Actividad 2

Introducción Scikit-learn

TIC

Universidad Tecnológica de Pereira

faceIT

Introducción Scikit-learn

Scikit-learn es una biblioteca de aprendizaje automático de código abierto para Python que ofrece una amplia variedad de algoritmos de aprendizaje supervisado y no supervisado, así como herramientas para preprocesamiento de datos, selección de modelos, evaluación de modelos y mucho más. Es una de las bibliotecas más utilizadas y respetadas en el campo del aprendizaje automático debido a su simplicidad, e cacia y versatilidad.

La importancia deScikit-learn en IA radica en varios aspectos

Scikit-learn es una herramienta fundamental en el campo de la inteligencia arti cial debido a su facilidad de uso, variedad de algoritmos, e ciencia y rendimiento, integración con Python y una comunidad activa de usuarios y desarrolladores. Ya sea para tareas de clasi cación, regresión, agrupamiento u otras, Scikit-learn proporciona las herramientas necesarias para construir modelos de aprendizaje automático efectivos y escalables.

Comunidad Activa

Scikit-learn cuenta con una gran comunidad de usuarios y desarrolladores que contribuyen con nuevas características, correcciones de errores y mejoras continuas. Esto asegura que la biblioteca esté en constante evolución y actualizada con los últimos avances en el campo del aprendizaje automático.

Integración con Python

Al estar integrado con el ecosistema de Python, Scikit- learn se integra fácilmente con otras bibliotecas populares como NumPy, Pandas y Matplotlib. Esto facilita la manipulación de datos, la visualización de resultados y la construcción de ujos de trabajo completos en Python para tareas de aprendizaje automático.

Facilidad de uso

Scikit-learn está diseñado para ser fácil de aprender y usar, lo que lo hace accesible tanto para principiantes como para expertos en aprendizaje automático. Su API coherente y bien diseñada facilita el proceso de desarrollo de modelos, lo que permite a los usuarios concentrarse en el problema en cuestión en lugar de preocuparse por detalles de implementación complejos.

Eficiencia y Rendimiento

Scikit-learn está construido sobre bibliotecas de computación numérica e cientes como NumPy y SciPy, lo que garantiza un rendimiento óptimo incluso en conjuntos de datos grandes y complejos. Además, muchos de sus algoritmos están implementados en código C o Cython para garantizar una ejecución rápida.

Variedad de Algoritmos

Scikit-learn ofrece una amplia gama de algoritmos de aprendizaje automático, incluidos clasi cación, regresión, agrupamiento, reducción de dimensionalidad, selección de características y más. Esto permite a los usuarios experimentar con diferentes enfoques y encontrar el mejor modelo para su conjunto de datos y problema especí co.

Cómo empezar a trabajar con Scikit-learn

Instalación: **Asegúrate de tener Python instalado en tu sistema. Luego, puedes instalar Scikit-learn y sus dependencias utilizando pip, el administrador de paquetes de Python. Abre tu terminal o símbolo del sistema y ejecuta el siguiente comando:**

```
pip install scikit-learn
```

Esto instalará la última versión de Scikit-learn en tu sistema.

Cómo importar Scikit-learn

Una vez instalado, puedes importar Scikit-learn en tus scripts de Python utilizando la siguiente línea de código: Python

```
import sklearn
```

Esto permitirá acceder a todas las funciones y herramientas proporcionadas por Scikit-learn en tu código.

Explorar la Documentación y Ejemplos

La documentación o cial de Scikit-learn es una excelente fuente de información para aprender sobre las diferentes funcionalidades y técnicas disponibles en la biblioteca. Puedes acceder a la documentación en línea en el sitio web de Scikit-learn:

https://scikit-learn.org/stable/documentation.html

Además, puedes encontrar una variedad de ejemplos y tutoriales en línea que te ayudarán a comprender cómo utilizar Scikit-learn en diferentes contextos y aplicaciones.

