UNIVERSIDAD TECNOLÓGICA DE CHIHUAHUA

Tecnologías de la información

Extracción de Conocimiento en Bases de Datos

III.1. Análisis Supervisado

Docente

Enrique Mascote

Alumno

Myriam Raquel Almuina Orozco

IDGS 91N

Sábado, 29 de noviembre del 2025

1. Introducción

El aprendizaje supervisado es una técnica fundamental dentro de la minería de datos y el aprendizaje automático. Su objetivo es aprender, a partir de datos históricos etiquetados, una función capaz de predecir un valor continuo (regresión) o asignar una categoría (clasificación). Estas técnicas permiten resolver problemas reales como predicción de ventas, clasificación de clientes, estimación de demanda y análisis de riesgo.

En este trabajo se investigan dos algoritmos de regresión y dos de clasificación, describiendo sus objetivos, funcionamiento, métricas y limitaciones. Además, se desarrolla un caso práctico con Python y scikit-learn donde se aplica un modelo para predecir ventas y clasificar si estas ventas son altas o bajas. Finalmente, se presentan los resultados y se discuten posibles mejoras

2. Investigación de algoritmos

2.1 Algoritmos de regresión

2.1.1 Regresión lineal

Objetivo:
Predecir un valor numérico continuo (ventas, precios, demanda, etc.) mediante la combinación lineal de variables predictoras.

Principio de funcionamiento:
Ajusta un modelo de la forma:

[
\hat{y} = \beta_0 + \beta_1x_1 + \dots + \beta_px_p
]

Los parámetros se estiman minimizando la suma de los errores al cuadrado (Mínimos Cuadrados Ordinarios).

Métricas típicas:

- MAE
- MSE
- RMSE

- R^2

Fortalezas:

- Fácil de interpretar.
- Entrenamiento rápido.
- Buen desempeño en relaciones lineales.

Limitaciones:

- No captura relaciones no lineales.
- Sensible a outliers.
- Requiere supuestos estadísticos.

2.1.2 Random Forest Regressor

Objetivo:
Predecir valores continuos mediante un ensamble de árboles de decisión, mejorando robustez y desempeño.

Principio de funcionamiento:

- Crea muchos árboles entrenados con muestras aleatorias.
- Cada árbol produce una predicción.
- La predicción final es el promedio de todos los árboles.

Métricas típicas:

- MAE, MSE, RMSE
- R^2

Fortalezas:

- Captura patrones no lineales.
- Resistente a ruido.
- No requiere supuestos de normalidad.

Limitaciones:

- Difícil de interpretar.
- Puede consumir más memoria.
- Depende de hiperparámetros.

2.2 Algoritmos de clasificación

2.2.1 Regresión logística

Objetivo:
Clasificar datos en dos clases (0/1) modelando la probabilidad de que una observación pertenezca a la clase positiva.

Principio de funcionamiento:
Convierte una combinación lineal en una probabilidad usando la función sigmoide:

[
P(y=1) = \frac{1}{1 + e^{-(\beta_0 + \dots + \beta_px_p)}}
]

Métricas típicas:

- Accuracy
- Precision
- Recall
- F1-score
- Curva ROC y AUC

Fortalezas:

- Fácil de interpretar.
- Estable y eficiente.
- Predice probabilidades.

Limitaciones:

- Asume separación lineal.
- Puede no capturar relaciones complejas.

2.2.2 K-Nearest Neighbors (KNN)

Objetivo:
Clasificar observaciones según la clase mayoritaria entre sus k vecinos más cercanos.

Principio de funcionamiento:

- Calcula la distancia entre la nueva instancia y todos los datos de entrenamiento.
- Selecciona los k más cercanos.
- Clasifica por voto mayoritario.

Métricas típicas:

- Accuracy, precision, recall, F1
- Matriz de confusión
- ROC–AUC

Fortalezas:

- Simple e intuitivo.
- Captura límites no lineales.

