minsait

An Indra company

idealista

UNIVERSITYHACK 2020

DATATHON

CAJAMAR UNIVERSITYHACK 2020:

RETO MINSAIT LAND CLASSIFICATION

SentinelAdmins

Alejandro Vaca Serrano & Armando Robles Liberato

ÍNDICE

INTRODUCCIÓN

En este trabajo, se busca crear un modelo automatizado para clasificación de uso de suelo, dadas ciertas características extraídas de imágenes de satélite, concentradas en la provincia de Madrid. Aquí se describen las fuentes de datos y las técnicas utilizadas en la creación del modelo en cuestión.

Lo que aquí se resume, en diferentes pasos secuenciales, ha sido en realidad producto de un proceso iterativo circular, en el que simultáneamente entrenamos modelos mientras seguimos indagando en nuestros datos y buscando nuevas variables. Este método nos ha permitido aprender las regiones óptimas de los parámetros de cada modelo, al mismo tiempo que descartar variables o idear otras nuevas, en base a la interpretación de los mismos. Se ha escogido contarlo de manera secuencial para simplificar la comprensión del trabajo.

1. EXPLORACIÓN INICIAL DE LOS DATOS

Como inicio del Análisis Exploratorio, se hizo un gráfico de las variables X e Y, y se identificaron ciertas características propias de la ciudad de Madrid, lo que nos llevó a la conclusión de que podíamos utilizar la posición de los puntos para cruzar con datos geolocalizados.

Figura 1. Mapa de los puntos en coordenadas X e Y.

Luego, por los nombres y por su distribución, vimos que las variables de color correspondían a quantiles, y que se distribuirían muy similares entre sí. Más adelante en la competencia se aclaró esto para todos los equipos.

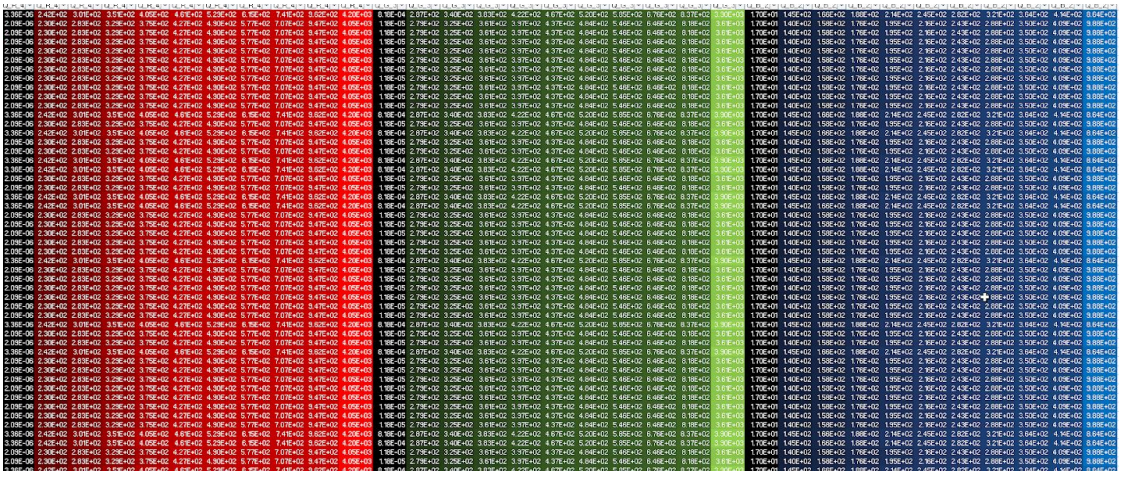


Figura 2. Vista de las columnas de color, donde el fondo de cada celda refleja la intensidad.

Después, vimos las variables GEOM. Nuestra primera intuición fue que podrían describir los lados de un cuadrangular que describa el terreno donde se encuentre ese dato, pero se descartó esta teoría rápidamente, pues no cumplen la desigualdad triangular como para ser partes de un polígono. Nuestra conjetura es que son variables que describen la forma del inmueble, vista desde la imagen de satélite; estas "formas" podrían obtenerse aplicando ciertos filtros de convolución a cada imagen.

Por último, para comprender mejor las variables de color, nos hemos descargado imágenes de Madrid, provenientes del mismo Sentinel II, y obtenidas del Copernicus Open Science Hub . Las hemos visualizado con el Sentinels Application Platform (SNAP), una aplicación de código abierto:

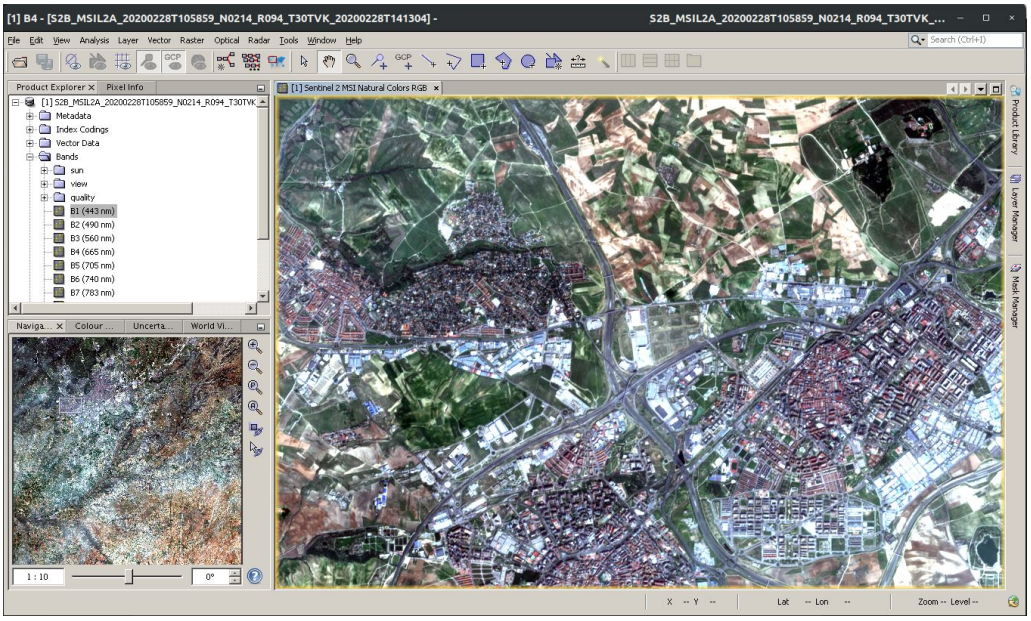


Figura 3. Captura de Pantalla del Sentinels Application Platform.

Gracias a esta herramienta, comprendimos qué longitudes de onda correspondían a cada banda de color, y vimos qué tipo de estructura reflejaba más luz en cada banda, especialmente para la banda infrarroja cercana (ver Anexo 1). También nos ayudó a comprender mejor los diferentes estudios que leíamos sobre Remote Sensing aplicados a problemas similares al de esta competencia.

2. FEATURE EXTRACTION & ANALYSIS

Respecto al análisis descriptivo de los datos y las conclusiones que obtuvimos de los mismos, sólo diremos brevemente que se utilizaron KDEplots con Seaborn, como los de

las figuras 4 a 6, para analizar el posible impacto que diferentes variables podrían tener sobre la clase. Esto es con el objetivo de agilizar la lectura de este documento y sintetizar las ideas, resaltando lo que es claramente diferencial en este trabajo, sin detallar el análisis de cada una de las casi trescientas variables que se utilizan al final en este trabajo.