Scikit-learn en el aprendizaje supervisado

Clasi cación: **Scikit-learn ofrece una variedad de algoritmos de clasi cación para predecir la clase o categoría a la que pertenecen las muestras. Algunos de los algoritmos de clasi cación disponibles incluyen:**

Vecinos más cercanos (KNeighborsClassi er)
Máquinas de vectores de soporte (SVC)
Árboles de decisión (DecisionTreeClassi er)
Bosques aleatorios (RandomForestClassi er)
Regresión logística (LogisticRegression)

Empezar con Ejemplos Simples: **Comenzar a experimentar con ejemplos simples de uso de Scikit-learn es una excelente manera de familiarizarse con la biblioteca y comprender cómo utilizarla en la práctica.**

Clasificación con el Conjunto de Datos Iris

El conjunto de datos Iris es un conjunto de datos clásico en el aprendizaje automático que contiene medidas de longitud y anchura de sépalos y pétalos de tres especies de iris.

Ejemplo de clasi cación utilizando el algoritmo de clasi cación de vecinos más cercanos (KNeighborsClassi er):

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Cargar el conjunto de datos Iris
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
                                                    iris.target,
                                                    test_size=0.2,
                                                    random_state=42)

# Crear el clasificador de vecinos más cercanos
clf = KNeighborsClassifier(n_neighbors=3)

# Entrenar el clasificador
clf.fit(X_train, y_train)

# Predecir las etiquetas para los datos de prueba
y_pred = clf.predict(X_test)

# Calcular la precisión del clasificador
accuracy = accuracy_score(y_test, y_pred)
print('Precisión del clasificador:', accuracy)
```

```
Precisión del clasificador: 1.0
```

Clasificación con el Conjunto de Datos Iris

Regresión: **Para problemas de regresión, Scikit-learn proporciona varios algoritmos que pueden predecir valores numéricos en función de variables de entrada. Algunos de los algoritmos de regresión disponibles incluyen:**

Regresión lineal (LinearRegression)
Regresión Ridge (Ridge)
Regresión Lasso (Lasso)
Máquinas de vectores de soporte para regresión (SVR)
Bosques aleatorios para regresión (RandomForestRegressor)

Empezar con Ejemplos Simples: **Comenzar a experimentar con ejemplos simples de uso de Scikit-learn es una excelente manera de familiarizarse con la biblioteca y comprender cómo utilizarla en la práctica.**

Regresión con el Conjunto de Datos California Housing

El conjunto de datos CaliforniaHousing contiene información sobre precios de viviendas en California basada en diferentes características. Aquí tienes un ejemplo de regresión utilizando el algoritmo de regresión lineal:

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Cargar el conjunto de datos California Housing
California = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(California.data,
                                                    California.target,
                                                    test_size=0.2,
                                                    random_state=42)

# Crear el modelo de regresión lineal
model = LinearRegression()

# Entrenar el modelo
model.fit(X_train, y_train)

# Predecir los precios de las viviendas para los datos de prueba
y_pred = model.predict(X_test)

# Calcular el error cuadrático medio
mse = mean_squared_error(y_test, y_pred)
print('Error cuadrático medio:', mse)
```

```
Error cuadrático medio: 0.5558915986952422
```

Validación del Modelo

La validación del modelo es una parte crucial del desarrollo de modelos de aprendizaje automático para asegurarse de que funcionen bien en datos nuevos y no vistos. Scikit-learn proporciona varias herramientas y funciones para ayudar en este proceso.

División de Conjuntos de Datos:

La función train_test_split de Scikit-learn se utiliza para dividir un conjunto de datos en conjuntos de entrenamiento y prueba. Esto permite evaluar el rendimiento del modelo en datos no vistos. Se puede especi car la proporción de los conjuntos de entrenamiento y prueba mediante el parámetro test_size.

