# Cajamar UniversityHack 2020

# Primera Fase

Reto:

Mindsait Land Classification

Participantes:

Javier Moral, David Colás, Laura Méndez

Equipo:

DataShow

# Preporcesado de Datos:

La clase DataTreatment realiza un proceso secuencial de preprocesado de datos basado en 6 módulos. Cada método programado en la clase realiza un proceso distinto perteneciente a un módulo concreto. Cada uno de ellos está programado para que se ajusten los modelos de preprocesado con los datos de train y se mapeen los cambios a los datos de test y estimar. Al igual que todo el repositorio, la clase DataTreatment ha sido programada OOP para una mayor facilidad y comodidad de comprensión a la vez que una buena claridad de estructura.

## Train y test split:

El primer paso para tratar los datos es dividir el dataset de modelar en train y test con una proporción del 70% y 30% respectivamente. La idea de la clase DataTreatment es ajustar todos los modelos con los datos de train y mapearlos a los sets de test y estimar.

## Imputación de missing values:

Se han programado 3 métodos distintos de imputación de missing values. Datawig, Imputación Simple y eliminación de registros. Después de realizar pruebas con los 3, ninguno mejoraba los resultados en predicción respecto a los demás, por lo que se ha optado por imputación simple debido a su bajo coste computacional. Los 3 modelos están programados en la clase Missings del fichero missings.py. A continuación una breve descripción de los tres métodos implementados:

- Datawig: uso de redes neuronales para imputar variables categóricas y numéricas
- Eliminación de registros: se eliminan los missing values
- Imputación simple: Imputación de variables numéricas por mediana y variables categóricas por moda

Como se ha mencionado se imputan los datasets de test y se estima con los valores/modelos obtenidos en train. Un ejemplo sería imputar los missing values de la variable numérica ficticia 'RZ' en el set de test con la mediana de la variable 'RZ' de los datos de train.

## Eliminación de variables con valores constantes:

Pese a no eliminar ninguna variable, se ha programado un método dentro de DataTreatment (dropConstants) que elimina toda aquella variable que tenga un valor constante para todos sus registros.

## Feature Engineering:

- Estadísticos de variables:

Para aquellas variables, cuya información viene expresada en deciles de su distribución, hemos realizado las siguientes transformaciones en sus variables:

- Antes de nada, debido a la magnitud de las variables y con el objetivos de cada uno de los deciles sean comparables entre sí, vamos a transformar estos datos a través de una función logarítmica previamente desplazada a 1.
- Bajo la premisa de que los deciles de forma independiente no son capaces de aportar señales sino ruido, vamos a hacer una transformación para más adelante sacar una serie de estadísticos básicos de su conjunto.
- Ya homogeneizado cada uno deciles, y con el objetivo de medir las irregularidades causadas por la forma o naturaleza de cada una de las parcelas, calculamos los cambios porcentuales de un decil a otro para posteriormente calcular sus estadísticos básicos de media, mediana y desviación estándar.

Para la variable AREA, primero se ha realizado una transformación logarítmica. Seguidamente, se han calculado los deciles 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 95, 99 del área para cada grupo de la variable CADASTRALQUALITYID usando el método *groupby* de pandas.

La idea es crear una variable por cada decil pero que el valor de cada registro en esas nuevas variables se corresponda con el decil de su categoría de CADASTRALQUALITYID. También se crean las mismas variables realizando el mismo cálculo de deciles para cada categoría de la variable MAXBUILDINGFLOOR. Finalmente, para los dos casos anteriores se calcula la diferencia entre los deciles 99 y 1 en otra variable. El código para obtener las variables mencionadas se encuentra en el método featureEngineeringStatictics.

- Clustering de Comunidades:

    - Para explotar las variables de longitud y latitud que previamente han sido escaladas y desplazadas aleatoriamente, vamos a calcular el número óptimo de clúster o comunidades para aprovechar la única información útil en estas dos variables: la distancia entre sus respectivos puntos.
    - Previamente a este cálculo y con el objetivo de que tanto la longitud como la longitud sean comparables entre sí, escalamos dichas variables con un escalador *MinMax* para no distorsionar tampoco su distribución.
    - El cálculo de los clústeres se calcula con *K-means*, tomando como número óptimo de clústeres aquel punto en cual, el coste de añadir una comunidad más, no suponga una reducción significativa de la suma de distancias a cada uno de sus respectivos centroides.
    - Una vez calculados los clústeres, calcularemos la distancia euclídea de cada uno de sus puntos a sus respectivos centroides. De esta forma, con la nueva información de comunidades y distancias, seremos capaces de explotar toda la información que reside en las variables de longitud y latitud que sin previa transformación carecen de valor.

