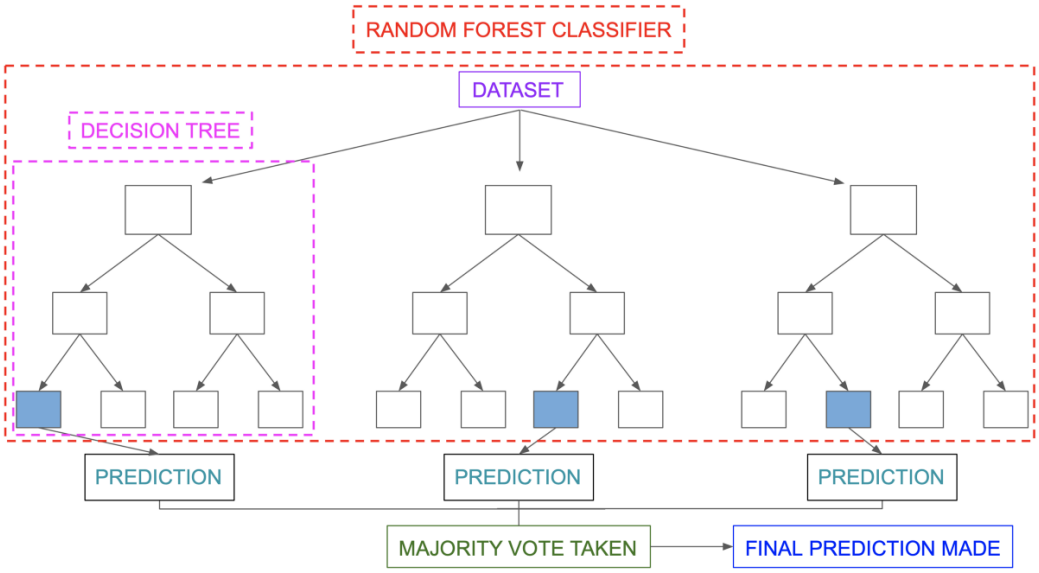
A lo largo del desarrollo de este trabajo práctico, consideramos diversos modelos de Machine Learning, como Arboles de decisión, XGBoost, CatBoost, LightGBM, KNN y Neuronal Links. Debido a esto, pudimos corroborar debilidades y fortalezas de cada uno de estos, así mismo, un modelo que generó la mejor precisión.

3.1. Arboles de decisión

3.1.1. Random Forests

El Random Forest es un modelo que construye múltiples arboles de decisión y toma como decisión el valor que más se repite entre sus arboles en el caso de un modelo clasificador y el promedio de sus arboles en caso de utilizar un modelo regressor.

Para ejemplificar el funcionamiento de un árbol de decisión observemos el siguiente gráfico que muestra como funciona un árbol de decisión clasificador.