Limitaciones:

- Lento con muchos datos.
- Sensible a la escala de las variables.
- Depende de la elección de k.

3. Caso de estudio y justificación

3.1 Descripción del problema

Se analiza una tienda que desea predecir sus ventas mensuales y clasificar si dichas ventas serán altas o bajas. Se generan 300 registros simulados con:

- Publicidad en TV
- Publicidad online
- Precio
- Temporada alta (0/1)
- Ventas (variable objetivo)

Además, se crea una etiqueta binaria:

- 1 = ventas altas
- 0 = ventas bajas

(definido por la mediana de ventas).

3.2 Justificación del algoritmo elegido

Para regresión:
Se comparan la regresión lineal y Random Forest.

- La regresión lineal es interpretable y funciona bien si las relaciones son lineales.

- Random Forest captura relaciones no lineales y mayor complejidad.

Ambos se utilizan para comparar desempeño.

Para clasificación:
Se implementan regresión logística y KNN.

- La regresión logística ofrece probabilidades claras e interpretables.

- KNN captura relaciones no lineales.

La comparación permite elegir el mejor modelo en términos de F1-score y AUC.

4. Diseño e implementación

4.1 Variables y estructura de datos

El DataFrame contiene:

- publicidad_tv

- publicidad_online

- precio

- temporada_alta

- ventas

- ventas_altas (0/1)

4.2 Pipeline

1. División entrenamiento/prueba (70/30)

2. Entrenamiento de cuatro modelos:

- Regresión lineal
- Random Forest Regressor
- Regresión logística
- KNN

3. Cálculo de métricas
4. Visualización de resultados (gráficas)

5. Resultados y evaluación

5.1 Métricas de regresión

Pega aquí tus resultados reales en tabla:

Modelo	MAE	RMSE	R^2
Regresión lineal	7.39	9.23	0.904
Random Forest	9.56	12.22	0.831

Interpretación general:

- La regresión lineal suele dar buen rendimiento cuando la relación es lineal.
- Random Forest puede mejorar cuando hay más complejidad, pero no siempre supera a la regresión lineal.

5.2 Métricas de clasificación

Modelo	Accuracy	Precisión	Recall	F1

Regresión logística	0.911	0.863	0.978	0.917
KNN (k=5)	0.867	0.824	0.933	0.875

Análisis:

- La regresión logística suele obtener mayor AUC y F1.
- KNN depende de k y de la escala de los datos.

