FACULTAD DE INGENIERIA
Universidad de Buenos Aires

75.06 - Organización de Datos

Primer Cuatrimestre 2020

Trabajo Práctico 2

Machine Learning

Alumno	**Padrón**	**Correo electrónico**
Agustín Rodríguez	101570	agrodriguez@fi.uba.ar
Facundo Fernández	89843	ffelfis@gmail.com

Nombre del grupo: Data Sicarios

Fecha de entrega: 29 de Julio de 2021

Repositorio: https://github.com/ffelfis/OrgaDatosTPs

Vínculo a presentación: https://youtu.be/nM7aY_aOvVA

Introducción

Este trabajo es la continuación del análisis exploratorio realizado en el Trabajo Práctico 1. A partir del mismo conjunto de datos, se busca desarrollar un modelo capaz de predecir el nivel de daño en edificios luego del terremoto ocurrido en 2015 en Gorkha, Nepal.

Se participa en una plataforma que organiza competencias para problemas de Data Science, Driven Data. En la página de la competencia se realizan las cargas de predicciones obtenidas por los modelos construidos.

Se cuenta con:

- Set de entrenamiento: train_values.csv
- Set de resultados esperados para el set de entrenamiento: train_labels.csv
- Set de test, para realizar los submits a la plataforma de la competencia: test_values.csv
- Ejemplo de submit: submission_format.csv

Página de copetencia: https://www.drivendata.org/competitions/57/nepal-earthquake/

Para verificar los resultados, en la competencia se usa la métrica **micro averaged F1 Score**, usada en métodos de clasificación cuando hay más de dos clases. El rango del puntaje es [0, 1], y mejor es el puntaje cuanto más cerca de 1 se encuentra.

Objetivos

La idea es utilizar los pasos y métodos vistos en clase para dar robustez al modelo. Se pretende usar varios algoritmos para analizar los resultados y el desempeño de los mismos durante el trabajo. Se busca experimentar con métodos de optimización para la búsqueda de hiper parámetros óptimos. Finalmente se desea desarrollar una serie de conclusiones y también contemplar ideas que pudieron haber dado frutos de haberlas puesto en práctica.

Preprocesamiento de datos

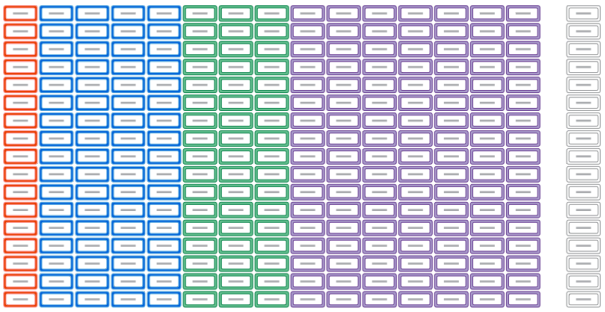

Figura 1: Representación de set de datos inicial y los labels.

Se cambiaron los tipos de datos para disminuir el uso de memoria. La mayoría de los datos se pueden usar con tipos numéricos. Hay atributos de tipo categórico que necesitan ser codificados.

Features descartadas

Si bien building_id es necesaria para el submit, resta puntaje a los modelos entrenados. Esto tiene sentido, ya que son números únicos y pueden no tener ningún orden de asignación. Se percibe como un valor aleatorio que incorpora ruido al modelo.

Luego de entrenar con el modelo Random Forest se puede verificar la importancia de una feature. Lo mismo se puede hacer con XGBoost y LightGBM. Como se usó este último modelo para probar resultados en la competencia, y se obtuvieron buenos números, se utilizaron las importancias obtenidas mediante este modelo. Luego, con un criterio para determinar el valor de importancia de las features, se formó un criterio de selección a modo de filtro. Las columnas que no obtuvieron el nivel de importancia requerido se descartaron.

El criterio usado para descartar features fue construido con los valores de importancia que el modelo de LightGBM registró por los splits que realizaba. El modelo informaba las cantidades de veces que se usaba una columna para hacer las divisiones. Como se trata de una composición de muchos árboles, puede haber muchas divisiones realizadas por valores de distintos atributos en distintos niveles de cada árbol. Con estas variaciones, las columnas que acumulan más usos para splits se consideran más importantes.

Una vez conseguido el vector de importancias se lo normaliza y se planta un nivel de tolerancia para la importancia. Usamos un nivel de 0.001 como importancia mínima para mantener los atributos.

Features categóricas

Los sets de datos tienen variables categóricas y es necesario aplicarles alguna codificación para que los modelos de Random Forest, XGBoost y LightGBM funcionen sin inconvenientes. En este caso hay ocho columnas categóricas que necesitan ser codificadas.

Se probó con Binary Encoding y se crearon 28 columnas más. También se probó con One Hot Encoding y se crearon 38 columnas más.

Con Binary Encoding el mejor resultado obtenido, después de muchas pruebas y con la métrica de evaluación de la plataforma, fue 0.7227 con LightGBM. Con XGBoost se obtuvieron mejores resultados con el uso de One Hot Encoding. Cuando se pasó a esta misma codificación con LightGBM, se lograron las mejores predicciones. Así que se continuó con One Hot Encoding.

Feature Engineering

Debido a la elección de modelos basados en árboles, se experimentó con algunas transformaciones y operaciones entre features. Estos modelos no pueden calcular relaciones entre columnas.

Lo más directo es la introducción de relaciones entre atributos. Se consideró la elaboración de tasas entre variables. Por ejemplo: altura por antigüedad, pisos por antigüedad, altura por área, familias por área, familias por antigüedad del edificio, cantidad de familias en edificación sobre cantidad de edificios por nivel de región geográfica, área de edificación por media de antigüedad en nivel geográfico, antigüedad de edificación por cantidad de edificaciones en nivel geográfico, etc.

Luego de calcular estas features e incorporarlas en el set de datos, la mayoría acumuló mucha importancia proveniente de las que antes eran muy relevantes. Pero no mejoraron la predicción final. Esto, sumado a que los resultados obtenidos mejoraron con el tratamiento de las columnas ya incluídas, derivó en la optimización del modelo con los atributos originales.

Modelos usados

Random Forest

Se utilizó como modelo inicial para obtener un panorama del problema y poder formatear los datos para el entrenamiento. Se comenzó utilizando una cantidad reducida de seis Features, al igual que en el tutorial de la plataforma Driven Data (Benchmark). Estos eran tres de tipo numérico, dos de tipo binario y uno de tipo categórico. Para tratar a este último se utilizó One Hot Encoding. El score obtenido fue de 0.5708 con el set de entrenamiento, menor al 0.5815 con el set de test del tutorial. Se prosiguió variando a mano algunos hiper parámetros. En la primera prueba se utilizaron 20 árboles de decisión (*n_estimators*), por lo que se probó variar esta cantidad. Cuando disminuyó, también lo hizo el score. Al aumentar la cantidad, el score mejoraba hasta cierto punto. Se decidió dejar esta cantidad por el momento en 80. Durante estas pruebas también se usó un valor de 5 para *min_samples_leaf*. Como seguía sin superar el Benchmark, se decidió incluir más Features. Para este punto había 4 de tipo numérico, 12 de tipo binario y 5 categóricos. De esta forma se obtuvo un score con el set de entrenamiento de 0.6039, superando el tutorial.

Después se decidió agregar los Features categóricos restantes, y los Features numéricos 'age' y 'geo_level_1_id'. El score obtenido se incrementó muy notablemente a 0.6824, dejando claro que estos Features numéricos poseen mucha importancia. Este incremento del score fue el más alto registrado en todo el trabajo práctico. Luego, se incluyó a los Features restantes y también se probó codificar a los atributos categóricos con Binary Encoding. Incluso se usaron 200 estimators. Pero el score en la plataforma no superó 0.6890 (con el set de test). En este punto se decidió que era mejor utilizar otros modelos mejores. De todas formas, Random Forest funcionó bien para las primeras pruebas y sirvió para identificar a los Features Importantes.

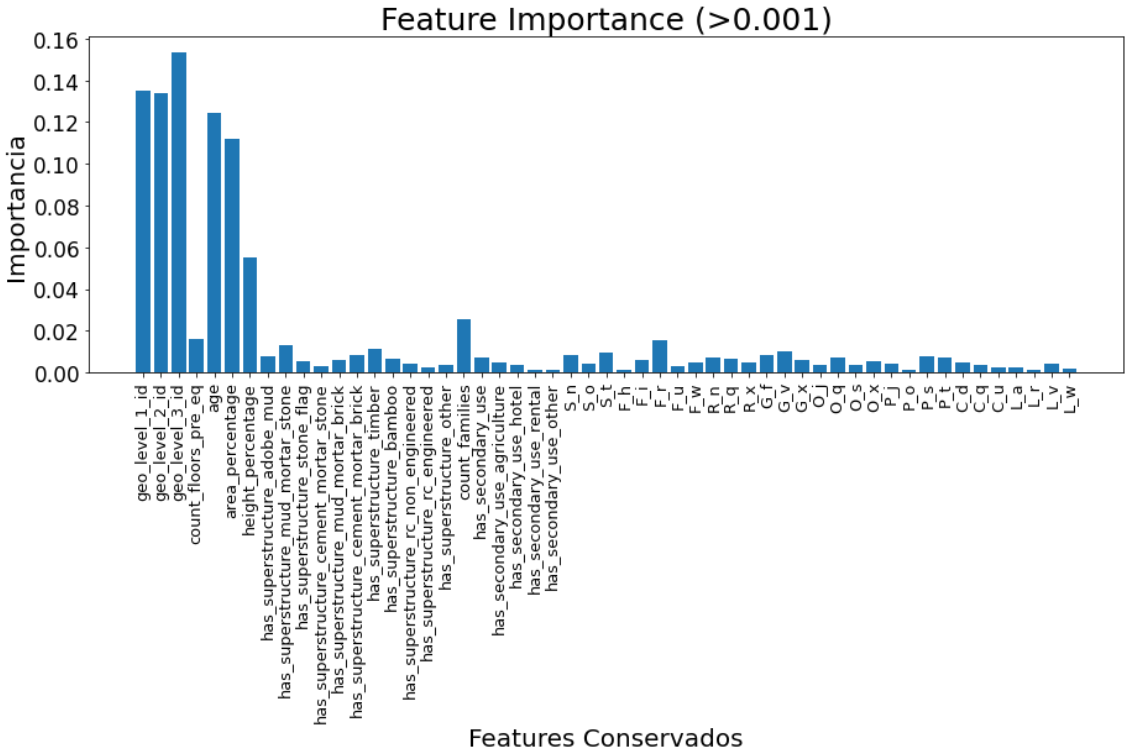

Figura 2: Cantidad de importancia que aportan las columnas conservadas.

XGBoost

Debido al estancamiento del score con el modelo de Random Forest, se decidió realizar pruebas utilizando otros modelos, entre ellos XGBoost. Al comienzo se probó este modelo con una cantidad reducida de Features, 4 de tipo numérico, 12 de tipo binario y 5 categóricos. Y sorprendentemente el score (0.5740) era menor al obtenido con el modelo anterior (0.6039) a pesar de estar utilizando la misma muestra de features. Se probó entonces aumentar *n_estimators* de 10 a 100, *col_sample_bytree* con 0.4, *learning_rate* con 0.2, *max_depth* con 5 y *alpha* en 10, pero todavía no superaba el score anterior. Solo recién cuando se incluyeron features importantes como 'geo_level_1_id' se logró superar (levemente) al modelo anterior con 0.6846 frente al 0.6824 del Random Forest. Estas comparaciones se realizan sobre el score del set de entrenamiento y con ambos modelos utilizando los mismos features.

A partir de este punto se fueron optimizando a mano los hiper parámetros, obteniéndose cada vez mejores resultados. Se aumentó el número de *n_estimators* a 200 y se obtuvo un score de 0.7205. Luego aumentando a 500 y con un valor de 0.3 para el *learning_rate* se mejoró el score a 0.7395. Haber dejado de lado los Features binarios de usos secundarios con menor importancia contribuyó un poco a mejorar los resultados. Estos resultados fueron obtenidos con el set de test. En este punto, como optimizando a mano no se obtenían mejoras significativas, se realizó una búsqueda automatizada de hiperparámetros por medio de Grid Search. La grilla de valores abarcaba 500, 550 y 580 para *n_estimators*, 0.4, 0.5 y 0.6 para *colsample_bytree* y 0.2, 0.3 y 0.4 para *learning_rate*. El entrenamiento duró seis horas y arrojó que los mejores valores eran 580, 0.6 y 0.3 respectivamente para los hiper parámetros anteriormente mencionados. El proceso fue fructífero ya que mejoró el score a 0.7434 con el set de test.

LightGBM

Al igual que el modelo anterior, es un modelo de Boosting muy ágil y poderoso. Se comenzó usando los valores por defecto del clasificador y se obtuvieron resultados superiores a Random Forest, alrededor de 0.72 de F1 Score. La forma de especificar los hiper parámetros es igual a la de los modelos anteriores.

Se filtraron features por importancia y se comenzó una serie de entrenamientos simples. Luego de una pequeña optimización se alcanzó el mejor puntaje. Desde aquí se continuó con la optimización con K-Fold Cross Validation para distintos valores de K. Más adelante se muestran algunos resultados.

Para la optimización se probó con el siguiente conjunto de parámetros: *learning_rate*, num_leaves, *max_depth* y *num_estimators*. Dependiendo del tamaño del set, determinado por el número de divisiones del Fold, se obtuvieron varias combinaciones de parámetros. Entre todas las configuraciones registradas se hicieron 570 evaluaciones con esta librería, que consumieron 19 horas.

Apreciaciones

Los modelos se pueden guardar ya entrenados. Para LightGBM los modelos más pesados eran de alrededor de 40 MiB (10-Fold Cross Validation, entrenados con *num_leaves*: 123, *n_estimators*: 1065, *max_depth*: 36). Los más livianos pesaron alrededor de 14 MiB (9-Fold Cross Validation, entrenados con *num_leaves*: 59, *n_estimators*: 755, *max_depth*: 59).

Se realizaron entrenamientos con el mismo set de datos entre LightGBM y XGBoost y en general se notó que el modelo desarrollado por Microsoft (LightGBM) era más rápido.

Optimización

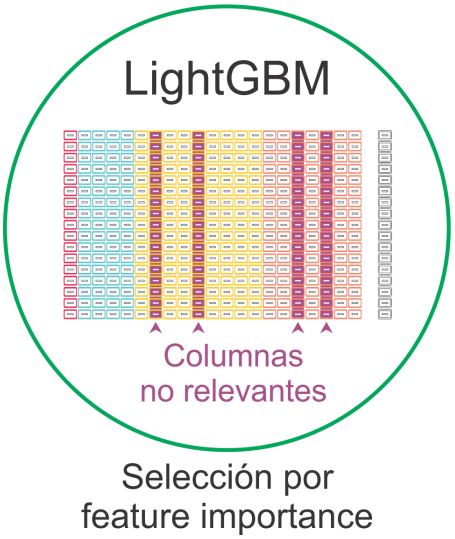

Figura 3: Modelo entrenado con un set de datos pre-procesado.

Luego del entrenamiento, el modelo basado en árboles de decisión puede informar la importancia de las columnas. Con esta información se pueden filtrar las features según sus importancias. A partir de este paso se comenzaron a probar métodos de optimización.

A continuación se mencionan diferentes mecanismos en donde se variaron los hiper parámetros para optimizarlos. El orden en que aparecen coincide con el orden en que fueron utilizados para las búsquedas. El orden también va de menor a mayor complejidad y eficiencia.

Manual

Este es un paso directo de optimización. Uno prueba modificando los parámetros en el constructor del modelo. Pero rápidamente se vuelve tedioso. Los siguientes métodos tienen como objetivo la asistencia a la búsqueda de parámetros.

Grid Search

Aquí se definió una grilla con valores para los hiper parámetros y el método entrenó un modelo para cada combinación posible de los mismos. Al final de las pruebas se procede a cambiar los valores de la grilla para buscar mejorar los resultados.

Random Search

En este caso se definió una distribución de valores para los hiper parámetros. Luego el método procede de manera similar a Grid Search, solo que entrena una cantidad de modelos menor a todos los posibles, eligiendo combinaciones entre los valores de forma aleatoria. Así realiza pruebas y elige nuevas distribuciones para mejorar los resultados.

Grid Search y Random Search fueron métodos sencillos de entender para la introducción a la optimización de parámetros automatizada.

Búsqueda Bayesiana

Se usaron tres módulos para trabajar con Búsqueda Bayesiana: Scikit-Optimize, HyperOpt y Optuna. El procedimiento de puesta en marcha es similar con los tres.

Se tiene una función a minimizar y un espacio de parámetros. La función a minimizar se arma de forma tal de poder recibir el set de datos, los labels y, dependiendo de la implementación del módulo, nombres de parámetros y/o espacio de parámetros. En este caso lo que se quiere minimizar es la métrica usada para evaluar el modelo. Como se emplea la métrica F1 Score, al ser un valor que mejora al acercarse a 1 hay que multiplicar por (-1). Entonces, mientras más chico sea, es decir más negativo, se tendrá un mejor resultado desde el modelo.

En Optuna solo se necesita definir el espacio de parámetros en la definición de función a minimizar. El espacio de parámetros cumple la función de grilla de parámetros o distribución de parámetros.

Scikit-Optimize

La introducción al método de búsqueda Bayesiana se realizó con el método *gp.minimize* del módulo Scikit-Optimize. A partir de este paso en las pruebas para optimización

HyperOpt y Optuna

Son módulos que implementan el método de optimización de hiper parámetros por Bayesiana. HyperOpt es muy conocido y utilizado. Se usó el método *fmin* para iniciar la búsqueda de mejores resultados.

Optuna es una librería más nueva y cuenta con más documentación. Con esta opción hay que definir un objeto Study con la orden para minimizar. Se trabaja sobre la instancia de Study y la optimización comienza aplicando el método *optimize* a la misma.

Es con el objeto Study que se puede analizar el avance en la búsqueda. Tiene un método que devuelve un DataFrame con las evaluaciones realizadas. Allí se informan el número de evaluación (las iteraciones empiezan desde 0), el tiempo que duró el entrenamiento, los parámetros buscados, el puntaje logrado.

Por menciones en diversas fuentes (blogs, videos de difusión de Data Science) se usaron en gran medida estos módulos. Sobre todo se usó Optuna, por la facilidad para definir la función a optimizar, la reducción de parámetros usados en dicha función y la claridad con la que se puede analizar el avance de la optimización.

Los hiper parámetros que se usaron para realizar tunning son:

- *num_leaves*
- *n_estimators*
- *max_depth*
- *learning_rate*

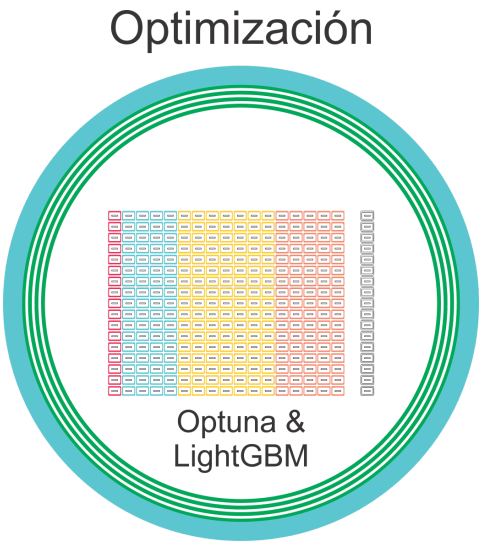

Figura 4: Luego de obtener un set de datos modificado para un entrenamiento efectivo, se procede a la optimización. El módulo Optuna se representa con la circunferencia más gruesa de color truquesa. Aquí se usan muchas circunferencias interiores verdes para simbolizar el uso de LightGBM en varios ciclos de evaluación.

Para optimizar se usó el notebook: *LightGBM_Optuna.ipynb*

Apreciaciones

Las siguientes apreciaciones se consiguieron usando búsqueda Bayesiana con HyperOpt y, en gran medida, Optuna.

Con Optuna, en algunas iteraciones los mejores resultados de las métricas obtenidas permitían identificar diferentes regiones de mejora. Es decir, que se podían realizar optimizaciones con varios espacios de parámetros. En estos casos se pueden realizar búsquedas separadas para cada espacio. Así se pueden encontrar algunas tendencias que son más favorables que otras en alguna de las regiones. De encontrar una diferencia se puede ahondar en esa región para lograr resultados que justifiquen su elección.

Mientras se avanza en las búsquedas, los rangos de valores para los parámetros se van refinando. El intervalo de valores a revisar se achica y es necesario disminuir el tamaño del paso. Por ejemplo, si se está buscando por el parámetro de número de estimadores:

- número de estimadores: (200, 1500, 100) permite tomar entre 13 valores;
- número de estimadores: (550, 800, 25) permite tomar entre 10 valores;
- número de estimadores: (625, 775, 10) permite tomar entre 15 valores;
- número de estimadores: (685, 705, 2) permite tomar entre 10 valores;
- número de estimadores: (693, 701, 1) permite tomar entre 8 valores;

Es muy útil tener opciones de persistencia. Uno puede modificar los espacios de búsqueda entre sesiones de entrenamiento para optimización.

Resultados de optimizaciones

Resultados de optimizaciones realizadas. Duración es el tiempo en segundos que tardaron las evaluaciones consideradas.

Optimización para 5-Fold Cross Validation

Iteración	F1 Score	Duración	learning_rate	max_depth	n_estimators	num_leaves
123	0.745358	113	0.069143	61	880	112
125	0.745277	113	0.070696	61	870	112
135	0.745212	131	0.069161	61	870	138
111	0.745201	119	0.070074	63	860	122
110	0.745185	121	0.071428	59	870	122
33	0.745143	108	0.083838	75	850	110
12	0.745024	113	0.070147	50	1050	80
92	0.745024	122	0.066258	64	925	115
75	0.745009	106	0.071106	62	800	115
44	0.744997	116	0.060453	60	750	140

Tabla 1: Evaluaciones de optimización para 5-Fold Cross Validation. Total: 140 iteraciones.

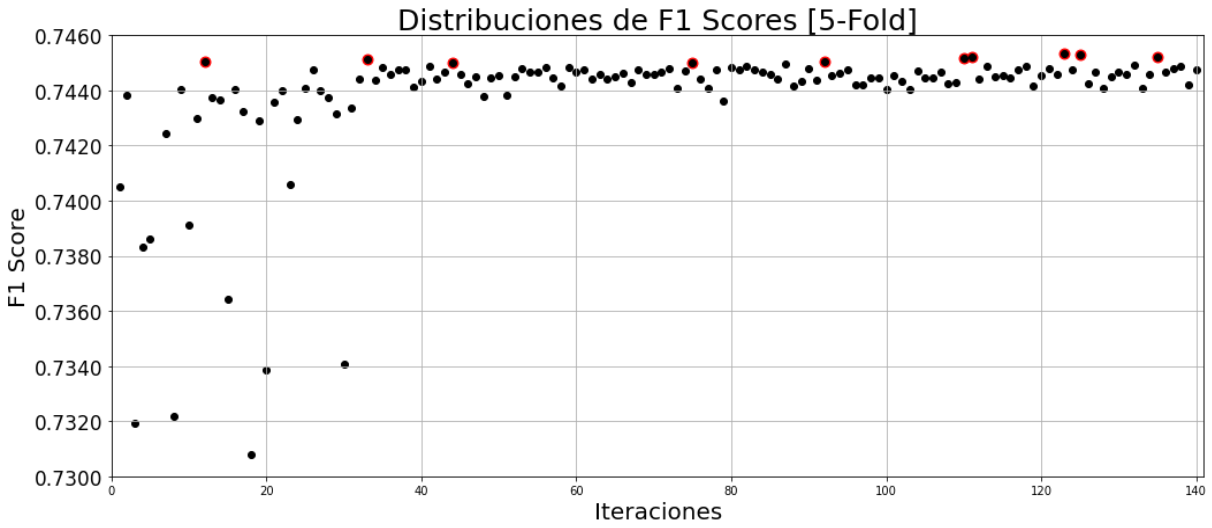

Figura 5: Se quitó el valor más bajo para desplegar los puntos. Se resaltan las diez evaluaciones que mejor puntúan.

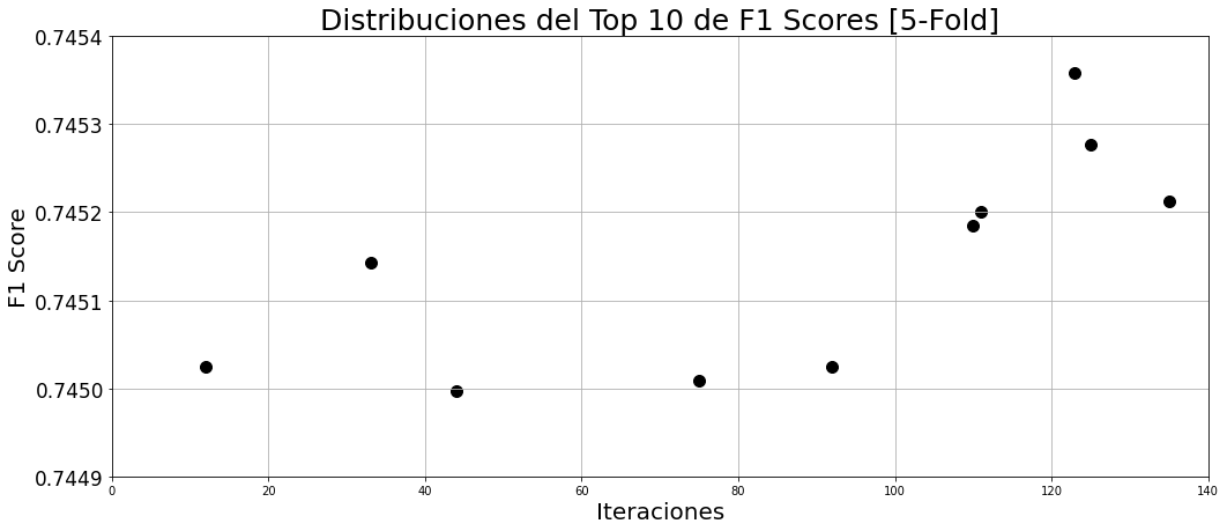

Figura 6: Se muestran las mejores diez evaluaciones para optimizaciones de 5-Fold.

En este caso, hasta la iteración 80 hubo dos regiones en las que los híper parámetros parecían converger. Se realizó una serie de iteraciones para un espacio de parámetros similar al de la iteración n° 11 y se obtuvieron tres mejoras, pero en posiciones bajas del top 10 hasta ese

momento. La única que quedó luego de probar con las del otro espacio es la 11. Con los nuevos rangos de parámetros se obtuvieron mejores resultados entre las iteraciones 100 y 120. Luego se realizó una nueva búsqueda para confirmar esta tendencia y el top 3 tiene mejora de esta última sesión.

Optimización para 6-Fold Cross Validation

Iteración	F1 Score	Duración	learning_rate	max_depth	n_estimators	num_leaves
41	0.745838	134	0.067163	60	620	168
28	0.745822	125	0.071980	55	600	170
66	0.745803	131	0.076449	68	640	167
77	0.745761	137	0.072644	61	638	169
63	0.745726	136	0.075836	68	630	169
31	0.745696	127	0.077294	65	600	170
69	0.745607	130	0.075368	58	605	170
40	0.745565	134	0.068490	60	630	168
50	0.745542	139	0.061789	55	630	172
57	0.745515	135	0.060640	50	630	172

Tabla 2: Evaluaciones de optimización para 6-Fold Cross Validation. Total: 80 iteraciones.

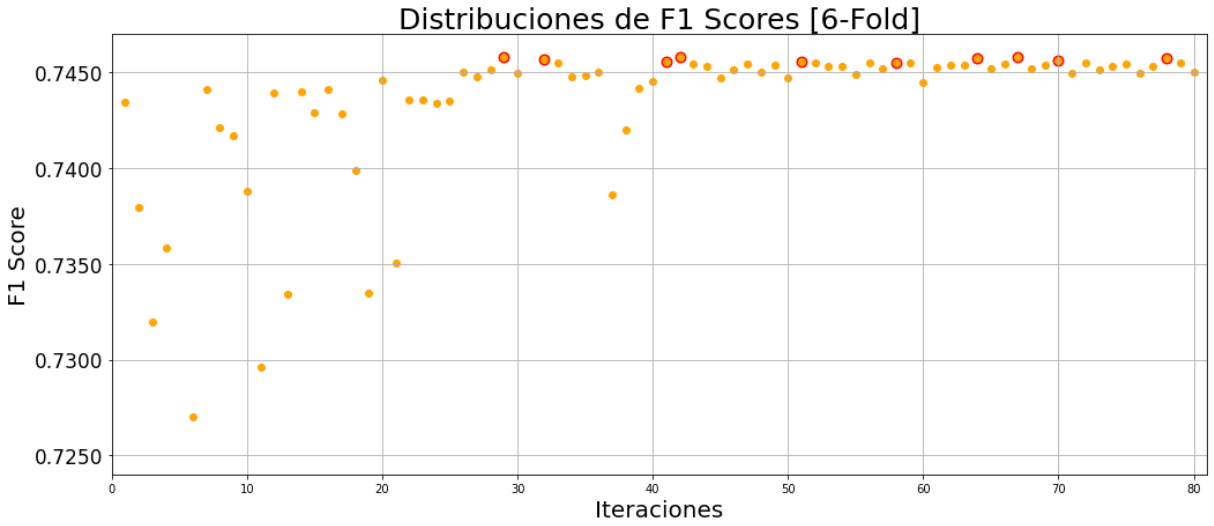

Figura 7:Se quitó el valor más bajo para desplegar los puntos. Se resaltan las diez evaluaciones que mejor puntúan.

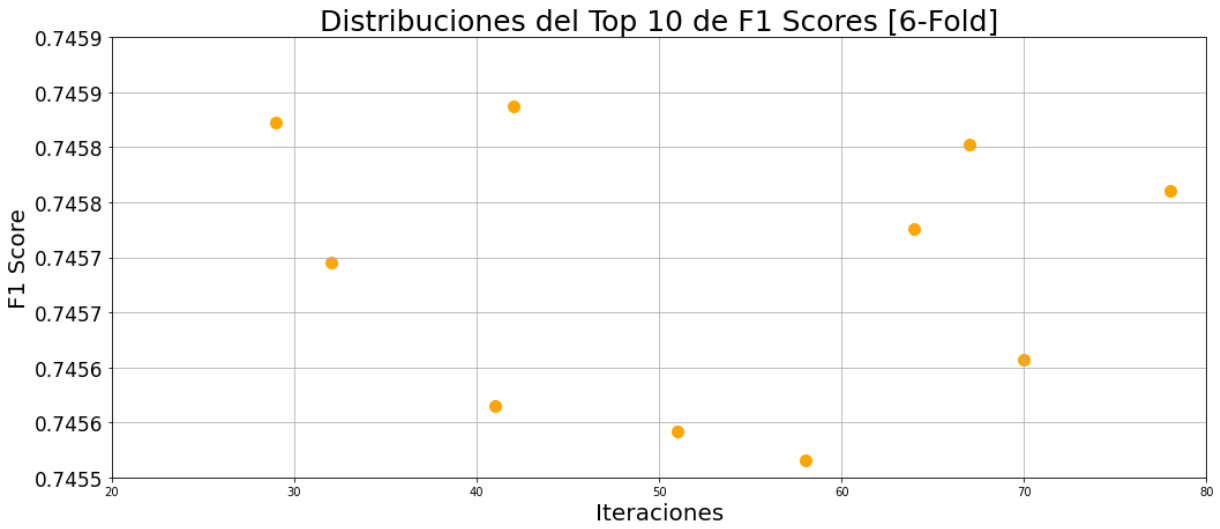

Figura 8: Se muestran las mejores diez evaluaciones para optimizaciones de 6-Fold.

Optimización para 7-Fold Cross Validation

Iteración	F1 Score	Duración	learning_rate	max_depth	n_estimators	num_leaves
53	0.746087	145	0.077811	85	830	106
52	0.745910	152	0.077106	90	870	107
15	0.745795	105	0.134264	55	560	110
60	0.745768	162	0.087114	90	890	113
47	0.745768	146	0.090898	80	840	107
86	0.745726	161	0.078081	92	875	112
64	0.745665	171	0.079717	90	930	113
68	0.745653	163	0.084628	85	920	111
45	0.745642	149	0.088214	90	890	107
44	0.745626	159	0.086915	90	940	107

Tabla 3: Evaluaciones de optimización para 7-Fold Cross Validation. Total: 100 iteraciones.

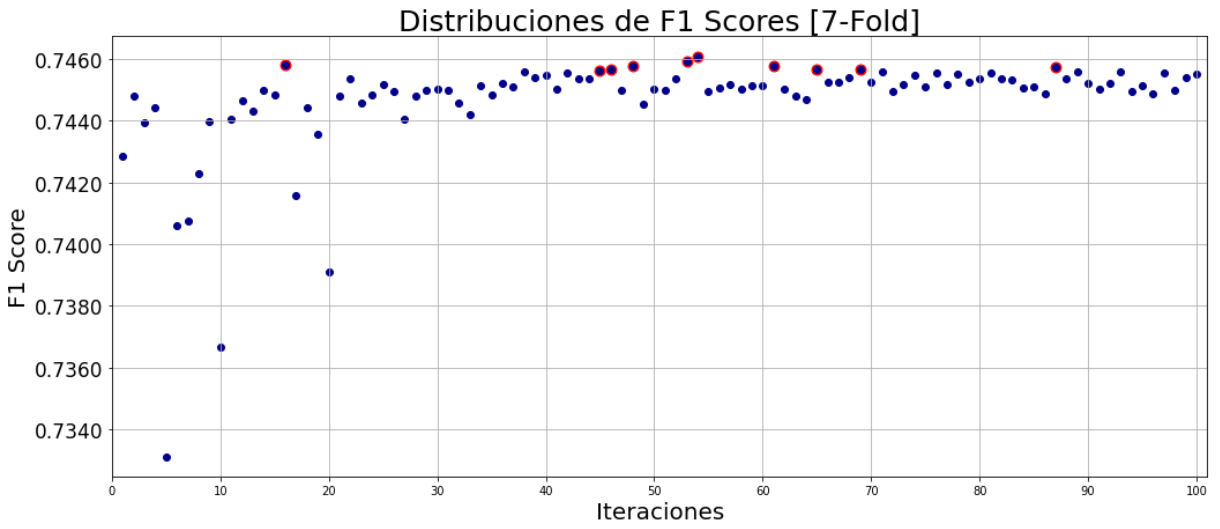

Figura 9: Se resaltan las diez evaluaciones que mejor puntúan.

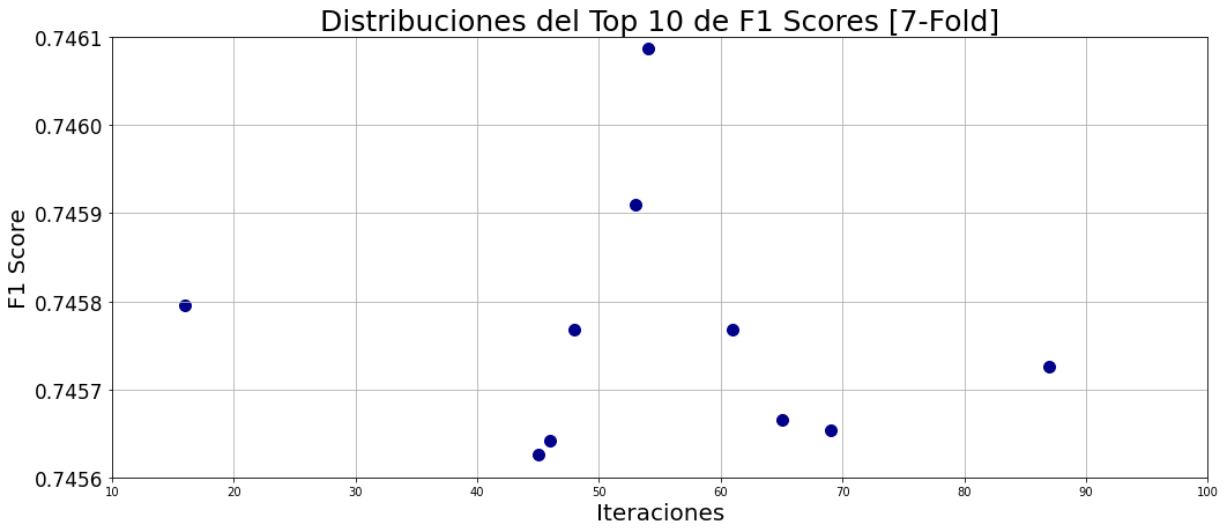

Figura 10: Se muestran las mejores diez evaluaciones para optimizaciones de 7-Fold.

Optimización para 8-Fold Cross Validation

Iteración	F1 Score	Duración	learning_rate	max_depth	n_estimators	num_leaves
63	0.745853	103	0.124505	52	620	82
52	0.745519	112	0.127113	57	620	88
74	0.745496	106	0.139288	52	600	79
39	0.745488	108	0.124221	59	625	87
58	0.745392	109	0.128586	53	625	82
88	0.745392	107	0.129808	54	624	81
80	0.745385	104	0.128156	59	602	82
67	0.745358	99	0.138673	54	600	79
43	0.745323	116	0.128069	60	625	93
60	0.745273	104	0.131844	51	600	81

Tabla 4: Evaluaciones de optimización para 8-Fold Cross Validation. Total: 90 iteraciones.

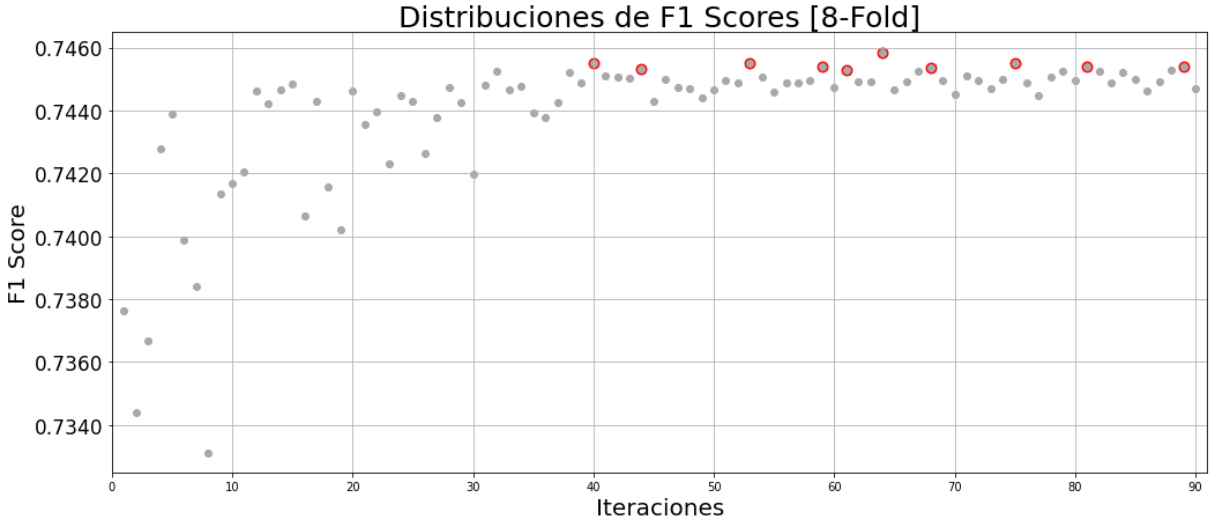

Figura 11: Se resaltan las diez evaluaciones que mejor puntúan.

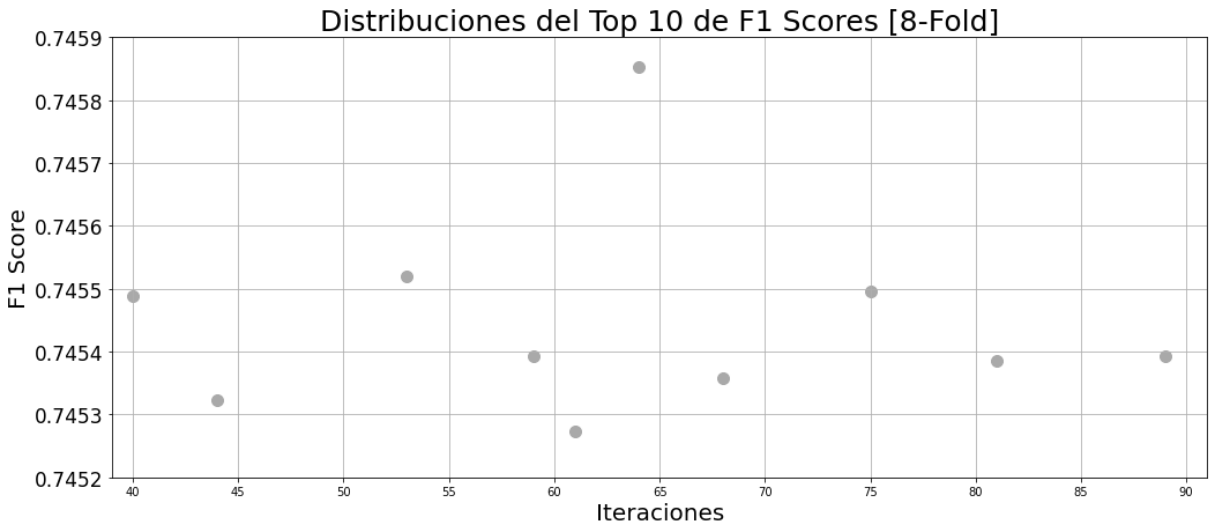

Figura 12: Se muestran las mejores diez evaluaciones para optimizaciones de 8-Fold.

Optimización para 9-Fold Cross Validation

Iteración	F1 Score	Duración	learning_rate	max_depth	n_estimators	num_leaves
48	0.746168	121	0.156732	59	755	59
54	0.745891	127	0.154063	43	785	61
39	0.745876	111	0.143274	42	690	58
37	0.745864	109	0.142617	44	665	58
58	0.745807	141	0.157941	41	795	64
56	0.745703	122	0.158248	41	800	53
42	0.745603	116	0.166596	41	735	58
27	0.745588	111	0.141549	44	625	72
41	0.745561	115	0.164910	43	725	58
55	0.745554	123	0.153417	45	790	61

Tabla 5: Evaluaciones de optimización para 9-Fold Cross Validation. Total: 60 iteraciones.

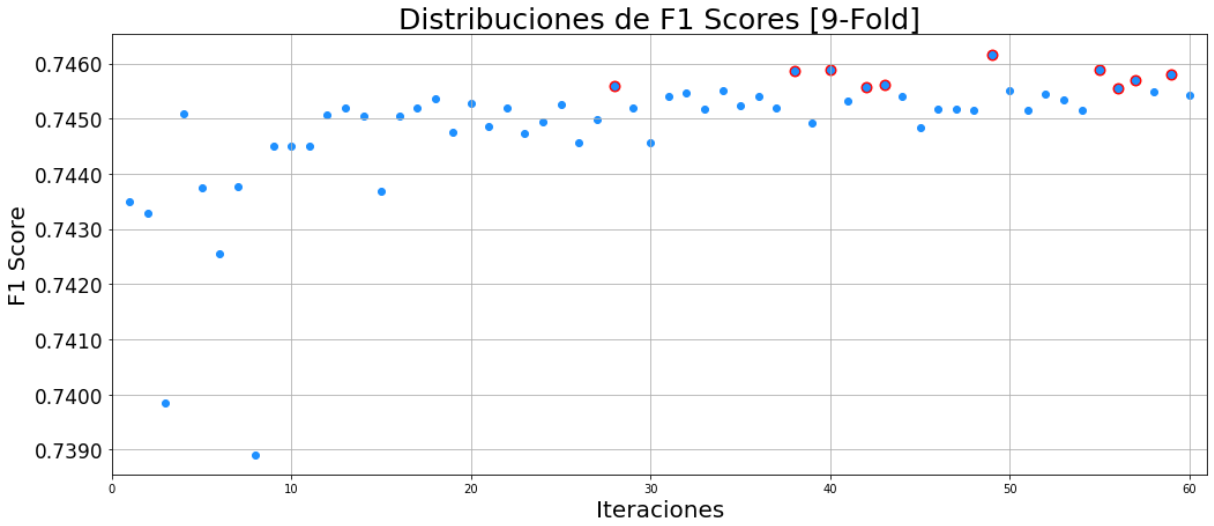

Figura 13: Se resaltan las diez evaluaciones que mejor puntúan.

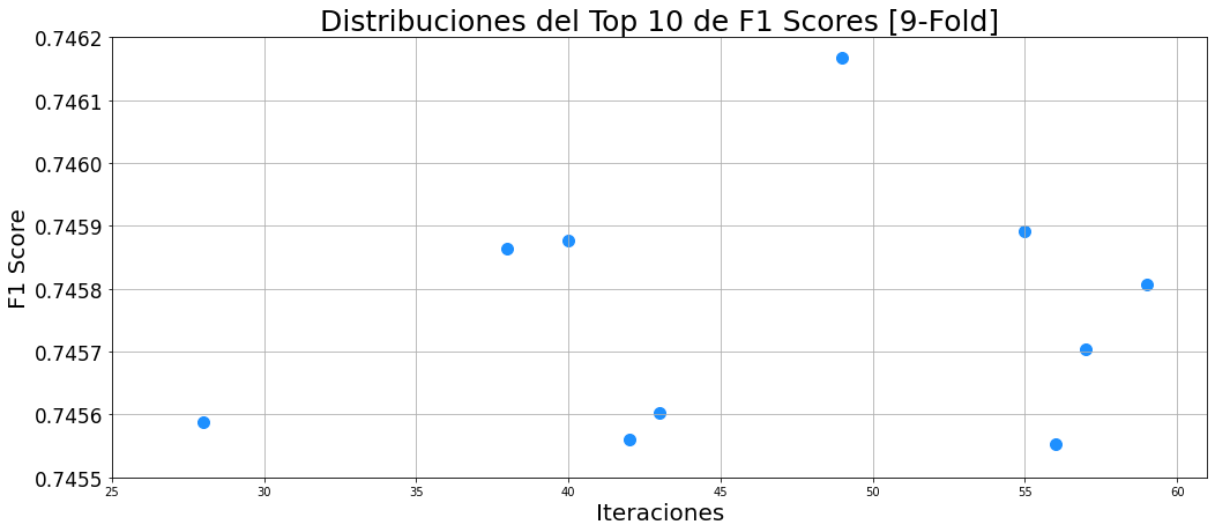

Figura 14: Se muestran las mejores diez evaluaciones para optimizaciones de 9-Fold.

Optimización para 10-Fold Cross Validation

Iteración	F1 Score	Duración	learning_rate	max_depth	n_estimators	num_leaves
87	0.746283	304	0.094065	36	1065	123
93	0.746144	271	0.096061	33	1035	111
95	0.746068	302	0.089759	38	1145	114
98	0.746068	293	0.089326	28	1155	108
80	0.745933	297	0.092172	37	1120	114
83	0.745903	261	0.092550	36	1095	114
82	0.745845	301	0.091553	35	1045	123
99	0.745841	300	0.089135	25	1190	108
81	0.745765	273	0.092103	27	1045	111
96	0.745742	303	0.089803	38	1155	114

Tabla 6: Evaluaciones de optimización para 10-Fold Cross Validation. Total: 100 iteraciones.

Esta optimización se cortó en 100 iteraciones porque requería un promedio de cinco minutos por evaluación. Son más de mil estimadores por modelo y se tienen que entrenar diez veces por evaluación. Es probable que se pueda optimizar más.

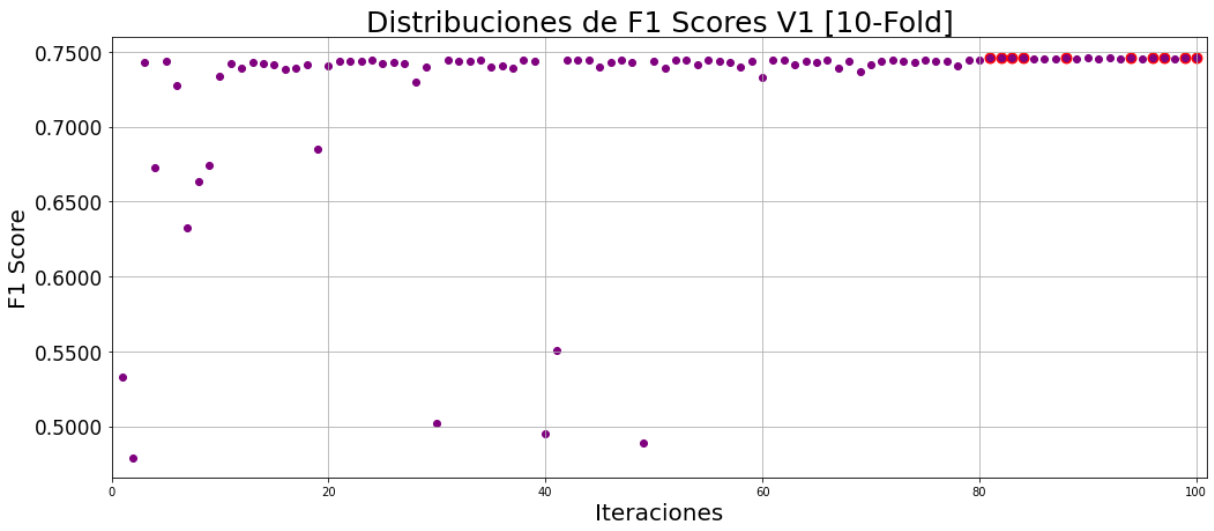

Figura 15: Se resaltan las diez evaluaciones que mejor puntúan.

Esta fue la primera optimización realizada en grandes lotes. Al ser 10 Fold el set de datos usado para entrenar es más grande, entonces se esperaba una duración mayor que en otros Folds. Resultó que además fue el experimento que más estimadores utilizó. Así que fue la más costosa.

No se modificaron los rangos de búsqueda durante muchas de las evaluaciones, por eso aparecen algunos valores bajos en iteraciones avanzadas.

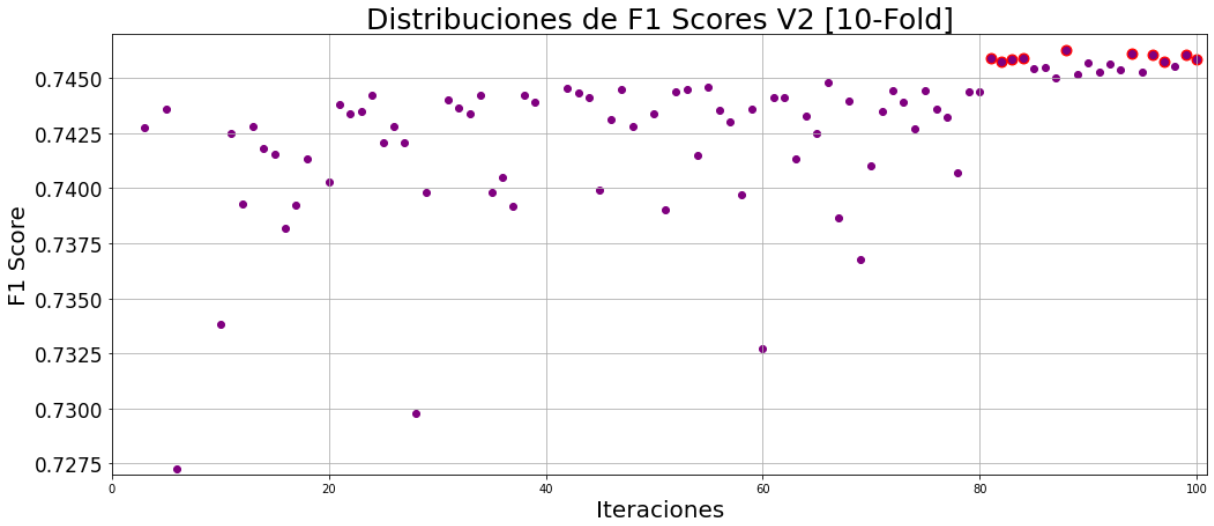

Figura 16: Aquí se modificó el eje de Scores para poder visualizar mejor los puntos. Se resaltan las diez evaluaciones que mejor puntúan.

En este gráfico se ven mejor las evaluaciones con buen puntaje. Se nota un salto de calidad en la optimización a partir de la iteración 80. Aquí se inició una última ronda de iteraciones con rangos de hiper parámetros ajustados según usados por los mejores puntajes.

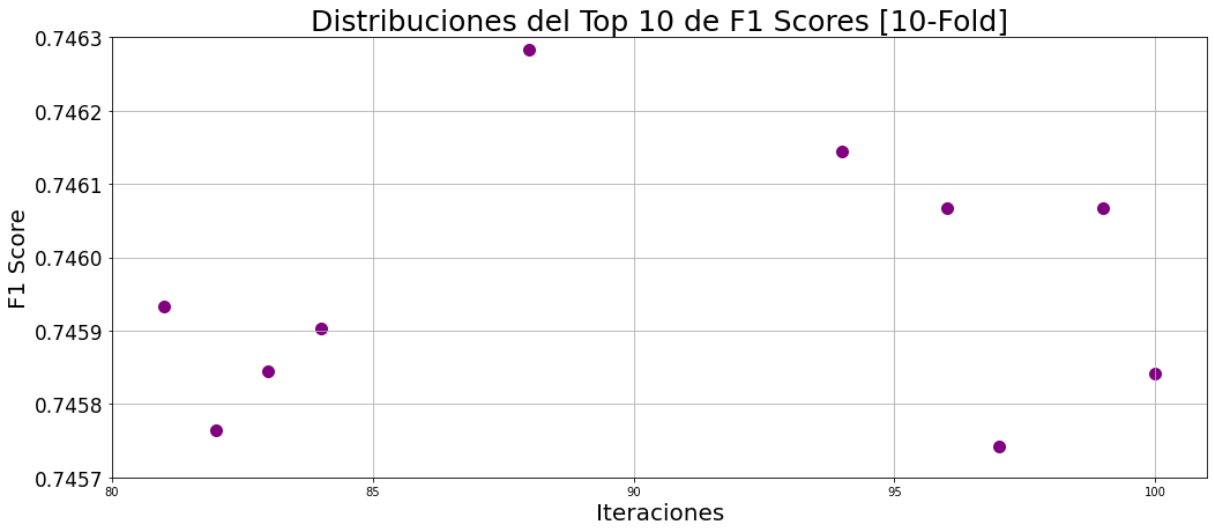

Figura 17: Se muestran las mejores diez evaluaciones para optimizaciones de 10-Fold.

En todas las visualizaciones dispersión completa de puntajes se puede ver cómo el módulo encargado de la optimización tiende a encaminar el proceso de optimización hacia mejores resultados. Como es de esperar, los puntajes más altos se acumulan hacia las últimas iteraciones (los puntos resaltados en rojo se acumulan hacia la derecha, hacia donde crece el número de iteración). Esto se manifiesta de forma clara en la figura 16.

Estadísticas de mejores resultados

K (Folds)	F1 Score		
	Media	Mediana	Desvío
5	0.745143	0.745164	0.000125
6	0.745687	0.745711	0.000122
7	0.745764	0.745747	0.000143
8	0.745448	0.745392	0.000162
9	0.745762	0.745755	0.000197
10	0.745959	0.745918	0.000176

Tabla 7: Valores estadísticos del conjunto de mejores 10 resultados para las optimizaciones de cada K-Fold usado.

En general, el promedio de puntajes tiende a subir con el aumento de Folds. Al usar un set de datos más filas el modelo entrena con más información. En la figura 18 se ve que esta tendencia suele mantenerse. Solo el caso de 8-Fold parece anormal, ubicándose sólo por encima de 5-Folds.

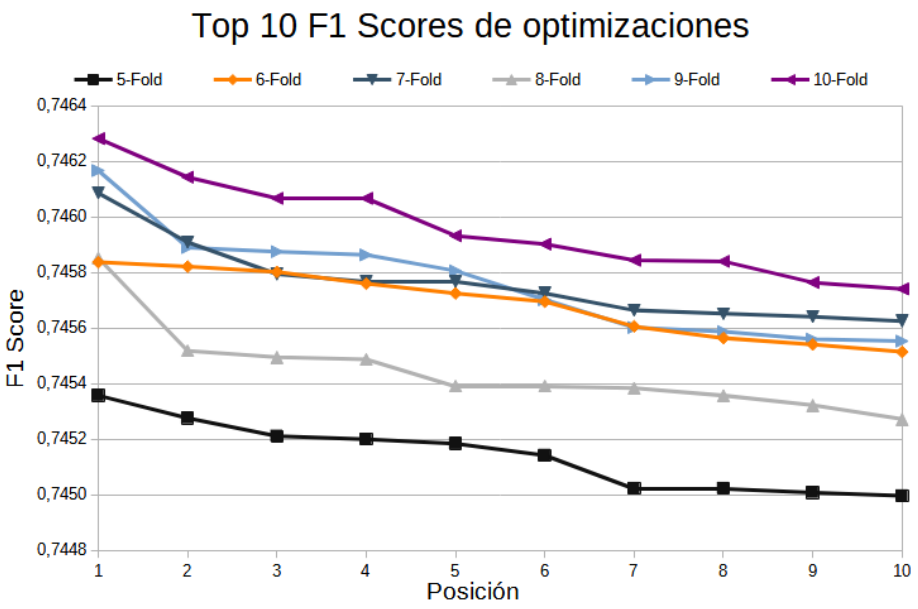

Figura 18: Posicionamiento de los 10 mejores resultados para cada optimización.

Se puede notar que la optimización de 8-Folds está por debajo de 6 y-Folds. Quizás se puede seguir con la optimización o ampliar los rangos de búsqueda para hallar mejores regiones.

Para decidir dejar de optimizar consideraron los siguientes criterios:

- Se compararon los scores con el mejor puntaje de la competencia de Driven Data;
- Se compararon los resultados con otros modelos entrenados ya optimizados;
- Luego de las iteraciones asignadas, se verificó si había scores que justificaran otra sesión de optimización (en 7-Fold no hubo muchas mejoras hacia el final de la optimización. Figura 9);
- Si se alcanza un valor aceptable y la optimización demanda mucho tiempo.

Pipeline elegido

LightGBM es un modelo de Boosting basado en árboles de decisión, por lo cual ya funciona como un ensamble. Lo que funcionó para conseguir el mejor resultado, en este caso, fue realizar Cross Validation para diferentes números de Folds y combinar las predicciones a través de la moda. El mejor puntaje obtenido fue 0.7487, con la moda de las predicciones de 10, 9, 8 y 7-Folds.

El pipeline y los pasos tomados para llegar al mejor resultado se describe a continuación:

- Pre-procesamiento de datos.
    - Se modificaron los tipos de datos de las features y los labels.
    - Se trataron las columnas categóricas con One Hot Encoding.
- Selección de features.
    - Se aplicó un entrenamiento con un modelo basado en árboles de decisión para encontrar la importancia de las columnas.
    - En base a un criterio, se procedió a descartar las columnas no consideradas importantes.
- Optimización.
    - Se eligieron rangos de valores para los hiper parámetros a optimizar.
    - Se realizaron sesiones de optimización entre las cuales se fueron ajustando los rangos de hiper parámetros.
- Predicción.
    - Se entrenaron modelos para la predicción a subir a la plataforma con los hiper parámetros encontrados.
    - Se aplicó el mismo pre-procesamiento y selección de features usado para el set de entrenamiento al set de evaluación.
    - Se realizaron las predicciones y se subieron a Drive Data.
- De aquí en adelante se puede volver a revisar cualquiera de los pasos anteriores.

Para entrenar los modelos se usó el notebook: *LightGBM_board.ipynb*

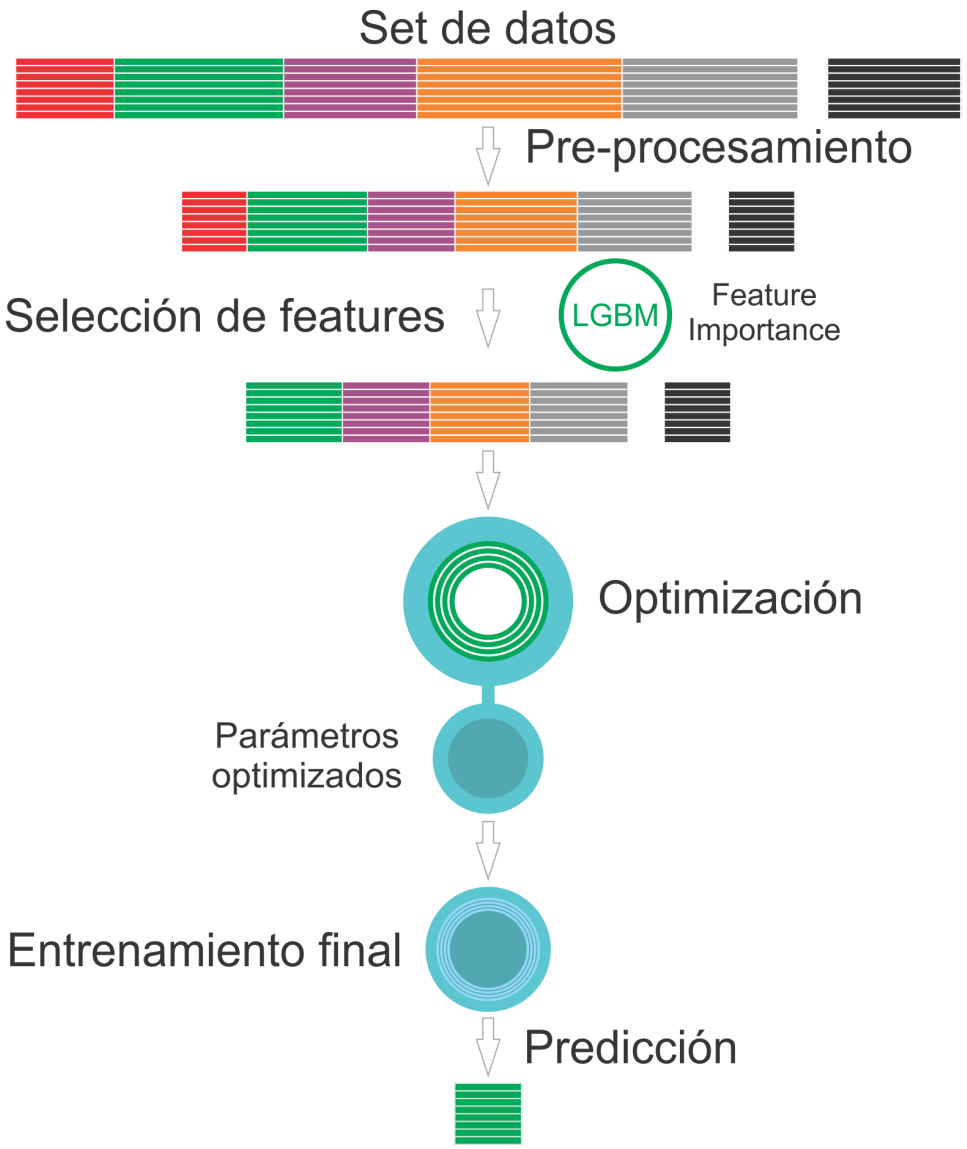

Figura 19: Diagrama de pipeline. Al final del entrenamiento se calculan las predicciones y se almacenan en un DataFrame. Luego se calcula la moda de una selección de modelos.

Conclusiones

- Aún luego de decidir terminar con las iteraciones, queda claro que el modelo o la solución al problema planteado se puede mejorar. Nuestro mejor score fue de 0.7487 y el que encabeza la competencia vale 0.7558.
- Se puede entrenar más modelos, más simples o complejos, para ensayar un ensamble.
- Quedó por realizar un análisis de clustering para ver si en el set de datos hay alguna serie de clusters que mejoren las predicciones.
- Se pueden probar más hiper parámetros. Los usados aquí fueron los que mostraron mejoras que acercaban los scores al del líder de la competencia.
- La optimización por búsqueda Bayesiana fue la que mejor desempeño tuvo. Entre Hyperopt y Optuna, optamos por esta última y no tuvimos ningún inconveniente (tampoco con Hyperopt). Optuna pareció más rápida y con una respuesta al usuario más intuitiva.
- Para la optimización de parámetros se podrían elaborar gráficos de dispersión de puntos para ubicar los parámetros en regiones observables. Así pueden verse más intuitivamente zonas con más densidad de puntos para casos en donde aparecen varias tendencias y los rangos no se achican.
- Los hiper parámetros con buenas evaluaciones que quedan fuera del rango común de la búsqueda pueden usarse para buscar otras regiones de optimización. Se pueden usar como puntos centrales de otros espacios. Con esto se puede encontrar un nuevo lugar de búsqueda o se pueden descartar, con más evidencia, estos parámetros aislados.
- Algunas optimizaciones se pueden profundizar. Hay algunas que obtuvieron las mejores puntuaciones en las últimas iteraciones. En este caso se puede seguir optimizando para averiguar si el procedimiento alcanzó un techo de mejora. Hay otras pruebas que tienen pocas iteraciones.
- Hacia el final del trayecto quedó claro que la parte que más tiempo lleva es la de optimización. Además, en este trabajo no se pudo encontrar atributos nuevos a partir de las columnas dadas. Seguramente con un análisis más profundo y exhaustivo se pueda dar con algunas que funcionen satisfactoriamente.