Minority Report en Londres: un análisis de los sesgos en los datos policiales

Azucena González Muiño

PyConES 2021

https://github.com/azucenagm/TFM-MUCD

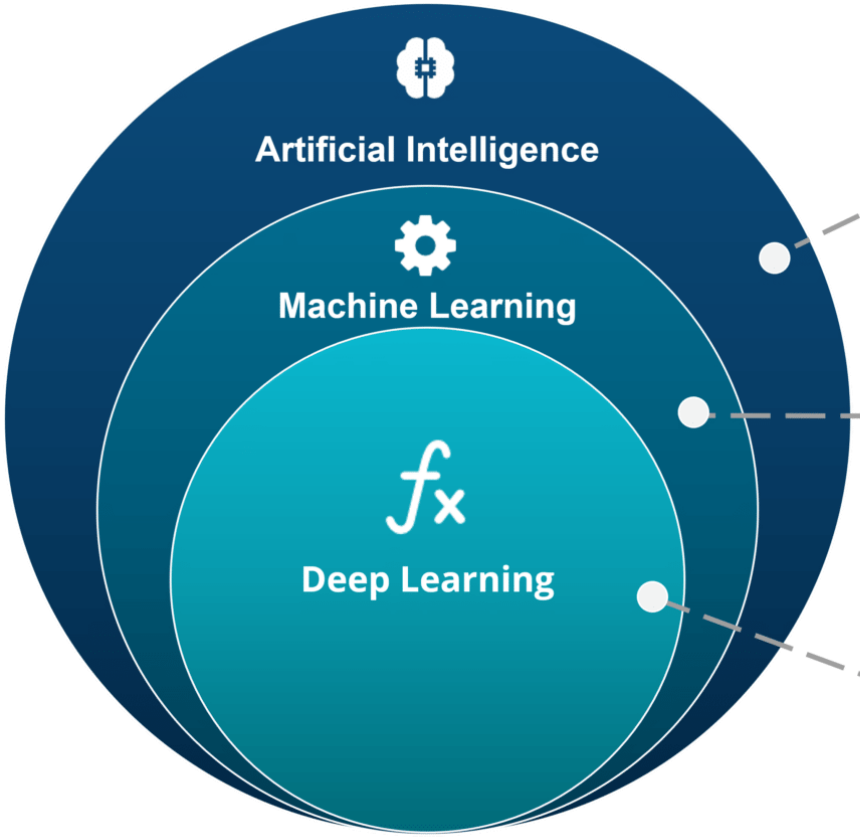
Foto: Jeremy Bishop en Unsplash

1. Introducción
2. Construcción del modelo
3. Análisis de sesgos
4. Interpretabilidad
5. Equidad algorítmica
6. Conclusiones

Foto: Jeremy Bishop en Unsplash

Introducción

Definición de machine learning



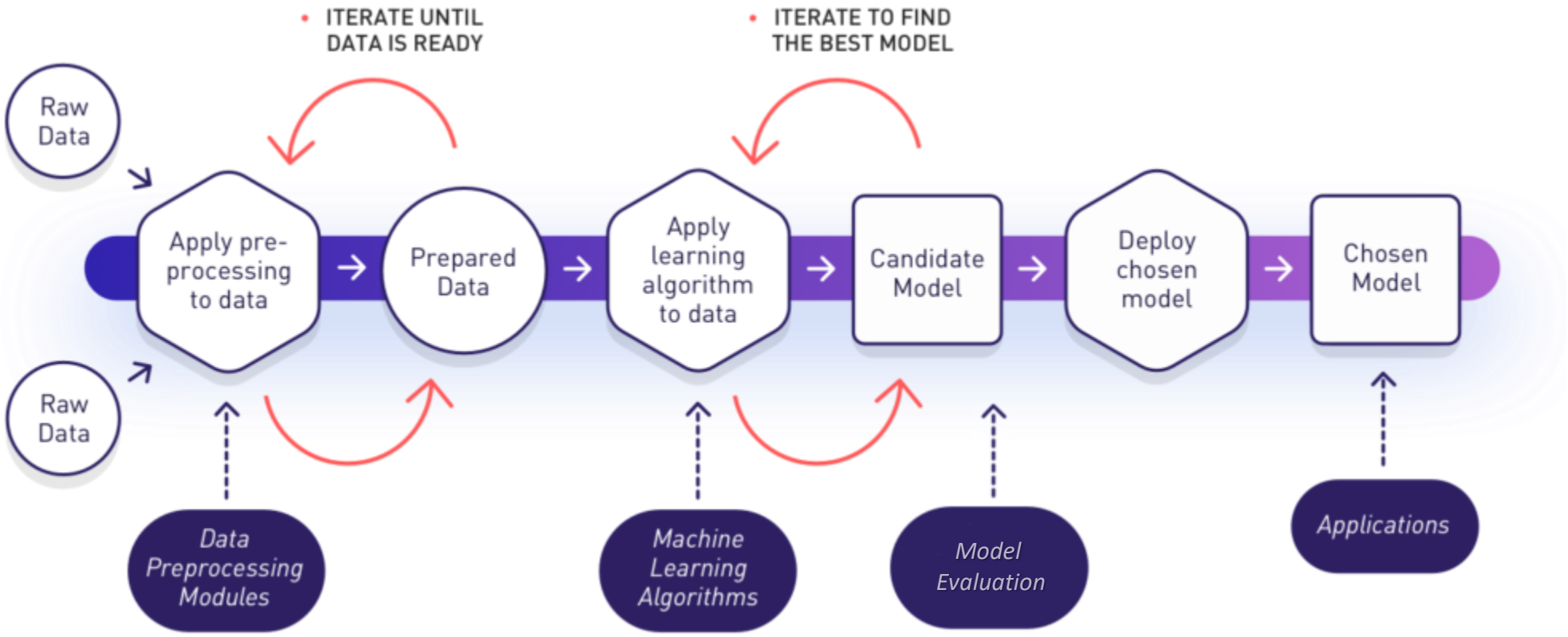
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Osprey Data. Getting To Know AI and ML Models

Ciclo de vida de un proyecto de ML



Basado en: PNGKey. Machine Learning Diagram

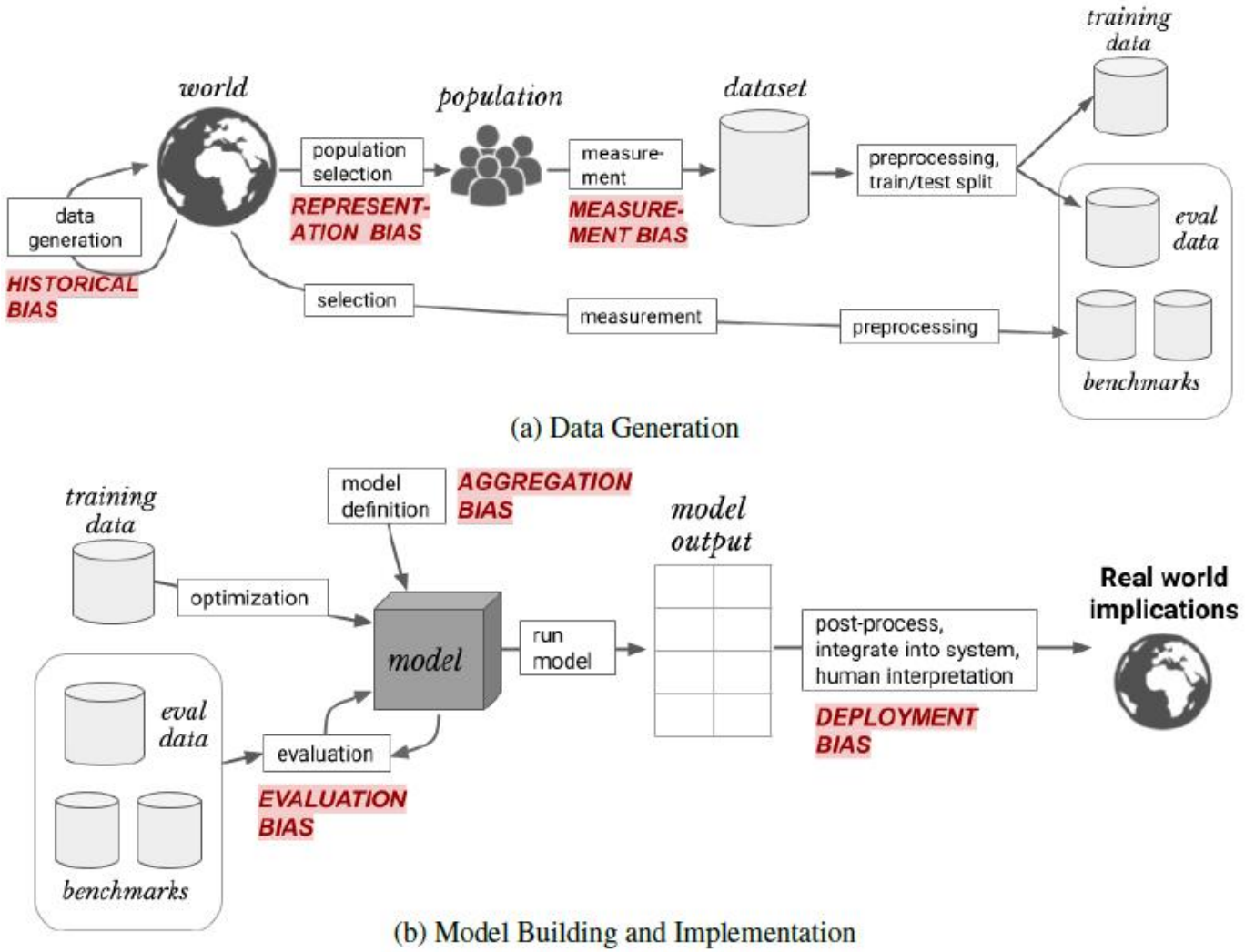
En resumen…

Podemos obtener modelos:

- No programados explícitamente para un problema concreto
- Que emplean en datos reales para detectar patrones
- Y obtienen conclusiones que ayudan a resolver el problema

¿Qué puede salir mal?

