FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Richter's Predictor: Modeling Earthquake Damage

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

Alumno 1:	LUIZAGA, Ricardo - 87528
Alumno 2:	CONDE, Iván - 102497
Alumno 3:	ALVARO, Enrique - 103991

Índice

1. Introduccion

Según los aspectos de la ubicación y la construcción de los edificios, nuestro objetivo es predecir el nivel de daño a los edificios causado por el terremoto de Gorkha de 2015 en Nepal.

Los datos 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. Esta encuesta es uno de los conjuntos de datos posteriores a un desastre más grandes jamás recopilados , que contiene información valiosa sobre los impactos de los terremotos, las condiciones de los hogares y las estadísticas socioeconómicas y demográficas.

2. Objetivos

Los objetivos de este trabajo práctico son:

- Entender los diferentes modelos de machine learning vistos a lo largo de la cursada.
- Familiarizarse con las diversas métricas presentadas y aprender a compararlas entre sí.
- Entender y aprender distintas formas de preprocesar los datos.
- Aprender los caminos que hay para obtener las mejores combinaciones de hiper parámetros en los diferentes modelos.

Este trabajo va a consistir en el procedimiento necesario para poder conseguir buenos resultados a la hora de probar distintos algoritmos de machine learning.

Para dar un poco de contexto, esta investigación se desarrolló bajo la materia Organización de Datos de la facultad de ingeniería de la Universidad de Buenos Aires. Al mismo tiempo se basa en una competencia de DrivenData llamada "Richter's Predictor: Modeling Earthquake Damage".

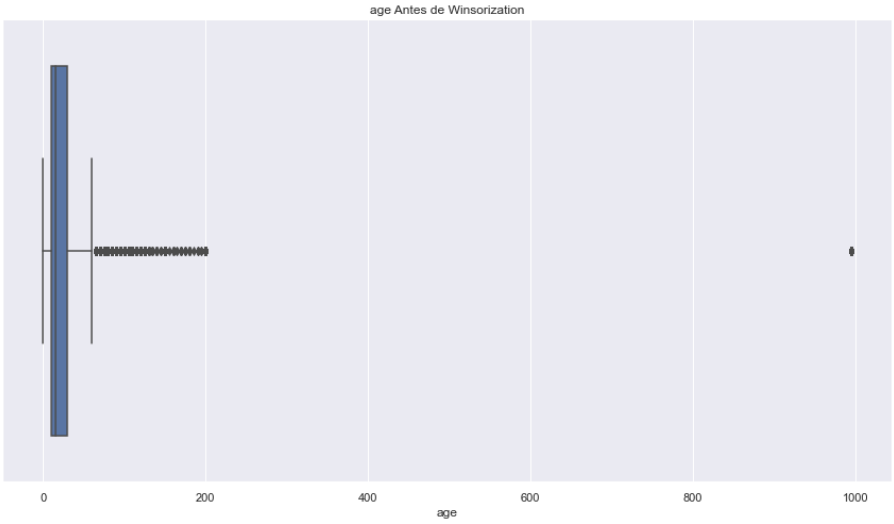
Previo a esta presentación hicimos un análisis exploratorio. En base a los resultados obtenidos vamos a tener una base sobre que datos interesantes se pueden generar como features. Por ende, luego se comienza un amplio proceso de preprocesamiento de los datos y feature engineering.

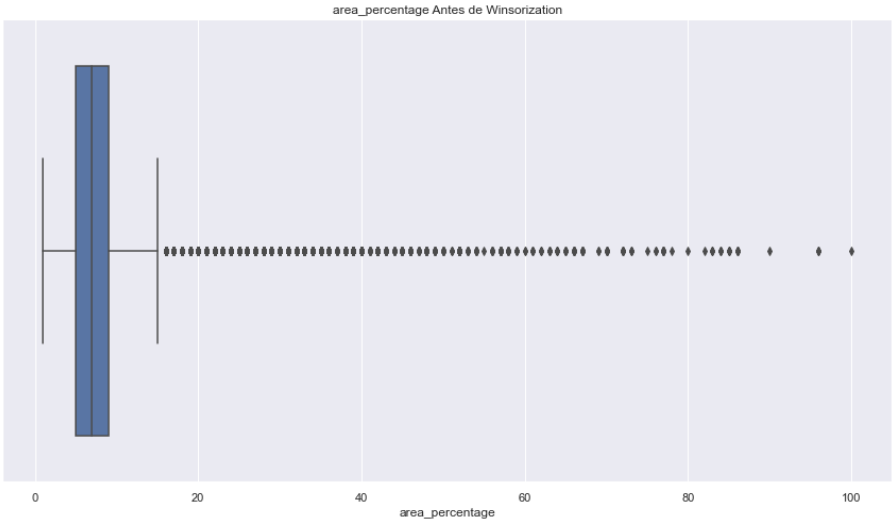
Una vez que ya se procesó el set de entrenamiento y se creó el data frame deseado, se pasó a probar los distintos algoritmos de Machine Learning que aparecen en el índice, en la sección de Modelos Utilizados. Para poder obtener lo mejor de estos algoritmos se usaron algoritmos para hacer una optimización de hiperparametros.

