FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico Nº2: Pipeline de Machine Learning

[75.06] Organización de Datos - FIUBA

Autores

Buceta, M. Belen	102121	bbuceta@fi.uba.ar
Companys, Gonzalo Alejo	103026	gcompanys@fi.uba.ar
Di como, Juan Pablo	102889	jpdicomo@fi.uba.ar
Jure, Federico	97598	fedejure@gmail.com

Grupo: Hakuna Ma-Data

https://github.com/jpdicomo/Tps-Datos/tree/main/TP2-Entrega
Cátedra: Argerich
Cuatrimestre: Primer Cuatrimestre 2021
Lenguaje elegido: Python

Índice

1. Introducción **3**

2. Objetivos **3**

3. Feature Engineering **4**
3.1. Análisis de los atributos . . . 4
3.1.1. Hipótesis . . . 4
3.1.2. Feature Importance . . . 5
3.1.3. Corroboración de la hipótesis . . . 5
3.2. Creacion de features . . . 6
3.2.1. Volumen . . . 6
3.2.2. Frecuencia para atributos categóricos . . . 6
3.2.3. Valores estadísticos por regiones . . . 7
3.3. Encodeo de los geo levels . . . 7
3.4. Outliers . . . 7
3.4.1. Detección . . . 7
3.4.2. Criterio para remover Ouliers . . . 7
3.4.3. Hipótesis . . . 8
3.4.4. Proceso de comprobación . . . 8

4. Prueba de modelos **10**
4.1. Introducción . . . 10
4.2. Búsqueda de hiperparámetros . . . 10
4.3. Entrenamiento . . . 10
4.4. Overfitting y Underfitting . . . 10

5. Red Neuronal **11**
5.1. Teorema de aproximación universal . . . 11
5.2. Hiperparámetros . . . 11
5.3. Resultados . . . 11
5.4. Conclusiones . . . 11

6. DecisionTree **12**
6.1. Hiperparámetros . . . 12
6.2. Resultados . . . 12
6.3. Conclusiones . . . 12

7. Random Forest **13**
7.1. Prueba utilizando la totalidad de los features . . . 13
7.1.1. Resultados . . . 13
7.1.2. Conclusiones . . . 13
7.2. Prueba utilizando una menor cantidad de features . . . 13
7.2.1. Resultados . . . 13
7.2.2. Conclusiones . . . 14
7.3. Utilizando los features catalogados como importantes . . . 14
7.3.1. Resultados . . . 14
7.3.2. Conclusión . . . 14
7.4. Prueba con importantes y ensamble . . . 14
7.4.1. Hipótesis . . . 14
7.4.2. Conclusión . . . 14
7.5. Prueba removiendo del dataset los outliers . . . 15
7.5.1. Resultados . . . 15

7.5.2. Conclusión de la prueba realizada 15

8. XGBoost **16**
8.1. Hiperparámetros 16
8.2. Pruebo haciendo uso de los features importantes 16
8.2.1. Resultados 16
8.2.2. Conclusiones 16
8.3. Prueba haciendo uso de los features importantes y geo_levels encodeados 16
8.3.1. Resultados 17
8.3.2. Conclusiones 17
8.4. Prueba con features creados y Geo levels encodeados 17
8.4.1. Resultados 17
8.4.2. Conclusión 17

9. Conclusiones Finales **18**

1. Introducción

El presente informe reúne la documentación correspondiente al segundo trabajo práctico de la materia Organización de Datos. El mismo consiste en el desarrollo de un modelo de Machine Learning que permita predecir el daño producido en una cierta edificación por una catástrofe natural.
Los sets de datos pueden obtenerse en:
https://www.drivendata.org/competitions/57/nepal-earthquake.

2. Objetivos

El trabajo tiene los siguientes objetivos:

- Realizar feature engineering, encontrando codificaciones que sirvan, generando nuevos atributos, analizando cuáles atributos aportan y cuáles no.
- Plantear hipótesis en conjunto con preguntas y responderlas con experimentación y resultados.
- Explorar modelos predictivos con el fin de encontrar cúal se adapta mejor al problema.
- Búsqueda de hiperparámetros óptimos de alguna manera automatizada.

3. Feature Engineering

3.1. Análisis de los atributos

3.1.1. Hipótesis

En base al análisis de correlación entre features realizado en el trabajo anterior. Inferimos de cúales podríamos valernos y de cúales no. Posteriormente, comprobamos la hipotésis en un modelo predictivo.

Cabe resaltar que las siguientes deducciones no son tomadas como válidas a priori sino que son supuestos a comprobar en los modelos que desarrollaremos más adelante.

Para mayor claridad en las explicaciones dadas, adjuntamos el gráfico de correlación entre features producido en el análisis exploratorio.

