Trabajo Práctico N°1 - Grupo XX

Clasificación

La alternativa elegida para construir la variable *tipo_precio* será la de tener en cuenta el tipo de propiedad para entender mejor la distribución de precios relativos. Hicimos un gráfico de la distribución de la variable en el mapa y notamos que, objetivamente, lo obtenido es bastante fiel a la realidad ya que los precios más elevados para las propiedades se encuentran desde el norte hacia el noreste en toda la periferia, algunos puntos del noroeste y cerca del centro geográfico de la ciudad.
En cuanto a la comparación con el gráfico obtenido a través de K-means, no podemos sacar una conclusión ya que la variable *tipo_precio* depende pura y exclusivamente de *pxm2*, haciendo que la clasificación esté "sesgada". A la hora de aplicar K-means se trabajó con más variables en vez de la relación directa entre superficie total y precio.

a. Construcción del modelo

Arbol de Decision

- ¿Optimizaron hiperparámetros? ¿Cuáles?
- ¿Utilizaron K-fold Cross Validation?¿Cuántos folds utilizaron?
- ¿Qué métrica utilizaron para buscar los hiperparámetros?
- Añadir imagen del árbol generado e incluir descripciones que consideren adecuadas para entender el mismo. Si es muy extenso mostrar una porción representativa.

Para empezar, hicimos una primera prueba con todos los parámetros por defecto para evaluar el funcionamiento general, ver los primeros puntajes y la primera matriz de confusión. Los resultados que obtuvimos fueron muy buenos, considerando puntajes de precisión, f1 y recall por encima de 0,7 en promedio. Cabe mencionar que los hiperparámetros que el árbol base tiene por defecto son, entre otros, "infinita" profundidad y se basa en el coeficiente de Gini para hacer las decisiones.

Como siguiente paso en la exploración y búsqueda de un mejor modelo, haremos Randomized Search para encontrar la mejor combinación de hiperparámetros. Vamos a probar distintas combinaciones alterando las siguientes características:

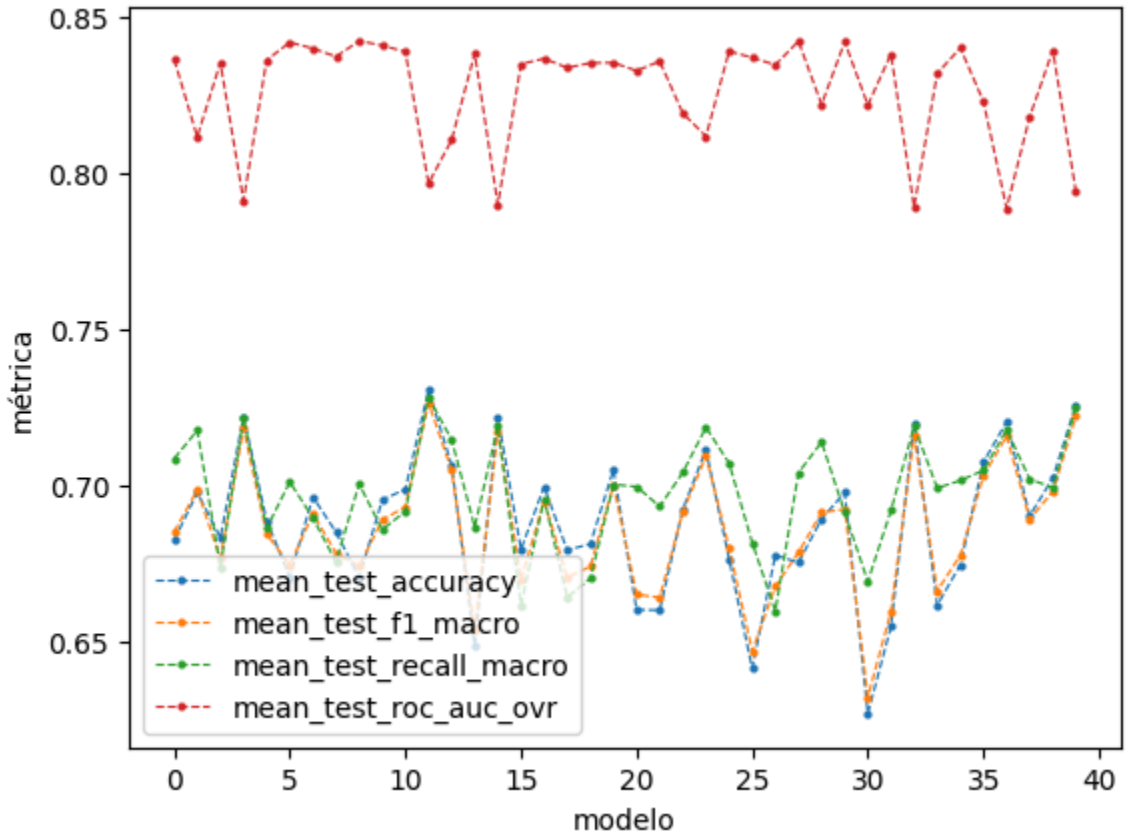
- criterion: la función utilizada para medir la calidad de cada división
- min_samples_leaf: la cantidad mínima de muestras que debe tener una hoja