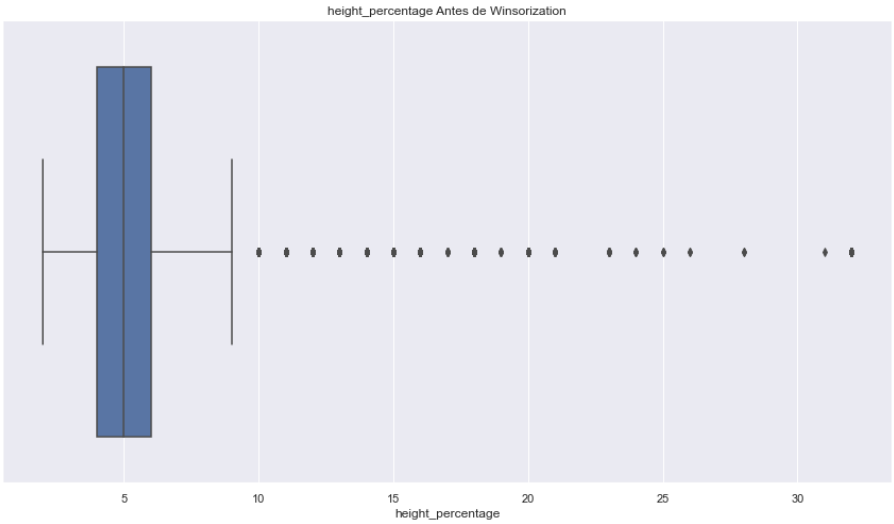
Dentro de este informe y del repositorio provisto en la primer carilla van a poder observar cómo fue que hicimos todo lo anterior mencionado.

3. Preprocesamiento

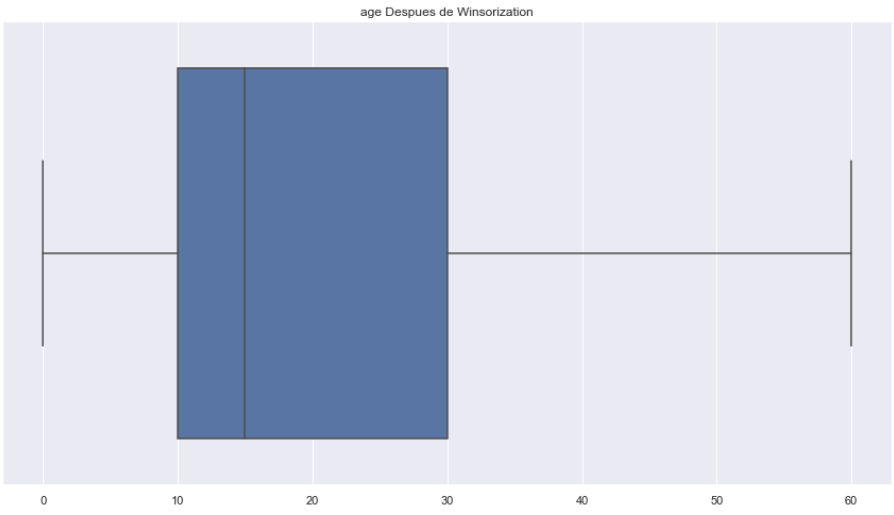
Para el preprocesamiento de los datos guiandonos por el analisis exploratorio hecho en el TP1 sabiamos que tanto el area_percentage, height_percentage y age tenian valores con ruido como se puede observar a continuacion.

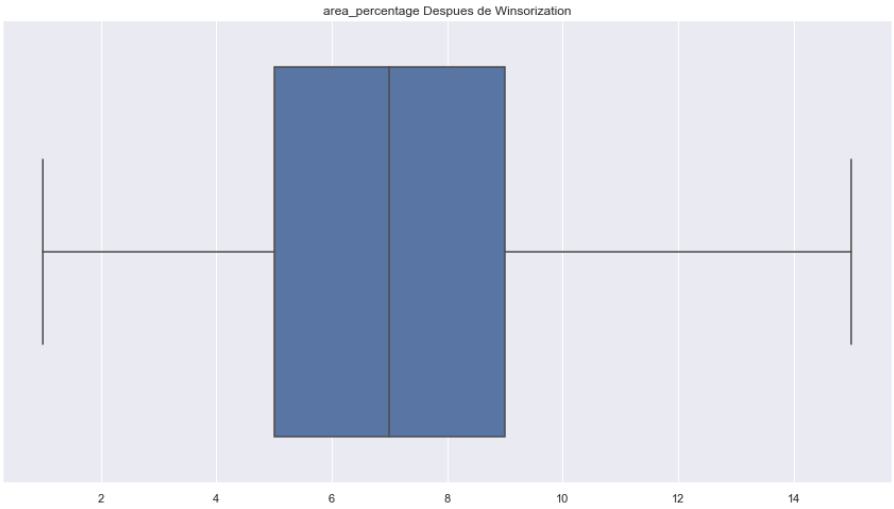


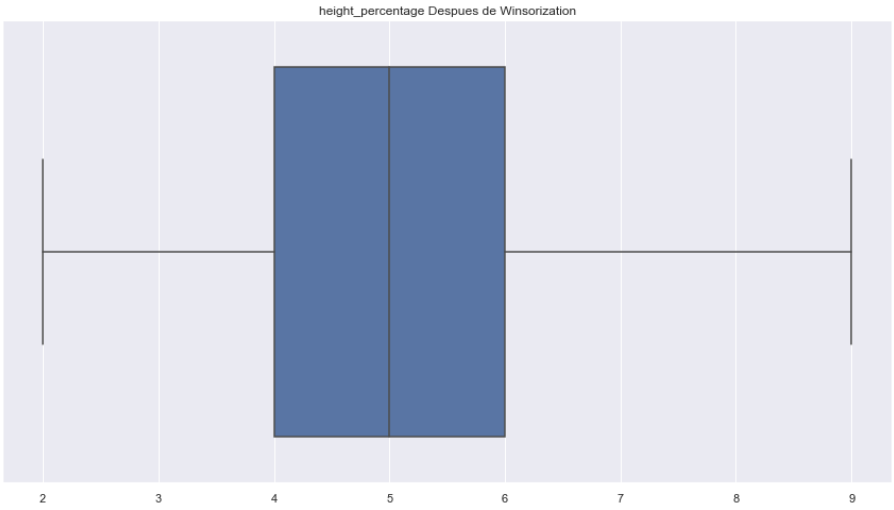




Por esta razon se decidio aplicar la funcion winsorize que es una forma de minimizar la influencia de valores atípicos en los datos al asignar al valor atípico un peso menor y cambiar el valor para que esté cerca de otros valores del conjunto y de esta manera los puntos de datos se modifican y no se eliminan. De esta forma los datos nos quedan de esta manera




area_percentage Despues de Winsorization
2
4
6
8
10
12
14


height_percentage Despues de Winsorization
2
3
4
5
6
7
8
9

4. Feature Engineering

4.1. Nuevos Features Implementados

4.1.1. volumen_percentage

Para la creacion de este nuevo feature utilizamos tanto el área y la altura multiplicadas para obtener el volumen normalizado de cada edificacion en nuestro set de datos ya que entendiamos que tanto la altura como el area es un factor determinante a la hora de predecir el daño ocasionado por el terremoto en la edficicacion.