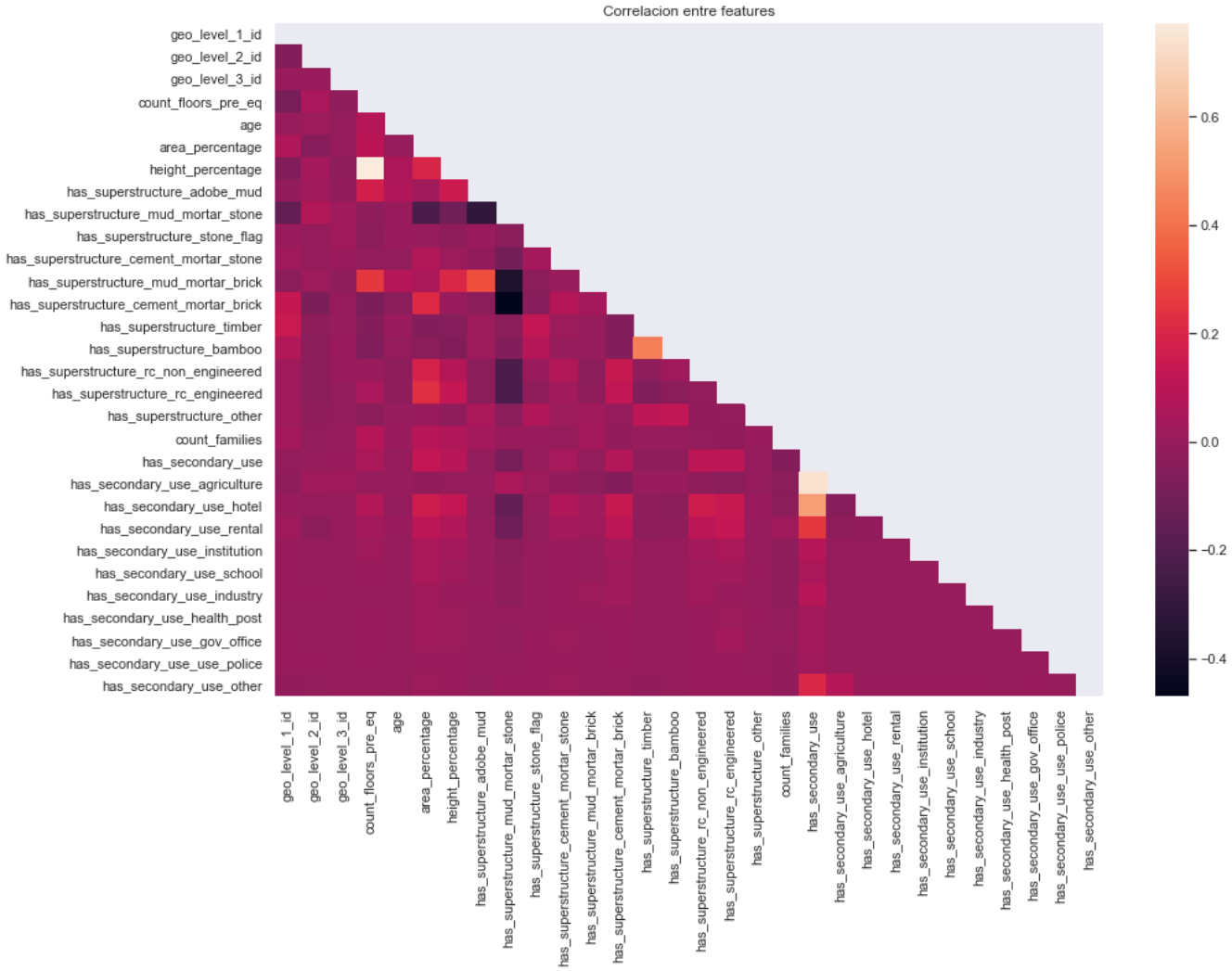

Figura 1: Relación entre todas las variables

En esta línea, podemos decir que los siguientes features no aportan al modelo de machine learning.

- has_superstructure_bamboo
- has_secondary_use_agriculture

- has_secondary_use_hotel
- has_secondary_use_rental
- has_secondary_use_institution
- has_secondary_use_school
- has_secondary_use_industry
- has_secondary_use_health_post
- has_secondary_use_gov_office
- has_secondary_use_use_police
- has_secondary_use_other
- count_floor_pre_eq
- area_percentage
- height_percentage

En el caso de has_superstructure_bamboo y count_floor_pre_eq podemos observar que está altamente correlacionado con has_superstructure_timber y height_percentage respectivamente, por lo tanto mantenerlo no nos aportan mayor información debido a su correlacionalidad.

Por otro lado, se visualiza que cualquiera de los casos particulares de has_secondary_use contiene has_secondary_use por ese motivo resulta irrelevante quedarase con dichos casos particulares.

Finalmente, para los atributos de area_percentage y height_percentage no fueron desechados por alguna relación de correlatividad con los atributos sino que se los utilizó para convertilos en un feature de volumen_percentage. Por lo tanto, luego decidimos no utilizarlos individualmente.

