Módulo V: Scikit-learn y TensorFlow para Aprendizaje Automático

Introducción a la Programación en Python para Aprendizaje por Refuerzo

Dr. Darío Ezequiel Díaz

Instituto Provincial de Estadística y Censos de Misiones

24 de abril de 2025

Contenido

Contenido

¿Qué es el Aprendizaje Automático?

- Rama de la inteligencia artificial que permite a las computadoras **aprender sin programación explícita**
- Identifica patrones en grandes volúmenes de datos
- Construye modelos matemáticos capaces de:
  - Hacer predicciones
  - Tomar decisiones
  - Adaptarse a nuevos datos
- Combina principios de estadística, matemáticas y ciencias de la computación
- Permite extraer conocimiento y tomar decisiones basadas en datos

Paradigmas de Aprendizaje Automático

Aprendizaje Supervisado

- Datos etiquetados
- Correspondencia entrada-salida
- Ejemplos:
  - Regresión lineal
  - Regresión logística
  - Árboles de decisión
  - Redes neuronales

Aprendizaje No Supervisado

- Datos no etiquetados
- Descubrimiento de patrones
- Ejemplos:
  - Clustering (K-means)
  - Análisis de componentes
  - Detección de anomalías

Aprendizaje por Refuerzo

- Agentes que aprenden acciones
- Maximizan recompensas
- Ejemplos:
  - Q-Learning
  - Deep Q-Networks (DQN)
  - Policy Gradient

Contenido

Importancia del Preprocesamiento

- Fase crítica que impacta directamente en la calidad del modelo
- Transforma datos crudos en formato optimizado
- Facilita la convergencia de algoritmos
- Mejora la precisión de las predicciones
- Reduce la dimensionalidad y ruido

Etapas del preprocesamiento

1. Limpieza: manejo de valores faltantes, outliers
2. Transformación: normalización, estandarización
3. Reducción: selección de características, extracción
4. Codificación: variables categóricas

Normalización y Estandarización

StandardScaler

- Media $\mu = 0$, desviación estándar $\sigma = 1$
- Fórmula:

$$z_i = \frac{x_i - \mu}{\sigma}$$

- Ideal para:
  - SVM
  - Regresión logística
  - Redes neuronales

MinMaxScaler

- Escala a intervalo $[0, 1]$
- Fórmula:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Ideal para:
  - Redes neuronales con funciones sigmoideas
  - Algoritmos que requieren datos no negativos

Consideración importante

La elección entre normalización y estandarización depende del algoritmo y la distribución de los datos

Ejemplo de Normalización en Scikit-learn

```python
from sklearn.preprocessing import
    StandardScaler
import numpy as np

# Datos de ejemplo
data = np.array([[0, 0], [0, 0],
                 [1, 1], [1, 1]])

# Creamos y ajustamos el scaler
scaler = StandardScaler()
scaler.fit(data)

# Transformamos los datos
scaled_data = scaler.transform(data)
```

```python
print("Datos originales:")
print(data)
print("\nDatos escalados:")
print(scaled_data)
print("\nMedia:",
      scaled_data.mean(axis=0))
print("Desviaci n est ndar:",
      scaled_data.std(axis=0))
```

Resultado

Media: [0. 0.]
Desviación estándar: [1. 1.]

Codificación de Variables Categóricas

OneHotEncoder

- Transforma variables categóricas en vectores binarios
- Evita imponer relaciones de orden artificial entre categorías
- Genera una nueva columna para cada categoría única

Color	color_rojo	color_verde	color_azul
rojo	1	0	0
verde	0	1	0
azul	0	0	1
verde	0	1	0

Consideraciones

- Puede incrementar significativamente la dimensionalidad
- Crucial para algoritmos como regresiones y redes neuronales
- Alternativas: Label Encoding (para variables ordinales)

Contenido

Scikit-learn: Biblioteca Fundamental

- Biblioteca Python de código abierto para aprendizaje automático
- Características principales:
  - API consistente y bien documentada
  - Amplia variedad de algoritmos
  - Herramientas de preprocesamiento y evaluación
  - Integración con ecosistema científico de Python
  - Optimizada para rendimiento
- Ideal para investigación y producción
- Incluye implementaciones eficientes de algoritmos clásicos y modernos
- Proporciona herramientas para evaluación y validación de modelos
- Facilita la creación de pipelines completos de procesamiento y modelado

