Lab 2 - Zastosowanie Perceptronu w klasyfikacji obrazów

1. Perceptron podstawowe informacje - przypomnienie z Lab 1

Perceptron to prosty model sztucznego neuronu, który rozwiązuje podstawowe zadania obliczeniowe i stanowi punkt wyjścia do budowy bardziej zaawansowanych sieci neuronowych.

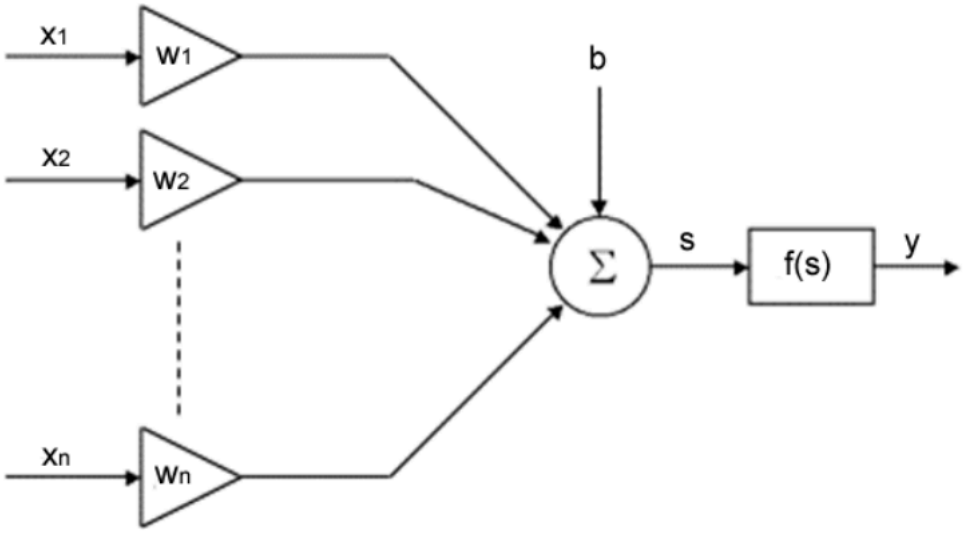
Działa on poprzez klasyfikowanie danych wejściowych i odpowiednie ustawianie wartości wyjściowych. Zanim jednak perceptron zacznie działać prawidłowo, musi zostać wytrenowany. Proces ten polega na podawaniu danych treningowych i modyfikowaniu wag wejściowych w celu uzyskania oczekiwanych wyników na wyjściu. Perceptrony mogą klasyfikować dane na zbiory, które są **liniowo separowalne.**

Wyjście perceptronu oblicza się według wzoru:

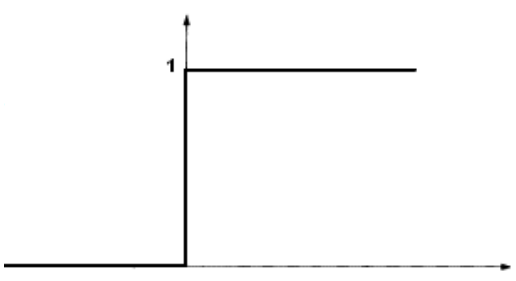
$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

gdzie: n - liczba wejść w perceptronie, x_i - wejście, w_i - waga dla i-tego wejścia, b - bias (stała wartość), f - funkcja aktywacji, y - wyjście.

Poniższy rysunek przedstawia działanie perceptronu:



Istnieje kilka funkcji aktywacyjnych (progowa, bipolarna, tanh, sigmoid). Funkcja progowa może być zdefiniowana jako:



$$f(s) = \begin{cases} 1, & gdy\ s > 0 \\ 0, & gdy\ s \leq 0 \end{cases}$$

2. Uczenie perceptronu - przypomnienie z Lab 1

Załóżmy, że nie są znane wagi $w_i, i = 1, 2,...$, oraz bias b. Nieznane wagi wyznaczymy w procesie uczenia. Jest to tzw. uczenie nadzorowane, polegające na podawaniu perceptronowi wartości ze zbioru treningowego X_t, dla których znamy prawidłowe wynik Y_t.

Reguła uczenia (aktualizacji wag) perceptronu przebiega w następujący sposób:

1. wybieramy w sposób losowy wagi początkowe w_i i bias b oraz zakładamy współczynnik uczenia η (powinien być w zakresie (0,1), z reguły liczba w okolicach 0.01),
2. na wejścia neuronu podajemy wektor danych treningowych X_t,
3. dla wszystkich danych treningowych obliczamy wartość wyjściową perceptronu y zgodnie ze wzorem:
$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$
4. porównujemy wartości wyjściowe ze wzoru y z wartościami z danych treningowych y_t
5. Dokonujemy modyfikacji wag według zależności:
   jeśli $y = y_t$ to wagi pozostają niezmienione,
   jeśli $y \neq y_t$ to aktualizujemy wagi oraz bias:
$$w_i = w_i + \eta\left(y - y_t\right)x_i$$
$$b = b + \eta\left(y - y_t\right)$$
6. Wracamy do punktu 2.