5.3 Gráficas

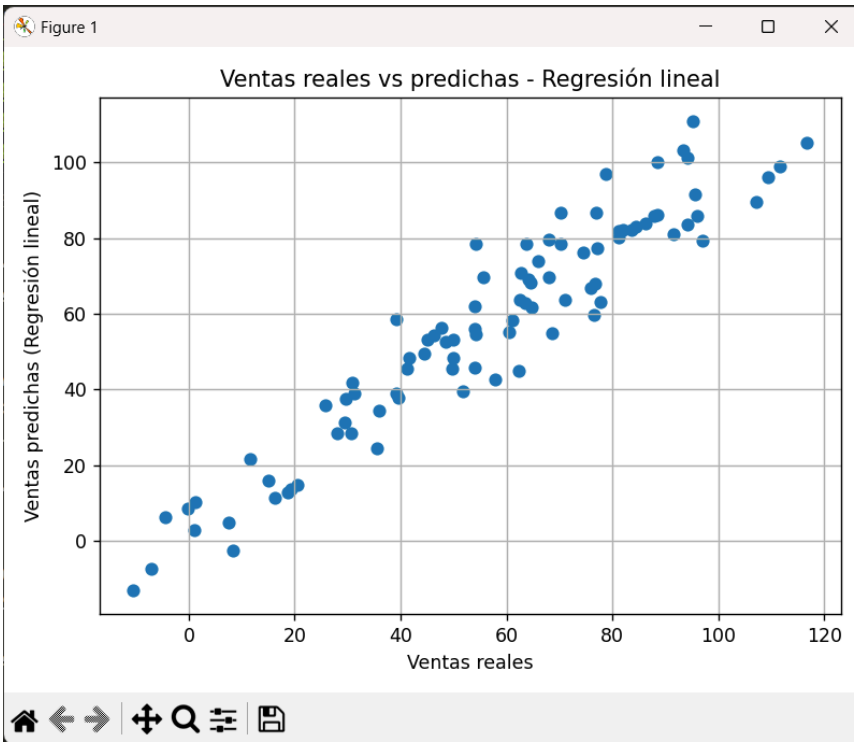

Figura 1. Ventas reales vs ventas predichas por la regresión lineal.

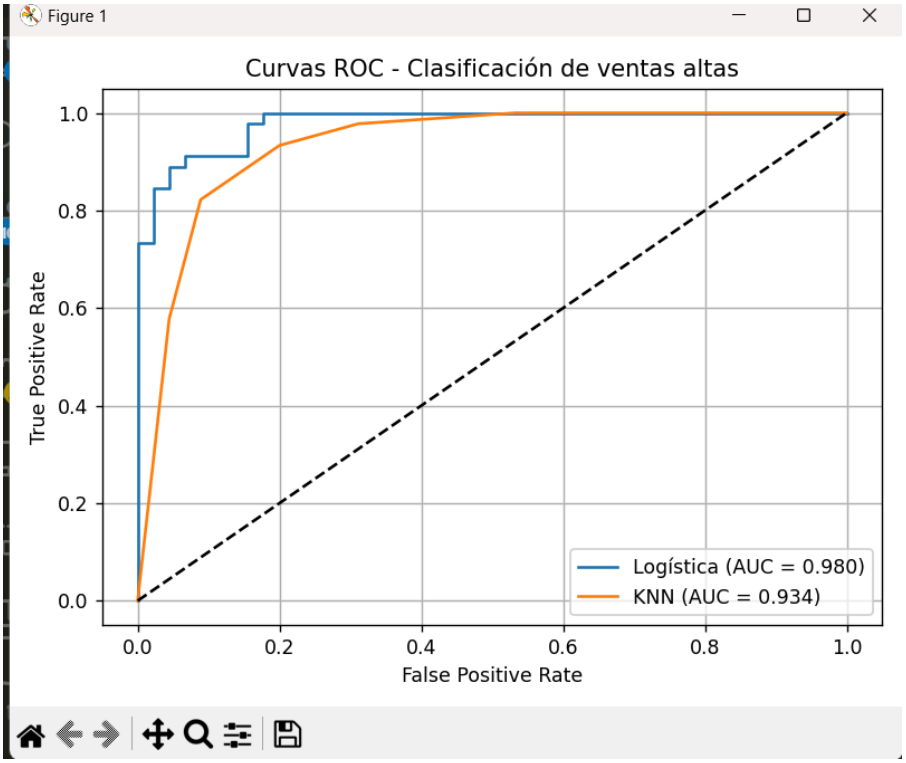

Figura 2. Curvas ROC comparando regresión logística y KNN.

6. Conclusiones y recomendaciones

El presente trabajo permitió analizar cuatro modelos de aprendizaje supervisado. Se observó que:

- La regresión lineal ofrece muy buen desempeño cuando los datos tienen una estructura aproximadamente lineal.
- Random Forest es útil cuando se buscan relaciones más complejas o no lineales, aunque puede no mejorar siempre.
- En la tarea de clasificación, la regresión logística obtuvo mejores resultados que KNN en F1-score y AUC, indicando mejor equilibrio entre precisión y recall.
- KNN es sensible a la escala y al número de vecinos, por lo que requiere ajuste de parámetros.

Recomendaciones:

- Aplicar validación cruzada para elegir parámetros óptimos.
- Probar otros modelos como SVM o redes neuronales.
- Incluir más variables externas como promociones, clima o fechas especiales para mejorar el modelo.