3.1.2. Feature Importance

3.1.3. Corroboración de la hipótesis

Nos propusimos comprobar nuestra hipótesis de la siguiente manera.
Primero utilizamos Decision Tree para entrenar el modelo al cual posteriormente le aplicamos la función permutation_importance() con el fin de obtener la importancia de los features.
De esta manera obtuvimos que las suposiciones de importancia de los atributos inferidos en la hipótesis 3.1.1 eran correctos. Además, consideramos de manera arbitraria que los features más importantes eran los que tenían un valor de 0.02 o superior. Y con dichos seleccionados, procedimos a realizar otra prueba al modelo.
El listado de features que seleccionamos se detalla a continuación:

- geo_level_1_id
- geo_level_2_id
- has_superstructure_mud_mortar_stone
- geo_level_3_id
- foundation_type_r
- area_percentage
- has_superstructure_cement_mortar_brick

- other_floor_type_q
- age
- height_percentage,
- foundation_type_i
- has_superstructure_timber
- roof_type_n
- count_floor_pre_eq
- roof_type_x
- has_superstructure_mud_mortar_brick
- position_s
- ground_floor_type_f
- other_floor_type_x
- has_superstructure_adobe_mud
- ground_floor_type_v
- roof_type_q
- has_secondary_use
- foundation_type_u
- land_surface_condition_t
- count_families
- building_id

3.2. Creacion de features

En ciertos modelos utilizados, nos pareció adecuado crear ciertos features. Se detallan a continuación los mismos.

3.2.1. Volumen

Creamos un feature llamado volumen es cual correspondía al producto de el porcentaje de área normalizada y el porcentaje de altura normalizada de edificio. Dado que no conocemos la forma geométrica real de los edificios, dicho feature corresponde a una estimación del volumen del edifico.

3.2.2. Frecuencia para atributos categóricos

Para los atributos categóricos, además de encondearlos con one-hot encoding, se creo un feature para cada uno de ellos, el cual contenía la frecuencia de valor especifico del atributo para cada observación. Los atributos utilizados fueron:

- roof_type
- land_surface_condition
- other_floor_type

- position
- plan_configuration
- legal_ownership_status
- foundation_type
- ground_floor_type

La razón se debe a que, como pudimos ver en el trabajo practico 1, había ciertos valores de cada atributo que predominaban en el dataset, por ende agregamos su frecuencia para en los debidos casos, considerarlos.

3.2.3. Valores estadísticos por regiones

Creamos tres features llamados geo_age_mean, geo_area_mean y geo_height_mean, los cuales representaban el promedio de cada feature (age, area_percentage y height_percentage respectivamente) agrupando por geo levels.

3.3. Encodeo de los geo levels

Dado que los geo levels nos parecían los que mayor información podían tener en relación al daño, debido a que la ubicación geográfica de una edifico iba a influir en el daño recibido por el terremoto. Mientras mas cerca del epicentro del terremoto mayor daño iba a recibir.

Siguiendo las ideas planteadas en el paper [2], utilizando el dataset train_values, calculamos una estimación de la probabilidad del daño recibido dependiendo del geo_level utilizando una mezcla del metodo empirical Bayes y shrinkage. Ejemplo de uno de los 9 posibles features creados es: est_proba_DG1_geo1, el mismo corresponde a la probabilidad de que el edificio haya recibido daño igual a 1, segun su zona correspondiente a geo_level_1.

Estos valores calculados los guardamos en una estructura separada, para luego ubicarlos en las observaciones que tengan los geo levels correspondientes. En los casos donde teniamos un test te prueba donde una combinacion de geo levels no se encontraba en la estructura de las estimaciones, se le asignaba probabilidad de $\frac{1}{3}$ a cada daño posible.

3.4. Outliers

Un outlier es un valor que escapa de la distribución general "normal"que vemos en los datos, y causa anomalías en los resultados obtenidos a través de los modelos de algoritmos empleados.

3.4.1. Detección

En vistas de su definición, en nuestra data set encontramos varios casos de outliers. Un ejemplo específico de muchos, es el atributo 'age' en el cual se puede encontrar un caso aislado de una edificación de antigüedad 995 años. A partir del hallazgo de estas anomalías llegamos a la siguiente pregunta: ¿Es razonable excluirlos en el training dataset solo por ser outliers?

En base a la investigación realizada para comprender en profundidad estás anomalías podemos decir que sin más información que aquella, removerlos no parece razonable. Ahora bien, también se despliega la cuestión: ¿Cuando sí resulta razonable hacerlo? Para responder las incógnitas planteadas establecimos los siguientes criterios para decidir qué hacer con ellos.

3.4.2. Criterio para remover Ouliers

- Si el outlier representa un error, por ejemplo, una edad negativa.
- Si tenemos un set de datos reducido, una sola anomalía puede hacer una gran diferencia, especialmente cuanto más se diferencie del resto.

3.4.3. Hipótesis

Considerando el criterio anterior, deducimos que no era conveniente remover los outliers ya que no nos encontrábamos en ninguno de esos casos. Ninguna de las anomalías correspondía a un valor erróneo y a su vez, nuestro data set era lo suficientemente grande como para no incurrir en el caso del último criterio.