Posibles sesgos en un proyecto de ML



SURESH, Harini y GUTTAG, John V. A Framework for Understanding Unintended Consequences of Machine Learning (2020)

Sistemas de vigilancia predictiva: definición

Aunque no existe una definición universal, hay consenso en sus principales características:

- Se fundamentan en el análisis de datos
- Pueden aplicarse sobre individuos o sobre localizaciones
- Su objetivo principal: mejorar las actuaciones preventivas policiales

Sistemas de vigilancia predictiva: críticas

Principales críticas:

- Poca transparencia en su diseño
- Poco conocimiento de su comportamiento interno
- Opacidad sobre el uso de estas herramientas: quién y cómo
- Falta de estudios sobre su efectividad e impacto

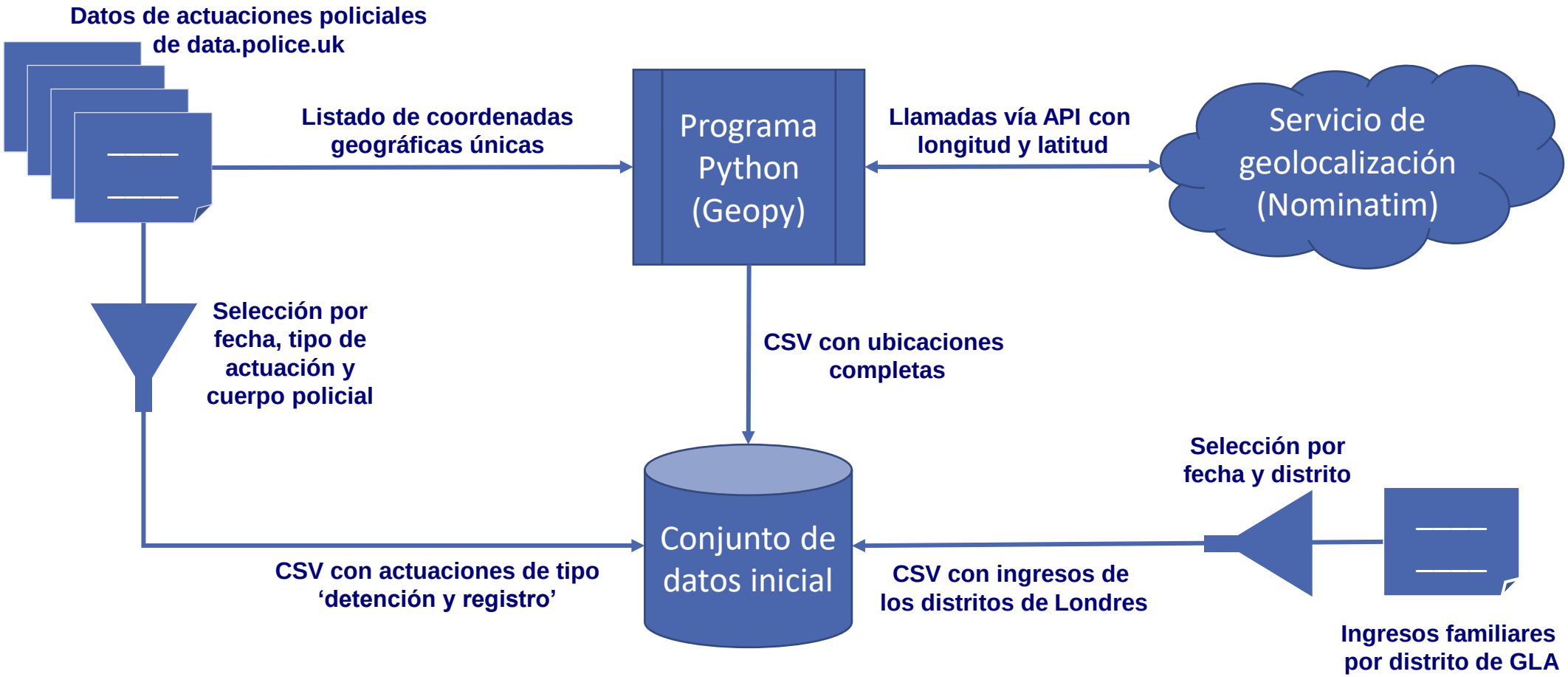
2 Construcción del modelo

Descripción del modelo

Principales características:

- Se basa en la operativa policial de “detención y registro”
- Objetivo: predecir qué operaciones de este tipo serán exitosas (se encuentran indicios de delito)
- Población: zona de Gran Londres (33 distritos, +8 millones de habitantes)
- Variable objetivo: registro con resultado positivo (se requiere acción policial o judicial posterior) o negativo (no se encuentran indicios de delito)

Fuentes de datos



Obtención del distrito


```python
# Gestión de coordenadas y localizaciones
import geopy.geocoders
from geopy.extra.rate_limiter import RateLimiter
from geopy.geocoders import Nominatim
```

```python
# Función que devuelve la dirección completa encontrada a partir de las
# coordenadas dadas o None si no posible recuperarla
def get_location(coords):
    try:
        # Se formatea la localización y se solicita su dirección
        coords = ['{:f}'.format(coords[0]), '{:f}'.format(coords[1])]
        address = str(geolocator.reverse(coords))
        return address
    except Exception as exception:
        print(f'Error al tratar las coordenadas {coords}: {exception}')
        return None

# Lista para almacenar las direcciones encontradas
addresses = []

# Se genera un agente para obtener la dirección física a partir de las coordenadas
geolocator = Nominatim(user_agent="data-police-uk-tfm")

# Se limita la velocidad a la que se solicitarán los datos para evitar saturar
# el servidor y cumplir con las políticas de uso de Nominatim
geocode = RateLimiter(geolocator.geocode, min_delay_seconds=1)

# Se aumenta el tiempo de espera de respuesta
geopy.geocoders.options.default_timeout = 60

# Se obtienen las direcciones completas
for l in full_data_locs.values.tolist():
    address = get_location(l)
    addresses.append(address)
```

Nominatim

Análisis exploratorio de datos

Fundamental para conocer los datos. A partir de nuestro análisis:

- Se descartan algunos campos
- Se localizan sesgos de distinta naturaleza
- Se detectan variables parecidas
- Se determina que la clase a predecir está desbalanceada