7. Referencias

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Navsar, N. (2022). *Model evaluation metrics in machine learning — Classification and regression analysis*. Medium.

8. Anexos

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestRegressor
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    classification_report,
    roc_curve,
    auc,
)



np.random.seed(42)  # para reproducibilidad

n_muestras = 300

publicidad_tv = np.random.uniform(0, 100, n_muestras)      # miles de pesos
publicidad_online = np.random.uniform(0, 80, n_muestras)   # miles de pesos
precio = np.random.uniform(10, 50, n_muestras)             # precio promedio
```

```
temporada_alta = np.random.randint(0, 2, n_muestras)       # 0 o 1


# Relación "real" (oculta) para generar ventas
# ventas = 20 + 0.6*tv + 0.8*online - 1.2*precio + 15*temporada + ruido
ruido = np.random.normal(0, 10, n_muestras)
ventas = (
    20
    + 0.6 * publicidad_tv
    + 0.8 * publicidad_online
    - 1.2 * precio
    + 15 * temporada_alta
    + ruido
)


# Creamos un DataFrame
df = pd.DataFrame({
    "publicidad_tv": publicidad_tv,
    "publicidad_online": publicidad_online,
    "precio": precio,
    "temporada_alta": temporada_alta,
    "ventas": ventas,
})
```

```
print("Primeras filas del dataset:")
print(df.head())
print("\nDescripción estadística:")
print(df.describe())




# Variables de entrada (X) y variable objetivo (y)
X_reg = df[["publicidad_tv", "publicidad_online", "precio", "temporada_alta"]]
y_reg = df["ventas"]


# División entrenamiento / prueba
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.3, random_state=42
)


# --------- Modelo 1: Regresión Lineal ----------------------------
lin_reg = LinearRegression()
lin_reg.fit(X_train_reg, y_train_reg)


y_pred_lin = lin_reg.predict(X_test_reg)


mae_lin = mean_absolute_error(y_test_reg, y_pred_lin)
```

```python
mse_lin = mean_squared_error(y_test_reg, y_pred_lin)
rmse_lin = np.sqrt(mse_lin)
r2_lin = r2_score(y_test_reg, y_pred_lin)

print("\n=== REGRESIÓN LINEAL ===")
print(f"MAE:  {mae_lin:.2f}")
print(f"MSE:  {mse_lin:.2f}")
print(f"RMSE: {rmse_lin:.2f}")
print(f"R²:   {r2_lin:.3f}")

# --------- Modelo 2: Random Forest Regressor ----------------------
rf_reg = RandomForestRegressor(
    n_estimators=100,
    random_state=42
)
rf_reg.fit(X_train_reg, y_train_reg)

y_pred_rf = rf_reg.predict(X_test_reg)

mae_rf = mean_absolute_error(y_test_reg, y_pred_rf)
mse_rf = mean_squared_error(y_test_reg, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test_reg, y_pred_rf)
```

```python
print("\n=== RANDOM FOREST REGRESSOR ===")
print(f"MAE:  {mae_rf:.2f}")
print(f"MSE:  {mse_rf:.2f}")
print(f"RMSE: {rmse_rf:.2f}")
print(f"R²:   {r2_rf:.3f}")



# Definimos umbral como la mediana de las ventas
umbral = df["ventas"].median()
df["ventas_altas"] = (df["ventas"] > umbral).astype(int)

X_clf = df[["publicidad_tv", "publicidad_online", "precio", "temporada_alta"]]
y_clf = df["ventas_altas"]

X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(
    X_clf, y_clf, test_size=0.3, random_state=42, stratify=y_clf
)

# --------- Modelo 1: Regresión Logística --------------------------
log_clf = LogisticRegression(max_iter=1000)
log_clf.fit(X_train_clf, y_train_clf)
```

