



Maestría en Inteligencia Artificial Aplicada

Materia: Proyecto Integrador

Profesor Titular: Dra. Grettel Barceló Alonso / Dr. Luis Eduardo Falcón Morales

Asesor de Proyecto: Dr. Carlos Alberto Villaseñor Padilla

Avance 4. Modelos Alternativos

Equipo 10

David García Robles A01152606

David Nava Jiménez A01168501

José Antonio Hernández Hernández A01381334

Fecha: 25 de Mayo de 2025

Optimización de ventas en Nacional Monte de Piedad

Avance 4. Modelos Alternativos

Contenido

Avance 4. Modelos Alternativos.....	1
1.1 Relojes - Separación del Dataframe.....	4
1.2 Partición de datos y Evaluación del Desempeño del Modelo con Métricas Predictivas.....	4
1.3 Comparación de Modelos de Machine Learning y Evaluación Inicial de Desempeño.....	5
1.4 Alhajas - Separación del Dataframe.....	8
1.5 Partición de datos y Evaluación del Desempeño del Modelo con Métricas Predictivas.....	9
1.6 Comparación de Modelos de Machine Learning y Evaluación Inicial de Desempeño.....	10
1.7 Ajuste de Hiperparámetros para XGBoost mediante Validación Cruzada.....	13
1.8 Importancia de Variables según el Modelo XGBoost Final.....	14
1.9 Desempeño del Modelo Final sobre el Conjunto de Prueba.....	15

Introducción

A lo largo de esta semana tuvimos diversas reuniones con Nacional Monte de Piedad, así como con nuestro Tutor, el Dr. Carlos Alberto Villaseñor para comentar el estatus de nuestros avances. Nuestro primer baseline que tuvo lugar en la entrega de la semana 5 nos percatamos de que nuestras métricas de desempeño estaban muy por debajo de lo esperado. Nuestra regresión lineal alcanzaba un R^2 máximo de 0.2 (rango 0—>1) donde 1 es una predicción perfecta, así mismo estamos utilizando el RMSE que mide el error cuadrático medio, en término de las unidades que tiene nuestra variable objetivo, que en este caso son días. (DIAS_ALMONEDA).

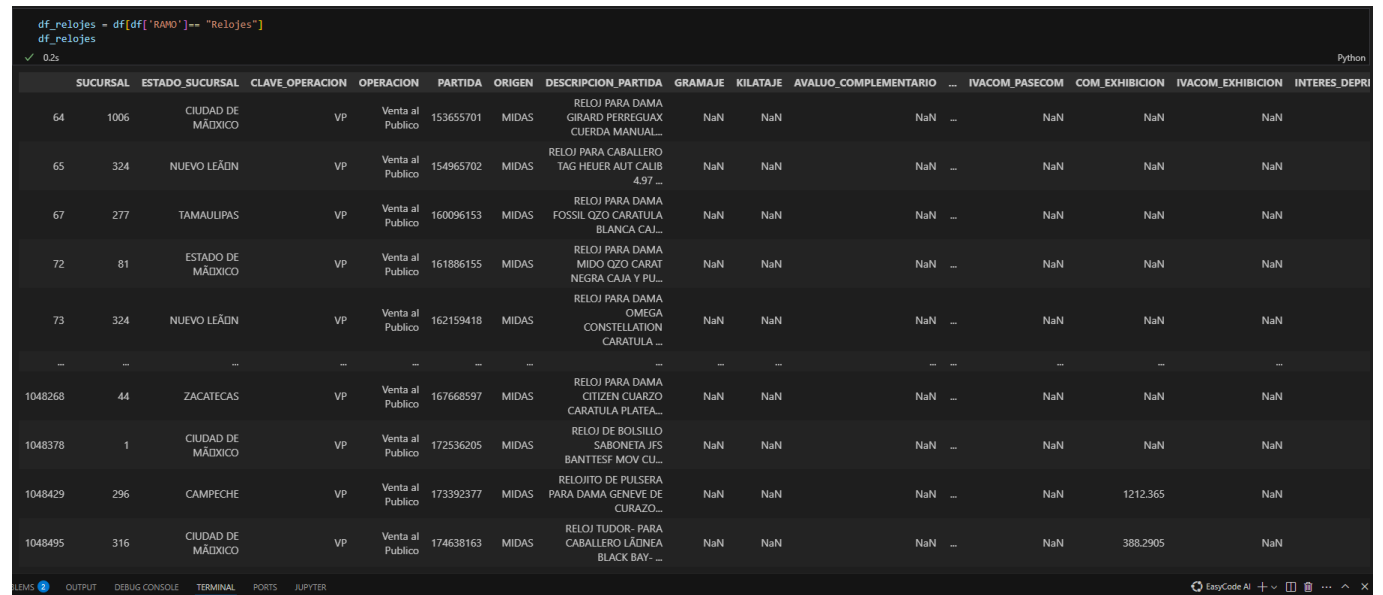
Encontramos que XGboost mejoraba dichas métricas a 0.45 y un error medio (RMSE) de 100 días, siendo todavía insuficiente para lo esperado. Discutiendo con el Dr. Carlos, nos comentó que el dataset tenía mucho ruido, por lo que al modelo le iba a ser insuficiente su capacidad para detectar los patrones y distribuciones que se esperan. Nos sugirió realizar un split entre los dos grandes grupos del que se conforma el dataset original (Alhajas y relojes). Esto se discutió con NMP y nos confirmaron que efectivamente los datos que nos comparten el proceso de su captura es manual y actualmente no existe un procedimiento estandarizado para la captura de dichos datos, ya que el core business de NMP está en otorgar un crédito prendario, más no en la comercialización de los artículos que se quedan para venta. Es por esa razón que les pareció conveniente hacer el análisis por separado de alhajas y relojes.