Ejemplo de cómo dividir conjuntos de datos usando train_test_split:

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Cargar el conjunto de datos Iris
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
                                                    iris.target,
                                                    test_size=0.2,
                                                    random_state=42)

print("Número de muestras en el conjunto de entrenamiento:", len(X_train))
print("Número de muestras en el conjunto de prueba:", len(X_test))
```

```
Número de muestras en el conjunto de entrenamiento: 120
Número de muestras en el conjunto de prueba: 30
```

Validación del Modelo

Validación Cruzada: **La validación cruzada es una técnica para evaluar el rendimiento del modelo utilizando múltiples divisiones de los datos en conjuntos de entrenamiento y prueba. Scikit-learn proporciona la función cross_val_score para realizar validación cruzada y calcular métricas de evaluación del modelo en cada división. Ejemplo de cómo realizar validación cruzada usando cross_val_score:**

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

# Crear un clasificador de regresión logística
clf = LogisticRegression()

# Realizar validación cruzada
scores = cross_val_score(clf, iris.data, iris.target, cv=5)

print("Precisión de validación cruzada:", scores)
```

```
Precisión de validación cruzada: [0.96666667 1.         0.93333333 0.96666667 1.        ]
```

Validación del Modelo

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Cargar el conjunto de datos Iris
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
                                                    iris.target,
                                                    test_size=0.2,
                                                    random_state=42)

# Crear el clasificador de vecinos más cercanos
clf = KNeighborsClassifier(n_neighbors=3)
# Entrenar el clasificador
clf.fit(X_train, y_train)
# Predecir las etiquetas para los datos de prueba
y_pred = clf.predict(X_test)
# Calcular la precisión del clasificador
accuracy = accuracy_score(y_test, y_pred)
print('Precisión del clasificador:', accuracy)
# Predicciones del modelo
y_pred = clf.predict(X_test)
# Calcular métricas de evaluación del modelo
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')
print("Precisión:", accuracy)
print("Precisión promedio ponderada:", precision)
print("Recall promedio ponderado:", recall)
print("F1-score promedio ponderado:", f1)
```

```
Precisión del clasificador: 1.0
Precisión: 1.0
Precisión promedio ponderada: 1.0
Recall promedio ponderado: 1.0
F1-score promedio ponderado: 1.0
```

Métricas de Evaluación del Modelo: **Scikit-learn proporciona una variedad de métricas para evaluar el rendimiento de los modelos de aprendizaje supervisado, como precisión, recall, F1- score, AUC-ROC, entre otras. Estas métricas se pueden calcular utilizando funciones especí cas de Scikit-learn. Ejemplo de cómo calcular métricas de evaluación del modelo:**

+Info

Estos ejemplos ilustran cómo utilizar algunas de las herramientas de validación del modelo proporcionadas por Scikit-learn. Es importante elegir la técnica de validación adecuada y calcular las métricas de evaluación relevantes para comprender cómo se desempeña un modelo en datos no vistos y tomar decisiones informadas sobre su capacidad predictiva.

Preprocesamiento de Datos

El preprocesamiento de datos es una parte crucial en el proceso de construcción de modelos de aprendizaje automático. Scikit-learn ofrece varias herramientas para realizar estas tareas.

Escalar características (Feature Scaling):

El escalado de características es una técnica común para estandarizar o normalizar las características de un conjunto de datos, lo que ayuda a que los algoritmos de aprendizaje automático funcionen mejor. Scikit-learn proporciona diferentes métodos para escalar características, como StandardScaler y MinMaxScaler. Ejemplo de cómo escalar características usando StandardScaler:

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

# Datos de ejemplo
X = np.array([[1, 2], [3, 4], [5, 6]])

# Crear el objeto StandardScaler
scaler = StandardScaler()

# Ajustar el scaler a los datos y transformarlos
X_scaled = scaler.fit_transform(X)

print("Datos originales:\n", X)
print("Datos escalados:\n", X_scaled)
```

Preprocesamiento de Datos

Codificar variables categóricas (One-Hot Encoding): **Las variables categóricas deben ser convertidas en variables numéricas antes de aplicar algoritmos de aprendizaje automático. One-Hot Encoding es una técnica común para convertir variables categóricas en representaciones numéricas. Scikit-learn proporciona la clase OneHotEncoder para realizar esta tarea. Ejemplo de cómo codi car variables categóricas usando OneHotEncoder:**

```python
from sklearn.preprocessing import OneHotEncoder

# Datos de ejemplo
X = [['Male'], ['Female'], ['Male'], ['Female']]

# Crear el objeto OneHotEncoder
encoder = OneHotEncoder()

# Ajustar el encoder a los datos y transformarlos
X_encoded = encoder.fit_transform(X).toarray()

print("Datos originales:\n", X)
print("Datos codificados:\n", X_encoded)
```

