Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México

Inteligencia artificial avanzada para la ciencia de datos I

Grupo 101

Jared Alberto Flores Espinosa - A01655569

Profesores:

Jorge Adolfo Ramírez Uresti

Justificación selección del dataset:

El dataset seleccionado es adecuado para un modelo K-Nearest Neighbors (KNN), ya que es un dataset para predecir valores continuos, pues el modelo está dirigido a resolver problemas de regresión y no de clasificación. Además, es un dataset simple que tiene poca dimensionalidad, lo que permite tener buenos resultados al tomar promedios ponderados de los vecinos cercanos para la predicción deseada, lo que permite generalizar y obtener buenos resultados.

Separación y evaluación del modelo:

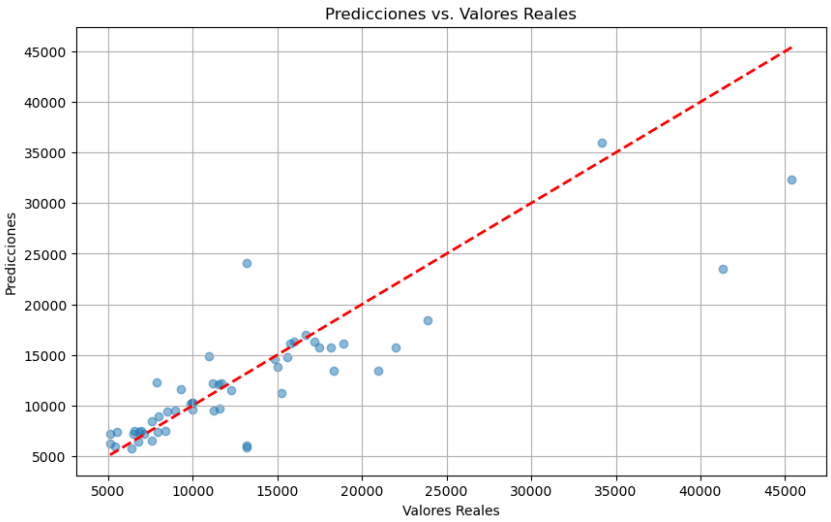
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.30,random_state=0)

