ALGORYTMY SZTUCZNEJ INTELIGENCJI W PRZEMYŚLE 4.0

Autorzy
KONRAD HARASIMOWICZ 252934
KACPER KLUPŚ 252870
SZYMON CHORAŁA 252941

Prowadzący

dr inż. Radosław Idzikowski

Spis treści

1 Cel projektu

Github: https://github.com/Kacper314/OWW-AIP.git
Tematem projektu jest klasyfikacja gatunków pingwinów. Do klasyfikacji planujemy użyć trzech algorytmów:

- Algorytm KNN - najbliższych sąsiadów
- Perceptron wielowarstwowy MLP w dwóch formach tj.: zaimplementowany od początku oraz gotowy algorytm z bibliotek sklearn.

Końcowym etapem będzie porównanie, który algorytm daje najlepsze rezultaty dla konkretnych hiperparametrów oraz atrybutów klas.

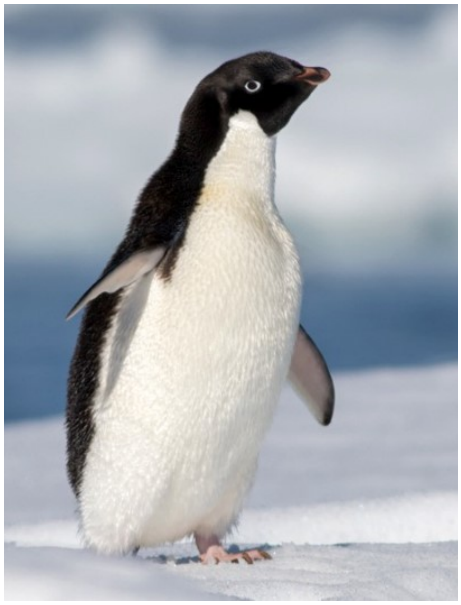
1.1 Zbiór

Jako zbiór danych wybrano Palmer penguins. Dane zostały zebrane w ramach badań mających na celu zbadanie zachowań żerowania pingwinów antarktycznych i ich związku ze zmiennością środowiska.

1.2 Liczba danych oraz klasy

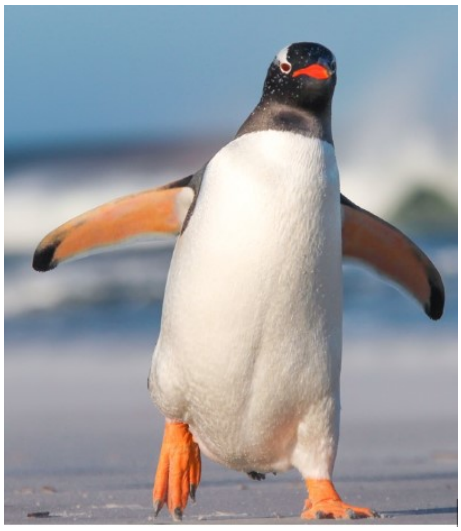
Zbiór zawiera 344 instancji (zbadanych pingwinów) oraz w jego skład wchodzą 3 klasy (gatunki):

