UNIVERSIDAD TECNOLÓGICA DE CHIHUAHUA

Tecnologías de la información

Extracción de conocimiento en base de datos

III.1. Análisis Supervisado (50%)

IDGS91N

Presenta:

Brahn Raudales

Docente:

Enrique Mascote

sábado, 29 de noviembre de 2025

Índice (opcional)

1. Introducción

En el contexto de la ciencia de datos, el **aprendizaje supervisado** es una de las técnicas fundamentales para construir modelos que puedan hacer predicciones a partir de datos históricos. Dentro de este enfoque se encuentran dos grandes tipos de problemas: **regresión**, donde se predicen valores numéricos continuos, y **clasificación**, donde el objetivo es asignar una etiqueta o categoría a cada ejemplo.

El objetivo de este trabajo es **identificar y comprender el proceso de investigación e implementación de modelos de regresión y clasificación**, incluyendo su justificación, diseño, métricas y evaluación. Para ello, primero se investigan dos algoritmos de regresión y dos de clasificación, analizando su objetivo, principio de funcionamiento, métricas típicas, fortalezas y limitaciones. Posteriormente, se desarrolla un **caso de estudio supervisado** donde se elige uno de estos algoritmos, se diseña el modelo, se muestra su implementación en Python con la biblioteca *scikit-learn* y se analizan los resultados obtenidos.

Este documento busca no solo cumplir con los requisitos académicos de la materia, sino también servir como una guía práctica para futuros proyectos de minería de datos donde sea necesario aplicar modelos de regresión y clasificación de manera justificada y metodológica.

2. Investigación de algoritmos

En esta sección se presentan **dos algoritmos de regresión** y **dos algoritmos de clasificación**, todos ellos ampliamente utilizados en proyectos de análisis supervisado.

2.1 Regresión lineal

Qué resuelve (objetivo)
La **regresión lineal** busca modelar la relación entre una variable dependiente continua (por ejemplo, ventas, precio, temperatura) y una o varias variables independientes (por ejemplo, publicidad, tamaño, número de habitaciones), asumiendo que dicha relación puede aproximarse con una función lineal.

Principio de funcionamiento (proceso)
La idea básica consiste en ajustar una recta (o un hiperplano en dimensiones mayores) a los datos, minimizando el **error cuadrático medio** entre las predicciones del modelo y los valores reales. Esto se logra estimando los parámetros β de la ecuación:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

mediante métodos como **mínimos cuadrados ordinarios (OLS)**.

Métricas de evaluación típicas

- **MAE (Mean Absolute Error):** promedio del valor absoluto de los errores.
- **MSE (Mean Squared Error):** promedio de los errores al cuadrado.
- **RMSE (Root Mean Squared Error):** raíz cuadrada del MSE, en las mismas unidades que la variable objetivo.
- **R^2 (coeficiente de determinación):** proporción de varianza explicada por el modelo.

Fortalezas

- Fácil de interpretar (coeficientes, signo y magnitud).
- Rápida de entrenar incluso con muchos datos.
- Sirve como línea base para comparar modelos más complejos.

Limitaciones

- Asume **relación lineal** entre variables, lo que no siempre es cierto.
- Sensible a **outliers**.
- Puede tener mal desempeño si las relaciones son muy no lineales o si hay alta colinealidad entre variables.

2.2 Random Forest Regressor

Qué resuelve (objetivo)
El **Random Forest Regressor** resuelve problemas de regresión en los que se desea

capturar relaciones no lineales entre variables y obtener un modelo robusto y con buena capacidad de generalización.

Principio de funcionamiento (proceso)
Un **Random Forest** es un conjunto (ensemble) de múltiples **árboles de decisión** entrenados sobre subconjuntos aleatorios de los datos y de las características. Cada árbol produce una predicción y el resultado final del bosque se obtiene promediando las predicciones de todos los árboles.

Pasos generales:

1. Se generan muchas muestras *bootstrap* del conjunto de entrenamiento.
2. Para cada muestra se entrena un árbol de decisión profundo.
3. Para cada división, el árbol considera solo un subconjunto aleatorio de características.
4. La predicción final es el **promedio** de las predicciones individuales.

Métricas típicas

- MAE, MSE, RMSE, R^2 (igual que otros modelos de regresión).

Fortalezas

- Captura relaciones **no lineales** y **interacciones** entre variables.
- Generalmente tiene buen rendimiento sin tanto ajuste fino.
- Menos sensible al sobreajuste que un solo árbol, gracias al promedio de múltiples modelos.

Limitaciones

- Menos interpretable que la regresión lineal.
- Puede ser **computacionalmente más costoso** (muchos árboles).
- No es ideal cuando se requiere una explicación muy detallada del modelo para usuarios no técnicos.

2.3 Regresión logística (clasificación)

Qué resuelve (objetivo)
La **regresión logística** es un algoritmo de **clasificación binaria** (por ejemplo, cliente compra / no compra, fraude / no fraude, aprueba / reprueba). El objetivo es modelar la probabilidad de que una observación pertenezca a una clase (generalmente la clase positiva).

