Detección automática de calles sin pavimentar en zonas urbanas del Conurbano Bonaerense mediante aprendizaje automático aplicado a imágenes satelitales multiespectrales

14/07/2023

Introducción

La falta de pavimentación en calles de áreas urbanas representa un problema importante en muchas ciudades de Argentina, trayendo consecuencias negativas para sus habitantes como dificultades en el tránsito vehicular, propagación de material particulado, acumulación de agua y aumento de enfermedades respiratorias y dermatológicas. Identificar de manera eficiente las calles sin asfaltar en zonas urbanas del país permitiría priorizar obras y asignar recursos de forma óptima para mejorar la calidad de vida de los ciudadanos.

Si bien los mapas de calles pavimentadas pueden generarse manualmente, esto requiere un trabajo de campo exhaustivo. Por ello, en los últimos años se han propuesto metodologías basadas en teledetección y aprendizaje automático para mapear características urbanas de forma automática a partir del análisis de imágenes satelitales. En particular, el uso de imágenes multiespectrales de mediana y alta resolución espacial, como las provistas por el satélite Sentinel-2 de la Agencia Espacial Europea permite identificar patrones urbanos complejos[1], y, utilizado en conjunto con algoritmos de aprendizaje automático, permite obtener resultados a mayor escala[2].

En este trabajo se propone un enfoque basado en árboles de decisión para la detección automática de calles sin asfaltar en el Conurbano Bonaerense a partir del análisis de imágenes satelitales Sentinel-2. El objetivo es desarrollar un modelo de aprendizaje automático capaz de identificar vías de circulación no pavimentadas en zonas urbanas, como primer paso hacia la generación de mapas actualizados que puedan apoyar la planificación y priorización de obras públicas. La disponibilidad de conjuntos de datos etiquetados junto con las capacidades de las modernas arquitecturas de *machine learning* brindan una oportunidad prometedora para avanzar en la solución de esta problemática urbana que afecta a millones de habitantes.

Metodología

Área de estudio

El área de estudio seleccionada para este trabajo es la zona del partido de Almirante Brown en la Provincia de Buenos Aires, Argentina. Esta región fue elegida por presentar una alta heterogeneidad en términos de pavimentación de calles en distintos partidos y localidades, representando así un desafío interesante para la detección automática.

---

[1] Mapa de Fragmentación Urbana en la Provincia de Córdoba (2020). IDECOR, Gobierno de la Provincia de Córdoba, octubre 2022.

[2] Piumetto, A., Monzani, L., & Córdoba, M. (2019). Valuación masiva de la tierra urbana mediante inteligencia artificial. El caso de la ciudad de San Francisco, Córdoba, Argentina, 2019.

Herramientas

Sentinel-2

Sentinel-2 es un satélite especial que fue lanzado al espacio en 2015 por la Agencia Espacial Europea (ESA) con el objetivo de estudiar y monitorear la Tierra en tiempo real. Este satélite cuenta con un instrumento de imagen multiespectral que puede capturar imágenes de la superficie terrestre con una resolución media, de 10 metros.

Las imágenes que toma Sentinel-2 se capturan en diferentes longitudes de onda de luz, lo que permite ver cosas que el ojo humano no puede ver. Por ejemplo, el satélite puede capturar imágenes en el espectro visible de luz, que es lo que nuestros ojos pueden ver, pero también puede capturar imágenes en el espectro infrarrojo, que es invisible para nosotros.

En este estudio, se seleccionaron imágenes censadas durante el año 2023, entre el primero de enero y el 30 de marzo, con baja nubosidad. Las imágenes fueron descargadas a través de la plataforma Google Earth Engine.

Desarrollo[3]

Google Earth proporciona una interfaz de programación de aplicaciones (API) que vincula las imágenes tomadas por Sentinel-2 y las pone a disposición en Python.

API Python

Esto permite, dado un polígono, extraer las imágenes satelitales en las bandas espectrales disponibles:

python

```
# Se autentica y inicializa la API de Google Earth Engine (“ee”)

import ee

ee.Authenticate()
ee.Initialize()
```

Luego, es necesario proporcionar un polígono en formato GeoJSON. Si bien es posible cargarlo desde un archivo predeterminado, la herramienta en línea provista por GeoJSON.io permite generar manualmente el GeoJSON requerido simplemente dibujando en el área deseada.

[3] El desarrollo presentado si bien contiene código, debe ser modificado y adaptado a cada situación.

Por ejemplo, utilizando la herramienta GeoJSON.io se podría generar manualmente un polígono alrededor del Aeropuerto Internacional de Ezeiza, como se muestra en la siguiente captura:

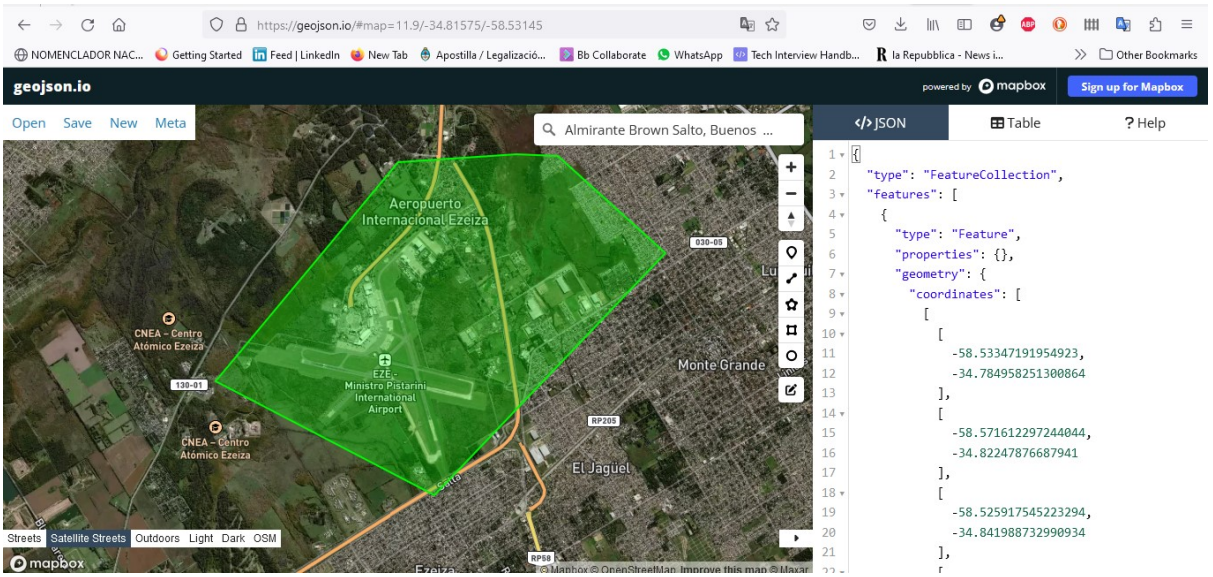


Imagen 1: Extracto de pantalla de la web para determinar manualmente un polígono.

Como alternativa, también es posible utilizar un archivo GeoJSON pregenerado. En este ejemplo, usaremos el archivo `alte_brown.geojson` que contiene el polígono de la localidad de Alte Brown.

```
geoJSON= # Aquí se ubica el json cargado.

aoi = ee.Geometry.Polygon(geoJSON['features'][0]['geometry']['coordinates']) # Se establece el "área de interés" con el argumento de una lista de listas que contienen las coordenadas que constituyen el polígono

# La biblioteca permite establecer un filtro de nubes
 CLOUD_FILTER = 15

#Se genera la solicitud de imagen, luego esta solicitud puede descargarse o simplemente mostrarse
ffa_db =
ee.Image(ee.ImageCollection('COPERNICUS/S2_SR').filterBounds(aoi).filterDate(ee.Date('2023-01-01'),
ee.Date('2023-03-30')).filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE',CLOUD_FILTER)).first().clip(aoi))

 # La librería permite obtener la imagen para mostrar directamente, sin descargarla
 import IPython.display as disp

 # Aqui, TCI indica "True Color Image", es decir "Imagenes de Color real".
 # Está conformado por los rojos (_R), verdes (_G) y azules (_B).
```

```
url = ffa_db.select('TCI_R','TCI_G','TCI_B').getThumbURL()
disp.Image(url=url, width=800)
```

Mapa 1. Mapa del polígono correspondiente a Alte Brown, utilizando las bandas True Color Image (TCI), B4, B3,B2 de Sentinel 2

Debido a limitaciones de tamaño, si el área es muy grande se debe dividir en secciones más pequeñas. Para ello se desarrolló una función (`create_grid_polygon`) que genera una grilla de polígonos a partir de las coordenadas del área total. Para esto, primeramente se deben importar las clases Polygon y Point del módulo `geometry` de la librería `shapely` en Python.

Esta función tiene como objetivo dividir un polígono definido por una serie de coordenadas en polígonos más pequeños organizados en filas y columnas. Esto es útil debido a las

limitaciones de tamaño de imagen impuestas por la API de Python y Google Earth. Si un polígono es demasiado grande para ser extraído en una sola solicitud, se puede dividir en fragmentos más pequeños.

La función toma como entrada un polígono representado por una lista de coordenadas y lo divide en un número específico de filas y columnas (`grid_rows` y `grid_columns`, respectivamente). Es importante tener en cuenta que esta función funciona mejor cuando el polígono de entrada es un cuadrado o rectángulo.

El resultado de la función es una lista de coordenadas que representa los polígonos resultantes organizados en una cuadrícula de tamaño `grid_rows` * `grid_columns`.

Parámetros:
- `polygon_coords`: Lista de coordenadas que definen el polígono completo.
- `grid_rows`: Número de filas deseadas para dividir el polígono.
- `grid_columns`: Número de columnas deseadas para dividir el polígono.

```
from shapely.geometry import Polygon, Point
def create_grid_polygon(polygon_coords, grid_rows, grid_columns):

# Obtener el cuadro delimitador (bounding box) del polígono de entrada.
min_x, min_y, max_x, max_y = (
min(x for x, y in polygon_coords),
min(y for x, y in polygon_coords),
max(x for x, y in polygon_coords),
max(y for x, y in polygon_coords)
)

# Calcular el tamaño de la celda de la cuadrícula basado en el número de
filas y columnas.
cell_width = (max_x - min_x) / grid_columns
cell_height = (max_y - min_y) / grid_rows

# Generar puntos de la cuadrícula dentro del cuadro delimitador.
grid_points = []
for row in range(grid_rows):
for col in range(grid_columns):
x = min_x + col * cell_width
y = min_y + row * cell_height
grid_points.append([x, y])

# Crear polígonos de la cuadrícula a partir de la ventana deslizante.
grid_polygons = []
for row in range(grid_rows - 1):
```

```
for col in range(grid_columns - 1):


# Crear los vértices del polígono de la ventana deslizante.
vertices = [
grid_points[row * grid_columns + col],
grid_points[row * grid_columns + col + 1],
grid_points[(row + 1) * grid_columns + col + 1],
grid_points[(row + 1) * grid_columns + col]
]

 # Verificar si el polígono de la ventana se intersecta con el polígono
de entrada.
window_polygon = Polygon(vertices)
if window_polygon.intersects(Polygon(polygon_coords)):


 # Obtener la intersección entre el polígono de la ventana y el polígono
de entrada.
intersection = window_polygon.intersection(Polygon(polygon_coords))
grid_polygons.append(list(intersection.exterior.coords))

return grid_polygons
```