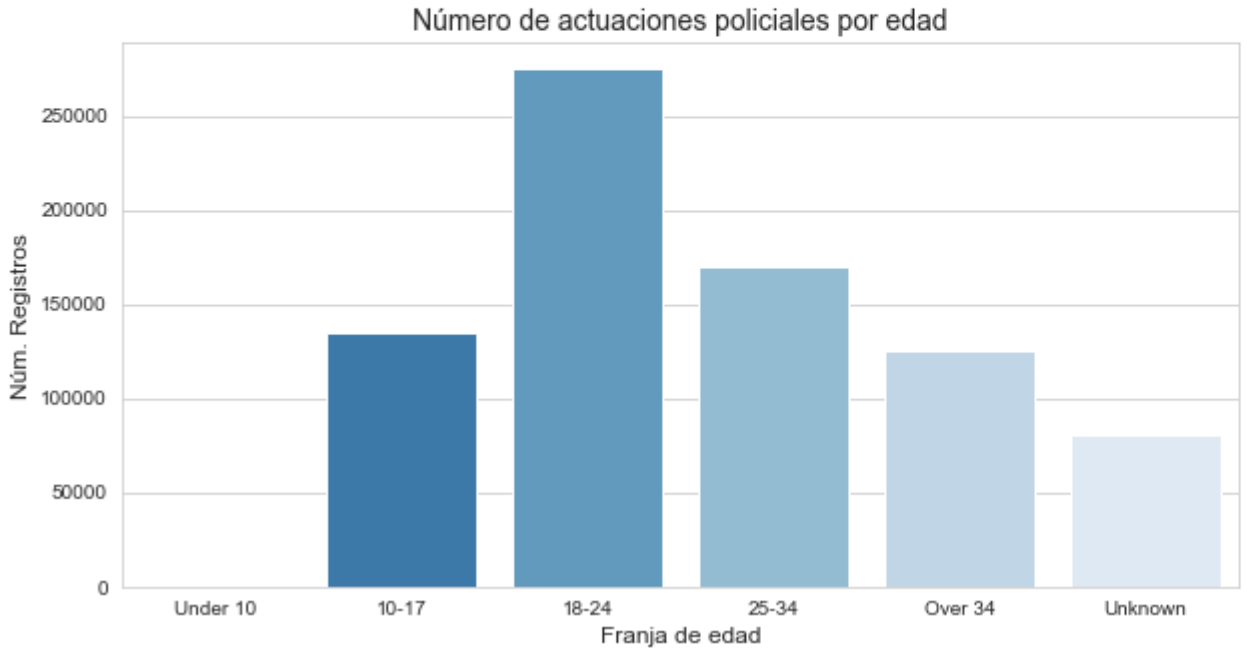
Principales librerías usadas:

- Pandas y numpy para la manipulación de datos
- Seaborn y geopandas para representaciones visuales

Análisis exploratorio de datos


```
# Creación de gráficas
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Gráfica con el número de actuaciones policiales por franja de edad
plt.figure(figsize=(10, 5))
sns.set_style('whitegrid')
sns.countplot(x='Age-range', data=london_data, palette='Blues_r')
plt.title('Número de actuaciones policiales por edad', fontsize=14)
plt.xlabel('Franja de edad', fontsize=12)
plt.ylabel('Núm. Registros', fontsize=12)
plt.show()
```



Uso de geopandas


```
# Generación de mapas
import geopandas as gpd
```

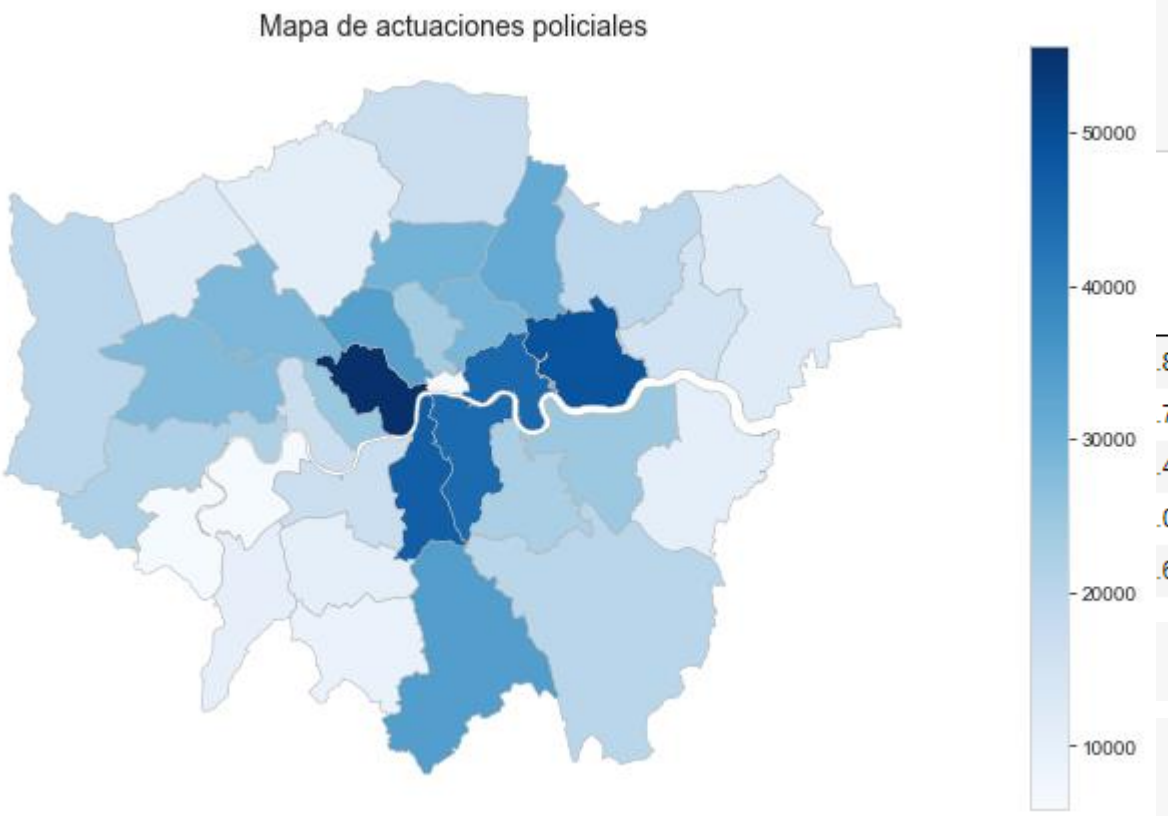
```
# Se carga el mapa de Londres
london_map = gpd.read_file('.
print('Variables:', london_ma
london_map.head()
```

```
Variables: ['NAME' 'GSS_CODE'
 'SUB_2006' 'geometry']
```

	NAME	GSS_CODI	geometry
0	Kingston upon Thames	E0900002	.800, 516407.300 16...
1	Croydon	E0900000	.700, 535005.500 15...
2	Bromley	E0900000	.400, 540361.200 15...
3	Hounslow	E0900001	.000, 521967.700 17...
4	Ealing	E0900000	.600, 510249.900 18...

```
# Se hace un join por distri
map_data = london_map.set_in
```

```
# Se genera un mapa coropléti
fig, ax = plt.subplots(figsiz
ax.axis('off')
ax.set_title('Mapa de actuaciones policiales', fontsize='14')
map_data.plot(column='Location', cmap='Blues', ax=ax, edgecolor='0.7', linewidth=0.5, legend=True)
```

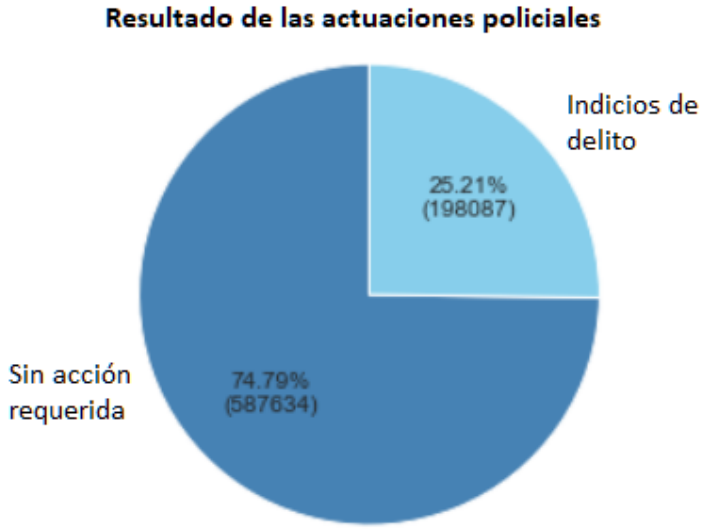


Evaluación: selección de métrica

Métrica seleccionada: *F1 score*

$$F1\ score = 2 \times \frac{precision \times recall}{precision + recall}$$

- Problema de clasificación
- Clases desbalanceadas
- Importancia de la clase positiva
- Equilibrio en el tipo de error



