

75.06 Organización de datos

Trabajo Práctico 2: Análisis exploratorio de datos



Grupo: DataMasters

Nombre	Padrón	Mail
Ezequiel Vilardo	104980	ezequielvilardo@gmail.com
Santos Emanuel Amaya	96891	samaya@fi.uba.ar
Rogger Aldair Paredes Tavera	97976	rpardest@fi.uba.ar

Github:

<https://github.com/EzequielVF/7506-Datos-TP2>

1. Introduccion	3
2. Organización del Trabajo	3
2.1. Investigación Previa	3
2.2. Creación de dataframes	3
2.3. Parameter tuning	3
2.4. Modelos	3
2.4.1. RandomForest	4
2.4.2. CatBoost	4
2.4.3. LightGbm	4
2.5. Modelo Principal	4
2.5.1. XgBoost	4
3. Features	4
3.1. Creación de dataframes	4
3.2. Feature engineering	5
4. Modelos	6
4.1. Algoritmos de clasificación utilizados	6
4.1.1 RandomForest	6
4.1.2 XGBoost	6
4.1.3 CatBoost	6
4.1.4 LightGBM	6
4.2. Parameter tuning	7
5. Resultados	7
6. Conclusión	8

1. Introduccion

El objetivo del segundo trabajo práctico es resolver un problema de machine learning con los datos obtenidos del análisis exploratorio previo que se hizo sobre el mismo trabajo práctico.

Puntualmente, **se busca predecir la variable 'damage_grade'**, que representa el nivel de daño recibido por la edificación a causa del desastre ocurrido, a través de un modelo predictivo de Machine Learning.

Para crear dicho modelo predictivo, se usaron combinaciones de algoritmos basados en Machine Learning principalmente diagramas de árboles, sumados a eso también los conocimientos adquiridos previos y durante el desarrollo del trabajo.

2. Organización del Trabajo

Para un mejor desarrollo del trabajo se decidió dividir en distintos archivos .py, que son usados en común por los distintos modelos que se fueron creando en este trabajo práctico.

2.1. Investigación Previa

<https://github.com/EzequielVF/7506-Datos-TP1/blob/main/analisis.ipynb> en este documento se encuentra el análisis exploratorio previo que se hizo sobre el dataSet.

2.2. Creación de dataframes

https://github.com/EzequielVF/7506-Datos-TP2/blob/main/preparo_datos.ipynb en este documento se encuentran el código que se usaron para la creación de los distintos CSV que cada modelo usara en común o individualmente en su propio documento.

2.3. Parameter tuning

https://github.com/EzequielVF/7506-Datos-TP2/blob/main/Parameter_tuning.ipynb en este notebook se hacen diversas pruebas sobre cada modelo hasta encontrar los hiper-parámetros óptimos de cada uno.

2.4. Modelos

Con el objetivo de que puedan ver el desarrollo de cada modelo, en esta sección se encontrarán los enlaces a los distintos notebooks con sus respectivos modelos que se fueron creando a lo largo del trabajo práctico:

2.4.1. RandomForest

<https://github.com/EzequielVF/7506-Datos-TP2/blob/main/RandomForest.ipynb>

2.4.2. CatBoost

<https://github.com/EzequielVF/7506-Datos-TP2/blob/main/CatBoost.ipynb>

2.4.3. LightGbm

<https://github.com/EzequielVF/7506-Datos-TP2/blob/main/LightGBM.ipynb>

2.5. Modelo Principal

2.5.1. XgBoost

<https://github.com/EzequielVF/7506-Datos-TP2/blob/main/xgBoost.ipynb>

3. Features

Como primer avance en el trabajo, se decidió analizar los resultados obtenidos en la investigación previa que se realizó en el mismo Set de Datos. El TP1 es un análisis exploratorio, hecho sobre un conjunto de datos posteriores a un desastre natural (Terremoto), ubicado en Nepal, que contiene información valiosa sobre los impactos de los terremotos, las condiciones de los hogares y las estadísticas socioeconómicas y demográficas.

Podemos deducir ideas parecidas, el análisis hecho en búsqueda de patrones, ideas y conceptos pueden ser aplicados en este trabajo. En particular, se buscan atributos escondidos en el set original que puedan ser codificados de tal forma que luego los algoritmos de Machine Learning puedan utilizar a su favor. Los atributos encontrados son especificados en la sección de Feature Engineering.