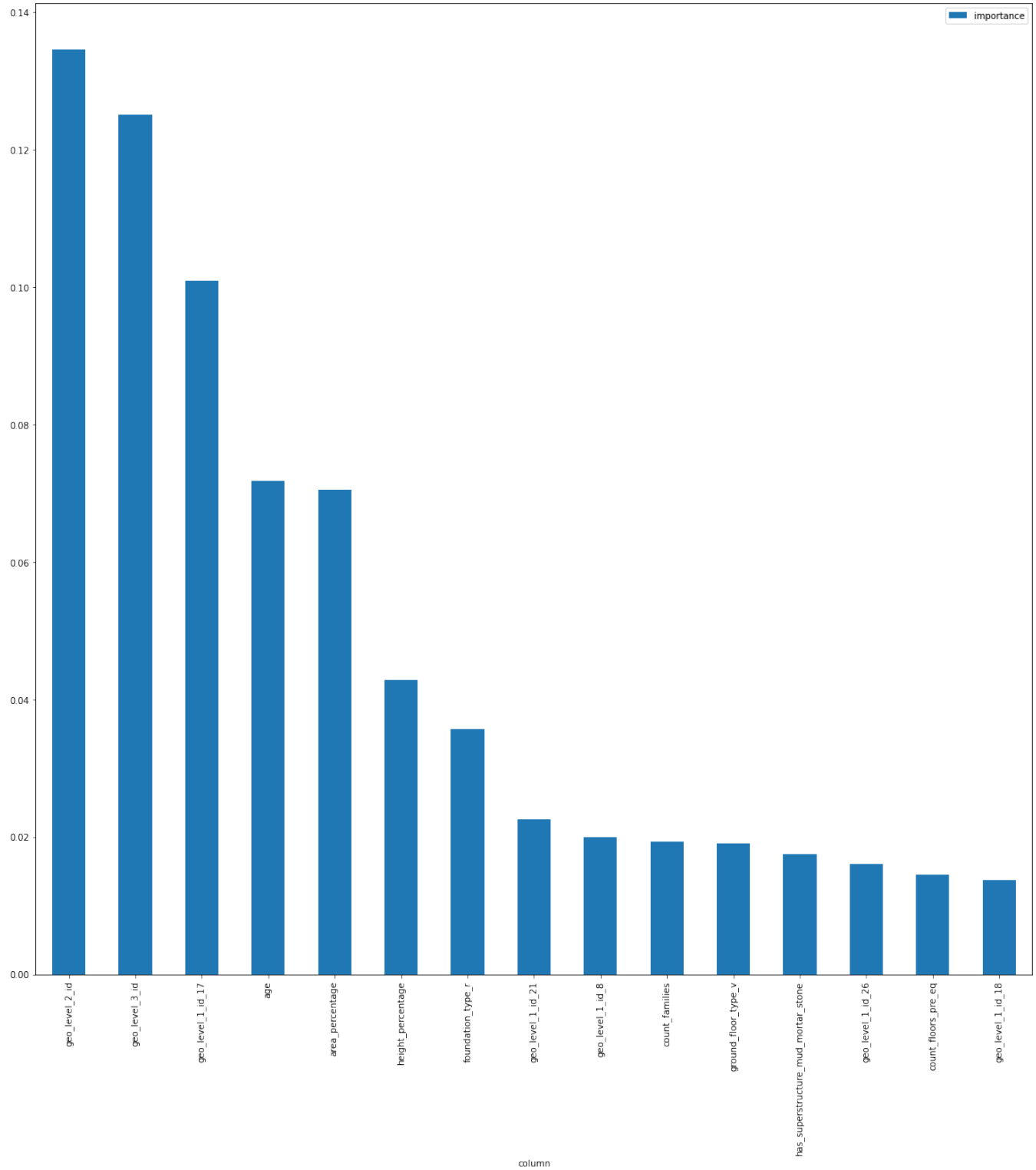


Podemos observar como el árbol impone una condición sobre un feature y basado en el resultado de esa condición se sigue una de las ramas que a su vez tienen sus condiciones propias. Esto se repite hasta llegar a un nodo hoja que es el que tiene la decisión final que se debe tomar.

Estos arboles pueden tener mucha más profundidad, pero se utilizo uno relativamente chico por motivos de visualización, pero la lógica se mantiene para un árbol de cualquier profundidad.

Inicialmente, entrenamos un modelo clasificador utilizando features tanto categóricos (encodeados) como numéricos, lo cual devolvió un modelo con una performance súper aceptable.

Pudimos aumentar la precisión de este modelo utilizando un Grid Search de la librería SKlearn modificado a nuestro criterio, el cuál nos facilito los hiperpárametros que necesitábamos para optimizar el Random Forest.


importance
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
geo_level_2_id
geo_level_3_id
geo_level_1_id_17
age
area_percentage
height_percentage
foundation_type_r
geo_level_1_id_21
geo_level_1_id_8
count_families
ground_floor_type_v
has_superstructure_mud_mortar_stone
geo_level_1_id_26
count_floors_pre_eq
geo_level_1_id_18
column

Importancia de cada uno de los 15 features más importantes para el modelo en cuestión.

3.1.2. Trabajo Realizado

En un principio empezamos con:

- `n_estimators = 100`
- `max_depth = 15`
- `max_features = 45`

Y utilizamos los features:

- `geo_level_1_id`
- `foundation_type`
- `ground_floor_type`
- `age`
- `area_percentage`
- `y las que indicaban las superestructuras.`

Además, encodeamos las columnas categóricas y 'geo_level_1_id' con OneHotEncoding. El f_1 score resultó ser 0,6770.

En la próxima iteración aumentamos los 'n_estimators' a 150 luego de verificar a partir de testeos manuales que este era un buen valor. Por otro lado, dejamos de lado la reducción de la cantidad de columnas, comenzando a utilizar todas las del dataset ya que eso nos generaba los mejores resultados. El f_1 score aumentó a 0,7108.

Corriendo varias veces el GridSearch mejoramos el f_1 score cambiando:

- `n_estimators := 150`
- `min_samples_split := 15`
- `min_samples_leaf := a 1 (aunque podría generar overfitting).`

Eliminando además la máxima profundidad al árbol (esto también podría generar overfitting en teoría, pero en la práctica mejoraba la performance). Esta resulta ser la quinta iteración de cambios y el f_1 score resultó ser 0,7392.

Luego excluyendo del entrenamiento aquellas columnas que un entrenamiento anterior del modelo nos decían que tenían pequeña importancia y ajustando:

```
- max_features := 0.2
```

Obtuvimos un f_1 score de 0,7331. Por lo tanto, al reducirse la performance del modelo, la idea fue descartada.

Luego, alrededor de la novena iteración eliminamos todo el feature engineering que veníamos probando desde el sexto modelo (la ya mencionada reducción de la cantidad de columnas del dataset) ya que nos traía, a pesar de nuestra sorpresa, peores resultados. El mejor modelo seguía siendo el de 0,7392.