Evaluación: selección de métrica


```
from sklearn import metrics
```

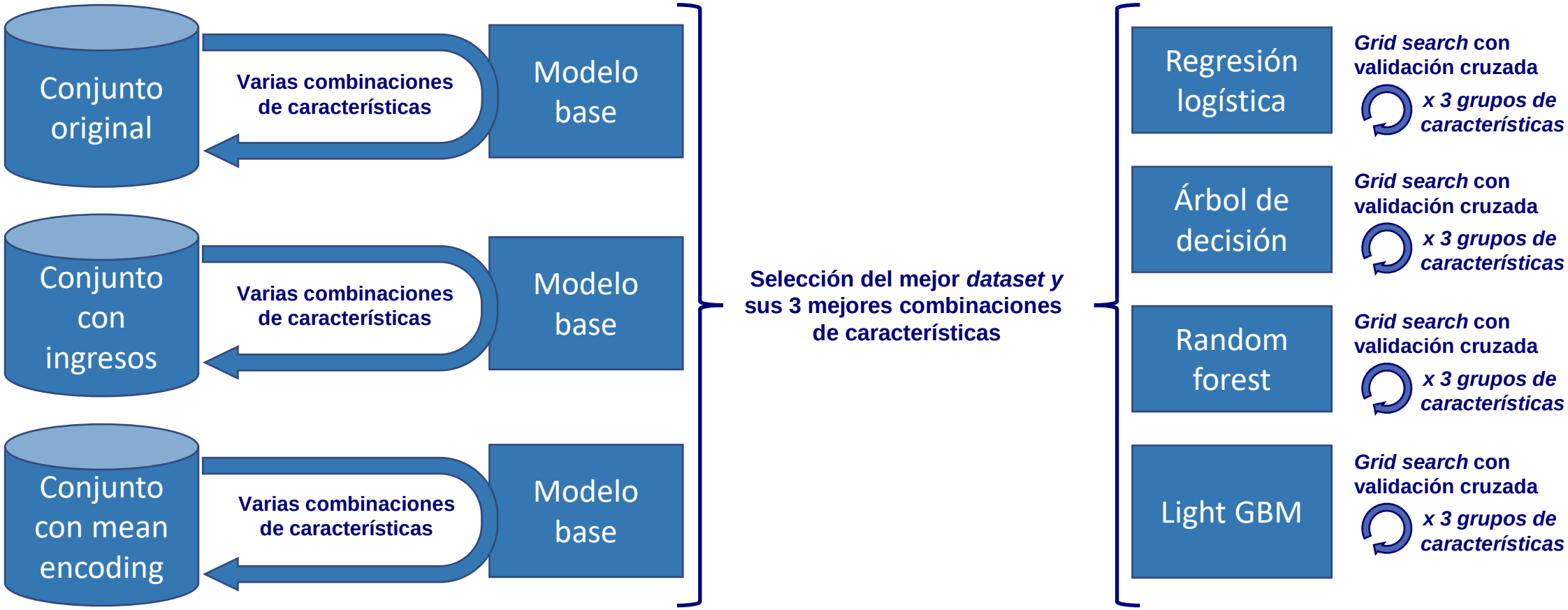
```
from sklearn.tree import DecisionTreeClassifier
```

```
# Se entrena el modelo y se evalúa con el conjunto de validación
tree_model = DecisionTreeClassifier(random_state=seed,
                                    class_weight='balanced',
                                    splitter=tree_best_params.get('splitter'),
                                    criterion=tree_best_params.get('criterion'),
                                    max_depth=tree_best_params.get('max_depth'),
                                    min_samples_leaf=tree_best_params.get('min_samples_leaf'),
                                    min_impurity_decrease=tree_best_params.get('min_impurity_decrease'),
                                    max_features=tree_best_params.get('max_features'))

tree_model.fit(tree_train, y_train)
# Predicción para el conjunto de validación
tree_y_pred = tree_model.predict(tree_val)
```

```
print(metrics.classification_report(y_val, tree_y_pred, output_dict=False, zero_division=0))
```

	precision	recall	f1-score	support
False	0.80	0.82	0.81	127303
True	0.32	0.30	0.31	37699
accuracy			0.70	165002
macro avg	0.56	0.56	0.56	165002
weighted avg	0.69	0.70	0.69	165002

Experimentos



Experimentos


```
# Se prueban distintas combinaciones de variables en el modelo base
best_f1 = .0
features = []
for i in range(len(last_cols)+2):
    # Se seleccionan las variables a emplear
    if i == 0:
        features = first_cols.copy()
    elif i == len(last_cols)+1:
        features = first_cols + last_cols
    else:
        features.append(last_cols[i-1])

    # Se recoge la selección de variables seleccionada de los conjuntos de datos
    bl_train = pd.DataFrame(X_train, columns=get_features(X_train.columns, features))
    bl_val = pd.DataFrame(X_val, columns=get_features(X_val.columns, features))

    # Modelo de referencia
    bl_model = DecisionTreeClassifier(random_state=seed, class_weight='balanced')
    bl_model.fit(bl_train, y_train)

    # Se aplica el modelo sobre el conjunto de validación y se recogen su métricas
    y_pred = bl_model.predict(bl_val)
    bl_class_metrics = metrics.classification_report(y_val, y_pred, output_dict=True)

    # Si el resultado no supera la métrica anterior, se descarta esa variable
    if (i < len(last_cols)+1):
        if bl_class_metrics.get('macro avg').get('f1-score') < best_f1:
            features.remove(last_cols[i-1])
        else:
            best_f1 = bl_class_metrics.get('macro avg').get('f1-score')
```

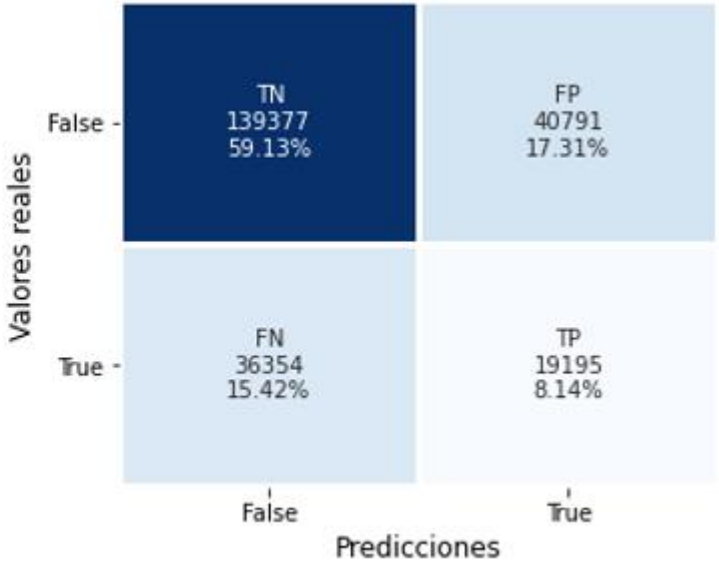
Experimentos


```
# Hiperparámetros a probar
params = {'splitter': ['best', 'random'],
          'criterion': ['gini', 'entropy'],
          'max_depth': [5, 15, 20, 30, None],
          'min_samples_leaf': [1, 5, 9],
          'min_impurity_decrease': [0, 0.01, 0.1, 0.25],
          'max_features': ['sqrt', 'log2']}

# Se busca la mejor combinación de atributos e hiperparámetros con validación cruzada
for features in best_features:
    # Selección de variables
    tree_train = X_train[get_features(X_train.columns, features)]
    tree_val = X_val[get_features(X_val.columns, features)]

    # Definición del modelo
    tree_model = DecisionTreeClassifier(random_state=seed, class_weight='balanced')

    # Grid con cross validation
    tree_grid = GridSearchCV(tree_model, params, scoring='f1_macro', cv=tscv)
    tree_grid.fit(tree_train, y_train)

    # Se muestra la combinación de hiperparámetros con mejor rendimiento
    print('\nVariables:', features)
    print('Hiperparámetros con mejores resultados:', tree_grid.best_params_)
    print('Puntuación media del mejor estimador: %.6f' % tree_grid.best_score_)

    # En cada iteración se comprueba si el resultado obtenido es mejor
    if tree_grid.best_score_ > tree_best_score:
        tree_best_score = tree_grid.best_score_
        tree_best_features = features
        tree_best_params = tree_grid.best_params_
```

Modelo final

- Random forest
  - Conjunto de datos: versión 2. Sustitución de la variable distrito por los ingresos familiares estimados. *Ordinal encoding* y *one hot encoding*.
  - Características: tipo de registro, etnia informada por el agente, ingresos del distrito y año.
  - *F1 score (conjunto de prueba)*: 56%

