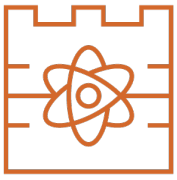
POLITECHNIKA KRAKOWSKA
im. Tadeusza Kościuszki
Wydział Inżynierii Materiałowej i Fizyki
Katedra Fizyki



Kierunek studiów: Fizyka Techniczna
Specjalność: Modelowanie Komputerowe

STUDIA STACJONARNE

PRACA DYPLOMOWA

INŻYNIERSKA

Marcin Kubański

135385

Przegląd metod analizy danych i uczenia maszynowego w zastosowaniach do zagadnień fizycznych

Review of data analysis and machine learning methods in applications to the problems in physics

Promotor pracy dyplomowej:
dr Radosław Kycia

Recenzent pracy dyplomowej:
prof. dr hab. Włodzimierz Wójcik

Kraków, rok akad. 2022/23

Abstrakt

W pracy przeprowadzono przegląd algorytmów uczenia maszynowego w zastosowaniach do zagadnień fizycznych. Jako przykład wybrano zbiór danych dotyczący orbit asteroid i ich ryzyka uderzenia w Ziemię. Dane zostały umieszczone na platformie Kaggle przez NASA. Wykorzystano środowisko Jupyter, język programowania Python oraz biblioteki takie jak Opendatasets, NumPy, Pandas, Matplotlib, SciPy, scikit-learn oraz Seaborn. Do przetwarzania danych wykorzystano wybrane algorytmy uczenia nienadzorowanego i nadzorowanego oraz określono, czy metody te są skuteczne w wykrywaniu zagrożeń ze strony asteroid. Ponieważ uczenie maszynowe zyskuje coraz większą popularność, była to dobra okazja do poszerzenia swoich umiejętności.

Abstract

The paper reviews machine learning algorithms in applications to physical issues. As an example, a dataset on the orbits of asteroids and their risk of hitting the Earth was chosen. The data was shared on the Kaggle platform by NASA. Jupyter environment, Python programming language and libraries such as Opendatasets, NumPy, Pandas, Matplotlib, SciPy, scikit-learn and Seaborn were used. Selected unsupervised and supervised learning algorithms were used in data processing and it was determined whether the methods were effective in detecting asteroid hazards. As machine learning gains more popularity, this was a good opportunity to enhance my skills.

Spis treści

Wstęp

0.1 Cel pracy

Celem pracy jest przegląd metod analizy danych i uczenia maszynowego w zastosowaniach do zagadnień fizycznych. Jako konkretny przykład wybrano zbiór danych asteroid niebezpiecznych dla Ziemi. Zbadano skuteczności tych metod w identyfikowaniu asteroid, które mogą stanowić zagrożenie dla Ziemi. Wybrane metody są tak uniwersalne, że z powodzeniem mogą być zastosowane do innych zagadnień fizyki oraz techniki.

0.2 Zakres pracy

Zakres pracy obejmuje krótki wstęp dotyczący astrofizyki, przedstawienie zbioru danych dotyczących asteroid niebezpiecznych dla Ziemi, w tym opis parametrów.

Następnie opisano algorytmy uczenia nienadzorowanego - wyjaśniono czym jest klastrowanie, algorytm k-średnich, skaler standardowy, analiza głównych składowych, analiza sylwetki, metoda łokciowa, kryterium oceny jakości klastrowania oraz model Gaussowski.

W dalszej części pracy, przedstawiono zagadnienia związane z uczeniem nadzorowanym, takie jak maszyna wektorów wspierających, perceptron, las losowy, drzewo decyzyjne, walidacja krzyżowa, tablica pomyłek (zwana również macierzą błędu) oraz przeszukiwanie siatki.

Opisano również krok po kroku jak został wykonany projekt od strony programistycznej, jakie oprogramowanie i biblioteki były użyte do analizy danych.

0.3 Metodyka pracy

Wykorzystując wybrany zbiór danych dotyczący asteroid niebezpiecznych dla Ziemi, przeprowadzono analizę danych za pomocą języka Python w środowisku programistycznym Jupyter. W celu przetworzenia danych, zastosowano różne metody uczenia nienadzorowanego i nadzorowanego. Aby przedstawić wyniki, przygotowano wykresy ilustrujące skuteczność różnych metod. Przedstawiono również wnioski dotyczące wykonanej wcześniej analizy.

Część I

Część Teoretyczna

Rozdział 1

Przygotowanie środowiska

Do analizy użyto środowiska **Jupyter** [1], które jest interfejsem służącym do wizualizacji wcześniej wprowadzonego kodu. Wszelkie obliczenia były wykonane za pomocą języka **Python** [2]. Podczas pisania projektu niezbędne były biblioteki: **Opendatasets** [3], **NumPy** [4], **Pandas** [5], **Matplotlib** [6], **SciPy** [7], **scikit-learn** [8] oraz **Seaborn** [9].

1.1 Jupyter

Jupyter [1] jest środowiskiem programowania, które pozwala na tworzenie i udostępnianie dokumentów z kodem, wykresami czy wyrażeniami matematycznymi. Jupyter to skrót od Julia, Python oraz R, ponieważ początkowo środowisko było przeznaczone do pracy z tymi językami programowania ale obecnie obsługuje ponad sto różnych języków. Zaletą Jupyter'a jest to, że pozwala na pracę w tzw. środowisku interaktywnym. Oznacza to, że można wprowadzać i uruchamiać kod, który zostanie wyświetlony w następnej komórce przez środowisko. Pozwala także łatwo testować i debugować różne fragmenty kodu [1].

1.2 Python

Python [2] jest jednym z najpopularniejszych języków programowania, znanym z łatwości użycia, czytelności kodu i bogatych bibliotek. Python jest szczególnie przydatny w analizie danych, ponieważ ma wiele bibliotek do przetwarzania i analizy danych, takich jak NumPy, Pandas i SciPy. Te biblioteki pozwalają na efektywne przetwarzanie tabelarycznych danych, obliczenia numeryczne i statystyki.

Język ten jest również popularny w uczeniu maszynowym i sztucznej inteligencji, dzięki bibliotekom takim jak scikit-learn, TensorFlow i Keras. Te biblioteki pozwalają na szybkie i łatwe stworzenie modeli uczenia maszynowego i wykorzystanie ich do rozwiązywania różnych problemów.

1.3 Wykorzystane biblioteki

- **Opendatasets** [3] służy do importowania danych z platformy Kaggle oraz dysku Google.
- **NumPy** [4] jest biblioteką, która służy do obliczeń numerycznych i pracy z tablicami wielowymiarowymi. Zawiera ona wiele funkcji matematycznych i algorytmów, takich jak obliczanie statystyk, algebra liniowa i wiele innych. Dzięki wykorzystaniu tablic wielowymiarowych NumPy umożliwia efektywne przetwarzanie danych, przyspieszając działanie skryptów i ułatwiając pracę z danymi numerycznymi w Pythonie.

- Kolejną biblioteką jest **Pandas** [5], która służy do analizowania danych. Zawiera ona szereg narzędzi i struktur danych, takich jak DataFrame i Series, które pozwalają na łatwe i efektywne przetwarzanie danych.
- **Matplotlib** [6] to biblioteka służąca do tworzenia różnego rodzaju wykresów i grafik. Biblioteka ta jest bardzo rozbudowana i pozwala na tworzenie wykresów 2D i 3D, a także różnego rodzaju wykresów statystycznych.
- **SciPy** [7] zapewnia obsługę algorytmów do optymalizacji, integracji, równań algebraicznych, różniczkowych, statystyki i wielu innych klas problemów.
- Następnym pakietem jest **scikit-learn** [8] - moduł zbudowany na bazie SciPy. Umożliwia przeprowadzanie algorytmów uczenia maszynowego - klasyfikacji, regresji i klastrowania. Zapewnia również różne narzędzia do dopasowania modelu, wstępnego przetwarzania danych, wyboru modelu, oceny modelu i wiele innych narzędzi.
- Ostatnim z użytych pakietów jest **Seaborn** [9] - zbudowany na popularnej bibliotece wizualizacji danych Matplotlib. Zapewnia tworzenie niestandardowych wykresów w języku Python.

Rozdział 2

Asteoridy i ich fizyka

Astrofizyka jest dziedziną nauki, która zajmuje się badaniem fizycznych aspektów kosmosu. Zajmuje się ona m.in. badaniem planet, gwiazd, galaktyk i całego Wszechświata. Astrofizycy próbują zrozumieć jak powstają ciała niebieskie oraz jakie zjawiska zachodzą we Wszechświecie. Do badań astrofizycznych wykorzystuje się różne narzędzia np. teleskopy czy sondy kosmiczne. Jest to ważna dziedzina nauki, ponieważ pomaga lepiej zrozumieć Wszechświat i jego historię.

Dla porównania, **astronomia** jest nauką opartą głównie na obserwacjach, która zajmuje się badaniem obiektów i zjawisk poza Ziemią, takich jak planety, gwiazdy, galaktyki i mgławice. Astronomowie wykorzystują teleskopy, radioteleskopy i satelity, aby zbierać dane i poznawać strukturę, skład i historię Wszechświata [10].

2.1 Definicja asteroid

Asteroidy to małe, skaliste ciała, które krążą po orbicie wokół Słońca. Mogą być resztkami po nieukończonym procesie tworzenia planety i pozostałościami po procesie tworzenia się Układu Słonecznego. Większość z nich znajduje się w pasie asteroid między Marsem a Jowiszem, choć niektóre mogą krążyć w innych obszarach Układu Słonecznego. Asteroidy są ważnym obiektem badań dla astrofizyków, ponieważ mogą one dostarczyć informacji o procesach zachodzących w Układzie Słonecznym w czasach jego powstawania. Przykład takiego obiektu możemy zobaczyć na rysunku 2.1.

Rysunek 2.1: Zdjęcie asteroidy [11].

2.2 Historia zderzeń asteroid z Ziemią

W naszej historii zapisały się asteroidy, które uderzyły w Ziemię. Jednym z najbardziej znanych jest uderzenie meteorytu na półwyspie Jukatan w Meksyku. W okolicach dzisiejszego miasta Chicxulub, znajdują się pozostałości krateru o takiej samej nazwie, który powstał około 65 milionów lat temu. Wydarzenie to przyczyniło się wyginięcia dinozaurów [12].

Innymi ważnymi wydarzeniami, które były szeroko opisywane w mediach są: zdarzenie tunguskie (rok 1908) oraz meteor z Czelabińska (rok 2013). Pierwsze z nich jest największym w historii, asteroida rozpadła się około 10 kilometrów nad ziemią, jej wpływ jest porównywalny do wybuchu 10-20 megaton trotylu. Drugie wydarzenie, które miało miejsce w Czelabińsku, jest najnowszym znanym incydentem tego typu. Eksplozja spowodowała falę uderzeniową na obrzeżach miasta, w wyniku której rannych zostało ponad 1700 osób [12].

Asteroidy to niebezpieczne i nie do końca zbadane obiekty, naukowcy po dziś dzień badają i unowocześniają technologie mogące wykrywać asteroidy. Jedną z nich jest **Sentry** [13] - należący do NASA zaawansowany system. Sentry to wysoce zautomatyzowany system monitorowania kolizji, który nieustannie skanuje najnowszy katalog asteroid pod kątem możliwości przyszłego zderzenia z Ziemią w ciągu najbliższych 100 lat. Za każdym razem, gdy zostanie wykryte potencjalne zagrożenie, zostanie ono przeanalizowane, a wyniki natychmiast opublikowane, z wyjątkiem nietypowych przypadków, w których szukamy niezależnego potwierdzenia [13].

2.3 Opis parametrów asteroid

Parametry orbity to informacje takie jak: oś orbity, jej ekscentryczność, nachylenie, długość geograficzna węzła, Argument Peryhelium, średnia anomalia, odległość peryhelium, odległość aphelium, okres orbitalny, minimalna odległość przecięcia orbity, odniesienie orbitalne, wielkość asteroidy, klasyfikacja. Te parametry opisują położenie i ruch asteroidy w przestrzeni kosmicznej.