En el presente documento vamos a plantear las alternativas que se trabajaron para esta semana.

Durante esta etapa, nos tuvimos que regresar a la etapa de exploración de datos e ingeniería de características con base a los elementos que por parte del negocio tienen una relación con cada grupo (Alhajas y Relojes) (en el documento se describen únicamente los procesos adicionales que se realizaron durante esta semana). Posteriormente se llevaron a cabo pruebas con distintos algoritmos de regresión (ElasticNet, Árboles de Decisión, Random Forest y XGBoost), seguidas de un proceso de ajuste de hiperparámetros para optimizar el desempeño del modelo seleccionado. Finalmente, se evaluó el modelo utilizando métricas como el coeficiente de determinación R^2 y la raíz del error cuadrático medio (RMSE), tanto en escala logarítmica como en su forma original, con el objetivo de validar su capacidad predictiva y nivel de generalización en datos nuevos.

1.1 Relojes - Separación del Dataframe

```
df_relojes = df[df['RAMO']== "Relojes"]  
df_relojes
```



SUCURSAL	ESTADO_SUCURSAL	CLAVE_OPERACION	OPERACION	PARTIDA	ORIGEN	DESCRIPCION_PARTIDA	GRAMAJE	KILATAJE	AVALUO_COMPLEMENTARIO	IVACOM_PASECOM	COM_EXHIBICION	IVACOM_EXHIBICION	INTERES_DEPRI
64	1006	CIUDAD DE MÃXICO	VP	Venta al Publico	153655701	MIDAS	RELOJ PARA DAMA GIRARD PERREGUAX CUERDA MANUAL...	NaN	NaN	NaN	NaN	NaN	NaN
65	324	NUEVO LEÃDN	VP	Venta al Publico	154965702	MIDAS	RELOJ PARA CABALLERO TAG HEUER AUT CALIB 4.97 ...	NaN	NaN	NaN	NaN	NaN	NaN
67	277	TAMAULIPAS	VP	Venta al Publico	160096153	MIDAS	RELOJ PARA DAMA FOSSIL OZO CARATULA BLANCA CAJ...	NaN	NaN	NaN	NaN	NaN	NaN
72	81	ESTADO DE MÃXICO	VP	Venta al Publico	161886155	MIDAS	RELOJ PARA DAMA MIDO OZO CARAT NEGRA CAJA Y PU...	NaN	NaN	NaN	NaN	NaN	NaN
73	324	NUEVO LEÃDN	VP	Venta al Publico	162159418	MIDAS	RELOJ PARA DAMA OMEGA CONSTELLATION CARATULA ...	NaN	NaN	NaN	NaN	NaN	NaN
--	--	--	--	--	--	--	--	--	--	--	--	--	--
1048268	44	ZACATECAS	VP	Venta al Publico	167668597	MIDAS	RELOJ PARA DAMA CITIZEN CUARZO CARATULA PLATEA...	NaN	NaN	NaN	NaN	NaN	NaN
1048378	1	CIUDAD DE MÃXICO	VP	Venta al Publico	172536205	MIDAS	RELOJ DE BOLSILLO SABONETIA JFS BANTTESF MOV CU...	NaN	NaN	NaN	NaN	NaN	NaN
1048429	296	CAMPECHE	VP	Venta al Publico	173392377	MIDAS	RELOJITO DE PULSERA PARA DAMA GENEVE DE CURAZO...	NaN	NaN	NaN	1212.365	NaN	NaN
1048495	316	CIUDAD DE MÃXICO	VP	Venta al Publico	174638163	MIDAS	RELOJ TUDOR- PARA CABALLERO LÃDNEA BLACK BAY- ...	NaN	NaN	NaN	388.2905	NaN	NaN

