UNIVERSIDAD TECNOLÓGICA DE CHIHUAHUA

Ingeniería en Desarrollo y Gestión de Software

Extracción de Conocimientos de Bases de Datos

III.2. Reporte de Métricas de Evaluación

IDGS91N

PRESENTAN:

Giselle Cantú Chávez

NOMBRE DEL DOCENTE:

Ing. Luis Enrique Mascote Cano

Chihuahua, Chih., 29 de noviembre de 2025

Universidad Tecnológica de Chihuahua

Índice

1. Introducción

En esta evidencia trabajo con un escenario típico de aprendizaje supervisado, donde me interesa evaluar el desempeño de modelos de clasificación a partir de métricas cuantitativas y no solo “a ojo”. En particular, uso un clasificador **K-Nearest Neighbors (KNN)** aplicado a una matriz de datos con niveles de glucosa, edad y una etiqueta binaria que indica si la persona presenta o no riesgo asociado.

Antes de pasar a la parte práctica, hago una revisión breve pero clara de varias métricas de evaluación usadas tanto en **clasificación** como en **regresión**, porque en la vida real no existe una única métrica perfecta; cada una resalta un aspecto diferente del modelo.Google for Developers+1

Después, construyo un pequeño flujo de trabajo que incluye la división de datos en entrenamiento y prueba, el preprocesamiento (escalado de las variables), el entrenamiento del modelo KNN con distintos valores de *k* y el cálculo de métricas como **accuracy, precision, recall, F1-score y ROC-AUC**, complementadas con la matriz de confusión y la curva ROC. Con eso cierro con un análisis honesto del rendimiento y algunas ideas de mejora.

2. Investigación de métricas de evaluación

2.1 Métricas para modelos de clasificación

Para clasificación binaria parto de la matriz de confusión, donde se registran verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN). A partir de ahí se definen varias métricas clave.

2.1.1 Exactitud (accuracy)

- **Definición y fórmula**
  La exactitud mide la proporción de predicciones correctas sobre el total de predicciones:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Interpretación práctica**
  Me dice qué fracción de los casos el modelo acierta, sin distinguir entre clases positiva y negativa.

- **Ventajas**

  - Es muy intuitiva y fácil de comunicar.

- **Limitaciones**

  - Puede ser engañosa en conjuntos de datos desbalanceados; un modelo que casi siempre predice la clase mayoritaria puede tener una accuracy alta pero ser inútil para detectar la clase minoritaria.

2.1.2 Precisión (precision)

- **Definición y fórmula**
  La precisión mide de todo lo que el modelo marcó como positivo, qué proporción realmente era positiva:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Interpretación práctica**
  La uso cuando me preocupa **no etiquetar falsos positivos**, por ejemplo, al clasificar correos como spam o al generar alertas de riesgo.

- **Ventajas**

  - Útil cuando el costo de un falso positivo es alto.

- **Limitaciones**
    - Si se optimiza solo precisión se puede sacrificar recall, dejando fuera muchos casos positivos reales.

2.1.3 Exhaustividad / sensibilidad (recall)

- **Definición y fórmula**

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **Interpretación práctica**
  Mide qué tan bien recupero a los positivos reales. En un contexto de salud, un recall alto significa que casi no dejo pasar casos de riesgo.

- **Ventajas**
    - Importante cuando el costo de un falso negativo es crítico (por ejemplo, perder un paciente en riesgo).

- **Limitaciones**
    - Se puede inflar el recall a costa de marcar casi todo como positivo, lo que daña la precisión.

2.1.4 F1-score

- **Definición y fórmula**
  El F1-score es la media armónica entre precisión y recall:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Interpretación práctica**
  Es útil cuando quiero un equilibrio entre precisión y recall y no quiero mirar solo una de las dos. Un F1 cercano a 1 indica que el modelo mantiene buenos valores en ambas métricas.

- **Ventajas**
  - Resume en un solo número el balance entre errores de tipo FP y FN.

