Laboratorium: Regresja

May 12, 2022

1 Cel/Zakres

- Przypomnienie zasady działania i ograniczeń perceptronu
- Budowa sieci neuronowej przy pomocy Tensorflow/Keras
- Trening sieci i analiza wyników

2 Zadania

2.1 Perceptrony i irysy

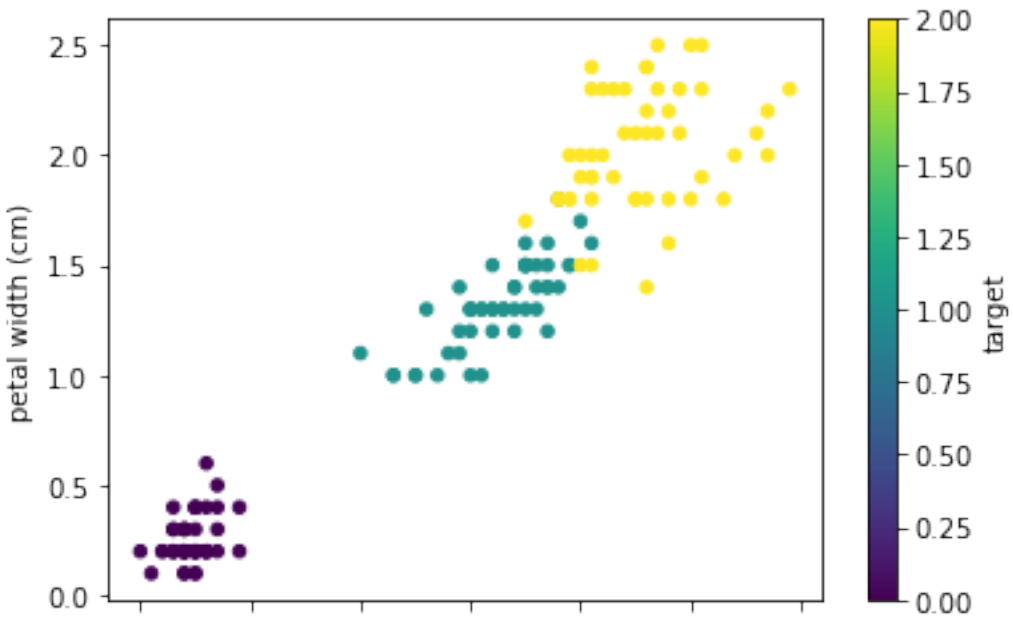
Pobierz zbiór danych

```
from sklearn.datasets import load_iris
iris = load_iris(as_frame=True)
```

Zwróć uwagę, jak długość i szerokość płatka jest powiązana z gatunkiem irysów.

```
pd.concat([iris.data, iris.target], axis=1).plot.scatter(
    x='petal length (cm)',
    y='petal width (cm)',
    c='target',
    colormap='viridis'
)
```

```
<AxesSubplot:xlabel='petal length (cm)', ylabel='petal width (cm)'>
```



Podziel zbiór danych na zbiór uczący oraz zbiór testowy (8/2), a następnie kolejno dla każdego z gatunków $(0, 1, 2)$ zbuduj perceptron, przeprowadź jego uczenie i oceń jego dokładność dla zbioru uczącego i dla zbioru testowego.

Dla każdej z klas sprawdź wagę biasu (`w_0`) oraz poszczególnych wejść (`w_1`, `w_2`). Czy wartości te korespondują z tym co widzisz na obrazku?

Czy, patrząc na ten obrazek, potrafisz wyjaśnić różnice w dokładności dla poszczególnych klas?

Dokładność dla każdej z klas zapisz jako krotkę `(a_tr, a_te)`, a wszystkie krotki zapisz w pliku `per_acc.pkl` jako listę krotek: `[(a_tr_0, a_te_0), (a_tr_1, a_te_1), (a_tr_2, a_te_2)]`.

Podobnie, wagi dla poszczególnych trzech klas zapisz jako listę 3-elementowych krotek `(w_0, w_1, w_2)` w pliku `per_wght.pkl`.

