Scikit-Learn

Spis treści

Scikit-Learn

`Scikit-Learn` (lub też `sklearn`) jest to prosta biblioteka służąca do uczenia maszynowego. Opiera się ona na kilku podstawowych bibliotekach, takich jak `numpy` czy `matplotlib` i zawiera wiele gotowych algorytmów służących do klasyfikacji, regresji, grupowania i analizy danych. Jest niezwykle przydatna na różnych etapach pracy z danymi, począwszy od testowania modeli, które dzięki niej są proste w implementacji, kończąc na zaawansowanych modelach eksploracji danych, które mogą dawać pożądane przez nas wyniki. Biblioteka posiada też wiele modułów użytecznych w preprocessingu lub ewaluacji.

Instalacja

Instalacja biblioteki `Scikit-Learn` jest analogiczna do instalacji innych bibliotek w naszym środowisku pracy, zwykle możemy zrobić to na dwa sposoby:

- Za pomocą instalatora pakietów PIP, jeśli korzystamy ze środowiska systemowego:
  ```
  pip install scikit-learn
  ```
- Za pomocą instalatora pakietów CONDA, jeśli korzystamy ze środowiska Anaconda:
  ```
  conda install -c anaconda scikit-learn
  ```

Import

Ponieważ biblioteka posiada wiele różnych możliwości, z których niemalże nigdy nie będziemy korzystali, jednocześnie konwencją stało się, że zamiast importować całą bibliotekę importujemy poszczególne funkcje z konkretnych modułów, np.:

```
from sklearn.metrics import accuracy_score
```

Powyższy kod informuje, że chcemy zaimportować funkcję `accuracy_score` z modułu `metrics` biblioteki `Scikit-Learn`. Przy importowaniu korzystamy ze skróconej nazwy: `sklearn`.

Tworzenie modelu

Model w uczeniu maszynowym to tak naprawdę serce całej operacji, to właśnie model chcemy nauczyć, żeby później móc za jego pomocą prognozować. Czym tak właściwie jest model? Model to algorytm, który możemy nauczyć lub który jest już nauczony. Co oznacza uczenie? W zależności od algorytmu może mieć to różne definicje, natomiast w większości przypadków kiedy mówimy o uczeniu modelu, chcemy, aby po podaniu do niego danych wejściowych dostać oczekiwany przez nas wynik, tak aby później, kiedy podajemy nowe dane wejściowe w takim samym formacie móc otrzymać wynik na podstawie danych przeanalizowanych wcześniej.

Kwestia utworzenia modelu, niezależnie od tego czy jest to klasyfikator, regresor czy model grupujący, opiera się na wybraniu algorytmu odpowiadającego naszemu problemowi, zaimportowaniu go a następnie utworzenie jego instancji.

```
from sklearn.neural_network import MLPClassifier

model = MLPClassifier()
```

W ten sposób utworzyliśmy wielowarstwową sieć neuronową funkcjonującą jako `klasyfikator` (ang. *Multi-Layer Perceptron Classifier*). Każdy z tym podobnych modeli ma oczywiście parametry, które możemy dostosować, ten temat będzie omówiony szerzej podczas pracy z konkretnymi modelami.

Trening

Mając przygotowany model możemy przystąpić do trenowania algorytmu. Odpowiada za to metoda `fit()` wywoływana na konkretnej instancji modelu. W większości przypadków przyjmuje ona dwa parametry: x - czyli dane wejściowe oraz y - oczekiwane wyniki (przygotowane wcześniej na podstawie opracowywanego przez nas datasetu).

```
from sklearn.neural_network import MLPClassifier

model = MLPClassifier()
model.fit(x_train, y_train)
```

Predykcja

Wytrenowany model możemy wykorzystać do prognozowania wyników na danych, których nie użyliśmy do treningu, wykonujemy to w następujący sposób:

```
from sklearn.neural_network import MLPClassifier
model = MLPClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

Algorytm spróbuje dopasować dane stosując analogie poznane na zbiorach treningowych. W zmiennej `y_pred` dostaniemy przewidywane wartości odpowiadające konkretnym danym wejściowym ze zbioru testowego.

Sprawdzenie trafności modelu - moduł `sklearn.metrics`

Biblioteka `scikit-learn`, a dokładniej mówiąc moduł `metrics`, oferuje szeroką gamę sposobów na sprawdzenie dokładności naszego modelu. Wybór sposobu weryfikacji należy do programisty i będzie różnił się w zależności od typu modelu. Najpopularniejsza metoda w przypadku klasyfikacji to `accuracy_score` - prosta funkcja przyjmująca dwa argumenty: prawdziwe wartości y oraz wypredykowane wartości y. Funkcja następnie liczy w ilu przypadkach algorytm ocenił trafnie oraz przedstawia wynik z przedziału od 0 do 1, gdzie 1 oznacza stuprocentową trafność.