Comentarios: Podemos observar que el dataset de relojes representa un 5% del total del dataset

Comentarios. Es importante mencionar que revisando las columnas del dataframe aplicables a alhajas no existen, son redundantes o no agregan valor al análisis. En esta etapa, se regresó al análisis exploratorio y preprocesamiento para hacer los ajustes debidos, como eliminación de valores atípicos, columnas que no agregan valor y transformaciones correspondientes a variables numéricas y categóricas.

1.2 Partición de datos y Evaluación del Desempeño del Modelo con Métricas Predictivas

A continuación, vamos a realizar una partición 70-15-15 en entrenamiento, validación y prueba respectivamente.

```
Xtrain, xtemp, ytrain, ytemp = train_test_split(X, y, test_size = .30, )  
Xval, Xtest, yval, ytest = train_test_split(xtemp, ytemp, test_size= 0.5, )
```

```
(23747, 64) (23747,)  
(5089, 64) (5089,)  
(5089, 64) (5089,)
```

A continuación vamos a evaluar el modelo con las métricas raíz del error cuadrático medio RMSE, y la métrica R^2

```
lr_model = LinearRegression() # Inicializar un modelo de regresión
lineal
lr_model.fit(Xtrain, ytrain)

y_prediction = lr_model.predict(Xval) # Realizar predicciones sobre el conjunto
de validación
mse = mean_squared_error(yval, y_prediction)
rmse = np.sqrt(mse)
r2 = r2_score(yval, y_prediction)
print(f'MSE: {mse:.4f}')
print(f'RMSE: {rmse:.4f}')
print(f'R2: {r2:.4f}')

print(f" MSE: {mse:.4f}")
print(f" RMSE: {rmse:.4f}")
print(f" R2: {r2:.4f}")
```

MSE: 37775.9678

RMSE: 194.3604

R2: 0.2641

Interpretaciones

RMSE: 194 días. En promedio, el modelo comete un error de 194 días al predecir la cantidad de días que un artículo tarda en llegar a almoneda. Esta métrica es útil y directamente interpretable, ya que está en la misma escala que la variable objetivo (DIAS_ALMONEDA). Por lo que el error es importante y vamos a revisar modelos más complejos.

R2: 0.26 El modelo explica aproximadamente el 26% de la varianza total de la variable DIAS_ALMONEDA. Esto indica que el modelo no tiene un buen poder explicativo.

1.3 Comparación de Modelos de Machine Learning y Evaluación Inicial de Desempeño

Derivado que vamos a utilizar validación-cruzada, se concatenan los conjuntos de entrenamiento y validación para utilizarlo como entrenamiento

```
Xtrainval = pd.concat([Xtrain, Xval], axis=0)
ytrainval = pd.concat([ytrain, yval], axis=0)

print(Xtrainval.shape, ytrainval.shape)
(28836, 64) (28836,)
```

```
def mis_modelos():
    modelos, nombres = list(), list()
```

```

#ElasticNet
modelos.append(ElasticNet(alpha=0.7, l1_ratio=0.5, tol=0.01, max_iter=1500, selection='random' ))
nombres.append('ElasticNet')

#Árbol de Decisiones
modelos.append(DecisionTreeRegressor(max_depth=15, max_features=15, min_samples_split=3, ))
nombres.append('Dtree')

#Random Forest
modelos.append(RandomForestRegressor(n_estimators=700, max_features=25, max_depth=25,
min_samples_split=0.5,max_samples=1.0, ccp_alpha=0.05 ))
nombres.append('RandomF')

#XGBoosting
modelos.append(XGBRegressor(booster='gblinear',n_estimators=100, max_depth=60, learning_rate=0.2,
subsample=1.0,objective ='reg:squarederror'))
nombres.append('XGBoost')

# Red neuronal de Perceptrón Multicapa
modelos.append(MLPRegressor(hidden_layer_sizes=(40,),activation='relu',max_iter=15,alpha=0.1,learning_rate='constant'))
nombres.append('MLP')

return modelos, nombres

#Posteriormente vamos a entrenar cada uno de los modelos y vamos a desplegar las métricas de Train y val

modelos, nombres = mis_modelos()
resultados =list()

cv = RepeatedKFold(n_splits=10, n_repeats=1, )

for modelo, nombre in zip(modelos, nombres):
    pipeline = Pipeline(steps=[('m', modelo)])

scores = cross_validate(pipeline,
                        Xtrainval,
                        ytrainval,
                        scoring={'r2':'r2', 'rmse': 'neg_root_mean_squared_error'},
                        cv=micv,
                        return_train_score=True,
                        n_jobs=1)

    #Se guarda el resultado de cada modelo
    resultados.append(scores)

    #Desplegamos los valores de las métricas para verificar si no hay subentrenamiento o
sobreentrenamiento
    print(f">> {nombre}")
    print(f"\tTrain R2: {np.mean(scores['train_r2']):.3f} ± {np.std(scores['train_r2']):.3f}")
    print(f"\tVal R2: {np.mean(scores['test_r2']):.3f} ± {np.std(scores['test_r2']):.3f}")
    print(f"\tTrain RMSE: {-np.mean(scores['train_rmse']):.3f} ± {np.std(scores['train_rmse']):.3f}")
    print(f"\tVal RMSE: {-np.mean(scores['test_rmse']):.3f} ± {np.std(scores['test_rmse']):.3f}")

