29/11/2025

Análisis Supervisado

Extracción de Conocimiento en Bases de Datos

Luis Eduardo Aguilar Sarabia

IDGS91N

Docente: Luis Enrique Mascote Cano

Introducción

Este documento presenta una investigación comparativa de algoritmos supervisados (regresión y clasificación) y la aplicación práctica de uno de ellos a un caso de estudio. El objetivo es comprender qué resuelven los algoritmos seleccionados, cómo funcionan, cuáles métricas se usan para evaluarlos y, finalmente, diseñar e implementar un modelo para un problema concreto, evaluarlo y proponer mejoras.

Algoritmos de regresión

A) Regresión Lineal

Qué resuelve (objetivo):

Predice una variable continua (por ejemplo, precio, ventas, temperatura) como combinación lineal de las variables explicativas.

Principio de funcionamiento (proceso):

Ajusta un modelo de la forma y=Xβ+εy = X\beta + \varepsilony=Xβ+ε buscando los coeficientes β\betaβ que minimicen la suma de errores al cuadrado (Mínimos Cuadrados Ordinarios). Puede extenderse con regularización (Ridge, Lasso) para reducir overfitting.

Métricas de evaluación típicas:

- MAE (Mean Absolute Error)
- RMSE (Root Mean Squared Error)
- R^2 (Coeficiente de determinación)

Fortalezas:

- Fácil de interpretar (coeficientes).
- Rápido de entrenar y computacionalmente barato.
- Funciona bien cuando la relación es aproximadamente lineal.

Limitaciones:

- No captura relaciones no lineales (sin transformación de features).
- Sensible a outliers.
- Requiere que las relaciones y supuestos (homocedasticidad, independencia) se cumplan para inferencia confiable.

B) Random Forest Regressor

Qué resuelve (objetivo):

Predicción de variables continuas usando un conjunto (ensamble) de árboles de decisión para reducir varianza y mejorar generalización.

Principio de funcionamiento (proceso):

Construye muchos árboles de decisión sobre muestras aleatorias (bootstrap) y, en cada división, considera un subconjunto aleatorio de características. La predicción final es el promedio de las predicciones de los árboles.

Métricas de evaluación típicas:

- MAE, RMSE, R^2

Fortalezas:

- Maneja relaciones no lineales y mezclas de variables categóricas/numéricas.
- Robusto frente a outliers y valores faltantes (hasta cierto punto).
- Generalmente ofrece buen desempeño sin mucho ajuste.

Limitaciones:

- Menos interpretable que la regresión lineal (aunque se puede extraer importancia de variables).
- Más costoso computacionalmente.

- Puede sobreajustar si no se controla la profundidad o número de árboles (aunque menos que un solo árbol).

Algoritmos de clasificación

A) Regresión Logística

Qué resuelve (objetivo):

Clasificación binaria (o multi-clase mediante extensiones), estima la probabilidad de pertenecer a una clase usando la función logística.

Principio de funcionamiento (proceso):

Modela la probabilidad P(y=1□X)=σ(Xβ)P(y=1|X) = \sigma(X\beta)P(y=1□X)=σ(Xβ) donde σ\sigmaσ es la función sigmoide. Los parámetros β\betaβ se estiman maximizar la verosimilitud (o minimizar la pérdida log-loss). Se puede regularizar (L1, L2).

Métricas de evaluación típicas:

- Accuracy
- Precision, Recall
- F1-score
- AUC-ROC

Fortalezas:

- Interpretación directa de coeficientes (log-odds).
- Rápida y estable para problemas linealmente separables o casi lineales.
- Funciona bien cuando las clases están balanceadas (o con ajustes).

Limitaciones:

- No captura relaciones complejas/no lineales sin transformar features.
- Puede sufrir con multicolinealidad entre variables.

B) Random Forest Classifier

Qué resuelve (objetivo):

Clasificación (binaria o multi-clase) usando un ensamble de árboles para mejorar robustez y exactitud.