4.1.2. use_mud

Con lo visto en nuestro analisis exploratorio habiamos observado que habia mucha correlacion entre una edificacion que era construida con barro y el grado de daño.
Por lo que se nos ocurrió agregar un feature que indique si la edificación fue construida con barro independientemente del otro material que la acompaña. Este nuevo feature nos agrego un mayor score a nuestro modelo

4.1.3. var_geo_level_1_id

Lo que hicimos en este feature fue calcular la dispersion que tenia cada geo level de la media total haciendo para cada uno de los datos $(x - media)^2$. Este nuevo feature mejoro nuestro score y al realizar el feature importance vimos que este feature nos ayuda a mejorar el modelo.

4.1.4. use_brick

Para este feature lo que queremos obtener es que edificios que usan ladrillos independientemente del mortero utilizado.

4.1.5. use_cement

Viendo que el feature 'use mud' nos había funcionado intentamos conseguir el mismo resultado con este feature que indica si una edificacion utiliza cemento independientemente con que material es acompañado, aunque ocn este nuevo no conseguimos la misma importancia pero aún así vimos que el modelo mejoró en pequeña medida por lo que se decidió dejarlo incorporado.

4.1.6. mortar_count

Este nuevo feature nos muestra lo tipos de morteros utilizados en una misma construccion. Se logro llegar a este feature sumando los tipos de morteros, en el featurre importance se observo que este no logra tener un impacto significativo pero aun asi logra que el modelo lo tenga en cuenta por esa razon se decidio dejarlo para el modelo final.

4.1.7. wood_count

En este feature lo hicimos fue sumar dos features que indican qué tipo de madera se utilizó en la construcción al sumarla podemos observar cuántos tipos de madera se habían utilizado en esa edificación

4.1.8. use_stone

Para este feature lo que queremos obtener es que edificios que usan piedra independientemente del mortero utilizado.

4.1.9. superstructure_count

Siguiendo la misma idea que veiamos utilizando se nos ocurrió sumar todos los features que indican la variedad de materiales con las que fue construido la edificación.

4.1.10. secondary_use

Al ver que la mayoría de usos secundarios que tenía las edificaciones tenía muy poca correlación con el target lo que hicimos fue unificar todos los usos secundarios en un solo feature para ver si podíamos conseguir que el modelo le dé más importancia a este nuevo feature, el objetivo se cumplio pero no singifico un aumento sustancial en el score final.

4.1.11. family_per_floor

Al tener como dato la cantidad de familias que vivian en cada edificio creamos este feature que nos indica cuantas familias vivian por piso, este feature solo nos agregaria complejidad a nuestro modelo ya que no mejoraba la prediccion de nuestro modelo.

4.1.12. 2_clusters 4_clusters 10_clusters

En estos 3 features lo que se implemento fue clustering con K means, se usaron diferentes parametros para cada uno de los features

4.1.13. Features Dropeadas

Muchos de los features que no estan entre los utilizados fueron descartados porque durante nuestro analisis vimos que no aportaban realmente nada al modelo e incluso ralentizaban al mismo al momento del entrenamiento, Otros tampoco tenían sentido lógico como para influir en el mismo.

4.2. Codificaciones para variables categoricas

4.2.1. One Hot Encoding

La primera idea (y la más básica) de representar nuestras features categoricas era usando One Hot encoding, esto implica crear una columna para cada tipo de keyword. Es decir a cada categoría se asigna con una variable binaria que contiene 0 o 1. Aquí, 0 representa la ausencia y 1 representa la presencia de esa categoría.

4.2.2. Dummy Encoding

El esquema de codificación Dummy es similar a la codificación one-hot. Este método de codificación de datos categóricos transforma la variable categórica en un conjunto de variables binarias (también conocidas como variables ficticias). En el caso de la codificación one-hot, para N categorías en una variable, utiliza N variables binarias. La codificación Dummy es una pequeña mejora con respecto a la codificación one-hot. La codificación Dummy utiliza características N-1 para representar N etiquetas / categorías. Este algoritmo fue el que usamos casi en la totalidad del tp y con el que nos sentimos mas comodos ya que nos dio buenos resultados.

4.2.3. Binary Encoding

Binary encoding es un tipo de encoding para labels donde tomamos cada label como un número y luego representamos al número en binario, usando una feature nueva para cada dígito. Esta feature nos trae los beneficios de One Hot encoding sin el aumento drástico de dimensiones que tiene ese encoding.

4.2.4. Mean Encoding

Mean encoding es un tipo de encoding que implica hacer una codificación de los labels utilizando el target. Por ejemplo, el promedio de los labels para cada valor posible. Aunque es algo polémico ya que se puede filtrar informacion de los labels a los features de entrenamiento.