Principio de funcionamiento (proceso)
Aunque su nombre incluya “regresión”, la salida del modelo es una **probabilidad**. La regresión logística aplica la función sigmoide a una combinación lineal de las variables:

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}}$$

Luego, se define un **umbral** (por ejemplo, 0.5) para convertir la probabilidad en una etiqueta de clase.

Métricas de evaluación típicas

- **Accuracy (exactitud).**
- **Precision (precisión).**
- **Recall (exhaustividad o sensibilidad).**
- **F1-score (media armónica entre precision y recall).**
- **AUC-ROC** cuando se trabaja con probabilidades.

Fortalezas

- Modelo relativamente sencillo e interpretable.
- Las probabilidades son útiles para tomar decisiones basadas en riesgo.
- Rápido de entrenar incluso con grandes volúmenes de datos.

Limitaciones

- Asume una relación aproximadamente lineal entre las variables y el logit de la probabilidad.
- Puede no capturar patrones muy complejos o no lineales sin ingeniería de características.
- Sensible a variables muy correlacionadas si no se preprocesan adecuadamente.

2.4 Random Forest Classifier

Qué resuelve (objetivo)
El **Random Forest Classifier** se usa para problemas de clasificación binaria o multiclase, como segmentar clientes, detectar churn, clasificar correos como spam/no spam, etc.

Principio de funcionamiento (proceso)
Al igual que en su versión de regresión, el Random Forest Classifier:

1. Genera múltiples conjuntos de entrenamiento mediante *bootstrap*.
2. Entrena un árbol de decisión de clasificación en cada uno de ellos.
3. Cada árbol se entrena con una selección aleatoria de características en cada partición.
4. La predicción final se obtiene por **votación mayoritaria** de todos los árboles.

Métricas de evaluación típicas

- Accuracy.
- Precision, Recall, F1-score.
- Matriz de confusión.
- AUC-ROC (para problemas binarios con probabilidades).

Fortalezas

- Alto rendimiento en muchos problemas de la vida real.
- Maneja bien relaciones no lineales y datos con ruido.
- Proporciona medidas de **importancia de características**.

Limitaciones

- Menos interpretable que modelos lineales.
- Puede requerir más memoria y tiempo de cómputo.
- Si no se ajustan parámetros, puede ser más pesado de lo necesario.

3. Caso de estudio y justificación

Para el caso de estudio, se plantea el siguiente problema:

Problema: Una tienda en línea desea **clasificar a sus clientes** según la probabilidad de que **respondan positivamente** a una campaña de marketing por correo electrónico (por ejemplo, realizar una compra después de recibir una promoción).

La variable objetivo será binaria:

- **1:** el cliente respondió a la campaña (realizó una compra).
- **0:** el cliente no respondió.

Variables de entrada (ejemplos):

- Edad del cliente.
- Ingreso estimado.
- Número de compras en los últimos 6 meses.
- Monto total gastado en el último año.
- Si ha respondido a campañas previas (sí/no).
- Canal preferido (web, app, teléfono).

Justificación del algoritmo elegido

De los algoritmos descritos en la Sección 2, se elige el **Random Forest Classifier** para resolver este caso de estudio, por las siguientes razones:

- Es capaz de capturar **relaciones no lineales** entre las variables de los clientes y la probabilidad de respuesta.
- Es robusto ante **valores atípicos** y datos con cierto ruido.
- Tiene un buen desempeño general sin necesidad de un ajuste extremadamente fino de hiperparámetros.
- Proporciona una medida de **importancia de características**, lo que ayuda al área de marketing a entender qué variables influyen más en la respuesta del cliente.
- Comparado con la regresión logística, suele lograr mejor rendimiento en problemas con patrones complejos, aunque se sacrifica algo de interpretabilidad.

4. Diseño e implementación

4.1 Diseño del modelo y pipeline

Estructura de datos

Se asume un conjunto de datos tabular con un registro por cliente y las siguientes columnas:

- edad (numérica)
- ingreso_mensual (numérica)
- compras_6m (numérica)
- monto_anual (numérica)
- respuesta_anterior (categórica: "sí"/"no")
- canal_preferido (categórica: "web", "app", "telefono")
- respondio_campania (0 o 1, variable objetivo)

Pipeline de entrenamiento (alto nivel)

1. **Carga de datos** desde un archivo CSV (clientes_marketing.csv).
2. **Limpieza básica** (manejo de valores nulos y tipos de datos).
3. **Codificación de variables categóricas** (One-Hot Encoding).
4. **División del conjunto de datos** en entrenamiento (80%) y prueba (20%) usando train_test_split.
5. **Entrenamiento del modelo** RandomForestClassifier con hiperparámetros iniciales.
6. **Predicción** sobre el conjunto de prueba.
7. **Cálculo de métricas**: accuracy, precision, recall, F1-score y matriz de confusión.
8. (Opcional) Uso de **validación cruzada** y **GridSearchCV** para mejorar hiperparámetros.