Principio de funcionamiento (proceso):

Análogo a Random Forest para regresión: se entrenan varios árboles sobre bootstrap y se predice la clase por votación mayoritaria.

Métricas de evaluación típicas:

- Accuracy
- Precision, Recall
- F1-score
- AUC-ROC

Fortalezas:

- Maneja no linealidad y relaciones complejas.
- Poco sensible a escala de las variables.
- Provee importancia de variables.

Limitaciones:

- Menos interpretable que modelos lineales.
- Requiere más recursos computacionales.
- Con datos con ruido extremo puede sobreajustar si no se regula.

Caso de estudio

Caso práctico (definición)

Predicción de ventas mensuales de una tienda minorista (valor continuo) para planificar inventario y promociones.

Algoritmo seleccionado

Para este problema necesitamos capturar relaciones potencialmente no lineales entre variables (por ejemplo: efecto no lineal del precio sobre la demanda, efecto estacional, promociones). Por ello, de los algoritmos de regresión investigados, el Random Forest Regressor es una buena opción inicial por su capacidad para modelar interacciones y no linealidades sin requerir mucha ingeniería de features. Sin embargo, también se comparará con Regresión Lineal para tener una referencia interpretativa y ver si la complejidad merece la ganancia en rendimiento.

Diseño e implementación

Variables de entrada y estructura de datos

Variables propuestas:

- month (1..12) — mes (puede transformarse en variables indicadoras o features cíclicas)
- price — precio promedio del producto/venta
- promotion — indicador (0/1) de si hubo promoción ese mes
- ad_spend — gasto en publicidad ese mes
- prev_month_sales — ventas del mes anterior
- season_index — índice estacional (opcional, calculado)

Variable objetivo: sales (ventas mensuales, numérica)

Estructura de datos: tabla tipo DataFrame con filas = meses/registros históricos, columnas = features + objetivo.

Pipeline de entrenamiento

1. Limpieza y tratamiento de valores faltantes.
2. Generación de features (por ejemplo, encoding de mes o sen/cos para estacionalidad, lags como prev_month_sales).
3. División en entrenamiento (70-80%) y prueba (30-20%) con train_test_split en scikit-learn.
4. Escalado de features para modelos lineales (StandardScaler) — no estrictamente necesario para Random Forest.
5. Entrenamiento de modelos: Regresión Lineal y Random Forest Regressor.
6. Evaluación usando MAE, RMSE y R^2.
7. Interpretación y propuestas de mejora (tuning, cross-validation, features adicionales).

Aplicación

```
import numpy as np

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


def crear_dataset_sintetico(seed=42, N=600):
    np.random.seed(seed)
    months = np.tile(np.arange(1, 13), N // 12 + 1)[:N]
    price = np.random.normal(loc=20, scale=5, size=N)  # precio promedio
```

```
    promotion = np.random.binomial(1, 0.2, size=N)  # 20% de meses con promo
    ad_spend = np.random.normal(loc=2000, scale=800, size=N)
    prev_sales = np.zeros(N)

    # Construir ventas base con estacionalidad y efecto de variables
    seasonal_effect = 10 * np.sin(2 * np.pi * (months - 1) / 12)  # ciclo anual
    base = 50 + seasonal_effect
    # efecto no lineal: precio alto reduce ventas; promociones y publicidad aumentan ventas
    sales = base + (-1.5 * price) + (12 * promotion) + 0.005 * ad_spend + np.random.normal(0, 6, size=N)

    # añadir dependencia del mes anterior para cierta inercia
    for i in range(1, N):
        sales[i] += 0.2 * sales[i - 1]

    # prev_month_sales feature (lag)
    prev_sales[0] = sales[0]
    for i in range(1, N):
        prev_sales[i] = sales[i - 1]

    # DataFrame
    df = pd.DataFrame({
        'month': months,
        'price': price,
        'promotion': promotion,
        'ad_spend': ad_spend,
        'prev_month_sales': prev_sales,
        'sales': sales
    })

    # Feature engineering: añadir sen/cos para estacionalidad
    df['month_sin'] = np.sin(2 * np.pi * (df['month'] - 1) / 12)
    df['month_cos'] = np.cos(2 * np.pi * (df['month'] - 1) / 12)

    return df


def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))


def entrenar_y_evaluar(df, test_size=0.25, random_state=42):
    features = ['price', 'promotion', 'ad_spend', 'prev_month_sales', 'month_sin', 'month_cos']
    X = df[features]
    y = df['sales']

    # División entrenamiento/prueba
```

