COLOMBIA POTENCIA DE LA VIDA

TIC

Modelos de redes neuronales en scikit-learn

TALENTO TECH | AZ PROYECTOS EDUCATIVOS | UTP Universidad Tecnológica de Pereira

Modelos de redes neuronales en scikit-learn

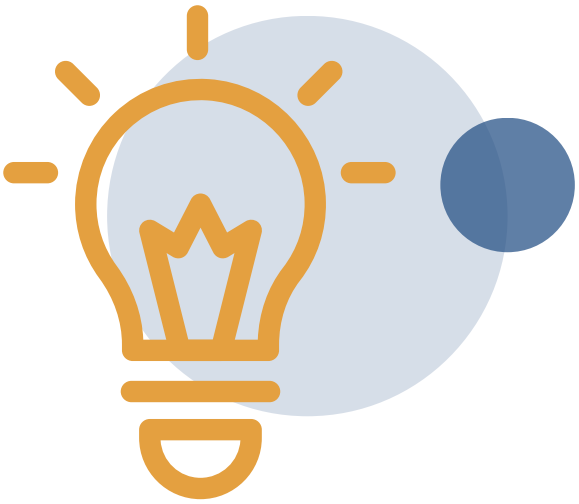
Para ver la documentación oficial de Sklearn
https://scikit-learn.org/stable/modules/neural_networks_supervised.html

En la práctica 2, Programación de Redes Neuronales del módulo 1, se exploraron diversas herramientas y frameworks para la implementación de redes neuronales, entre ellas TensorFlow, Keras y Sklearn. La experiencia de implementación con Sklearn se destacó por su simplicidad y facilidad de uso, ofreciendo una solución directa y accesible para aquellos que están dando sus primeros pasos en el campo del aprendizaje profundo.

Con Sklearn, los participantes pudieron apreciar cómo la biblioteca facilita la creación y entrenamiento de modelos de redes neuronales con una sintaxis clara y concisa, lo que resulta especialmente beneficioso para tareas de clasificación y regresión. Esta experiencia práctica brindó una visión valiosa sobre las diferentes herramientas disponibles y sus respectivas fortalezas, preparando a los participantes para futuras exploraciones en el fascinante mundo del aprendizaje profundo.

Es importante destacar que, si bien los modelos de redes neuronales de Sklearn son una excelente opción para comenzar y comprender los fundamentos del aprendizaje profundo, tienen ciertas limitaciones en cuanto a su escalabilidad y rendimiento en comparación con otras bibliotecas diseñadas específicamente para aplicaciones a gran escala.

Una de las limitaciones más destacadas es la falta de soporte para el uso de unidades de procesamiento gráfico (GPU), lo que puede ralentizar significativamente el entrenamiento y la inferencia de modelos en conjuntos de datos grandes.



Esta limitación puede ser un factor decisivo al considerar la implementación de soluciones en entornos donde el tiempo de procesamiento es crítico, como en la investigación a gran escala o en aplicaciones de producción en tiempo real. Sin embargo, a pesar de estas limitaciones, Sklearn sigue siendo una opción sólida para el desarrollo rápido de prototipos y la experimentación con modelos de redes neuronales en conjuntos de datos más pequeños o en escenarios donde el rendimiento no es una preocupación primordial.

Es importante tener en cuenta estas consideraciones al seleccionar la herramienta adecuada para cada proyecto de aprendizaje profundo.

Las unidades de procesamiento gráfico (GPU)

Son dispositivos especializados diseñados para realizar cálculos intensivos en paralelo de manera extremadamente eficiente. A diferencia de la unidad de procesamiento central (CPU), que está optimizada para realizar una variedad de tareas de propósito general, las GPU están específicamente diseñadas para manejar operaciones matriciales y vectoriales de manera simultánea, lo que las hace ideales para aplicaciones que requieren un alto rendimiento computacional, como el procesamiento gráfico, la simulación científica y, crucialmente, el aprendizaje profundo.