3.4.4. Proceso de comprobación

De todos modos, sometimos nuestro supuesto a prueba. La misma consistió en remover todos los outliers que encontramos en los features, y probar como funcionaba el modelo sin ellos. Para realizar la tarea de removerlos se utilizó IQR based filtering.
El IQR consiste en identificar los outliers definiendo limites en los valores de la muestra donde el factor k del IQR pertenece al percentil 25 o por encima del percentil 75. El valor común a usar para el factor k es el valor 1.5.
A continuación se detallan los gráficos correspondientes al proceso de eliminación de los outliers de los features Age, Area y Height.

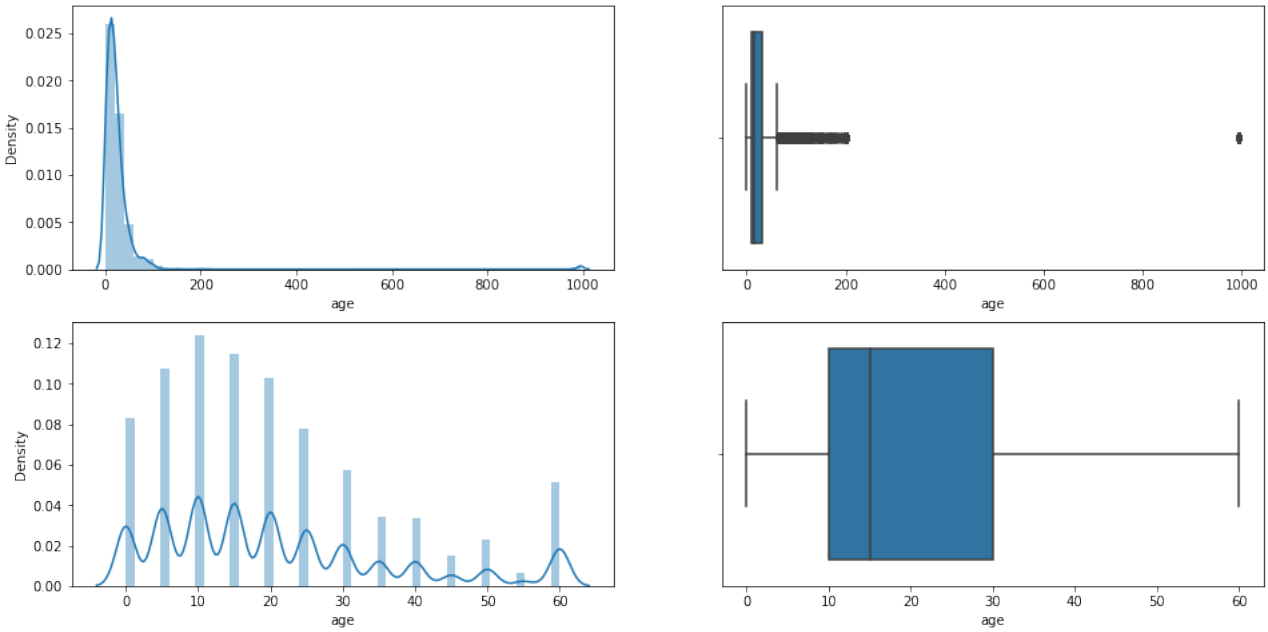

Figura 2: Removiendo outliers del atributo age

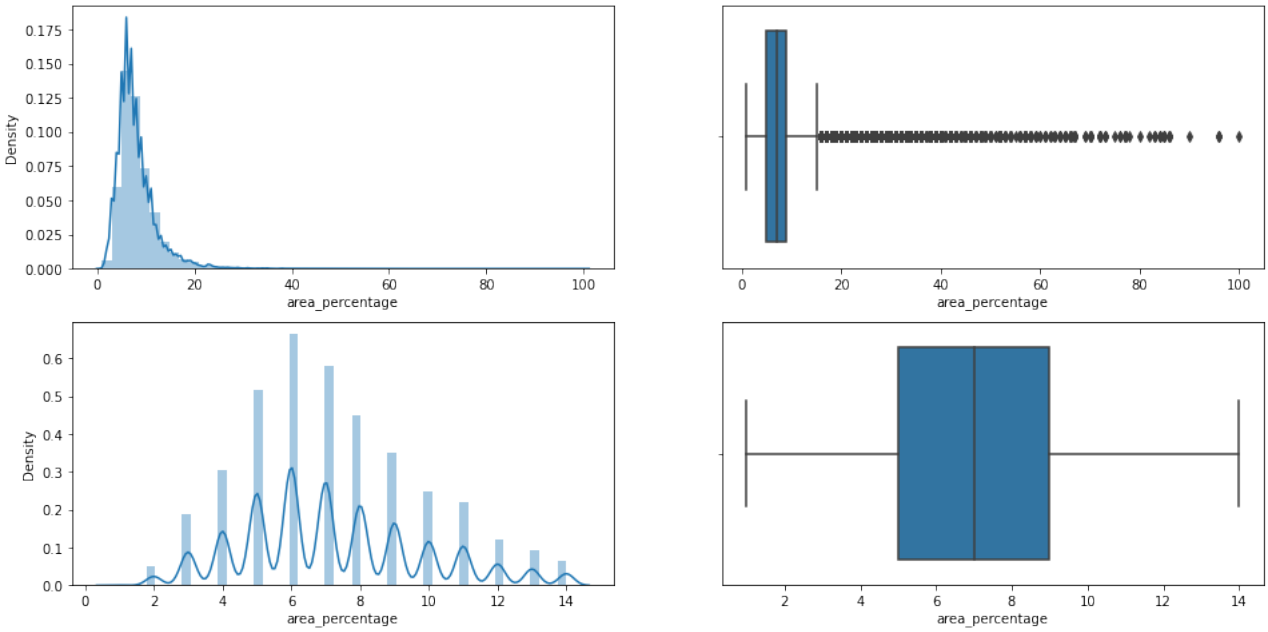

Figura 3: Removiendo outliers del atributo Area

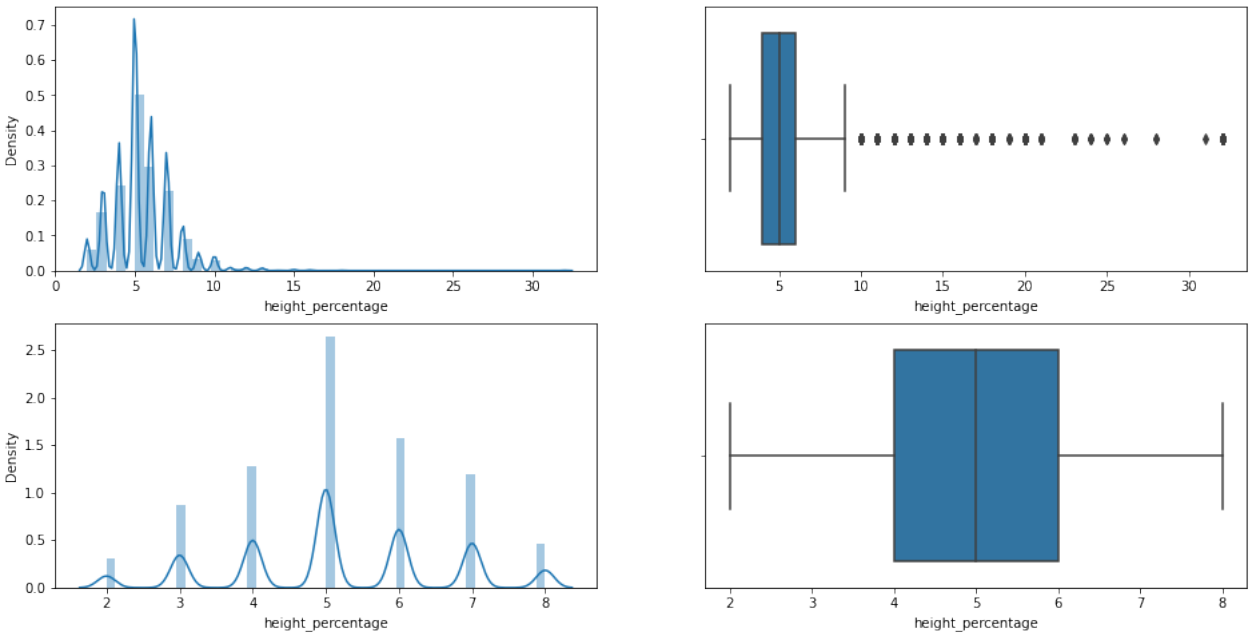

Figura 4: Removiendo outliers del atributo Height

4. Prueba de modelos

4.1. Introducción

El problema a estudiar en este caso es uno de clasificación, teniendo en cuenta que el feature de "damage_grade.[es] categórico, nos separa a los datos en tres clases: 1,2,3. Por lo tanto, todos los modelos a utilizar serán de clasificación.

4.2. Búsqueda de hiperparámetros

Para la búsqueda de hiperparámetros utilizamos los tres métodos vistos durante el curso, GridSearch, RandomSearch y Optimización Bayesiana. Se usaron arbitrariamente para comparar los resultados de cada uno y los tiempos de ejecución.

Junto con la optimización Bayesiana se utilizó como método de prueba de hiperparámetros K-Fold cross-validation

4.3. Entrenamiento

Todos los modelos utilizados fueron entrenados de la misma forma. Se dividió el set de datos en dos, un set de train (X_train,y_train) y un set de test (X_test,y_test) teniendo un tamaño del set de test de 20 % del total de datos.

4.4. Overfitting y Underfitting

Para todos los modelos, se busca la puntuación de que el modelo prediga bien el set de entrenamiento y el set de test, con la metrica elegida(f1_micro).

Para comprobar que el modelo entrenado no esté överfitteandoïos fijamos en la puntuación del set de entrenamiento en comparación a la del set de test, si la diferencia entre ambas es muy grande, teniendo la puntuación del set de entrenamiento mayor a la del test, entonces podemos asumir que el modelo överfittea:

Para comprobar que el modelo no esté ünderfitteando; se corrobora que el error de entrenamiento no sea "muy bajo".