Ahora, supongamos que queremos dividir él área anterior, en 4 partes:
En este caso, se usan las latitudes y longitudes máximas y mínimas para que el recorte sea rectangular.

```
 polygon_coords = [[-58.292471, -34.754401316999974],
[-58.292471, -34.909881],
[-58.447753, -34.909881],
[-58.447753, -34.754401316999974]]

grid_rows = 4
grid_columns = 4
```

Aquí obtenemos la lista de polígonos que efectivamente vamos a solicitar a GoogleEarth

```
grid_polygons = create_grid_polygon(polygon_coords, grid_rows, grid_columns)
```

Extracción

Se itera sobre cada polígono de la cuadrícula:
Cada imagen, si no es descargada, queda en una URL.

```
urls=[]
 for igx in range(len(grid_polygons)):
 "Proceso de Extracción"
 aoi = ee.Geometry.Polygon(grid_polygons[igx])
 CLOUD_FILTER = 15
ffa_db = ee.Image(ee.ImageCollection('COPERNICUS/S2_SR').filterBounds(aoi).filterDate(ee.Date('2023-01-01'), ee.Date('2023-03-30')).filter(ee.Filter.lte('CLOUDY_PIXEL_PERCENTAGE', CLOUD_FILTER)).first().clip(aoi))
'Se define la extracción de TrueColorImage
 url = ffa_db.select('TCI_R','TCI_G','TCI_B').getThumbURL()
 urls.append(url) # Se agrega al listado
```

Para cada sección se obtiene una imagen y se guarda la URL en una lista.

Se pueden visualizar las imágenes obtenidas:

```
disp.Image(url=urls[0], width=250)
disp.Image(url=urls[1], width=250)
disp.Image(url=urls[2], width=250)
disp.Image(url=urls[3], width=250)
```

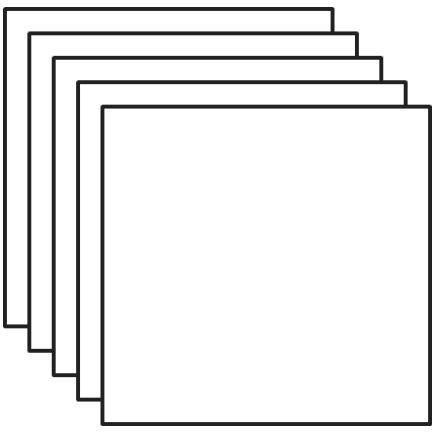
Mapa 2. Mapa del polígono correspondiente a Alte Brown, utilizando las bandas True Color Image (TCI), B4, B3,B2 de Sentinel 2. Fraccionado rectangularmente en 4 subsecciones.

De esta manera se obtuvieron imágenes satelitales por secciones para luego ser procesadas y analizadas.

Descarga

Se solicitaron las bandas espectrales B2, B3, B4, B8 y B11, necesarias para este análisis:

- B2 (azul): Captura información sobre la reflectancia de la luz azul en la superficie. Útil para identificar cuerpos de agua y objetos que reflejan luz azul de forma característica.
- B3 (verde): Captura información sobre la reflectancia de luz verde en la superficie. Sirve para caracterizar la vegetación y detectar cambios en la cobertura vegetal.
- B4 (rojo): Captura información sobre la reflectancia de luz roja en la superficie. Especialmente útil para evaluar salud vegetal y detectar cambios en el contenido de clorofila.
- B8 (infrarrojo cercano): Captura información sobre la reflectancia de infrarrojo cercano en la superficie. Sirve para identificar tipos de vegetación, evaluar salud de cultivos y detectar cambios.



- B11 (infrarrojo térmico): Captura información sobre la radiación térmica emitida por la superficie. Útil para estimar temperatura superficial y realizar análisis de temperaturas específicas.

```
from urllib import request
from zipfile import ZipFile

link = ffa_db.select('B2','B3','B4','B8','B11').getDownloadURL({'scale': 1,'crs': 'EPSG:4326','fileFormat':'GeoTIFF'})
```

1) La descarga, como es multiespectral, se descargan archivos comprimidos.

```
file = 'raster.zip' #como se va a llamar el archivo de salida
response = request.urlretrieve(link, file)
pathout = '../almte_brown/almte_brown/rasters'
```

2) Una vez descargado, extraer todo.

```
with ZipFile(file, 'r') as zObject:
zObject.extractall(path=pathout)
```

3) Una vez descomprimidos, se listan:

```
import os
raster_files = [os.path.join(pathout,i) for i in os.listdir(pathout) if ".tif" in i]
raster_files = [x for x in raster_files if "HUB.B" in x]
import rasterio
```

4) Se abre el primer raster (CAPA A):

```
blue_raster = rasterio.open(raster_files[0])
blue = blue_raster.read(1, masked=True)
out_meta = blue_raster.meta.copy()
out_meta.update({"count": len(raster_files)})
```

5) Se define el raster final:

```
filename=f'raster_test_{igx}.tif'
out_img = os.path.join(pathout,filename)
omit_deleted.append(out_img)
```