```

```

>> ElasticNet
    Train R2: 0.197 ± 0.002
    Val   R2: 0.196 ± 0.013
    Train RMSE: 191.598 ± 1.016
    Val   RMSE: 191.470 ± 9.776

>> Dtree
    Train R2: 0.735 ± 0.024
    Val   R2: 0.359 ± 0.064
    Train RMSE: 109.903 ± 5.251
    Val   RMSE: 170.619 ± 12.554

>> RandomForest
    Train R2: 0.271 ± 0.003
    Val   R2: 0.264 ± 0.014
    Train RMSE: 182.573 ± 0.640
    Val   RMSE: 183.241 ± 6.603

>> XGBoost
    Train R2: 0.248 ± 0.002
    Val   R2: 0.243 ± 0.021
    Train RMSE: 185.351 ± 0.870
    Val   RMSE: 185.798 ± 7.858

>> MLP
    Train R2: 0.248 ± 0.009
    Val   R2: 0.245 ± 0.014
    Train RMSE: 185.446 ± 1.529
    Val   RMSE: 185.473 ± 10.600

```

Interpretaciones

En la comparación de modelos utilizando validación cruzada, se evaluaron seis algoritmos: ElasticNet, Árbol de Decisión, Random Forest, XGBoost, Red neuronal de perceptrón multicapa y Máquina de Vector Soporte.

El modelo ElasticNet mostró un desempeño estable pero bajo con un R2 de entrenamiento de 0.197 y un R2 de validación de 0.196. Esta similitud indica que el modelo no presenta sobreajuste sin embargo, también revela una capacidad predictiva muy baja en comparación con modelos más complejos, ya que sólo explica alrededor del 19% de la variabilidad de la variable objetivo.

El Árbol de Decisión obtuvo un R2 de entrenamiento de 0.73 y un R2 de validación de 0.359. Aunque su rendimiento es inferior al de ElasticNet, la diferencia entre ambos valores sugiere un amplio sobreentrenamiento. Esto indica que el modelo tiende a ajustarse más a los datos de entrenamiento, pero no generaliza tan bien, posiblemente debido a la complejidad de las divisiones del árbol y la ausencia de mecanismos de regularización más avanzados.

Random Forest mostró un rendimiento claramente superior a los modelos anteriores, pero todavía muy por debajo de lo esperado, alcanzando un R2 de entrenamiento de 0.27 y un R2 de validación de 0.26. La diferencia entre ambos valores es mínima, pero no suficiente para considerarlo con buen desempeño al momento de la predicción.

El modelo XGBoost obtuvo un R2 de entrenamiento de 0.248 y un R2 de validación de 0.243. Además, el modelo logra explicar el 24% de la variabilidad en los datos de validación, siendo una baja capacidad explicativa entre los modelos evaluados.

MLP, por su parte, se desempeñó de forma muy cercana a XGBoost, con un R2 de entrenamiento de 0.248 y un R2 de validación de 0.245. Por lo que tiene un desempeño todavía pobre.

Conclusión dataframe Relojes. Como podemos observar los diferentes modelos para predecir la variable DIAS_ALMONEDA del dataset de relojes tiene un rendimiento muy pobre, por lo que se confirma lo que NMP comentó que los datos cuentan con mucho ruido y es muy complicado encontrar relaciones explicar la varianza con la variable objetivo.

