

## Implementacja algorytmu Nearest Neighbours

Projekt wykonali: Jakub Płoskonka, Rafał Nowicki

Prowadzący: dr inż. Marcin Pietroń

Data wykonania: 10.01.2025r.

## Spis treści

---

Cel projektu.....	3
Funkcje programu .....	3
Kod programu .....	4

## Cel projektu

---

Implementacja algorytmu Nearest Neighbours oraz przedstawienie wyników działania algorytmu na danych testowych w postaci wizualizacji graficznych za pomocą biblioteki `matplotlib`. W ramach realizacji celu projekt obejmie również analizę porównawczą skuteczności i czasu działania własnej implementacji w odniesieniu do wersji dostępnej w standardowych bibliotekach.

## Funkcje programu

---

### Wczytywanie danych

Program umożliwia wczytanie danych wejściowych z pliku CSV lub bezpośrednio z kodu. Dane te są następnie przetwarzane do odpowiedniego formatu dla algorytmu.

### Implementacja algorytmu Nearest Neighbours

Program zawiera własną implementację algorytmu k najbliższych sąsiadów (KNN). Główne kroki algorytmu:

- Obliczanie odległości (np. euklidesowej) między punktami.
- Znajdowanie najbliższych sąsiadów dla każdego punktu testowego.
- Klasyfikacja lub przewidywanie na podstawie sąsiadów.

### Wykorzystanie wersji bibliotecznej algorytmu

Program porównuje własną implementację z wersją dostępną w popularnych bibliotekach, takich jak `scikit-learn`.

### Wizualizacja wyników

Wyniki działania algorytmu (np. dokładność klasyfikacji, rozkład danych) są przedstawiane w postaci wykresów przy użyciu `matplotlib`.

- Wykresy rozkładu danych i ich klasyfikacji.
- Wykres porównujący czas działania implementacji własnej i bibliotecznej.
- Wykresy analizy jakości działania algorytmu (np. macierz pomyłek).

### Analiza czasowa i jakościowa

Program mierzy czas działania obu wersji algorytmu oraz dokładność ich wyników. Dane te są następnie prezentowane na wykresach porównawczych.

### Obsługa parametrów algorytmu

Program umożliwia użytkownikowi zmianę parametrów, takich jak:

- Liczba sąsiadów ( $k$ ).
- Euklidesowa metryka odległości.
- Podział danych na zbiór treningowy i testowy.

## Testy jednostkowe

W programie zintegrowano testy jednostkowe, które sprawdzają poprawność działania implementacji algorytmu oraz jego funkcji pomocniczych.

## Kod programu

---

Klasa **NearestNeighbours** implementuje algorytm k najbliższych sąsiadów (k-NN) do klasyfikacji danych. Konstruktor inicjalizuje wartość **k**, określającą liczbę najbliższych sąsiadów, oraz pola do przechowywania danych treningowych i ich etykiet.

- Metoda **fit** zapisuje dane treningowe i odpowiadające im etykiety w postaci tablic NumPy.
- Metoda **\_euclidean\_distance** oblicza odległość euklidesową między dwoma punktami.
- Metoda **predict** klasyfikuje dane testowe, iterując przez każdy punkt testowy, obliczając jego odległości do wszystkich punktów treningowych, a następnie wybierając **k** najbliższych sąsiadów.
- Najczęściej występująca etykieta wśród tych sąsiadów jest przypisywana jako wynik dla danego punktu testowego.

Funkcja **compare\_algorithms()** porównuje działanie własnej implementacji algorytmu najbliższych sąsiadów (**Nearest Neighbours**) z wersją biblioteczną z **scikit-learn**. Najpierw generuje dane treningowe i testowe za pomocą funkcji **generate\_data()**. Następnie mierzy czas wykonania i dokładność klasyfikacji dla obu wersji algorytmu. Własna implementacja wykorzystuje obiekt klasy **NearestNeighbours**, natomiast wersja bibliteczna – **KNeighborsClassifier**. Po uzyskaniu wyników funkcja wypisuje dokładność oraz czas wykonania dla obu wersji. Na końcu wizualizuje wyniki za pomocą wykresów porównujących czas wykonania i dokładność oraz wykresów przedstawiających najbliższych sąsiadów oraz klasyfikację punktów testowych.