- **Limitaciones**
  - No toma en cuenta los verdaderos negativos, así que puede no reflejar toda la historia en problemas muy desbalanceados.

2.1.5 ROC-AUC

- **Definición**
  La curva ROC (Receiver Operating Characteristic) relaciona la tasa de verdaderos positivos (TPR o recall) contra la tasa de falsos positivos (FPR) a distintos umbrales de decisión. El **AUC (Area Under the Curve)** condensa el rendimiento en un solo valor entre 0 y 1.

- **Interpretación práctica**
  Un AUC cercano a 1 indica que el modelo separa muy bien la clase positiva y la negativa en casi todos los umbrales; un valor cercano a 0.5 indica un desempeño similar a adivinar al azar.

- **Ventajas**
  - Es menos sensible al desbalance de clases porque considera toda la curva, no solo un punto.

- **Limitaciones**
  - Puede ser menos intuitiva para alguien que no está acostumbrado a leer curvas ROC.

2.2 Métricas para modelos de regresión

Aunque en esta evidencia trabajo con un problema de clasificación, también reviso dos métricas muy usadas en regresión: **MAE** y **RMSE**.

2.2.1 Error absoluto medio (Mean Absolute Error, MAE)

- **Definición y fórmula**

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} \quad | \, y_i - \hat{y}_i \, |$$

donde $y_i$es el valor real y $\hat{y}_i$es la predicción del modelo.

- **Interpretación práctica**
  Me dice, en promedio, cuánta distancia hay entre mis predicciones y los valores reales, en las **mismas unidades** de la variable objetivo.

- **Ventajas**
  - Es fácil de interpretar y relativamente robusta a valores atípicos.

- **Limitaciones**
  - No penaliza tan fuertemente los errores grandes como lo hace RMSE.

2.2.2 Raíz del error cuadrático medio (Root Mean Squared Error, RMSE)

- **Definición y fórmula**

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \quad (y_i - \hat{y}_i)^2}$$

- **Interpretación práctica**

  También está en las mismas unidades que la variable objetivo, pero al elevar al cuadrado los errores antes de promediarlos y tomar la raíz, los errores grandes pesan más.

- **Ventajas**

  - Es muy usada cuando quiero ser especialmente sensible a errores grandes (por ejemplo, en finanzas o energía).

- **Limitaciones**

  - Es más sensible a outliers, lo cual puede ser un problema si el conjunto de datos tiene valores extremos poco representativos.

3. Solución con KNN (preprocesamiento, entrenamiento y evaluación)

3.1 Descripción del conjunto de datos

Para el caso práctico utilizo la **matriz "Matriz.csv"** que contiene 30 registros con tres columnas:

- glucosa: nivel de glucosa en sangre.
- edad: edad de la persona en años.
- etiqueta: variable binaria (0 = sin riesgo, 1 = con riesgo).

El objetivo es entrenar un modelo KNN que, a partir de glucosa y edad, clasifique a una nueva persona en riesgo o no riesgo.

KNN es un algoritmo de aprendizaje supervisado **basado en vecinos más cercanos**, que clasifica un punto nuevo según la "votación" de los *k* ejemplos más cercanos en el espacio de características.

3.2 Preprocesamiento y partición de datos

Los pasos que sigo son:

1. **Separar características y etiqueta**
    - Variables de entrada: glucosa y edad.
    - Variable objetivo: etiqueta.
2. **Dividir en entrenamiento y prueba**
    - Uso una división **70 % entrenamiento / 30 % prueba**.
    - Mantengo el balance de la etiqueta con un muestreo estratificado para no deformar la proporción de clases.
3. **Escalar las variables**
    - Aplico **StandardScaler** para llevar glucosa y edad a una escala comparable (media 0 y desviación estándar 1).
    - Esto es importante porque KNN se basa en distancias; si una variable tiene valores mucho más grandes que la otra, domina la distancia y sesga el modelo.