5. Modelos Utilizados

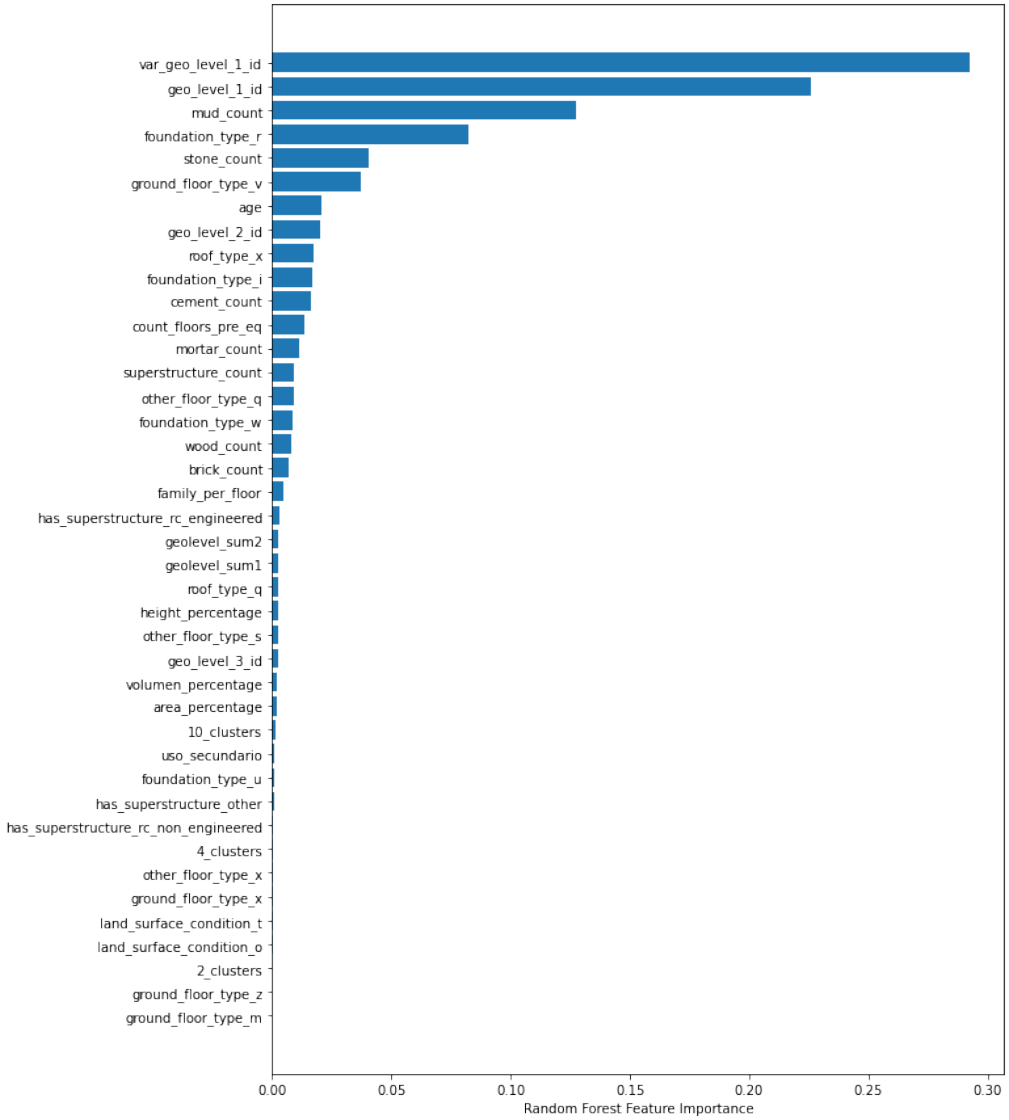
5.1. Random Forest

El primer modelo basado en árboles que probamos fue el algoritmo de Random Forests. Se supo de antemano que dicho algoritmo no iba a arrojar mejores resultados que los árboles basados en boosting pero igualmente se probó, ademas, debido a esto no sentimos necesario buscar sus hiperparametros optimos ademas de que solo tiene 2 significativamente importantes los cuales son la cantidad de árboles a crear y la cantidad de atributos de cada uno.

Los hiperparametros que han sido utilizados son los siguientes:

- **n_estimators:** La cantidad de árboles a construir.
- **max_features:** La cantidad de features a considerar cuando se busca el mejor split.
- **max_depth:** La maxima profundidad de cada arbol.

Una de las caracteristicas mas importantes y utiles de los Random Forest es su metodo de "Feature Importance.[el] cual nos indica cuales features son las mas relevantes e influyentes en nuestro modelo.



Como se puede apreciar este modelo esta fuertemente influenciado por los features de "geo_level_1_id" su correspondiente varianza, seguido de la columna "mud_count".

5.2. Boosting

La idea central de usar algoritmos de boosting yace en usar árboles simples(profundidad baja), pero usar varios de estos para que se vayan generando nuevos árboles que predigan mejor lo que los anteriores predijeron mal, de tal modo que se puedan complementar.
El resultado final de la predicción tendrá en cuenta todos los árboles creados, siendo la predicción final la sumatoria de las predicciones de cada árbol analizado de forma individual.