1. Adélie (152 pingwinów)



Rysunek 1: Adélie

2. Gentoo (124 pingwinów)



Rysunek 2: Gentoo

3. Chinstrap (68 pingwinów)

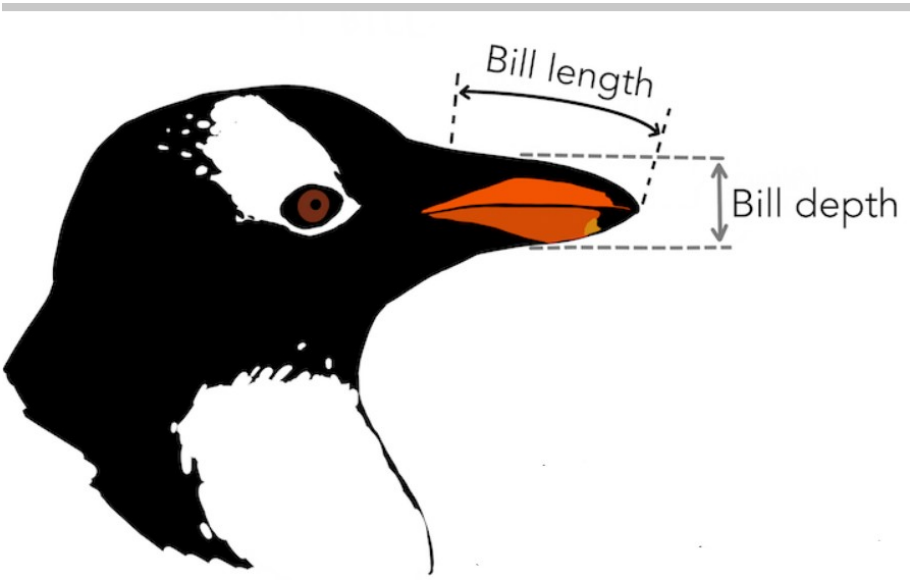
Rysunek 3: Chinstrap

1.3 Atrybuty

Każda z danych posiada 5 atrybutów. Są to:

1. Bill length [mm] - Długość dzioba
2. Bill depth [mm] - Grubość dziobu
3. Flipper length [mm] - Długość płetwy/ogona

4. Body mass [g] - masa ciała
5. Płeć



Rysunek 4: bill leght/ depth

1.4 Przykład danych

Kilka pierwszych wierszy z używanego zbioru:

species	island	culmen_le...	culmen_de...	flipper_len...	body_mas...	sex
Adelie	Torgersen	39.1	18.7	181	3750	MALE
Adelie	Torgersen	39.5	17.4	186	3800	FEMALE
Adelie	Torgersen	40.3	18	195	3250	FEMALE
Adelie	Torgersen	NA	NA	NA	NA	NA
Adelie	Torgersen	36.7	19.3	193	3450	FEMALE
Adelie	Torgersen	39.3	20.6	190	3650	MALE
Adelie	Torgersen	38.9	17.8	181	3625	FEMALE
Adelie	Torgersen	39.2	19.6	195	4675	MALE

Rysunek 5: Przykładowe dane

2 Podział danych

Wczytane dane zostały podzielone na dwa podzbiory: zbiór treningowy oraz zbiór testowy.

- Treningowy - na podstawie tego zbioru danych model uczy się klasyfikować.

- Testowy - na zbiorze testowym dokonujemy ostatecznego testu wybranego modelu wraz z odpowiednimi hiperparametrami.

Zbiory zostały podzielone procentowo. 80% danych to zbiór treningowy, a 20% to zbiór testowy. W programie podziału na zbiory dokonywała jedna funkcja, która zapisywała podzielone zbiory w oddzielnych plikach, na których zostały wykonywane testy programów.

2.1 Przygotowanie danych

W celu przeprowadzenia obliczeń na danych należy zamienić tekst na wartości liczbowe oraz przeprowadzić normalizację. Na liczby zamieniono płeć pingwinów, wyspę z której pochodzą oraz gatunek (klasę) przy pomocy poniższej funkcji.

```
def ZamianaNaLiczby_Plec(Zbior_Klas):
    species = []
    for i in range(len(Zbior_Klas)):
        if Zbior_Klas[i] == "MALE":
            species.append(0)
        elif Zbior_Klas[i] == "FEMALE":
            species.append(1)
        else:
            species.append(-1)

    return species
```

Normalizacja ma na celu przeskalowanie wartości do wartości w zakresie [0,1]. Pozwala to przekształcić dane w taki sposób, aby były bardziej porównywalne, eliminując wpływ różnic w skali wartości.
Proces normalizacji jest realizowany na podstawie poszukiwania maksymalnej i minimalnej wartości w zbiorze treningowym i testowym. Następnie każda wartość w tych zbiorach jest przekształcana według wzoru normalizacji min-max:

$$x_{\text{norm}} = \frac{x - \min}{\max - \min} \tag{1}$$

gdzie:

x_{norm} to znormalizowana wartość,
x to oryginalna wartość,
min to najmniejsza wartość w zbiorze danych (znaleziona wcześniej),
max to największa wartość w zbiorze danych (znaleziona wcześniej).