3.3 Implementación del clasificador KNN

Para implementar el modelo uso Python con pandas y scikit-learn. El flujo general de código es:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import (accuracy_score, precision_score, recall_score,
```

```
f1_score, roc_auc_score, confusion_matrix, roc_curve)


# 1. Cargar datos
df = pd.read_csv("Matriz.csv")  # columnas: glucosa, edad, etiqueta


X = df[["glucosa", "edad"]]
y = df["etiqueta"]


# 2. División entrenamiento / prueba (70/30)
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y
)


# 3. Escalado
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)


# 4. Entrenar y evaluar varios valores de k
resultados = {}
for k in [3, 5, 7]:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)

    y_pred = knn.predict(X_test_scaled)
    y_proba = knn.predict_proba(X_test_scaled)[:, 1]

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, zero_division=0)
    rec = recall_score(y_test, y_pred, zero_division=0)
    f1 = f1_score(y_test, y_pred, zero_division=0)
    auc = roc_auc_score(y_test, y_proba)
    cm = confusion_matrix(y_test, y_pred)

    resultados[k] = {
```

```
        "accuracy": acc,
        "precision": prec,
        "recall": rec,
        "f1": f1,
        "auc": auc,
        "cm": cm
    }

print(resultados)
```

Con este código pruebo tres vecinos distintos: **k = 3, 5 y 7**, y guardo las métricas de cada uno.

3.4 Selección del valor de *k*

Los resultados aproximados que obtengo son:

k	Accuracy	Precision	Recall	F1-score	ROC-AUC
3	0.778	0.750	0.750	0.750	0.925
5	0.889	1.000	0.750	0.857	0.975
7	0.889	1.000	0.750	0.857	1.000

Tanto **k = 5** como **k = 7** dan el mismo F1-score (≈ 0.857) y la misma accuracy (≈ 0.889), pero el AUC llega hasta 1.0 cuando uso **k = 7**.

Para mantener un equilibrio entre complejidad del modelo y rendimiento, elijo **k = 5** como valor final, por ser un valor intermedio que ya ofrece muy buen desempeño y evita usar un vecindario demasiado grande en un conjunto de datos tan pequeño.

4. Resultados (tablas, matriz de confusión y curva ROC)

Con **k = 5**, la matriz de confusión es:

$$\text{CM} = \begin{bmatrix} 5 & 0 \\ 1 & 3 \end{bmatrix}$$

Esto se interpreta así:

- 5 **verdaderos negativos** (clases 0 clasificadas correctamente).
- 3 **verdaderos positivos** (clases 1 clasificadas correctamente).
- 0 **falsos positivos**.
- 1 **falso negativo**.

Las métricas para este valor de *k* quedan aproximadamente:

- Accuracy ≈ 0.889
- Precision ≈ 1.000
- Recall ≈ 0.750
- F1-score ≈ 0.857

- ROC-AUC ≈ 0.975

La curva ROC asociada a este modelo se acerca bastante a la esquina superior izquierda, lo que refleja una buena capacidad de discriminación entre ambas clases. El valor de AUC cercano a 1 indica que, para casi todos los umbrales posibles, el modelo mantiene una combinación adecuada de TPR alto y FPR bajo.Deepchecks+1

En resumen, el clasificador KNN con *k = 5* casi no comete falsos positivos y solo deja escapar un caso positivo real, lo cual se ve reflejado en la combinación de precisión perfecta (1.0) y un recall aceptable.

5. Conclusiones y recomendaciones

En esta evidencia pude **aterrizar las métricas de evaluación** en un ejemplo concreto con KNN. Primero revisé métricas teóricas para clasificación (accuracy, precision, recall, F1-score, ROC-AUC) y regresión (MAE, RMSE), entendiendo qué mide cada una, cuándo conviene usarla y qué limitaciones tiene.

Después, a partir de un conjunto pequeño con glucosa, edad y una etiqueta binaria, construí un flujo de trabajo sencillo: dividir datos, escalar variables, entrenar KNN con varios valores de *k* y comparar métricas. Eso me permite justificar por qué me quedo con un valor de *k* y no con otro, en lugar de elegirlo al azar.