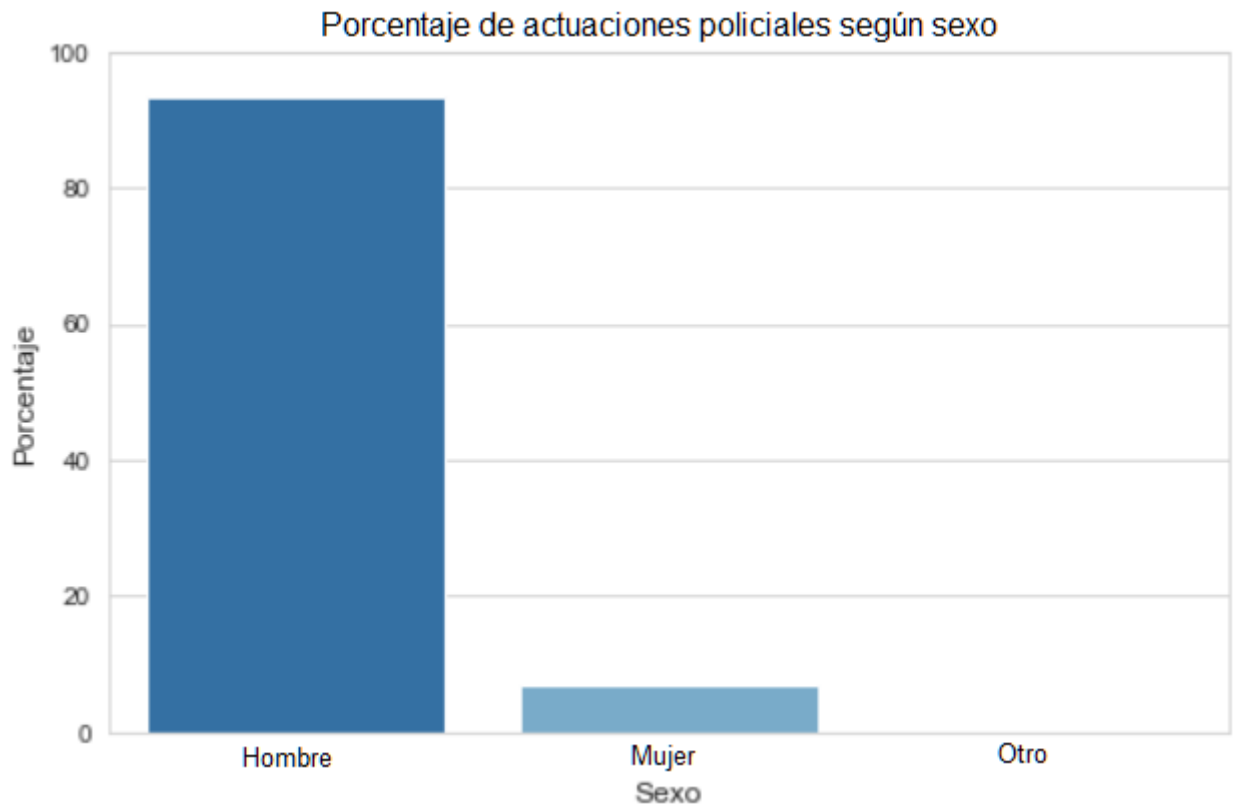
	precision	recall	f1-score	support
False	0.79	0.77	0.78	180168
True	0.32	0.35	0.33	55549
accuracy			0.67	235717
macro avg	0.56	0.56	0.56	235717
weighted avg	0.68	0.67	0.68	235717



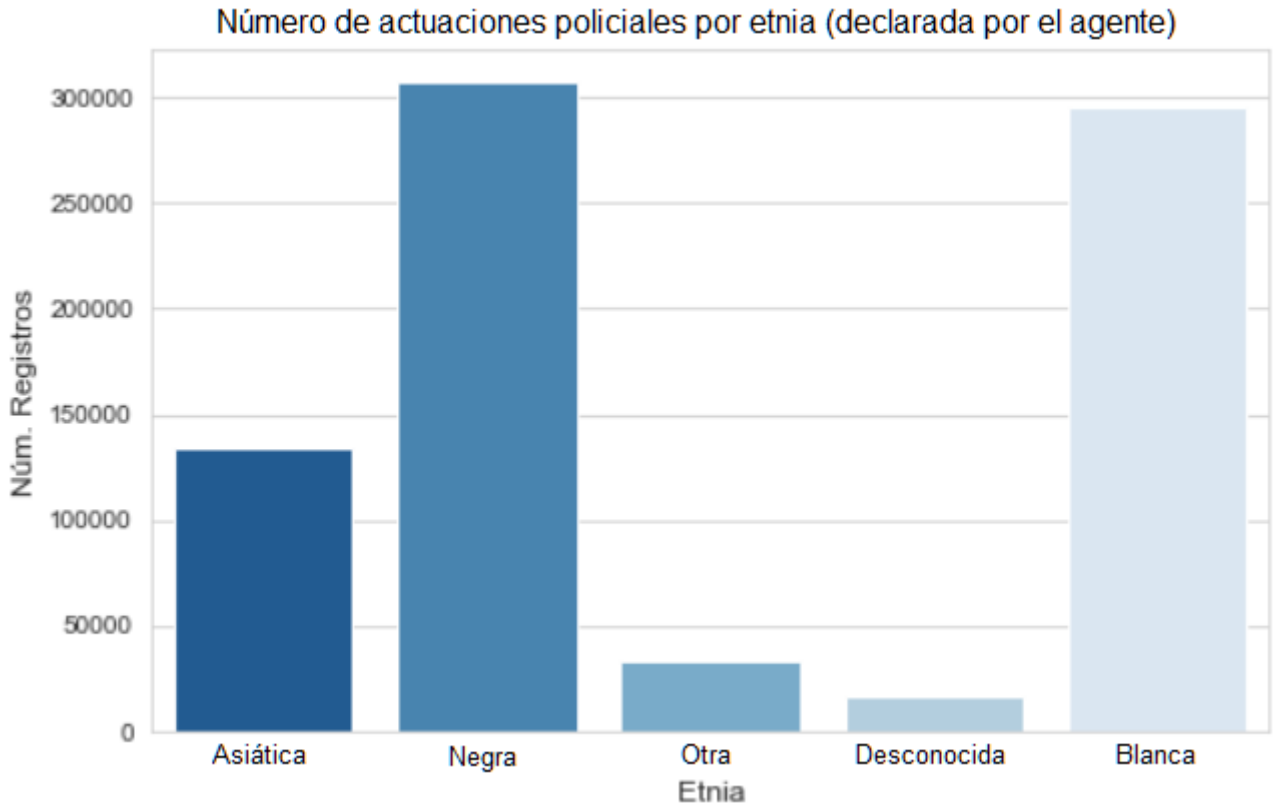
TN: verdaderos negativos
FP: falsos positivos
FN: falsos negativos
TP: verdaderos positivos