Finalmente agregamos el hiperparámetro 'class_weights' con valor 'balanced' para balancear el peso de cada registro según el grado de daño (ya que habia menos registros de grado de daño 1, que de 2 o 3). Esto bajó en aproximadamente 0,01 el f_1 score.

También se probó el BaggingClassifier para el Random Forest, pero al ser un método que ya incluye la randomización, el f_1 score permanece igual.

3.2. Gradient Boosting

Gradient Boosting, es una técnica de aprendizaje automático utilizado para el análisis de la regresión y para problemas de clasificación estadística, el cual produce un modelo predictivo en forma de un conjunto de modelos

de predicción débiles, típicamente árboles de decisión.

Primero preparamos los set de datos cambiando los tipos de las columnas numericas a float64, y transformando en categoricas a los objects para luego poder hacerles OneHotEncoding. Luego hicimos un GridSearch utilizando el ya implementado por Sklearn con multiples valores para cada hiperparametro, Estos valores fueron bastante espaciados entre si para darnos una idea de "para donde"mejoraba el modelo y para "donde empeoraba"para poder asi probar luego valores detro de la interseccion de los hiperparametros que salian victoriosos del GridSearch.

Luego de seguir esta metodologia por un par de dias conseguimos un modelo optimo con los hiperparametros:

```
-n_estimators = 350,
-max_depth = 9,
-min_samples_split = 2, #default
-min_samples_leaf = 3, #default
-subsample=0.6,
-learning_rate=0.15
```

Un comentario que queremos hacer acerca de los hiperparametros es que notamos que al aumentar los n-estimators, el modelo tendia a mejorar; sin embargo esto alargaba mucho el tiempo de entrenamiento y no era una mejora lo suficientemente sustanciosa para que valiese la pena. Por estos motivos lo que los dejamos en 350 estimators, una cantidad que obtenia buenos resultados sin tener que esperar varias horas para ver como variaba el modelo ante los cambios que le intentabamos hacer.

Luego de encontrar estos parametros intentamos hacer feature engineering quitando columnas que nos parecian ruido como las del uso que se les daban a las estructuras sin embargo quitarlas solo nos reducia el puntaje. Despues probamos crear otras columnas como una de volumen o booleanas

de si cumplian o no con cierta condicion (como si supera x cantidad de años, pisos, etc) sin embargo esto tampoco mejoro el puntaje. Por ultimo probamos aplicarle una operacion matematica a algunas columnas numericas, mas precisamente buscamos aplicarles un cuadrado (con la idea de amplificar las distancias) y aplicarles log(valor + 1) (para normalizar algunas columnas) sin embargo esto tampoco nos dio buenos resultados.

Al final nos quedamos con un puntaje de 0,7432145506638357, Un resultado bastante bueno pero como en simultaneo estabamos trabajando con XGBoost y CatBoost dejamos de intentar subir este modelo para focalizarnos en esos dos.

3.2.1. 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.

Lo que hace bueno a XGBoost es: La velocidad y el rendimiento del algoritmo central es paralelizable, tiene algoritmos de resolución de modelos lineales y de aprendizaje de árboles. Entonces, lo que lo hace rápido es su capacidad para realizar cálculos paralelos en una sola máquina y el rendimiento de vanguardia en muchas tareas de Machine Learning.

El GridSearch a partir de este modelo lo realizamos de manera manual, es decir, implementamos uno propio. Esto fue debido a que notamos que en ciertas situaciones, el resultado que el GridSearch de SKlearn indicaba como óptimo, no lo terminaba siendo para nuestro set de validación (que en gran medida estaba correlacionado con el resultado final en DrivenData).

A partir de varios GridSearch exhaustivos (mas de 100 combinaciones que debían correr durante toda la noche), conseguimos un primer óptimo con los hiperparámetros:

- `n_estimators := 350`
- `subsample := 0.95`
- `gamma := 1`
- `learning_rate := 0.55`

En cuanto a feature engineering no hubo cambios con respecto al RandomForest óptimo: no sacamos columnas y las categóricas fueron encodeadas con OneHotEncoding, obtuvimos un f_1 score: 0,7431.

A partir de nuevos GridSearch nos topamos con el modelo óptimo, modificando con respecto al anterior: learning_rate a 0,45 y, con un poco de suerte (ya que la idea era poner otro valor y no este que fue arbitrario) subsample a 0,885, obtuvimos un f_1 score de: 0,743577.

Nuevas ejecuciones de GridSearch nos hicieron darnos cuenta de los afortunados que fuimos al probar el valor 0,885: alejándonos muy poco de este valor, el score bajaba notablemente. Sin embargo, encontramos lo que termino siendo nuestro segundo mejor f_1 score para XGBoost (0,7434), cambiando con respecto al óptimo, gamma a 0,5 y subsample a 0,95.[1]

Acá cometimos un error ya que utilizamos como parámetro de decisión de eliminación de features, el feature_importance que nos dio el XGBoost, que no es un indicador tan acertado como lo puede ser el RandomForest. A partir de los datos de feature_importance, entrenamos nuevamente los mejores modelos quitando las 10 columnas que el XGBoost indicaba como menos importantes. En todos los casos, el f_1 score bajo.

