Guía Completa de Scikit-Learn para Machine Learning

1. Introducción a Scikit-Learn

Scikit-Learn es la biblioteca más popular de Python para Machine Learning. Fue desarrollada en 2007 y se ha convertido en el estándar de la industria para implementar algoritmos de aprendizaje automático de forma práctica y eficiente.

Características principales:

- **Simplicidad y consistencia**: Todos los modelos siguen la misma API (fit, predict, transform)
- **Código abierto**: Completamente gratuito y con una comunidad activa
- **Bien documentada**: Excelente documentación con ejemplos prácticos
- **Integración perfecta**: Funciona de manera natural con NumPy, Pandas y Matplotlib
- **Algoritmos probados**: Implementaciones optimizadas y validadas científicamente

Instalación:

python

```
pip install scikit-learn
```

Importación básica:

python

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

---

2. Arquitectura y API Consistente

Scikit-Learn utiliza una API uniforme para todos sus modelos, lo que facilita enormemente el aprendizaje y uso.

Elementos principales:

Estimadores (Estimators)

Cualquier objeto que puede aprender de los datos.

python

```
from sklearn.linear_model import LinearRegression
modelo = LinearRegression()
modelo.fit(X_train, y_train)  # Entrenar el modelo
```

Predictores (Predictors)

Estimadores que pueden hacer predicciones.

python

```python
# Para regresión y clasificación
y_pred = modelo.predict(X_test)

# Para probabilidades en clasificación
probabilidades = modelo.predict_proba(X_test)
```

Transformadores (Transformers)

Objetos que transforman datos.

python

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)  # Ajustar y transformar
X_test_scaled = scaler.transform(X_test)   # Solo transformer
```

División de Datos: Train, Validation y Test

Una práctica fundamental en Machine Learning es dividir los datos correctamente para evaluar el modelo de forma honesta.

Train-Test Split (División básica)

python

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,      # 20% para test
    random_state=42,    # Semilla para reproducibilidad
    stratify=y          # Mantener proporción de clases
)
```

Parámetros importantes:

- test_size: Proporción para el conjunto de prueba (0.2 = 20%)
- random_state: Fija la semilla aleatoria para reproducibilidad
- stratify: Mantiene la distribución de clases (útil en clasificación desbalanceada)
- shuffle: Por defecto True, mezcla los datos antes de dividir

Train-Validation-Test Split (División en tres conjuntos)

python

```python
# Primero separamos test (20%)
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Luego dividimos el resto en train (64%) y validation (16%)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.2, random_state=42)
```

Uso típico de cada conjunto:

- **Train (64-80%)**: Para entrenar el modelo
- **Validation (10-20%)**: Para ajustar hiperparámetros y seleccionar modelos
- **Test (10-20%)**: Para evaluación final, se usa UNA SOLA VEZ

Validación Cruzada (Cross-Validation)

python

```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(
    modelo, X, y,
    cv=5,            # Número de folds
    scoring='accuracy'
)
print(f"Accuracy promedio: {scores.mean():.4f} ± {scores.std():.4f}")
```

Tipos de validación cruzada:

python

```python
from sklearn.model_selection import KFold, StratifiedKFold, TimeSeriesSplit
# K-Fold estándar (regresión)
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
# Stratified K-Fold (clasificación)
skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
# Time Series Split (datos temporales)
tscv = TimeSeriesSplit(n_splits=5)
```

Comparación de enfoques:

Enfoque	Ventajas	Desventajas	Cuándo usar
Train-Test	Simple, rápido	Puede ser poco representativo	Datasets grandes (>10,000)
Train-Val-Test	Permite ajuste de hiperparámetros	Reduce datos de entrenamiento	Cuando necesitas ajustar modelos
Cross-Validation	Más robusto, usa todos los datos	Más costoso computacionalmente	Datasets pequeños/medianos

Regla de Oro:

El conjunto de test debe ser una "caja cerrada" que solo se abre al final para la evaluación final. Si ajustamos el modelo basándonos en resultados del test, estamos haciendo "data leakage" y sobreestimando el rendimiento.

---

3. Modelos Disponibles en Scikit-Learn

3.1 Aprendizaje Supervisado

Regresión (predicción de valores continuos):

python

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
```

- **Linear Regression**: Regresión lineal simple y múltiple
- **Ridge / Lasso**: Regresión con regularización L2/L1
- **Decision Tree Regressor**: Árboles de decisión para regresión
- **Random Forest Regressor**: Conjunto de árboles de decisión
- **Support Vector Regression (SVR)**: Máquinas de soporte vectorial
- **Gradient Boosting Regressor**: Boosting con árboles
- **K-Nearest Neighbors Regressor**: Vecinos más cercanos

Clasificación (predicción de categorías):

python

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
```