En este caso, el modelo con **k = 5** ofrece un rendimiento bastante sólido: buena exactitud, F1-score alto y una curva ROC con AUC cercano a 1. Aun así, el hecho de tener solo 30 registros limita la capacidad de generalización; con más datos reales podrían aparecer patrones distintos y el rendimiento del modelo cambiaría.

Como recomendaciones para mejorar este trabajo a futuro, dejaría anotadas estas ideas:

- Probar **validación cruzada** para estimar el rendimiento de manera más estable con tan pocos datos.

- Experimentar con otras métricas cuando el problema tenga clases desbalanceadas (por ejemplo, PR-AUC).

- Comparar KNN contra otros modelos supervisados (logistic regression, árboles de decisión) usando las mismas métricas para decidir cuál se adapta mejor al problema.

- Ajustar hiperparámetros adicionales del KNN, como el tipo de distancia o el peso de los vecinos, no solo el valor de *k*.

6. Fuentes de apoyo

DataSource.ai. (2023, 25 septiembre). *Métricas de evaluación de modelos en el aprendizaje automático*. DataSource.ai. https://www.datasource.ai/es/data-science-articles/metricas-de-evaluacion-de-modelos-en-el-aprendizaje-automatico datasource.ai

Google Developers. (s. f.). *Clasificación: exactitud, recuperación, precisión y métricas relacionadas*. Machine Learning Crash Course. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall Google for Developers

Gupta, P. (2024, 18 noviembre). *Precision and recall in machine learning: A complete guide*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2024/11/precision-and-recall-in-machine-learning-a-complete-guide Analytics Vidhya

Shafi, A. (2023, 20 febrero). *K-Nearest Neighbors (KNN) classification with scikit-learn*. DataCamp. https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn DataCamp

Statology. (2025, 23 enero). *A complete guide to model evaluation metrics in machine learning*. Statology. https://www.statology.org/model-evaluation-metrics-machine-learning Statology

Vega-Huerta, H., Pajuelo-Leon, J., De-la-Cruz-VdV, P., Calderón, D., Maquen-Niño, G. L. E., Rios-Castillo, M. E., Camara-Figueroa, A., Gil-Calvo, R., Guerra-Grados, L., y Benito-Pacheco, O. (2025). K-Nearest Neighbors model to optimize data classification according to the water quality index of the upper basin of the city of Huarmey. *Applied Sciences, 15*(18), 10202. https://doi.org/10.3390/app151810202 MDPI

7. Anexos

Anexo A. Código completo de ejemplo

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import (accuracy_score, precision_score, recall_score,

                             f1_score, roc_auc_score, confusion_matrix,

                             roc_curve)



# Cargar datos

df = pd.read_csv("Matriz.csv")  # glucosa, edad, etiqueta



X = df[["glucosa", "edad"]]

y = df["etiqueta"]



# División 70 % / 30 %

X_train, X_test, y_train, y_test = train_test_split(

    X, y,
```

```
    test_size=0.3,
    random_state=42,
    stratify=y
)

# Escalado
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Entrenar varios k
resultados = {}
for k in [3, 5, 7]:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)

    y_pred = knn.predict(X_test_scaled)
    y_proba = knn.predict_proba(X_test_scaled)[:, 1]
```

```
    acc = accuracy_score(y_test, y_pred)

    prec = precision_score(y_test, y_pred, zero_division=0)

    rec = recall_score(y_test, y_pred, zero_division=0)

    f1 = f1_score(y_test, y_pred, zero_division=0)

    auc = roc_auc_score(y_test, y_proba)

    cm = confusion_matrix(y_test, y_pred)



    resultados[k] = {

        "accuracy": acc,

        "precision": prec,

        "recall": rec,

        "f1": f1,

        "auc": auc,

        "cm": cm

    }


print("Resultados por k:", resultados)
```