Intentamos ver si cambiaba algo el agregar weighting para reducir la influencia de que hubiera mas registros clasificados con cierto grado de daño que con cierto otro. Sin embargo, el f_1 score permaneció igual.

[1] Se intento predecir con el XGBRegressor aproximando cada predicción al entero mas cercano, sin embargo, el f_1 score fue de únicamente 0,5498.

Agregamos Bagging sobre el mejor modelo de XGBoost. Elegimos 5 n_estimators para el Bagging (mas darían un mejor resultado, pero la diferencia es despreciable y el incremento de tiempo al entrenar es lineal). Con esta optimización, el modelo escala su f_1 score hasta aproximadamente 0,7448.

3.2.2. CatBoost

CATBoost nos provee herramientas para hacer Gradient Boosting para problemas de clasificación.

Los beneficios que ofrece CATBoost son: Ordered Boosting para prever over-fitting y Handling nativo para features categóricos.

Esta fue una de las primeras herramientas de boosting que utilizamos. Sin embargo, este algoritmo resultó esencial en otros métodos de ensamble que explicaremos mas adelante que nos consiguieron nuestros mejores resultados en ese momento, dada la combinación con otros modelos.

CATBoost fue el primer tipo de modelo que nos encontramos con una muy alta cantidad de hiperparámetros disponibles, por lo que estuvimos un tiempo investigando la importancia de cada uno y probando combinaciones que nos generen mejores modelos.

En un principio no nos quedamos con un único modelo. Lo que hicimos fue correr un GridSearch y guardar en un csv los resultados. La estructura del dataset fue la misma que la utilizada para el modelo óptimo de XGBoost.

Gracias a estos GridSearches encontramos un buen modelo:

- `iterations := 5000`
- `max_depth := None`
- `learning_rate := 0.15`

```
- etc
```

Le indicamos al modelo las columnas geo_level_2 & 3_id como categóricas y le pasamos un segundo parámetro, eval_set, con el objetivo de evitar el overfitting. Esto causó distorsiones entre el f_1 score estimado localmente y el obtenido en DrivenData. El f_1 score indicado en los notebooks era de 0,7480 pero en las entregas disminuía 0,0015 puntos aproximadamente.

Luego, con respecto a los dos features indicados como categóricos, nos enteramos que CatBoost considera que los datos están ordenados temporalmente, y a pesar de no ser así esto en nuestro caso, obtuvimos mejores resultados, por lo tanto quedó como parte del modelo.

Aquí ocurre algo similar a XGBoost, cuando uno aumenta las iteraciones, mejora el modelo, pero el tiempo de entrenamiento aumenta linealmente, entonces uno debe decidir cuando no tiene sentido seguir aumentando la cantidad de iteraciones. Finalmente decidimos aumentar la cantidad de iteraciones, de 5000 a 7500, un 50 % más.

Seguimos utilizando el set de validación parámetro eval_set, con lo cual el f_1 score en el notebook fue 0,7492 y en las entregas llego a 0,7476.

3.2.3. LightGBM

LightGBM al igual que XGBoost y CATBoost también es una implementación de Gradient Boosting.

Su principal diferencia con el anterior reside en que sus árboles tienden a crecer verticalmente. Esto genera una mejor perfomance cuando se trabaja con una gran cantidad datos pero asimismo tiende a hacer overfitting con sets de datos pequeños.