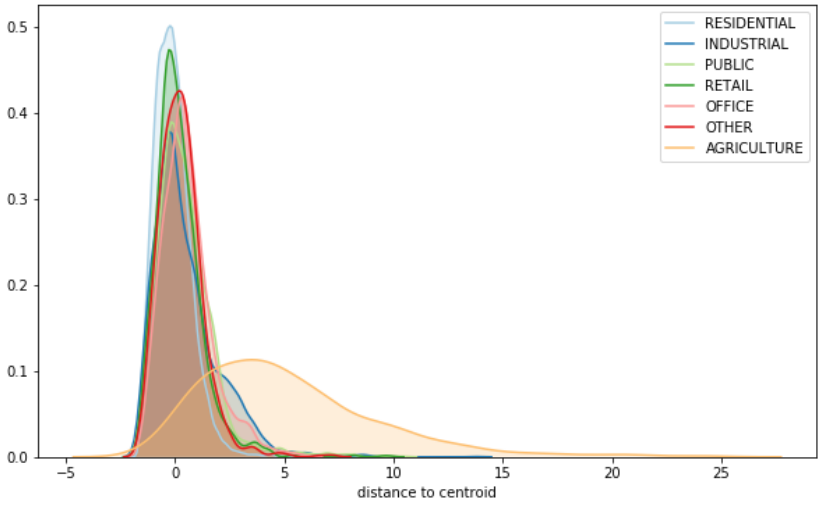


Figura 4. Distribución de la Distancia al Centroide[1] (norm.) por clase.

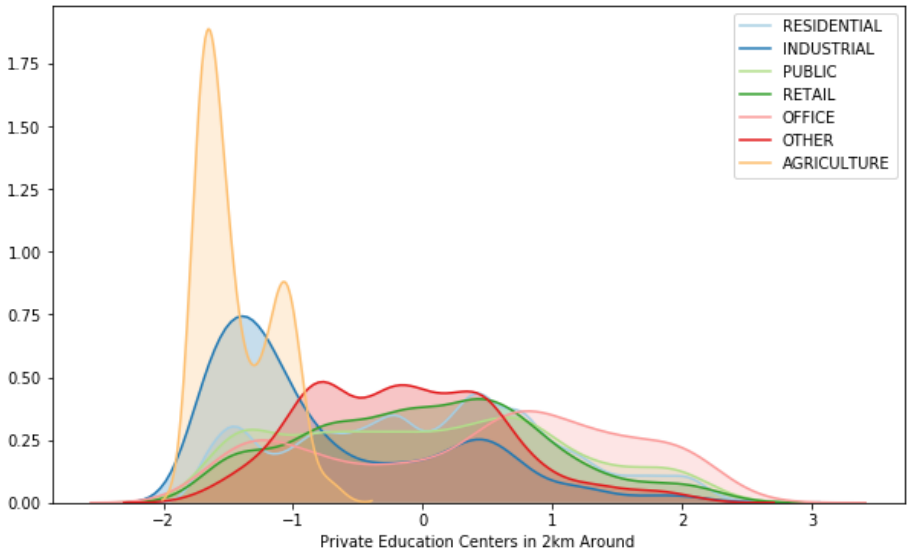


Figura 5. Distribución de los centros de educación privada (normalizado) en 2km alrededor por clase.

[1] La variable "distance to centroid" se explica más adelante; es un ejemplo ilustrativo de una variable que parece tener una distribución claramente distinta para la clase agriculture.

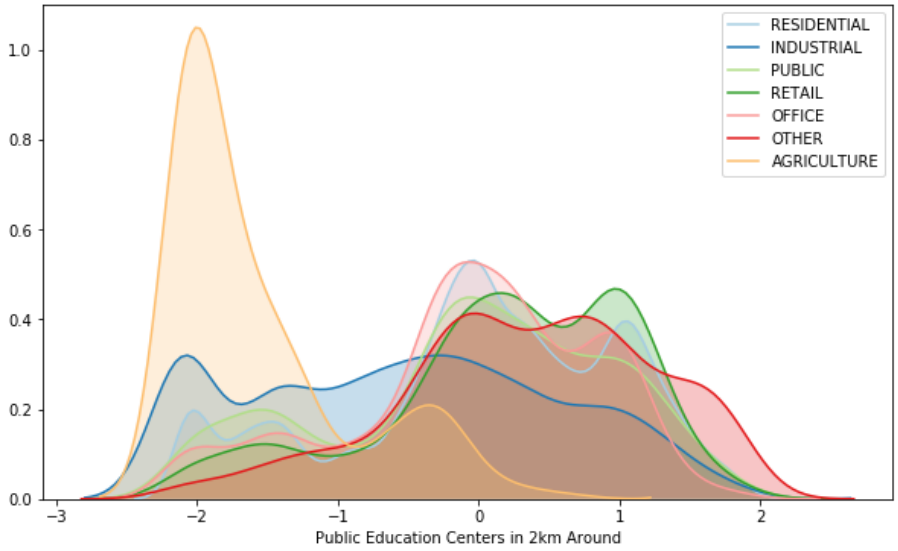


Figura 6. Distribución de los centros de educación pública (normalizado) en 2km alrededor por clase.

2.1. INTERPOLACIÓN DE LA POSICIÓN REAL

Como parte de la generación de nuevas variables para la mejora de los modelos, se decide estimar la latitud y la longitud aproximadas de cada punto del plano a partir de las variables X e Y. Para ello, se utilizan herramientas de visualización como Leaflet y Plotly, para encontrar puntos conocidos de Madrid en base a las formas de la representación espacial de X e Y. Algunos de estos puntos son el Retiro, Las Ventas, intersecciones entre calles, y otros. Estos puntos identificados pueden verse en la figura 7.

Figura 7. Puntos identificados de Madrid para la interpolación lineal.

Se diseñan modelos de regresión lineal simple de la longitud sobre la X y la latitud sobre la Y, añadiendo en ambos casos dos variables binarias: Sur y Oeste, para corregir posibles errores en la precisión de los puntos detectados. Así, la interpolación que queda es la que se observa en la figura 8.