En el aprendizaje profundo, las GPU juegan un papel fundamental en el entrenamiento y la inferencia de modelos de redes neuronales, especialmente en arquitecturas profundas y complejas. Debido a la naturaleza altamente paralela de los cálculos involucrados en el aprendizaje profundo, como las operaciones de multiplicación de matrices y la propagación hacia atrás (backpropagation), las GPU pueden procesar grandes volúmenes de datos de manera simultánea y acelerar significativamente el tiempo de entrenamiento de los modelos.

La capacidad de las GPU para ejecutar múltiples operaciones en paralelo permite a los investigadores y profesionales del aprendizaje profundo experimentar con arquitecturas más complejas y conjuntos de datos más grandes, lo que a su vez impulsa la innovación y el avance en el campo. Además, las GPU también son utilizadas en la inferencia en tiempo real de modelos de aprendizaje profundo en aplicaciones como:

Reconocimiento de imágenes

Procesamiento del lenguaje natural

Conducción autónoma

Esto sucede, porque requieren una respuesta rápida y eficiente del modelo.

Red Neuronal utilizando Scikit-learn

Para implementar una red neuronal utilizando la biblioteca Scikit-learn (Sklearn), puedes seguir estos pasos generales:

1. Preparación de los datos:

Asegúrate de tener los datos correctamente estructurados en matrices NumPy u otros formatos compatibles con Sklearn. Luego, divide los datos en conjuntos de entrenamiento y prueba utilizando la función **train_test_split** de Sklearn si no lo has hecho aún.

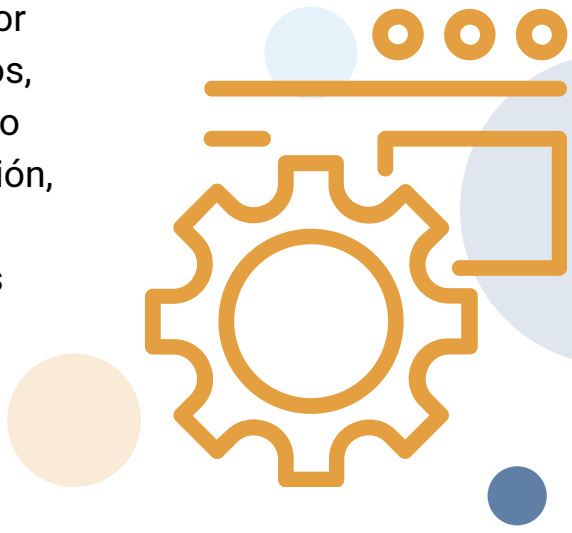
2. Importación de la biblioteca:

Desde el módulo **neural_network** de Sklearn
Importa la clase **MLPClassifier** (para clasificación)
Importa la clase **MLPRegressor** (para regresión)

Estas son las clases que te permiten crear redes neuronales.

3. Configuración del modelo:

Crea una instancia del clasificador o regresor especificando los hiperparámetros deseados, como el número de capas ocultas, el número de neuronas por capa, la función de activación, etc. Puedes consultar la documentación de Sklearn para obtener más detalles sobre los hiperparámetros disponibles.



4. Entrenamiento del modelo:

Utiliza el método **fit** para entrenar tu modelo con los datos de entrenamiento preparados anteriormente. Esto ajustará los pesos de la red neuronal utilizando el algoritmo de optimización especificado (por defecto, utiliza el algoritmo de retropropagación).

5. Evaluación del modelo:

Evalúa el rendimiento de tu modelo utilizando los datos de prueba utilizando métricas relevantes para tu problema, como precisión, exactitud, F1-score, MSE (Mean Squared Error), etc.

6. Ajuste de hiperparámetros:

Opcionalmente, puedes realizar una búsqueda de hiperparámetros utilizando técnicas como GridSearchCV o RandomizedSearchCV para encontrar la combinación óptima de hiperparámetros que maximice el rendimiento del modelo.

7. Predicciones

Utiliza el método **predict** para realizar predicciones en nuevos datos utilizando el modelo entrenado.

Es importante tener en cuenta que, si bien Sklearn ofrece una implementación básica de redes neuronales, está diseñada principalmente para aplicaciones sencillas y no es tan flexible o potente como otras bibliotecas específicamente dedicadas al aprendizaje profundo, como TensorFlow, PyTorch o Keras. Sin embargo, puede ser útil para prototipado rápido y aplicaciones simples donde la complejidad de una red neuronal completa no es necesaria.