Utilizando este tipo de boosting vimos que la performance se incrementa-

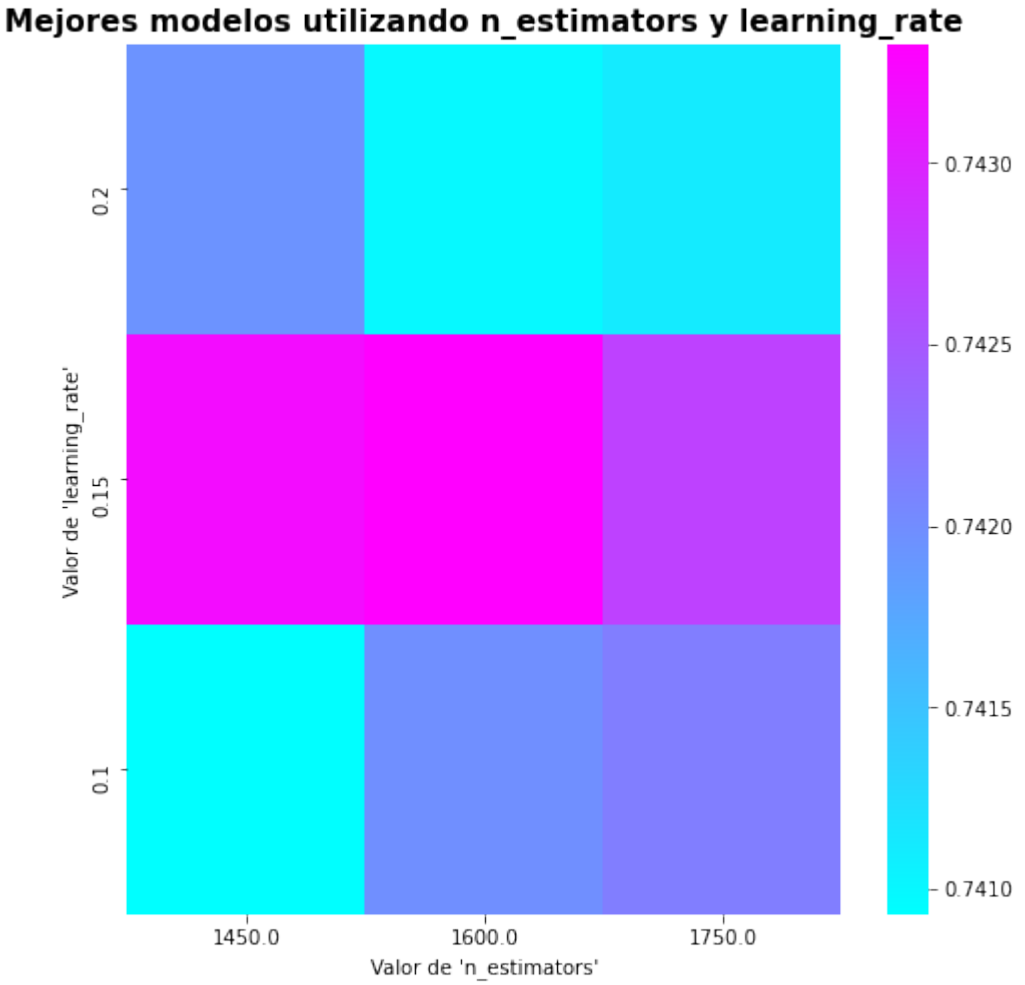
ba en comparación con los otros métodos de boosting, por lo que intentamos dedicarle mas tiempo ajustando sus hiperparámetros.

Una gran ventaja de este modelo es que entrena muy rápido. Cada entrenamiento tardaba en el peor de los casos 30 segundos, con lo cual nos permitía probar muchas cosas distintas sin tener que destinarle días o incluso semanas como con otros modelos.

Implementamos el Grid Search, el cual nos permitió arrancar con buenos hiperparámetros iniciales. En cuanto a feature engineering, seguimos con el mismo modelo de utilizar todas las columnas. En este caso en puntual probamos, además, aplicar mean encoding para geo_level_2 & 3_id, pero el resultado fue peor que si no lo hubieramos hecho (para sorpresa nuestra) con lo cual la idea quedo descartada, el f_1 score de esta iteración fue de 0,7366.

Quitando el mean encoding, el OneHotEncoding al geo_level_1_id, aumentando el hiperparámetros n_estimators de 1000 a 1600 y dejando el resto igual obtuvimos un f_1 score de 0,7395.

Luego, pasamos nuevamente a 1000 n_estimators, y aplicamos OneHotEncoding a las columnas geo_level_1 & 2_id. A pesar de obtener un muy buen f_1 score, decidimos de ahí en mas solo encodear el geo level 1, ya que al hacerlo con el 2 nos agregaba demasiadas columnas extra debido a la gran cantidad de valores posibles. Empleando estos elementos obtuvimos un f_1 score de 0,7438.