5.2.1. XGBoost

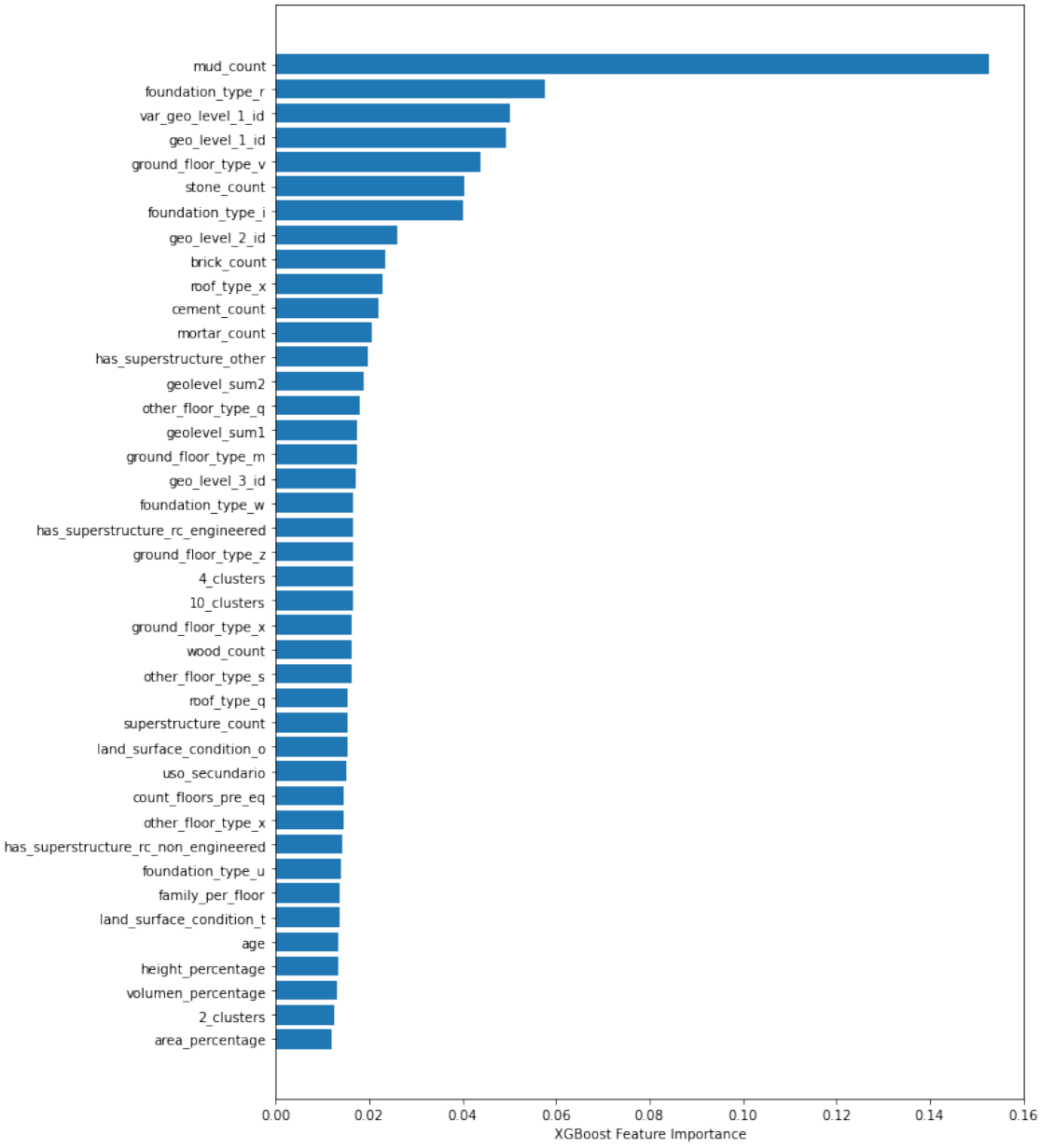
El primer árbol basado en boosting que usamos fue XGBoost. Algo muy destacable que pudimos ver de este algoritmo fue su gran cantidad de hiperparametros y la notable diferencia respecto a los demas que se puede encontrar al usar los hiperparametros "óptimos" frentea los default.

Los resultados obtenidos al usar este modelo fueron los más altos de todos los árboles de boosting probados.

Los hiper parametros que hemos estado tuneando son los siguientes:

- **n_estimators:** La cantidad de árboles a construir.
- **learning_rate:** Representa la tasa de aprendizaje y reduce el peso de los features para evitar overfitting y hacer el proceso de boosting más conservativo.
- **max_depth:** La maxima profundidad del arbol.
- **colsample_bytree:** Porcentaje de features usadas para cada árbol (valores muy alto, posible overfitting).
- **alpha:** Regularización para los pesos de las hojas. Un valor más alto genera una mayor regularización..
- **subsample:** Porcentaje de muestras usadas para cada árbol.
- **objective:** Función de error a utilizar (algunas: reg:squarederror para regresión, reg:logistic o binary:logistic para clasificación).

Los Features mas importantes para este modelo fueron:



Como se puede observar este modelo fue mas equitativo al momento de utilizar los distintos features, y a diferencia del Random Forest fue altamente influenciado por el feature "mud_count".

5.2.2. LightGBM

LightGBM fue otro algoritmo basado en arboles, el cual a diferencia de XGBoost, puede procesar variables categóricas sin necesidad de usar algún tipo de encoding previamente(ya que internamente lo puede calcular). Una característica importante que se remarcó en clase de este modelo es que es mucho más rápido que los otros algoritmos de boosting, por lo que fuera de las competencias se puede ver con mayor intensidad esta ventaja.