      Dichos cálculos se mapean a los datasets de test y estimar con los modelos ajustados en train. Todo el código se encuentra en el método clusteringCommunities.

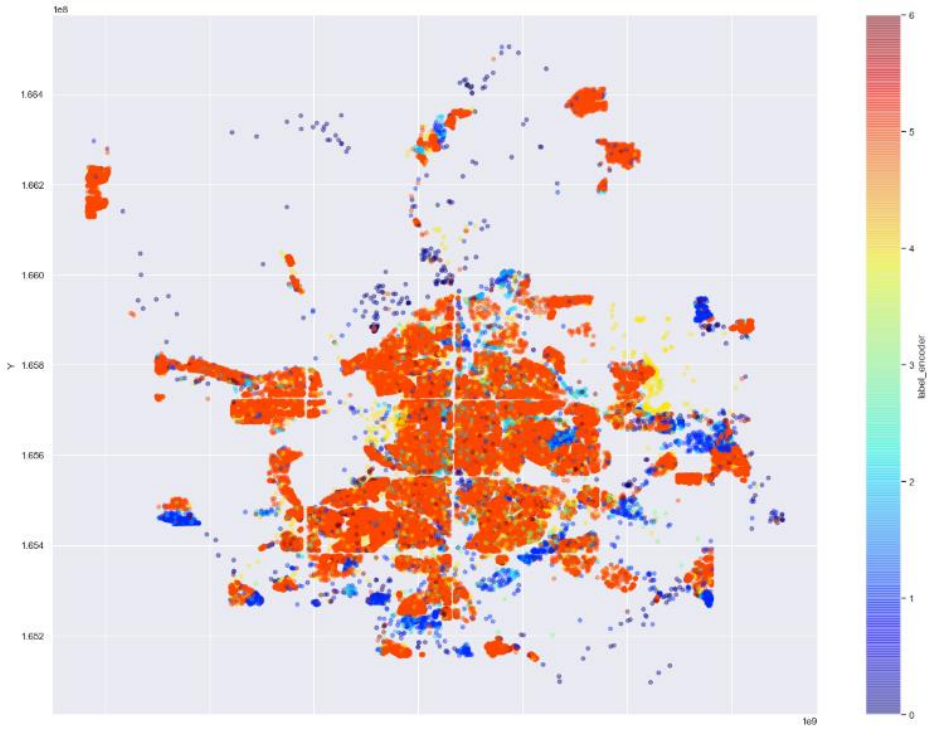
- Encoding de variables categóricas:

  Por lo general, los modelos de sklearn no admiten variables categóricas con strings indicando la categoría a la que pertenece cada registro, es por eso que se debe aplicar un encoding. Nosotros hemos optado por CatBoost encoder como modelo para convertir nuestras variables categóricas a numéricas. Para más información sobre cómo dicho modelo realiza el encoding, diríjase a la documentación de la librería category encoders de Python. Método categoricalEncoding.

- Creación de variables derivadas de coordenadas:

  Este es uno de los pasos más destacables del módulo de feature engineering. Sabemos de antemano que X e Y son una versión escalada y desplazada aleatoriamente de las coordenadas (manteniendo la relación de posición con el resto de puntos

registros). Si hacemos una representación gráfica de los puntos en las coordenados podemos ver como se corresponde con el mapa de Madrid




Teniendo en cuenta dicha relación, hemos estimado de forma subjetiva las coordenadas límite reales del mapa. Longitud de -3.898 a -3.507 y latitud de 40.337, 40.545. Lo primero que hacemos es trasladar las variables X e Y al rango estimado mediante la función rangeTranfer del módulo mapCoordinates de funciones auxiliares.

Una vez escaladas las variables, calculamos el centroide de coordenadas de nuestros registros. Con el centroide calculamos la distancia euclidiana de cada punto al centro. También obtenemos la distancia Harvesiana con y una serie de transformaciones con seno y coseno. Todo el proceso se puede observar en el método featureEngineeringCoordinates.

## Escalado Min-Max:

Para tener todas las variables en la misma escala y hacer un uso consistente y no sesgado de modelos como redes neuronales o regresiones, se escalan todos los datos entre cero y uno con el MinMax scaler del paquete sklearn.

Si se opta por modelos con árboles de decisión, la escala no afecta al ajuste del modelo, por lo tanto, preferimos escalar y tener los datos listos para el uso de cualquier clasificador. El scaler se ajusta con los datos de train y se mapea a los datos de test y estimar usando el modelo ya ajustado. Todo el desarrollo se encuentra en el método de featureScaling.