- splitter: el criterio adoptado para dividir los nodos

- class_weight: balancea, o no, las clases en el conjunto de datos.
- Min_samples_split: la cantidad mínima de muestras requeridas para dividir a un nodo.
- ccp_alpha: ayuda a controlar el over-fitting y mejora la capacidad de generalización del modelo.

La cantidad de árboles que vamos a tomar sobre todas las combinaciones posibles es 40. Luego, aplicaremos k-fold, con 10 folds, para evaluar el rendimiento de los árboles y nos quedaremos con el de mejor rendimiento.

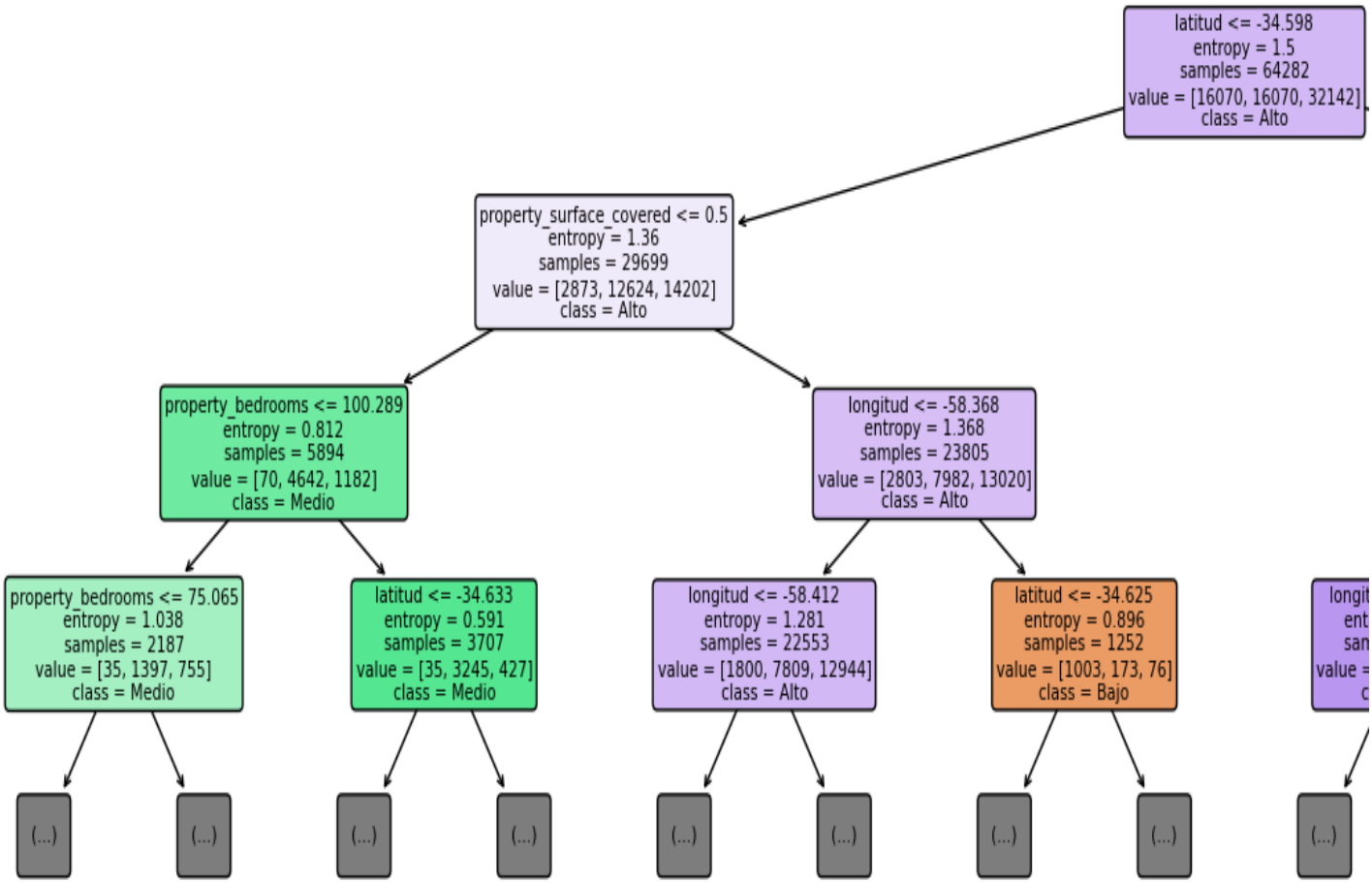
La intención de este gráfico es mostrar cómo va variando la performance según las distintas combinaciones de hiperparámetros.