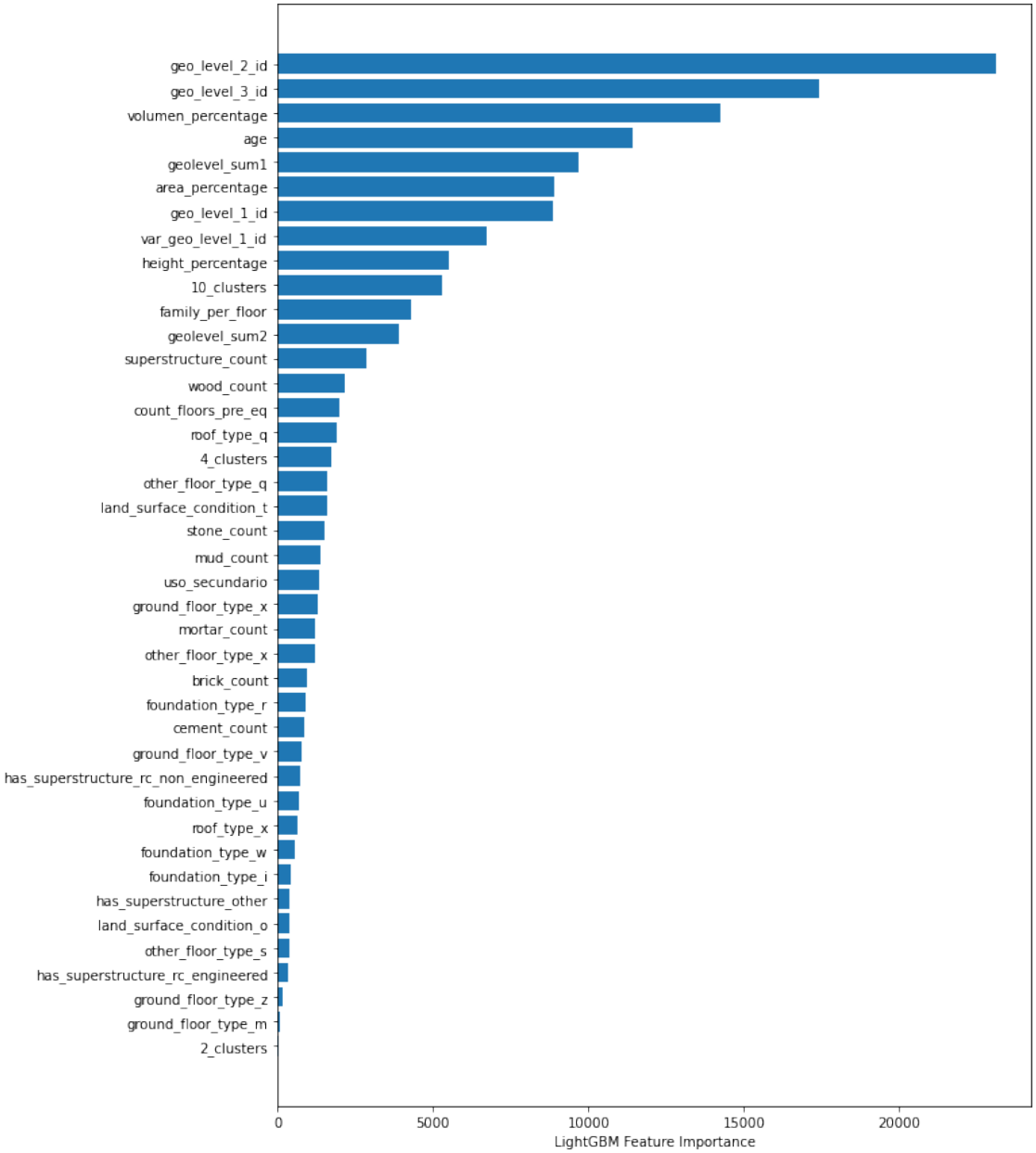
Los resultados obtenidos al usar este modelo fueron casi los más altos de todos los árboles de boosting probados solo por debajo de XGBoost.

Los hiper parametros que hemos estado tuneando son los siguientes:

- **n_estimators:** La cantidad de árboles a construir.

- **learning_rate:** Representa la tasa de aprendizaje y reduce el peso de los features para evitar overfitting y hacer el proceso de boosting más conservativo.
- **max_depth:** La maxima profundidad del arbol.
- **num_leaves:** Cantidad máxima de hojas en un árbol
- **objective:** Función de error a utilizar (regression, binary, multiclass, etc).

Los Features mas importantes para este modelo fueron:



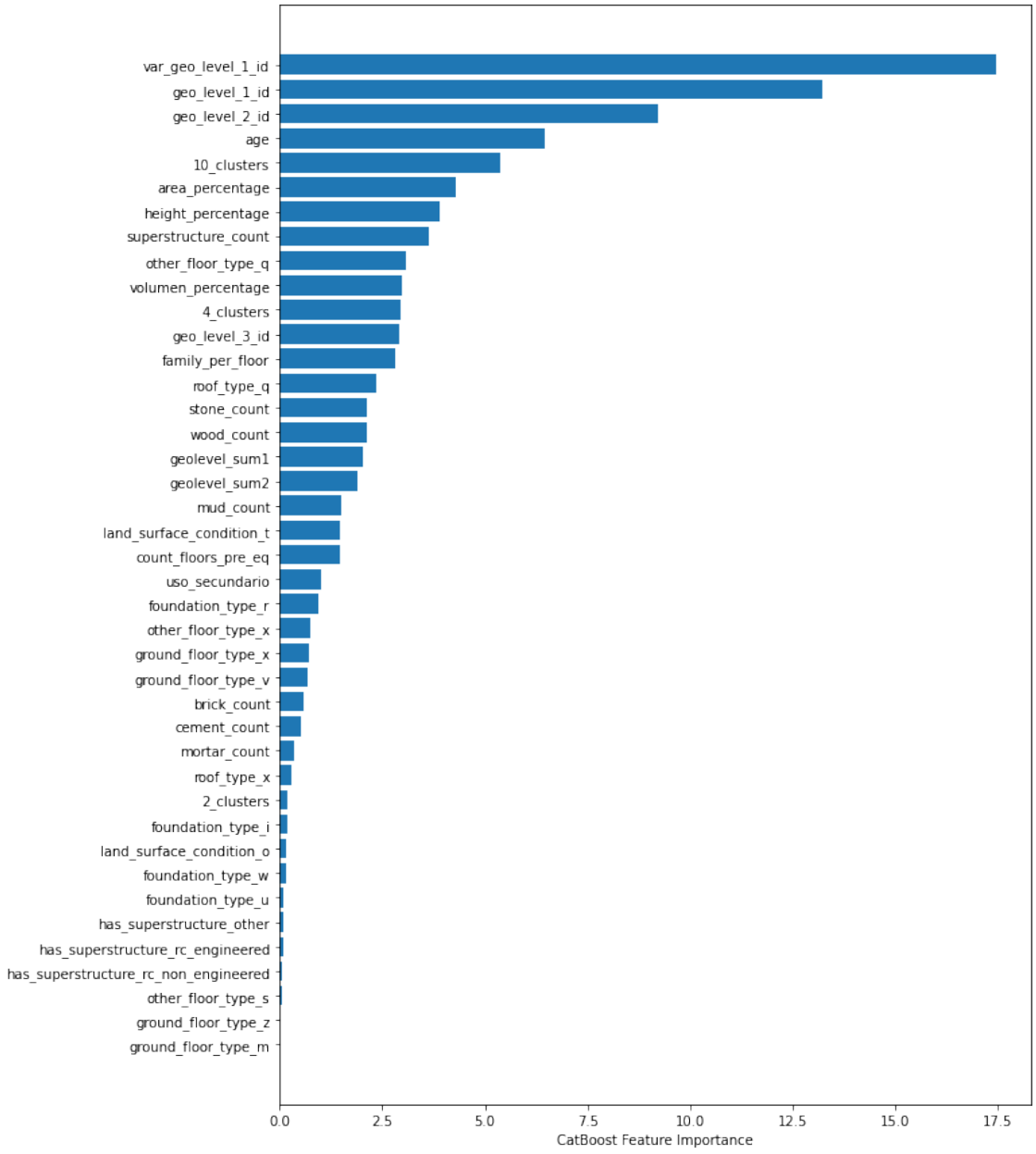
Como se puede apreciar este modelo esta fuertemente influenciado por los features de "geo_level_2_idz "geo_level_3_id.ª diferencia del resto que eran mas influenciados por el nivel 1, ademas destacan la edad y el volumen.

