Módulo 4: Aprendizaje automático con Python (3 horas)

Introducción al aprendizaje automático y sus principales conceptos

El aprendizaje automático es una rama de la inteligencia artificial que se enfoca en desarrollar algoritmos y modelos capaces de aprender de los datos. Los principales conceptos en el aprendizaje automático incluyen:

- **Aprendizaje supervisado**: Se proporcionan ejemplos etiquetados (entrada y salida conocida) al algoritmo para aprender a predecir resultados.
- **Aprendizaje no supervisado**: El algoritmo aprende patrones y relaciones en los datos sin contar con ejemplos etiquetados.
- **Aprendizaje por refuerzo**: El algoritmo aprende a tomar decisiones a través de la interacción con su entorno y la retroalimentación que recibe.

Implementación de modelos de clasificación y regresión con Scikit-learn

Scikit-learn es una biblioteca popular de Python para aprendizaje automático que proporciona una amplia gama de algoritmos de clasificación y regresión. A continuación, se muestra un ejemplo de cómo entrenar y utilizar un modelo de clasificación y uno de regresión utilizando Scikit-learn.

Clasificación

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

# Cargar y dividir los datos
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)

# Escalar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Entrenar el modelo de clasificación
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train_scaled, y_train)

```

```
# Realizar predicciones
y_pred = classifier.predict(X_test_scaled)

# Evaluar la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print("Precisión del modelo:", accuracy)
```

Regresión

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

# Cargar y dividir los datos
boston = load_boston()
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target,
test_size=0.2, random_state=42)

# Entrenar el modelo de regresión
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Realizar predicciones
y_pred = regressor.predict(X_test)

# Evaluar el error cuadrático medio del modelo
mse = mean_squared_error(y_test, y_pred)
print("Error cuadrático medio:", mse)
```

Evaluación de modelos de aprendizaje automático y selección de hiperparámetros

La evaluación de modelos y la selección de hiperparámetros son pasos críticos en el proceso de aprendizaje automático. Scikit-learn proporciona herramientas como la validación cruzada y GridSearchCV para realizar estas tareas de manera eficiente.

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

# Cargar el conjunto de datos iris
iris = load_iris()
X = iris.data
y = iris.target

# Dividir los datos en conjuntos de entrenamiento y prueba
```

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

# Crear el modelo de clasificación
classifier = KNeighborsClassifier()

# Definir el espacio de búsqueda de hiperparámetros
param_grid = {'n_neighbors': [1, 3, 5, 7, 9, 11]}

# Realizar la búsqueda de hiperparámetros usando GridSearchCV
grid_search = GridSearchCV(classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Obtener el mejor modelo y sus hiperparámetros
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
print("Mejores hiperparámetros:", best_params)

# Evaluar la precisión del mejor modelo en el conjunto de prueba
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Precisión del mejor modelo:", accuracy)
```

Este bloque de código carga el conjunto de datos iris, crea un modelo de clasificación KNeighborsClassifier y utiliza GridSearchCV para buscar el mejor valor de hiperparámetro 'n_neighbors'. Luego, evalúa la precisión del mejor modelo en el conjunto de prueba.

Árboles de decisión y bosques aleatorios

Los árboles de decisión y los bosques aleatorios son algoritmos de aprendizaje automático populares que pueden abordar problemas de clasificación y regresión. A continuación, se muestra cómo entrenar un árbol de decisión y un bosque aleatorio utilizando Scikit-learn.

Árbol de decisión

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Cargar y dividir los datos
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)

# Entrenar el árbol de decisión
tree_classifier = DecisionTreeClassifier(max_depth=3)
tree_classifier.fit(X_train, y_train)

# Realizar predicciones
y_pred = tree_classifier.predict(X_test)
```

```
16
17 # Evaluar la precisión del modelo
18 accuracy = accuracy_score(y_test, y_pred)
19 print("Precisión del árbol de decisión:", accuracy)
```

Bosque aleatorio