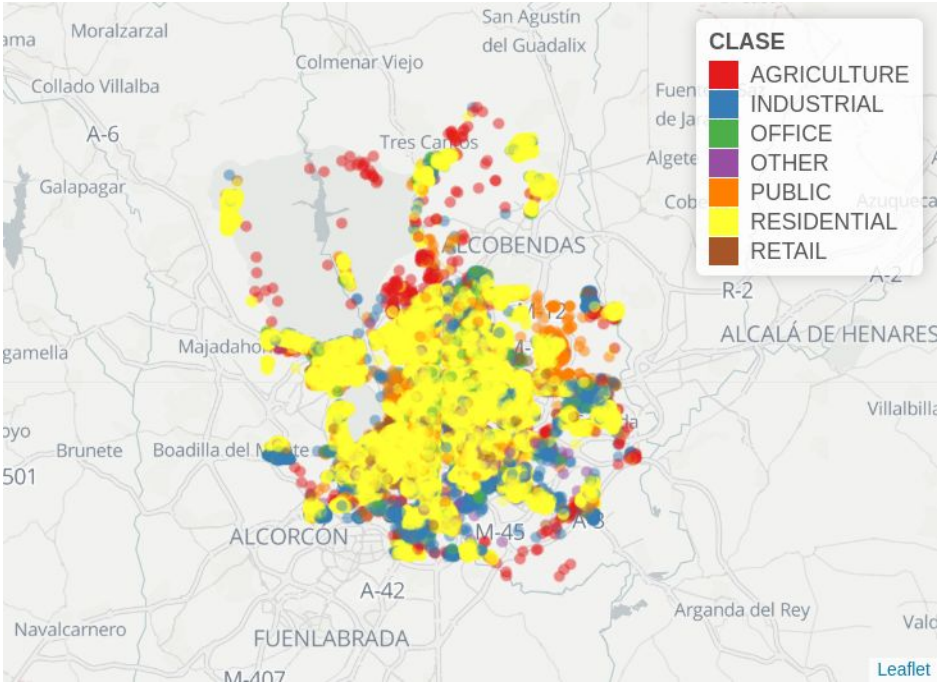


Figura 8. Puntos interpolados de entrenamiento.

Lo primero que se hace con esta interpolación es añadirle como variables la **distancia euclídea y la distancia en taxi** (distancia Manhattan) **a Sol**.

2.2. DATOS GEOGRÁFICOS

Una parte importante del trabajo consiste en, a partir de las latitudes y longitudes estimadas en el anterior punto, generar nuevas variables con las que enriquecer los datos. Estas variables vienen de distintas fuentes, y se explicarán a continuación, así como la definición de algunas variables y la forma de unión con los datos originales. Todas las variables de este sub-apartado 2 se obtienen con el archivo **programa_geovars.py**, como se explicará en detalle más adelante.

2.2.1. Variables de Nomecalles, Ayuntamiento de Madrid, Comunidad de Madrid, Consorcio Regional de Transportes de Madrid.

Nos descargamos todos los puntos de interés de https://www.madrid.org/nomecalles/, y a raíz de estos, se generaron variables principalmente de dos tipos:

- **Puntos de interés "x" por región**: se diseña una región de 2km alrededor de cada punto, y se cuenta el número de coincidencias de **cada variable de Nomecalles (unas 83)** en esa región. Así, cada punto tiene el número de bocas de metro, museos, comercios, etc. que hay en 2 km. También se añaden otros puntos, como el **número de paradas de autobús** en ese rango, datos que sacamos de https://data-crtm.opendata.arcgis.com/.

- **Distancia a puntos de interés**: se obtienen las distancias euclídeas a ciertos puntos como el **metro o el cercanías** más cercano, la **parada de bus**, el establecimiento de **Zara** más cercano, el de **Zara HOME,** entre otros.

- Otras variables añadidas son el **tráfico** en la hora punta del tramo más cercano en el que se tienen datos del tráfico (así como ésta hora punta y la distancia a ese tramo), o la **zona metropolitana** a la que pertenece (ambos obtenidos a través de la página web del Ayuntamiento de Madrid: https://datos.madrid.es/portal/site/egob/).

2.2.2. Otras Variables Relacionadas con la Latitud y Longitud

A partir de la latitud y longitud estimadas también se obtuvo el **código postal** asociado a estas, y de ahí se agregó el área de los mismos, introduciéndola como variable. Los códigos postales se utilizaron posteriormente para generar variables demográficas, como se explica en el apartado 1.2.3.

Además, se dibujaron los shapefiles de la **Castellana**, la **M-40** y la **M-30**, y se calcularon **las distancias de cada punto a cada uno de estos lugares**.

Se obtuvieron las zonas de protección acústica de https://geoportal.madrid.es/IDEAM_WBGEOPORTAL/index.iam, y se utilizaron para generar dos variables: el **ruido en la zona** y la **distancia al ruido**. Como había muchos puntos que estaban muy lejos de las zonas de protección acústica, se diseñó un listón de 0.016 grados (en latitud o longitud) observando el histograma; los puntos cuya distancia es inferior al listón tienen el ruido que se refleja en la zona de protección acústica más cercana, mientras que los puntos que queden más lejos tendrán ruido_no_registrado como valor en esta variable.

Se aplicó el algoritmo de clustering K-Means sólo sobre la latitud y longitud estimadas escaladas, con 100 clusters. De esta forma se separa el espacio por puntos que están cerca. Posteriormente, se calcula la **distancia de cada punto al centroide del cluster al que pertenece**, y se añade esto como variable. Esto puede verse en la figura 9. Especialmente para aquellas parcelas que quedan fuera de la M-40, se espera que los lugares donde más concentración de parcelas haya es en aquellas zonas residenciales, quedando el retail, las oficinas y las zonas industriales o agrícolas más alejadas. Por lo tanto, es probable que los centroides se encuentren en zonas de alta concentración de viviendas residenciales, siendo la distancia de cada punto a este centroide una variable rica para separar los diversos tipos de uso de las parcelas.

La **altitud** se obtiene con ayuda de los datos de https://datos.madrid.es/, asociando a cada punto la altitud de la curva de nivel más cercana. También de esta fuente se saca la **calidad del aire**, la cual unimos en función de si los polígonos que dibujan la calidad del aire en las distintas partes de Madrid contienen o no cada uno de los puntos del dataset original.

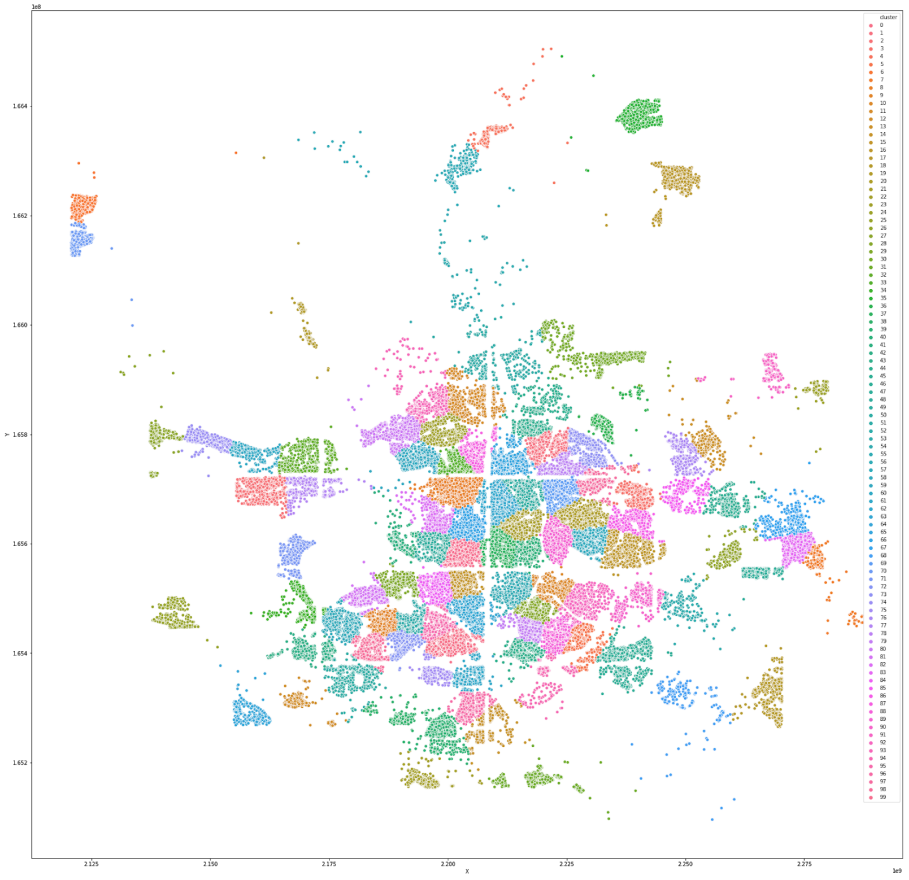


Figura 9. Clusters espaciales de los datos de entrenamiento.