4.2 Implementación en Python (scikit-learn)

Nota: en el documento final puedes poner este código en formato de código o en el Anexo A.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report


# 1. Cargar datos

datos = pd.read_csv("clientes_marketing.csv")


# 2. Definir variables de entrada (X) y objetivo (y)

X = datos.drop("respondio_campania", axis=1)

y = datos["respondio_campania"]


# 3. Identificar columnas numéricas y categóricas

columnas_numericas = ["edad", "ingreso_mensual", "compras_6m", "monto_anual"]

columnas_categoricas = ["respuesta_anterior", "canal_preferido"]


# 4. Preprocesamiento: One-Hot Encoding para categóricas

preprocesador = ColumnTransformer(

  transformers=[

    ("num", "passthrough", columnas_numericas),

    ("cat", OneHotEncoder(handle_unknown="ignore"), columnas_categoricas)

  ]

)
```

```
# 5. Definir el modelo
modelo_rf = RandomForestClassifier(
   n_estimators=100,
   max_depth=None,
   random_state=42
)

# 6. Construir el pipeline completo
pipeline = Pipeline(steps=[
   ("preprocesamiento", preprocesador),
   ("clasificador", modelo_rf)
])

# 7. División entrenamiento / prueba
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=42, stratify=y
)

# 8. Entrenar el modelo
pipeline.fit(X_train, y_train)

# 9. Realizar predicciones
y_pred = pipeline.predict(X_test)

# 10. Calcular métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
mat_conf = confusion_matrix(y_test, y_pred)
```

```
print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

print("Matriz de confusión:\n", mat_conf)

print("\nReporte de clasificación:\n", classification_report(y_test, y_pred))
```

5. Resultados y evaluación

Dado que este es un caso de estudio, se asume que, tras ejecutar el código anterior con un conjunto de datos realista, se obtuvieron métricas similares a las siguientes:

Métrica	Valor aproximado
Accuracy	0.82
Precision	0.79
Recall	0.76
F1-score	0.77

La **matriz de confusión** (ejemplo) podría ser:

	Predicho 0	Predicho 1
Real 0	210	35
Real 1	30	125

Análisis de resultados

- Una **accuracy de 82 %** indica que la mayoría de las predicciones del modelo son correctas.
- La **precision de 79 %** significa que, de todos los clientes que el modelo etiquetó como “responde a la campaña”, el 79 % realmente respondió. Esto es importante para no saturar de correos promocionales a clientes que probablemente no responden.
- El **recall de 76 %** indica que el modelo detecta correctamente el 76 % de los clientes que realmente responden. Un recall más alto sería deseable, porque significa que se dejan fuera menos clientes potencialmente interesados.
- El **F1-score de 0.77** refleja un balance razonable entre precision y recall.

En general, el modelo Random Forest Classifier presenta un **buen desempeño para una primera versión**, y serviría como base para tomar decisiones de marketing más informadas.

Posibles mejoras

- Aplicar **GridSearchCV** o **RandomizedSearchCV** para optimizar hiperparámetros como n_estimators, max_depth, min_samples_split y min_samples_leaf.
- Utilizar **validación cruzada** (por ejemplo, K-fold) para obtener una estimación más robusta del rendimiento.
- Analizar el **desbalance de clases** (si lo hay) y considerar técnicas como sobremuestreo (SMOTE) o submuestreo.
- Realizar más **ingeniería de características**, por ejemplo, variables de recencia, frecuencia y valor (RFM), días desde la última compra, etc.

6. Conclusiones y recomendaciones

En este trabajo se exploraron diferentes algoritmos de regresión y clasificación, destacando sus objetivos, funcionamiento, métricas habituales, fortalezas y limitaciones.

Los resultados simulados muestran un rendimiento satisfactorio, con valores aceptables de accuracy, precision, recall y F1-score. No obstante, el análisis también revela oportunidades de mejora mediante técnicas como validación cruzada, optimización de hiperparámetros y un tratamiento más detallado de las características y del posible desbalance de clases.

Como recomendación general, antes de adoptar un modelo en producción, es importante:

- Validar su desempeño con distintos subconjuntos de datos.
- Compararlo con otros algoritmos (por ejemplo, regresión logística, SVM, XGBoost).
- Involucrar al área de negocio para interpretar las métricas y ajustar los umbrales de decisión de acuerdo con los objetivos (por ejemplo, maximizar ventas o minimizar costos de campañas).

7. Referencias

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning: With Applications in R* (2nd ed.). Springer.

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

Documentación oficial de scikit-learn. (s.f.). Recuperado de https://scikit-learn.org

Anexos

Anexo A. Código completo del modelo en Python

(Aquí pegas el código de la sección 4.2. Si añadieras GridSearchCV o validación cruzada, lo puedes poner también aquí.)

Anexo B. Tablas y gráficos de métricas

- Tabla de métricas (accuracy, precision, recall, F1) presentada en la sección 5.
- Gráfico sugerido: curva ROC (se puede generar con roc_curve y roc_auc_score de scikit-learn).
- (Si el profesor lo pide, puedes añadir capturas de pantalla del notebook o de la salida de consola.)