6) Se leen los demás rasters

```
file_list_o = [rasterio.open(i) for i in raster_files]
file_list = [x.read(1, masked=True) for x in file_list_o]
```

7) Se apilan los rasters y se guardan

```
with rasterio.open(out_img, 'w', **out_meta) as dest:
 for band_nr, src in enumerate(file_list, start=1):
dest.write(src, band_nr)
```

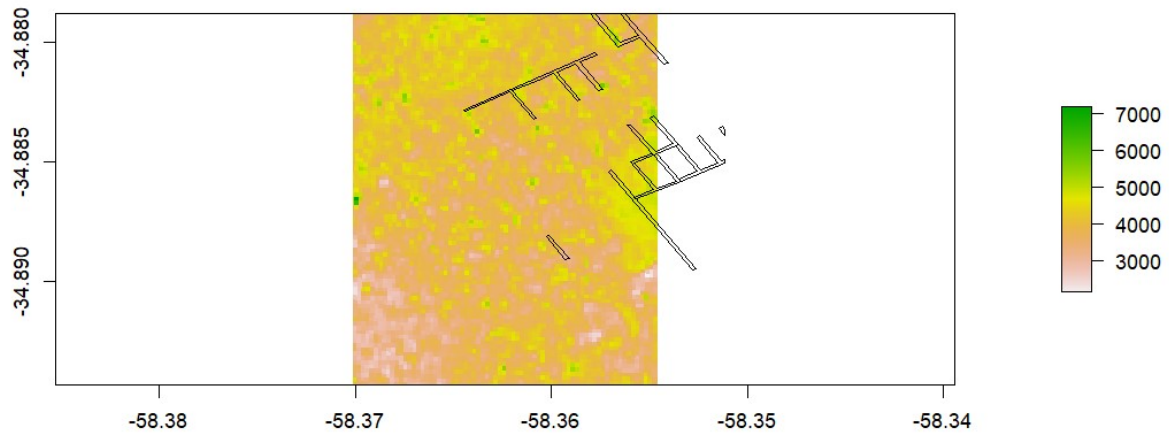
8) Se cierran los archivos abiertos.

```
for k in file_list_o:
k.close()
blue_raster.close()
for h in [x for x in raster_files if x not in omit_deleted]:
os.remove(h)
```

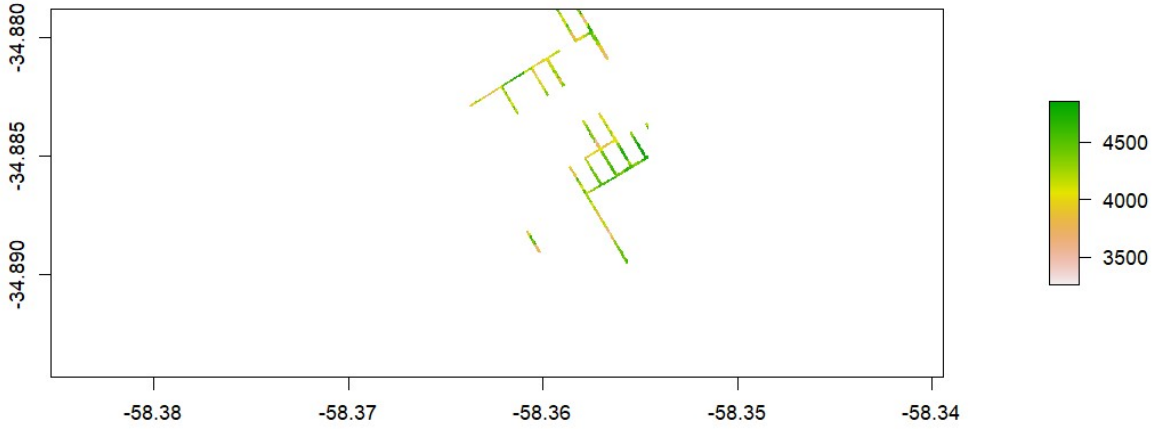
Ahora, después de este proceso, se encuentra la disponibilidad de los *rasters* con las capturas multiespectrales. El proceso de Python termina en esta instancia.

Esquemáticamente, el segundo proceso tiene los siguientes pasos:

- Requiere un relevamiento previo, una muestra de las áreas de interés a trabajar. Esto quiere decir que son necesarios polígonos asfaltados y no asfaltados. Gracias a este relevamiento previo, se construyó una base de datos de 322 polígonos con calles asfaltadas y 278 polígonos con calles sin asfalto o de tierra.
- Utilizar las vías de circulación para ver la intersección del valor del *ráster* en el lugar donde se encuentra la calle.