2.2.3. Variables del INE, Catastro, IE, AEAT, Cybo

Se recolectaron, de diferentes fuentes, datos a nivel municipal, de código postal, y de sector censal, y se agruparon a nivel de código postal. Las siguientes variables sociodemográficas, económicas y de uso de suelo dan un perfil de cada área geográfica, que usamos para enriquecer nuestros datos. Estas fueron:

Del **Instituto Nacional de Estadística** (https://ine.es/):

- Edad media por Código Postal
- Porcentaje de Población menor de 18 años
- Porcentaje de Población mayor de 65 años
- Media de Personas por Hogar
- Población a nivel de Sector Censal
- Porcentaje de Crecimiento (o Reducción) Poblacional en 2015
- Renta Media por Persona por Sector Censal
- Renta Media por Hogar por Sector Censal

De la **Dirección Nacional de Catastro** (http://www.catastro.minhap.es/):

- Área de Suelo por tipo de uso, por código postal. Con esta información, agrupada por código postal, se sacaron las siguientes proporciones de uso de suelo:
    - Suelo Urbano
    - Suelo Agropecuario
    - Espacios Culturales
    - Espacios Deportivos
    - Espacios Industriales
    - Espacios de Oficinas
    - Espacios Públicos
    - Espacios Residenciales
    - Espacios dedicados a Retail
    - Espacios dedicados a Servicios de Salud
    - Otros espacios

Del **Instituto de Estadística de Madrid** (http://www.madrid.org/iestadis/):

- Porcentajes de Población Soltera, Casada, Viuda, Separada y Divorciada
- Proporción de Hogares Unipersonales
- Población ocupada según sector económico, de donde se construyeron las siguientes proporciones de ocupación por código postal:
    - Porcentaje ocupado en Agricultura
    - Porcentaje ocupado en Pesca
    - Porcentaje ocupado en Minería
    - Porcentaje ocupado en Manufactura
    - Porcentaje ocupado en Distribución de Gas, Electricidad o Agua
    - Porcentaje ocupado en Construcción
    - Porcentaje ocupado en Talleres de Reparación
    - Porcentaje ocupado en Hostelería
    - Porcentaje ocupado en Transporte
    - Porcentaje ocupado en Servicios Financieros
    - Porcentaje ocupado en Inmobiliarias
    - Porcentaje ocupado en Administración Pública
    - Porcentaje ocupado en Educación
    - Porcentaje ocupado en Servicios de Salud
    - Otras ocupaciones
- Porcentaje de Casas con Servicio Doméstico
- Índice de Masculinidad

- Índice de Reemplazamiento
- Índice de Dependencia
- Porcentaje de Analfabetismo
- Índice de Desempleo (Paro)

Para completar valores ausentes en códigos postales para los que las fuentes pasadas no tenían información, se utilizaron además las siguientes fuentes:

De la **Agencia Estatal De Administración Tributaria** (https://www.agenciatributaria.es/):

- Se obtuvieron datos de Renta Media y Renta Media por Hogar

De **Cybo** (https://xn--cdigos-postales-vrb.cybo.com/españa/):

- Se obtuvieron datos demográficos como:
    - Índice de Masculinidad
    - Índice de Reemplazamiento
    - Índice de Dependencia
    - Población por código postal

Para aquellas variables que no estaban disponibles a nivel de código postal, sino a nivel municipal o por sectores censales, se han cruzado los códigos postales que intersectan esas áreas, y se han calculado de manera ponderada por código postal; y para cualquier otro dato faltante que quedara, se utilizó la media en el municipio como imputación. Estos procesos no aparecen reflejados en el código porque los datos de las distintas fuentes se estandarizaron y compilaron en una hoja de cálculo, y se exportaron al fichero **vars_censo_codigo_postal_def.csv .**

En la figura 10 se puede ver de forma muy resumida las fuentes y el proceso de extracción de variables relacionadas con la posición geográfica.

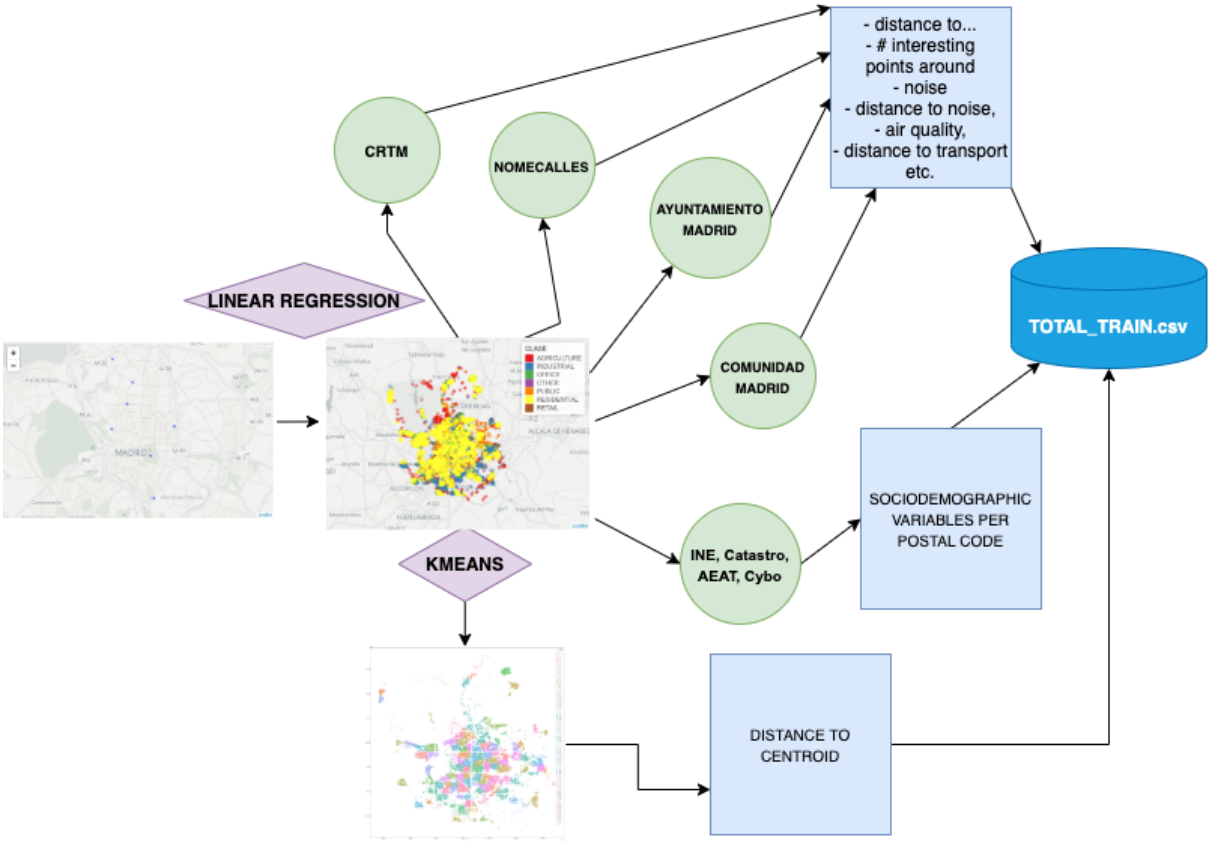


Figura 10. Diagrama de los procesos que forman nuestro Feature Generation.