5.2.3. CatBoost

El último algoritmo basado en Boosting que probamos fue CatBoost. Nos resulto el modelo de boosting con peor puntuacion, sin contar a Random Forest. Los hiper parametros que hemos estado tuneando son los siguientes:

- **n_estimators:** La cantidad de árboles a construir.
- **learning_rate:** Representa la tasa de aprendizaje y reduce el peso de los features para evitar overfitting y hacer el proceso de boosting más conservativo.
- **max_depth:** La maxima profundidad del arbol.
- **loss_function:** La metrica a utilizar en el entrenamiento.
- **l2_leaf_reg:** Es el coeficiente en el término de regularización L2 de la función de costo.

Los Features mas importantes para este modelo fueron:



Este modelo similarmente a Random Forest esta fuertemente influenciado por los features de "geo_level_1_idz su varianza pero seguido por la edad del edificio en vez de mud_count, dandole menos importancia a este ultimo en contraste.

6. Busqueda De Hiper Parámetros

6.1. Grid Search Cross-Validation

Un tipo de búsqueda que usamos a la hora de encontrarlos mejores hiperparametros fue Grid Search con Cross Validation, el cual calcula todas las combinaciones posibles y determina cuales son los hiperparametros que generan el mejor resultado para un tipo de score. Este algoritmo fue el que usamos casi en la totalidad del tp y con el que nos sentimos mas comodos.

6.2. Random Search

Este algoritmo presenta una gran ventaja frente al anterior, ya que, dados unos hiperparametros uno puede elegir la cantidad de tiempo a utilizar en la búsqueda de estos. Esto es sumamente útil ya que nos permite utilizar una mayor cantidad de parámetros a la hora de la búsqueda sin necesidad de esperar mucho tiempo.

6.3. Bayesian Search

Por último, tenemos la búsqueda de parámetros de forma bayesiana. Este algoritmo presenta una búsqueda de manera que a cada resultado le asigna un "peso" De esta forma, para el próximo paso, se samplean los resultados con sus respectivos hiperparametros y se prueban valores cercanos a los hiperparametros que mejor resultado obtuvieron, es decir, aquellos que tienen más peso.

7. Algoritmos y Modelo Utilizado

Inicialmente en nuestro modelo, luego de realizar el preprocesamiento de los datos, se comenzó con el Feature Engineering, la cual fue la parte mas exhaustiva del trabajo en donde añadimos nuevas features, dropeamos algunas que no aportaban realmente nada al modelo e incluso ralentizaban al mismo al momento del entrenamiento, sin mencionar que algunas no tenían sentido lógico alguno para el mismo.

Como método de encoding para las variables categóricas se utilizó "Dummy Encoding", ya que fue el que usamos casi en la totalidad del TP y con el que nos sentimos más cómodos ya que nos dio buenos resultados como se menciono anteriormente en el informe.

Luego, una ves finalizado el proceso de Feature Engineering se procedió a probar cada uno de los modelos descriptos en el informe, Random Forest, XGBoost, LightGBM y CatBoost.

Siendo el que nos proveyó mejor score XGBoost, seguido por LightGBM y CatBoost quedando Random Forest en último lugar.

Para la optimización de hiper parámetros utilizamos Grid Search con Cross Validation, ya que también fue el que usamos casi en la totalidad del TP y con el que nos sentimos más cómodos al momento de realizar los test.

Cabe destacar que gracias a Google Colab y su función de utilizar una TPU de Google o GPU de Nvidia reducimos drásticamente el tiempo de entrenamiento, permitiéndonos asi usar Grid Search cómodamente.

Los Hiper Parámetros que nos resultaron óptimos para cada uno de los modelos fueron los siguientes:

Para el modelo de XGBoost:

Un max_depth de 10, learning_rate de 0.1, n_estimators=600, objective='reg:logistic', subsample=0.6, alpha=0 y colsample_bytree=0.75.

Dándonos un score aproximado de 0.7469 en los mejores de los casos.

Para el modelo de LightGBM:

Un learning_rate de 0.1, max_depth igual a 12, n_estimators=250, num_leaves= 200, y para objective usamos binary.

Dandonos un score aprox de 0.743 en el mejor de los casos el cual es un poco inferior al de XGBoost pero no muy por detrás, aun mas considerando la velocidad de este modelo para entrenar.