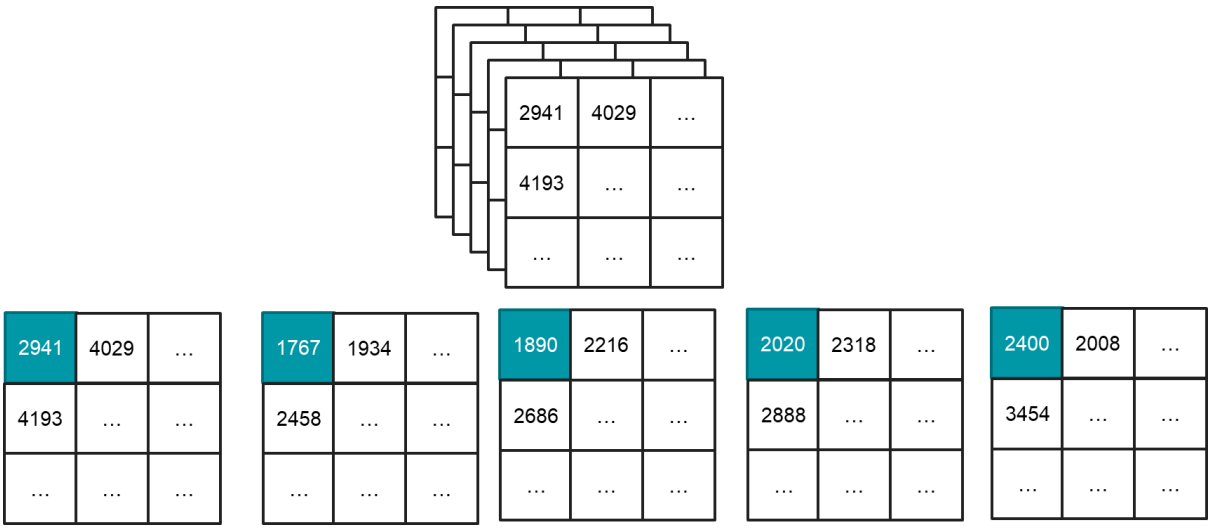


Mapa 3: En color: valores del Ráster para la capa B4. Líneas: Calles relevadas como sin pavimentar



Mapa 4: En color: Valor que toma el Raster para la capa B4 solo en las secciones donde se superpone con la calle relevada sin pavimentar.

- Los rasters anteriores se tabulan de acuerdo a la posición que ocupan:



Por ejemplo, el primer pixel, quedaría tabulado así:

B2	B3	B4	B5	B11
2941	1767	1890	2020	2400

Se le agrega la columna "**fl_no_asfaltada**", dependiendo la etapa del proceso. Si se están procesando los relevamientos pavimentados, **fl_no_asfaltada** tomará en esta instancia valores de 0. Si se están procesando los relevamientos no-pavimentados, el **fl_no_asfaltada** tomará valores de 1.

Una muestra de la tabla resultante sería:

B2	**B3**	**B4**	**B8**	**B11**	**fl_no_asfaltada**
2941	1767	1890	2020	2400	0
4193	2458	2686	2888	3454	0
4029	1934	2216	2318	3008	0
3665	1995	2248	2538	3230	0
4066	1764	1968	2288	3348	0
4112	2022	2114	2422	3312	0
4232	2272	2466	2746	3396	0
3942	1907	2376	2376	3304	0
3600	1860	2094	2084	2854	0

3472	1971	2184	2284	2728	0
3738	1725	1896	2070	3574	1
4214	3338	3558	3468	4480	1
3567	1910	2118	2396	3824	1
3969	2250	2506	2780	3864	1
4388	1916	2152	2416	3832	1
4670	2428	2668	2924	3878	1
3610	1716	1938	2146	3192	1
3849	2174	2390	2706	4128	1
4351	2590	2936	3048	3736	1
4654	1890	2198	2448	3690	1

Cuadro 1: Tabla resultante una vez realizado el proceso que tiene como resultado el Mapa 4. Para todos los polígonos relevados.

Entrenar un modelo de clasificación a partir de los datos tabulados.

A la hora de predecir, se repiten los primeros tres puntos anteriores y se le agrega:

- A partir de los datos tabulados, usar el modelo anteriormente entrenado para generar la predicción.
- Utilizar las coordenadas asociadas a cada predicción para encontrar la calle más cercana. Esto permite vincular la predicción de cada punto a una calle.
- En esta instancia se tiene un listado de calles y cada calle tiene una predicción correspondiente a cada punto que la compone. Para obtener un solo resultado por calle, se puede promediar el resultado de todos los puntos.

Por ejemplo, una calle tiene asociadas una predicción por cada punto que lo compone:

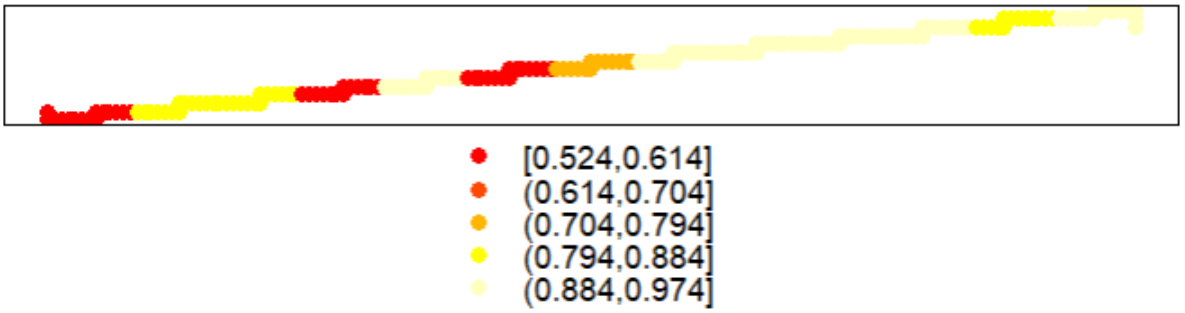
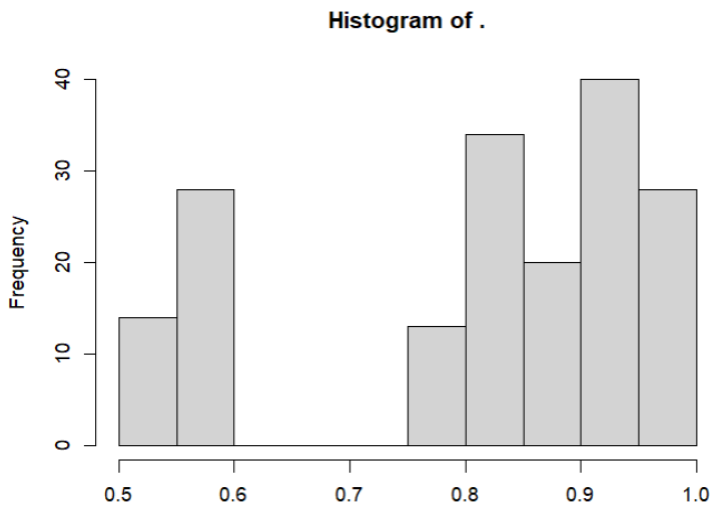