Regresión Lineal

- Modelo fundamental para predecir valores continuos
- Relación lineal entre variables independientes y dependiente
- Ecuación matemática:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_n x_{in} + \varepsilon_i$$

- Estimación por mínimos cuadrados ordinarios (OLS)
- El objetivo es minimizar la suma de los cuadrados de los residuos:

$$\min_{\beta} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \min_{\beta} \sum_{i=1}^{m} (y_i - \beta_0 - \sum_{j=1}^{n} \beta_j x_{ij})^2$$

Ventajas y limitaciones

- **Ventajas:** Interpretabilidad, eficiencia computacional, base para modelos más complejos
- **Limitaciones:** Asume relación lineal, sensible a outliers, no captura interacciones complejas

Ejemplo de Regresión Lineal

```python
from sklearn.linear_model import
    LinearRegression
import numpy as np
import matplotlib.pyplot as plt

# Generamos datos sint ticos
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Creamos y entrenamos el modelo
model = LinearRegression()
model.fit(X, y)
```

```python
# Examinamos los coeficientes
print(f"Intercepto: {model.intercept_
    [0]:.2f}")
print(f"Pendiente: {model.coef_[0, 0]:.2
    f}")

# Visualizamos el resultado
plt.scatter(X, y, alpha=0.6)
plt.plot(X, model.predict(X),
         color='red', linewidth=2)
plt.xlabel('X')
plt.ylabel('y')
plt.title('Regresi n Lineal')
```

Resultado: Intercepto 4.0, Pendiente 3.0 (cercanos a los valores reales)

Regresión Logística

- Modelo para problemas de clasificación binaria
- Predice probabilidad de pertenencia a una clase
- Función logística (sigmoide):

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta^T x)}}$$

- Estimación por máxima verosimilitud
- Función de log-verosimilitud:

$$-\log L(\beta) = -\sum_{i=1}^{m} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

Aplicaciones

- Problemas de clasificación binaria: spam/no spam, fraude/legítimo
- Base para clasificación multiclase (one-vs-rest)
- Casos donde se requiere explicabilidad y probabilidades

Árboles de Decisión

- Modelo de aprendizaje no paramétrico
- Divide recursivamente el espacio de características
- Estructura jerárquica de reglas de decisión
- Métricas de impureza para optimización:
  - **Impureza de Gini** para clasificación:

$$Gini(t) = 1 - \sum_{i=1}^{k} p(i|t)^2$$

  - **Entropía** como medida alternativa:

$$Entropía(t) = -\sum_{i=1}^{k} p(i|t) \log_2 p(i|t)$$

  - **Error Cuadrático Medio** para regresión:

$$MSE(t) = \frac{1}{N_t} \sum_{i \in t} (y_i - \bar{y}_t)^2$$

Contenido

Validación: Estrategias Fundamentales

Train-Test Split

- División de datos en conjuntos de entrenamiento y prueba
- Simula rendimiento en datos no vistos
- Implementación:

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
  02, random_state = 42)
2
```

Validación Cruzada

- División en k subconjuntos (folds)
- Evaluación más robusta
- Implementación:

```
3 scores = cross_val_score(model, X, y, cv = 5)
4 print(
5 "Puntuaciones:", scores) print("Media:", scores.mean())
```

Ventajas de la validación cruzada

- Usa todos los datos para entrenamiento y evaluación
- Proporciona una estimación más robusta del rendimiento
- Reduce la varianza en la evaluación del modelo
- Especialmente útil con conjuntos de datos pequeños

Métricas de Evaluación

Matriz de Confusión

	Pred +	**Pred -**
Real +	VP	FN
Real -	FP	VN

Métricas derivadas:

- Exactitud: $\frac{VP+VN}{VP+FP+VN+FN}$
- Precisión: $\frac{VP}{VP+FP}$
- Sensibilidad: $\frac{VP}{VP+FN}$
- F1-Score: $2 \cdot \frac{Prec \cdot Sens}{Prec+Sens}$

Métricas para problemas específicos

- **Clasificación:**
  - Área bajo la curva ROC (AUC-ROC)
  - Área bajo la curva Precision-Recall
  - Log-loss (pérdida logarítmica)
  - Kappa de Cohen
- **Regresión:**
  - Error cuadrático medio (MSE)
  - Error absoluto medio (MAE)
  - R^2 (coeficiente de determinación)
  - Error absoluto porcentual medio (MAPE)

Selección de métricas

La elección de métricas depende del problema y los objetivos del proyecto

Contenido

TensorFlow: Marco para Aprendizaje Profundo

- Biblioteca de código abierto para aprendizaje automático
- Desarrollada por Google para investigación y producción
- Características principales:
  - Computación mediante grafos de tensores
  - Soporte para GPU/TPU
  - Diferenciación automática
  - Ecosistema completo: TF Extended, TF.js, TF Lite
  - Keras como API de alto nivel integrada
- Arquitectura flexible para despliegue en diversos entornos
- Optimizada para entrenamiento distribuido y paralelo
- Compatible con múltiples lenguajes y plataformas

TensorFlow vs. Scikit-learn

- **TensorFlow:** Modelos complejos, grandes datasets, producción
- **Scikit-learn:** Algoritmos clásicos, prototipado rápido, datasets pequeños-medianos

Fundamentos de Redes Neuronales

- Inspiradas en el sistema nervioso biológico
- Compuestas por neuronas artificiales interconectadas
- Cada neurona: combinación lineal + función no lineal

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f(W^T X + b)$$

- Funciones de activación comunes:
  - ReLU: $f(x) = \max(0, x)$
  - Sigmoide: $f(x) = \frac{1}{1+e^{-x}}$
  - Tanh: $f(x) = \tanh(x)$
  - Softmax (para múltiples clases): $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$
- Organizadas en capas: entrada, ocultas, salida
- El aprendizaje consiste en ajustar los pesos W y sesgos b
- Retropropagación (backpropagation) como mecanismo de aprendizaje

Construyendo una Red Neuronal con Keras

```python
import tensorflow as tf
from tensorflow.keras import layers, models

# Creamos un modelo secuencial
model = models.Sequential([
    layers.Dense(64, activation='relu', input_shape=(10,)),  # Capa de entrada
    layers.Dense(32, activation='relu'),                     # Capa oculta
    layers.Dense(16, activation='relu'),                     # Capa oculta
    layers.Dense(1, activation='sigmoid')                    # Capa de salida
])

# Compilamos el modelo
model.compile(
    optimizer='adam',               # Algoritmo de optimizaci n
    loss='binary_crossentropy',     # Funci n de p rdida
    metrics=['accuracy']            # M trica a monitorear
)
```

Este código crea una red feedforward para clasificación binaria
con 3 capas ocultas y una capa de salida

Entrenamiento de Redes Neuronales

- **Componentes esenciales:**
  - Función de pérdida: mide error del modelo
  - Optimizador: actualiza pesos para minimizar error
  - Regularización: evita sobreajuste
- **Algoritmos de optimización:**
  - SGD (descenso de gradiente estocástico)
  - Adam, RMSprop, Adagrad
- Ecuación de actualización:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t)$$

Hiperparámetros críticos:

- Tasa de aprendizaje
- Tamaño del batch
- Número de épocas
- Arquitectura de red (número de capas y neuronas)
- Inicialización de pesos
- Función de activación
- Tasa de dropout

Técnicas de regularización:

- Dropout
- L1/L2 regularización
- Batch normalization

Redes Convolucionales (CNN)

- Especializadas para datos con estructura reticular (imágenes)
- Principios fundamentales:
  - Conectividad local
  - Pesos compartidos
  - Invarianza a traslaciones
- Capas principales:
  - Convolucional
  - Pooling (max, average)
  - Fully connected

Operación de convolución

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n)$$

donde:

- I - tensor de entrada (imagen)
- K - kernel o filtro
- (i,j) - coordenadas en el mapa de características

Arquitecturas CNN populares:

- LeNet, AlexNet
- VGG, ResNet
- Inception, MobileNet
- EfficientNet

Transfer Learning

- Reutiliza conocimiento de modelos preentrenados
- Especialmente útil cuando:
  - Datos limitados
  - Recursos computacionales reducidos
  - Tareas similares a las originales
- Estrategias comunes:
  - Feature extraction: congelar capas base
  - Fine-tuning: reentrenar parte del modelo