Ejemplo de cómo implementar una red neuronal para clasificación con (MLPClassifier) utilizando Sklearn

Veamos el procedimiento:

1 **Cargamos las librerías a utilizar.**

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

2 **Cargamos las librerías a utilizar.**

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

```
# Explorando los datos (Opcional)
type(iris)
iris.keys()
iris['data']
iris['target']
iris['target_names']
iris['DESCR']
iris['feature_names']
```

3 **Dividimos los datos en conjuntos de entrenamiento y prueba.**

```
# 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)
```

4 **Escalamos las características utilizando StandardScaler para asegurarnos de que todas tengan la misma escala.**

```
# Escalar las características para un mejor rendimiento del
modelo
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

5 **Creamos una instancia de MLPClassifier con una capa oculta de 100 neuronas, función de activación ReLU, algoritmo de optimización 'adam' y un máximo de 500 iteraciones.**

```
# Crear una instancia de MLPClassifier
mlp_clf = MLPClassifier(hidden_layer_sizes=(100),
                        activation='relu',
                        solver='adam',
                        max_iter=100,
                        random_state=42,
                        verbose=True)
```

6 **Entrenamos el modelo utilizando los datos de entrenamiento escalados.**

```
# Entrenar el modelo
mlp_clf.fit(X_train_scaled, y_train)
```

7 **Realizamos predicciones en el conjunto de prueba.**

```
# Realizar predicciones en el conjunto de prueba
y_pred = mlp_clf.predict(X_test_scaled)
print(y_test)
print(y_pred)
```

8 **Calculamos la precisión del modelo utilizando la función accuracy_score.**

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

El anterior es un ejemplo básico y se pueden ajustar muchos otros hiperparámetros de MLPClassifier para optimizar el rendimiento del modelo según el conjunto de datos específico. Además, también puedes explorar otras funciones de activación, solucionadores y arquitecturas de capas ocultas para adaptar el modelo a tus necesidades específicas.

Ejemplo básico de cómo implementar una red neuronal para regresión utilizando MLPRegressor de Scikit-learn

Veamos el procedimiento:

1 Cargamos las librerías a utilizar.

```
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
```

2 Cargamos el conjunto de datos de Boston Housing.

```
# Cargar el conjunto de datos de california Housing
housing = fetch_california_housing()
X, y = housing.data, housing.target
```

```
# Explorando los datos (Opcional)
type(housing)
housing.keys()
housing['data']
housing['target']
housing['target_names']
housing['DESCR']
housing['feature_names']
```

3 **Dividimos los datos en conjuntos de entrenamiento y prueba.**

```
# 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)
```

4 **Escalamos las características utilizando StandardScaler para asegurarnos de que todas tengan la misma escala.**

```
# Escalar las características para un mejor rendimiento del
modelo
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

5 **Creamos una instancia de MLPRegressor con una capa oculta de 100 neuronas, función de activación ReLU, algoritmo de optimización 'adam' y un máximo de 500 iteraciones.**

```
# Crear una instancia de MLPRegressor
mlp_reg = MLPRegressor(hidden_layer_sizes=(100,),
                       activation='relu',
                       solver='adam',
                       max_iter=100,
                       random_state=42,
                       verbose=True)
```

6 **Entrenamos el modelo utilizando los datos de entrenamiento escalados.**

```
# Entrenar el modelo
mlp_reg.fit(X_train_scaled, y_train)
```

7 **Realizamos predicciones en el conjunto de prueba.**

```
# Realizar predicciones en el conjunto de prueba
y_pred = mlp_reg.predict(X_test_scaled)
```

8 **Calculamos el error cuadrático medio del modelo utilizando la función mean_squared_error.**

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

Al igual que en el ejemplo de clasificación, puedes ajustar muchos otros hiperparámetros de MLPRegressor para optimizar el rendimiento del modelo según el conjunto de datos específico. Además, también puedes explorar otras funciones de activación, solucionadores y arquitecturas de capas ocultas para adaptar el modelo a tus necesidades específicas de regresión.