3.1. Creación de dataframes

Como parte del feature engineering, se realizó la creación de dos datasets nuevos:

- `datos_procesados.csv`: Contiene el dataset con las columnas categóricas transformadas con OneHotEncoding.

- **Test_values_procesados.csv:** Contiene los datos de test values transformados con OneHotEncoding.
- **Create_features_train.csv:** Contiene nuevos feature nuevos que se fueron obteniendo con la información preexistentes del train_set.csv.que contendrán la información necesaria que será usada para el entrenamiento de los distintos modelos que se crearon.

3.2. Feature engineering

Para un mejor análisis en cada modelo se decidió por comodidad, ya que algunos modelos de árboles eran incapaces de interpretar columnas categóricas como el caso LightGBM - XgBoost, aplicar el concepto de OneHotEncoding usando el método de `get_dummies()` para obtener una columna booleana por cada valor categórico que contenga la columna.

También se realizó la creación de columnas

- **epicentro:** Esta es una columna que se obtuvo del análisis exploratorio previo en el dataset, donde se encontró como epicentro las regiones 17 - 18 del `geo_label_id_1`.
- **Materiales_feos:** Esta columna contendrá la cantidad de materiales, que fueron entendidas como las peores a usar en una construcción, que se usaron en la estructura.
- **Materials_goods:** Esta columna contendrá la cantidad de materiales, que fueron entendidas como las mejores a usar en una construcción, que se usaron en la estructura.
- **Volume_percentage:** Esta columna se creó con la finalidad de ver cómo podría ayudar el agregar el volumen de porcentaje del piso.
- **Height_percentage_per_floor:** Esta columna se creó para ver el porcentaje de la altura por piso.
- También lo que se hizo fue ver cómo era la distribución de datos categóricos, comparando la cantidad de datos entre ellos, para poder ver la cantidad de ruido que su propia distribución podría generar.

- Modificación columna '*building_id*', explicado más adelante en '*resultado*'.
- Aplicando Feature Importance con los distintos modelos desarrollados en el trabajo práctico, si intento remover columnas que supuestamente no tenían mucha importancia para el modelo, pero aun removiendo esas columnas no se pudo mejorar la predicción de los modelos.



4. Modelos

4.1. Algoritmos de clasificación utilizados

Para cada uno de estos modelos se desestimó la idea de usar modelos Regresores pues buscábamos encontrar predicciones exactas de los labels, y es por eso que usamos solo modelos Classifiers.

4.1.1 RandomForest

Este algoritmo fue el primero que decidimos desarrollar directamente, para ir probando qué tanto podríamos optimizar los resultados obtenidos de este modelos, usando los modelos posteriores, puesto que se considera que los otros 3 modelos desarrollados principalmente se pueden obtener mejores resultados. Para la búsqueda de hiperparametros comenzamos con Random y terminamos con una búsqueda grande con Grid, ya que el proceso no era muy demandante en cuanto a tiempo.

0.7110	EzequielV 	2021-07-09 02:52:35 UTC 
--------	---	---

4.1.2 XGBoost

Este algoritmo basado en gradient boosting, fue el que nos dio los mejores resultados en comparación con los otros modelos desarrollados, para obtener los mejores hiper-parámetros se hizo análisis de distintos posibles valores que el modelo pudiera entrenar. se usó el método de Random Search combinado con cross-validation.

0.7515	EzequielV 	2021-07-24 19:34:05 UTC
--------	---	-------------------------

También se probó cambiar el hyperparameter 'booster' por defecto: gbtree por gblinear y dart que afecta en cómo el modelo se "organiza" pero no lograron mejorar el score final.

Un detalle más de este modelo será mencionado más adelante en *Resultados*.

4.1.3 CatBoost

Después de probar gradient boosting, optamos por usar más modelos de esa índole. Cat Boost es el último algoritmo de boosting probado y sus resultados no fueron tan buenos como para justificar el tiempo perdido en cada ejecución.

La mejora más notable se consiguió al subir el número de iteraciones.

0.7247

EzequielV 

2021-07-08 18:12:29 UTC

4.1.4 LightGBM

LightGBM es un algoritmo que principalmente destacó en la rapidez de obtener los mejores HiperParametros, ya que en comparación con los otros modelos era más rápido entrenar el modelo, esto es principalmente a su poco consumo de memoria y su gran manejo de dimensionalidad de datos.