```
from sklearn.metrics import accuracy_score

score = accuracy_score(y_true, y_pred)
```

Dla regresji możemy wyróżnić kilka najpopularniejszych metod:

- `mean_squared_error` - błąd średniokwadratowy

```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(y_true, y_pred)
```

- `r2_score` - współczynnik determinacji

```
from sklearn.metrics import r2_score

error = r2_score(y_true, y_pred)
```

- `mean_absolute_error` - średni błąd bezwzględny

```
from sklearn.metrics import mean_absolute_error

error = mean_absolute_error(y_true, y_pred)
```

Wszystkie metody za pomocą różnego rodzaju wzorów matematycznych wyznaczają jak blisko wyników prawdziwych były wyniki wypredykowane.

Wykorzystanie scikit-learn w preprocessingu

Warto zwrócić uwagę na najważniejsze moduły oraz funkcje, z których możemy korzystać na tym etapie pracy:

sklearn.model_selection

- `train_test_split` - funkcja pozwalająca na podział danych na zbiór testowy i treningowy

```
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,
                                                    shuffle=True)
```

Powyższy kod podzieli nasze zbiory X i y na zbiory treningowy oraz testowy X i treningowy oraz testowy y w proporcjach 80:20 (wartość parametru `test_size`) oraz przemiesza losowo wartości (parametr `shuffle`).

Możemy wyróżnić najważniejsze parametry:

- `test_size` - rozmiar zbioru testowego,
- `train_size` - rozmiar zbioru treningowego

Dwa powyższe są bezpośrednio zależne, tzn. jeśli `test_size` zostanie ustawione na 0.2 to `train_size` automatycznie ustawiany jest na 0.8.

- `shuffle` - parametr typu bool, informujący czy zbiór ma zostać przemieszany

sklearn.datasets

Moduł zawierający gotowe datasety, na których możemy ćwiczyć oraz generatory losowych datasetów odpowiednich do trenowania modeli.

```
from sklearn.datasets import load_iris

df = load_iris()
```

Przykładowo, powyższy kod importuje funkcję, która załaduje nam dataset `iris` oraz korzystając z niej przypisze ten dataset do zmiennej `df`.

sklearn.preprocessing

- `LabelEncoder` - algorytm kodujący etykiety (na przykład w postaci napisów) na liczby naturalne od 0 (np. [„kot”, „mysz”, „pies”] na [0, 1, 2]). Aby z niego skorzystać musimy stworzyć instancję `LabelEncoder()` oraz na tej instancji wywołać metodę `fit_transform()`, podając jako parametr zbiór etykiet, które chcemy zakodować.

```
from sklearn.preprocessing import LabelEncoder

arr = ["kot", "mysz", "pies"]
le = LabelEncoder()
transformed = le.fit_transform(arr)
```

Po wykonaniu tych operacji, w zmiennej *transformed* otrzymamy listę [0, 1, 2].

- `MinMaxScaler` - algorytm skalujący nasze dane na wartości od 0 do 1, gdzie 0 odpowiada najmniejszej, a 1 największej wartości w naszym zbiorze. Ważne jest, aby przekazywana struktura danych była dwuwymiarowa.

```
import numpy as np

arr = np.array([0, 5, 15])
arr = arr.reshape(-1, 1)
```

Jeśli nasze dane są jednowymiarowe najłatwiej skorzystać z tablic biblioteki `numpy`, a następnie korzystając z metody `reshape()` uzyskać „sztuczny" drugi wymiar.
Nasza tablica po skorzystaniu z tej metody wygląda następująco:

```
[[ 0], [ 5], [15]]
```

Skoro nasze dane są już dwuwymiarowe możemy przeskalować je za pomocą `MinMaxScaler()`.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
transformed = scaler.fit_transform(arr)
```

- `StandardScaler` - algorytm skalujący nasze dane za pomocą wzoru *z = (x - u) / s*, gdzie *x* to nasze dane, *u* to średnia naszych danych, a *s* to odchylenie standardowe. Podobnie jak w przypadku `MinMaxScaler()`, niezbędne są dwuwymiarowe dane.

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

arr = np.array([0, 5, 15])
arr = arr.reshape(-1, 1)
scaler = StandardScaler()
transformed = scaler.fit_transform(arr)
```