3. FEATURE ENGINEERING & PREPROCESSING

Una vez se tienen todas las variables de programa_geovars.py, se procede a preprocesar los datos antes de modelizar. El código perteneciente a esta parte está en **preprocessing.py**. Una parte importante de este preprocesado es la generación de nuevas variables a partir de las existentes.

Se calculan variables como la **media de cada uno de los canales de color**, el **rango interdecílico** de estos, y se aplica **PCA sobre los deciles de todos los canales**, quedándonos con 3 componentes como variables. Esta práctica también está inspirada por los métodos revisados de Remote Sensing, ya que la descomposición de los canales de color puede reportar componentes que resalten otros aspectos de la imagen, como el brillo.

Se calcularon también múltiples índices utilizados en remote sensing para detectar zonas de alta vegetación, como el **NDVI, ExG, SAVI, MSAVI, BAI**, y otros diez, cuyo

cálculo se puede ver en preprocessing.py. Estos índices se calculan usualmente utilizando los canales de color. Tras muchas pruebas (seleccionando sólo los más relevantes, calculándolo sobre cada cuartil, sobre la media, sobre la mediana, etc) se decide descartar estas variables, pues empeoran el rendimiento de los modelos, además de ralentizar el entrenamiento. También se descartó utilizar la proyección **YUV** (https://es.wikipedia.org/wiki/YUV) de los canales de color, pues también distorsionaba los resultados de los modelos.

Las variables **CADASTRALQUALITYID** y **MAXBUILDINGFLOOR** se pasan a numéricas, en rangos de 1 a 12 en el primer caso y de 1 a 25 en el segundo. Además, para las variables **CONTRUCTIONYEAR**, **distance_to_transporte** y **GEOM_[...]**, se calculan variables de **"vecinos"**, es decir, se calcula la diferencia entre el valor de estas variables en cada punto y la mediana de los puntos en un rango de 7 km.

Se sacan variables espaciales, utilizando **X** e **Y**. Así, se calcula por ejemplo una proyección en tres dimensiones: **GEO_X, GEO_Y** y **GEO_Z.** También se realizan **rotaciones de las coordenadas 90 y 180 grados**, y se desarrolla el **polinomio de grado dos de ambas**.

Por último, se detecta que las variables **GEOM_[...]** y el **AREA** son algunas de las variables que más impacto tienen en los modelos, por lo que se deciden diseñar variables de interacción entre estas.

Como parte del preprocesado, se limpian los valores ausentes con Random Forest, utilizando la librería missingpy. Finalmente, se escalan todas las variables numéricas con el Z-Scaler, y se calcula la **densidad de población del código postal**.

Para las variables categóricas (código postal, zona metropolitana, ruido y calidad del aire), se hicieron distintas pruebas con la librería category_encoders. De entre todos los algoritmos de codificación de variables categóricas que se probaron, se escogió el CatBoostEncoder, que las transforma en numéricas de forma similar a como hace el CatBoost (de hecho es una de sus principales innovaciones).

Como puede apreciarse, se probaron muchas combinaciones de variables de distintos tipos, tratando de explotar la información contenida en los colores de la imagen (aplicando técnicas inspiradas en remote sensing), pero también en la geografía en la que esa imagen está contextualizada, sus características expresadas en términos de

distancias a lugares, de número de puntos “interesantes” en la zona, calidad del aire en esa zona, etc. que hacen diferente a esa parcela de otras.

Estas variables se probaron mediante un proceso iterativo, en el que la modelización y la búsqueda constante de nuevas variables fueron en paralelo. De esta forma, se utilizaba la información extraída a partir del análisis de los modelos para identificar posibles buenas nuevas variables, que pudieran ayudarles en sus debilidades, así como para descartar otras.

4. MODELIZACIÓN

Aunque se han realizado múltiples pruebas con arquitecturas muy diferentes y formas de aprendizaje distintas, en este apartado vamos a referirnos sólo a la arquitectura de entrenamiento final y a las partes que la forman, con el fin de aliviar la complejidad del texto.

4.1. PROCESO DE MODELIZACIÓN FINAL

En primer lugar, se entrenaron distintos modelos de Boosting y Bagging utilizando las librerías imbalanced-learn, scikit-learn y scikit-optimize. Los modelos que utilizamos para esta primera fase son: **LGBMClassifier, XGBClassifier, RandomForestClassifier, ExtraTreeClassifier, BalancedBaggingClassifier con HistGradientBoosting(s) como estimadores base y BalancedRandomForest**.

Estos dos últimos pertenecen a la librería imbalanced-learn. Excepto éstos, al resto de modelos se les incluye en un Pipeline de imbalanced-learn, cuyo primer paso es siempre un RandomUnderSampler. De esta manera, nos aseguramos que los modelos no sobre-ajusten a la clase mayoritaria. Como estrategia de muestreo en RandomUnderSampler utilizamos el porcentaje de reducción de la clase mayoritaria que nos resultó óptimo tras múltiples iteraciones de una búsqueda bayesiana de estos parámetros. Esto pertenece a fases previas del modelizado, en las que diseñamos Pipelines con los modelos a los que les incluímos un submuestreo (ver **models.py**), cuyos parámetros también se optimizaron con BayesSearchCV (de scikit-optimize), utilizando la métrica f1-macro. El valor óptimo medio, tras varias pruebas, fue de 0.11.

Además de hacer undersampling, se les pone el parámetro class_weights='balanced', de tal manera que el peso de cada observación en la función de pérdida sea inversamente proporcional al número de observaciones de esa clase; esto ayuda al modelo a centrarse más en las clases minoritarias, más difíciles de distinguir por falta de muestra.

Posteriormente, se analizaron los resultados de cada uno de esos modelos, en base a las métricas: accuracy, f1-macro, f2-macro, f05-macro. Después, se usaron estos modelos para entrenar un **StackingClassifier**. Se probaron diferentes combinaciones de modelos, y se volvió a utilizar la búsqueda bayesiana de Scikit-Optimize para encontrar los mejores parámetros del final_estimator con esa arquitectura de Stacking. Como estimadores finales se probaron tanto LogisticRegression como LGBMClassifier, ambos con RandomUnderSampler primero.

Una ventaja de esta arquitectura es que cada modelo base del StackingClassifier habrá visto unos datos de RESIDENTIAL, la clase mayoritaria, distintos, por el RandomUnderSampler. Esto aumenta la diversidad de la población de modelos con los que construir el Stacking, propiciando la generalización del método.

En la figura 11 se puede observar un pequeño resumen del proceso de modelización final. La tarea final es decidir un modelo de Stacking en base al rendimiento obtenido en el proceso de optimización bayesiana.