3 Análisis de sesgos

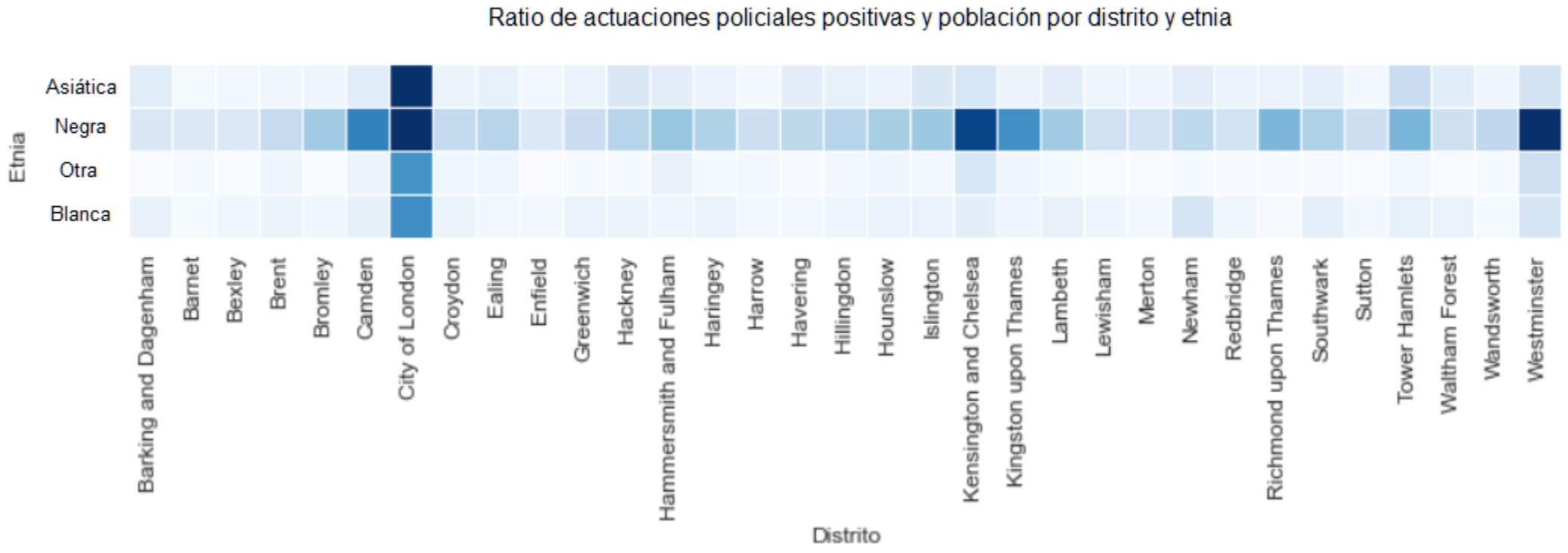
Sesgo de representación: sexo



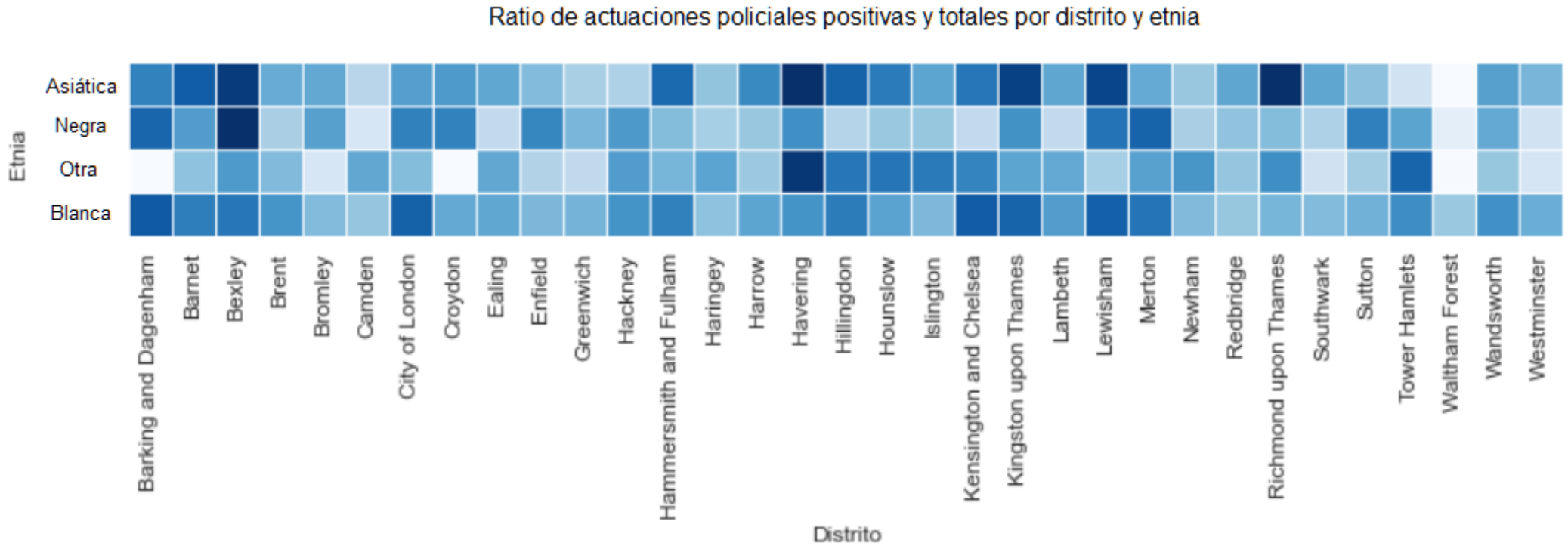
Sesgo de representación: etnia



Sesgo de representación: etnia



Sesgo de representación: etnia

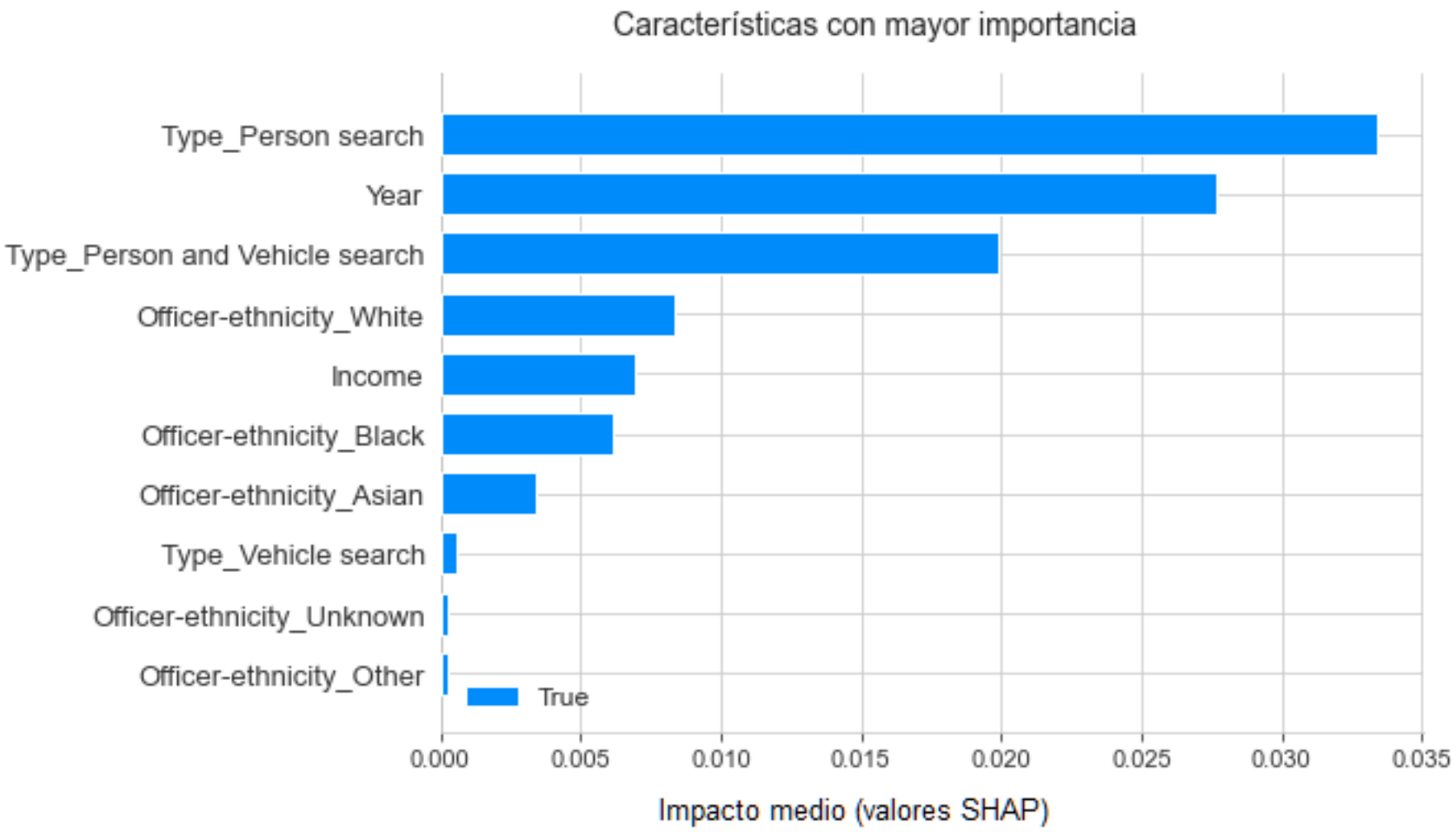


Interpretabilidad

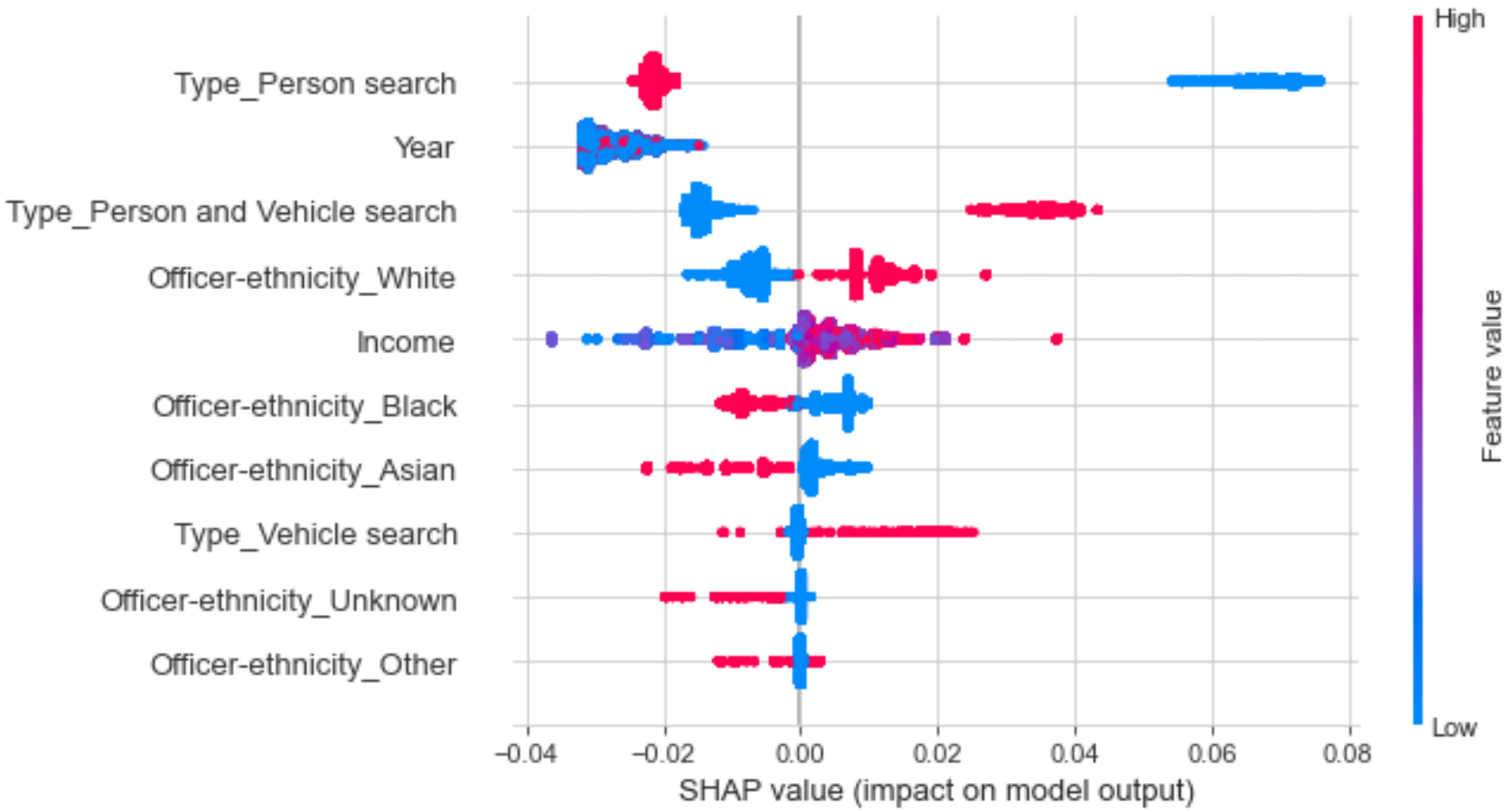
Definición y motivación

- La interpretabilidad puede entenderse como el grado en que una persona puede comprender la causa de una decisión tomada por el algoritmo.
- Aunque existen algoritmos transparentes o muy fácilmente interpretables, denominados de caja blanca, los más potentes y utilizados suelen ser difíciles de comprender, actuando como una caja negra por la que pasan las entradas del sistema y se obtienen unos resultados.
- Últimamente ha habido un aumento del número de técnicas externas de interpretación de modelos, siendo dos de las más usadas LIME y SHAP.

Importancia de las características (SHAP)



Visión global con SHAP



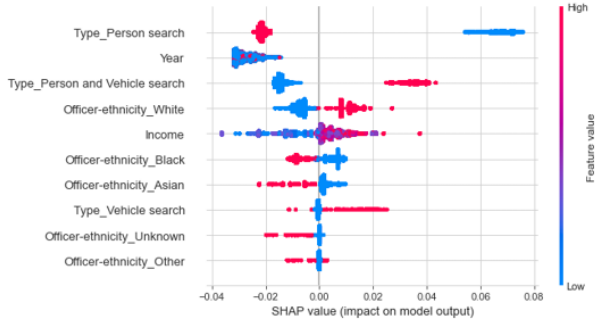
Ejemplo de uso de SHAP


```
# SHAP
import shap
```

```
# Se generan los objetos SHAP necesarios para el uso de esta librería y se activa javascript
shap.initjs()
shap_explainer = shap.TreeExplainer(model)
shap_values = shap_explainer.shap_values(X_test)
```

```
# Impacto de las principales características en el modelo
shap.summary_plot(shap_values, X_test, max_display=10, class_inds=[1], class_names=['False', 'True'])
```


```
# Se muestra el impacto de las características para cada observación del conjunto de datos seleccionado
# en relación a la clase positiva
shap.summary_plot(shap_values[1], X_test)
```



Equidad algorítmica

Definición y motivación

La equidad algorítmica implica que un modelo se comporta como está previsto en relación a los diferentes grupos representados en los datos, según la noción de igualdad seleccionada.

Para comprobarlo, suelen establecerse métricas que deben validarse contra el modelo. En nuestro caso, se han seleccionado las siguientes:

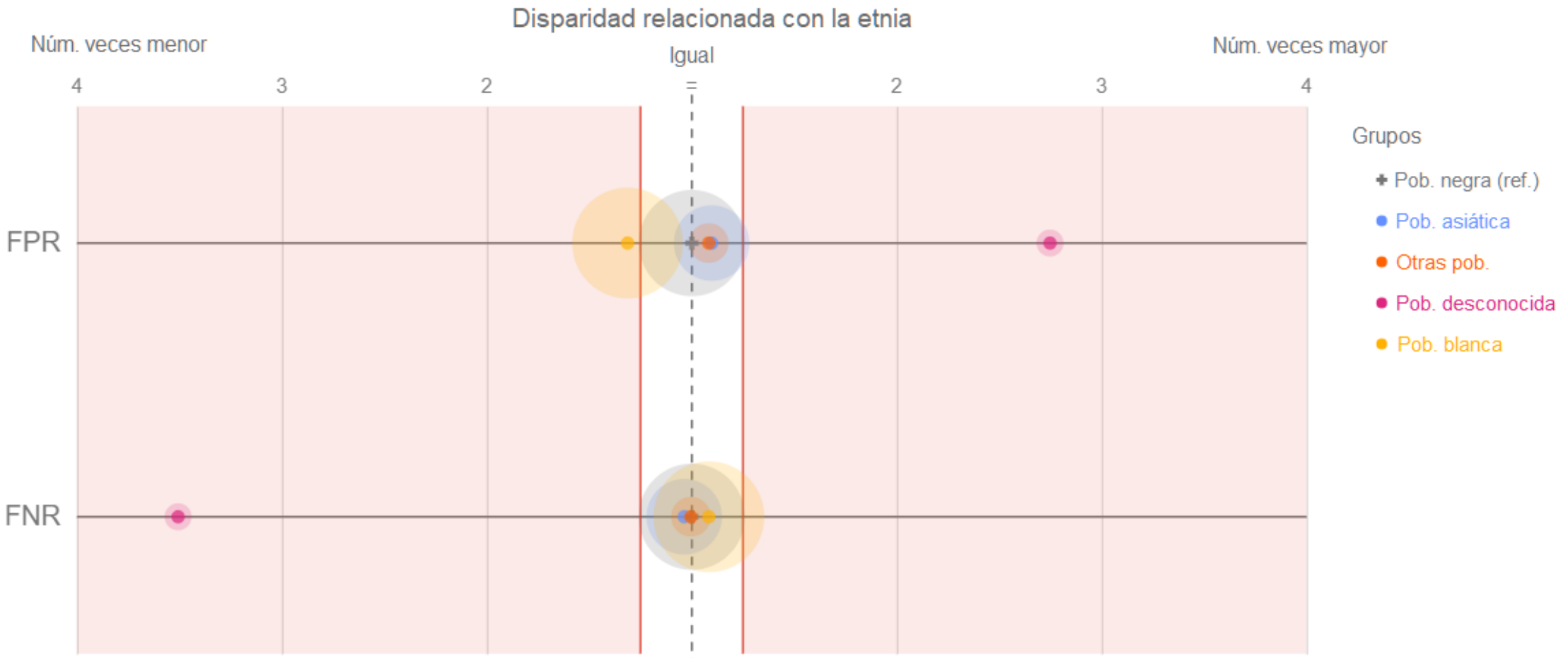
- FPR o tasa de falsos positivos: mide la probabilidad de que la policía detenga y registre a un inocente.

$$FPR = \frac{FP}{FP + TN}$$

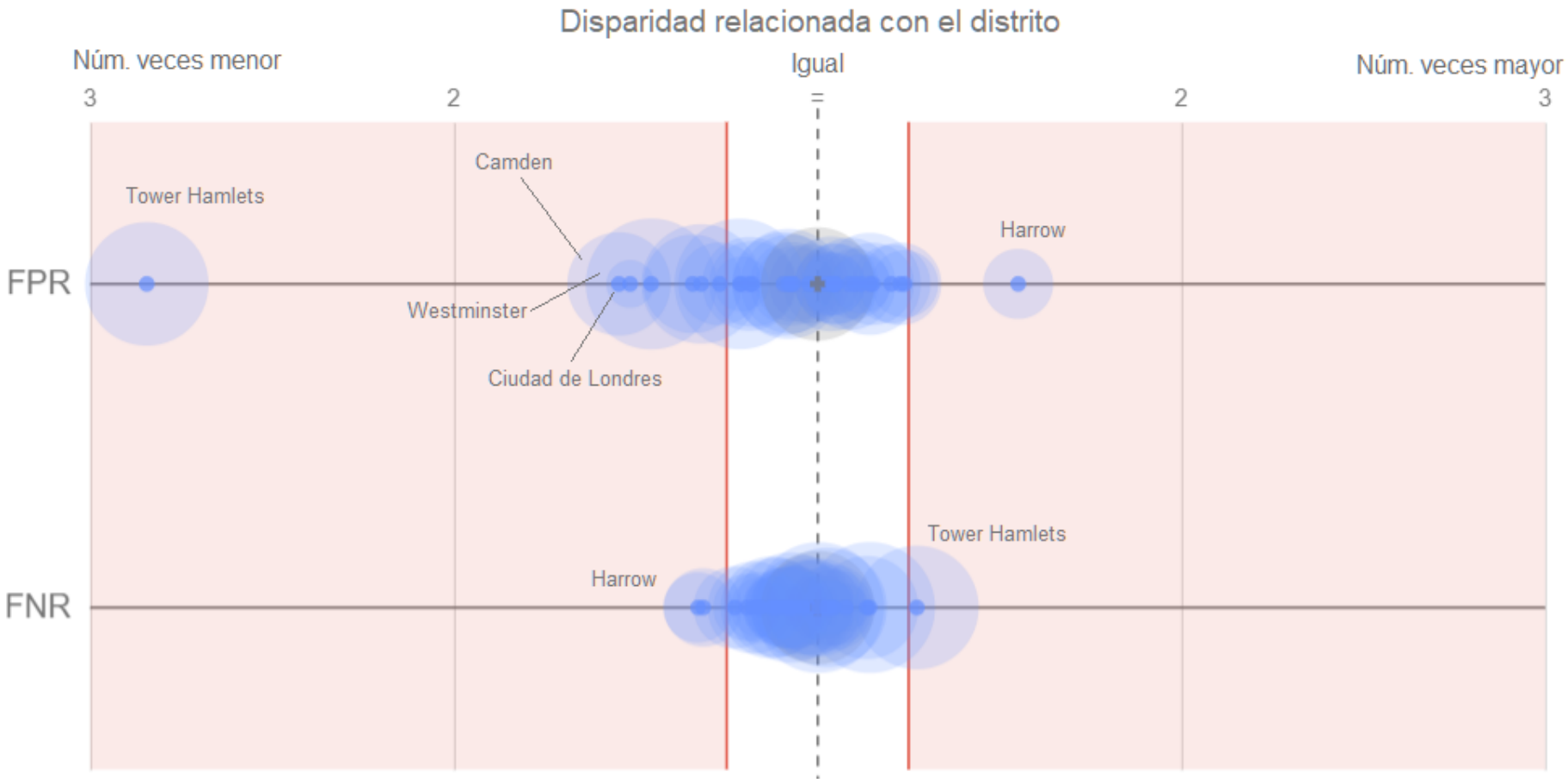
- FNR o tasa de falsos negativos: mide la probabilidad de que la policía no detenga ni registre a un infractor.

$$FNR = \frac{FN}{FN + TP}$$

Estudio por etnia



Estudio por distrito



Ejemplo de uso de Aequitas: estudio por etnia


```
# La librería Aequitas necesita que los valores reales estén en una columna llamada 'label_value'
# y las predicciones en 'score'
test_data.rename(columns={'Further-action':'label_value'}, inplace=True)
test_data['score'] = y_pred
test_data.head()

# Se inicializan algunos elementos necesarios para la ejecución de la librería
g = Group()
b = Bias()
aqp = Plot()

# Se indican las variables a estudio y las métricas seleccionadas
attributes_to_audit = ['Officer-ethnicity', 'Borough']
metrics = ['fpr', 'fnr']

# Se obtiene un dataframe con las estadísticas por variable y grupo y las métricas de fairness calculadas
xtab, _ = g.get_crosstabs(test_data, attr_cols=attributes_to_audit)

# Para las gráficas de disparidad se toman como variables de referencia la etnia negra y el distrito de Lambeth (ya
# que tiene un nivel de ingresos familiares intermedio)
bdf = b.get_disparity_predefined_groups(xtab, test_data,
                                        ref_groups_dict={'Officer-ethnicity':'Black', 'Borough':'Lambeth'},
                                        fill_divbyzero=0)

# Estadísticas y métricas para la etnia
xtab_eth = xtab[(xtab['attribute_name']=='Officer-ethnicity')]
xtab_eth
```

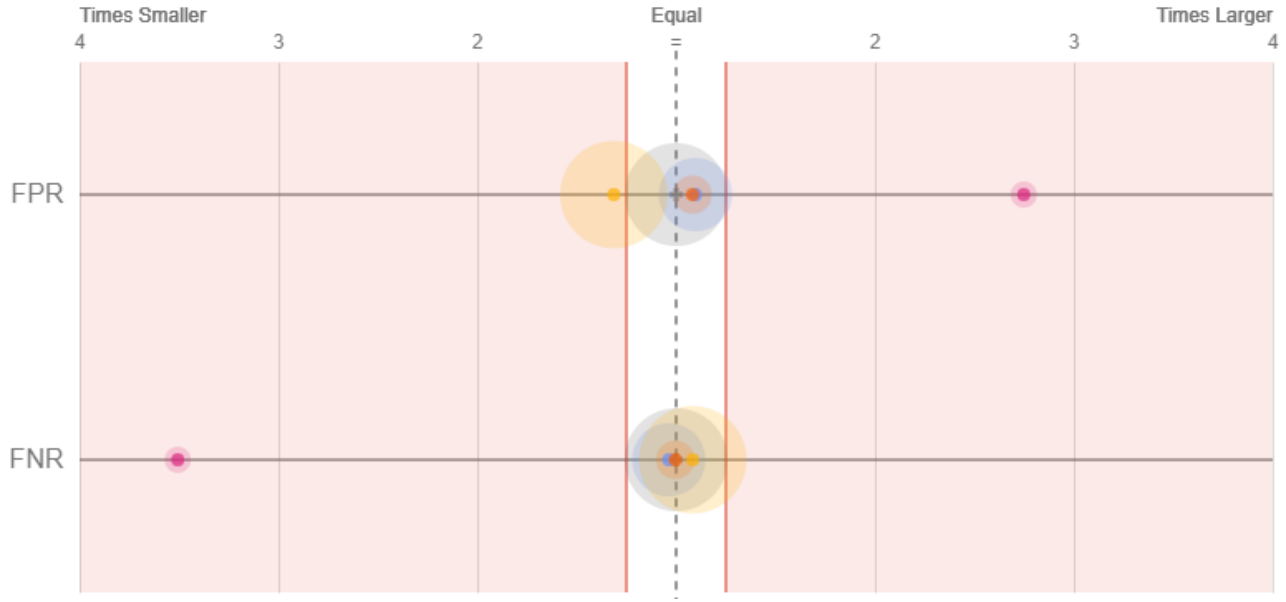
Aequitas
Bias & Fairness Audit

Ejemplo de uso de Aequitas: estudio por etnia

	model_id	score_threshold	k	attribute_name	attribute_value	tpr	tnr	for	fdr	fpr	fnr	npv	precision	pp	
0	0	binary 0/1	59986	Officer-ethnicity	Asian	0.373237	0.743206	0.203727	0.693985	0.256794	0.626763	0.796273	0.306015	12104	3(
1	0	binary 0/1	59986	Officer-ethnicity	Black	0.350147	0.765906	0.202639	0.690604	0.234094	0.649853	0.797361	0.309396	21988	62
2	0	binary 0/1	59986	Officer-ethnicity	Other	0.352804	0.746512	0.197787	0.716432	0.253488	0.647196	0.802213	0.283568	3195	8
3	0	binary 0/1	59986	Officer-ethnicity	Unknown	0.814669	0.356683	0.169664	0.667557	0.643317	0.185331	0.830336	0.332443	3742	1
4	0	binary 0/1	59986	Officer-ethnicity	White	0.295824	0.821763	0.213716	0.655114	0.178237	0.704176	0.786284	0.344886	18957	72

Aequitas
Bias & Fairness Audit

```
# Gráficas para FPR y FNR
ap.disparity(bdf, metrics, 'Officer-ethnicity')
```



Conclusiones

Conclusiones

Respecto al modelo obtenido:

- Tiene un rendimiento muy bajo
- Presenta importantes sesgos relativos a la etnia, en particular a favor de la blanca y en detrimento de la negra, tanto en los datos de entrada como en los resultados del modelo

Para evitar efectos no deseados en nuestros modelos es fundamental:

- Entender su comportamiento y en qué basan sus decisiones
- Identificar los sesgos que se producen no sólo durante su construcción y evaluación sino también en la recolección y selección de datos
- Establecer a priori las características sensibles y los criterios de equidad que deben cumplir

Conclusiones

Este tipo de sistemas:

- Plantean dudas sobre su eficacia e impacto en la sociedad
- Se basan en gran parte en datos policiales que, dado su carácter histórico, es probable que presenten sesgos
- Son cada vez más requeridos por las fuerzas policiales
- Se desconoce si son operados por personas conocedoras de la tecnología que emplean y sus limitaciones
- No facilitan información sobre su diseño ni cómo miden sus resultados

Minority Report en Londres: un análisis de los sesgos en los datos policiales
Azucena González Muiño
Presentación para la PyConES 2021

Extracto del TFM del Máster Universitario en Ciencia de Datos
Estudios de Informática, Multimedia y Telecomunicación
Universitat Oberta de Catalunya

Código fuente disponible en: https://github.com/azucenagm/TFM-MUCD

Foto: Jeremy Bishop en Unsplash