1.4 Alhajas - Separación del Dataframe

```
df = df[df["RAMO"] == "Alhajas"].copy() # Filtrar el dataframe para conservar únicamente los registros correspondientes al ramo "Alhajas"
```

	SUCURSAL	ESTADO_SUCURSAL	CLAVE_OPERACION	PARTIDA	GRANJE	KILATAJE	AVALLU_COMPLEMENTARIO	FACTOR_HECHURA	FACTOR	VALOR_MONTE	...	TIPO_PRENDA	FECHA_EMPENO_OK	FECHA_HORA_MOV_OK	FECHA_COMERCIALIZACION_OK	SALDO_INSOLUTO	COP_ALMACENAJE	COMPASE_COMERCIALIZACION	COP_EXHIBICION	imp_minusvalia
0	1005	CIUDAD DE MEXICO	VP	181615421	4	14	0	Abollado Ligero / Funcional	F3	0	...	anillo	NaN	NaN	NaN	1309.42	NaN	NaN	NaN	NaN
1	1005	CIUDAD DE MEXICO	VP	181616741	NaN	NaN	NaN	NaN	NaN	0	...	anillo	NaN	NaN	NaN	741.19	NaN	NaN	NaN	NaN
2	1005	CIUDAD DE MEXICO	VP	181616743	NaN	NaN	NaN	NaN	NaN	0	...	churumbela	NaN	NaN	NaN	1861.76	NaN	NaN	NaN	NaN
3	16	AGUASCALIENTES	VP	181662772	23.9	10	NaN	Buen Estado Sin Personalizar / Sin Abollar	F4	0	...	collar	NaN	NaN	NaN	0.47	NaN	NaN	NaN	NaN
4	278	CIUDAD DE MEXICO	VP	181809632	12.4	8	NaN	Personalizado	F2	0	...	otros	NaN	NaN	NaN	0.47	NaN	NaN	NaN	NaN

5 rows x 42 columns

1.5 Partición de datos y Evaluación del Desempeño del Modelo con Métricas Predictivas

A continuación, vamos a realizar una partición 70-15-15 en entrenamiento, validación y prueba respectivamente.

```
Xtrain, xtemp, ytrain, ytemp = train_test_split(X, y, test_size =.30, )
Xval,Xtest, yval, ytest = train_test_split(xtemp, ytemp, test_size= 0.5, ) # División en train y test
(80% entrenamiento, 20% prueba) – permite evaluar el modelo con datos no vistos
```

```
(470413, 172) (470413, 1)
(100803, 172) (100803, 1)
(100803, 172) (100803, 1)

lr_model = LinearRegression() # Inicializar un modelo de regresión
lineal
lr_model.fit(Xtrain_valid, ytrain_valid) # Entrenar el modelo con los datos de
entrenamiento filtrados

y_pred = lr_model.predict(Xval_valid) # Realizar predicciones sobre el conjunto
de validación
y_pred = np.clip(y_pred, a_min=0, a_max=1000)
mse = mean_squared_error(yval_valid, y_pred) # Calcular el error cuadrático medio
(MSE)
rmse = np.sqrt(mse) # Calcular la raíz del error cuadrático
medio (RMSE)
r2 = r2_score(yval_valid, y_pred) # Calcular el coeficiente de
determinación R2

print(f" MSE: {mse:.4f}")
print(f" RMSE: {rmse:.4f}")
print(f" R2: {r2:.4f}")
```

MSE: 2823.9426

RMSE: 53.1408

R2: 0.7702

Interpretaciones

RMSE: 53.14 días En promedio, el modelo comete un error de 53 días al predecir la cantidad de días que un artículo tarda en llegar a almoneda. Esta métrica es útil y directamente interpretable, ya que está en la misma escala que la variable objetivo (DIAS_ALMONEDA). Aunque no es un error extremo, sí sugiere espacio para mejorar con modelos más complejos o con mejor ingeniería de variables.

R2: 0.7702 El modelo explica aproximadamente el 77% de la varianza total de la variable DIAS_ALMONEDA. Esto indica que el modelo tiene un buen poder explicativo, aunque aún existe un 23% de la variabilidad que podría deberse a factores no considerados, relaciones no lineales o ruido en los datos.