5. Red Neuronal

De los modelos de redes neuronales optamos por probar un perceptrón multicapa clasificador. Del dataframe original nos quedamos solo con los features mas importantes dados por el análisis hecho previamente en las secciónes 3.1 y 3.1.2. A esto luego se le hace el one-hot encoding a los datos categóricos.

5.1. Teorema de aproximación universal

Nos garantiza que con una sola capa oculta de suficiente numero de neuronas, podemos representar cualquier función. Por lo tanto un perceptrón multicapa podrá separár cualquier set de datos.

5.2. Hiperparámetros

Teniendo en cuenta el teorema 5.1, el hiperparámetro más importante a refinar es hidden_layer_sizes. Con este y algunos más, los hiperparámetros buscados fueron:

- hidden_layer_sizes: {10,20,30,40,50,100,200}
- activation: {tanh,relu}
- solver: {sgd,adam}
- learning_rate: {constant,invscaling,adaptive}
- max_iter:{500}

5.3. Resultados

La busqueda de hiperparámetros dió como resultado los siguientes Hiper-Parámetros Optimos: {activation: relu, hidden_layer_sizes: 200, learning_rate: constant, max_iter: 500, solver: adam}

Luego entrenando y prediciendo con el training y el test set, las puntuaciones que salieron fueron:

Training Score: 0.6074

Test Score: 0.6064

5.4. Conclusiones

Los resultados concuerdan con un claro ejemplo de underfitting, esto suponemos que es por la inmensa cantidad de datos y la baja complejidad del modelo.

En este caso tambien se puede hacer uso de una predicción inicial del training_set, como una ̈first_prediction; para luego con un modelo mas complejo predecir de nuevo. Esto se hizo con un Random Forest Classifier, explicado en la sección 7.4.

6. DecisionTree

Otro de los modelos que nos propusimos probar fue el DecisionTree Classifier, con el cual hicimos varias pruebas con todos los datos, refinando los hiper-parámetros. En particular este modelo nos permitía comprobar resultados con mayor rapidez que otros ya que no presenta la complejidad de los mismos.
Por lo tanto, el proceso a seguir era que si los resultados que nos arrojaba el DecisionTree Classifier eran razonables y favorecedores esto es, que no tenga overfitting, avanzábamos el transcurso de comprobación utilizando modelos más complejos como Random Forest o XGBoost. O también, si visualizamos underfitting en los resultados, procedíamos a realizar más pruebas con aquellos modelos señalados, debido a que podríamos obtener mejoras si utilizábamos algoritmos más complejos.
En caso contrario, no proseguíamos con las pruebas ya que si observáramos overfitting en los resultados dados por el Decision Tree, entonces significaría que debemos bajar la complejidad, por lo tanto, no procedemos a comprobar con modelos más complejos.
Por último podemos resaltar que con el presente modelo hallamos los features más importantes,como se detalla en la sección 3.1.2.

6.1. Hiperparámetros

Los hiperparámetros que utilizamos fueron:

- criterion
- splitter
- min_samples_split
- max_features

El hiperparámetro que refinamos varias veces, debido a que los valores que daban eran cercanos a los maximos del intervalo, fue el de min_samples_split. Empezando con un rango arbitrario de (2;40), llegando a un rango de (99,200).

6.2. Resultados

Los decisionTree empleados cumplieron la funcionalidad de ser guías ó tentativas de qué tan buenos eran nuestros momentos, antes de pasar a otra eventual etapa de prueba. Por ello, podemos decir que los resultados obtenidos fueron valiosos para ajustar el modelo a una mejor versión.

6.3. Conclusiones

El modelo de Decision Tree no se adapta a la complejidad del problema a resolver. Por lo tanto, no fue utilizado para tal fin sino para hacer .ªproximaciones"de qué tan viable resultaría cierto modelo sometido a prueba con otros algoritmos más complejos.

7. Random Forest

A continuación se detallan las pruebas realizadas con el modelo de Random Forest.

7.1. Prueba utilizando la totalidad de los features

Como primera aproximación, decidimos probar cuál era el resultado de este modelo haciendo uso de todos los features. La única manipulación realizada en esta prueba fue el encodeo de los features categóricos con one-hot-encoder.

7.1.1. Resultados

Entrenando y prediciendo con el training y el test set, las puntuaciones obtenidas fueron:

Training Score: 0.8246834228702993
Test Score: 0.7236238752134456

7.1.2. Conclusiones

Se observa en base a los resultados obtenidos un score superior para el Training set que para el Test, aunque ambos muestran un valor elevado. A su vez, la marcada diferencia entre los scores de ambos señalan un situación clara de overfitting. Esto es, el modelo está aprendiendo en exceso los casos particulares siendo de esta manera incapaz de reconocer nuevos datos de entrada (excepto los idénticos a los ya aprendidos en el entrenamiento).

