Wstęp do sztucznej inteligencji
Modele bayesowskie

Jakub Robaczewski

Algorytm klasyfikacji:

Celem zadania było zaimplementowanie naiwnego klasyfikatora Bayesa wykorzystującego algorytm walidacji krzyżowe. Stworzone przeze mnie funkcje możemy podzielić na 3 grupy: zarządzanie danymi, klasyfikator i walidację. By ułatwić klasyfikację klasy podane w zadaniu (1, 2, 3) zostały przemianowane na (0, 1, 2).

Zarządzanie danymi:

Funkcje zarządzania danymi służą do pobrania danych z podanego pliku oraz dalszej obróbki

- read_from_file() – pobiera dane z podanego pliku
- split_data() – dzieli podany zbiór na 2, wykorzystując podane proporcje
- group() – dzieli zbiór na podzbiory według przynależności do klas
- pick_parameters () – wybiera z listy parametrów podane parametry i tworzy listę w postaci [id, a, b].

Klasyfikator:

Mój klasyfikator do obliczenia prawdopodobieństwa należenia do którejś z klas wykorzystuje wzór:

$$P(C_k|x) = \frac{P(C_k) * P(x_1|C_k) * P(x_2|C_k)}{P(x)}$$

Gdzie x_1 i x_2 to dwa wybrane parametry

- bayes() – przyjmuje dane i podaje parametry wykorzystując funkcje likelihood() i get_means_variances()
- likelihood() – oblicza prawdopodobieństwo warunkowe za pomocą wzoru:

$$P(x_k|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} * exp\left(\frac{-(x_k - \overline{x_k})^2}{2\sigma_k^2}\right)$$

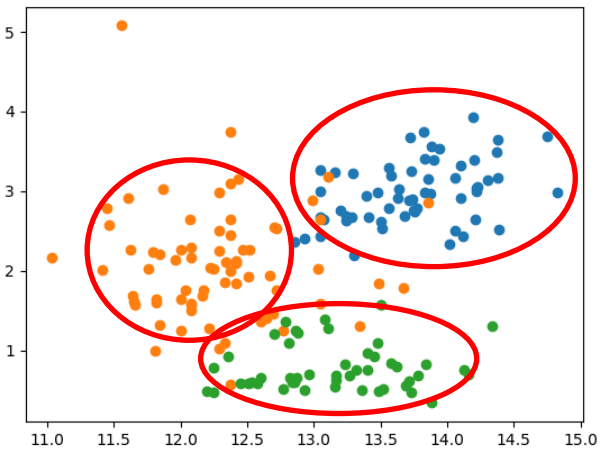
- calc_class_probs() – oblicza prawdopodobieństwo należenia do klasy na podstawie ilości elementów
- get_means_variances() – oblicza średnie i wariancje z danych

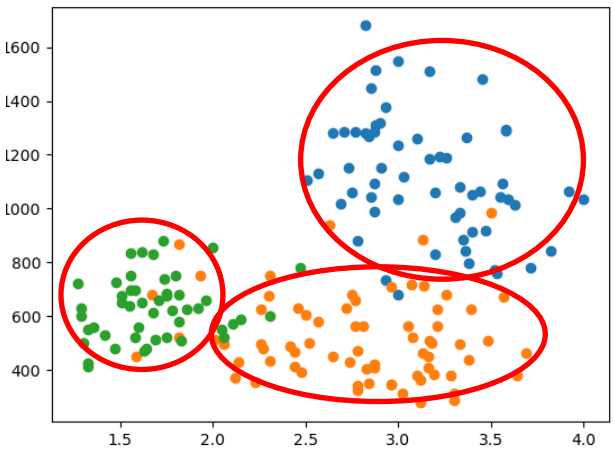
Walidacja i testowanie:

- check_prediction() – przewiduje do jakiego zbioru należy x, bazując na prawdopodobieństwie z liczby elementów, średniej i wariancji
- cross_validate – dzieli test na n bloków i znajduje najlepsze parametry dla zbioru
- make_test() – wykonuje test na zbiorze test ucząc się na zbiorze train i zwraca skuteczność dla każdej klasy

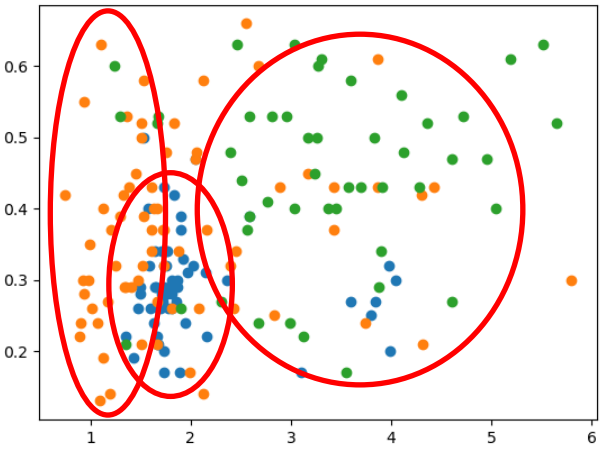
Wyniki:

Best parameters: 0, 6 Accuracy 0: 93.88% Accuracy 1: 94.34% Accuracy 2: 70.00% Total accuracy: 87.32%	Best parameters: 0, 6 Accuracy 0: 97.78% Accuracy 1: 83.87% Accuracy 2: 100.00% Total accuracy: 92.25%	Best parameters: 0, 6 Accuracy 0: 100.00% Accuracy 1: 86.21% Accuracy 2: 94.12% Total accuracy: 92.96%	Best parameters: 0, 6 Accuracy 0: 93.33% Accuracy 1: 92.86% Accuracy 2: 80.49% Total accuracy: 89.44%	Best parameters: 0, 6 Accuracy 0: 93.88% Accuracy 1: 90.74% Accuracy 2: 84.62% Total accuracy: 90.14%
Best parameters: 11, 12 Accuracy 0: 95.56% Accuracy 1: 94.64% Accuracy 2: 73.17% Total accuracy: 88.73%	Best parameters: 0, 6 Accuracy 0: 100.00% Accuracy 1: 83.05% Accuracy 2: 97.50% Total accuracy: 92.25%	Best parameters: 11, 12 Accuracy 0: 93.48% Accuracy 1: 87.72% Accuracy 2: 97.44% Total accuracy: 92.25%	Best parameters: 0, 6 Accuracy 0: 97.87% Accuracy 1: 87.93% Accuracy 2: 86.49% Total accuracy: 90.85%	Best parameters: 11, 12 Accuracy 0: 73.58% Accuracy 1: 94.12% Accuracy 2: 92.11% Total accuracy: 85.92%





Jak zauważamy na powyższych testach, algorytm osiąga 85%-92% procentową skuteczność dla parametrów 0, 6 oraz 11, 12, co jest dobrym wynikiem. Jak wydać na wykresach dane przy tych parametrach układają się w dość łatwo wyróżnialne zbiory.



Przykładowo dla parametrów 1,7 nie da się wyznaczyć tak dobrze wyróżnialnych zbiorów, dlatego nie są to dobre parametry dla algorytmu.