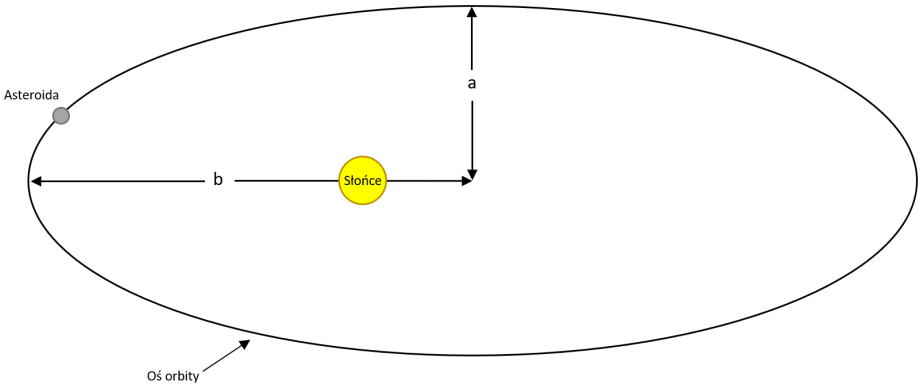
- **Orbita asteroidy** jest określona przez sześć parametrów (zwanymi elementami orbitalnymi) w początkowym czasie - zwanym epoką. Orbita jest wyznaczana przez równania ruchu, które modelują siły, jakie mają działać na obiekt w danym czasie. Siły te to przede wszystkim grawitacyjne przyciąganie Słońca, planet, Księżyca i 16 największych planetoid. Obliczanie pozycji obiektu w różnych momentach czasu może odbywać się przez numeryczne całkowanie, propagację lub równania ruchu [13].

- **Oś orbity** [14] to linia wokół której asteroida krąży wokół Słońca (rysunek 2.2). Oś ta jest zdefiniowana przez środek masy układu Słońce-asteroida i jest prostopadła do płaszczyzny orbity asteroidy.

- **Nachylenie osi orbity** [14] to kąt pomiędzy płaszczyzną orbity a **płaszczyzną ekliptyki**, która jest płaszczyzną, wokół której krążą planety w Układzie Słonecznym. Płaszczyzna ekliptyki przechodzi przez centra Ziemi i Słońca oraz jest prostopadła do osi obrotu Ziemi.

- **Ekscentryczność orbity** [14] jest parametrem określającym stopień „elipsoidalności" orbity, gdzie elipsa jest skrajnym przypadkiem ekscentryczności równej 1, a która jest określona wzorem:

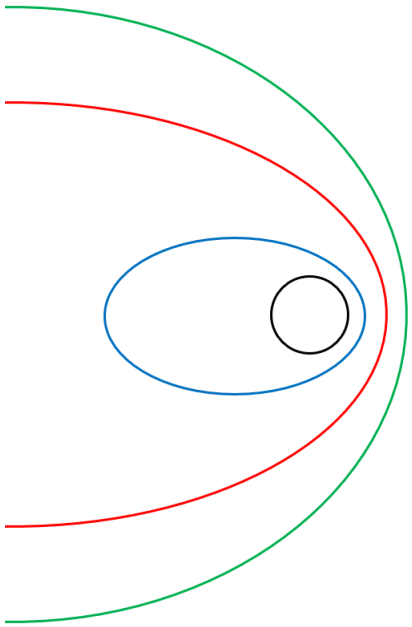
$$e = \sqrt{1 - \frac{b^2}{a^2}}, \tag{2.1}$$



Rysunek 2.2: Asteroida krążąca wokół Słońca po swojej orbicie - ekscentryczność orbity, a jest małą półosią orbity, natomiast b to duża półoś orbity.

gdzie a jest małą półosią orbity, a b jest dużą półosią orbity (rysunek 2.2).

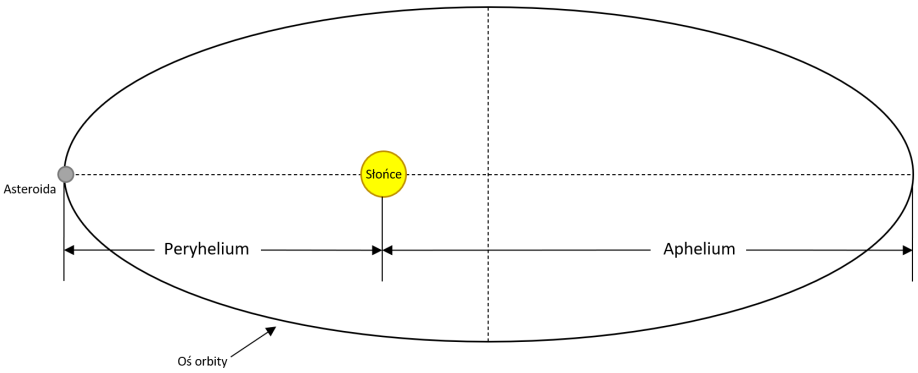
Ekscentryczność może być orbitą kołową gdzie $e = 0$, orbitą eliptyczną gdzie $0 < e < 1$, orbitą paraboliczną gdzie $e = 1$ lub orbitą hiperboliczną gdzie $e > 1$. Przykład ekscentryczności znajduje się na rysunku 2.3.



Rysunek 2.3: Ekscentryczność orbity - orbita kołowa (kolor czarny), orbita eliptyczna (kolor niebieski), orbita paraboliczna (kolor czerwony), orbita hiperboliczna (kolor zielony).

- **Argument peryhelium** [15] to kąt między punktem peryhelium (punktem najbliższym Słońcu na orbicie asteroidy) a węzłem równikowym (jeden z dwóch punktów na orbicie ciała niebieskiego, w których orbita przecina płaszczyznę równika). Jest to kolejny z parametrów orbitalnych określających położenie asteroidy w przestrzeni.
- **Średnia anomalia** [15] jest kątem między punktem początkowym orbity a punktem, który ma taką samą odległość od osi orbity jak obiekt.
- **Odległość peryhelium** [15] to odległość między asteroidą a punktem peryhelium

(punktem najbliższym Słońcu) w momencie przechodzenia przez niego (rysunek 2.5).

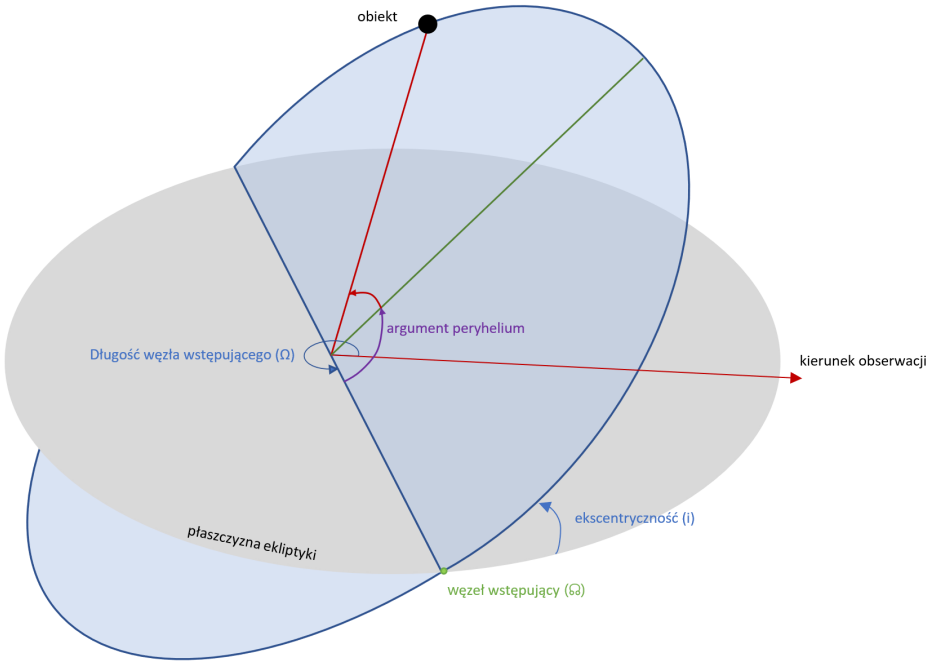


Rysunek 2.4: Asteroida krążąca wokół Słońca po swojej orbicie - przedstawienie parametrów aphelium oraz peryhelium.

- Przeciwieństwem, a za razem następnym parametrem jest **dystans aphelium** [16]. Jest to odległość między asteroidą, a punktem aphelium (punkt najdalej od Słońca) w momencie przechodzenia przez niego (rysunek 2.5).
- **Okres orbitalny** [16] asteroidy to czas potrzebny na przebycie pełnego obiegu wokół Słońca.
- **Odległość przecięcia orbity** [16] to odległość między asteroidą a punktem, w którym jej orbita przecina orbitę innego ciała niebieskiego. Minimalna odległość przecięcia orbit asteroid może wynosić kilka milionów kilometrów. W przypadku bliższych Ziemi obiektów minimalna odległość przecięcia orbity jest znacznie mniejsza, może wynosić kilkaset tysięcy kilometrów.
- **Orbitalny układ odniesienia** [16] to system punktów, który posługuje się układem współrzędnych do określenia pozycji i ruchu asteroidy w przestrzeni kosmicznej. W większości przypadków odniesienie orbitalne asteroidy jest zsynchronizowane z odniesieniem orbitalnym Ziemi, co oznacza, że jego oś obrotu jest zorientowana w kierunku osi obrotu Ziemi. Jest to ważne dla celów nawigacyjnych i śledzenia pozycji asteroidy w przestrzeni kosmicznej.
- **Średnia anomalia** [16] jest kątem między linią poprowadzoną od Słońca do peryhelium, a linią poprowadzoną od Słońca do punktu poruszającego się po orbicie w jednolitym tempie odpowiadającym okresowi obrotu asteroidy. Wzór wygląda następująco:

$$M = n(t - t_p), \tag{2.2}$$

  gdzie M to średnia anomalia, t to moment czasu, dla którego liczymy anomalię, t_p to moment przejścia ciała przez peryhelium, n to ruch średni równy $\frac{2\pi}{T}$ (T to okres orbitalny).
- **Wielkość absolutna** [16] asteroidy to wielkość wizualna, jaką zarejestrowałby obserwator, gdyby asteroida znajdowała się w odległości 1 jednostki astronomicznej i 1 jednostki astronomicznej od Słońca oraz przy zerowym kącie fazowym.
- **Długość węzła** [16] to kąt w płaszczyźnie ekliptycznej między osią płaszczyzny odniesienia, a linią przechodzącą przez węzeł wstępujący.



Rysunek 2.5: Przykład długości węzła wstępującego, oznaczony jest literą Ω - opracowanie własne na podstawie [17].

Rozdział 3

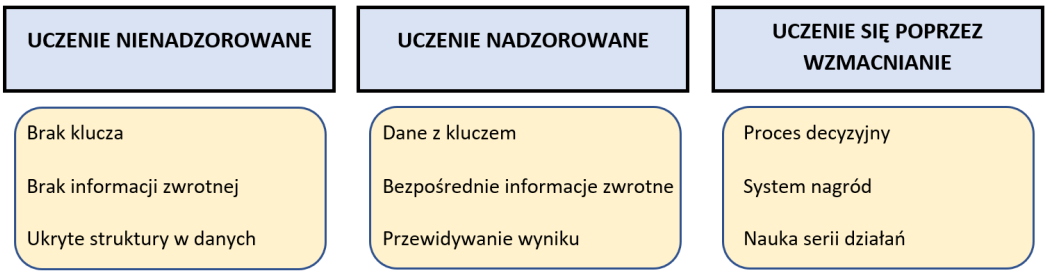
Uczenie maszynowe

3.1 Definicja uczenia się

Uczenie maszynowe jest dziedziną informatyki, która wykorzystuje algorytmy umożliwiające komputerom uczenie się na podstawie otrzymanych danych. Taka maszyna doskonali analizę danych oraz zdobywa doświadczenie - naśladuje sposób myślenia człowieka.

3.2 Rodzaje uczenia maszynowego

Istnieją trzy główne rodzaje uczenia maszynowego: uczenie nienadzorowane (opisane w rozdziale 4), uczenie nadzorowane (opisane w rozdziale 5) oraz uczenie się poprzez wzmacnianie. Pierwsze dwa rodzaje uczenia maszynowego zostały przedstawione w kolejnych rozdziałach pracy. Ostatni rodzaj uczenia się obejmuje szkolenie modelu do podejmowania decyzji w środowisku poprzez dostarczanie pozytywnych lub negatywnych informacji zwrotnych (więcej informacji można znaleźć w książce [18]). Podział uczenia maszynowego wraz z ich cechami przedstawiono na rysunku 3.1.