7.2. Prueba utilizando una menor cantidad de features

En la segunda tentativa, escogimos de forma arbitraria quedarnos sólo con unos atributos específicos, con el objetivo de observar el comportamiento en un caso contrario al puesto en la sección 7.1.
De esta manera, los features seleccionados fueron:

- building_id
- geo_level_1_id
- age
- foundation_type
- land_surface_condition
- ground_floor_type

7.2.1. Resultados

Entrenando y prediciendo con el training y el test set, las puntuaciones obtenidas fueron:

Training Score: 0.6748752877973906
Test Score: 0.6692887703612748

7.2.2. Conclusiones

En este caso, podemos notar en principio una disminución en los valores dados por el score del Training y el Test, sin embargo, no hay presencia de overfitting como ocurría en el caso analizado en las conclusiones de la sección 7.5.2.
Por otro lado, estos resultados nos dan muestra de una situación de underfitting. Es decir, nuestro modelo no tiene la capacidad. Ahora bien, lo resultados arrojados por el modelo son notablemente bajos y esto es debido a que nuestros datos de entrenamiento son muy pocos por lo que el modelo no será capaz de generalizar el conocimiento e incurre en underfitting.

7.3. Utilizando los features catalogados como importantes

A raíz del análisis de los atributos realizado en la sección 3.1.1 y los resultados conseguidos, desarrollados en la sección 3.1.2, pusimos a prueba un modelo que contemple solamente los features catalogados como importantes, es decir, los especificados en dichas secciones.
Nuevamente, se detalla que del listado de features con sus importancias decimos arbitrariamente conservar las que poseían una importancia mayor o igual a 0.02.

7.3.1. Resultados

Entrenando y prediciendo con el training y el test set, las puntuaciones obtenidas fueron:

Training Score: 0.8216039907904835
Test Score: 0.7279599393718463

7.3.2. Conclusión

En un balance general con los resultados obtenidos hasta el momento en las secciones 7.1.1 y 7.2.1, vemos que este resultado se acerca más al del caso donde conservamos la totalidad de los atributos. Por lo que nuestra suposición en el grado de importancia de los features era acertado, es decir, los que excluimos del modelo no representaban mayor relevancia tal que los valores obtenidos fueron similares a los de la sección 7.1.

7.4. Prueba con importantes y ensamble

En esta prueba nos propusimos mejorar la predicción del modelo, agregando una columna extra al dataset. Esta columna vendría a ser una primera predicción hecha con algún modelo menos complejo, con la esperanza de que, como varios algoritmos numéricos, mejore.

7.4.1. Hipótesis

La idea principal, es que agregando una columna más al dataset, no solo reducimos el overfitting, sino que también, reducimos el error de test del modelo.

Como varios métodos numéricos, que al hacer un paso inicial en el algoritmo, la hipótesis inicial fue que esto mismo pasaría con un modelo de machine learning, dado que los modelos mas complejos, internamente hacen un ensamble de previas predicciones para reducir el error de las mismas.

7.4.2. Conclusión

Hecha la prueba, se puede ver que la hipótesis se cumple, aunque los resultados de la prueba no son significativos, debido al bajo decrecimiento de los errores de entrenamiento y de prueba.

Por lo tanto se concluye que el enfoque tomado fue inteligente, pero no lo suficientemente significativo como para dar un giro drástico a como se encararon las siguientes pruebas.

7.5. Prueba removiendo del dataset los outliers

La siguiente prueba se desarrolló utilizando la totalidad de los features excluyendo los outliers de los atributos age, height_percentage y area_percentage, la explicación de los mismos se encuentra desarrollada en la sección 3.4.

7.5.1. Resultados

Entrenando y prediciendo con el training y el test set, con los siguientes hiperparámetros encontrados con RandomSearch. 'n_jobs': -1, 'n_estimators': 400, 'max_features': 'auto', 'max_depth': 170, 'class_weight': 'balanced', 'bootstrap': True las puntuaciones obtenidas fueron:

Training Score: 0.9946621979751687
Test Score: 0.5295528524927189

7.5.2. Conclusión de la prueba realizada

Los resultados apoyan nuestro supuesto inicial de la sección3.4.3, podemos notar que el modelo está sobreentrenado y aprende rapidamente los datos. De esta manera, El modelo es incapáz de generalizar de forma viable nuevos datos de entrada que se salen siquiera un poco de los rasgos ya establecidos.
Además, estas observaciones nos dejan concluir que en este trabajo en particular no es conveniente remover los outliers.
Por último, cabe destacar que también se consideró que podría existir cierta mejoría si no removíamos la cantidad total de outliers de particularmente el atributo age. Es decir, no remover los outliers menores a la antiguedad de cien años y sí remover los que superen dicho número. Pero no se obtuvo un mejor score ni un menor overfitting que en la consideración anterior. Por lo tanto, se concluye que no era conveniente tampoco excluir ciertos outliers del atributo age.