0.7465

EzequielV 

2021-07-26 18:03:12 UTC

4.1.5 Ensamblés

Con el objetivo de mejorar los resultados obtenidos de los modelos, se decidió combinar algunos de los algoritmos, empleando métodos de ensamble como Bagging Classifier y Voting Classifier, con los cuales pese a las expectativas no se pudo mejorar el score obtenido por el modelo individual de XGBoost.

0.7490

EzequielV 

2021-07-28 20:32:52 UTC

(voting classifier de 2 XGBoost (el de los mejores hiperparametros) + CatBoost)

0.7495

EzequielV 

2021-07-22 17:37:32 UTC

(bagging de 20 repeticiones de XGBoost con los mejores hiperparametros)

4.2. Parameter tuning

En los primeros modelos corridos fue cuando se empezó a notar lo que ya se sabía: los hiper-parámetros son sencillamente lo más importante de cada algoritmo, y

la diferencia entre un buen modelo y uno promedio o incluso malo. Inicialmente, debido a la gran cantidad de opciones para algunos algoritmos, optamos por un método greedy, que consistía en un pequeño framework donde se van probando distintos hiper-parámetros con distintos valores progresivamente.

Dependiendo del modelo ya que unos consumen más tiempo que otros usamos:

- Random Search combinado con K Fold (cross-validation)
 - De esta forma se combinaban de forma aleatoria los posibles hiper-parámetros y sobre ello aplicamos el cross-validation para contar con resultados más precisos.
- GridSearch
 - Aca se probaban todas las posibilidades porque lo que era un algoritmo de búsqueda muy costoso en cuanto a tiempo por lo que terminamos usando en su mayoría la otra alternativa.
 - A la hora de ejecutarlo probamos usarlo sobre pequeñas secciones de posibilidades.

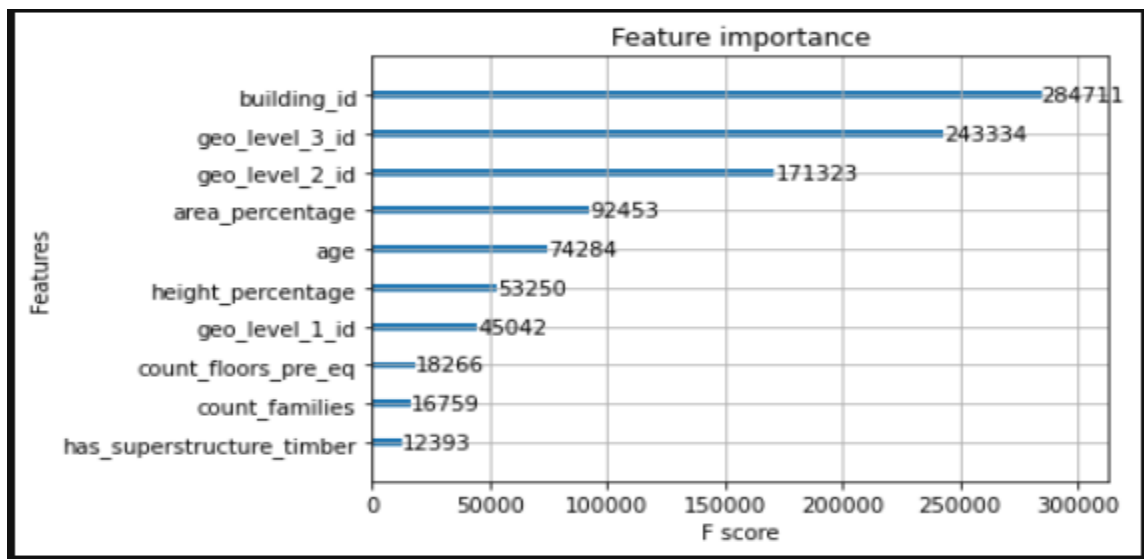
5. Resultados

A partir de los distintos submits realizados se deduce como fue mencionado previamente que XGBoost es el algoritmo que arrojó mejores resultados, sin embargo vimos un estancamiento en el puntaje de 0.7475.