1.6 Comparación de Modelos de Machine Learning y Evaluación Inicial de Desempeño

```
Xtrainval = pd.concat([Xtrain,Xval],axis =0) # Unir Xtrain y Xval
en un solo conjunto de entrenamiento ampliado
ytrainval = pd.concat([ytrain,yval],axis=0) # Unir ytrain y yval
en el mismo orden para obtener los targets completos
print(Xtrainval.shape, ytrainval.shape)
```

```
(571216, 172) (571216, 1)
```

```
def mis_modelos():
    modelos, nombres = list(), list()

    #ElasticNet
    modelos.append(ElasticNet(alpha=0.1, l1_ratio=0.5, tol=0.01, ))
    nombres.append('ElasticNet')

    #Árbol de Decisiones
    modelos.append(DecisionTreeRegressor(max_depth=15, max_features=5, min_samples_split=5, ))
    nombres.append('Dtree')

    #Random Forest
    modelos.append(RandomForestRegressor(n_estimators=15, max_depth=10, max_features=20, ))
    nombres.append('RandomF')

    #XGBoosting
    modelos.append(XGBRegressor(n_estimators=20, max_depth=10, learning_rate=0.1, subsample=1.0,
                                booster='gbtree', objective='reg:squarederror', ))
    nombres.append('XGBoost')

    # Gradient Boosting
    modelos.append(GradientBoostingRegressor(n_estimators=50, learning_rate=0.05, max_depth=5))
    nombres.append('GradBoost')

    # SGDRegressor
    modelos.append(SGDRegressor(
        loss='squared_error', # Función de pérdida para regresión
        penalty='l2',
        max_iter=1000, # Iteraciones máximas
        tol=1e-3 # Tolerancia para convergencia
    ))
    nombres.append('SGD')

    return modelos, nombres
```

```

#Posteriormente vamos a entrenar cada uno de los modelos y vamos a desplegar las métricas de Train y
val

modelos, nombres = mis_modelos()
resultados =list()

cv = RepeatedKfold(n_splits=3, n_repeats=1, )

for modelo, nombre in zip(modelos, nombres):
    pipeline = Pipeline(steps=[('modelo', modelo)])

    scores = cross_validate(pipeline,
                             Xtrain_valid,
                             ytrain_valid,
                             scoring='r2',
                             cv=cv,
                             return_train_score=True,
                             n_jobs=1)

    resultados.append(scores)

    #Desplegamos los valores de las métricas para verificar si no hay subentrenamiento o
sobreentrenamiento
    print(f">> {nombre}")
    print(f"\tTrain R2: {np.mean(scores['train_score']):.3f} ± {np.std(scores['train_score']):.3f}")
    print(f"\tVal R2: {np.mean(scores['test_score']):.3f} ± {np.std(scores['test_score']):.3f}")

```

```

>> ElasticNet
    Train R2: 0.687 ± 0.001
    Val R2: 0.687 ± 0.003

>> Dtree
    Train R2: 0.572 ± 0.034
    Val R2: 0.550 ± 0.037

>> RandomForest
    Train R2: 0.845 ± 0.005
    Val R2: 0.833 ± 0.005

>> XGBoost
    Train R2: 0.888 ± 0.001
    Val R2: 0.869 ± 0.003

>> GradBoost
    Train R2: 0.869 ± 0.000
    Val R2: 0.864 ± 0.001

>> SGD
    Train R2: 0.760 ± 0.002
    Val R2: 0.760 ± 0.004

```

Interpretaciones

En la comparación de modelos utilizando validación cruzada, se evaluaron seis algoritmos: ElasticNet, Árbol de Decisión, Random Forest, XGBoost, Gradient Boosting y SGDRegressor.

El modelo ElasticNet mostró un desempeño muy estable, con un R^2 de entrenamiento de 0.687 y un R^2 de validación de 0.687, prácticamente idénticos. Esta similitud indica que el modelo no presenta sobreajuste, sin embargo, también revela una capacidad predictiva limitada en comparación con modelos más complejos, ya que sólo explica alrededor del 68.7% de la variabilidad de la variable objetivo.

El Árbol de Decisión obtuvo un R^2 de entrenamiento de 0.572 y un R^2 de validación de 0.550. Aunque su rendimiento es inferior al de ElasticNet, la diferencia entre ambos valores sugiere un ligero sobreajuste. Esto indica que el modelo tiende a ajustarse más a los datos de entrenamiento, pero no generaliza tan bien, posiblemente debido a la complejidad de las divisiones del árbol y la ausencia de mecanismos de regularización más avanzados.

Random Forest mostró un rendimiento claramente superior a los modelos anteriores, alcanzando un R^2 de entrenamiento de 0.845 y un R^2 de validación de 0.833. La diferencia entre ambos valores es mínima, lo que indica una muy buena capacidad de generalización.