La conclusión a la que llegamos es que no hay mucha mejora por hacer, ya que los resultados obtenidos con el árbol con parámetros default fueron muy buenos.

Por último, mostraremos de manera gráfica el árbol que mejor resultados tuvo:

El árbol obtenido es muy extenso para visualizarlo entero, pero desde su raíz ya podemos ver cómo se van tomando las decisiones.



Para su interpretación debemos situarnos en la raíz. El gráfico mostrado es el sub-árbol izquierdo de nuestro árbol principal. La primera línea de los nodos es la condición de división. La segunda línea nos dice la entropía del subgrupo obtenido a partir de la división. Cuanto más baja la entropía, más pura es la clase.

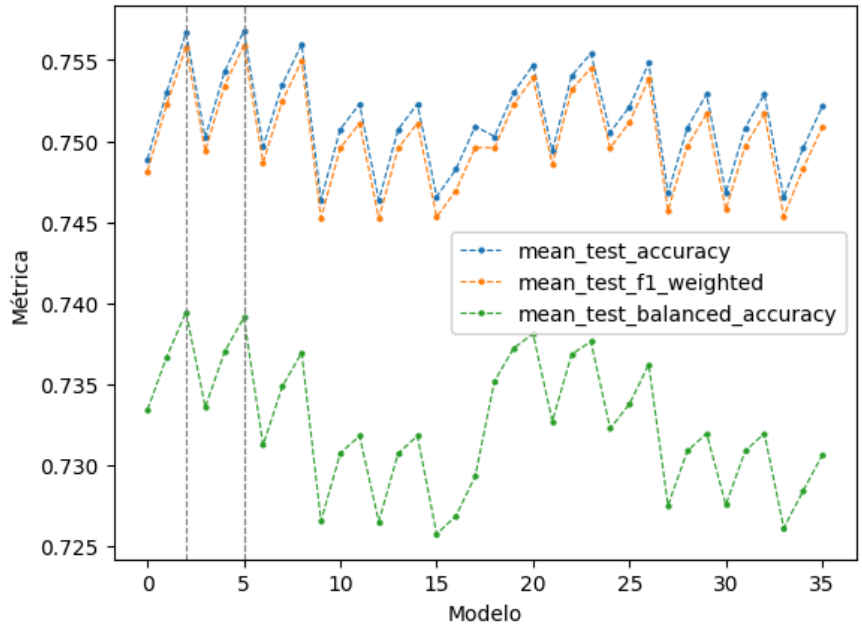
Random Forest

- ¿Optimizaron hiperparámetros? ¿Cuáles?
- ¿Utilizaron K-fold Cross Validation?¿Cuántos folds utilizaron?
- ¿Qué métrica utilizaron para buscar los hiperparámetros?
- Mostrar la conformación final de uno de los árboles generados. Si es muy extenso mostrar una porción representativa y explicar las primeras reglas.

En Random Forest se optimizaron los siguientes hiperparámetros mediante Grid Search Cross Validation:

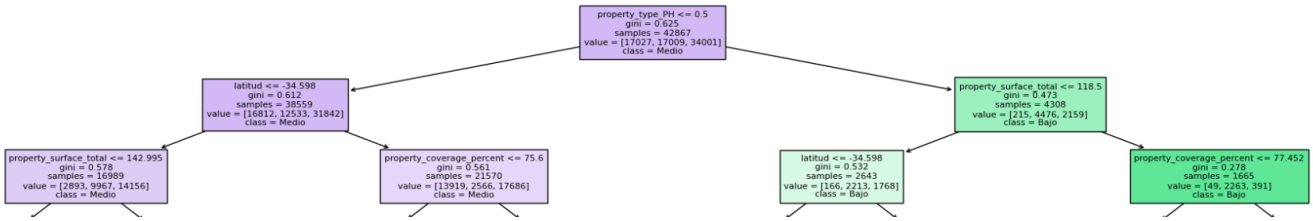
- criterion: entre Impureza de Gini y Ganancia de la Información
- min_samples_leaf, min_samples_split: Se probaron distintos criterios de división de nodos
- n_estimators: se probaron distintas cantidades de árboles

Para buscarlos, tuvimos en cuenta Accuracy, F1 ponderado (una versión de F1 para predicciones de múltiples clases desbalanceadas) y Balanced Accuracy (una versión de Accuracy para predicciones de múltiples clases desbalanceadas).



Sin embargo, entre las 3 priorizamos Accuracy sobre los demás, ya que por el hecho de que es la única métrica no ponderada, la consideramos la menos sesgada.

Esta optimización dió como resultado un modelo con árboles que priorizaron a Superficie Total/Cubierta, Porcentaje de Superficie Cubierta y Latitud/Longitud como las características más importantes a la hora de clasificar. Sin embargo, no todos los árboles priorizaron solamente esas características:



Podemos ver que, a diferencia de los atributos importantes que destacamos antes, este árbol en particular decidió tener como raíz al atributo categórico property_type_PH, partiendo el conjunto en 2 ramas: una con Propiedades Horizontales, y la otra con Casas y Departamentos.

Modelo a Elección

- ¿Optimizaron hiperparámetros? ¿Cuáles?
- ¿Utilizaron K-fold Cross Validation?¿Cuántos folds utilizaron?
- ¿Qué métrica utilizaron para buscar los hiperparámetros?
- Mostrar la conformación final de uno de los árboles generados. Si es muy extenso mostrar una porción representativa y explicar las primeras reglas.

Nota: Para cada modelo mencionar si realizaron nuevas transformaciones sobre los datos (encoding, normalización, etc)

b. Cuadro de Resultados

Modelo	F1-Test (Promedio Ponderado)	Precision Test (Promedio Ponderado)	Recall Test (Promedio Ponderado)	Accuracy Test	
Arbol de Decision	0.74	0.74	0.74	0.74	
Random Forest	0.767	0.769	0.767	0.767	
Gradient	0.684	0.711	0.669	0.695	

Boosting					

En Random Forest, Accuracy y el F1 Weighted durante el Testing fueron ligeramente mejores (Ambos alrededor del 76.7%) que los de Training con Cross Validation (Ambos alrededor del 75.5%).

En Gradient Boosting, Accuracy y el F1 Weighted durante el Testing fueron bastante similares a los de Training con Cross Validation (Todas las métricas alrededor del 69.5%).

Cada modelo

Arbol de decision: {'max_depth': 7, 'criterion': 'entropy', 'ccp_alpha': 0.02}
Random Forest: {'criterion': 'gini', 'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 50}
Gradient Boost: {'n_estimators'=125, 'learning_rate': 0.9}

c. Elección del modelo

Nosotros elegiríamos Random Forest, ya que no solo tiene las mejores métricas, sino que también podemos extraer de él los atributos más importantes del dataset, lo cual puede ayudarnos en otros modelos predictivos tanto de regresión como de clasificación

Regresión

Mencionar en cada caso si realizaron nuevas transformaciones sobre los datos (encoding, normalización, etc) completando los ítems a y b:

a. Construcción del modelo

KNN

- Se uso 3 folds, debido que se probo con mas, y el tiempo de de ejecución demoraba muchísimo más, para que termine dando un resultado muy similar, y se prefirio usar 3 folds para no aumentar la complejidad computacional

- Se utilizaron las distancias 'euclidean' y ''manhattan'.
  - Euclidiana: Es la raíz cuadrada de la suma de los cuadrados de las diferencias entre las coordenadas de dos puntos en un espacio n-dimensional.
  - Manhattan: Se calcula como la suma de las diferencias absolutas entre las coordenadas de dos puntos en un espacio n-dimensional.
  Los algoritmos para ver los vecinos mas cercanos que use fueron 'ball_tree' y 'kd_tree'.
  - Ball Tree: organiza los datos en una estructura de árbol de tipo esférico (un árbol de bolas) para facilitar la búsqueda de vecinos más cercanos.
  - KD Tree: organiza los datos en una estructura de árbol binario multidimensional.
  Divide el espacio de manera recursiva en hiperplanos ortogonales (planos divididos) para organizar los datos y facilitar la búsqueda de vecinos más cercanos.
- Se utilizaron los pesos uniforme y de distancia.
  - Uniforme, no importa que tan lejo este el vecino del centroide, pesa lo mismo que el que está más cerca
- Distancia: Dependiendo la distancia al centroide, el vecino pesa mas o menos, es decir y un vecino está mas cerca pesa mas que uno que esta mas lejo

- Evaluar la performance del modelo en el conjunto de evaluación, explicar todas las métricas. Comparar con la performance de entrenamiento.

- Mejores hiperparámetros: {'algorithm': 'ball_tree', 'metric': 'manhattan', 'n_neighbors': 19, 'weights': 'uniform'}, se puede ver ademas de que la mejor métrica se da con 19 vecinos

- Mejor métrica: 0.6932258659259926: Es un coeficiente de determinacion razonablemente bueno y sugiere que el modelo tiene una capacidad predictiva medianamente alta

- Root Mean Squared Error (RMSE): 106906.95759304542: Como me da un número tan alto, el modelo no es tan preciso

- Mean Absolute Error (MAE): 55727.928660385536: Lo mismo con RSME, el modelo no están precisio

Conjunto de Entrenamiento:

Mean Squared Error (MSE): 10831863242.774

Root Mean Squared Error (RMSE): 104076.237

Mean Absolute Error (MAE): 53836.115

Conjunto de Evaluación:

Mean Squared Error (MSE): 11429097581.801

Root Mean Squared Error (RMSE): 106906.957

Mean Absolute Error (MAE): 55727.928

Se puede ver como el conjunto de entrenamiento de un poco mejor que el de Evaluación

XGBoost

- Probamos con RandomizedSearchCV y GridSearchCV en ambos usamos un total de 2 folds.
- Para buscar los hiperparámestro pasamos los siguientes {'n_estimators': [30,40], 'max_depth': [9, 12, 20], 'learning_rate': [0.1, 0.5], 'subsample': [0.9, 1.0], 'colsample_bynode': [0.9, 1.0], 'reg_alpha': [0.1, 0.5], 'reg_lambda': [0.1, 0.5], 'min_child_weight': [1, 3, 5], 'gamma': [0.1, 0.5], 'objective': ['reg:squarederror', 'reg:squaredlogerror']} para que GridSearchCV busque la mejor combinación.
- Tambien se uso RandomizedSearchCV para con este modelo usamos las siguientes hiperparámetros { 'n_estimators': [10, 20, 30], 'max_depth': [3, 6, 9], 'learning_rate': [0.01, 0.1, 0.5], 'subsample': [0.8, 0.9, 1.0], 'colsample_bynode': [0.8, 0.9, 1.0], 'reg_alpha': [0, 0.1, 0.5], 'reg_lambda': [0, 0.1, 0.5], 'min_child_weight': [1, 3, 5] 'gamma': [0, 0.1, 0.5], 'objective': ['reg:squarederror', 'reg:squaredlogerror'] }
- Como se mencionó anteriormente se usa GridSearchCV y RandomizedSearchCV
    - La mejor combinación para RandomizedSearchCV {'subsample': 0.8, 'reg_lambda': 0, 'reg_alpha': 0.5, 'objective': 'reg:squarederror', 'n_estimators': 80, 'min_child_weight': 3, 'max_depth': 20, 'learning_rate': 0.5, 'gamma': 0.5, 'colsample_bynode': 1.0} y las métricas son las siguientes:
        - Error cuadrático medio (MSE): 11144455273.136312
        - Error absoluto medio (MAE): 62533.27721657511
        - Raíz del error cuadrático medio (RMSE): 105567.30210219599
    - Y para GridSearchCV las métricas son las siguientes:
        - Error cuadrático medio (MSE) - Entrenamiento: 11319822129.318222
        - Error cuadrático medio (MSE) - Prueba: 11403405875.356577
        - Error absoluto medio (MAE) - Entrenamiento: 62968.63964632755
        - Error absoluto medio (MAE) - Prueba: 63046.104818521715

    - Raíz del error cuadrático medio (RMSE) - Entrenamiento: 106394.65272897047
    - Raíz del error cuadrático medio (RMSE) - Prueba: 106786.73080189587

- Entrenamos un par de modelos más las cuales están en la notebook, es tos las mejores métricas son:
    - El mejor modelo en términos de Error Absoluto Medio (MAE) es: Modelo Dos
    - Error absoluto medio (MAE) del mejor modelo: 62533.27721657511
    - El modelo dos en nuestro caso es el modelo encontrado con RandomizedSearchCV

- Con dicho modelo se hizo predicción se corre en test y las métricas son las siguientes:
    - Error cuadrático medio (MSE) - Test: 76425807736.26505
    - Error absoluto medio (MAE) - Test: 87484.94506065312
    - Raíz del error cuadrático medio (RMSE) - Test: 276452.17983634176

<u>Modelo a elección</u>

- Como modelo de elección escogimos SVM para buscar hiperparámetros usamos GridSearchCV con folds = 2.
- Para buscar los hiperparámetros se usa GridSearchCV y se le pasó los siguientes hiperparámetros param_grid = { 'kernel': ['rbf'], 'C': [1, 10], 'epsilon': [0.1, 0.2] }
- Los hiperparámetros que mejor resultado dieron según GridSearchCV son los siguientes {'C': 10, 'epsilon': 0.1, 'kernel': 'rbf'}
- Las métricas son las siguientes:
    - Error cuadrático medio (MSE) - Entrenamiento: 35737017864.7711
    - Error cuadrático medio (MSE) - Prueba: 35319047272.65273
    - Error absoluto medio (MAE) - Entrenamiento: 85086.15818877425
    - Error absoluto medio (MAE) - Prueba: 84344.6167079897
    - Raíz del error cuadrático medio (RMSE) - Entrenamiento: 189042.37055425195
    - Raíz del error cuadrático medio (RMSE) - Prueba: 187933.6246461839

- Y en test nos dió los siguientes resultados:
  - Error cuadrático medio (MSE) - Test: 35319047272.65273
  - Error absoluto medio (MAE) - Test: 84344.6167079897
  - Raíz del error cuadrático medio (RMSE) - Test: 187933.6246461839

b. Cuadro de Resultados

Los distintos modelos de regresión nos dieron los siguientes resultados en el conjunto de TEST:

Modelo	MSE	RMSE	MAE
KNN	11429097581.801	106906.957	55727.928
XGBoost	76425807736.26505	87484.94506065312	276452.17983634176
SVM	35319047272.65273	84344.6167079897	187933.6246461839

c. Elección del modelo

Nos quedamos con XGBoost pensamos que es más robusto.

Conclusiones Finales

Pudimos ver que creando nuevos atributos aportó significativamente en la performance de los modelos., por lo que nos hubiese gustado invertir más tiempo creándolo

Tiempo dedicado

Integrante	Tarea	Prom. Hs Semana
Gonzalo Olmos	Clasificación Arbol de Desición	6

Elvis Claros	Split Train-Test XGBoost SVM	6
Ramiro Gareis	Knn	6
Matías Venglar	Clasificación Random Forest Gradient Boost	6