Rysunek 3.1: Podział uczenia maszynowego - opracowanie własne na podstawie [18].

Rozdział 4

Uczenie nienadzorowane

Uczenie nienadzorowane to rodzaj uczenia maszynowego, w którym algorytm uczenia nie otrzymuje klucza odpowiedzi ani nie jest nadzorowany przez "nauczyciela". Algorytm dopasowuje, analizuje i tworzy klucz dla danych wejściowych. Jest to pierwszy etap w procesie uczenia maszynowego, który pozwala na zrozumienie struktury danych i wybranie odpowiednich algorytmów uczenia do dalszego przetwarzania. Przykładem uczenia nienadzorowanego jest algorytm klastrowania, który na podstawie danych dotyczących klientów sklepu internetowego uczy się grup klientów o podobnych zachowaniach zakupowych.

4.1 Metody wstępnego przetwarzania

4.1.1 Normalizacja

Normalizacja to proces przekształcania danych wejściowych w taki sposób, aby ich wartości mieściły się w określonym zakresie. W uczeniu maszynowym, normalizacja jest często stosowana, aby dostosować dane wejściowe do algorytmu uczącego.

Przykładem normalizacji jest **skaler standardowy** (*ang. StandardScaler*) [19] - narzędzie, którego głównym celem jest przygotowanie danych do dalszego przetwarzania, tak aby dane po standaryzacji miały średnią 0 i odchylenie standardowe 1.

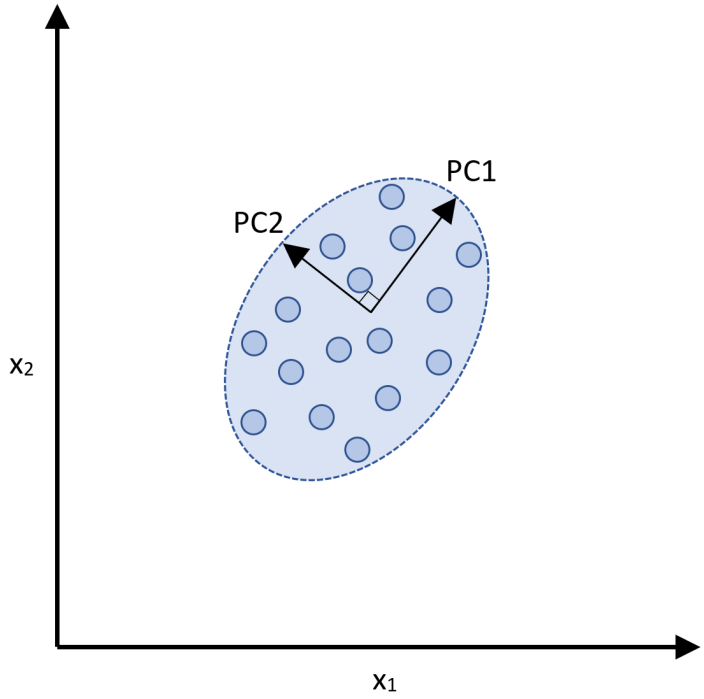
Przeskalowanie danych jest ważne, ponieważ różne cechy mogą mieć różne zakresy, co może prowadzić do problemów podczas treningu modelu. Na przykład, jeśli jedna cecha ma zakres od 0 do 100, a druga od 0 do 1 000 000, model może lepiej reagować na cechę z większym zakresem, co może prowadzić do nierównomiernego traktowania tych dwóch cech podczas treningu. Skaler standardowy pomaga temu zapobiec, przeskalowując dane tak, aby każda cecha miała taki sam zakres.

4.2 Metody transformacji

4.2.1 Analiza głównych składowych

Analiza głównych składowych (*ang. Principal Component Analysis, PCA*) [20] ma na celu znalezienie kierunków maksymalnej wariancji macierzy w danych wielowymiarowych i rzutuje dane na nową podprzestrzeń o równych lub mniejszych wymiarach niż oryginalna. Osie ortogonalne (główne składowe) nowej przestrzeni mogą być interpretowane jako kierunki maksymalnej wariancji przy założeniu ograniczenie, że nowe osie cech są do siebie ortogonalne (rysunek 4.1).

Metoda PCA jest szczególnie przydatna w przypadku dużych zbiorów danych o wysokiej liczbie wymiarów, ponieważ pozwala na znalezienie najważniejszych składowych danych



Rysunek 4.1: x_1 oraz x_2 to osie cech, a PC1 i PC2 są głównymi składowymi - opracowanie własne na podstawie [18].

i zredukowanie liczby wymiarów, co pozwala na lepsze zarządzanie danymi. Może być również stosowana do przeprowadzenia analizy skupień lub do wizualizacji danych w formie wykresów.

Aby przeprowadzić analizę głównych składowych, należy najpierw obliczyć korelacji danych lub macierz kowariancji.

Macierz korelacji [18] to nic innego jak przeskalowana wersja macierzy kowariancji. W rzeczywistości macierz korelacji jest identyczna z macierzą kowariancji obliczoną z cech standaryzowanych. Macierz korelacji jest macierzą kwadratową, która zawiera współczynnik korelacji (r Pearson'a), która mierzy liniową zależność między parami cech. Współczynniki korelacji zawierają się w przedziale od -1 do 1. Jeśli dwie cechy posiadają współczynnik korelacji równy $r = 1$ oznacza to, że jest to silna korelacja. Odwrotna sytuacja zachodzi, wówczas gdy współczynnik ten wynosi $r = -1$.

Współczynnik korelacji Pearsona obliczyć można jako kowariancję pomiędzy dwoma cechami x i y, podzieloną przez iloczyn ich odchyleń standardowych:

$$r = \frac{\sum_{i=1}^{n} \left[\left(x^i - \mu_x \right) \left(y^i - \mu_y \right) \right]}{\sqrt{\sum_{i=1}^{n} \left(x^i - \mu_x \right)^2} \sqrt{\sum_{i=1}^{n} \left(y^i - \mu_y \right)^2}} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}, \tag{4.1}$$

Tutaj μ oznacza średnią odpowiedniej cechy, σ_{xy} to kowariancja między cechami x i y, a σ_x oraz σ_y to odchylenia standardowe cech.

Macierz kowariancji jest macierzą kwadratową, która określa, jak bardzo różne zmienne są ze sobą skorelowane. Element w wierszu i i j kolumnie macierzy reprezentuje kowariancję między zmiennymi i oraz j. Elementy diagonalne macierzy są wariancjami poszczególnych zmiennych, a elementy poza przekątną są kowariancjami między różnymi zmiennymi.

W PCA tworzymy symetryczną macierz kowariancji o wymiarach $d \times d$, gdzie d to liczba wymiarów w zbiorze danych, przechowuje parzyste kowariancje pomiędzy różnymi cechami. Na przykład kowariancja między dwoma cechami x_j i x_k na poziomie populacji może być obliczona za pomocą następującego równania:

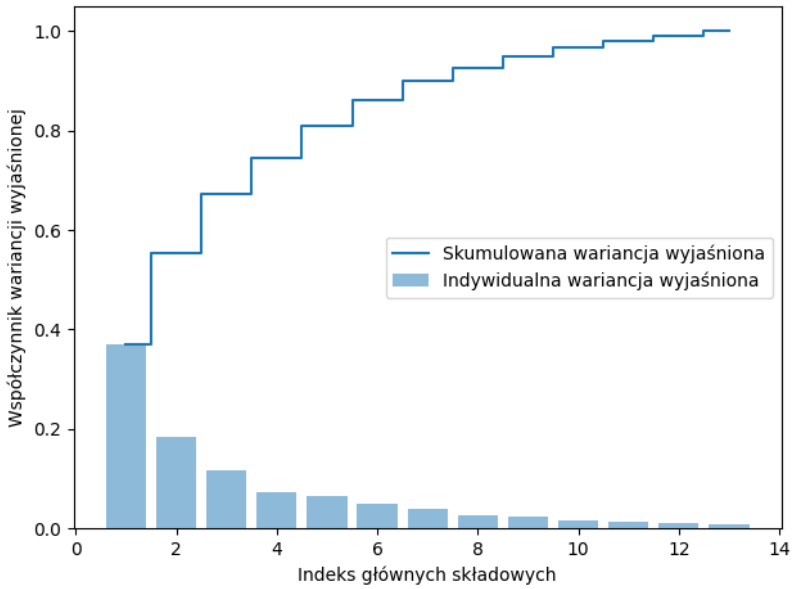
$$\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^{n} \left(x_j^i - \mu_j \right) \left(x_k^i - \mu_k \right). \tag{4.2}$$

gdzie μ_j oraz μ_k są średnimi cech j i k. Dane po normalizacji będą wynosiły 0. Gdy kowariancja jest dodatnia, oznacza to, że zmienne rosną lub maleją razem. Natomiast ujemna kowariancja wskazuje na to, że zmienne zmieniają się w przeciwnym kierunku.

W PCA, wektory własne macierzy kowariancji reprezentują główne składowe (kierunki maksymalnej wariancji) danych. Odpowiadające im wartości własne określają wielkość tych składowych.

Kolejnym krokiem w analizie jest otrzymywanie wartości własnych, będą one potrzebne do współczynnika wariancji wyjaśnionej. Współczynnik wariancji wyjaśnionej to po prostu ułamek wartości własnej λ_j i całkowitej sumy wartości własnych:

$$\text{współczynnik wariancji wyjaśnionej} = \frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j}. \tag{4.3}$$



Rysunek 4.2: Przykład analizy PCA.

Na rysunku 4.2 widzimy indywidualne wariancje wyjaśnione oraz skumulowane wariancje wyjaśnione. Dla przykładu cztery pierwsze składowe, które są skumulowane, odpowiadają współczynnikowi wariancji wyjaśnionej równemu w okolicy 80%.

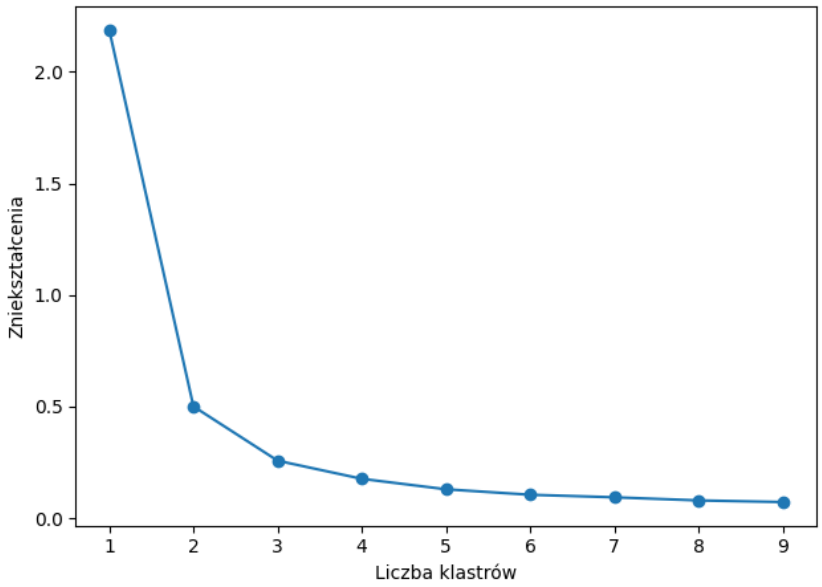
4.3 Klasteryzacja

Klasteryzacja (*ang. Clustering*) [21] jest techniką analizy danych eksploracyjnych, która pozwala nam zorganizować informacje w znaczące podgrupy (klastry) bez posiadania jakiejkolwiek wcześniejszej wiedzy o ich przynależności do grupy. Każdy klaster, który powstaje w trakcie analizy definiuje grupę obiektów, które mają pewien stopień podobieństwa, ale są bardziej zróżnicowane w stosunku do obiektów w innych klastrach.

4.4 Metody klastrowania

4.4.1 Metoda łokciowa

Metoda łokciowa (*ang. Elbow method*) [22] jest używana do określenia optymalnej liczby klastrów do użycia w algorytmie k-średnich. Robi się to poprzez dopasowanie modelu z różnymi liczbami klastrów, a następnie wykreślenie wynikowych błędów (SSE - suma kwadratów błędów) dla każdej liczby klastrów. Suma kwadratów błędów to suma kwadratów różnicy między wartościami rzeczywistymi, a przewidywanymi przez model. Im niższa suma błędów, tym lepsze dopasowanie modelu do danych.