Funkcja **generate\_data()** generuje sztuczne dane klasyfikacyjne, a następnie dzieli je na zbiór treningowy i testowy. Oto szczegóły jej działania:

### 1. Generowanie danych:

Używana jest funkcja **make\_classification()**, która tworzy zbiór danych do klasyfikacji. Parametry:

- **n\_samples=1000**: liczba próbek wynosi 1000,
- **n\_features=2**: dane mają 2 cechy (feature),
- **n\_informative=2**: wszystkie cechy są istotne dla klasyfikacji,
- **n\_redundant=0**: brak cech redundantnych,
- **n\_clusters\_per\_class=1**: każda klasa jest skupiona w jednym klastrze,
- **random\_state=42**: ustawienie ziarna losowości dla powtarzalności.

## 2. Podział danych:

Funkcja `train_test_split()` dzieli dane na zbiór treningowy i testowy:

- `test_size=0.3`: 30% danych trafi do zbioru testowego, a 70% do treningowego,
- `random_state=42`: ponownie ustawiane jest ziarno losowości.

## 3. Zwracanie wyników:

Funkcja zwraca dane podzielone na cztery części: `X_train`, `X_test`, `y_train`, `y_test`.

Funkcja **`plot_nearest_neighbors()`** tworzy wizualizację danych treningowych i testowych, przedstawiając klasyfikację najbliższych sąsiadów.

1. **Inicjalizacja wykresu** – Ustawiany jest rozmiar wykresu (8x8).
2. **Rysowanie danych treningowych** – Funkcja iteruje przez unikalne etykiety (`y_train`). Dla każdej klasy wybiera odpowiednie punkty z `X_train` i rysuje je na wykresie jako punkty o różnym kolorze z etykietą klasy.
3. **Rysowanie danych testowych** – Model dokonuje predykcji dla danych testowych `X_test`. Następnie każdy punkt testowy jest rysowany na wykresie:
  - Jeśli predykcja dla danego punktu jest poprawna (zgodna z `y_test`), punkt jest zaznaczony na czerwono ('red').
  - Jeśli predykcja jest błędna, punkt jest zaznaczony na czarno ('black').
4. **Stylizacja wykresu** – Funkcja dodaje tytuł, opisy osi (`Feature 1` i `Feature 2`), legendę oraz siatkę, a następnie wyświetla wykres za pomocą `plt.show()`.

Wizualizacja ta pozwala łatwo zauważyć, jak model klasyfikuje punkty testowe w odniesieniu do danych treningowych.

Funkcja **`plot_test_classification()`** wizualizuje klasyfikację punktów testowych w porównaniu do danych treningowych.

1. **Inicjalizacja wykresu** – Tworzony jest wykres o rozmiarze 8x8.
2. **Rysowanie danych testowych** – Model wykonuje predykcję dla danych `X_test`. Następnie funkcja iteruje przez unikalne etykiety klas (`y_train`), wybiera punkty testowe sklasyfikowane jako dana klasa i rysuje je na wykresie jako punkty o wyższym poziomie przezroczystości (`alpha=0.6`) i z obramowaniem (`edgecolor='k'`).

3. **Rysowanie danych treningowych** – Funkcja iteruje przez klasy i wybiera odpowiadające im punkty z danych treningowych `X_train`. Punkty te są rysowane z niższym poziomem przezroczystości (`alpha=0.3`), aby nie dominowały nad punktami testowymi.
4. **Stylizacja wykresu** – Dodawany jest tytuł, opisy osi (`Feature 1` i `Feature 2`), legenda oraz siatka. Na końcu wykres jest wyświetlany przy użyciu `plt.show()`.

Wizualizacja pozwala zobaczyć, jak model klasyfikuje dane testowe i jak te punkty są rozmieszczone względem danych treningowych.