Algorytmy powtarza się tak długo, aż błąd na wyjściu będzie mniejszy od założonej tolerancji. Tolerancję można zdefiniować jako odsetek poprawnie przewidzianych wartości na podstawie danych treningowych w stosunku do całkowitej liczby próbek w zbiorze.
Jedno przejście algorytmu nazywane jest **epoką uczenia** t**.**

3. **Zadanie**

Celem zadania jest wykorzystanie perceptronu do klasyfikacji dwóch klas obiektów graficznych z wykorzystaniem zbioru danych MNIST (plik znajduje się w folderze Lab2 na platformie Teams). Należy skorzystać z implementacji perceptronu opracowanej podczas Lab1.

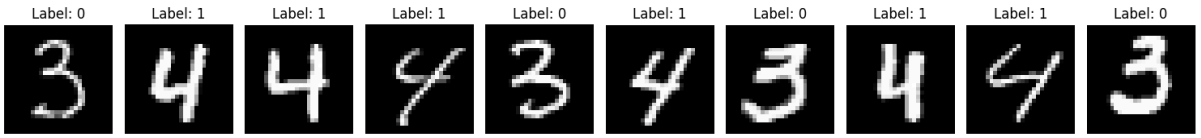
Kroki do wykonania:

1. Zaimportuj pliki *mnist_train.csv* oraz *mnist_test.csv* jako obiekty typu dataframe. Podziel dane na zbiory wejściowe X (zawierające piksele) oraz etykiety y (zawierające klasy), aby uzyskać zestawy X_train, y_train, X_test i y_test.

   Protip: Pierwsza kolumna danych to etykiety, a pozostałe kolumny zawierają wartości pikseli.

2. Wybierz dwie cyfry z dostępnego zbioru danych. Przefiltruj dane, aby zbiór zawierał tylko wybrane cyfry. Zamień etykiety na potrzeby klasyfikacji binarnej: przypisz wartość 1 dla jednej i 0 dla drugiej.
3. Wyświetl kilka przykładowych obrazów cyfr.

   Protip: możesz użyć matplotlib.pyplot.imshow

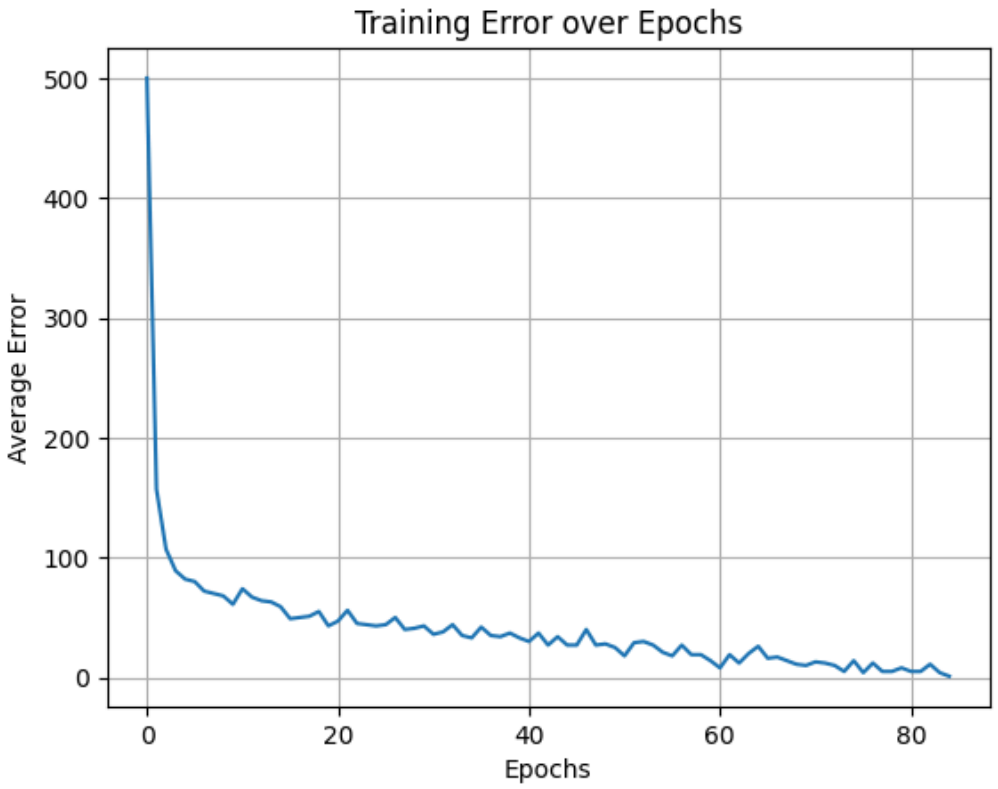


4. Dokonaj normalizacji danych treningowych, tak aby wartości X_train i X_test mieściły się w przedziale (0,1).

   Protip: Zbiór X zawiera wartości pikseli w przedziale (0, 255).

5. Wykorzystaj klasę *Perceptron* z Laboratorium 1 i zainicjalizuj ją odpowiednim rozmiarem danych treningowych.

6. Zmodyfikuj metodę *train* w klasie *Perceptron* w taki sposób, aby zwracała lub wyświetlała wartość błędu po każdej epoce trenowania. Następnie uruchom proces trenowania z wykorzystaniem danych treningowych.



7. Za pomocą metody *evaluate_test* sprawdź, ile próbek zostało błędnie sklasyfikowanych przez model.
8. Opracuj macierz pomyłek (confusion matrix) dla zbioru testowego.

   Protip: Wykorzystaj metodę *predict* z klasy *Perceptron* do uzyskania przewidywanych etykiet (y_predicted), a następnie porównaj je z rzeczywistymi wartościami y_test za pomocą funkcji *confusion_matrix* z biblioteki *sklearn.metrics*.

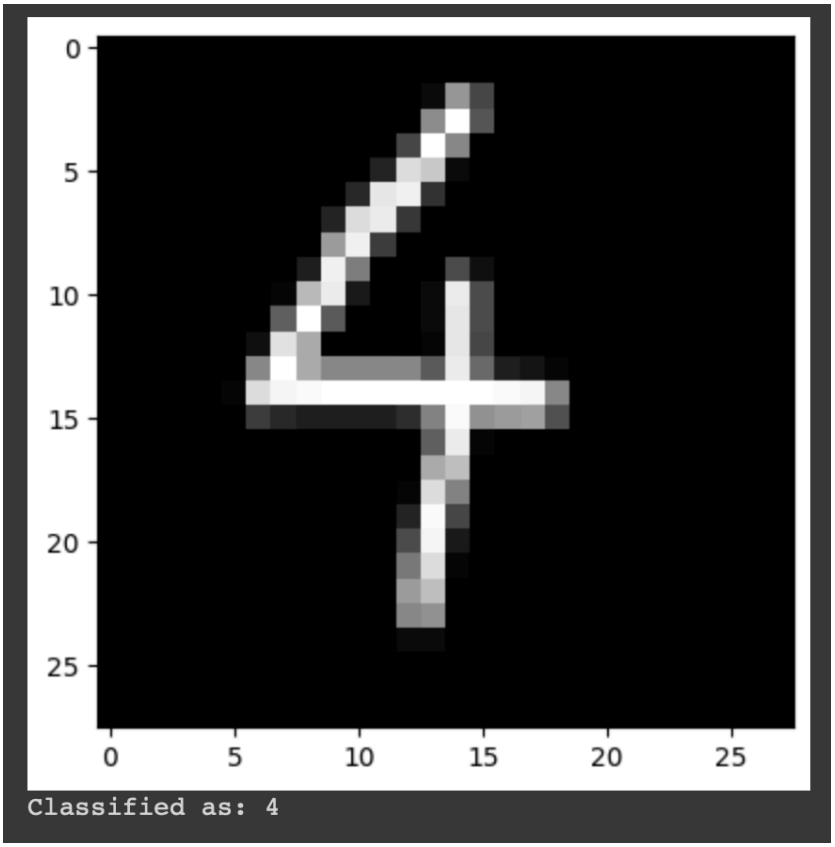
9. Oblicz następujące miary: dokładność (accuracy), precyzję (precision), czułość (recall) oraz F1-score dla zbioru testowego.

   Protip: Wykorzystaj metody *accuracy_score*, *precision_score*, *recall_score* oraz *f1_score* z biblioteki *sklearn.metrics.*

```
accuracy: 0.9919678714859438
precision: 0.9908536585365854
recall:  0.9928716904276986
f1_score: 0.9918616480162767
```

10. Narysuj dowolną cyfrę w programie graficznym i przekonwertuj obraz do bitmapy o rozdzielczości 28x28 pikseli. Przekształć obraz na odpowiedni wektor, dokonaj normalizacji i przetestuj klasyfikator na tym przykładzie.

    Protip: Skorzystaj z biblioteki *PIL*, aby zmienić rozmiar obrazu do 28x28 pikseli, skonwertować go do wektora. Wykorzystaj metodę *predict* z klasy *Perceptron*.



Pytania dla chętnych:

1. Co można powiedzieć o modelu na podstawie miar wydajności?
2. Czy perceptron może klasyfikować próbki z więcej niż dwóch klas?
3. Jakie są efekty wykorzystania innej funkcji aktywacyjnej na wydajność i dokładność perceptronu?
4. Co wnosi analiza PCA dla zbioru X_train?