```
1 Congelar capas del modelo base base_modeltrainable = False
2 Aadir nuevas capas para la tarea especfica model = tf.keras.Sequential([ base_model, tf
3     keras.layers.GlobalAveragePooling2D(), tf.keras.layers.Dense(1, activation='sigmoid') ]) .
```

Modelos preentrenados populares

- **Clasificación de imágenes:** VGG, ResNet, MobileNet, EfficientNet
- **NLP:** BERT, GPT, Transformer

Contenido

Aprendizaje por Refuerzo: Fundamentos

- Paradigma basado en interacción agente-entorno
- Optimización de comportamientos secuenciales
- Modelado como Proceso de Decisión de Markov (MDP)
- Componentes clave:
  - Estados (S)
  - Acciones (A)
  - Función de transición (P)
  - Función de recompensa (R)
  - Factor de descuento (γ)
- Objetivo: maximizar recompensa acumulada

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \right]$$

Funciones de valor

- **Función valor-estado:** $V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) | s_0 = s \right]$

Q-Learning

- Algoritmo de aprendizaje por refuerzo libre de modelo
- Estima función de valor acción-estado $Q(s, a)$
- Ecuación de actualización:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

- Política derivada:

$$\pi^*(s) = \arg\max_a Q(s, a)$$

Deep Q-Networks (DQN)

- Extensión que utiliza redes neuronales para aproximar función Q
- Innovaciones clave: Experience replay, Target networks
- Permite aplicar RL a espacios de estados continuos y de alta dimensionalidad

```
4 Mejor valor Q para el siguiente estado next_qmax = npmax(Q[next_state, :])
5 Actualizacin Q-learning Q[state, action] = current_q + alpha * (reward + gamma * next_qmax - current_q)
6
```

Entornos de Simulación: OpenAI Gym

- Interfaz estándar para entornos de RL
- Proporciona variedad de entornos para pruebas:
  - Clásicos (CartPole, MountainCar)
  - Atari
  - Control robótico
  - Físicas realistas
- Interfaz unificada:
  - `env.reset()`: Inicializa episodio
  - `env.step(action)`: Ejecuta acción
  - `env.render()`: Visualiza estado

```
Crear entorno env = gym.make('CartPole-v1')
Inicializar episodio observation, info = env.reset(seed=42)
for ¡nrange(1000) : Elegiraccin(ejemplo : aleatoria)action = env
action₅pacesample()
Ejecutar acci.n en el entorno observation, reward, terminated, truncated, info = env.step(action)
Finalizar episodio si es necesario if terminated or truncated: observation, info = env.reset()
```

Transición a Gymnasium

OpenAI Gym ha evolucionado al proyecto Gymnasium mantenido por Farama Foundation,

Contenido

Integración de Herramientas para Aprendizaje Automático

El ecosistema Python

- Scikit-learn: base sólida para modelos clásicos
- TensorFlow/Keras: arquitecturas avanzadas y deep learning
- OpenAI Gym: entornos para aprendizaje por refuerzo
- Pandas/NumPy: manipulación y procesamiento de datos
- Matplotlib/Seaborn: visualización

Herramientas complementarias

- Pipelines de preprocesamiento
- Selección automatizada de hiperparámetros
- Integración con infraestructura cloud
- Frameworks para despliegue de modelos
- Monitoreo y reentrenamiento automático

Próximos pasos en su formación

- Profundizar en algoritmos específicos de interés
- Desarrollar proyectos prácticos aplicados
- Explorar métodos avanzados de aprendizaje por refuerzo

Referencias Bibliográficas

- Chollet, F. (2021). *Deep Learning with Python* (2ª ed.). Manning Publications.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- McKinney, W. (2022). *Python para análisis de datos: Manipulación de datos con pandas, NumPy y Jupyter* (3ª ed.).
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2ª ed.). MIT Press.
- VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media.
- Documentación oficial:
  - `https://scikit-learn.org/`
  - `https://www.tensorflow.org/`
  - `https://www.gymlibrary.dev/`
  - `https://pandas.pydata.org/`
  - `https://numpy.org/`

¡Gracias por su atención!

¿Preguntas?

Dr. Darío Ezequiel Díaz
drdarioezequieldiaz@gmail.com

Instituto Provincial de Estadística y Censos de Misiones