8. XGBoost

Otro de los modelos probados, fue XGBoost, Para comenzar con XGBoost, decidimos realizar una búsqueda de hiperparámetros con la totalidad de los features, en donde aquellos categóricos fueron encodeados con la técnica de one-hot encoding.

8.1. Hiperparámetros

Para la búsqueda de los hiper-parámetros utilizamos la totalidad de los features, en donde aquellos categóricos fueron encodeados utilizando la técnica de one hot encoding. Debido a la cantidad de tiempo consumida por dicha tarea, se realizó una única vez. En los casos en donde agregábamos nuevos features, se analizaba para cada caso específico si alguno debía variar. Los hiper-parámetros obtenidos fueron:

- n_estimators 2000
- max_depth 12
- learning_rate 0.02
- max_features
- num_class 3
- colsample_bytree 0.4
- importance_type gain
- objective multi:softprob
- subsample 0.8

8.2. Pruebo haciendo uso de los features importantes

La prueba detalla a continuación hizo uso de los features importantes señalados en la sección 3.1.2 y encodeando los features categóricos con la técnica de one-hot encoding.

8.2.1. Resultados

Entrenando y prediciendo con el training y el test set los resultados obtenidos fueron:

Training Score: 0.8234698772064467
Test Score: 0.7425797663130025

8.2.2. Conclusiones

Como primera conclusión que determinamos podemos decir que los resultados hallados son los mejores hasta el momento.

8.3. Prueba haciendo uso de los features importantes y geo_levels encodeados

La prueba detalla a continuación hizo uso de los features importantes señalados en la sección 3.1.2 y encodeando los geo_levels de la forma desarrollada en la sección 3.3.

8.3.1. Resultados

Entrenando y prediciendo con el training y el test set los resultados obtenidos fueron:

Training Score: 0.810941097467383
Test Score: 0.6654323593177414

8.3.2. Conclusiones

El resultado obtenido no resultó satisfactorio, los valores presentan un claro caso de overfitting. Las razones del por qué no resultó positiva esta prueba se las atribuimos a que el encodeo realizado a los geo_levels en la sección 3.3 no resultó beneficioso para el modelo.

8.4. Prueba con features creados y Geo levels encodeados

Se utilizaron la frecuencia de los features categóricos, dicha forma de cálculo se encuentra desarrollada en la sección 3.2, junto con los valores estadísticos por región desarrollados en la sección 3.2.3 para entrenar dicho modelo. Por un lado, se lo entreno sin agregar los geo levels encodeados y luego agregandolos.

8.4.1. Resultados

AL utilizar los geo levels encodeados, los resultados tuvieron una leve mejora en el score, que al no utilizarlos. Los resultados fueron:

Sin geo levels encodeados:

Training Score: 0.8795 Test Score: 0.7512
Con geo levels encodeados:

Training Score: 0.8654 Test Score: 0.7516

8.4.2. Conclusión

En este caso, lo que sucedió fue que teníamos un buen score para el set de pruebas, así como también un un alto score para el entrenamiento.
Sin embargo, al realizar el submit a DrivenData, el score obtenido fue menor. Por ende, nos dio a entender que nuestro modelo no era bueno para generalizar.

9. Conclusiones Finales

Empezando con redes neuronales para darnos una idea la implementación de modelos de Machine Learning. Rápidamente nos dimos cuenta que el problema no iba a ser tan fácilmente resuelto.

Los Arboles de decisión fueron una gran herramienta de prueba para ahorrar tiempo de búsqueda de hiperparámetros. Al tener un modelo menos complejo, la búsqueda de hiperparámetros resultó más rápida y pudimos observar si los datos con los que entrenamos el modelo overfitteaban. Si este no era el caso, llevábamos el problema a modelos más complejos como el Random Forest.

El Random Forest fue unos de los modelos que más probamos y que más tiempo llevó la búsqueda de hiperparámetros. Los algoritmos de búsqueda más optimizados como RandomizeSearchCV resultaron útiles para atacar este problema. Optar por este modelo significó, en general, una mejora en las predicciones del modelo.

El problema más grande que se presentó, fue errores en tiempo de ejecución en la búsqueda de hiperparámetros que derivaron en horas y hasta días perdidos.

Para este problema en particular, el modelo que mejor se adaptó a los datos fue el de XGBoostClassifier detallado en la sección 8, siendo este el que mejor métricas dio y uno de los que menos llegamos a probar debido a la gran pérdida de tiempo ocasionada por probar RandomForest .

Por ultimo, una posible mejor para la codificacion de geo levels, seria sacar provecho de la jerarquia interna. Esto es, la inclusion de cada uno de los geo levels mas chicos en los mas grandes, estos es $geo\_level\_3\_id \subset geo\_level\_2\_id \subset geo\_level\_1\_id$.

Referencias

[1] Luis Argerich ,Natalia Golmar, Damián Martinelli,Martín Ramos Mejía y Juan Andrés Laura., *75.06, 95.58 Organización de Datos, Apunte del Curso*

[2] Daniele Micci-Barreca, *A preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems*, Volume 3, Issue 1.

[3] Aurélien Géron*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition.