# Trabajo Práctico N°1 - Grupo 11

## Análisis Exploratorio

El dataset original consta de unas 460.154 registros y 20 columnas. Todos estos registros resultan ser anuncios de propiedades, ya sea en venta, alquiler o alquiler temporal, no solo en la Argentina, sino incluso en Uruguay y otros países. Además, hay anuncios no solo para casas, PHs y departamentos, sino también para lotes, locales comerciales, depósitos, oficinas, casas de campo, cocheras y otros.

Dentro de los features más destacables del dataset hallamos:

- `operation`: Detalla el tipo de operación bajo la cual está listada cada propiedad  Es una variable categórica cuyo valores incluyen: venta, alquiler, alquiler temporal. En este caso particular, sólo nos interesaban las propiedades en venta.
- `property_price`: Indica el precio de una propiedad. Es el valor que deseamos predecir. Es una variable cuantitativa continua.
- `property_currency`: Detalla el tipo de moneda del precio de la publicación Es una variable categórica cuyos posibles valores incluyen: ARS, USD. Solo nos interesan las propiedades listadas en USD.
- `property_surface_total`: Indica la superficie total de la propiedad. Es una variable cuantitativa continua.
- `property_rooms`: Detalla la cantidad de habitaciones en la propiedad. Es una variable cuantitativa discreta.
- `place_l3`: Renombrada a `neighbourhood`, describe el barrio donde se encuentra la propiedad. Es la variable categórica que más llama la atención, ya que la información acerca de la ubicación suele ser de suma importancia cuando se busca comprar una propiedad.

Supuestos e hipótesis tomados:

- Los datos de superficie están en metros cuadrados.
- Los precios indicados en dólares efectivamente estaban en dólares.

# Preprocesamiento de Datos

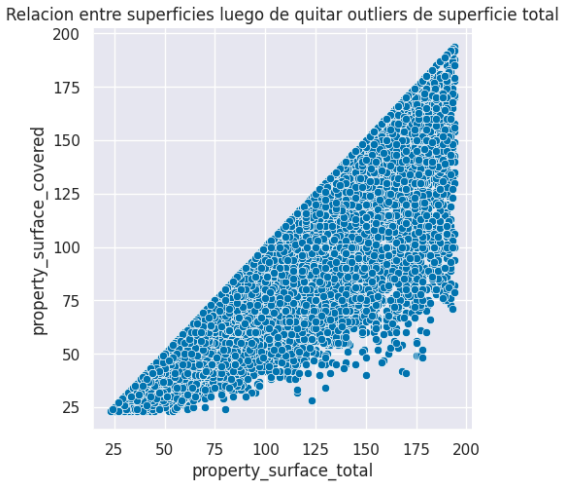
En el preprocesamiento de datos se decidió eliminar las columnas `place_l4`, `place_l5`, `place_l6` debido a que muy pocas de las propiedades de interés contaban con información

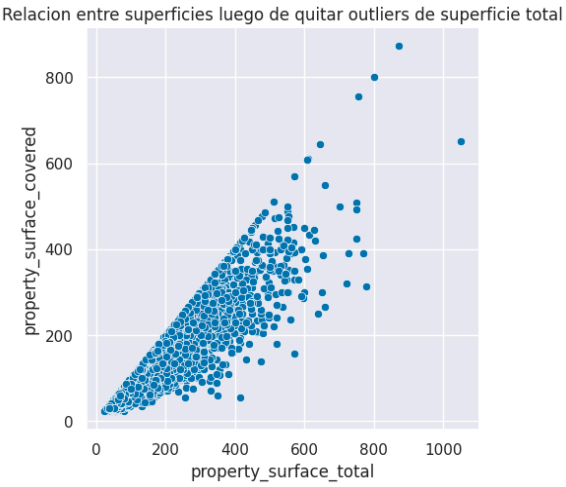
alguna en estos campos. De igual forma, se decidió eliminar las columnas `place_l2`, `operation` y `property_currency` debido a que todas estas columnas quedaban solamente con registros de un mismo valor después de hacer el filtrado del dataset (Capital Federal, Venta y USD respectivamente).

Se detectó que las variables `property_rooms` y `property_bedrooms` tenían una correlación muy elevada entre ellas, de 0,87. Previo al análisis de outliers e imputación de datos faltantes, las otras variables que resultaron tener una correlación algo elevada son `property_surface_total` y `property_surface_covered`, con un valor de 0,6. Si bien luego de hacer el manejo de datos nulos y el análisis de outliers en estas dos últimas columnas, su correlación incrementó hasta 0,85, optamos por conservar ambas columnas porque a nivel de concepto, nos parece significativo hacer una distinción entre el área cubierta y total de una propiedad, por ejemplo, esto sería muy significativo cuando se compra una casa con patio.

Durante el tratamiento de los análisis nulos de las variables `property_surface_total` y `property_surface_covered` se experimentó utilizando un método de imputación basado en un modelo de regresión lineal que predijera el valor de esta última a partir de la superficie total y el número de cuartos. Para mejorar la correlación entre las variables se probó escalando las variables de superficie para minimizar la importancia de las diferencias entre las unidades de medida de las cantidad de cuartos y las superficies. Se terminó hallando que la mejor opción era aplicar el logaritmo a las superficies. Sin embargo, finalmente se terminó optando el método MICE por practicidad, ya que el modelo de regresión también requería de un análisis de nulos adicional para evitar logaritmos de valores nulos o ceros.

Con respecto al análisis de outliers, se decidió utilizar el algoritmo de Local Outlier Factor para automatizar la detección de outliers multivariados. Utilizamos Grid Search con K-Folds Cross-Validation para determinar los parámetros óptimos de este modelo, y al filtrar el dataset a través del mismo, se terminaron detectando aproximadamente 10.000 outliers que optamos por dropear. Cabe destacar que previo a la inclusión de este algoritmo de detección de anomalías el único análisis de outliers multivariado que se estaba haciendo era manual, y al comparar las distribuciones del antes y después de aplicar el algoritmo LOF, notamos distribuciones que quedaron mucho más naturales al utilizar el algoritmo en comparación con el análisis manual, que provocaba abruptas cotas en las distribuciones de algunas propiedades. Por ejemplo, los siguientes gráficos muestran los gráficos de dispersión entre las variables de área, a la izquierda cómo quedaban después del filtro de outliers manuales, y a la derecha, cómo quedaban luego de que el modelo los analizara:

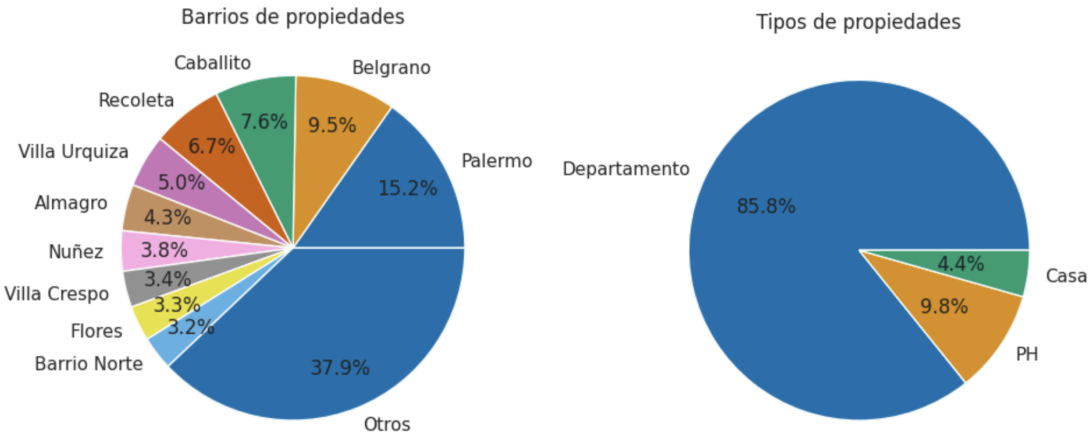



Se utilizaron las técnicas de boxplots e IQRs para el análisis de los outliers univariados, destacando el rol de estos filtros especialmente en el caso del análisis de los outliers de las cantidades de cuartos, que si bien no contaban con outliers con valores drásticos previos al filtro, si había un muy reducido grupo de propiedades que salían fuera de los rangos esperados. Contrario a cómo había sucedido inicialmente cuando se hacía el análisis multivariado de manera manual, no se encontraron muchos casos particulares que se pudieran analizar individualmente como habían sido las situaciones en las que coincidían los datos erróneos de propiedades del mismo edificio.

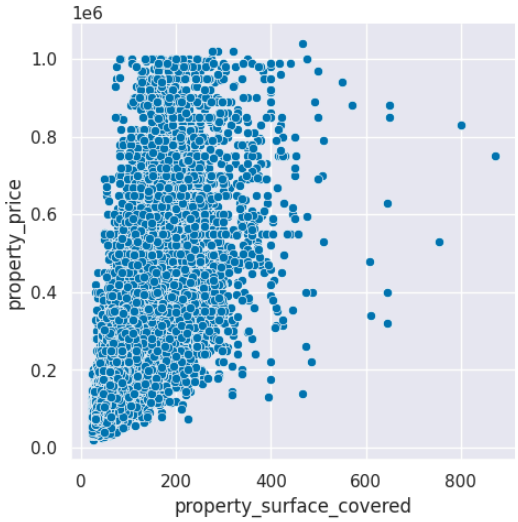
# Visualizaciones

Los gráficos de torta nos resultaron una forma útil de analizar las distribuciones de las variables categóricas del tipo de barrio y el tipo de propiedad.




Los datos más importantes que obtenemos de estas visualizaciones son que hay muestras de propiedades de una gran cantidad de barrios de la ciudad, por lo que no nos estamos concentrando únicamente en un grupo selecto de barrios. Y, principalmente, que la gran mayoría de propiedades totales de dataset son departamentos.

Otra visualización interesante para comprender el problema es el gráfico de dispersión entre el precio de las propiedades y la cantidad de superficie cubierta.




La variable `property_surface_covered` resultó ser la que más correlación tenía con el precio, existiendo un coeficiente de correlación de valor 0,78, positivo entre ambas variables, tal y como se aprecia en el gráfico.

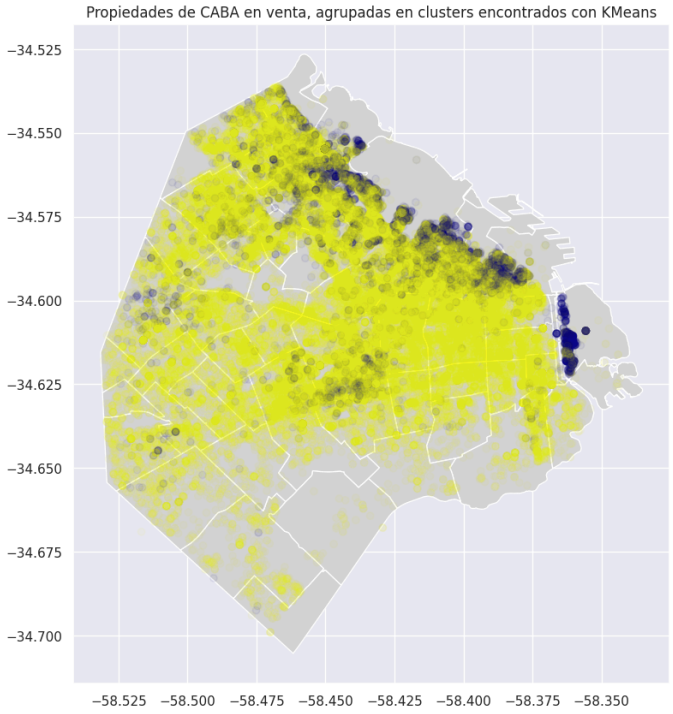
# Entrenamiento de Modelos

## Clustering

Pudimos descubrir mediante la estadística de Hopkins que el dataset presenta una fuerte tendencia al clustering, con un valor de alrededor de 0,00477 (para la implementación de esta métrica en librería `pyclustertend`, cuanto más cercano a 0 es más fuerte es la tendencia al clustering).

Luego, para hallar estos clusters, se utilizó el algoritmo de K-Means.

La cantidad apropiada de grupos que se deben formar resultó ser 2, llegamos a esta cantidad de grupos analizando el score de Silhouette para valores desde 2 clusters hasta 9 clusters y analizando y comparando el score de cada grupo. Este score cae cuánto más se incrementa la cantidad de grupos, siendo como se mencionó antes el más elevado de 0,74 aproximadamente para 2 grupos.



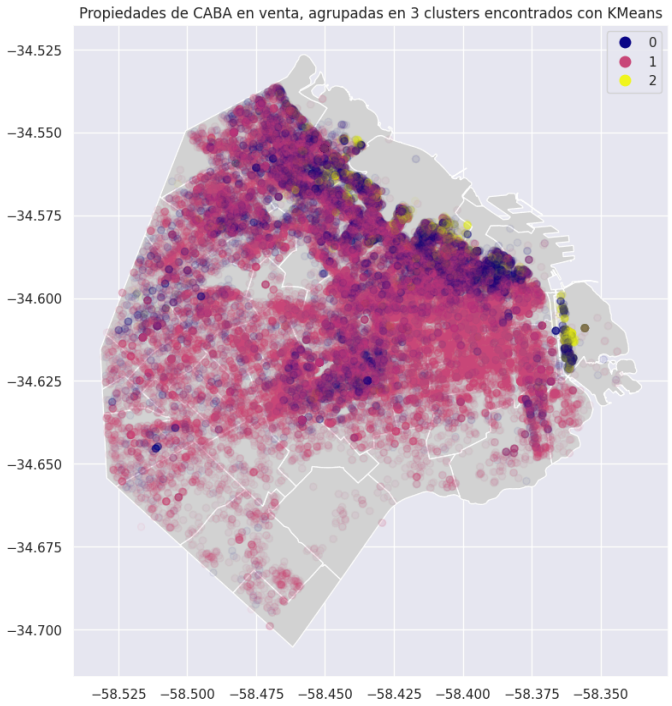

La diferencia más llamativa entre ambos clusters son los rangos de precios que abarca cada uno, siendo los precios del segundo cluster más altos que los del primero. Además estos rangos de precios son disjuntos, es decir, los precios de las propiedades de un clúster no se superponen con los precios de las del otro, generando así una separación total. Finalmente, notamos que las estadísticas de las demás columnas son similares entre las propiedades de ambos grupos, salvo por el precio y ligeramente por las medias de los datos de las áreas de las propiedades, ya que las propiedades del cluster más caro también suelen ser más espaciosas, con una media superficie cubierta de 150 metros cuadrados, contra una media de 58 metros cuadrados para las propiedades más baratas.

Con toda esta información, concluimos que el primer cluster está conformado por las propiedades más costosas y generalmente más espaciosas, que resultan también ser las menos numerosas; mientras que el segundo cluster está formado por las propiedades menos costosas y más pequeñas, que resultan conformar la mayor parte del dataset.

Cuando realizamos el clustering con 3 grupos, parecería suceder el mismo fenómeno anterior: los grupos se forman, principalmente, en función del precio de los anuncios. Lo hacen también marcando claros grupos de precios:

- Cluster 0: Precios de 220.772 a 500.000
- Cluster 1: Precios de 20.000 a 220.590
- Cluster 2: Precios de 500.600 a 1.040.000

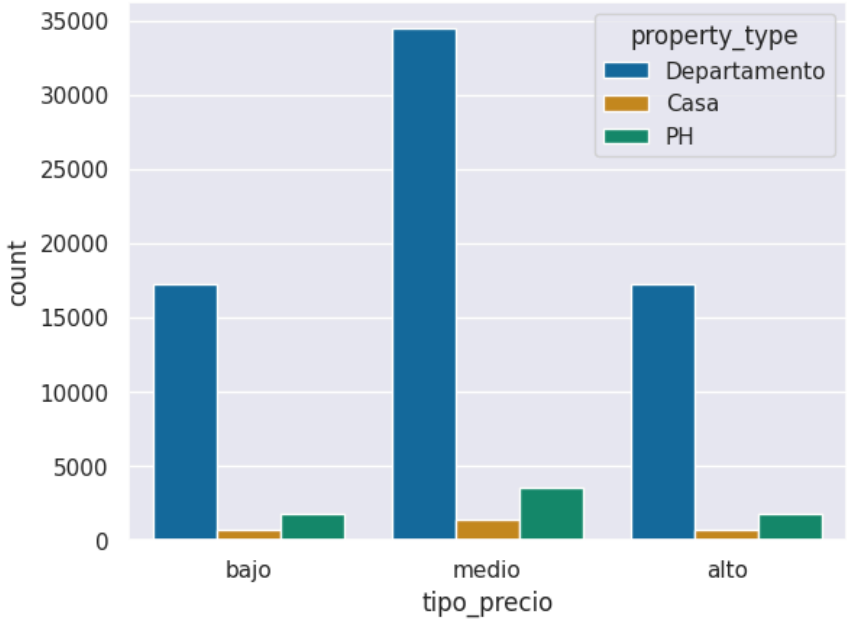
Es destacable que el grupo más numeroso es el de precios bajos, que agrupa 59,2k propiedades. Luego le seguirán el cluster de precios medios, con 15,9k propiedades; y el de precios altos, con 3,7k propiedades. Respecto de las demás columnas, los promedios mantienen la misma relación que cuando se usaron dos clusters. A mayor precio, más espaciosas son las propiedades en promedio.




## Clasificación

Optamos por utilizar la tercera estrategia de división propuesta, que consiste en trabajar la variable precio por metro cuadrado relativa a cada tipo de propiedad y luego dividirla en tres intervalos, el primero con el 25% de las observaciones, el segundo con el 50% y el último con el 25% restante. El motivo de esta elección fue principalmente la proporción entre la cantidad de departamentos comparada con los otros dos tipos de propiedad, por lo que si hacíamos la división de grupos en conjunto, los percentiles iban a estar altamente influenciados por los precios de los departamentos, y los precios

de las casas y PHs iban a tener poca relevancia. En el gráfico siguiente, se ilustran las distribuciones de los tipos de precio por propiedad con la estrategia elegida.




Notamos que a diferencia que cuando usamos KMeans con 3 grupos, la forma elegida para hacer la división de los tipos de precio hace que el grupo que contiene los precios intermedios sea el más numeroso. Sin embargo, cuando usamos KMeans, los otros dos clusters eran mucho más pequeños, y no llegaban a contener el 25% de los datos.

Otra diferencia muy clara es que cuando hicimos KMeans, la distribución de los tipos de propiedades era distinta en cada cluster. Usando este otro método, nos aseguramos que la distribución de tipos de propiedades de cada grupo sea exactamente la misma.

## Árbol de Decisión

Optimizamos los hiperparámetros mediante la utilidad de sklearn `RandomizedSearch` y optamos por no utilizar `GridSearch`, ya que resultó muy costosa en cuanto a tiempos de ejecución, llegando a tardar 1 hora o más para una grilla que no contemplaba todos los hiperparámetros posteriormente descritos. Los parámetros que buscamos optimizar son `criterion` (si se utiliza Gini o Entropía), `ccp_alpha` (parámetro de poda), `min_samples_split` (número mínimo de muestras que se requieren en un nodo antes de que se pueda dividir en nodos hijos adicionales),

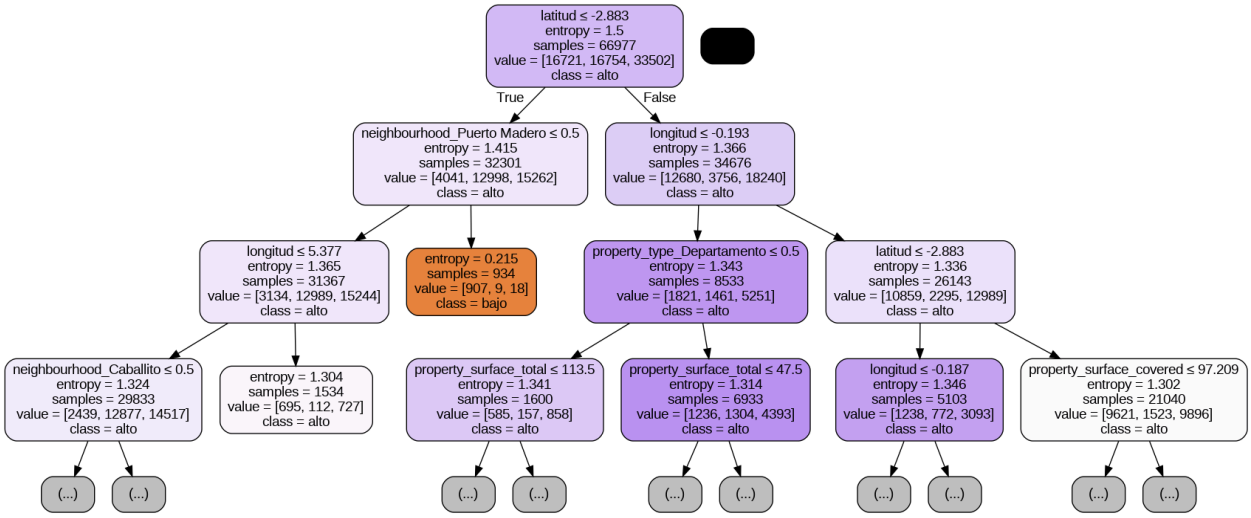
`min_samples_leaf` (número mínimo de muestras que se requieren en un nodo hoja), y `max_depth` (la profundidad máxima del árbol).

Utilizamos stratified K-Folds cross validation para que los folds que se generan estén balanceados respecto a las clases, contando con 5 folds.

Para buscar los hiperparámetros optimizados utilizamos la métrica de f1 score ya que, basándonos en la teoría, es la que nos permite minimizar los falsos positivos y negativos, así como maximizar los verdaderos positivos.

Para el preprocesamiento, se optó por utilizar One-Hot encoding para los barrios y para el tipo de propiedad, ya que previamente se utilizó `LabelEncoding` pero resultaba algo engorroso saber qué significaba cada número que se le asignaba a los barrios a ojo.

A continuación podemos ver el árbol que seleccionamos de todos los que se entrenaron:




En el mismo, la latitud del aviso toma mucha importancia a la hora de clasificar y junto con ella también la longitud, de modo que la ubicación de un aviso resulta de amplia importancia para predecir el tipo de precio que tendrá.

## Random Forest

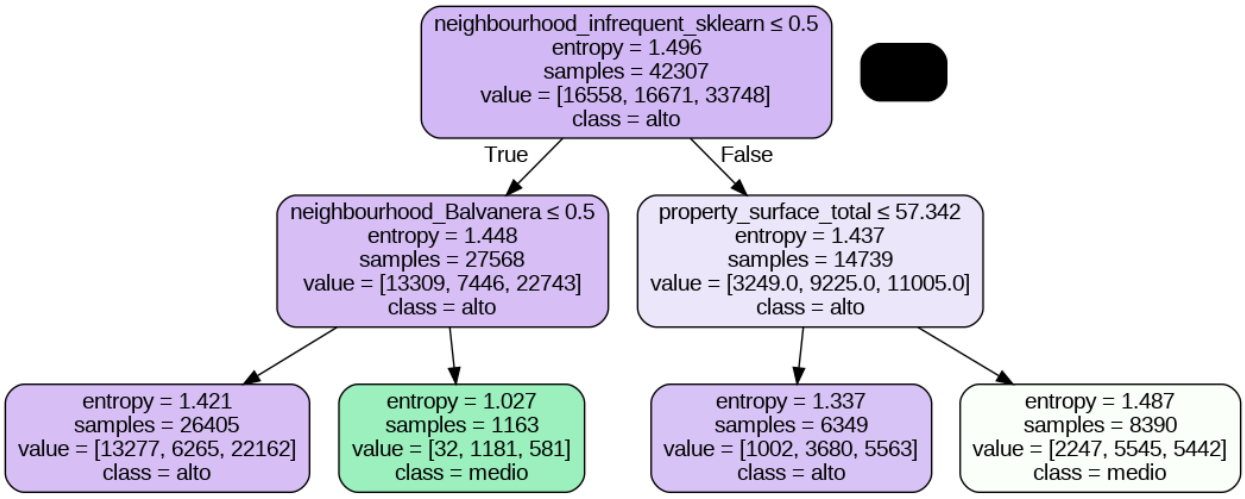
Para entrenar el modelo de Random Forest, usamos K-Folds Cross Validation para la búsqueda de hiperparámetros, con 5 folds. Usamos Stratified K-Folds en lugar de solo K-Folds porque hay una distribución dispareja en las categorías del target. Por cómo lo armamos, hay más propiedades con la categoría medio que de las otras dos. Con lo cual, queremos tratar de mantener esa proporción en cada fold.

Los hiperparámetros a optimizar fueron:

- Número de árboles.
- El criterio a utilizar para medir la calidad de cada *split*.
- Profundidad máxima.
- Número mínimo de samples para un *split*.
- Número máximo de features a considerar para realizar un *split*.

Consideramos que estos hiperparámetros eran los más relevantes para el entrenamiento del clasificador. Para buscarlos, consideramos que la métrica más adecuada era la de f1 score, que considera tanto la precisión del modelo, como su recall.

Se muestra a continuación la conformación de uno de los árboles generados por el modelo:



Pareciera que la misma se vio demasiado afectada por las columnas relacionadas a los barrios, lo cual no es óptimo, puesto que ignora gran parte de los datos de los que disponemos.

## Support Vector Machine

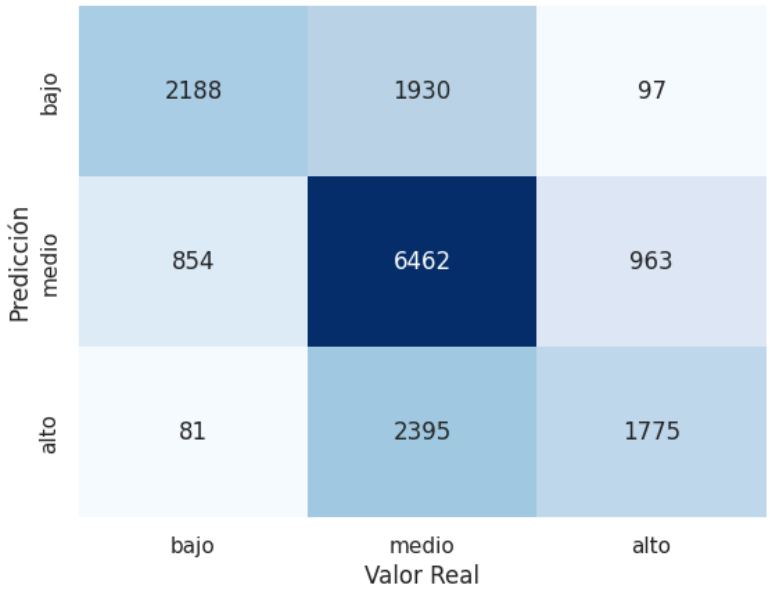
Se utilizó Grid Search con 5-fold cross-validation para encontrar los valores óptimos de los hiperparámetros dentro de los valores propuestos para probar. En este caso se se probó con diferentes valores de la constante `C` para cada uno de los diferentes kernels probados, entre los cuáles están el kernel lineal, polinómico, de grados 2 a 4, y finalmente el kernel RBF, para el que también se probaron diferentes valores del parámetro Gamma.

Los valores de la constante fueron propuestos de tal forma que se abarcara un amplio rango de valores. Esta constante sirve como parámetro de regularización para evitar el overfitting del algoritmo, algo especialmente importante para el kernel polinómico, ya que a mayores grados del polinomio más se tiende al overfitting, razón por la cuál se limitaron las pruebas hasta un polinomio de grado 4.

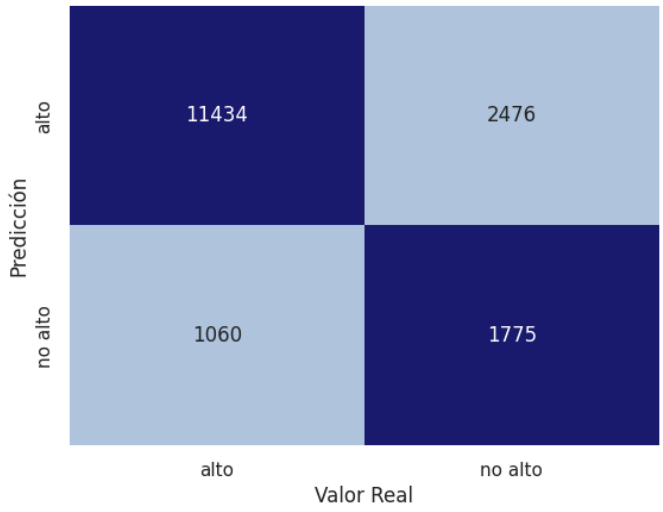
Los valores del parámetro Gamma para el kernel RBF fueron seleccionados según las sugerencias de Scikit Learn.

Se optó por utilizar la métrica accuracy o precisión debido a su facilidad de interpretación.

Tras un extenso proceso de entrenamiento, el modelo que mejor rendimiento tuvo fue el que utiliza el kernel RBF con parámetros `C` igual a 0,1 y `gamma` igual a 1,0. Al analizar los resultados del mismo, consideramos que en general hace un buen trabajo prediciendo las categorías de precios de propiedades nuevas, sin embargo, existe una tendencia a categorizar propiedades que no son de precio alto como si si lo fueran. Esto es fácil de distinguir si analizamos la matriz de confusión del modelo.




Si nos enfocamos particularmente en analizar la matriz de confusión para la categoría de precios altos, se hace más evidente esta tendencia.




La métrica False Positives Rate (FPR) termina teniendo un valor de aproximadamente 0,5824, lo que significa que es más probable que una propiedad que no sea de precio alto sea considerada como tal en vez de ser correctamente categorizada como una propiedad que no es de precio alto.

## Cuadro de Resultados

| Modelo | F1-Test | Precision Test | Recall Test | Accuracy Test |
|---|---|---|---|---|
| Arbol de Decisión | 0,57 | 0,58 | 0,58 | 0,58 |
| Random Forest | 0,27 | 0,40 | 0,35 | 0.50 |
| SVM | 0,61 | 0,63 | 0,62 | 0,62 |

Para el ***árbol de decisión***, la performance respecto al set de entrenamiento fue similar en testing, dando valores en torno a los registrados en el cuadro comparativo.

Los hiperparámetros obtenidos fueron:

| | |
|---|---|
| `min_samples_split` | 15 |
| `min_samples_leaf` | 5 |
| `max_depth` | 17 |
| `criterion` | entropy |
| `ccp_alpha` | 0.00144 |

Para el ***Random Forest***, su performance en comparación con su performance durante el entrenamiento fue muy similar, dando el mismo resultado de f1 score.

Los hiperparámetros obtenidos fueron los siguientes:

| | |
|---|---|
| `n_estimators` | 35 |
| `min_samples_split` | 0.2 |
| `max_features` | 15 |

| | |
|---|---|
| `max_depth` | 2 |
| `criterion` | entropy |

Para la ***SVM*** el rendimiento en el conjunto de entrenamiento dado por la métrica accuracy fue de 0,6504 para el conjunto de entrenamiento y 0,6226 para el de test. Los mejores hiperparámetros obtenidos fueron los siguientes:

| | |
|---|---|
| `C` | 0,1 |
| `kernel` | rbf |
| `gamma` | 1,0 |
| `criterion` | accuracy |

### Elección del modelo

En base a los resultados obtenidos, el modelo que elegimos para predecir el tipo de precio es SVM, ya que resultó tener las mejores métricas, si bien no fue el más rápido de entrenar. Si tuviéramos que basarnos en la velocidad de entrenamiento, se elegiría el árbol de decisión, ya que en promedio tomó 3 minutos para entrenarlo con 30 iteraciones de Random Search, comparado con las casi 10 horas que tomó el entrenamiento de la SVM, utilizando Halving Grid Search. No consideramos que las diferencias de rendimiento entre ambos modelos sean tan significativas, y si se desea utilizar en producción, se puede llegar a valorar más la facilidad de re-entreno que la mínima mejora en la calidad de predicción.

## Regresión

### KNN

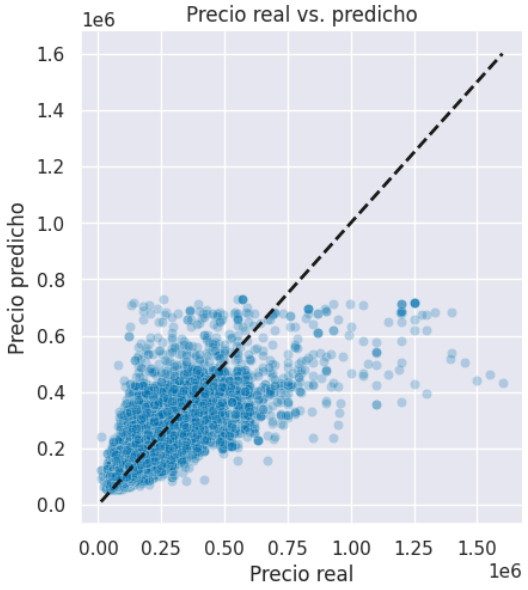
Para determinar los mejores hiperparámetros para el algoritmo KNN para regresión se utilizó Grid Search con 5-fold cross-validation. Para poder entrenar este algoritmo se tuvo que hacer scaling de las variables numéricas, ya que internamente se utiliza una métrica de distancia para hacer las comparaciones entre vecinos, por lo que las

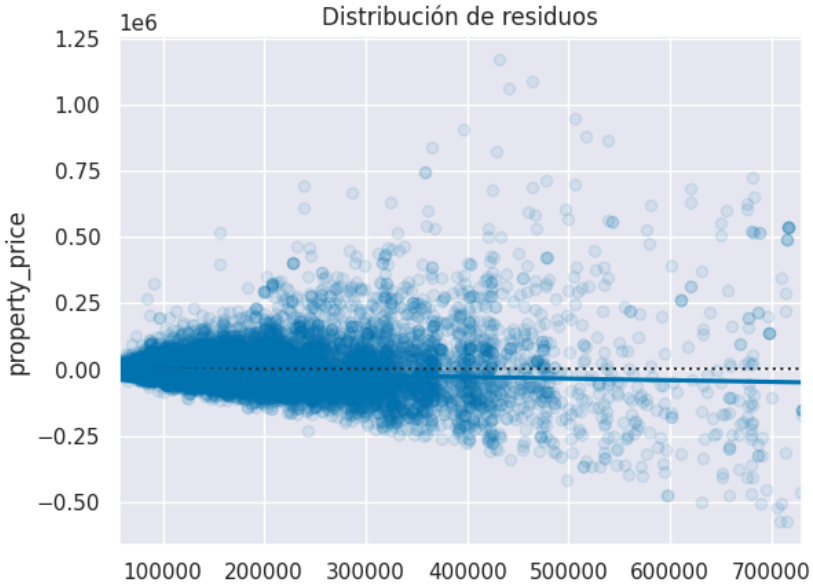
variables que tengan unidades de medida más grandes podrían influir en este cálculo de distancia cuando se comparan con los otros valores numéricos con otras unidades de medida.

Los hiperparámetros probados fueron la cantidad de vecinos utilizados para las comparaciones, mediante el parámetro `n_neighbors`; y también la métrica de distancia utilizada, mediante el parámetro `metric`. Para el primero de estos hiperparámetros se probaron valores cercanos al rango recomendado en esta publicación, mientras que para el parámetro de distancia se probaron tres distancias diferentes: minkowski, euclidean y manhattan, que fueron algunas de las mencionadas durante las clases.

Al final del proceso de entrenamiento los mejores resultados se obtuvieron con el modelo que compara con 238 vecinos y utiliza la distancia euclidiana. Se utilizó la métrica Root Mean Squared Error para la evaluación de los mismos.

Analizamos los resultados de manera gráfica mediante un gráfico de precios reales contra precios predichos y un gráfico de residuos para ver la distribución de los mismos.






A través de los mismos se puede notar una aparente cota superior en los precios de las propiedades predichas, indicando que este modelo es más impreciso para propiedades con precios más altos. De hecho, con el gráfico de residuos podemos confirmar que a medida incrementan los precios de las propiedades reales, la distribución de los residuos tiene mayor variación, indicando mayor imprecisión.

## XGBoost

Para entrenar el modelo de XGBoost usamos Stratified K-Folds Cross Validation para la búsqueda de hiperparámetros, con 5 folds.

También se optó por realizar One-Hot encoding sobre las variables `neighbourhood`, `property_type` y, finalmente a modo de prueba, sobre `tipo_precio`. Esta última no se tuvo en cuenta ya que es una variable que originalmente no formaba parte del dataset, pero luego la consideramos a modo de ver los efectos sobre la performance del modelo si se dispone de ella.

Usamos Stratified K-Folds en lugar de solo K-Folds porque hay una distribución dispareja en las categorías del target, por lo que buscamos generar un conjunto que esté balanceado para entrenar el modelo en cada fold.

Los hiperparámetros a optimizar fueron, para el caso de booster lineal:

- `objective`: Indica el objetivo de aprendizaje de la regresión. Puede ser linear, error al cuadrado o error absoluto, pero existen más.
- `eval_metric`: Métrica de evaluación para los datos de validación.
- `learning_rate`: Paso utilizado para la reducción de los pesos. A menor valor, más conservativo será el modelo. Se aplica luego de cada paso de boost y previene overfitting. Rango: [0, 1]
- `lambda`: parámetro de regularización L1.
- `alpha`: parámetro de regularización L2.
- `gamma`: parametro de poda.

Mientras que en el caso de un ***booster árbol*** se agregó un hiperparámetro más a los anteriores, ya que este es exclusivo para este tipo de booster:

- `max_depth`: Mayor profundidad de un árbol.

Para buscar los hiperparámetros utilizamos la métrica de $R^2$ (valor entre 0 y 1), ya que da un porcentaje de cuán bien el modelo fue capaz de aprender las relaciones entre las variables para predecir la variable de salida. Luego tenemos también las métricas RMSE o raíz cuadrada del error cuadrático medio, y MSE o error cuadrático medio.

Al evaluar la performance sobre el conjunto de evaluación, en general, tanto RMSE y MSE como $R^2$ se mantuvieron similares a la performance en entrenamiento para el booster lineal, llegando ésta última a un valor de 0,72 en evaluación luego de 50 iteraciones de random search, mientras que en entrenamiento alcanzó 0,73.
No obstante al utilizar un booster árbol la performance (respecto de la métrica $R^2$) en entrenamiento resultó casi perfecta, de 0,99, mientras que en evaluación empeoró respecto de entrenamiento, alcanzando 0,87, lo cual es una diferencia de 0,12 con respecto a entrenamiento y si bien es algo significativa, no resulta tan lejana, lo cual nos lleva a pensar que el modelo tiene algo de overfitting pero no es excesivo.

Por último, pudimos ver que al incluir la variable `tipo_precio` en el set de entrenamiento, se obtuvo una performance con 10 iteraciones mejor que antes, llegando a tener un score $R^2$ de 0,70 en ambas instancias en el caso de booster lineal, y en el caso de booster árbol, un score de 0,96 en entrenamiento y de 0,92 en evaluación, mejorando así ésta métrica en evaluación pero empeorando en

entrenamiento ligeramente. En ambos casos, todas las métricas se vieron afectadas positivamente sobre los conjuntos de evaluación, teniendo así mejores resultados en las predicciones del modelo.

## Gradient Boost

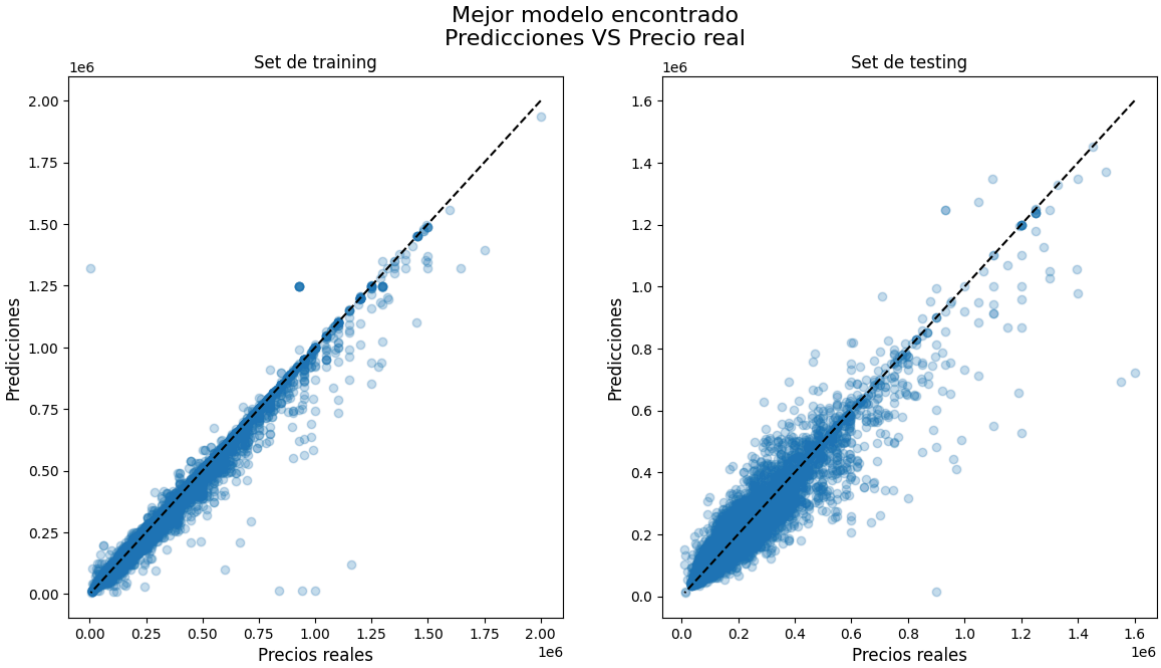
Para entrenar nuestro modelo de Gradient Boost utilizamos K-Folds Cross Validation con 5 folds, usando la métrica de $R^2$ para buscar los hiperparámetros.

Los hiperparámetros que intentamos optimizar fueron:

- `n_estimators`: Cantidad de árboles máximos que pueden estar presentes en el modelo.
- `max_depth`: Máximo nivel de profundidad que puede tener cada árbol.
- `learning_rate`: Valor utilizado para disminuir qué tanto contribuye cada árbol al resultado final.
- `loss`: Función de pérdida a optimizar. Vamos a probar con el error cuadrático y el error absoluto.

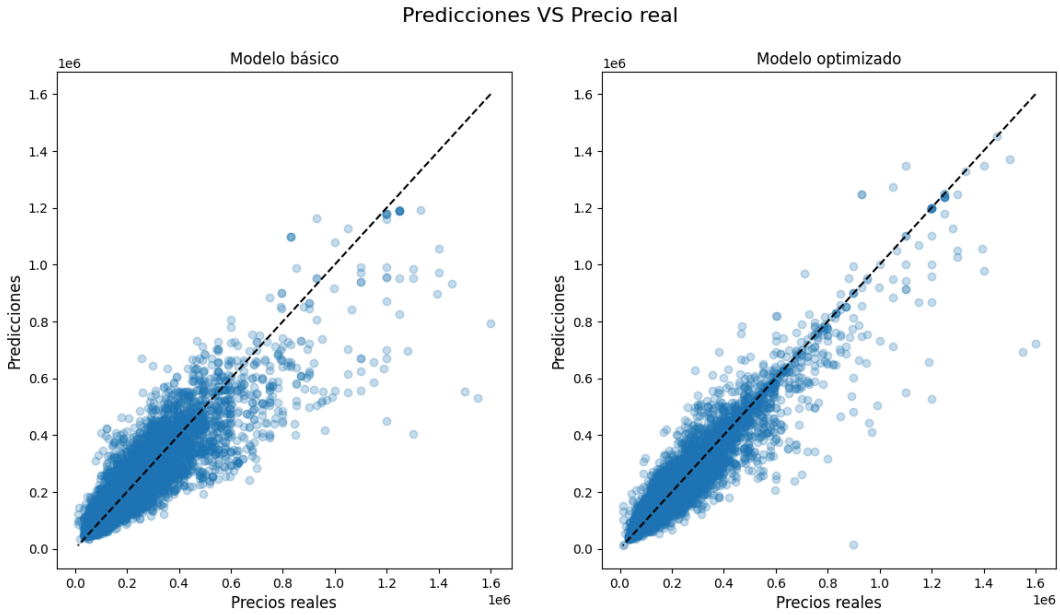
Consideramos utilizar `HalvingSearch` y `GridSearch` para optimizar hiperparametros pero resultaron muy lentos para esta tarea, por lo cual finalmente se utilizó `RandomizedSearch` con una semilla fija.

El modelo obtuvo un buen rendimiento en entrenamiento en el que fue casi perfecto y en el de test resultó también bastante bueno, con resultados de $R^2$ de 0,99 y 0,89 respectivamente. Esos valores para $R^2$ se pueden ver claramente en un gráfico de comparación de predicciones con los precios reales.




Si las predicciones fuesen perfectas, veríamos cómo se alinean perfectamente con la línea punteada trazada en ambos gráficos. Con el set de training, vemos que los datos se alinean bastante mejor que con el de testing.

Para comparar las mejoras de nuestro modelo luego de optimizar sus hiperparámetros, podemos comparar estos gráficos con los del primer modelo que entrenamos.



Se ve una mejora en el segundo gráfico en tanto los valores están menos dispersos en general. Esto también se ve con la métrica $R^2$, que en el primero es de 0.8008 y en el segundo es de 0.8973.

## Cuadro de Resultados

| Modelo | MSE | RMSE | $R^2$ |
|---|---|---|---|
| KNN | 6.857.535.904,13 | 82.810,24 | 0,61 |
| XGBoost | 2.238.680.656,25 | 47.314,70 | 0,87 |
| Gradient Boost | 1.883.568.568,26 | 43.400,10 | 0,89 |

Para ***KNN*** el comportamiento del modelo en entrenamiento y test es muy similar, sin embargo, en ambas ocasiones, los resultados fueron bajos, siendo los valores de la métrica $R^2$ 0,62 para el conjunto de entrenamiento y 0,61 para el de test. Los parámetros del mejor modelo fueron los siguientes:

| | |
|---|---|
| `n_neighbors` | 238 |
| `metric` | euclidean |

Para ***XGBoost***, el modelo elegido resultó performar ligeramente peor en evaluación con respecto al set de entrenamiento, en el cual alcanzó un score de $R^2$ casi perfecto de 0,99. Consta de los siguientes hiperparámetros:

| | |
|---|---|
| `objective` | reg:squarederror |
| `booster` | gbtree |
| `learning_rate` | 0,462 |
| `lambda` | 0,857 |

| | |
|---|---|
| `alpha` | 0,796 |
| `gamma` | 39,799 |
| `eval_metric` | rmse |
| `max_depth` | 12 |

Para ***Gradient Boost*** el modelo obtuvo un rendimiento similar al de XGBoost, en el que se exhibió un muy buen nivel de predicciones en el conjunto de entrenamiento y también en el de test, con resultados de $R^2$ de 0,99 y 0,89 respectivamente.

| | |
|---|---|
| `n_estimators` | 41 |
| `max_depth` | 32 |
| `learning_rate` | 0.2 |
| `loss` | absolute_error |

### Elección del modelo

En base a las métricas del rendimiento entre los tres modelos elegimos Gradient Boost, ya que es el que tuvo mejores resultados, sin embargo, las diferencias con respecto a XGBoost son pequeñas. A diferencia de lo que sucedía entre los modelos de clasificación, los tiempos de entrenamiento entre los modelos también fueron similares.

# Conclusiones Finales

Los datos con los que trabajamos el problema se alinean con lo que originalmente pensábamos sobre las tendencias del mercado inmobiliario en la Ciudad de Buenos Aires. Por ejemplo, que las propiedades más caras efectivamente están en los barrios popularmente categorizados como los más costosos de la ciudad, como Puerto Madero, Palermo Chico o Belgrano; o también que generalmente a mayor área tenga una propiedad, mayor precio va a tener la misma. Sin embargo, la gran cantidad de outliers y datos nulos con los que contaba la ya reducida porción del dataset que nos

interesaba nos hace ser un poco escépticos con los resultados de los modelos entrenados.

Producto el análisis de los resultados del modelo de regresión con mejor rendimiento, descubrimos que incluir información categórica sobre el tipo de precio de una propiedad ayuda significativamente a predecir el precio de la misma, algo que también sentimos que suelen se uno de los filtros principales que se nos vienen a la cabeza cuando pensamos en buscar una nueva propiedad.

Por otra parte, respecto al mejor modelo de clasificación, los resultados obtenidos no son tan fiables en este modelo por sobre los demás. Conocemos que el modelo tiene una tendencia a considerar propiedades como propiedades de precio alto cuando en realidad no lo son, por lo que hay que tener esta consideración en cuenta si se llega a utilizar este modelo.

Desde el punto de vista de la implementación, nos sorprendieron cosas como los elevados tiempos de entrenamiento y cómo estos fluctuaban a medida que íbamos cambiando los parámetros que se probaban. Un ejemplo de esto es el cambio en las cantidades de iteraciones de los algoritmos de Random Search que cambiaban bastante al utilizar un `LabelEncoder` en lugar de One-Hot Encoding.

Finalmente, cabe destacar que durante el proceso de entrenamiento de los modelos de clasificación, también nos llamó la atención el hecho de que el modelo de árbol particular tuviera mejor rendimiento que el random forest, cuando precisamente este segundo algoritmo fue creado con la intención de mejorar el usualmente bajo rendimiento de los árboles.

Entre los aspectos a mejorar, nos hubiese gustado hacer un análisis con más profundidad de los resultados de los modelos. Quizá por falta de experiencia no pudimos interpretarlos mucho más allá de sus métricas o pequeñas representaciones visuales que nos indican algo sus resultados, como por ejemplo, la tendencia de los resultados de regresión con KNN que tendía a acotar los precios posibles a medida se ascendía en el rango de precios reales. Ante dificultades como la de interpretar visualmente los márgenes de clasificación de la Support Vector Machine, debido a la alta dimensionalidad del problema, por ejemplo, nos hubiese gustado conocer más alternativas para tener una mejor comprensión del comportamiento del modelo candidato, aunque comprendemos que hasta cierto punto la complejidad de estos modelos puede exceder los límites de lo que podemos comprender.

También nos hubiese gustado hacer una comparación visual entre los modelos de clasificación, utilizando los gráficos ROC, por ejemplo. Sin embargo, por temas de tiempo, no pudimos re-entrenar los modelos de tal forma que sus resultados pudieran ser utilizados para generar estos gráficos.

Finalmente, aprendimos que el desarrollo de inteligencia artificial es un proceso en el que cada forma de experimentación puede culminar en resultados completamente distintos, dos modelos diferentes que resuelven el mismo problema con similar efectividad, o también que a veces las decisiones que no nos parecían intuitivas de antemano, terminan arrojando los mejores resultados, algo que solo podemos descubrir mediante la experimentación, la iteración sobre resultados pasados, y la puesta en conjunto de diferentes puntos de vista para tener una mejor visión general del problema y de cómo encararlo.

# Tiempo dedicado

| Integrante | Tarea | Prom. Hs Semana |
|---|---|---|
| Carlos Castillo | Entrenamiento SVM.<br>Entrenamiento KNN.<br>Corrección del proceso de tratamiento de outliers multivariados.<br>Análisis de distribuciones reparadas.<br>Redacción de informe. | 10 |
| Juan Pablo Destefanis | Correción del ánalisis de clustering con KMeans.<br>Entrenamiento RandomForest.<br>Entrenamiento Gradient Boost.<br>Corrección del análisis de los resultados de KMeans.<br>Redacción de informe. | 7 |

| | | |
|---|---|---|
| Celeste Gómez | Entrenamiento XGBoost.<br>Entrenamiento Árbol.<br>Análisis de distribuciones reparadas.<br>Corrección del proceso de tratamiento de outliers multivariados.<br>Redacción de informe. | 10 |