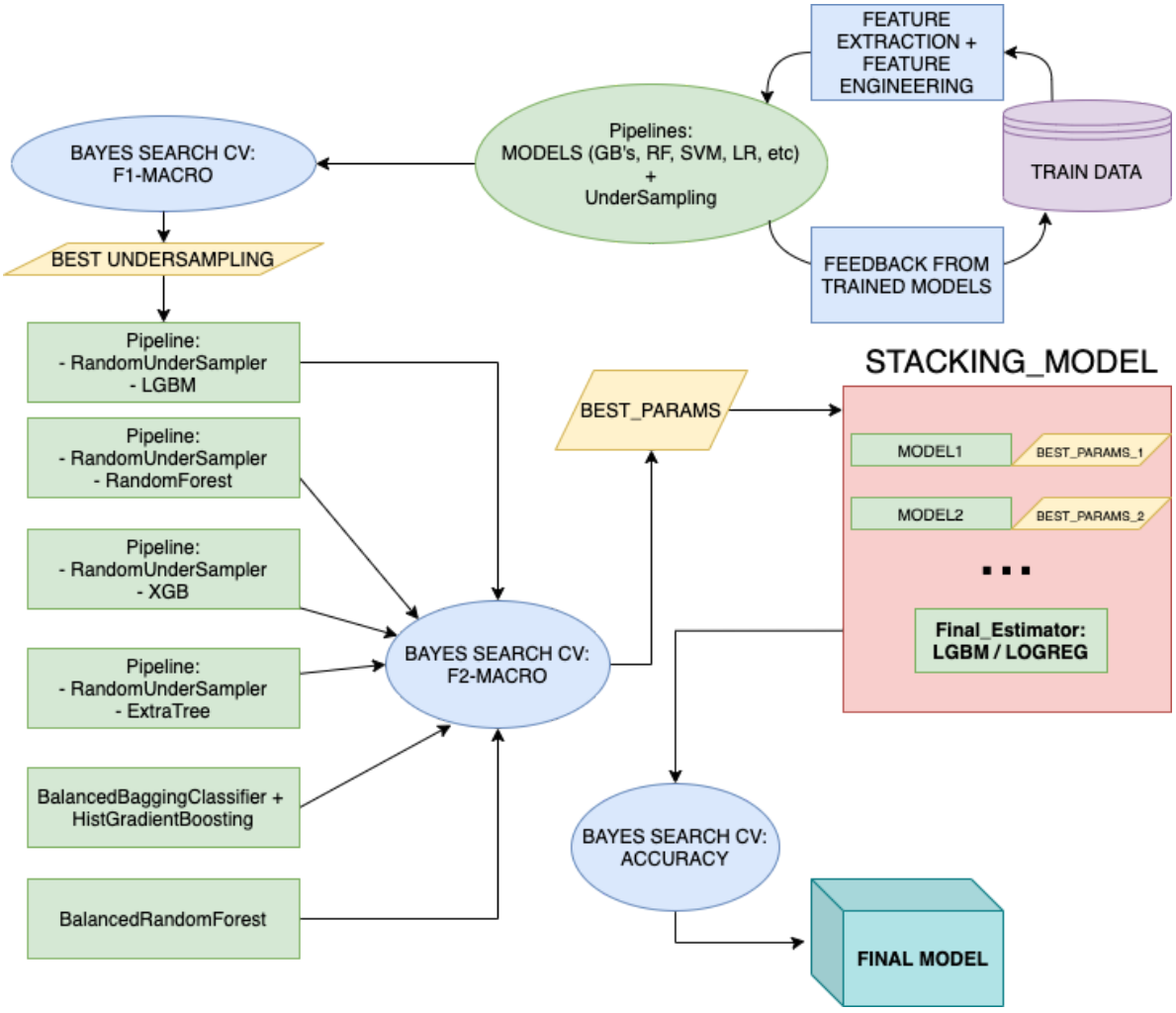


Figura 11. Resumen de los procesos de modelización

En todo el proceso de modelización hemos utilizado la herramienta **MLFlow**, que permite grabar experimentos, dentro de los cuales se pueden almacenar métricas, gráficos en png, incluso archivos binarios con los checkpoints de los modelos. Esto fue esencial para el flujo de aprendizaje que se refleja en la parte superior del diagrama.

En la figura 12 se observa claramente la mejora progresiva de los modelos según avanzó la competición, pues éstos comenzaron con valores de f1 y f2 (macro ambas) en torno a 0.35 y al final estaban por encima siempre de 0.6 en ambas métricas (siempre en un subconjunto de test o validación).

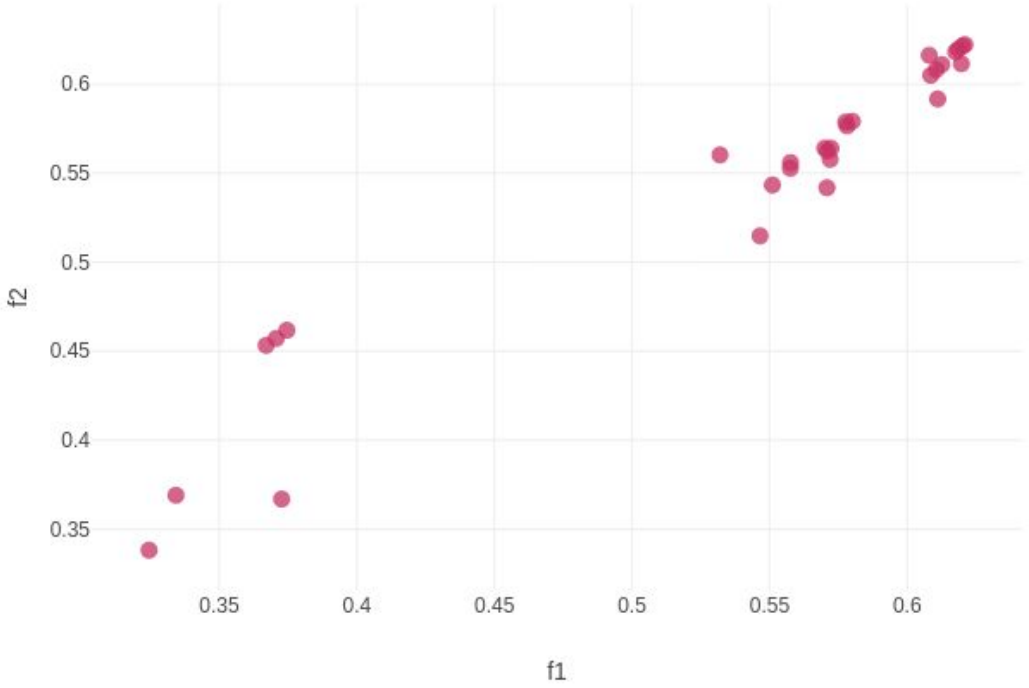


Figura 12. F2-macro vs. F1-macro de los experimentos de MLFlow.