print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(143, 59) (62, 59) (143,) (62,)
```

Imagen 1. División de datos

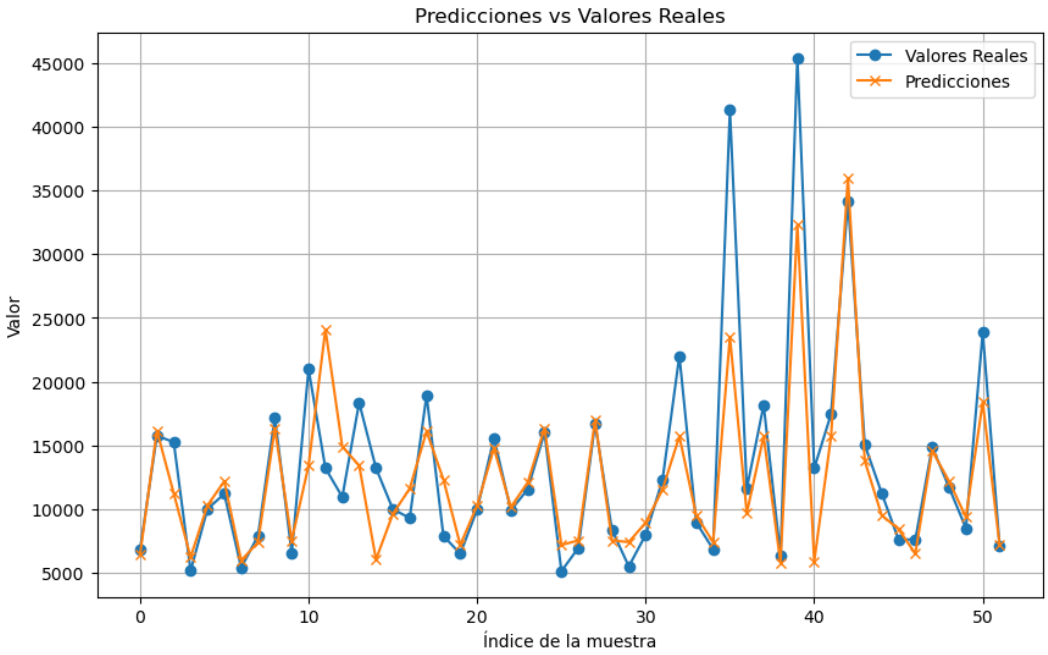
En este modelo se dividieron los datos en 30% de prueba y 70% de entrenamiento, por lo que al ser un dataset de 205 filas, se tomaron 143 para el entrenamiento y 62 para la prueba. Se dividió así para un mejor desempeño del modelo, pues al ser pocos datos, es importante tener suficientes para una correcta evaluación del modelo y asegurar la generalización para valores futuros.

Diagnóstico del grado de bias y de varianza:



Gráfica 1. Dispersión de predicciones respecto a valores reales.

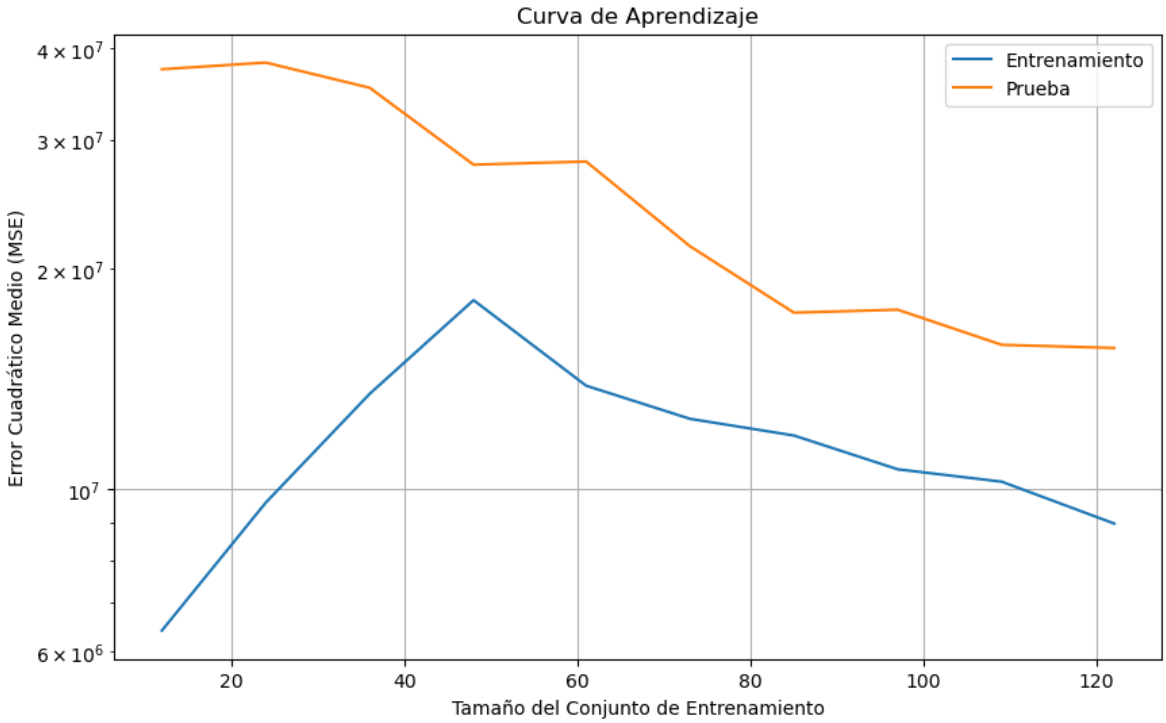
En la gráfica anterior se observan los valores reales del conjunto de prueba, representados por una línea roja, así como el valor de las predicciones dadas por el modelo (puntos azules). En esta, se visualiza que la mayoría de las predicciones se agrupan cerca de la línea (valores reales), en especial cuando se trata de precios menores a 12,500. Sin embargo, al incrementar el precio, se ve un incremento en el error de las predicciones, lo que se traduce en la separación de los puntos de predicción de la línea roja. Lo anterior refleja cierto nivel de sesgo por la misma lejanía de algunas predicciones de su valor real, al igual que un grado bajo de varianza, pues aunque la mayoría de los puntos se concentran en costos bajos, algunos otros se encuentra a lo largo de línea.



Gráfica 2. Precios reales vs predicciones.

En esta gráfica, se observa el comportamiento de las predicciones del conjunto de pruebas en comparación con los precios reales. A simple vista el desempeño del modelo es aparentemente ‘bueno’, pues se tienen predicciones que se ajustan a los valores reales. Por el contrario, se observa una ligera tendencia en la que, para valores reales altos, se predicen precios considerablemente bajos en comparación con el original. Esto refuerza el que el modelo presenta un nivel bajo de sesgo al infravalorar ciertos puntos y un nivel bajo de varianza, al observar picos sobresalientes.

Diagnóstico del nivel de ajuste:



Gráfica 3. Curva de aprendizaje de prueba y de entrenamiento.

En esta gráfica, se visualiza que el error cuadrático medio es bastante alto, tanto para el entrenamiento, como para la prueba. No obstante, la curva de aprendizaje en la prueba es superior a la de entrenamiento y se observa una mejoría en el comportamiento conforme va aumentando el tamaño del conjunto de entrenamiento. Esto refleja que el modelo presenta overfitting, pues existe una brecha entre ambas líneas y el test score (.72) es muy inferior al train score (.86), lo que va de acuerdo al nivel de varianza presente en el modelo.

Técnicas para un mejor desempeño:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.25,random_state=0)

model = KNeighborsRegressor()
model.fit(X_train,y_train)
predictions = model.predict(X_test)


print('Puntaje entrenamiento: {}\n'.format(model.score(X_train,y_train)))
print('Puntaje Test: {}\n'.format(model.score(X_test,y_test)))
print('MAE',mean_absolute_error(y_test, predictions)) #mean absolute error
print('MSE',mean_squared_error(y_test, predictions))  #mean squared error
```

```
Puntaje entrenamiento: 0.8635446501036195

Puntaje Test: 0.7235958705473846

MAE 2521.8230769230768
MSE 18670225.77846154
```