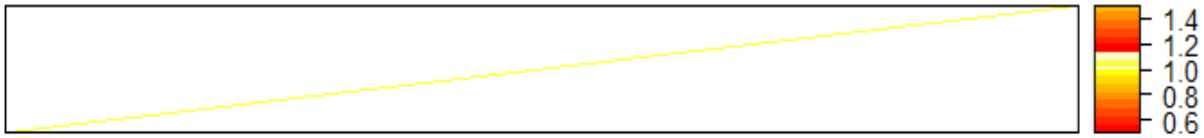
Imagen. Probabilidades asociadas a una sola vía de circulación.

Por lo tanto, cada calle tiene una distribución asociada:



Distribución de probabilidades asignadas a una calle. El promedio es 0,8130.

El promedio de esta calle es de 0.8130, por lo que es el valor asociado a esta calle. Luego, este valor puede ser 0 ó 1 según la decisión del analista.



Al promediar los valores de la distribución se obtiene un valor único para la calle.

Antes de iniciar el proceso en R, deben descargarse las vías de circulación: NomencladorVias

A continuación, en R, cargan varios paquetes para el análisis de datos espaciales. Luego se cargan dos conjuntos de datos de geojson, uno que representa calles no asfaltadas y otro que representa calles asfaltadas. Finalmente, se establece la proyección de ambos conjuntos de datos para que sea la misma que la proyección del conjunto de datos de calles del nomenclador.

```
library(tidyverse)
library(rgdal)
library(raster)
library(rgeos)
library(sp)
library(sf)
```

```
#Datos etiquetados
vias1 = readOGR('./data/vias_1.geojson')# contorno, área relevada como "no asfaltada"
vias0 = readOGR('./data/vias_0.geojson')# contorno, área relevada como "asfaltada"
vias = readOGR('../vias/vias.shp')# INDEC, nomenclador de vías de circulación


# Las proyecciones de los contornos deben tener la misma proyección de las vías del nomenclador
crs(vias1) = crs(vias)
crs(vias0) = crs(vias)
```

Se genera el *dataframe* para aquellas vías “no asfaltadas”

```
# Se busca la intersección entre el contorno y las vías de circulación del INDEC.
cropped<-gIntersection(vias1,vias )
# Se lee el raster asociado
sat = stack('./raster_folder/raster_1.tif')
my_extent_poly <- as(extent(sat), "SpatialPolygons")
#Se obtienen los valores rasters que están superpuestos sobre el raster. Los valores que no corresponden a la superposición, quedan como NA.
r_filtered <- mask(sat, cropped)
# Se transforman a dataframe, manteniendo las coordenadas y se eliminan los NA generados en el paso anterior
df1 = r_filtered %>% as.data.frame(xy = T) %>% as_tibble() %>% drop_na()

#Como se están procesando relevamientos no asfaltados, el valor que toma esta columna es 1
df1['fl_no_asfaltada'] = 1
```

Se repite el proceso para las vías no asfaltadas.

```
# Se busca la intersección entre el contorno y las vías de circulación del indecente.
cropped<-gIntersection(vias0,vias )
# Se lee el raster asociado
sat = stack('./raster_folder/raster_0.tif')
my_extent_poly <- as(extent(sat), "SpatialPolygons")
# Se obtienen los valores rasters que están superpuestos sobre el raster. Los valores que no corresponden a la superposición, quedan como NA.
r_filtered <- mask(sat, cropped)
# Se transforman a dataframe, manteniendo las coordenadas y se eliminan los NA generados en el paso anterior
df0 = r_filtered %>% as.data.frame(xy = T) %>% as_tibble() %>% drop_na()

#Cómo se están procesando relevamientos no asfaltados, el valor que toma esta columna es 1
df0['fl_no_asfaltada'] = 0
```

```
colnames(df1) = colnames(df0)
df = rbind(df1,df2)

#Se realiza una muestra de 100,000 registros para que el entrenamiento sea menos costoso en capacidad computacional

df = df %>% sample_n(100000)
```

En este punto, lo que obtenemos es una base de datos tabular con los valores que toma la imágen satelital en sus cinco bandas, es decir, nuestras variables independientes, y una columna que corresponde a la variable objetivo, es decir, nuestra variable dependiente.

Se toma una muestra de 100,000 registros de los 1,989,249 ya que cada fila corresponde a un punto en el espacio y puntos contiguos tendrán similares valores en las imágenes satelitales y la misma clasificación.

Esta estructura tabular permite realizar un modelo de *Machine Learning*.

Modelo Random Forest

1. Se generan los *dataset* de entrenamiento y *test*. De los 100.000 registros muestreados, se entrena con 70.000 y se dejan 30.000 para evaluación.

```
library(randomForest)
train_indices <- sample(1:nrow(df), 0.7 * nrow(df))
train_data <- df[train_indices, ]
test_data <- df[-train_indices, ]
train_data$t = train_data$t %>% as.factor()
```

2. Se entrena el modelo.

```
# Train a random forest model
rf_model <- randomForest(t ~ ., data = train_data%>% dplyr::select(-y,-x))
```

Predicciones

El problema de la estructura previamente desarrollada es que se pasó de líneas a puntos para generar los datos tabulares. Así, la salida del modelo también son puntos.