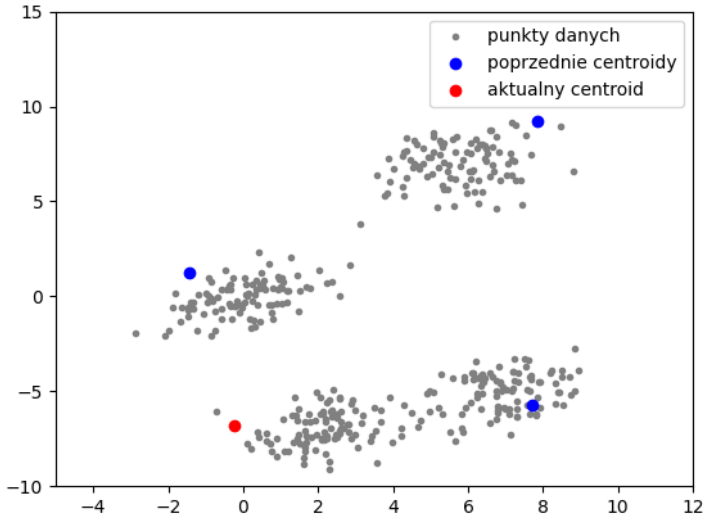
Rysunek 4.3: Przykład metody łokciowej.

"Łokieć" w wykresie to punkt, w którym błąd zmniejsza się znacząco z każdym dodatkowym klastrem i jest używany jako heurystyka do określenia optymalnej liczby klastrów (rysunek 4.3). Idea jest taka, że poza łokciem, spadek błędu będzie mniej znaczący, a więc dodawanie kolejnych klastrów może nie być warte czasu.

Metoda łokciowa jest szczególnie przydatna w przypadku dużych zbiorów danych, ponieważ pozwala uniknąć nadmiernego uczenia modelu i zapewnia lepszą wydajność. Punkt załamania jest punktem optymalnej liczby klastrów, dzięki czemu nie ma konieczności uczenia modelu dla każdej możliwej liczby klastrów. Może być również stosowana chociażby do identyfikowania optymalnej liczby składników w algorytmach redukcji wymiarów.

4.4.2 Algorytm k-średnich

Algorytm k-średnich (*ang. kmeans*) [23] to iteracyjna metoda używana do podziału zbioru danych na k klastrów. Działa poprzez losowe wybieranie k początkowych centroidów, a następnie iteracyjne przypisywanie każdego punktu danych do klastra, którego środek jest najbliżej niego. **Centroidy** to reprezentatywne punkty dla każdej z grup (klastrów) danych. Są one obliczane jako średnie wartości dla każdej zmiennej dla wszystkich punktów należących do danej grupy. Następnie algorytm aktualizuje centroidy, biorąc średnią ze wszystkich punktów danych przypisanych do każdego klastra. Proces ten jest powtarzany, aż centroidy się nie zmienią lub osiągnięta zostanie maksymalna liczba iteracji (rysunek 4.4).

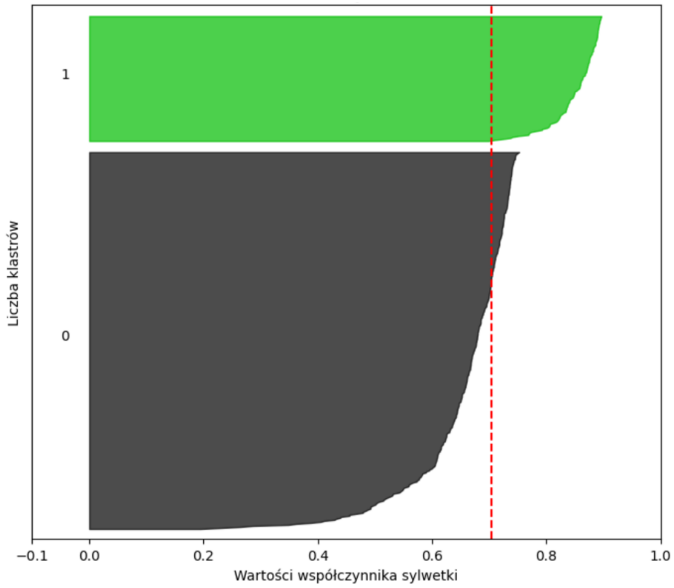


Rysunek 4.4: Przykład algorytmu k-średnich - opracowanie własne na podstawie [24].

Jest popularnym wyborem do tworzenia klastrów, ponieważ jest szybki, łatwy do wdrożenia i dobrze działa na dużych zbiorach danych. Ma jednak pewne ograniczenia. Na przykład wymaga od użytkownika określenia z góry liczby klastrów k, która może nie być wcześniej znana.

4.4.3 Analiza sylwetki

Analiza sylwetki (*ang. Silhouette analysis*) [25] to jedna z metod oceny jakości klastrowania danych. Polega ona na obliczeniu dla każdego punktu danych tzw. współczynnika sylwetki, który jest miarą jego przynależności do danego klastra.



Rysunek 4.5: Przykład analizy sylwetki. Czerwona linia przerywana oznacza współczynnik sylwetki. Tutaj współczynnik przy n_klastrów = 2 wynosi 0.7.

Współczynnik profilu jest obliczany jako różnica między średnią odległością punktu od innych punktów w tym samym klastrze i średnią odległością punktu od punktów w innych

klastrach. Im większa jest różnica tych dwóch wartości, tym lepiej punkt pasuje do swojego klastra. Na wykresie (rysunek 4.5) przyjmuje postać "nogawki od spodni".

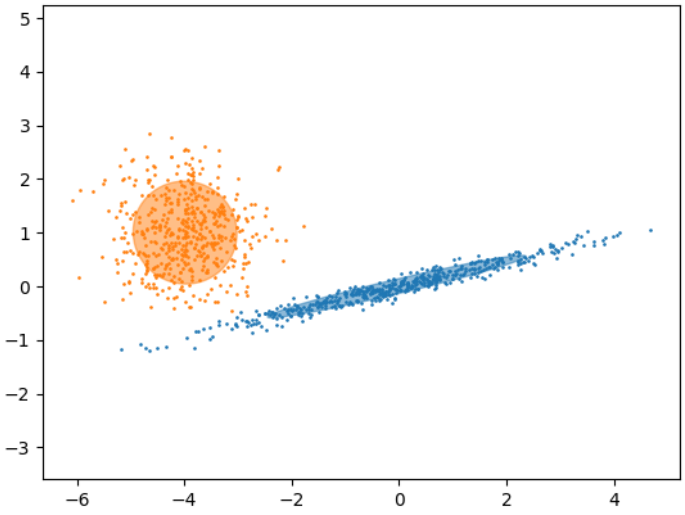
Aby obliczyć współczynnik dla całego zbioru danych, należy najpierw podzielić dane na klastry za pomocą jakiegoś algorytmu klastrowania. Następnie dla każdego punktu należy obliczyć jego współczynnik sylwetki.

Analiza sylwetki jest często stosowana jako narzędzie do wizualizacji jakości klastrowania oraz do porównywania różnych metod klastrowania. Może być również używana do doboru optymalnej liczby klastrów.

4.4.4 Model Gaussowski

Model Gaussowski (zwany również modelem Gaussa) [26] jest jednym z podstawowych modeli uczenia maszynowego, w szczególności w klasyfikacji. Model Gaussowski opiera się na założeniu, że dane są rozkładem Gaussa, czyli rozkładem normalnym o średniej (*ang. mean*) i odchyleniu standardowym (*ang. standard deviation*). Model Gaussowski przypisuje każdemu punktowi danych prawdopodobieństwo należenia do danej klasy, bazując na rozkładzie normalnym dla tej klasy.

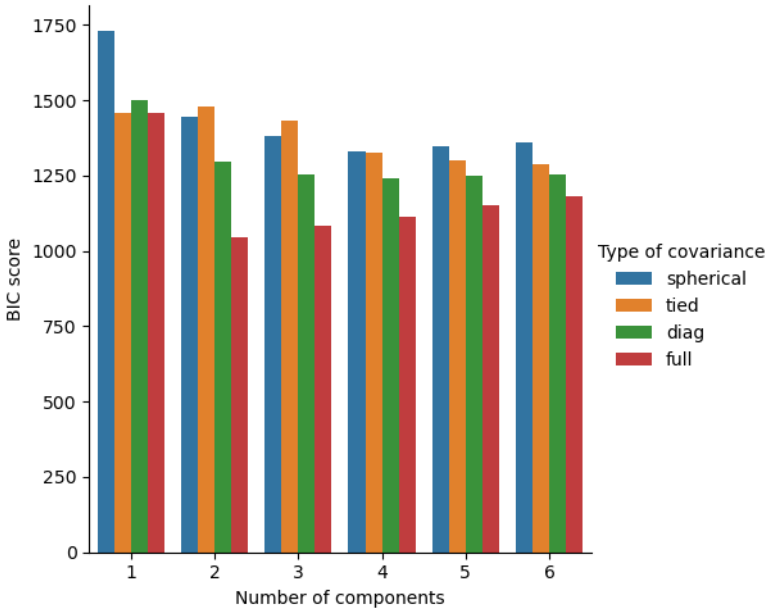
Model jest prosty w implementacji i działa dobrze w wielu sytuacjach, ale ma kilka ograniczeń. Przede wszystkim założenie o rozkładzie normalnym danych może być nierealistyczne w niektórych przypadkach, co może prowadzić do niepoprawnych wyników. Ponadto model Gaussowski nie radzi sobie dobrze z danymi dyskryminacyjnymi (jedna z klas jest znacznie mniej liczna niż druga). W takich przypadkach lepiej sprawdzają się inne modele, na przykład drzewo decyzyjne.



Rysunek 4.6: Przykład modelu Gaussowskiego z elementami Gaussowskimi. Przedstawiają one prawdopodobieństwo należenia punktu danych do danej klasy [27].

4.4.5 Kryterium oceny jakości klastrowania

Kryterium BIC (*ang. Bayesian Information Criterion*) [27] to jedno z narzędzi służących do oceny jakości modelu w uczeniu maszynowym. Jest to miernik, który ocenia, czy dany model dobrze opisuje dane oraz czy nie jest zbyt skomplikowany.



Rysunek 4.7: Przykład kryterium BIC - słupek z parametru *'full'* znajduje się w drugim komponencie [27].

W przedstawionym powyżej przykładzie (rysunek 4.7) mamy wyszczególnione cztery typy kowariancji:

- **Pełny** (ang. *full*) - każdy składnik ma swoją własną ogólną macierz kowariancji,
- **Związany** (ang. *tied*) - wszystkie składowe mają tę samą ogólną macierz kowariancji,
- **Diagonalny** (ang. *diag*) - każdy składnik ma swoją własną diagonalną macierz kowariancji,
- **Sferyczny** (ang. *spherical*) - macierz kowariancji jest diagonalna, a jej wartości na przekątnej są takie same.

Tam gdzie kolumna jest najmniejsza, tam jest najlepszy model. Z wykresu wynika, że najlepszy jest drugi komponent.

Kryterium BIC jest szczególnie przydatne do porównywania różnych modeli statystycznych i wyboru najlepszego z nich. Im niższa wartość BIC dla danego modelu, tym lepszy jest on w opisywaniu danych i tym mniej skomplikowany. Należy jednak pamiętać, że kryterium nie uwzględnia całkowitej złożoności danych ani nie zawsze dobrze ocenia modele nieliniowe. Dlatego też należy go stosować z rozwagą i łączyć go z innymi narzędziami oceny jakości modelu.

Rozdział 5

Uczenie nadzorowane

Uczenie nadzorowane to rodzaj uczenia maszynowego, w którym model uczy się na podstawie przykładów z danymi wejściowymi, podanymi już kluczami lub wynikami. W procesie uczenia nadzorowanego model próbuje znaleźć związek między danymi wejściowymi a wynikami, aby móc przewidzieć wyniki dla nowych danych. Przykładem uczenia nadzorowanego jest klasyfikator spamu, który na podstawie historii e-maili uczy się rozpoznawać wiadomości spamowe.