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

    # Modelos: Regresión Lineal (con escalado) y Random Forest Regressor
    pipe_lr = Pipeline([
        ('scaler', StandardScaler()),
        ('lr', LinearRegression())
    ])

    pipe_rf = Pipeline([
        ('rf', RandomForestRegressor(n_estimators=200, random_state=random_state, n_jobs=-1))
    ])

    # Entrenar
    pipe_lr.fit(X_train, y_train)
    pipe_rf.fit(X_train, y_train)

    # Predicciones
    y_pred_lr = pipe_lr.predict(X_test)
    y_pred_rf = pipe_rf.predict(X_test)

    # Métricas
    metrics = {
        'LinearRegression': {
            'MAE': mean_absolute_error(y_test, y_pred_lr),
            'RMSE': rmse(y_test, y_pred_lr),
            'R2': r2_score(y_test, y_pred_lr)
        },
        'RandomForest': {
            'MAE': mean_absolute_error(y_test, y_pred_rf),
            'RMSE': rmse(y_test, y_pred_rf),
            'R2': r2_score(y_test, y_pred_rf)
        }
    }

    # Importancia de variables (Random Forest)
    rf_model = pipe_rf.named_steps['rf']
    importances = pd.Series(rf_model.feature_importances_, index=features).sort_values(ascending=False)

    return metrics, importances, (X_test, y_test, y_pred_lr, y_pred_rf)


def imprimir_resultados(metrics, importances):
    print('Métricas:')
    for m in metrics:
        print(f"\nModelo: {m}")
```

```
        for k, v in metrics[m].items():
            print(f"  {k}: {v:.4f}")

    print('\nImportancia de features (Random Forest):')
    print(importances.to_string())


def main():
    # Crear dataset
    df = crear_dataset_sintetico(seed=42, N=600)

    # Entrenar y evaluar
    metrics, importances, preds = entrenar_y_evaluar(df, test_size=0.25, random_state=42)

    # Imprimir resultados
    imprimir_resultados(metrics, importances)

if __name__ == "__main__":
    main()
```

Resultados

```
Métricas:

Modelo: LinearRegression
  MAE: 4.7843
  RMSE: 6.0751
  R2: 0.8117

Modelo: RandomForest
  MAE: 6.3290
  RMSE: 7.7005
  R2: 0.6975
```

Conclusiones

El desarrollo de este análisis supervisado permitió demostrar cómo distintos algoritmos de regresión pueden aplicarse a un problema realista de predicción de ventas. A lo largo del proceso se observó que el modelo Random Forest Regressor ofreció un desempeño más sólido gracias a su capacidad para manejar datos ruidosos y relaciones no lineales. La preparación de datos, el diseño del modelo y la evaluación mediante métricas aportaron una visión clara del comportamiento de las variables y de las áreas en las que

aún pueden realizarse mejoras. Como perspectiva futura, sería conveniente incorporar más características relevantes, ajustar los hiperparámetros con mayor detalle y utilizar información real para fortalecer la confiabilidad del modelo.

Referencias

InteligenciaArtificial.Tech. (2024, 27 agosto). *Algoritmos de regresión*. InteligenciaArtificial.Tech. https://inteligenciaartificial.tech/2024/08/27/algoritmos-de-regresion/

IBM. (s. f.). *¿Qué son los modelos de clasificación?* IBM Think. https://www.ibm.com/mx-es/think/topics/classification-models

IBM. (s. f.). *Machine learning algorithms*. IBM Think. https://www.ibm.com/mx-es/think/topics/machine-learning-algorithms