Normalizację danych przedstawia poniższa funkcja

```
def Normalizacja(Treningowy, Testowy):
    normalized_trainging = []
    normalized_test = []
    max = 0;
    min = 999999;
    #szukamy wart maksymalnych i minimalnych
    for i in range(len(Treningowy)):
        if Treningowy[i] > max:
            max = Treningowy[i]
        if Treningowy[i] < min:
            min = Treningowy[i]

    for i in range(len(Testowy)):
        if Testowy[i] > max:
            max = Testowy[i]
        if Testowy[i] < min:
            min = Testowy[i]

    for i in range(len(Treningowy)):
        normalized_trainging.append((Treningowy[i] - min)/(max-
  min))

    for i in range(len(Testowy)):
        normalized_test.append((Testowy[i] - min)/(max-min))

    return normalized_trainging, normalized_test
```

3 Wykorzystane metryki

W celu porównania ze sobą wszystkich modeli klasyfikujących użyto macierzy pomyłek oraz miar dobroci klasyfikatorów.
Macierz pomyłek przedstawia, które elementy zbioru danych zostały sklasyfikowane pozytywnie, a które negatywnie. Rozmiar macierzy jest zależny od ilości klas w zbiorze danych. Przykładowo dla zbioru dwu-klasowego możemy uzyskać cztery przypadki, które zostaną zapisane w macierzy:

- Prawdziwie pozytywna - TP (true positive) - przewidywana klasa jest pozytywna i faktycznie zaobserwowana również jest pozytywna,

- Prawdziwie negatywna - TN (true negative) - przewidywana klasa negatywna i faktycznie zaobserwoana również negatywna,
- Fałszywie pozytywna - FP (false positive) - przewidywana klasa pozytywna, a faktycznie zaobserwowana negatywna,
- Fałszywie negatywna - FN (false negative) - przewidywana klasa negatywna, a faktycznie zaobserwowana negatywna.

		Klasa rzeczywista	
		pozytywna	**negatywna**
Klasa predykowana	**pozytywna**	prawdziwie pozytywna (TP)	fałszywie pozytywna (FP)
	negatywna	fałszywie negatywna (FN)	prawdziwie negatywna (TN)

Rysunek 6: Macierz pomyłek 2x2

Obliczenie atrybutów macierzy pomyłek jest niezbędne dla zastosowania miar dobroci klasyfikatorów. Są to:

- Dokładność (accuracy) - czyli stosunek poprawnie sklasyfikowanych próbek to liczby wszystkich próbek. Dana jest wzorem:

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} \tag{2}$$

- Precyzja (precision) - mierzy stosunek przypadków prawdziwie pozytywnych do wszystkich przypadków sklasyfikowanych pozytywnie. Dana jest wzorem:

$$Precyzja = \frac{TP}{TP+FP} \tag{3}$$

- Czułość (recall, sensitivity) - mierzy proporcję prawdziwie pozytywnych przypadków, które zostały poprawnie wykryte przez klasyfikator, w stosunku do wszystkich rzeczywiście pozytywnych przypadków w zbiorze danych. Dana jest wzorem:

$$Czulosc = \frac{TP}{TP+FN} \tag{4}$$

- Swoistość (specificity) - mierzy stosunek prawdziwie negatywnych przypadków do wszystkich rzeczywiście negatywnych przypadków w zbiorze danych. Dana jest wzorem:

$$swoistosc = \frac{TN}{FP+TN} \tag{5}$$

- F1-Score - służy do pomiaru precyzji i czułości w tym samym czasie. Współczynnik F1 dany jest wzorem:

$$2 * \frac{precyzja * czulosc}{precyzja + czulosc} \tag{6}$$

  Im wartość jest bliższa jedynki tym lepiej to świadczy o algorytmie klasyfikującym. Wynik 1 oznacza również idealną czułość oraz precyzję.

W programie macierz pomyłek oraz miary dobroci klasyfikatorów zostały liczone poprzez wykorzystanie funkcji dostępnych w bibliotekach oraz zaimplementowanych.