Para que las predicciones tengan sentido en términos de líneas, es necesario volver a realizar el pasaje.

```
predictions <- predict(rf_model, newdata = test_data%>% dplyr::select(-y,-x))
 test_data['pred'] = predictions

 #Se transforma el dato tabular a dato georreferenciado con las coordinadas del raster
 coordinates(test_data) <- ~x+y
 #Se establece la projección identica a las vias de circulacion
 crs(test_data) = crs(vias)

 # Se transforma el dataset a formato {sf}
 test_data_sf <- st_as_sf(
 test_data,
 coords = c("Longitude", "Latitude"), # coordenadas x,y en ese orden

crs = st_crs(vias) # debe ser equivalente entre ambos objetos
 )
 # Lo mismo con las vías de circulación predichas (en formato línea)
 vias_new_sf = st_as_sf(vias_new)
 vias_new_sf['idx'] = 1:nrow(test_data)


 # Este paso es el más crucial, se identifica la línea más cercana al punto
 nf = st_nearest_feature(test_data_sf, vias_new_sf)

 # Se genera un dataframe con el índice de la vía (línea) más cercana y la predicción asociada
 dfg = data.frame(idx=nf, predp=test_data$tpred)
 i = sort(unique(nf),decreasing = T)[1]
 # Se realiza el join de la predicción, a las vias (lineas)
 dfg1 = left_join(dfg,vias_new_sf)
```

El problema en este punto es que para cada línea tenemos las predicciones de todos los puntos que la componen. Para generar un dato por línea, es necesario agregar todos los puntos. Se calcula el valor máximo de los puntos y la desviación estándar de los puntos que componen cada línea, para evaluar qué tan consistente es el dato.

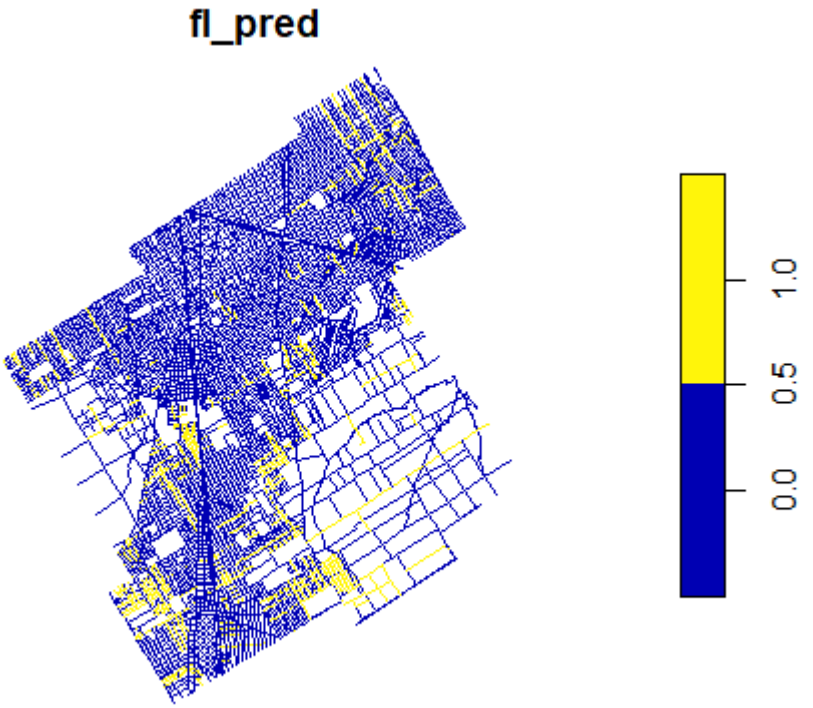
```
dfg2 = dfg1 %>% group_by(idx) %>% summarise(pred_max= max(predp),pred_sd = sd(predp))
```

Como resultado se obtiene un *dataframe* común con el índice de la vía y la predicción máxima y su desviación estándar. Nuevamente, se *joinea* el *dataframe* resultante con el *dataframe* georreferenciado anterior y así se obtiene finalmente un *dataframe* georreferenciado con líneas, donde cada línea tiene los atributos de la predicción.

```
dfg3 = left_join(vias_new_sf,dfg2)
```

```
library(tidyverse)
 library(rgdal)
 library(raster)
 library(rgeos)
 library(sp)
 library(sf)
 library(arrow)
 library(geojsonio)
 library(leaflet)

 fname = './prediccion_brown_v2.geojson'
 predicciones = read_sf(fname)#INDEC
 predicciones$pred_mean=as.numeric(predicciones$pred_max)
 predicciones$pred_sd=as.numeric(predicciones$pred_sd)
 predicciones$fl_pred=predicciones$pred_mean > 0.6
 plot(predicciones['fl_pred'])
```



Mapa 5: Vías de circulación del municipio de Alte Brown. En azul, las calles cuya predicción promedio asociada tienen una probabilidad menor o igual al 60% de no estar pavimentadas. Amarillo, las calles cuya prediccion promedio asociada tiene una probabilidad mayor al 60% de probabilidad de no estar pavimientadas

Asimismo, este resultado tiene la limitación, tal como se vio en la sección de Predicciones, de ser una predicción punto a punto y no para el vector que constituye la calle completa. Es decir, la unidad de análisis del estudio es distinta a las observaciones utilizadas para

entrenar y hacer las predicciones. Por lo tanto, se pierde cierta información al usar el promedio como agregación de las predicciones asignadas al vector.