Preprocesamiento de Datos

Imputar valores faltantes (Imputation):
Los conjuntos de datos a menudo contienen valores faltantes, lo que puede afectar el rendimiento de los modelos de aprendizaje automático. Scikit-learn proporciona la clase SimpleImputer para imputar valores faltantes utilizando diferentes estrategias, como la media, la mediana o la moda. Ejemplo de cómo imputar valores faltantes usando SimpleImputer:

```python
from sklearn.impute import SimpleImputer

# Datos de ejemplo con valores faltantes
X = [[1, 2], [np.nan, 3], [7, np.nan], [4, 5]]

# Crear el objeto SimpleImputer
imputer = SimpleImputer(strategy='mean')

# Ajustar el imputer a los datos y transformarlos
X_imputed = imputer.fit_transform(X)

print("Datos originales:\n", X)
print("Datos imputados:\n", X_imputed)
```

Preprocesamiento de Datos

Selección de Características: **Para seleccionar las características más importantes para un modelo, Scikit-learn ofrece métodos de selección de características como SelectKBest, SelectFromModel, Recursive Feature Elimination (RFE), y más. Estas herramientas te permiten reducir la dimensionalidad de tus datos y mejorar el rendimiento de tus modelos.**

Explorar la Comunidad y Recursos Adicionales

Únete a la comunidad de Scikit-learn en línea para obtener ayuda, hacer preguntas y compartir tus conocimientos con otros usuarios. También puedes explorar recursos adicionales, como blogs, libros y cursos en línea, que te ayudarán a profundizar en el uso de Scikit-learn y expandir tus habilidades en aprendizaje automático.

Siguiendo estos pasos, podrás empezar a trabajar con Scikit-learn y aprovechar todas las capacidades que ofrece para construir modelos de aprendizaje automático efectivos y escalables en Python. ¡Buena suerte en tu viaje de aprendizaje con Scikit-learn!

Ejercicios con Scikit-learn

Ejercicios relacionados con Scikit-learn (sklearn) que pueden ayudar a practicar y fortalecer tus habilidades en aprendizaje automático con Python:

1 Clasi cación con KNeighborsClassi er:

- Carga el conjunto de datos Iris utilizando load_iris de
- sklearn.datasets. Divide el conjunto de datos en conjuntos de entrenamiento y prueba utilizando train_test_split de
- sklearn.model_selection. Crea un clasi cador de vecinos más cercanos (KNeighborsClassi er) y ajústalo a los datos de
- entrenamiento. Evalúa el rendimiento del clasi cador utilizando métricas como precisión, recall y F1-score.

Ejercicios con Scikit-learn

Ejercicios relacionados con Scikit-learn (sklearn) que pueden ayudar a practicar y fortalecer tus habilidades en aprendizaje automático con Python:

2 Regresión con LinearRegression:

- Carga el conjunto de datos de CaliforniaHousing utilizando load_california de sklearn.datasets. Divide el conjunto de
- datos en conjuntos de entrenamiento y prueba utilizando train_test_split. Crea un modelo de regresión lineal
- (LinearRegression) y ajústalo a los datos de entrenamiento. Evalúa el rendimiento del modelo utilizando
- métricas como el error cuadrático medio (MSE) y el coe ciente de determinación (R^2).

Ejercicios con Scikit-learn

Ejercicios relacionados con Scikit-learn (sklearn) que pueden ayudar a practicar y fortalecer tus habilidades en aprendizaje automático con Python:

3 Validación Cruzada con cross_val_score:

- Utiliza la función cross_val_score de sklearn.model_selection para realizar validación cruzada en
- un conjunto de datos. Prueba diferentes algoritmos de aprendizaje automático (por ejemplo, SVM, árboles de decisión, regresión logística) con validación cruzada y compara sus resultados.