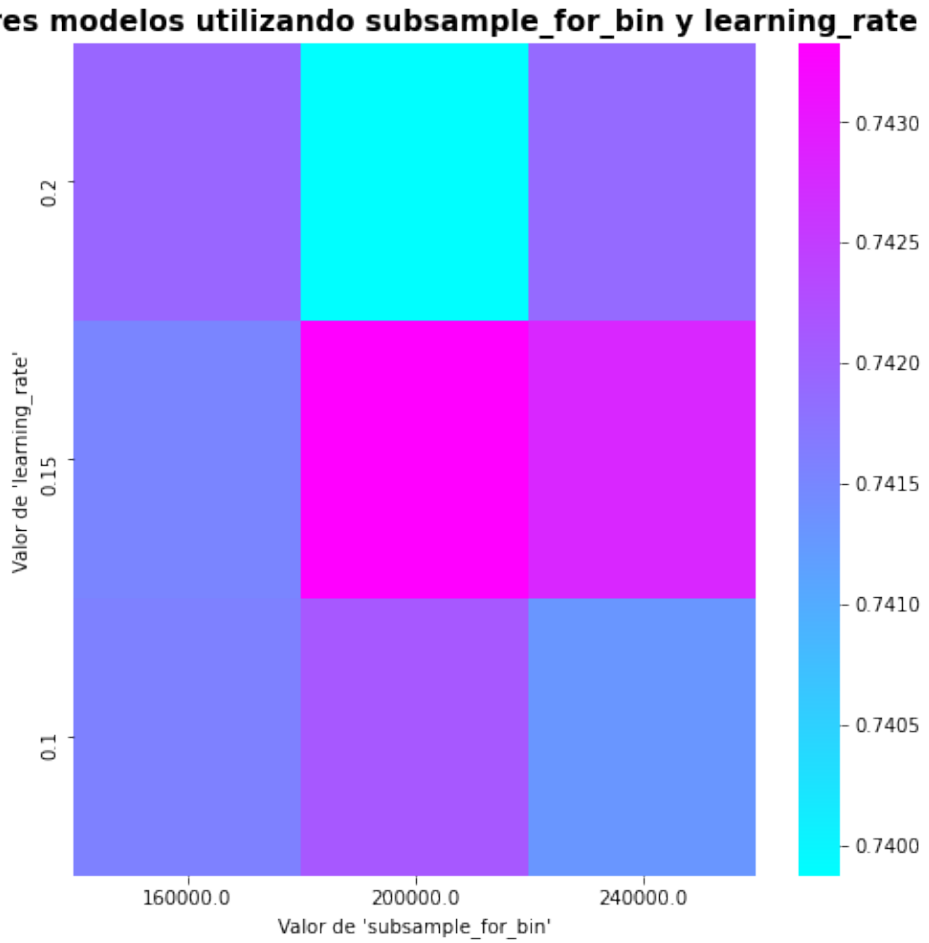
Dejamos de lado el OneHotEncoding para geo_level_2_id por la razón mencionada anteriormente y aumentamos levemente los n_estimators hasta 1800 debido a un resultado de un GridSearch. Además, realizamos un extra de feature engineering, calculando el volumen de las construcciones, la cantidad de familias por unidad de área y la altura promedio por piso. A pesar de agregar información extra, el f_1 score del modelo cayo hasta 0,7404 con lo cual se deshecho la idea.

Probamos balancear los registros según la clase (grado de daño), manteniendo todo el resto del modelo igual al óptimo encontrado (0,7433 usando el cuarto sin el OneHotEncoding de geo_level_2_id). Sin embargo, esto hizo

descender el f_1 score del modelo con lo cual se descarto la idea.

Al igual que en las otras iteraciones, utilizamos Bagging para intentar mejorar el modelo por si solo. En este caso obtuvimos una mejora notable, obteniendo un f_1 score de aproximadamente 0,7456 (incremento de 0,0023) utilizando 20 n_estimators para el Bagging.

Se intento nuevamente implementar el mean encoding para los geo_level para ver si cambiaba algo con el modelo optimizado. Sin embargo, no hubo resultados positivos: el geo_level_1_id encodeado mediante esta técnica casi no vario el f_1 score, y el 2 y el 3 lo empeoraron.



3.3. KNN

KNN es un algoritmo de clasificación de Machine Learning. Este funciona de forma que predice la clase de un punto a partir de la mayoría de sus k-vecinos más cercanos.

Este algoritmo no necesita tiempo de entrenamiento ya que calcula los vecinos a la hora de predecir, los tiempos de predicción son mayores a los de otros modelos.

Existe una desventaja en este algoritmo llamada el problema de la dimensionalidad, en este algoritmo a medida que aumentamos la cantidad de dimensiones se vuelve cada vez menos eficiente y necesitamos de mas cantidad de datos.

Para poder usar KNN hay que definir el valor de k, es decir, cuantos vecinos vamos a considerar. Para encontrarlo primero hicimos un pequeño GridSearch para hallar principalmente la forma optima de calcular los vecinos cercanos. Es asi que corrimos las opciones mas populares de los hiperparametros "metric", .ªlgorithmz "weightsçon un numero fijo de 10 vecinos cercanos.
Al terminar este proceso encontramos que la metrica "manhattan", con el algoritmo "ball_treez con el peso por distancia era lo que mejor resultado nos daba.

Luego de encontrar la mejor forma de medir la cercania pusimos esos hiperparametros fijos y nos dispusimos a hacer un gran GridSearch variando solo los k vecinos cercanos. Esto nos dejo los hiperparametros a usar:

- `algorithm = 'ball_tree',`
- `n_neighbors = 14,`
- `weights = 'distance',`
- `metric = "manhattan"`

Luego nuevamente intentamos hacer feature engineering ya que supues-

tamente KNN es muy sensible a no tener informacion ruidosa que pueda hacerle pensar al modelo que son similares casos que no lo terminan con el mismo label. Sin embargo nuevamente toda columna que quitabamos solo empeoraba el modelo.

El KNN quedo con una puntuacion de 0,7195 bastante debajo de el resto de los modelos pero al ser muy distinto pensamos que podria ser influyente en una mejoria para un modelo de ensamble.