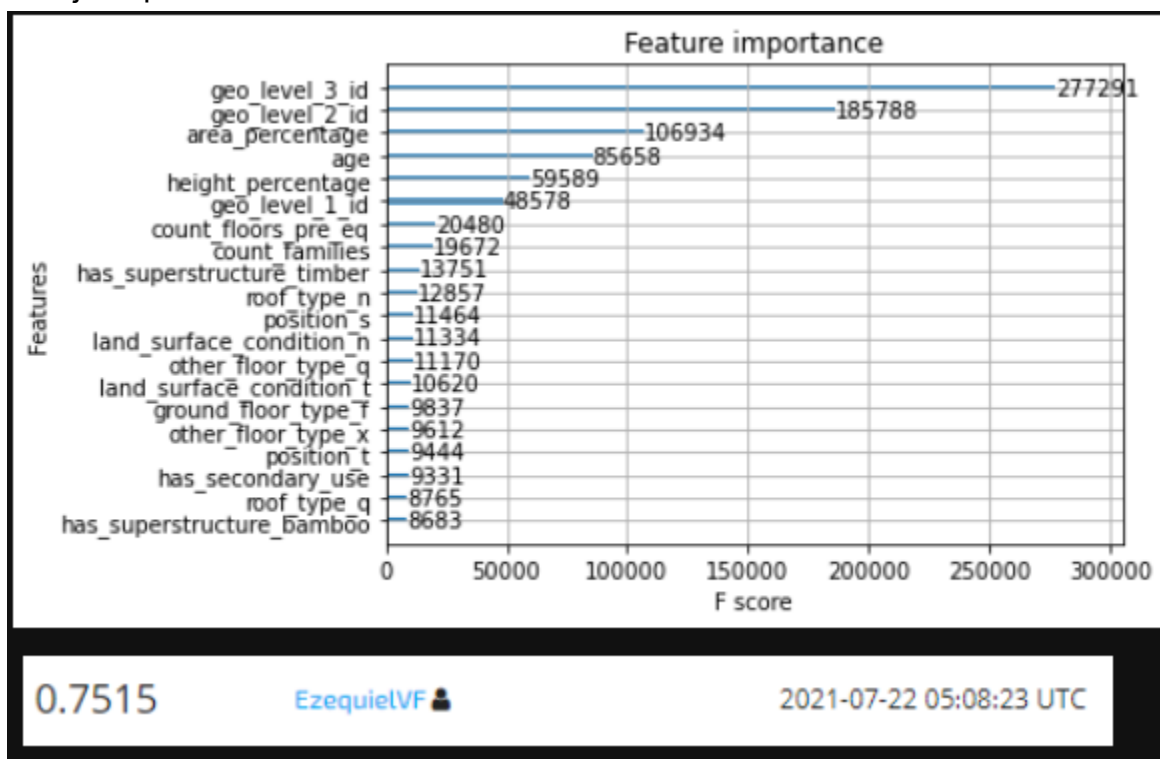
0.7475

EzequielVF 🧑

2021-07-19 20:41:28 UTC



Observamos el `plot_importance` del modelo y descubrimos que la columna con más importancia era “*building_id*”, una columna de ids únicos usada como identificación y que no tendría sentido que fuera importante a la hora de realizar predicciones. Pudimos resolverlo poniendo los “*building_id*” del set de training en cero, ya que no encontramos una forma que el modelo la ignorara cuando entrenamos y si la eliminamos luego no dejaba predecir.



Con este cambio el score aumentó hasta 0.7515.

Como observación, las columnas creadas durante el feature engineering fueron probadas en este modelo tanto individualmente como en conjunto y no se logró mejorar el puntaje final conseguido, tampoco hubo mejora al intentar sacar del set de datos columnas de baja importancia.

(<https://github.com/EzequielVF/7506-Datos-TP2/blob/main/TestNewFeatures.ipynb>)

6. Conclusión

En tp1 se concluye que los daños estructurales provocados por el terremoto dependen de los materiales edificio, y si estaba en el “epicentro”, sin embargo los modelos predictivos de Machine Learning pudieron comprobar que las locaciones y otras columnas omitidas tienen más relevancia, ya que al ver la importancia que le

daban los modelos predictivos que se desarrollaron, nos pudimos dar cuenta que era más importante saber la ubicación de las distintas edificaciones, ya arrojaban una mejor predicción.

Además con lo visto en feature, se puede decir tiene de base buenos features para el modelo final ya que la eliminación de cualquier de ellos termina repercutiendo negativamente en la predicción.