- **Logistic Regression**: Regresión logística
- **Decision Tree Classifier**: Árboles de decisión
- **Random Forest Classifier**: Bosques aleatorios
- **Support Vector Machines (SVM)**: Clasificadores de márgenes máximos
- **Naive Bayes**: Clasificadores bayesianos
- **K-Nearest Neighbors (KNN)**: Clasificación por vecinos
- **Gradient Boosting Classifier**: Métodos de boosting
- **Neural Networks (MLPClassifier)**: Redes neuronales básicas

3.2 Aprendizaje No Supervisado

Clustering (agrupamiento):

python

```
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering, MeanShift
from sklearn.mixture import GaussianMixture
```

- **K-Means**: Agrupamiento por centroides
- **DBSCAN**: Agrupamiento basado en densidad
- **Hierarchical Clustering**: Agrupamiento jerárquico
- **Gaussian Mixture Models**: Modelos de mezcla gaussiana
- **Mean Shift**: Agrupamiento por desplazamiento de media

Reducción de dimensionalidad:

python

```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

- **PCA (Principal Component Analysis)**: Análisis de componentes principales
- **t-SNE**: Visualización de datos de alta dimensión
- **LDA (Linear Discriminant Analysis)**: Análisis discriminante lineal

---

4. Aplicaciones Prácticas de Machine Learning

Sector Financiero:

- Detección de fraudes en transacciones
- Predicción de riesgo crediticio
- Pronóstico de precios de acciones
- Scoring de clientes

Salud:

- Diagnóstico de enfermedades
- Predicción de readmisiones hospitalarias
- Análisis de imágenes médicas
- Personalización de tratamientos

Retail y E-commerce:

- Sistemas de recomendación
- Segmentación de clientes
- Predicción de demanda
- Optimización de precios

Marketing:

- Predicción de churn (abandono de clientes)
- Segmentación de audiencias
- Análisis de sentimientos
- Optimización de campañas

Industria:

- Mantenimiento predictivo
- Control de calidad
- Optimización de procesos
- Detección de anomalías

---

5. Flujo de Trabajo en Machine Learning

Paso 1: Importar bibliotecas y cargar datos

python

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
# Cargar datos
df = pd.read_csv('datos.csv')
# o usar datasets de sklearn
from sklearn.datasets import load_iris
data = load_iris()
```

Paso 2: Exploración y Análisis de Datos (EDA)

python

```
# Exploración básica
print(df.head())
print(df.info())
print(df.describe())
print(df.isnull().sum())
```

Paso 3: Preprocesamiento de Datos

python

```python
# Separar características (X) y objetivo (y)
X = df.drop('target', axis=1)
y = df['target']

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

# Escalar características
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Paso 4: Selección y Entrenamiento del Modelo

python

```python
from sklearn.ensemble import RandomForestClassifier

modelo = RandomForestClassifier(n_estimators=100, random_state=42)
modelo.fit(X_train_scaled, y_train)
```

Paso 5: Evaluación del Modelo

python

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

y_pred = modelo.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
```

Paso 6: Ajuste de Hiperparámetros

python

```python
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15]
}

grid_search = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid, cv=5, scoring='accuracy'
)

grid_search.fit(X_train_scaled, y_train)

print(f'Mejores parámetros: {grid_search.best_params_}')
```

---

6. Técnicas de Preprocesamiento

6.1 Manejo de Valores Faltantes

python

```
from sklearn.impute import SimpleImputer

# Imputar con la media
imputer = SimpleImputer(strategy='mean')
X_imputado = imputer.fit_transform(X)

# Otras estrategias: 'median', 'most_frequent', 'constant'
```

6.2 Codificación de Variables Categóricas

python

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Label Encoding (ordinales)
le = LabelEncoder()
y_encoded = le.fit_transform(y_categorico)

# One-Hot Encoding (nominales)
ohe = OneHotEncoder(sparse_output=False)
X_encoded = ohe.fit_transform(X_categorico)
```

6.3 Escalado de Características

python

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Estandarización (media=0, std=1)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Normalización (rango 0-1)
minmax = MinMaxScaler()
X_normalized = minmax.fit_transform(X)
```

---

7. Métricas de Evaluación

Para Regresión:

python

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

Interpretación:

- **MSE/RMSE**: Error promedio de las predicciones
- **MAE**: Error absoluto medio
- **R^2**: Proporción de varianza explicada (0 a 1, donde 1 es perfecto)

Para Clasificación:

python

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

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
cm = confusion_matrix(y_test, y_pred)
```

Interpretación:

- **Accuracy**: Porcentaje de predicciones correctas
- **Precision**: De las predicciones positivas, cuántas son correctas
- **Recall**: De los casos positivos reales, cuántos fueron detectados
- **F1-Score**: Media armónica entre precision y recall

Para Clustering:

python