3.4. Voting Classifier

Este es un modelo que permite combinar, mediante votación, las predicciones de distintos modelos. El objeto sera potenciar unos modelos con otros, utilizando los mejores generados por nosotros.

En una primera versión usamos los mejores dos XGBoost, el mejor Random Forest, los dos mejores LGBM y el mejor Gradient Boosting. El parámetro voting lo definimos como 'soft' ya que observamos que daba mejores resultados que 'hard'.

Obtuvimos un f_1 score de 0,7502. El mejor hasta el momento.

Posteriormente intentamos reducir el Voting a las mejores combinaciones obtenidas de hiperparámetros de un único modelo (LightGBM) e incluimos el feature 'geo_level_2_id' con OneHotEncoder para experimentar con los resultados. Esto último nunca lo habíamos intentado por miedo al consumo de memoria. El resultado fue un f_1 score de 0,7285 por lo que la idea fue descartada.

Finalmente, creamos un modelo Voting agregándole nuestro mejor modelo de CatBoost al mejor modelo de Voting que teníamos (el primero de los mencionados en esta sección). Lo entrenamos con la totalidad del dataset, evitando así usar un set de validación, para maximizar nuestro f_1 score. Con

todo esto juntamos un f_1 score de 0,7522.

3.5. Ideas que no Prosperaron

Los siguientes modelos fueron descartados ya que su f_1 score para los modelos fue muy bajo.

3.5.1. Neuronal Links

MLP o Multi Layer Perceptron, es un algoritmo de Redes Neuronales, que mejora al Perceptron Simple (una sola capa de neuronas). El MLP utiliza varias capas y puede resolver problemas que no son linealmente separables

Utilizando el MLPClassifier obtuvimos el mejor resultado para las redes neuronales, el cual esta alejado del resultado óptimo que se consiguió con otros modelos.

3.5.2. Bernoulli Naive Bayes

El f_1 score mas alto al que se lo logro llevar fue 0,6242, y con Bagging no alcanzo el 0,63.

3.5.3. LogisticRegression

Se hizo un intento de incluirlo en el ensamble final ya que alcanzo un f_1 score de 0.68 con el tuneo de los hiperparámetros (y además es un modelo distinto a la mayoría de los presentes en el ensamble), pero fracasó ya que en todos los casos disminuía el f_1 score del modelo principal.

3.5.4. NearestCentroid

El f_1 score inicial fue de 0,44 y sin muchos hiperparámetros a tunear, con lo cual se descartó inmediatamente.

3.5.5. SGDClassifier

Nunca finalizó de correr. Se recopiló información acerca de que es un método que no es eficiente para grandes volúmenes de datos.

3.5.6. SVC

El f_1 score inicial fue de 0.57 con lo cual también se descarto rápidamente.

4. Modelo de Mejor Precisión

Nuestro mejor modelo hasta el momento era uno de tipo Voting Classifier. Decidimos cambiar a uno similar de la librería MLXtend (Ensemble Voting Classifier) pues nos permite enviarle mensajes a los modelos ya entrenados, potenciar unos modelos con otros, utilizando los mejores generados por nosotros, lo cual nos trajo dos beneficios:

- Podemos probar distintas combinaciones de pesos relativos entre los distintos modelos sin tener que reentrenarlos a todos.
- Poder pasarle parámetros al método fit de cada modelo en especifico (especialmente útil para el CatBoost).

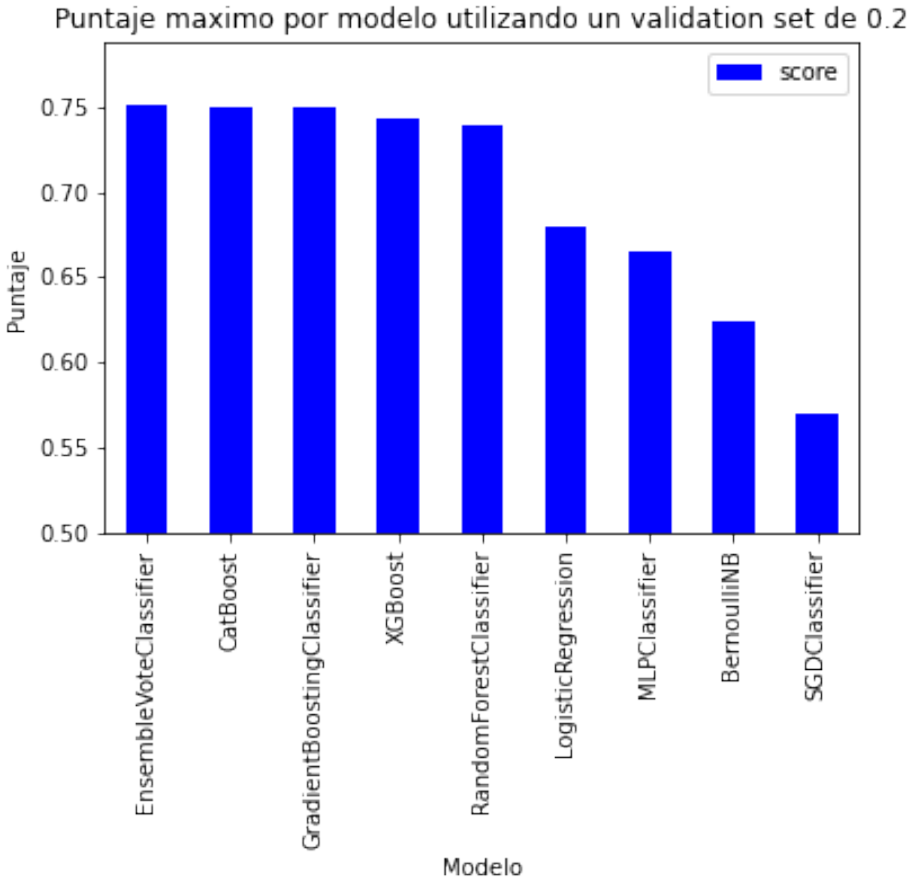
Utilizamos nuestros mejores modelos para cada tipo (XGBoost, RandomForest, LGBM, CatBoost, GradientBoosting). Además, agregamos los pesos que, probando a mano, nos generaron los mejores resultados. Arrancamos con un f_1 score de 0,7501.