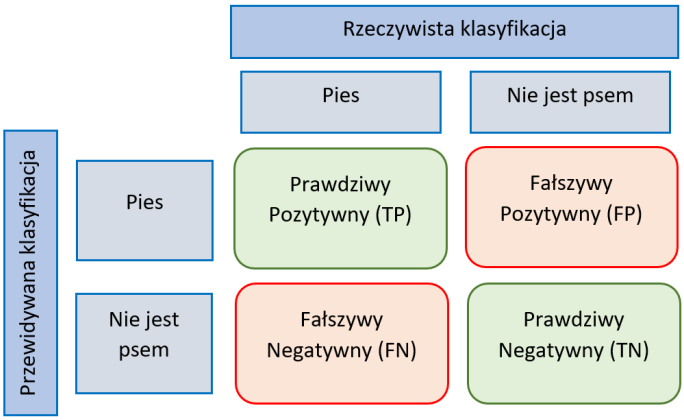
Aby skutecznie nauczyć model w procesie uczenia nadzorowanego, należy posiadać duży zbiór danych treningowych z etykietami lub wynikami. Model uczy się na podstawie tych danych i następnie jest testowany za pomocą zestawu danych testowych, aby ocenić jego skuteczność w przewidywaniu wyników dla nowych danych.

5.1 Miary oceny klasyfikacji

Często mając na myśli klasyfikację mówimy o **klasyfikacji binarnej**, czyli przypadek, gdzie przydzielamy dany obiekt do jednej z dwóch kategorii - 0 (prawda) lub 1 (fałsz). Jednakże, istnieją aż cztery możliwe wyniki klasyfikacji. Na przedstawionym przykładzie klasyfikowania psów (rysunek 5.1) możemy zauważyć podział na rzeczywistą klasyfikację i przewidywaną klasyfikację. Sprawdzamy czy dany obiekt jest psem czy nie, element trafia do jednego z czterech elementów tabeli - TP (prawdziwy pozytywny), FP (fałszywy pozytywny), FN (fałszywy negatywny) lub TN (prawdziwy negatywny).

- **Prawdziwy pozytywny** (TP) [18] - liczba pozytywnych przypadków, które zostały poprawnie sklasyfikowane jako pozytywne.
- **Prawdziwy negatywny** (TN) [18] - liczba negatywnych przypadków, które zostały błędnie sklasyfikowane jako pozytywne.
- **Fałszywy negatywny** (FN) [18] - liczba negatywnych przypadków, które zostały poprawnie sklasyfikowane jako negatywne.
- **Fałszywy pozytywny** (FP) [18] - liczba pozytywnych przypadków, które zostały błędnie sklasyfikowane jako negatywne.

Te cztery wartości zazwyczaj umieszcza się w tzw. **tablicy pomyłek** (*ang. confusion matrix*) [28].



Rysunek 5.1: Tablica pomyłek - opracowanie własne na podstawie [28].

Oceniając jakość modeli klasyfikacji w uczeniu maszynowym możemy je podzielić na dwie kategorie: parametry przedstawione graficznie (chociażby wcześniej przedstawiona tablica pomyłek) oraz parametry liczbowe do których należą:

- **Precyzja** (*ang. Precision*) [18] jest miarą dokładności klasyfikatora, w szczególności stosunku prawdziwych pozytywnych prognoz do całkowitej liczby pozytywnych prognoz. Wzorem opisującym precyzję jest:

$$\text{precyzja} = \frac{TP}{TP + FP}. \tag{5.1}$$

- **Dokładność klasyfikatora** (*ang. Accuracy*) [18] to liczba poprawnych prognoz dokonanych przez model jako ułamek całkowitej liczby prognoz. Oblicza się go, dzieląc liczbę prawdziwych pozytywów i prawdziwych negatywów przez całkowitą liczbę wykonanych prognoz:

$$\text{dokładność} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{5.2}$$

- **Czułość** (*ang. Recall*) [18] to liczba prawdziwie pozytywnych prognoz dokonanych przez model jako ułamek całkowitej liczby pozytywnych instancji w zbiorze danych. Oblicza się go, dzieląc liczbę wyników prawdziwie dodatnich przez sumę wyników prawdziwie dodatnich i fałszywie ujemnych:

$$\text{czułość} = \frac{TP}{P} = \frac{TP}{(TP + FN)}. \tag{5.3}$$

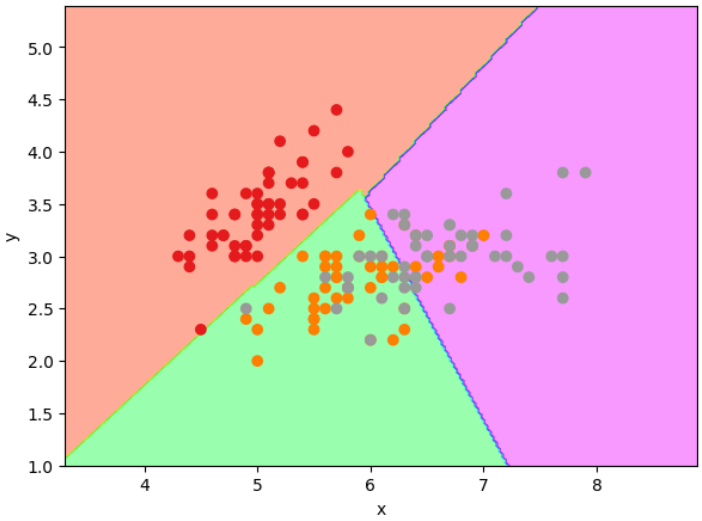
- **F1** [18] to wartość, która łączy w sobie precyzję i czułość. Oblicza się go, biorąc średnią harmoniczną precyzji i czułości, na podstawie wzoru:

$$F1 = 2 \cdot \frac{\text{precyzja} \cdot \text{czułość}}{\text{precyzja} + \text{czułość}}. \tag{5.4}$$

- **Nośnik** (*ang. Support*) [18] to ilość danych przypisana do zbioru testowego.

5.2 Maszyna wektorów wspierających

Maszyna wektorów wspierających (*ang. Support Vector Classification*, w skrócie SVC) [29] jest rodzajem nadzorowanego algorytmu uczenia maszynowego. Działanie polega na znalezieniu hiperpłaszczyzny w przestrzeni wielowymiarowej, która maksymalnie oddziela klasy.



Rysunek 5.2: Przykład liniowego SVC [29].

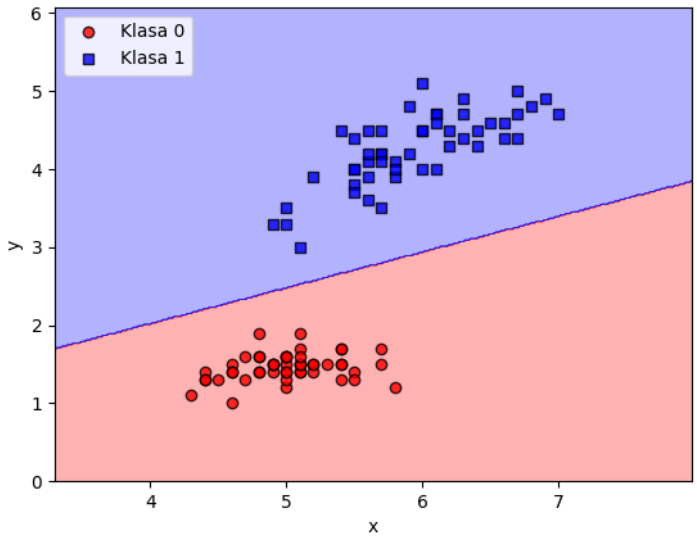
W kontekście klasyfikacji hiperpłaszczyzna jest granicą decyzyjną, która dzieli przestrzeń na dwie klasy. Na przykład w przestrzeni dwuwymiarowej hiperpłaszczyzna jest po prostu linią, która dzieli przestrzeń na dwa regiony, po jednym dla każdej klasy.

SVC jest często stosowana w przypadku dużych zbiorów danych, ponieważ jest w stanie efektywnie radzić sobie z dużą liczbą cech. Jest również odporna na szum w danych i ma dobrą zdolność generalizacji. Niestety, SVC może być wolna w przypadku bardzo dużych zbiorów danych i wymaga dobrej optymalizacji hiperparametrów, aby osiągnąć dobre wyniki.

5.3 Perceptron

Perceptron [30] to rodzaj sztucznej sieci neuronowej, która jest używana do klasyfikacji binarnej (podrozdział 5.1). W przypadku perceptronu, klasyfikacja binarna oznacza, że perceptron jest używany do rozróżniania dwóch kategorii obiektów na podstawie danych wejściowych. Jeśli perceptron jest stosowany do klasyfikacji binarnej, to ma tylko jedno wyjście, które jest używane do oznaczania przynależności obiektu do jednej z dwóch kategorii. Składa się z pojedynczej warstwy jednostek progowych liniowych, które przyjmują wiele wejść i generują pojedyncze wyjście. Jednostka progowa liniowa polega na tym, że wejście jest mnożone przez wagę, a następnie jest dodawane do biasu (stałej, która jest dodawana do iloczynu skalarnego wejścia i wag), a następnie jest przepuszczane przez funkcję progową, która zwraca 1 lub 0 w zależności od tego, czy wejście jest większe niż 0. Wyjściem perceptronu jest konkretna wartość, w zależności od tego, czy wartość pewnej funkcji na danych wejściowych przekroczy próg.

Algorytm perceptronu jest używany do nauki wag wejść tak, aby perceptron mógł poprawnie sklasyfikować dany zestaw danych wejściowych. Proces uczenia polega na dostosowywaniu wag wejść na podstawie błędu między przewidywaną wartością a prawdziwą wartością wyjścia. Wagi są dostosowywane za pomocą wariantu spadku gradientu zwanego "regułą delta", która polega na aktualizowaniu wag na podstawie pochodnej cząstkowej funkcji błędu względ-



Rysunek 5.3: Przykład perceptronu.

dem wag. Istnieją również inne algorytmy doboru wag [18] - Inicjalizacja losowa, inicjalizacja He, Inicjalizacja Glorot (lub Xavier), inicjalizacja LeCun, inicjalizacja ortogonalna czy wagi wstępne wytrenowane.

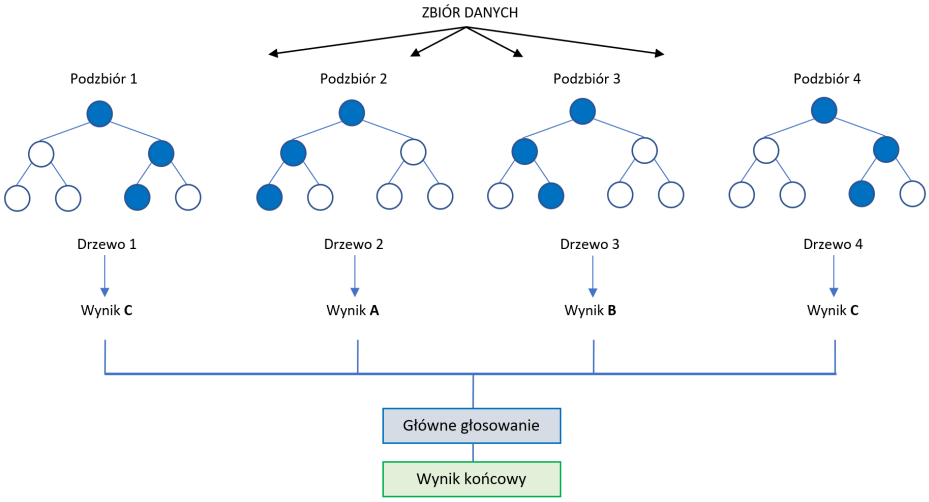
Może być również podatny na **przeuczenie** (*ang. overfitting*), szczególnie w przypadku gdy dane treningowe są zanieczyszczone lub nie są dobrze przystosowane do zadanego problemu. Przeuczenie to sytuacja, w której model uczenia maszynowego doskonale radzi sobie z przetwarzaniem danych treningowych, ale nie jest w stanie dobrze generalizować swoich wyników do nowych danych. Innymi słowy, model jest zbyt dopasowany do danych treningowych i nie jest w stanie dobrze przewidywać wyników dla nowych, nieznanych mu danych. Aby zmniejszyć ryzyko przeuczenia, można użyć większego zbioru danych treningowych.