2.2 Perceptron i XOR

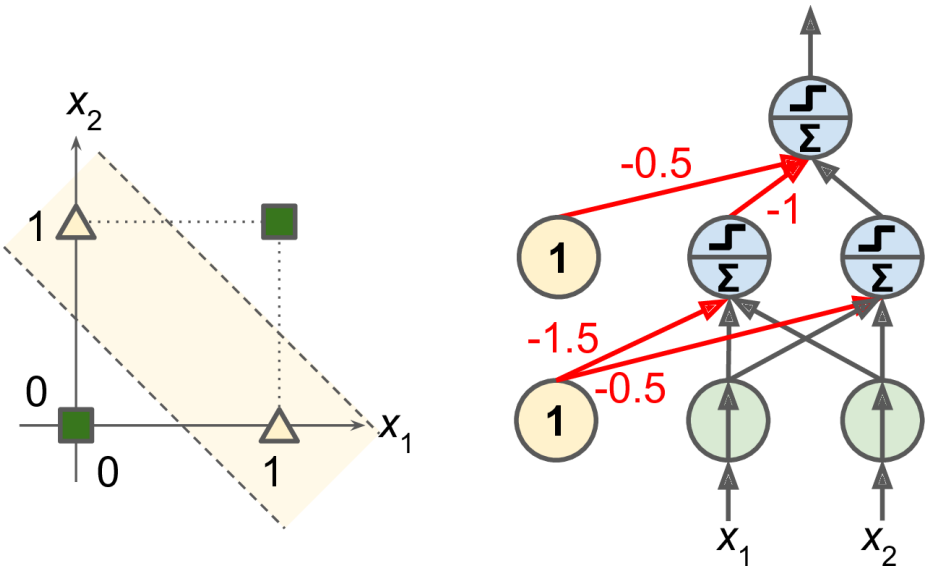
Przygotuj prosty zbiór danych modelujący operację XOR.

```
X = np.array([[0, 0],
     [0, 1],
     [1, 0],
     [1, 1]])
y = np.array([0,
     1,
     1,
     0])
```

Utwórz i przeprowadź uczenie perceptronu dla tych danych. Czy jest zdolny do poprawnego przeprowadzenia predykcji? Jak wyglądają jego wagi?

2.3 XOR, drugie podejście

Problem klasyfikacji XOR teoretycznie da się rozwiązać przy pomocy sieci przedstawionej na poniższym rysunku. Zamodeluj sieć neuronową (perceptron wielowarstwowy, MLP) zgodnie z nim przy pomocy Tensorflow/Keras.



Sieć powinna posiadać dwie warstwy gęste:

- warstwę ukrytą, składającą się z dwóch neuronów, o dwóch wejściach (`input_dim`) i funkcji aktywacji tanh,
- warstwę wyjściową, zawierającą jeden neuron i sigmoidalnej funkcji aktywacji.

Skompiluj model z entropią krzyżową jako funkcją straty oraz stochastycznym optymalizatorem gradientowym o domyślnych parametrach.

Przeprowadź uczenie przez 100 epok i wyświetl historię wartości funkcji straty. Jaka byłaby jej pożądana wartość dla rozważanego problemu?

```
history = model.fit(X, y, epochs=100, verbose=False)
print(history.history['loss'])
```

Sparwdź, jak sieć radzi sobie z klasyfikacją zbioru `X`. Czy wyniki są zadowalające? Powtórz cały proces (wraz z tworzeniem modelu) kilka razy. Czy wyniki są różne? Dlaczego?

Poeksperymentuj z hiperparametrami modelu:

- spróbuj zmienić optymalizator z SGD na Adam,
- sprawdź działanie z różnymi wartościami kroku optymalizatora (`learning_rate`),
- wypróbuj inne funkcje straty (np. MAE) oraz funkcje aktywacji poszczególych warstw.

Czy teraz częściej pojawiają się rozwiązania satysfakcjonujące (czyli dające predykcje bliskie $[0, 1, 1, 0]$)?

```
model = make_model()
model.compile(loss="binary_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.05))
```

Po uzyskaniu satysfakcjonującego modelu („zera” < 0.1, „jedynki” > 0.9) pobierz jego wagi przy pomocy funkcji `get_weights()`.

Zapisz jej wyjście do pliku `mlp_xor_weights.pkl`.