4 Metody i badania

4.1 K najbliższych sąsiadów

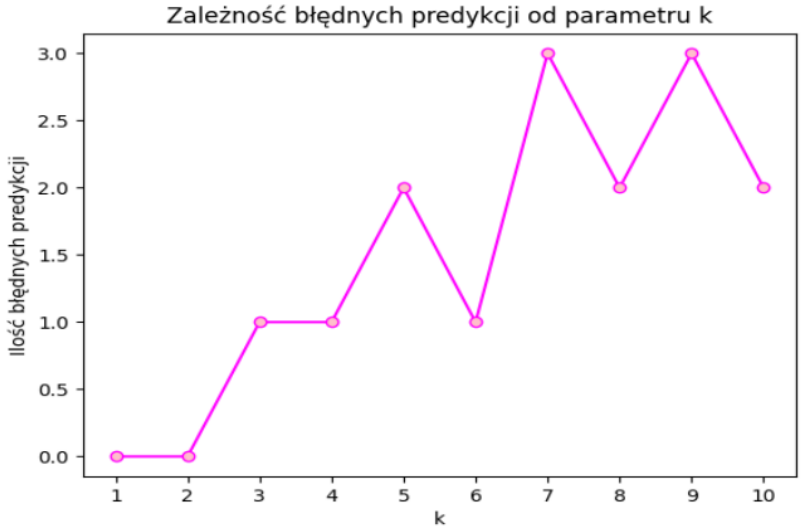
Algorytm k najbliższych sąsiadów (KNN) - należy do grupy algorytmów leniwych. Oznacza to, że metoda ta nie tworzy wewnętrznej reprezentacji danych uczących, a szuka rozwiązania w momencie pojawienia się zbioru testującego/walidacyjnego. Przechowuje wszystkie dane uczące, według których wyznacza odległość od wzorca testowego. Jak sama nazwa wskazuje algorytm KNN wyznacza k sąsiadów do których badane wejścia ma najbliżej według wybranej metryki (w tym przypadku Euklidesowej danej wzorem)

$$d(x,y) = \sqrt{\sum_{k=1}^{n} (y_i - x_i)^2} \tag{7}$$

Odległości są liczone dla każdej próbki ze testowego względem danych ze zbioru treningowego. W ten sposób zostaje utworzona tablica danych posortowana rosnąco aby łatwo sprawdzić najmniejsze obliczone odległości. Ostatnim etapem algorytmu jest klasyfikacja danej próbki. Czyli przypisanie jej odpowiedniej klasy w zależności od ilości najbliższych sąsiadów z danej klasy. Do sprawdzenia efektywności algorytmu zostały użyte funkcje opisane w paragrafie [4] Jedynym parametrem mającym wpływ na wynik jest parametr określający ilość sąsiadów - "k".

4.1.1 Parametr k

Dla zbioru testowego, sprawdzono precyzję algorytmu z różnymi parametrami k. Zakres wyboru k należy do zbioru [1,10]. Na poniższym wykresie zaprezentowano ilość błędnych predykcji w zależności od parametru k.



Rysunek 7: Wykres błędów w zależności od parametru k

Dla aktualnego zbioru, najlepszym parametrem k okazała się liczba 1.

4.1.2 Wyniki dla zbioru testowego

Następnie dla zbioru testowego sprawdzono podstawowe metryki opisane w sekcji 3:

Tabela 1: Wyniki dla różnych parametrów k

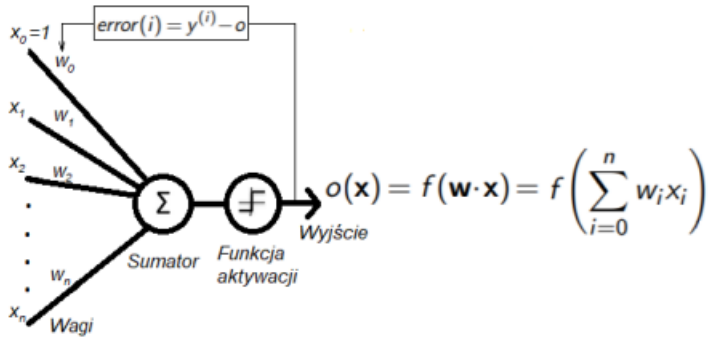
	Sensitivity	Specifitcity	Precision	Accuracy	F1
Dla k=2	1	1	1	1	1
Dla k=3	0.99	0.99	0.99	0.99	0.99
Dla k=5	0.97	0.99	0.97	0.98	0.97
Dla k=7	0.96	0.98	0.96	0.97	0.96
Dla k=9	0.96	0.98	0.96	0.97	0.96

Algorytm w znacznej większości przypadków dobrze sklasyfikował dane. W zależności od parametru k, algorytm najczęściej mylił się predykując, że dane odpowiadają pingwinowi Chinstrap, w momencie kiedy faktycznym pingwinem był Adelie