Perceptron jest prostą i szybką metodą uczenia maszynowego, ale posiada pewien problem - niemożność nauczenia się logicznej struktury bramki XOR. Bramka ta traktowana jako wykres na płaszczyźnie jest nieodseparowana liniowo. Jest to znany problem, który przyczynił się do zatrzymania rozwoju sieci neuronowych na wiele lat - tzw. "AI winter" (Zima Sztucznej Inteligencji). Pierwszy raz miało to miejsce w latach 70. XX wieku, gdy oczekiwania dotyczące możliwości AI były bardzo wysokie, ale postęp technologiczny był znacznie niższy niż oczekiwano. Kolejne zatrzymania rozwoju miały miejsce w latach 80. i 90. XX wieku.

5.4 Las losowy

Las losowy (*ang. Random Forest*) [32] to metoda uczenia maszynowego, która wykorzystuje wiele drzew decyzyjnych do rozwiązywania problemu. Każde drzewo w lesie losowym jest niezależnie trenowane na losowym podzbiorze danych treningowych i losowych cech. Wszystkie drzewa w lesie losowym są następnie używane do przewidzenia klasy dla danej próbki poprzez głosowanie większościowe.

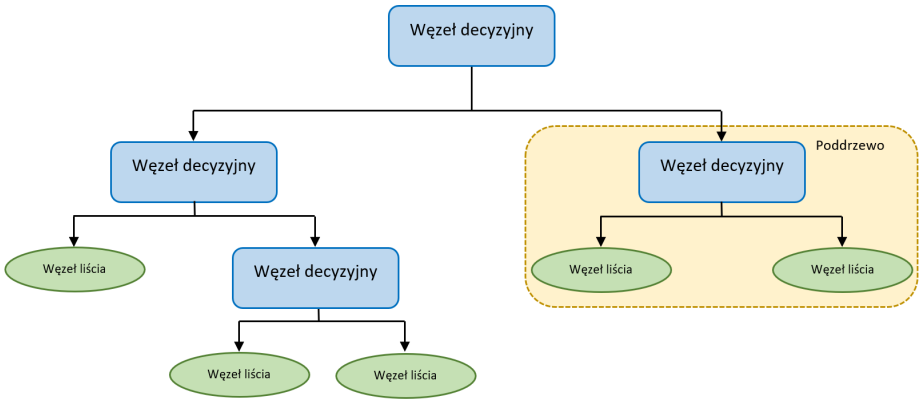
Jest to skuteczna metoda uczenia maszynowego, ponieważ wiele niezależnych drzew decyzyjnych umożliwia zmniejszenie ryzyka przeuczenia i zwiększenie dokładności predykcji.



Rysunek 5.4: Przykład Lasu losowego - opracowanie własne na podstawie [32].

5.5 Drzewo decyzyjne

Drzewo decyzyjne (*ang. Decision Tree*) [33] to rodzaj modelu uczenia maszynowego, który służy do przeprowadzania klasyfikacji lub regresji na podstawie danych wejściowych. Model ten składa się z wewnętrznych węzłów, które reprezentują cechy danych, oraz liści, które reprezentują decyzje lub wartości wyjściowe (rysunek 5.5).



Rysunek 5.5: Przykład Drzewa decyzyjnego - opracowanie własne na podstawie [33].

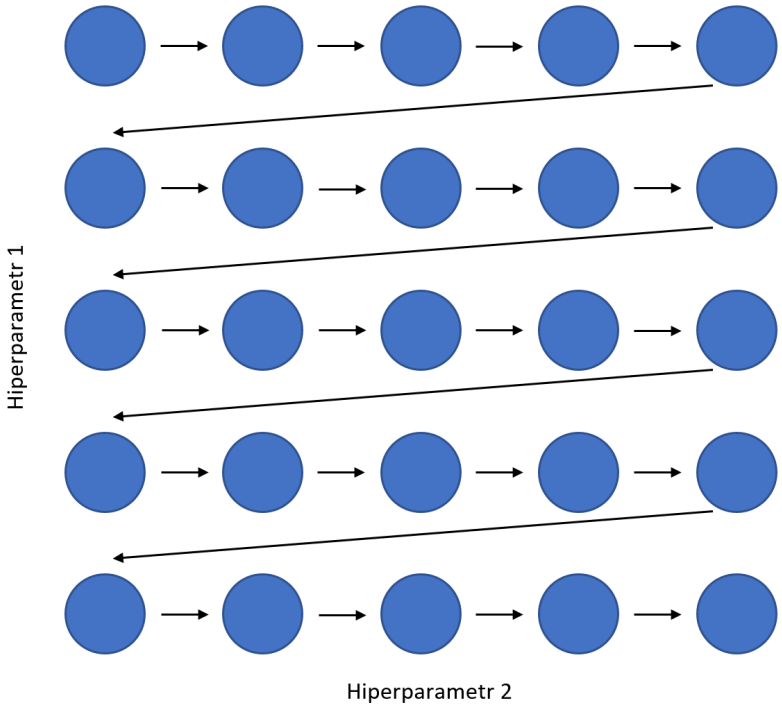
Aby przeprowadzić klasyfikację lub regresję za pomocą drzewa decyzyjnego, należy rozpocząć od korzenia drzewa i przejść przez kolejne węzły aż do liścia. Na każdym węźle wybierana jest najlepsza cecha do podziału danych, tak aby uzyskać jak największe zróżnicowanie między grupami. Proces ten jest powtarzany dla każdego podziału aż do osiągnięcia liścia, gdzie zostaje podjęta decyzja lub przewidziana wartość wyjściowa.

Drzewa decyzyjne są łatwe do zrozumienia i interpretacji, a także są skuteczne w wielu różnych problemach. Jednak mogą być podatne na przeuczenie, szczególnie w przypadku gdy drzewo ma zbyt dużo cech. Aby zmniejszyć ryzyko przeuczenia, można użyć metod takich jak ograniczanie głębokości drzewa lub wybieranie mniejszej liczby cech.

5.6 Metody walidacji

5.6.1 Przeszukiwanie siatki

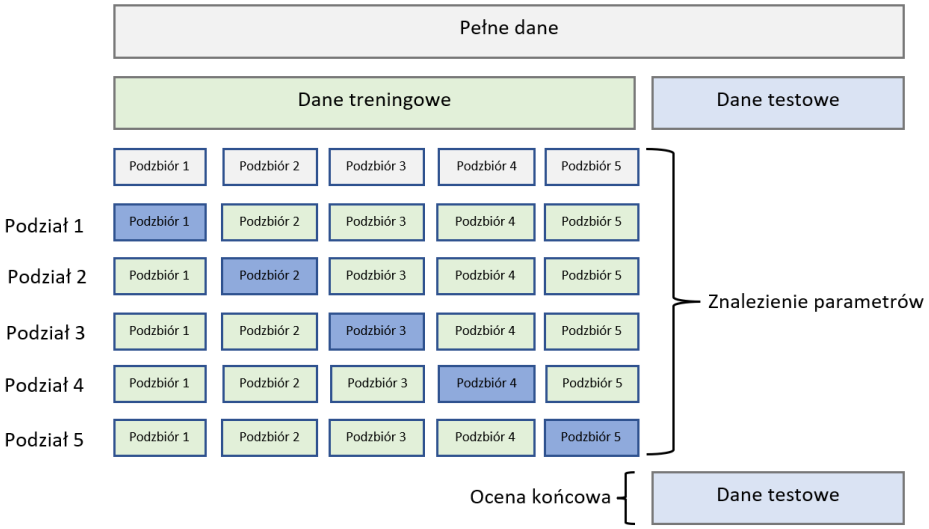
Przeszukiwanie siatki (*ang. GridSearchCV*) [34] to narzędzie z biblioteki scikit-learn, które służy do wyszukiwania najlepszych parametrów dla modelu uczenia maszynowego. Można z niego skorzystać, podając szereg możliwych wartości dla poszczególnych parametrów modelu oraz metrykę, która ma być używana do oceny jakości modelu. Następnie przeszukiwanie siatki automatycznie wygeneruje siatkę wszystkich możliwych kombinacji parametrów i uruchomi model z każdą z nich, a następnie zwróci najlepsze parametry, które osiągnęły najlepsze wyniki według wybranej metryki.



Rysunek 5.6: Przykład przeszukiwania siatki - opracowanie własne na podstawie [35].

5.6.2 Walidacja krzyżowa

Walidacja krzyżowa (*ang. Cross Validation*) [36] to procedura ponownego próbkowania stosowana do oceny wydajności modelu uczenia maszynowego. Działa poprzez podzielenie zestawu danych na zestaw treningowy i zestaw testowy, szkolenie modelu na zestawie treningowym i ocenę jego wydajności na zestawie testowym. Walidacji krzyżowej można użyć do dostrojenia hiperparametrów (parametrów modelu, które nie są uczone podczas trenowania, ale są ustawiane przed rozpoczęciem trenowania) modelu lub do wybrania najlepszego modelu z zestawu danych.



Rysunek 5.7: Przykład walidacji krzyżowej - opracowanie własne na podstawie [36].

Istotą walidacji krzyżowej jest to, że model jest trenowany i oceniany kilka razy na różnych podzestawach danych. Dzięki temu można uzyskać bardziej rzetelną ocenę jego skuteczności, ponieważ model jest w stanie lepiej radzić sobie z danymi, które nie zostały użyte do trenowania.

Część II

Część praktyczna

Rozdział 6

Wyniki analizy

6.1 Opis danych

Dane pochodzą z platformy **Kaggle** [37]. Jest to platforma, która zrzesza specjalistów i pasjonatów analizy danych. Strona umożliwia rozwiązywanie problemów dotyczących uczenia maszynowego. Oferuje również dostęp do różnych zbiorów danych oraz narzędzi do trenowania i testowania modeli.

	Object Name	Epoch (TDB)	Orbit Axis (AU)	Orbit Eccentricity	Orbit Inclination (deg)	Perihelion Argument (deg)	Node Longitude (deg)	Mean Anomoly (deg)	Perihelion Distance (AU)	Aphelion Distance (AU)	Orbital Period (yr)	Minimum Orbit Intersection Distance (AU)	Orbital Reference	Asteroid Magnitude	Classification	Hazardous
0	433 Eros	57800	1.4579	0.2226	10.8277	178.8050	304.3265	319.3111	1.1335	1.78	1.76	0.1492	598	11.16	Amor Asteroid	False
1	719 Albert	57800	2.6385	0.5479	11.5822	156.1409	183.9204	224.5535	1.1928	4.08	4.29	0.2004	78	15.50	Amor Asteroid	False
2	887 Alinda	57800	2.4787	0.5671	9.3561	350.3482	110.5444	351.3730	1.0731	3.88	3.90	0.0925	188	13.40	Amor Asteroid	False
3	1036 Ganymed	57800	2.6628	0.5338	26.6929	132.4690	215.5551	92.5640	1.2413	4.08	4.35	0.3421	597	9.45	Amor Asteroid	False
4	1221 Amor	57800	1.9191	0.4356	11.8795	26.6572	171.3448	313.7379	1.0832	2.76	2.66	0.1068	70	17.70	Amor Asteroid	False

Rysunek 6.1: Wybrany zbiór danych - pierwsze pięć rekordów.