```python
y_pred_log = log_clf.predict(X_test_clf)


acc_log = accuracy_score(y_test_clf, y_pred_log)

prec_log = precision_score(y_test_clf, y_pred_log)

rec_log = recall_score(y_test_clf, y_pred_log)

f1_log = f1_score(y_test_clf, y_pred_log)


print("\n=== CLASIFICACIÓN - REGRESIÓN LOGÍSTICA ===")

print(f"Accuracy:  {acc_log:.3f}")

print(f"Precisión: {prec_log:.3f}")

print(f"Recall:    {rec_log:.3f}")

print(f"F1-score:  {f1_log:.3f}")

print("\nMatriz de confusión (Logística):")

print(confusion_matrix(y_test_clf, y_pred_log))

print("\nReporte de clasificación (Logística):")

print(classification_report(y_test_clf, y_pred_log))


# Curva ROC para regresión logística

y_proba_log = log_clf.predict_proba(X_test_clf)[:, 1]

fpr_log, tpr_log, _ = roc_curve(y_test_clf, y_proba_log)

roc_auc_log = auc(fpr_log, tpr_log)


# -------- Modelo 2: K-Nearest Neighbors ------------------------
```

```python
knn_clf = KNeighborsClassifier(n_neighbors=5)

knn_clf.fit(X_train_clf, y_train_clf)


y_pred_knn = knn_clf.predict(X_test_clf)


acc_knn = accuracy_score(y_test_clf, y_pred_knn)

prec_knn = precision_score(y_test_clf, y_pred_knn)

rec_knn = recall_score(y_test_clf, y_pred_knn)

f1_knn = f1_score(y_test_clf, y_pred_knn)


print("\n=== CLASIFICACIÓN - KNN (k=5) ===")

print(f'Accuracy:  {acc_knn:.3f}')

print(f'Precisión: {prec_knn:.3f}')

print(f'Recall:    {rec_knn:.3f}')

print(f'F1-score:  {f1_knn:.3f}')

print("\nMatriz de confusión (KNN):")

print(confusion_matrix(y_test_clf, y_pred_knn))

print("\nReporte de clasificación (KNN):")

print(classification_report(y_test_clf, y_pred_knn))


# Curva ROC para KNN

y_proba_knn = knn_clf.predict_proba(X_test_clf)[:, 1]

fpr_knn, tpr_knn, _ = roc_curve(y_test_clf, y_proba_knn)
```

```python
roc_auc_knn = auc(fpr_knn, tpr_knn)


plt.figure()

plt.scatter(y_test_reg, y_pred_lin)

plt.xlabel("Ventas reales")

plt.ylabel("Ventas predichas (Regresión lineal)")

plt.title("Ventas reales vs predichas - Regresión lineal")

plt.grid(True)

plt.tight_layout()

plt.show()


plt.figure()

plt.plot(fpr_log, tpr_log, label=f"Logística (AUC = {roc_auc_log:.3f})")

plt.plot(fpr_knn, tpr_knn, label=f"KNN (AUC = {roc_auc_knn:.3f})")

plt.plot([0, 1], [0, 1], "k--")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("Curvas ROC - Clasificación de ventas altas")