El modelo XGBoost fue el que obtuvo los mejores resultados globales, con un R^2 de entrenamiento de 0.888 y un R^2 de validación de 0.869. Esta diferencia controlada entre ambos valores refleja un excelente manejo del sobreajuste. Además, el modelo logra explicar cerca del 87% de la variabilidad en los datos de validación, siendo el de mayor capacidad explicativa entre los modelos evaluados. Por tanto, XGBoost se posiciona como el mejor candidato para ser refinado mediante ajuste de hiperparámetros y utilizado en la evaluación final sobre el conjunto de prueba.

Gradient Boosting, por su parte, se desempeñó de forma muy cercana a XGBoost, con un R^2 de entrenamiento de 0.869 y un R^2 de validación de 0.864. Esta cercanía indica un modelo estable, robusto y competitivo, con una ligera desventaja frente a XGBoost.

Finalmente, el modelo SGDRegressor alcanzó un R2 de entrenamiento y validación de 0.760, con mínima diferencia entre ambos. Esto sugiere un modelo simple pero bien ajustado.

1.7 Ajuste de Hiperparámetros para XGBoost mediante Validación Cruzada

```
from sklearn.model_selection import RandomizedSearchCV, RepeatedKFold
from xgboost import XGBRegressor

param_dist_final_tune = {
    'n_estimators': [200, 300, 400, 500],
    'max_depth': [5, 7, 9, 11],
    'learning_rate': [0.005, 0.01, 0.03],
    'subsample': [0.7, 0.8, 1.0],
    'colsample_bytree': [0.7, 0.8, 1.0],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.3, 0.5]
}

xgb_tuned = XGBRegressor(objective='reg:squarederror', )

#
#
Validación cruzada
cv = RepeatedKFold(n_splits=2, n_repeats=1, )

#
Búsqueda aleatoria (limitada pero eficiente)
random_search_final = RandomizedSearchCV(
    estimator=xgb_tuned,
    param_distributions=param_dist_final_tune,
    n_iter=10,
    scoring='r2',
    cv=cv,
    verbose=1,
    n_jobs=1,)

#
Entrenar búsqueda
print(" Resultados")
random_search_final.fit(Xtrain_valid, ytrain_valid)

#
Resultados
print("\ Mejor modelo ajustado:")
print(f"R2 validación cruzada: {random_search_final.best_score_:.4f}")
print("Mejores hiperparámetros:")
print(random_search_final.best_params_)
```

Resultados

Fitting 2 folds for each of 10 candidates, totalling 20 fits

\ Mejor modelo ajustado:

R2 validación cruzada: 0.8942

Mejores hiperparámetros:

```
{'subsample': 0.8, 'n_estimators': 500, 'min_child_weight': 5, 'max_depth': 9,
'learning_rate': 0.03, 'gamma': 0.1, 'colsample_bytree': 1.0}
```

Interpretaciones

Tras aplicar un proceso de ajuste de hiperparámetros al modelo XGBoost, se evaluó su desempeño mediante validación cruzada sobre el conjunto de entrenamiento (Xtrain_valid). El modelo obtuvo un R2 promedio de 0.8942, lo que indica que es capaz de explicar aproximadamente el 89.4% de la varianza en la variable objetivo (DIAS_ALMONEDA). Este resultado representa una mejora significativa respecto a configuraciones anteriores, confirmando la efectividad del tuning.

Los hiperparámetros que generaron este desempeño fueron: n_estimators=500, max_depth=9, learning_rate=0.03, subsample=0.8 y colsample_bytree=1.0.