Zbiór danych przedstawia zaobserwowane przez NASA asteroidy wraz z ryzykiem ich uderzenia. Oprócz opisu ich podstawowych parametrów, które były zawarte w podrozdziale 2.3 mamy również dwa kolejne:

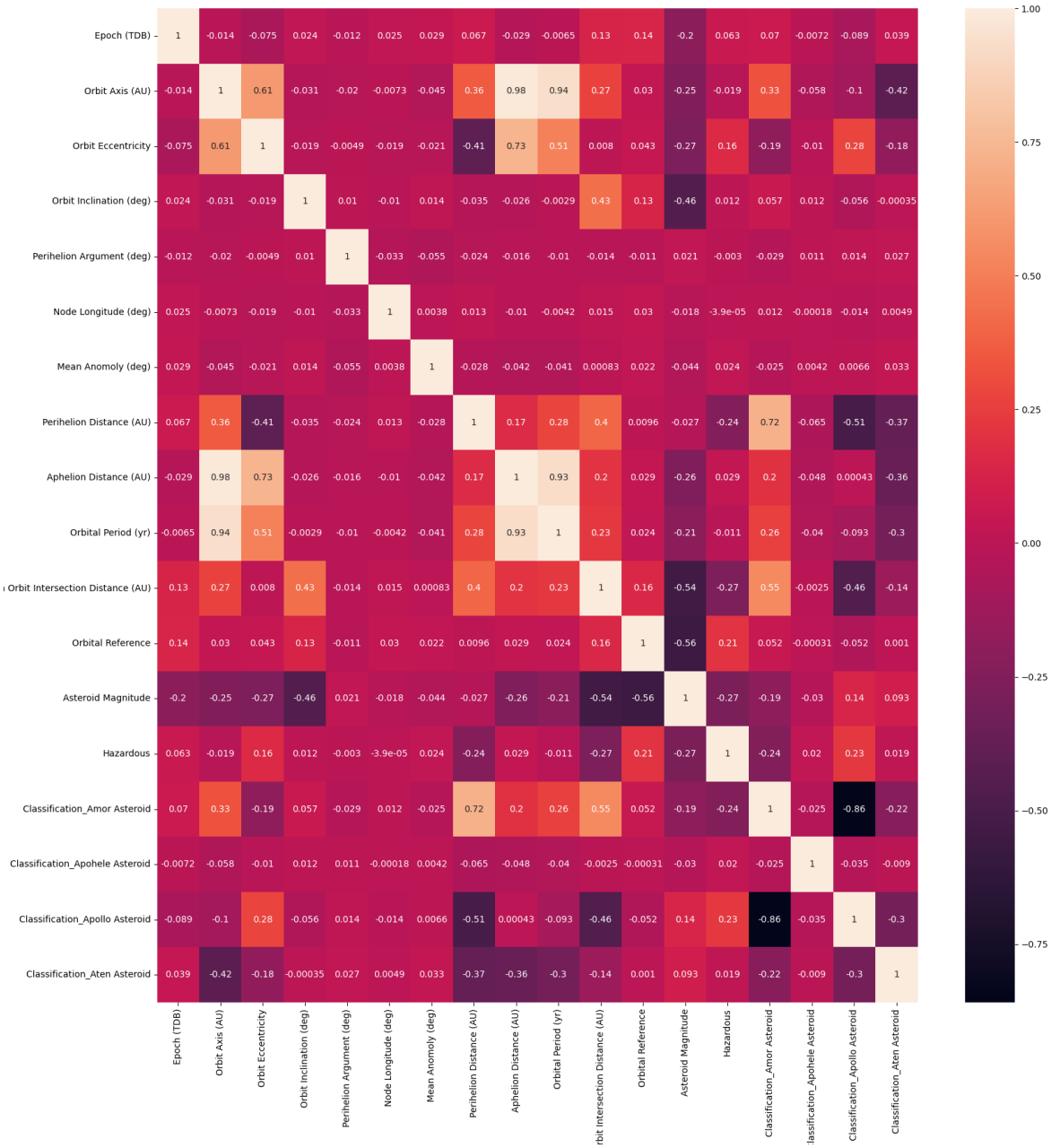
- **Klasyfikacja** określa do jakiej kategorii należy dana asteroida.
- Parametr **Niebezpieczny** zawiera informacje o tym, czy obiekt jest zagrożeniem dla Ziemi czy też nie.

Dane pochodzą z oficjalnego programu NASA - "Near Earth Object Program w Jet Propulsion Laboratory" [38], oparte są na licencji CC0: domena publiczna.

6.2 Zależność kolumn w zbiorze danych

Stworzono macierz korelacji w celu przedstawienia zależności kolumn w zbiorze danych.

```
import seaborn as sns

# tworzymy figurę wielkości 2ox20, za pomocą biblioteki Seaborn wyświetlamy
# macierz korelacji,
plt.figure(figsize=(20,20))
sns.heatmap(data=pd.get_dummies(dane).corr(), annot=True)
```



Rysunek 6.2: Macierz korelacji przedstawiająca zależność kolumn w zbiorze danych.

Można zauważyć, że część kolumn jest ze sobą skorelowana w mniejszym lub większym stopniu. W przypadku zależności kolumn dystans aphelium oraz oś orbity, współczynnik korelacji wynosi aż 0.98. Oznacza to, że im większa orbita, tym większy jest dystans aphelium.

Dystans aphelium koreluje w mniejszym stopniu z ekscentrycznością orbity ze współczynnikiem na poziomie 0.73. Im większy stopień ekscentryczności, tym większy dystans aphelium.

Okres orbitalny związany jest z długością orbity (współczynnik wynosi 0.94) - oznacza to, że im dłuższa orbita, tym asteroida potrzebuje więcej czasu na przebycie pełnego obiegu wokół Słońca.

Okres orbitalny koreluje również z dystansem aphelium (współczynnik na poziomie 0.93) - tutaj zależność jest podobna, dłuższy dystans aphelium to dłuższy czas na przebycie pełnego obiegu wokół Słońca.

6.3 Wstępne operacje na zbiorze danych

Wstępem w projekcie było zaimportowanie początkowych bibliotek, a następnie wczytanie danych. Można je łatwo wprowadzić do Jupyter'a i za pomocą funkcji *head*, można wyświetlić pierwsze pięć rekordów ze zbioru.

```
# zaimportowanie początkowych bibliotek

import opendatasets as ods
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# za pomocą biblioteki opendatasets pobieramy zbiór danych

ods.download(
    "https://www.kaggle.com/nasa/asteroid-impacts?select=orbits.csv")

# tworząc zmienną 'plik' wczytujemy nasz pobrany zbiór danych z folderu

plik = ('asteroid-impacts/\
orbits-orbits.csv')

# pobrany zbiór danych jest w formacie .csv więc musimy użyć funkcji
# read_csv('zmienna') aby wczytać dane

dane = pd.read_csv(file)

# za pomocą funkcji head wyświetlamy pierwsze pięć rekordów ze zbioru

dane.head()
```

Nazwy kolumn w zbiorze określamy jako zmienne X, wykluczamy jedynie kolumny nazwa obiektu (ang. *Object Name*) oraz klasyfikacja (ang. *Classification*), gdyż nie wniosą one niczego do analizy. Kolumna Niebezpieczny (ang. *Hazardous*) będzie y. Parametry orbit zostały dokładniej opisane w podrozdziale 2.3.

```
# wybrane kolumny potrzebne do analizy określono jako 'zmienne'

zmienne = ['Orbit Axis (AU)', 'Orbit Eccentricity',
       'Orbit Inclination (deg)', 'Perihelion Argument (deg)',
       'Node Longitude (deg)', 'Mean Anomoly (deg)',
       'Perihelion Distance (AU)', 'Aphelion Distance (AU)',
       'Orbital Period (yr)', 'Minimum Orbit Intersection Distance (AU)',
       'Orbital Reference', 'Asteroid Magnitude']

# naszymi X będą 'zmienne', natomiast kolumna 'Hazardous' to będzie nasz y

X, y = dane.loc[:, zmienne].values, dane.loc[:,['Hazardous']].values
```

W celu sprawdzenia czy zbiór posiada brakujące wartości używamy funkcji *isnan* i tworzymy instrukcję warunkową. Jeśli X oraz y zawierają brakujące wartości to wyświetli nam się wynik "Zawiera puste wartości". Natomiast jeśli wszystko jest w porządku, instrukcja zwróci nam wynik "Nie zawiera pustych wartości".

```
# jeśli dane X oraz y zawierają puste wartości to zwróci wynik "zawiera
# puste wartości"
# w przeciwnym razie zwróci wynik "nie zawiera pustych wartości"

if(np.isnan(X).any() & np.isnan(y).any()):
    print("Zawiera puste wartości")
else:
    print("Nie Zawiera pustych wartości")
```

Po uruchomieniu kodu, instrukcja warunkowa zwróciła nam wartość "Nie zawiera pustych wartości" co oznacza, że wybrane dane są pełne.

6.4 Uczenie nienadzorowane

Podzieliliśmy X oraz y na zbiory testowe oraz treningowe - odpowiednio 30% oraz 70% aby nie doszło do przeuczenia modeli. Chcąc użyć na naszych danych analizy głównych składowych (PCA) musimy najpierw poddać je normalizacji - używamy w tym celu narzędzia skaler standardowy. Dopasowujemy oraz przekształcamy dane wejściowe (X_train oraz X_test) na macierz danych przekształconych przez skaler.

```
# importujemy pakiet 'train_test_split' odpowiedzialny za podział
# danych na testowe i treningowe
# oraz importujemy narzędzie skaler standardowy

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Dzielimy x oraz y na dane testowe i treningowe
# funkcja 'test_size' = 0.3 przydziela 30% danym testowym
# pozostałe 70% przydzieli do treningowych
# funkcja 'stratify' dzieli y na dane warstwowe
# funkcja 'random_state' = 0 mówi nam, że otrzymamy te same zestawy treningowe
# i testowe w różnych wersjach
```

```
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)

sc = StandardScaler()

# wykonujemy tranfsormację X testowych i treningowych

X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

Obliczamy wartości własne macierzy kowariancji z wykorzystaniem instrukcji *linalg.eig*, następnie sortujemy wartości własne według wektorów własnych k na podstawie wartości odpowiadających im wartości własnych.

```
#  w zmiennej 'macierz_kow' tworzymy macierz kowariancji za pomocą funkcji 'np.cov'
# '.T' przy danych jest transponowaniem macierzy
# w zmiennych 'wartosci_wlasne' i 'wektory_wlasne'

macierz_kow = np.cov(X_train_std.T)
wartosci_wlasne, wektory_wlasne = np.linalg.eig(macierz_kow)
print('\nWartości własne \n%s' % wartosci_wlasne)
```

Wyświetlamy otrzymane wyniki na rysunku 6.3, te wartości będą nam potrzebne do zwizualizowania wykresu analizy głównych składowych.

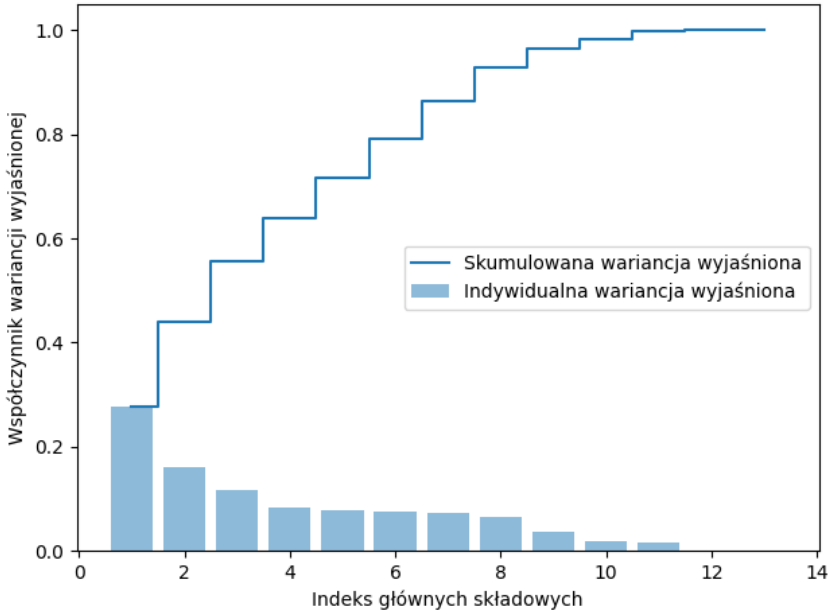
```
Wartości własne
[3.60591508e+00 2.03348109e+00 1.51565677e+00 2.83126596e-06
 1.72748251e-02 1.97128449e-01 2.36174475e-01 4.75762235e-01
 1.06223759e+00 9.16016745e-01 9.42718091e-01 9.98728404e-01]
```

Rysunek 6.3: Wartości własne macierzy.

Używając w funkcji *cumsum*, możemy obliczyć skumulowaną sumę wariancji wyjaśnionych, które następnie zwizualizujemy w postaci wykresu analizy PCA (rysunek 6.4).