## Eliminación de variables altamente correladas:

El último módulo de la clase DataTreatment trata de eliminar variables que puedan estar metiendo ruido a los modelos. Como hay muchas variables muy parecidas, se opta por usar la correlación de Pearson como indicador de similitud entre variables.

El proceso es el siguiente, el usuario elige un umbral de correlación, se calcula la matriz de correlaciones entre nuestras variables explicativas, se obtienen los pares de variables que superen el umbral elegido y se elimina las variables con menor poder predictivo.

Para ver qué variable de cada par tiene mayor poder predictivo, se crean 2 regresiones logísticas, una con cada variable, y se borra la variable que obtiene menor accuracy en predicción. Todo con los datos de train. La variable eliminada también se quita de los datasets de test y se procede a estimar. Todo el proceso se desarrolla en el módulo de deleteCorrelations. Se pone por defecto un umbral de 0,97.

## Orden del proceso

El orden que sigue la clase a la hora de ir procesando los datos es el siguiente:

1. Train test-split.
2. Imputación de missing values.
3. Eliminación de variables con valores constantes.
4. Estadísticos de variables.
5. Clustering de comunidades.
6. Encoding de variables categóricas.
7. Cálculo de variables derivadas de coordenadas.
8. Escalado Min-Max.
9. Eliminación de variables altamente correlacionadas.

# Balanceo de datos

El enorme desbalanceo que presentan los datos a modelar nos ha llevado a crear una clase que presenta distintos métodos para tratar este problema. La clase se llama UnbalancedDataSampling y mayormente hace uso de la librería *imblearn* de Python.

Como solo se ha hecho uso de una de las técnicas programadas, únicamente se van a mencionar:

- SMOTE + Edited Nearest Neighbours.
- SMOTE + Edited Nearest Neighbours (seleccionando manualmente el número de observaciones a los que hay que aumentar las clases minoritarias).
- SMOTE + Tomek Links.
- Random Undersampling.
- SMOTE (seleccionando manualmente el número de observaciones a los que hay que aumentar las clases minoritarias).

Se ha empleado el método de Random Undersampling sobre la clase mayoritaria ('RESIDENTIAL'), reduciendo las observaciones pertenecientes a la categoría hasta 35.000 registros.

# Modelización

En el módulo de modelización se han probado una amplia variedad de métodos hasta dar con el modelo de mayor poder predictivo. En el código solo se ha dejado la clase del modelo final para facilitar el trabajo al corrector, si se quisiera ver cualquiera de las demás técnicas probadas no tendríamos inconveniente en facilitar el código. Se han probado Redes Neuronales, ensembles secuenciales usando la técnica Cascading, GANS y autoencoders para hacer oversampling sobre las clases minoritarias, ensembles con clasificadores especializados en las clases mayoritaria y minoritaria, búsqueda de hiperparámetros con optimización Bayesiana…

Después de múltiples pruebas con una enorme variedad de clasificadores, métodos de combinación y técnicas para tratar el desbalanceo, el modelo que ha obtenido mejores resultados ha resultado ser de los más simples que se han probado. Consiste en ajustar un Extreme Gradient Boosting (XGBClassifier de la librería xgboost) sobre los datos de train optimizando sus hiperparámetros mediante una búsqueda aleatoria con validación cruzada (RandomSearchCV de la librería sklearn). Dicha búsqueda aleatoria prueba 20 combinaciones distintas haciendo validación cruzada sobre 3 particiones de los datos de train. Una vez se encuentra la combinación con mejor accuracy, se vuelve a ajustar el modelo con las tres particiones antes de predecir el dataset de test.

## RESUMEN

El mejor modelo en función de imágenes satelitales para predecir y clasificar los suelos fue un Extreme Gradient Boosting con los siguientes valores en sus hiperprámetros: base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.6, gamma=0.5, learning_rate=0.05, max_delta_step=0, max_depth=80, min_child_weight=1, n_estimators=1800, objective='multi:softprob', reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=None, subsample=0.8. Para llegar a esto, se realizó el pre-procesado de los datos que constó de la partición de los datos en train y test, imputación de los missing values por el método imputación simple, eliminación de variables con valores constantes para todos sus registros. Transformación de las variables expresadas en deciles a través de una función logarítmica previamente desplazada a 1 y el cálculo de estadísticos básicos de las variables.

Asimismo, se estima el número óptimo de comunidades, a través de las variables de longitud y latitud, las cuales se escalan previamente. Se aplica un encoding para transformar las variables categóricas a numéricas.

Luego, de forma subjetiva, se estiman las coordenadas límites del mapa y se trasladan las variables X e Y al rango estimado y se calcula el centroide de coordenadas de nuestros registros. Asimismo, todos los datos son escalados y las variables altamente correlacionadas son eliminadas.