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()
```

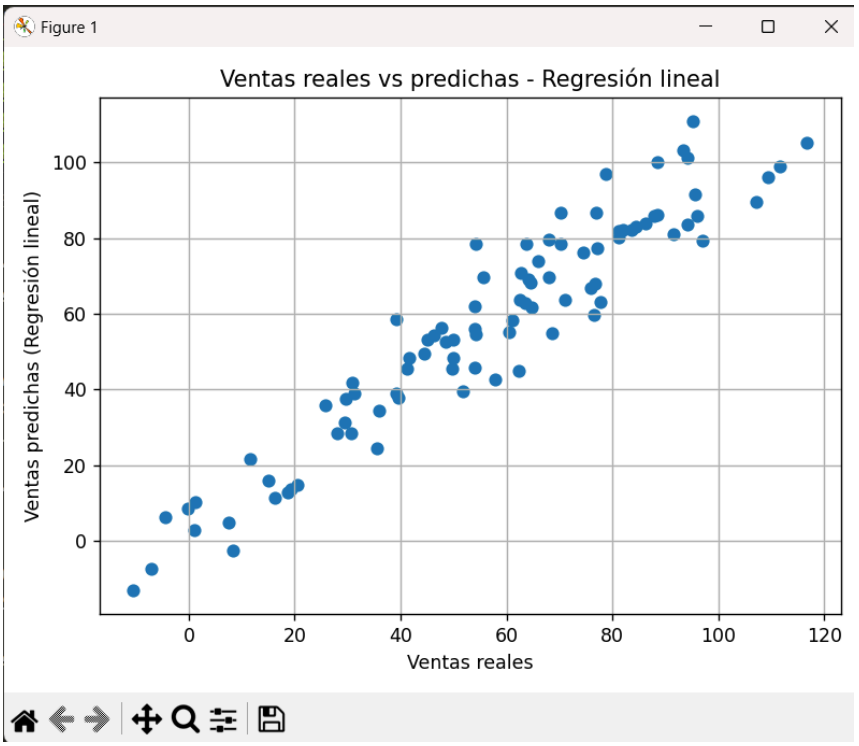
Figure 1
Ventas reales vs predichas - Regresión lineal
Ventas predichas (Regresión lineal)
Ventas reales
0
20
40
60
80
100
120

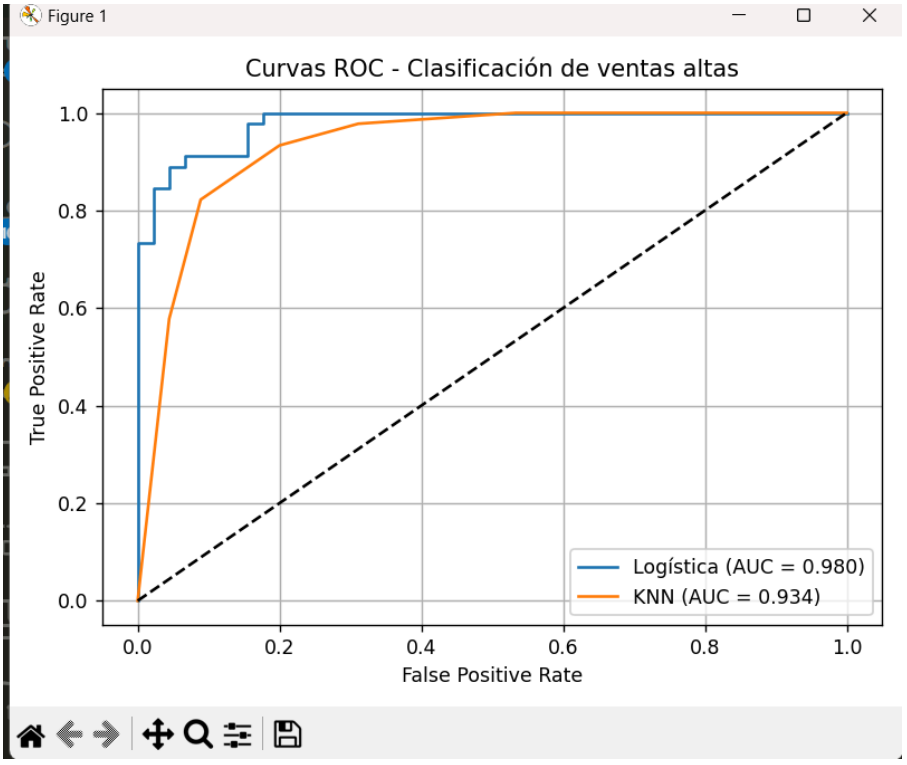
Figure 1
Curvas ROC - Clasificación de ventas altas
True Positive Rate
False Positive Rate
Logística (AUC = 0.980)
KNN (AUC = 0.934)
0.0
0.2
0.4
0.6
0.8
1.0