Para el modelo de CatBoost:

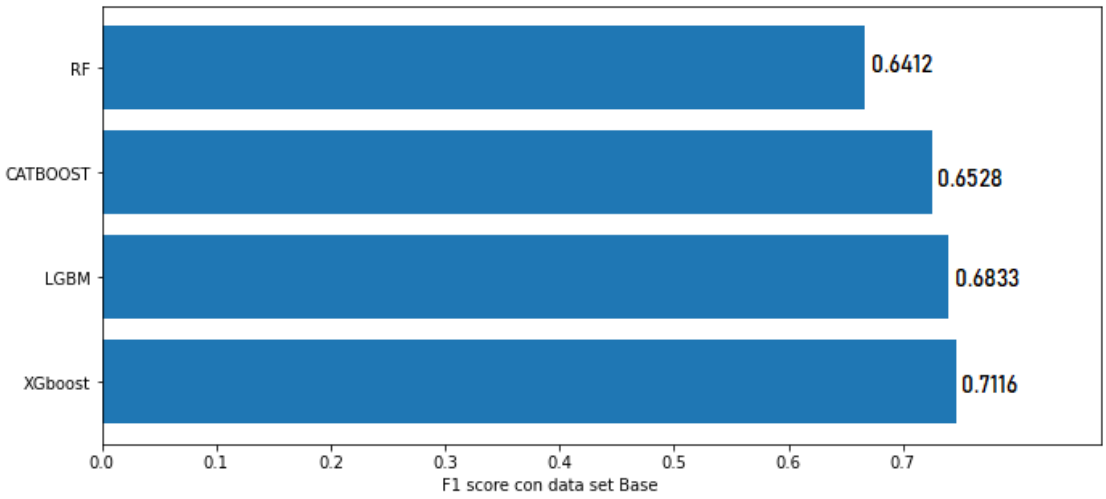
l2_leaf_reg=8, learning_rate=0.1, loss_function=MultiClass, max_depth=12, n_estimators=600.

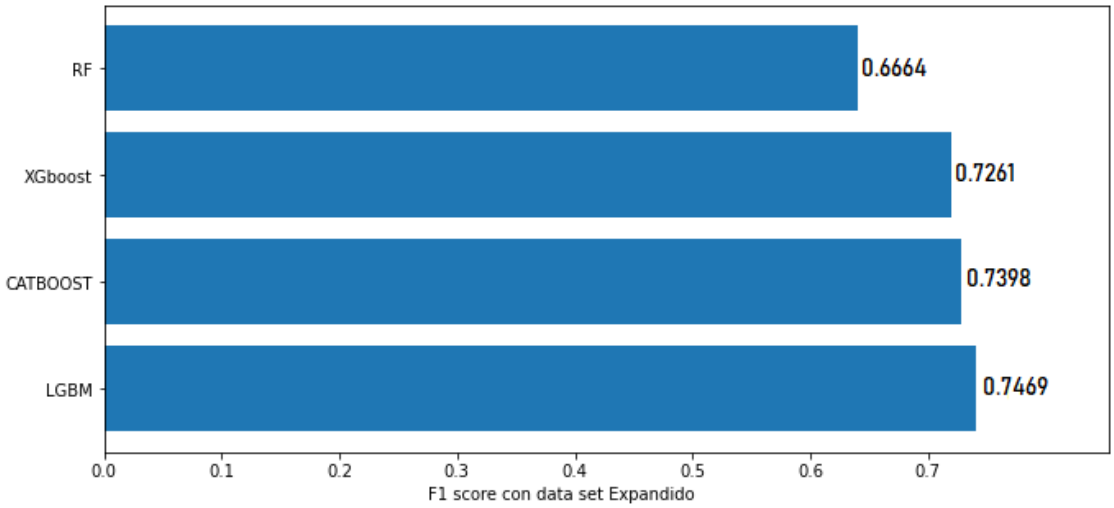
Esto dándonos un score de 0.7261 siendo el que mas bajo score nos dio de los 3 modelos de boosting.

Finalmente, para el modelo de Random Forest, como mencionamos anteriormente, no creimos que valga la pena perder tiempo en entrenar hiper parametros por lo cual utilizamos valores de n_estimators igual a 150, max_features a 13 y max_depth de 7. Con lo que obtuvimos un score de 0.666 en el mejor caso.

A continuacion se muestra muy resumidamente el progreso de nuestro score ya que inicialmente no estuvimos haciendo tracking de los mismos. Uno utilizado el dataset base y otro utilizando el

dataset expandido con los hiperparametros optimizados.





8. Conclusiones

En base a todo lo expuesto anteriormente, primero, antes que cualquier otra conclusión, aprendimos que los resultados que refleja un F1 score en Jupyter Notebook puede estar muy alejado a lo que verdaderamente se obtiene en DrivenData. Esto es así debido al error de entrenamiento existente al no usar el set de test real, por más que se reduzca este error haciendo un K Cross-Validation.

Por otro lado, analizando más los aspectos técnicos, se puede observar como varía mucho el resultado de algunos modelos en base a sus features. Esto da a entender la importancia de implementar una buena Feature Engineering.

Trabajos como este, en nuestra opinión, denotan la importancia de los algoritmos de machine learning. Un sector el cual esta creciendo de forma importante. Tan altos resultados de precisión ayudaran y ayudan hoy en día, en muchos ámbitos de la vida cotidiana.

Detectores anti-spam, reconocimiento de voz, reconocimiento facial, vehiculos que se autopilotan y evitan accidentes que para la reaccion humana seria dificil. Estas son solo algunas aplicaciones de las tantas como modelos medicinales, de seguridad y muchos más que nos van e iran mejorando la vida poco a poco.

Finalmente dejamos el enlace a nuestro repositorio de github donde estaran subidos los archivos que hemos utilizado.

https://github.com/IvanConde/TP2-Organizacion-De-Datos