Otra utilidad importante de esta herramienta son los gráficos de interacción de y los de parámetros vs. métricas, como se ve en las figuras 13 y 14. Éstos nos han permitido mejorar los espacios de búsqueda de los parámetros de los modelos, "ayudando" al proceso de optimización bayesiana:

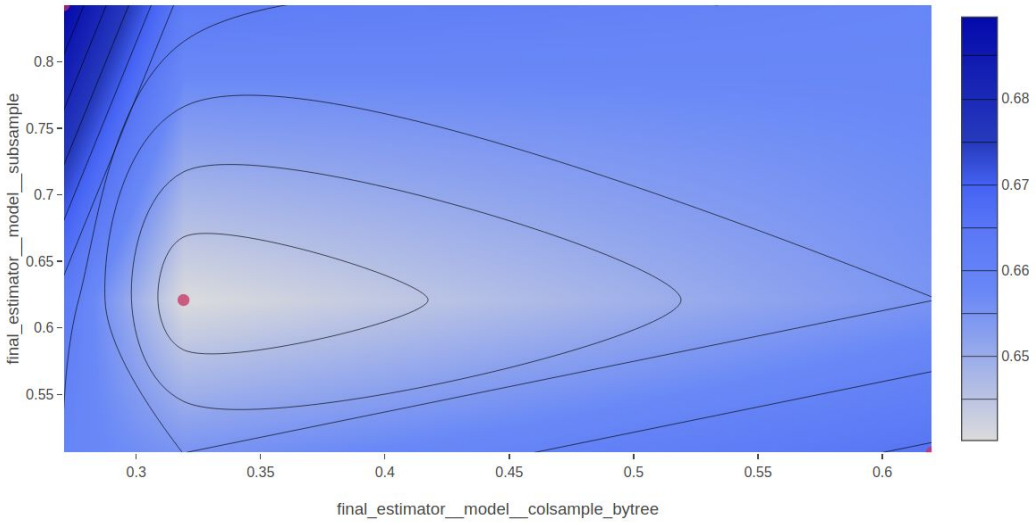


Figura 13. Relación de los parámetros colsample_bytree y subsample del final_estimator de uno de los modelos de Stacking probados.

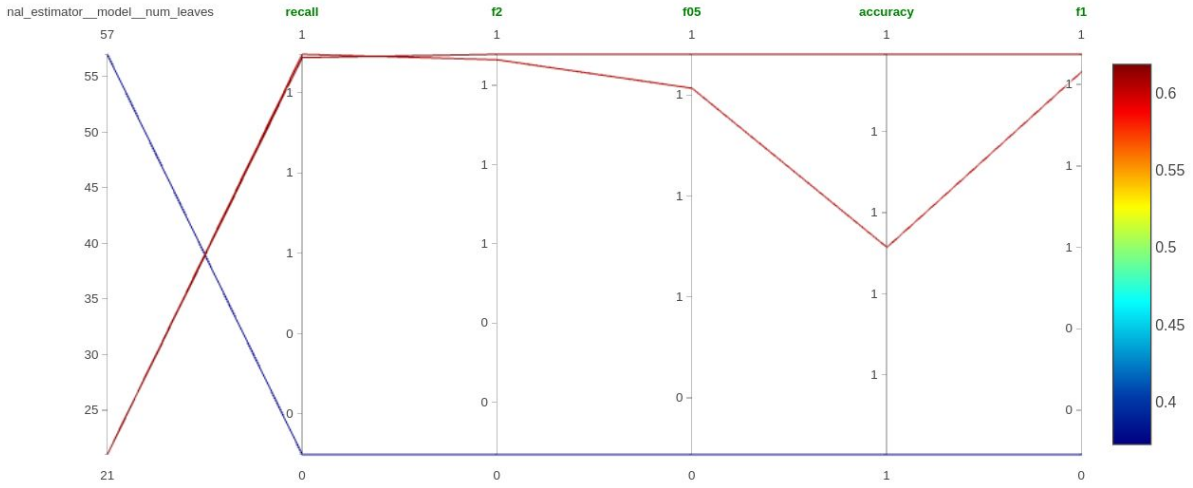


Figura 14. Distintas métricas para diferentes valores del parámetro n_leaves del final_estimator de uno de los modelos de Stacking probados.

4.2 OTROS ESFUERZOS REALIZADOS

Se probaron otros modelos para incluir al Stacking, como un MultiLayerPerceptron cuya arquitectura fue optimizada con una búsqueda bayesiana, pero sus resultados no fueron satisfactorios y se descartó para el modelo final.

Además, se utilizaron las librerías SHAP y ELI5 para analizar el impacto de las variables de los modelos. Aunque los modelos de árboles, como son la mayoría de los que hemos usado, pueden explicarse parcialmente por el feature_importance, estas librerías ofrecen alternativas distintas para explicar el impacto de cada variable en el desempeño de los algoritmos, por lo que se hicieron pruebas con ellas para contrastar las intuiciones generadas por los feature importance plots. SHAP se utilizó al principio, pero se descartó posteriormente ya que resulta muy costoso de calcular con muchas variables, en favor de Eli5. Aunque la explicabilidad de los modelos no es el objetivo principal de este concurso, consideramos que es una buena forma de recibir feedback al respecto de las nuevas variables introducidas, y de ganar insights sobre los datos. Un ejemplo de los resultados de Eli5 se puede ver en el Anexo 2.

Igualmente, hemos intentado utilizar las imágenes del Sentinel II (ver Exploración Inicial) para obtener features desde éstas utilizando Deep Learning. Se probó tanto una arquitectura propia de red convolucional, como ajustar las últimas capas de modelos pre-entrenados (InceptionV3 y Xception https://keras.io/applications/), pero no se obtuvieron resultados satisfactorios y no se incluyeron en el modelo final. Entendemos que esto fue en parte porque las imágenes tienen una resolución de 10 metros por pixel, haciendo que muchas imágenes de lugares cercanos fuesen similares a pesar de tener clases diferentes. En la figura 15 podemos ver el nivel de zoom que se tiene al generar imágenes de 75x75 pixeles, el mínimo aceptado por los modelos pre-entrenados.

Figura 15. Ejemplos de Imágenes con las que se entrenaron los modelos de redes.

5. SELECCIÓN DEL MODELO FINAL

Finalmente nos decantamos por un modelo de Stacking formado por: RandomForestClassifier, LGBMClassifier, XGBClassifier y ExtraTreeClassifier. Aunque todos los modelos de Stacking tuvieron métricas parecidas, y fue muy difícil tomar la decisión final, se tuvieron en cuenta muchos parámetros.

Por un lado, se tuvo en cuenta el accuracy obtenido en un subconjunto de test, que fue el mismo para todos los modelos, valorando también las métricas f1-macro y f2-macro (para ver la capacidad de generalización del modelo, tratando de no sobre-ajustar). Se consideró además el accuracy medio en el experimento con cross-validation (con optimización bayesiana), por ser más fiable que la métrica en un único conjunto de test. La desviación estándar entre los 3 folds de este último accuracy también se valoró, pues tratamos de encontrar un modelo que tenga buen rendimiento en general y de forma poco variable.

Otros elementos más subjetivos de la evaluación fueron los relacionados con el análisis detallado de las matrices de confusión. En todos los aspectos valorados, éste es el modelo que mejor rendimiento indicó tener, y es, por tanto, el que utilizamos finalmente. Para ver los parámetros concretos del final_estimator del modelo, así como de los estimadores base, ver **models.py**, en el diccionario FINAL_MODELS, con el nombre “StackingAlex1”.