4.2 Perceptron Rosenblatta

Następnym realizowanym algorytmem nauczania maszynowego jest perceptron. Zadaniem perceptronu jest naśladowanie pojedynczego neuronu z mózgu. Perceptron odbiera wiele sygnałów

wejściowych i jeśli suma tych sygnałów przekracza pewien próg to perceptron zwraca sygnał lub pozostaje cichy. Algorytm perceptronu polega na uczeniu się wag sygnałów wejściowych w celu narysowania liniowej granicy decyzyjnej, która pozwala nam rozróżniać dwie liniowo oddzielone klasy.



Rysunek 8: Model perceptronu

Podstawowymi składnikami perceptronu są:

1. Warstwa wejściowa: Warstwa wejściowa składa się z jednego lub więcej neuronów wejściowych, które odbierają sygnały wejściowe ze świata zewnętrznego lub z innych warstw sieci neuronowej.

2. Wagi: Każdy neuron wejściowy jest powiązany z wagą, która reprezentuje siłę połączenia między neuronem wejściowym a neuronem wyjściowym.

3. Bias: Do warstwy wejściowej dodawany jest termin bias, aby zapewnić perceptronowi dodatkową elastyczność w modelowaniu złożonych wzorców w danych wejściowych.

4. Funkcja aktywacji: Funkcja aktywacji określa wartość wyjściową perceptronu na podstawie sumy ważonej danych wejściowych i członu bias. Funkcją zastosowaną w projekcie jest sigm.

5. Wyjście: Wyjściem perceptronu jest pojedyncza wartość binarna, 0 lub 1, która wskazuje klasę lub kategorię, do której należą dane wejściowe.

Algorytm perceptronu uczy się wag poprzez podanie sygnałów wejściowych (posiadających odpowiednie atrybuty) w celu wyznaczenia liniowej granicy decyzyjnej.

4.2.1 Działanie algorytmu

Przykładowe działanie algorytmu:

1. Zainicjowanie wag jako zero

2. Dla każdej próbki treningowej x[i]
   (a) Obliczany jest iloczyn skalarny pomiędzy wektorem wag a wektorem cech
   (b) Za pomocą funkcji signum wyznaczamy klasę
   (c) Jeśli wyznaczenie jest poprawne - przejdź do następnej próbki. Jeśli wyznaczenie jest błędne zaktualizuj wagi
3. Aktualizacja wag: $w_i(t+1) = w_i(t) + learningrate * (targetoutput - predictedoutput) * x_{i,j}$

4.2.2 Zastosowana technika OneVsAll

Perceptron działa dobrze dla klas, które są liniowo separowalne. W przypadku klasyfikowania danych posiadających więcej niż jedną klasę konieczne jest użycie techniki One vs All. Biorąc pod uwagę zbiór danych Palmer penguins, który posiada trzy klasy, konieczne jest stworzenie trzech klasyfikatorów. Każdy z nich musi zostać nauczony, aby realizował klasyfikację innej klasy. Np. pierwszy klasyfikuje pingwiny Adelie, drugi Gentoo a trzeci Chinstrap. Następnie, aby przetestować bądź wyznaczyć klasę dla danych testowych należy dane te podać na wejście wszystkich trzech klasyfikatorów, a następnie wybrać klasę dla największej aktywacji.

4.2.3 Wyniki dla zbioru testowego

Po wyznaczeniu hiperparametrów i nauczeniu każdego z klasyfikatorów - podano na ich wejście zbiór testowy. Dla każdego z klasyfikatorów sprawdzono wartość aktywacji i na jej podstawie wyznaczono klasy. Otrzymane rezultaty:

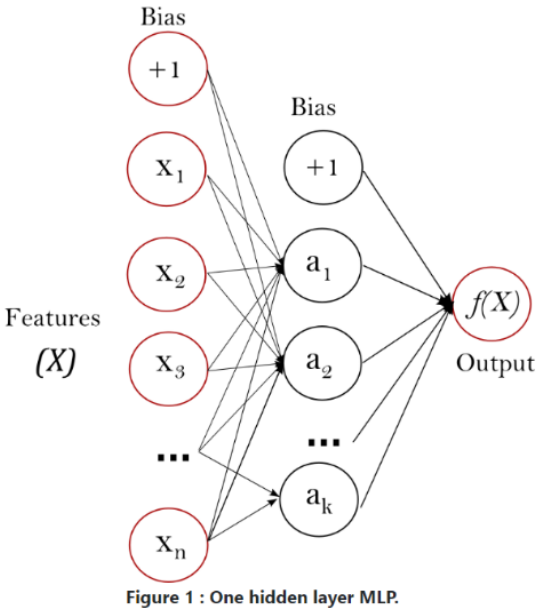
Tabela 2: Wyniki dla różnych parametrów epoki i lr

	Sensitivity	Specifitcity	Precision	Accuracy	F1
Dla epoki=10	0.99	0.99	0.99	0,99	0.99
Dla epoki=30	0.91	0.96	0.91	0.94	0.91
Dla epoki=100	0.93	0.96	0.93	0.95	0.93
Dla lr=0.1	0.93	0.96	0.93	0.95	0.93
Dla lr=0.01	0.93	0.96	0.93	0.95	0.93
Dla lr=0.001	0.93	0.96	0.93	0.95	0.93

Algorytm w znacznej większości przypadków dobrze sklasyfikował dane. Wyjątkiem są trzy błędne klasyfikacje. Dwukrotnie algorytm błędnie wytypował pingwina Adelie w momencie kiedy faktycznym był raz Gentoo a raz Chinstrap. Jednokrotnie błędnie wytypował Chinstrap kiedy prawdziwym był Gentoo.

4.3 Multi-Layer Perceptron

Perceptron i perceptron wielowarstwowy (MLP) to dwa rodzaje sztucznych sieci neuronowych wykorzystywanych w uczeniu maszynowym. Kluczowa różnica między nimi polega na ich architekturze i możliwościach.
Perceptron omówiony wcześniej jest najprostszą formą sieci neuronowej, składającą się z pojedynczej warstwy neuronów wejściowych połączonych z pojedynczym neuronem wyjściowym. Jest on używany głównie w przypadku problemów z klasyfikacją binarną, gdzie może nauczyć się rozdzielać punkty danych na dwie klasy w oparciu o liniową granicę decyzyjną.
MLP jest rozszerzeniem perceptronu, składającym się z wielu warstw neuronów, w tym warstwy wejściowej, jednej lub więcej warstw ukrytych i warstwy wyjściowej. MLP ma możliwość rozwiązywania złożonych problemów, w tym zarówno liniowo separowalnych, jak i nieliniowo separowalnych zadań. Warstwy ukryte w MLP pozwalają na uczenie się nieliniowych zależności i wyodrębnianie złożonych cech z danych wejściowych. Każdy neuron w MLP stosuje nieliniową funkcję aktywacji do swoich ważonych wejść, zazwyczaj funkcję sigmoidalną.
Wagi w MLP są uczone przy użyciu propagacji wstecznej, iteracyjnego algorytmu optymalizacji, który dostosowuje wagi w oparciu o błąd między przewidywanymi wyjściami a pożądanymi wyjściami.



Rysunek 9: Perceptron Wielowarstwowy

Trening perceptronu wielowarstwowego opiera się o metodę wstecznej propagacji błędu czyli algorytmu, który opowiada za ciągłą aktualizację, zmianę wag pomiędzy połączeniami aby uzyskać optymalne rozwiązanie. Algorytm został zrealizowany przy pomocy biblioteki

sklearn. Testowanie klasyfikacji dla zbioru danych odbywa się poprzez zmiany i dobieranie jak najlepszych hiperparametrów:

- max iter - ilość iteracji po zbiroze danych,
- hidden layers sizes - określa liczbę neuronów w warstwie ukrytej,
- activation - określa funkcję aktywacji:
  - identity - funkcja nie wprowadza nieliniowości i zwraca wartość wejścia bez żadnych zmian. W postaci liniowej $f(x) = x$
  - logistic - funkcja sigmoidalna.
  - tanh - funkcja tangensa hiperbolicznego przekształca wartości wejściowe do zakresu między -1 a 1.
  - relu - funkcja ta zwraca 0 dla wartości ujemnych i zachowuje wartości dodatnie bez zmian.
  - softmax - używany w warstwie wyjściowej perceptronu. Softmax przekształca wejścia na wartości z zakresu od 0 do 1, tak że sumują się do 1. Umożliwia to interpretację wyjścia jako rozkład prawdopodobieństwa przynależności do różnych klas.
- solver - wybór sposobu optymalizacji wag:
  - lbfgs - jest algorytmem optymalizacji gradientowej, który używa ograniczonej pamięci do przechowywania poprzednich iteracji. Algorytm iteracyjnie aktualizuje wagi, minimalizując funkcję straty, wykorzystując informacje z poprzednich kroków.
  - sgd - aktualizacja wag jest wykonywana dla każdego przykładu treningowego z osobna. Oznacza to, że gradient jest obliczany na podstawie pojedynczego przykładu, co prowadzi do bardziej losowego ruchu po przestrzeni wag. Solver ten sprawdza się sprawdza się dobrze w przypadku większych zbiorów danych.
  - adam - oblicza adaptacyjne tempo uczenia się dla każdej wagi. Ma zdolność do dostosowywania się do różnych temp uczenia się w różnych wagach i może skutecznie radzić sobie z różnymi typami zbiorów danych.

Wyniki dla MLP:

Tabela 3: Wyniki dla różnych parametrów Hidden layer sizes i lr

	Sensitivity	F1	Precision
Dla Hidden layer sizes=100	0.91	0.92	0.94
Dla Hidden layer sizes=1000	0.96	0.96	0.96
Dla Hidden layer sizes=10000	0.88	0.89	0.93
Dla lr=0.1	0.43	0.26	0.19
Dla lr=0.01	0.99	0.99	0.99
Dla lr=0.001	0.80	0.72	0.66

5 Wnioski i podsumowanie

1. Dla algorytmu k najbliższych sąsiadów, przy doborze parametru k=2, dla zbioru testowego uzyskano dokładność równą 100%. Zwiększając parametr k, uzyskano gorsze wyniki. Przyczyną może być fakt, iż gatunek Chinstrap oraz Adelie, dla części osobników posiadają bardzo zbliżone parametry. Dodatkową przyczyną jest także mały zbiór treningowy (280 instancji), oraz fakt, iż dla gatunek Chinstrap jest najmniej licznym w całym zbiorze.

2. Dla perceptronu najlepsze wyniki uzyskano dla licznby epok równej 10. Następnie wraz ze wzrostem epok dokładność oraz precyzja modelu malała. Przyczyną może być np. przeuczenie modelu które występuje, gdy model zbyt dobrze dopasowuje się do danych treningowych, a potem słabo generalizuje się do nowych danych. Wzrost liczby epok może prowadzić do nadmiernego dopasowania do szumów w danych treningowych, co może pogorszyć wyniki na danych testowych.
   Wraz ze spadkiem parametru learning rate (dla stałej ilości epok). W tym przypadku oznacza to, że aktualizacje wag są zbyt małe, co może prowadzić do bardzo powolnego procesu uczenia. Model może potrzebować znacznie więcej epok, aby dostosować się do danych treningowych.

3. Dla algorytmu MLP, oraz doborze parametrów - learning rate, hidden layers size, uzyskano dokładność 98%.

4. Ze względu na to, iż zbiór Palmer Penguins nie jest zbiorem licznym (zawiera jedynie 344 instancji), nie można jednoznacznie stwierdzić na podstawie uzyskanych wyników o jakości wybranych metod. W zależności od rodzaju danych, kryterium oceniającego każdy z algorytmów, może okazać się dobry.

5. W przeprowadzonym teście porównawczym trzech różnych algorytmów klasyfikacyjnych: K-nearest neighbors (Knn), One-vs-Rest oraz Multi-Layer Perceptron (MLP), uzyskaliśmy bardzo obiecujące wyniki. Każdy z algorytmów, przy odpowiednim dostosowaniu parametrów, wykazał się wysoką skutecznością klasyfikacji. Szczególnie imponujące rezultaty uzyskaliśmy przy użyciu algorytmu Knn, gdzie wszystkie miary oceny klasyfikacji osiągnęły poziom 1, co odpowiada 100% skuteczności. W przypadku pozostałych dwóch algorytmów, One-vs-Rest oraz MLP, uzyskaliśmy bardzo wysoką skuteczność na poziomie 99%, co również jest rezultatem bardzo obiecującym.