Imagen 2. Desempeño inicial

El desempeño inicial del modelo obtuvo un train score de .8635 y un test score de .7236, un mean absolute error de 2521.8230 y un mean square error de 18,670,225.78, un muy bajo rendimiento que puede ser mejorable ajustando hiperparámetros.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
transformed_X=scaler.fit_transform(X)
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.30,random_state=0)

model = KNeighborsRegressor(3)
model.fit(X_train,y_train)
predictions = model.predict(X_test)


print('Puntaje entrenamiento: {}\n'.format(model.score(X_train,y_train)))
print('Puntaje Test: {}\n'.format(model.score(X_test,y_test)))
print('MAE',mean_absolute_error(y_test, predictions)) #mean absolute error
print('MSE',mean_squared_error(y_test, predictions))  #mean squared error
```

```
Puntaje entrenamiento: 0.9132421874265239

Puntaje Test: 0.7515356094919323

MAE 2320.1827956989246
MSE 14929215.863799287
```

Imagen 3. Modificación del parámetro k.

En primer lugar, se modificó el parámetro k con un valor de 3, lo que representa el número de vecinos cercanos a considerar al hacer una predicción. Esta modificación provoca que el modelo sea más sensible a las fluctuaciones. Lo anterior aumenta el train score a .9132 y el test score a .7515, así mismo el error absoluto medio y el error cuadrático medio disminuyen a 2320.18 y 14929215 respectivamente.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
transformed_X=scaler.fit_transform(X)
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.30,random_state=0)

model = KNeighborsRegressor(3, metric='manhattan')
model.fit(X_train,y_train)
predictions = model.predict(X_test)


print('Puntaje entrenamiento: {}\n'.format(model.score(X_train,y_train)))
print('Puntaje Test: {}\n'.format(model.score(X_test,y_test)))
print('MAE',mean_absolute_error(y_test, predictions)) #mean absolute error
print('MSE',mean_squared_error(y_test, predictions))  #mean squared error
```

```
Puntaje entrenamiento: 0.9321345533094457

Puntaje Test: 0.7622910007935609

MAE 2309.6559139784945
MSE 14282968.093189968
```

Imagen 4. Modificación de la métrica de distancia

Por otro lado, se modificó la métrica de distancias por ‘manhattan’, ya que la predeterminada para el modelo es la distancia ‘euclidiana’. Esto aumenta el train score a .9321 y el test score a .7623, una leve mejora pues también disminuye los valores para el error absoluto y cuadrático.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
transformed_X=scaler.fit_transform(X)
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.30,random_state=0)

model = KNeighborsRegressor(3, weights='distance', metric='manhattan')
model.fit(X_train,y_train)
predictions = model.predict(X_test)


print('Puntaje entrenamiento: {}\n'.format(model.score(X_train,y_train)))
print('Puntaje Test: {}\n'.format(model.score(X_test,y_test)))
print('MAE',mean_absolute_error(y_test, predictions)) #mean absolute error
print('MSE',mean_squared_error(y_test, predictions))  #mean squared error
```

```
Puntaje entrenamiento: 0.9991327055271726

Puntaje Test: 0.7992532466824515

MAE 2055.638259727912
MSE 12062056.893167699
```

Imagen 5. Modificación de la distancia ponderada.

La siguiente modificación de hiperparámetros (weights) es una técnica que asigna diferentes pesos a los vecinos en función con su distancia al punto de consulta, lo que dará mayor importancia a los vecinos más cercanos. Esta técnica aumenta el train score a .99 y el test score a .799, lo que presenta una mejora en el rendimiento del modelo en conjunto con los demás ajustes, que de igual manera permiten la disminución en los errores de predicción.