```
from sklearn.metrics import silhouette_score
silhouette = silhouette_score(X_scaled, clusters)
```

Interpretación:

- **Silhouette Score**: Mide separación de clusters (-1 a 1)
    - Cerca de 1: Clusters bien definidos
    - Cerca de 0: Clusters superpuestos
    - Negativos: Posible mala asignación

---

8. Mejores Prácticas y Consejos

Antes de entrenar:

1. **Explorar los datos**: Entender distribuciones, valores atípicos, correlaciones
2. **Limpiar los datos**: Manejar valores faltantes y duplicados
3. **Dividir correctamente**: Usar train_test_split antes de cualquier preprocesamiento
4. **Escalar cuando sea necesario**: KNN, SVM, redes neuronales lo requieren

Durante el entrenamiento:

5. **Comenzar simple**: Probar primero modelos simples (regresión lineal, árbol de decisión)
6. **Usar validación cruzada**: No confiar solo en un conjunto de prueba
7. **Monitorear el overfitting**: Comparar métricas de entrenamiento vs prueba

Después del entrenamiento:

8. **Interpretar resultados**: No solo mirar la accuracy, analizar todas las métricas
9. **Probar con datos nuevos**: Validar con datos que el modelo nunca ha visto
10. **Documentar el proceso**: Mantener registro de experimentos y resultados

---

9. Datasets Disponibles en Scikit-Learn

Scikit-Learn incluye datasets listos para usar:

python

```
from sklearn.datasets import (
    load_iris,          # Clasificación de flores (3 clases)
    load_wine,          # Clasificación de vinos (3 clases)
    load_breast_cancer, # Clasificación binaria (cáncer)
    load_diabetes,      # Regresión (progresión diabetes)
    load_digits,        # Clasificación de dígitos (0-9)
    make_classification, # Generar datos sintéticos clasificación
    make_regression,    # Generar datos sintéticos regresión
    make_blobs          # Generar datos sintéticos clustering
)

# Ejemplo de uso
data = load_iris()
X, y = data.data, data.target
```

---

10. Pipelines: Automatización del Flujo

Los pipelines permiten encadenar pasos de preprocesamiento y modelado:

python

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier

# Crear pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=2)),
    ('classifier', RandomForestClassifier())
])

# Entrenar todo el pipeline
pipeline.fit(X_train, y_train)

# Predecir (aplica automáticamente todos los pasos)
y_pred = pipeline.predict(X_test)
```

Ventajas:

- Evita data leakage
- Código más limpio y mantenible
- Facilita el deployment

---

11. Glosario de Términos Clave

- **Feature (Característica)**: Variable de entrada usada para hacer predicciones
- **Target (Objetivo)**: Variable que queremos predecir
- **Training set**: Datos usados para entrenar el modelo
- **Validation set**: Datos para ajustar hiperparámetros
- **Test set**: Datos usados para evaluación final
- **Overfitting**: El modelo memoriza training set pero falla en datos nuevos
- **Underfitting**: El modelo es demasiado simple y no captura patrones
- **Hiperparámetros**: Parámetros configurados antes del entrenamiento
- **Cross-validation**: Evaluación usando múltiples particiones de datos
- **Pipeline**: Secuencia automatizada de preprocesamiento y modelado
- **Feature Engineering**: Creación de nuevas características a partir de las existentes
- **Regularización**: Técnica para prevenir overfitting (L1, L2)
- **Ensemble**: Combinación de múltiples modelos para mejorar predicciones

---

12. Recursos Adicionales

Documentación oficial:

- **Sitio web**: https://scikit-learn.org
- **Guía de usuario**: https://scikit-learn.org/stable/user_guide.html
- **Galería de ejemplos**: https://scikit-learn.org/stable/auto_examples/index.html
- **API Reference**: https://scikit-learn.org/stable/modules/classes.html

Para seguir aprendiendo:

- **Curso online**: "Machine Learning" de Andrew Ng en Coursera
- **Libro**: "Hands-On Machine Learning" de Aurélien Géron
- **Práctica**: Kaggle.com para competencias y datasets
- **Documentación en español**: https://scikit-learn.org/stable/

Comunidad:

- **Stack Overflow**: Para preguntas técnicas específicas
- **GitHub de Scikit-Learn**: Para reportar bugs o contribuir
- **Reddit r/MachineLearning**: Para discusiones y novedades
- **Discord/Slack**: Comunidades de ML en español

Conclusión

Scikit-Learn es una herramienta poderosa y accesible para implementar Machine Learning. Su API consistente, amplia documentación y variedad de algoritmos la convierten en la elección ideal tanto para principiantes como para profesionales.

Lo más importante es **practicar con datos reales** y entender no solo cómo usar los algoritmos, sino **cuándo y por qué** usar cada uno. Machine Learning es tanto ciencia como arte, y la experiencia práctica es fundamental.

Recuerda:

- Comienza con modelos simples
- Valida correctamente tus modelos
- Interpreta los resultados en contexto
- Documenta tu proceso
- Nunca dejes de aprender

¡Manos a la obra y a experimentar!

Documento preparado para curso de Machine Learning - Scikit-Learn

Versión para estudiantes de LATAM – 2025

Guayerd

LuisLQ & Claude IA