```
1  from sklearn.datasets import load_iris
2  from sklearn.model_selection import train_test_split
3  from sklearn.ensemble import RandomForestClassifier
4  from sklearn.metrics import accuracy_score
5
6  # Cargar y dividir los datos
7  iris = load_iris()
8  X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
   test_size=0.2, random_state=42)
9
10 # Entrenar el bosque aleatorio
11 forest_classifier = RandomForestClassifier(n_estimators=100, max_depth=3)
12 forest_classifier.fit(X_train, y_train)
13
14 # Realizar predicciones
15 y_pred = forest_classifier.predict(X_test)
16
17 # Evaluar la precisión del modelo
18 accuracy = accuracy_score(y_test, y_pred)
19 print("Precisión del bosque aleatorio:", accuracy)
```

Modelos de aprendizaje profundo con Keras

Keras es una biblioteca de Python para desarrollar y entrenar modelos de aprendizaje profundo. Es especialmente útil para redes neuronales y se ejecuta sobre TensorFlow. A continuación, se muestra un ejemplo de cómo construir y entrenar una red neuronal simple utilizando Keras para un problema de clasificación.

```
1  import numpy as np
2  from sklearn.datasets import load_iris
3  from sklearn.model_selection import train_test_split
4  from sklearn.preprocessing import StandardScaler
5  from sklearn.metrics import accuracy_score
6  from tensorflow.keras.models import Sequential
7  from tensorflow.keras.layers import Dense
8  from tensorflow.keras.utils import to_categorical
9
10 # Cargar y dividir los datos
11 iris = load_iris()
12 X = iris.data
13 y = to_categorical(iris.target)
14 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
   random_state=42)
```

```
15
16 # Escalar los datos
17 scaler = StandardScaler()
18 X_train_scaled = scaler.fit_transform(X_train)
19 X_test_scaled = scaler.transform(X_test)
20
21 # Construir la red neuronal
22 model = Sequential()
23 model.add(Dense(10, input_dim=4, activation='relu'))
24 model.add(Dense(3, activation='softmax'))
25 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
   ['accuracy'])
26
27 # Entrenar el modelo
28 model.fit(X_train_scaled, y_train, epochs=150, batch_size=10, verbose=0)
29
30 # Realizar predicciones
31 y_pred = model.predict_classes(X_test_scaled)
32
33 # Convertir las etiquetas de "one-hot" a clases numéricas
34 y_test_classes = np.argmax(y_test, axis=1)
35
36 # Evaluar la precisión del modelo
37 accuracy = accuracy_score(y_test_classes, y_pred)
38 print("Precisión del modelo de aprendizaje profundo:", accuracy)
```

Este bloque de código entrena y evalúa un modelo de aprendizaje profundo utilizando Keras. Primero, carga el conjunto de datos iris y divide los datos en conjuntos de entrenamiento y prueba. Luego, crea una red neuronal con una capa oculta y entrena el modelo. Finalmente, realiza predicciones en el conjunto de prueba y evalúa la precisión del modelo.

Conclusión

En este curso de Python para la Ciencia de Datos, hemos cubierto diferentes aspectos del análisis y procesamiento de datos, desde la manipulación y visualización de datos hasta el aprendizaje automático y el aprendizaje profundo. Hemos utilizado bibliotecas populares de Python como NumPy, Pandas, Matplotlib, Seaborn, Scikit-learn y Keras para implementar y evaluar diferentes técnicas y modelos.

Al completar este curso, los participantes deberían estar familiarizados con las siguientes habilidades y conceptos:

- Manipulación y análisis de datos con Python usando NumPy y Pandas.
- Visualización de datos y creación de gráficos personalizados con Matplotlib y Seaborn.
- Implementación de modelos de aprendizaje automático, como regresión lineal, árboles de decisión, bosques aleatorios y vecinos más cercanos con Scikit-learn.
- Entrenamiento y evaluación de modelos de aprendizaje profundo, como redes neuronales, utilizando Keras y TensorFlow.

Con estas habilidades y conceptos, los participantes estarán bien equipados para abordar problemas de ciencia de datos y desarrollar soluciones sólidas utilizando Python en su trabajo diario.

No olvidemos que la práctica hace al maestro. Para seguir mejorando y profundizando en el conocimiento adquirido en este curso, es importante seguir trabajando en proyectos reales, así como explorar nuevas técnicas y bibliotecas a medida que surjan.

¡Buena suerte en su viaje en la ciencia de datos y en el uso de Python para resolver problemas del mundo real!