Predicción de Enfermedades Cardiovasculares

Víctor Bórquez
vitocode.ds

Introducción

En este tutorial aprenderás a construir un modelo de **Machine Learning** que prediga el riesgo de enfermedades cardiovasculares usando un conjunto real de datos. Esto nos ayudará a entender cómo preparar datos, elegir un modelo adecuado, entrenarlo, evaluarlo y utilizarlo para hacer predicciones.

La predicción temprana de enfermedades cardiovasculares es fundamental para mejorar la prevención y el tratamiento. La **Inteligencia Artificial** y el análisis de datos ofrecen herramientas poderosas para apoyar esta tarea.

Paso 1: Preparación

Para trabajar con los datos y modelos usaremos las librerías `pandas` para manipulación de datos, `scikit-learn` para Machine Learning y `matplotlib` y `seaborn` para visualización.

Instálalas si aún no las tienes con:

```
!pip install pandas scikit-learn matplotlib seaborn
```

Paso 2: Carga y Exploración de Datos

Primero, sube el archivo `cardio_train.csv` y cárgalo en un `DataFrame` de pandas. Explorar los datos nos permite entender qué columnas contiene, qué tipo de datos hay y cómo están distribuidos.

```python
import pandas as pd

data = pd.read_csv("cardio_train.csv", sep=';')
print(data.head())
print(data.info())
print(data['cardio'].value_counts(normalize=True))
```

La columna `cardio` es nuestra variable objetivo: indica si una persona tiene enfermedad cardiovascular (1) o no (0).

Paso 3: Preprocesamiento de Datos

Antes de entrenar el modelo, limpiamos y preparamos los datos.

- Eliminamos la columna `id` porque no aporta información útil para predecir.
- Convertimos la edad de días a años para facilitar la interpretación.
- Eliminamos duplicados o filas con datos faltantes para evitar sesgos o errores.
- Codificamos las variables categóricas (como género, colesterol y glucosa) usando **One-Hot Encoding**, que crea columnas binarias para cada categoría. Esto es necesario porque los modelos matemáticos trabajan con números, no con etiquetas de texto.

```python
# Limpieza
data = data.drop_duplicates()
data = data.dropna()
data.drop(['id'], axis=1, inplace=True)

# Transformacion
data['age'] = (data['age'] / 365).round().astype(int)

# One-Hot Encoding para variables categoricas
data = pd.get_dummies(data,
        columns=['gender', 'cholesterol','gluc'])
```

Paso 4: Visualización de Datos

Visualizamos distribuciones y relaciones para entender mejor el conjunto.

- Histograma de la edad para ver qué rango domina.
- Mapa de calor de correlaciones para detectar variables que están relacionadas con la presencia de enfermedad.

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(data['age'], kde=True)
plt.title('Distribuci n de la Edad')
plt.show()

corr = data.corr()
plt.figure(figsize=(6,6))
sns.heatmap(corr, cmap='coolwarm')
plt.title('Mapa de calor de correlaciones')
plt.show()
```

Paso 5: Selección y Entrenamiento del Modelo

Elegimos la **Regresión Logística** porque es un modelo simple y fácil de entender que nos permite predecir la probabilidad de que un paciente tenga o no una enfermedad. Además, suele funcionar bien cuando queremos separar dos grupos (enfermos y sanos) y es rápido de entrenar.

- Separa tus datos en `X` (características) y `y` (objetivo).
- Normaliza las características para que tengan media 0 y desviación estándar 1, mejorando el entrenamiento.
- Divide en datos de entrenamiento y prueba.
- Entrena el modelo con los datos de entrenamiento.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X = data.drop('cardio', axis=1)
y = data['cardio']

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)
```

Paso 6: Evaluación del Modelo

Evaluamos el desempeño con varias métricas:

- **Accuracy (Exactitud)**: porcentaje de aciertos totales.
- **Precision (Precisión)**: proporción de verdaderos positivos entre las predicciones positivas.
- **Recall (Sensibilidad)**: proporción de verdaderos positivos detectados entre todos los positivos reales.
- **F1-Score**: medida que combina precisión y recall en un solo valor balanceado, destacando el equilibrio entre ambos.
- **Matriz de Confusión**: resumen de aciertos y errores clasificados por clase.

```
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix

y_pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Reporte de Clasificaci n:\n", classification_report(y_test,
    y_pred))
print("Matriz de Confusi n:\n", confusion_matrix(y_test, y_pred))
```

Nota importante: Un alto porcentaje de falsos negativos puede ser peligroso en medicina, porque significa no detectar a quienes realmente están enfermos. Aquí, es crucial balancear precisión y recall según el caso.

¡Felicidades! Has aprendido un flujo completo y fundamentado para construir un modelo de clasificación de machine learning desde datos crudos hasta predicción.

Este tutorial es parte de vitocode.ds por Víctor Bórquez.