En este caso, el nuevo cambio introducido tiene que ver con el Bagging. Ahora, para los modelos en los que notamos una mejora al aplicar esta técnica, la utilizamos antes de enviarlos al Voting. Por lo tanto, hicimos un Bagging del XGBoost y del LGBM. RandomForest no mostró mejora con Bagging (como ya mencionamos en la correspondiente sección). CatBoost y GB son modelos que tardan demasiado en entrenarse por si solos, el bagging los haría tardar mucho mas. Con esta nueva combinación, sumada a una

combinación de pesos relativos, obtuvimos un f_1 score de 0,7504.

Con un modelo idéntico al antes mencionado pero utilizando el 100 % del dataset, nos devolvió un f_1 score de 0,7522.

Realizamos una iteración que contenía un retoque de los pesos relativos del modelo anterior. En el que quedó como quinto modelo intentamos incluir la regresión logística como sexto modelo del Voting. A pesar de que parecía dar mejor resultado localmente, al momento de subirlo a DrivenData dio un resultado peor. Tener en cuenta que la regresión logística daba un f_1 score mucho peor comparada al resto de los modelos (0,68 versus el peor de los otros era 0,739). Finalmente, quitando la regresión logística, pero manteniendo los pesos relativos de los modelos, al subirlo a driven data con el 100 % del data set obtuvimos un f_1 score de: 0,7524. Este fue nuestro mejor modelo.



5. Conclusiones

En conclusión los diferentes modelos nos aportan diferentes formas de predecir el feature particular para cada uno.

Debido a la buena accuracy en nuestro set de test, nos dimos cuenta que los modelos nos estaban dando los resultados que esperábamos en el set de datos. Por lo que, decidimos profundiza en experimentar con los modelos clasificadores.

A lo largo del trabajo práctico, investigamos sobre los distintos hiperparámetros

metros que ofrecía cada modelo y como estos afectaban las predicciones que hacían. Los métodos de selección de modelos que nos provee sklearn, como el Grid Search, que nos ayudó fuertemente a la hora de elegir que modelo era el mejor para nuestros datos ya que podíamos tener un set de posibles hiperparámetros y el Grid Search intentaba todas las combinaciones posibles, los que nos ahorro mucho trabajo manual.

Por ultimo, investigamos sobre los distintos tipos de ensambles y como tomaban sus decisiones a partir de los modelos que teníamos. Tras probar con varios, nos decantamos por utilizar el Ensemble Voting Classifier como el ensamble final ya que era el que nos daba las mejores predicciones sobre el set de test.

A lo largo de este trabajo, se han puesto a prueba múltiples metodologías con el fin de elaborar un robusto modelo de Machine Learning capaz de predecir correctamente la naturaleza de cada oportunidad. Este trabajo nos sirvió además para tener experiencia de primera mano en la utilización de esta ciencia en un uso cotidiano y en su uso en la ciencia de datos. No solo esto sino que también, este trabajo práctico sirvió para darnos cuenta lo complejo que es el mundo del Machine Learning, no basta solo con usar un algoritmo sino todo lo que hay por detrás, las distintas técnicas de Feature Engineering o de busca de hiperparámetros.

Además, notamos que ante mínimos cambios, ya sea de los valores de los hiperparámetros o el agregado de un feature, el score empeoraba o mejoraba de forma notable, demostrando el efecto avalancha. Cambios que creíamos importante terminaban empeorando su comportamiento o cambios muy chicos benefician más de lo esperado.

6. Repositorio y Vídeo

El código fuente del trabajo se encuentra en el siguiente repositorio.

Aquí se encuentra el vídeo de presentación.