Resultados

La salida del modelo consiste en puntos y, al evaluar el desempeño sobre el conjunto de datos de evaluación, se obtuvo el siguiente resultado:

- Calles asfaltadas clasificadas correctamente: 20.393
- Calles sin asfaltar clasificadas incorrectamente como asfaltadas: 301
- Calles asfaltadas clasificadas incorrectamente como sin asfaltar: 448
- Calles sin asfaltar clasificadas correctamente: 8.858

	Asfalto	No-Asfalto
Asfalto	20393	301
No-Asfalto	448	8858

Los puntos resultantes, agregados de la manera mencionada previamente y cruzados con los polígonos relevados originalmente y las vías de circulación, pueden ser asignados a calles específicas. De esta forma se puede estimar el error general del proyecto.

```
preds_asfalto = intersect(predicciones,asfalto )

preds_tierra = intersect(predicciones,tierra)

dfres = data.frame(fl_pred=c((a$pred_mean),(b$pred_mean)), fl_real = c(rep(0,length(a)),rep(1,length(b))))
```

La tabla, *dfres*, resulta en:

id_via	fl_pred	fl_real
1	0	0
2	1	1
3	0	0
4	0	0
5	1	0

…	…	…

El relevamiento permitió identificar 609 calles asfaltadas y 135 no asfaltadas. Con esta información, se evaluó la clasificación mediante las siguientes métricas:

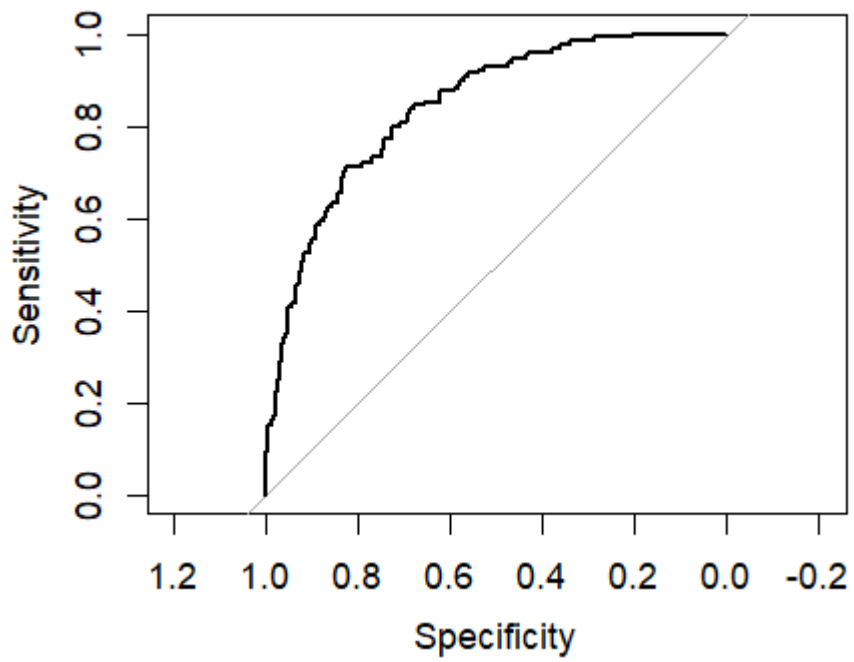
- Recall: 92,83%
- Precisión: 89,12%
- F1: 90.93%

La matriz de confusión, considerando como "sin asfaltar" aquellas calles con probabilidad de clase "sin asfaltar" mayor o igual a 0.6, y como "asfaltadas" las de probabilidad menor o igual a 0.6, es:

- Asfaltadas correctamente clasificadas: 557
- Sin asfaltar clasificadas incorrectamente como asfaltadas: 43
- Asfaltadas clasificadas incorrectamente como sin asfaltar: 68
- Sin asfaltar correctamente clasificadas: 60

	Asfalto	No-Asfalto
Asfalto	557	43
No-Asfalto	68	60

Dado que las métricas anteriores dependen del punto de corte, también se evaluó la clasificación mediante la curva ROC, obteniéndose un área bajo la curva de: AUC-ROC: 0.8437.



Imágen 2: Curva ROC

El área bajo la curva ROC (AUC-ROC) resume el rendimiento de la clasificación independientemente del punto de corte utilizado. Un AUC de 0.5 significa clasificación aleatoria y 1 representa una clasificación perfecta.

En este caso, se obtuvo un AUC-ROC de 0.8437. Esto indica que independientemente del punto de corte elegido, la clasificación tuvo un rendimiento razonablemente bueno, bastante mejor que aleatorio.

Conclusiones

Las imágenes utilizadas en este estudio tienen una resolución limitada de 10m2, lo cual introduce incertidumbre en las predicciones, especialmente en calles con abundante arborización, veredas con césped o calles angostas donde la señal del raster es más débil. Para reducir este ruido, se podrían incorporar imágenes satelitales de mayor resolución.

Es necesario evaluar el desempeño del modelo en otras ciudades y climas para confirmar la generalización de los resultados.

Más allá del valor práctico de ahorrar trabajo en el relevamiento de calles pavimentadas, este trabajo aporta una metodología reproducible ya que las máscaras de pavimentación pueden reemplazarse por otros temas de interés. El código desarrollado es adaptable para expandir estas predicciones a diversos casos de uso.