Resulta interesante que los cuatro modelos que forman este Stacking son los cuatro, de todos los probados, que mejor rendimiento tuvieron en sus entrenamientos individuales, en términos de f1-macro y f2-macro.

6. RESUMEN DEL CÓDIGO

Para ayudar al lector a comprender el funcionamiento completo de nuestro proyecto, así como para facilitar la reproducibilidad de nuestros resultados, realizaremos aquí un breve resumen del código que hemos utilizado, los programas que lo forman y su principal funcionalidad. Antes de comenzar a ejecutar el código, es importante tener Anaconda instalado e instalar en entorno que se encuentra en **environment.yaml**. Por la parte de R, es importante instalar los paquetes: **leaflet, tidyverse y plotly**.

La interpolación de la latitud y la longitud se encuentra en el archivo **InterpolacionLatLon.R**. En él, se diseña un dataframe de R con las X e Y de los puntos identificados con la ayuda de Plotly, así como con las latitudes y longitudes reales de esos lugares aproximados. Con estos datos, se realiza el proceso descrito en el apartado 1.1. Esto nos crea los ficheros dataset_train.csv y dataset_test.csv, necesarios para los siguientes pasos.

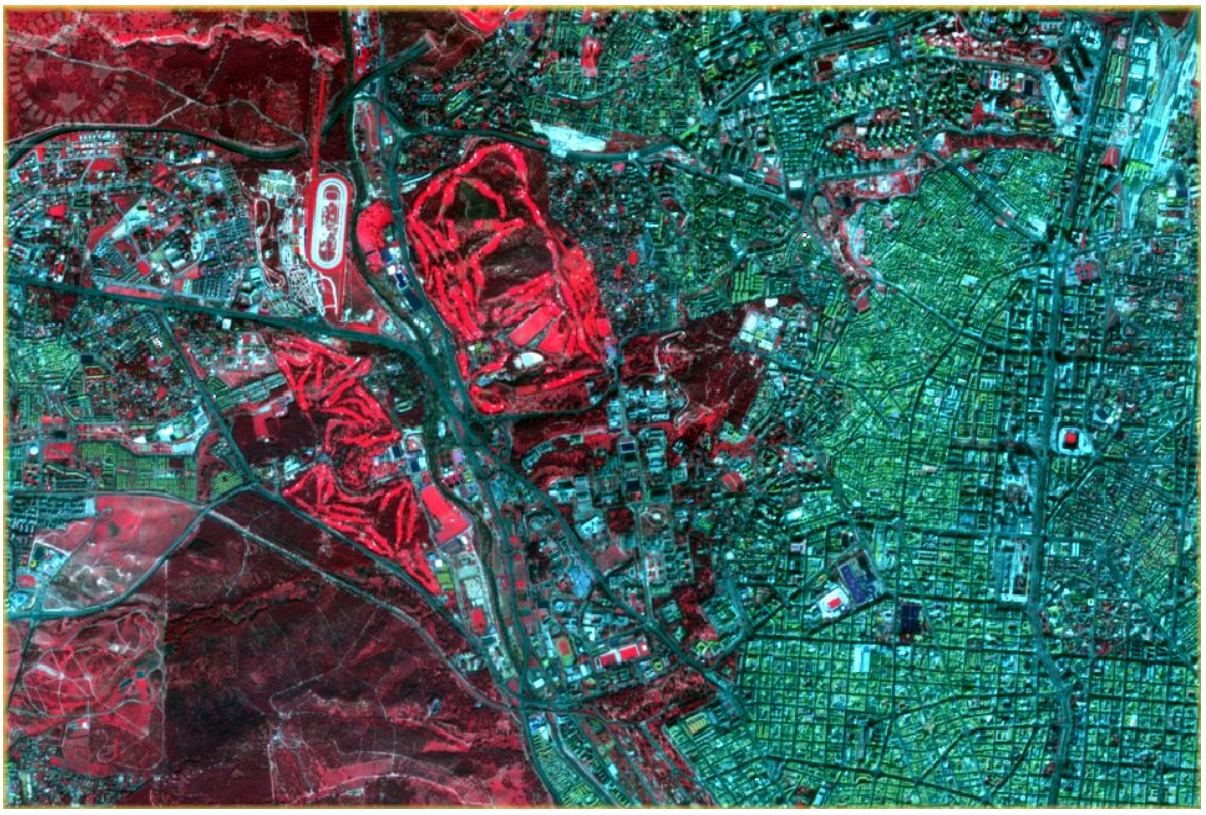
La mayoría de variables geográficas del apartado 1.2. se calculan con **02-programa_geovars.py**, excepto alguna excepción que necesita del escalado de todas las variables, que se calcula en **preprocessing.py**. Para mayor comprensión del funcionamiento interno del código, las funciones principales de **02-programa_geovars.py** están ampliamente comentadas. Este programa devolverá, según el parámetro de la variable global MODE, TOTAL_TRAIN.csv o TOTAL_TEST.csv, con todas las variables geográficas añadidas. Hay que ejecutarlo dos veces, una con MODE='train' y otra con MODE='test'.

Para entrenar los modelos se utilizan los programas **model_trainer_refactor.py** y **train_stacking.py**. Ambos programas hacen uso de **models.py**, archivo en el cual se encuentran los modelos finales usados, algunos de los intermedios utilizados por éstos, y las clases necesarias para construir todos los Pipelines probados. Además, en todos estos programas se utilizan las funciones de preprocesado de **preprocessing.py**, archivo en el cual se encuentra lo explicado en la sección 2 (FEATURE ENGINEERING & PREPROCESSING).

Una vez se tienen los modelos entrenados y se han encontrado los parámetros óptimos (ya están puestos en los diccionarios declarados en **models.py**), se procede a sacar el test con **03-sacar_test_stacking.py**.

7. ANEXOS

Anexo 1: Imagen en “Falso Infrarrojo” extraída del SNAP. El canal Rojo es el infrarrojo, mientras que los canales rojo y verde pasan a ser verde y azul, respectivamente. Usada para identificar las estructuras con mayor reflectividad infrarroja.



Anexo 2: Pesos de Eli5 para las variables de uno de los modelos entrenados. Estos pesos se utilizan para identificar qué variables son las que más están afectando a las predicciones del modelo.

Weight	Feature
0.0192	educapr.shp
0.0181	GEOM_R1_x_area
0.0170	GEOM_R2_x_GEOM_R3
0.0166	distance_to_centroid
0.0165	GEOM_R1_x_GEOM_R2
0.0160	ExG
0.0144	AREA
0.0142	GEOM_R2_x_area
0.0135	poblacion_cp
0.0135	GEOM_R1_2
0.0134	NEIGHBORS_AREA
0.0127	GEOM_R2_2
0.0127	GEOM_R1_x_GEOM_R3
0.0126	GEOM_R3_x_area
0.0126	educapu.shp
0.0125	GEOM_R1
0.0123	dir2019a.shp
0.0118	CONTRUCTIONYEAR
0.0115	bancos.shp
0.0112	bocas.shp
0.0111	GEOM_R3_2
0.0111	GEOM_R4_x_area
0.0110	GEOM_R2
0.0107	ss_2018_etrs89.shp
0.0105	GEOM_R3_x_GEOM_R4
0.0102	GEOM_R3
0.0099	GEOM_R1_x_GEOM_R4
0.0098	Q_B_2_0_7