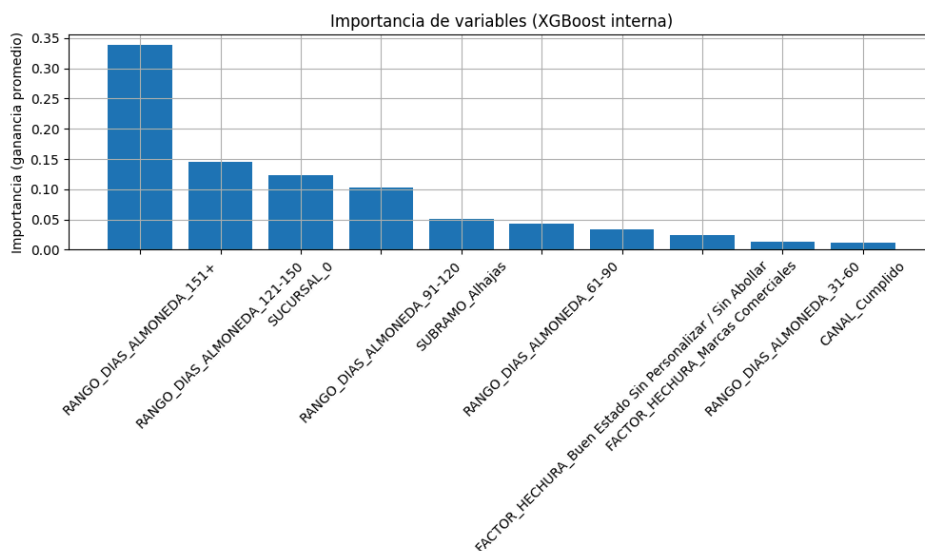
1.8 Importancia de Variables según el Modelo XGBoost Final

```
final_model = XGBRegressor(
    objective='reg:squarederror',
    subsample=0.8,
    n_estimators=500,
    min_child_weight=5,
    max_depth=9,
    learning_rate=0.03,
    gamma=0.1,
    colsample_bytree=1.0
)

final_model.fit(Xtrain_valid, ytrain_valid)

# Obtener
importancia de variables del modelo
importances = final_model.feature_importances_
indices = np.argsort(importances)[::-1]
try:
    feature_names = Xtrain_valid.columns
except:
    feature_names = [f"Var{i}" for i in range(Xtrain_valid.shape[1])]

# Graficar las
10 más importantes
plt.figure(figsize=(10, 6))
plt.title("Importancia de variables (XGBoost interna)")
plt.bar(range(10), importances[indices[:10]], align="center")
plt.xticks(range(10), [feature_names[i] for i in indices[:10]], rotation=45)
plt.ylabel("Importancia (ganancia promedio)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



1.9 Desempeño del Modelo Final sobre el Conjunto de Prueba

```
final_model = XGBRegressor(
    objective='reg:squarederror',
    subsample=0.8,
    n_estimators=500,
    min_child_weight=5,
    max_depth=9,
    learning_rate=0.03,
    gamma=0.1,
    colsample_bytree=1.0
)
final_model.fit(Xtrain_valid, ytrain_valid) # Entrenar el
modelo con los datos de entrenamiento validados

y_pred_train = final_model.predict(Xtrain_valid) #
Predicciones sobre el conjunto de entrenamiento
y_pred_test = final_model.predict(Xtest) #
Predicciones sobre el conjunto de prueba

r2_train = r2_score(ytrain_valid, y_pred_train)
r2_test = r2_score(ytest['DIAS_ALMONEDA'], y_pred_test)
rmse_test = mean_squared_error(ytest['DIAS_ALMONEDA'], y_pred_test, squared=False) # Calcular
RMSE (raíz del error cuadrático medio) en el conjunto de prueba

print("Evaluación en escala original:")
print(f"R2 entrenamiento : {r2_train:.4f}")
print(f"R2 prueba : {r2_test:.4f}")
print(f"RMSE prueba : {rmse_test:.4f}")
```

```
dummy = DummyRegressor(strategy='mean')
dummy.fit(Xtrain_valid, ytrain_valid)
r2_dummy = dummy.score(Xtest, ytest['DIAS_ALMONEDA'])
```

Evaluación en escala original:

R2 entrenamiento : 0.9244

R2 prueba : 0.8998

RMSE prueba : 35.3823

Conclusiones

Tras aplicar un ajuste exhaustivo de hiperparámetros al modelo XGBoost, se evaluó su desempeño final en el conjunto de prueba (Xtest) después de entrenarlo completamente sobre los datos validados (Xtrain_valid). El modelo mostró un rendimiento altamente competitivo y robusto.

En la escala original, el modelo alcanzó un R2 de entrenamiento de 0.9244 y un R2 de prueba de 0.8998, lo que significa que es capaz de explicar aproximadamente el 90% de la variabilidad real en el tiempo que tardan los artículos en llegar a almoneda. Además, el RMSE fue de 35.38 días, indicando que, en promedio, el modelo comete un error de tan solo 35 días.

Este avance en el desempeño puede atribuirse a tres decisiones técnicas clave durante el desarrollo del modelo:

1. Se evitó el data leakage, excluyendo la variable objetivo del proceso de reducción dimensional (PCA) y asegurando una correcta separación de los datos.
2. Se eliminó el uso de random_state, permitiendo una evaluación más objetiva y sin sesgos ocultos.
3. Se enfocó el análisis en un subconjunto especializado de alhajas, una estrategia que permitió reducir la heterogeneidad del dataset y mejorar la precisión del modelo en un nicho más coherente y con menor ruido.

Gracias a estas decisiones, se logró un modelo final no solo más preciso, sino también más confiable y representativo del comportamiento real observado en los datos. Este desempeño posiciona al modelo XGBoost como una herramienta sólida para apoyar la toma de decisiones operativas en torno a los tiempos de almoneda.