```
# sumujemy wartości własne maciery

suma = sum(wartosci_wlasne)

# obliczamy wariancję wyjaśnioną
# następnie obliczamy skumulowaną wariancję wyjaśnioną za pomocą funkcji 'cumsum'

war_wyjasniona = [(i / tot) for i in
      sorted(eigen_vals, reverse=True)]
skumul_war_wyjasniona = np.cumsum(war_wyjasniona)

# tworzymy wykres analizy PCA za pomocą biblioteki Matplotlib

plt.bar(range(1,14), war_wyjasniona, alpha=0.5, align='center',
    label='Indywidualna wariancja wyjaśniona')
```

```
plt.step(range(1,14), skumul_war_wyjasniona, where='mid',
    label='Skumulowana wariancja wyjaśniona')
plt.ylabel('Wskaźnik wariancji wyjaśnionej')
plt.xlabel('Indeks głównych składowych')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```



Rysunek 6.4: Wykres analizy głównych składowych.

Wybieramy skumulowany procent wariancji wyjaśnionej, czyli najmniejszą liczbę głównych składowych, dla których suma wariancji stanowi pewną część wariancji wszystkich zmiennych, które zostały poddane redukcji wymiarów. Zakładamy dolną granicę 80% co wskazuje na 6 składowych głównych.

Wykorzystując dane z metody PCA możemy teraz wprowadzić je do analizy sylwetki. Tworzymy wykresy dla 6 klastrów aby sprawdzić dla którego można osiągnąć najlepszą liczbę skupień. Kod pokazuje również średni wynik sylwetki dla odpowiedniego klastra.

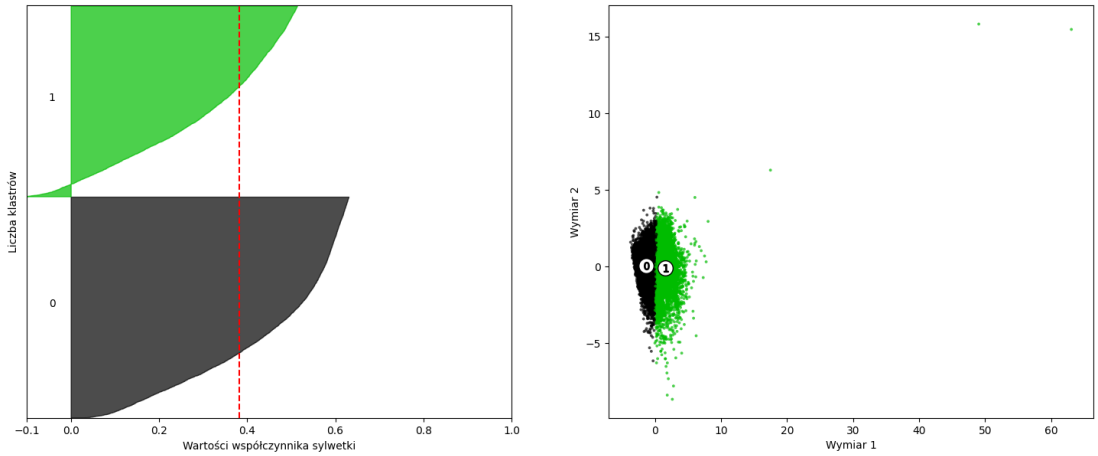
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.cm as cm

# tworzymy 2 wykresy - dla analziy sylwetki oraz dla k-średnich obok siebie
# wyznaczamy, które klastry mają być wyświetlone

zakres_n_klastrow = [2, 3, 4, 5, 6]
```

```
# pętla for wyświetli nam wykres dla danego klastra w zakresie klastrów

for n_klaster in zakres_n_klastrow:
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    ax1.set_xlim([-0.1, 1])
    ax1.set_ylim([0, len(X_train_pca) + (n_klaster + 1) * 10])
    klaster = KMeans(n_clusters=n_klaster, random_state=10)
    etykieta_klastra = klaster.fit_predict(X_train_pca)

# wyświetlamy średnie wyniki współczynnika sylwetki dla danego klastra
# zaokrąglamy je za pomocą funkcji 'round' do dwóch miejsc po przecinku

    sylwetka_srednia = silhouette_score(X_train_pca, etykieta_klastra)
    sylwetka_srednia_zaokr = round(sylwetka_srednia, 2)
    print(
        "Dla n_klastrów =",
        n_klaster,
        "Średni wynik współczynnika sylwetki wynosi :",
        sylwetka_srednia_zaokr,
    )

    sample_silhouette_values = silhouette_samples(X_train_pca, etykieta_klastra)
```

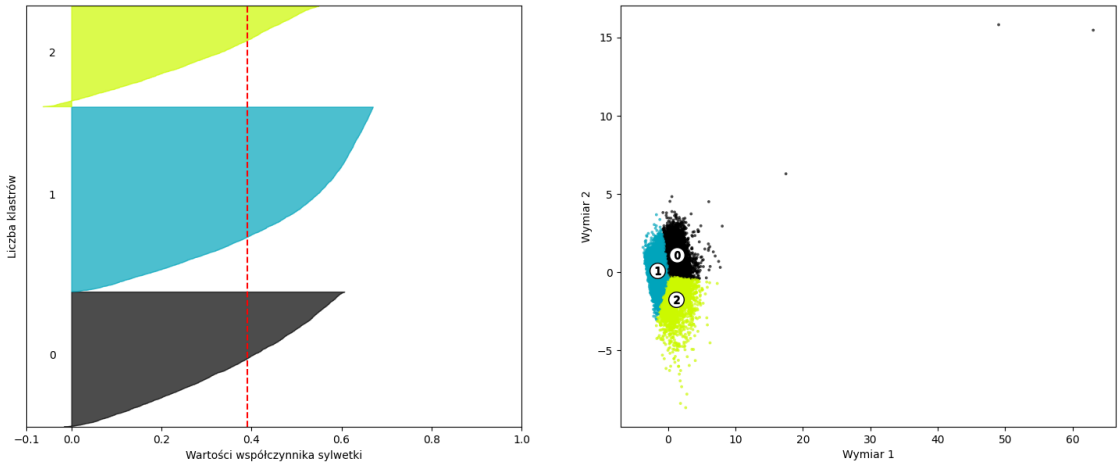
W analizie sylwetki mamy wizualizację dla sześciu klastrów z ich centroidami. Najlepszą analizą jaką możemy zauważyć (rysunek 6.10) jest ta dla n klastrów $= 6$ co potwierdzałoby zgodność z poprzednimi algorytmami. Możemy też zauważyć, że w analizie pojawiają się ujemne wartości. Oznacza to, że punkt jest bliżej punktów innego klastra niż do punktów w swoim klastrze.

```
Dla n_klastrów = 2 Średni wynik współczynnika sylwetki wynosi : 0.38
Dla n_klastrów = 3 Średni wynik współczynnika sylwetki wynosi : 0.39
Dla n_klastrów = 4 Średni wynik współczynnika sylwetki wynosi : 0.39
Dla n_klastrów = 5 Średni wynik współczynnika sylwetki wynosi : 0.38
Dla n_klastrów = 6 Średni wynik współczynnika sylwetki wynosi : 0.35
```

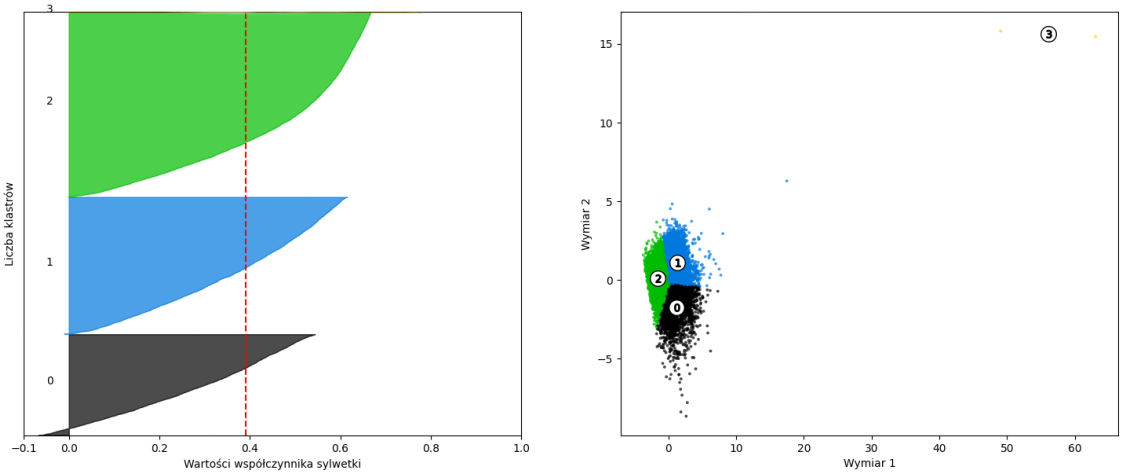
Rysunek 6.5: Średni wynik analizy sylwetki dla określonych klastrów.



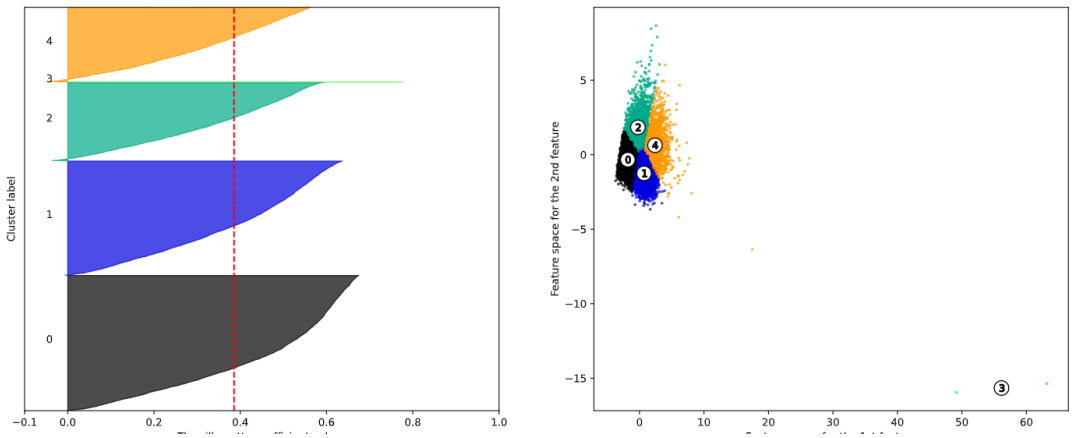
Rysunek 6.6: Analiza sylwetki dla n klastrów = 2.



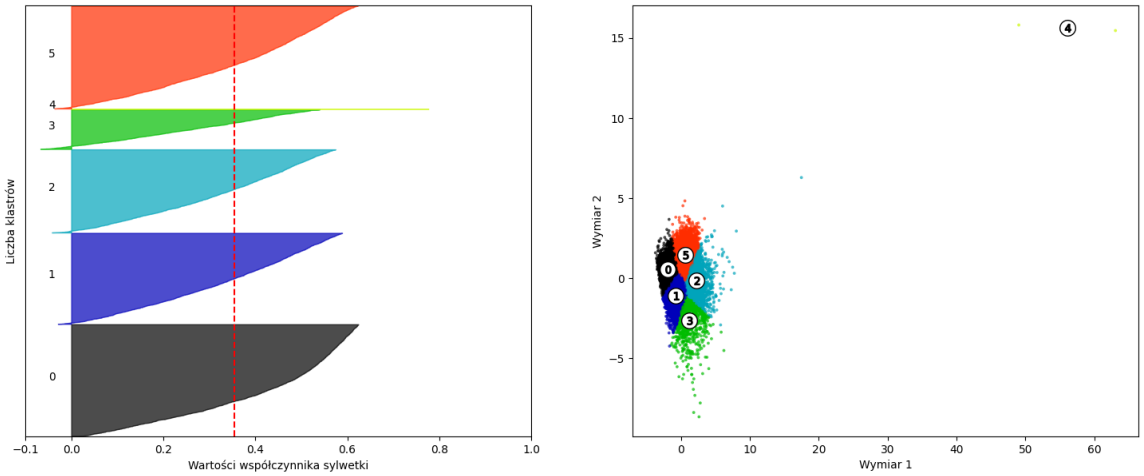
Rysunek 6.7: Analiza sylwetki dla n klastrów = 3.



Rysunek 6.8: Analiza sylwetki dla n klastrów = 4.



Rysunek 6.9: Analiza sylwetki dla n klastrów = 5.

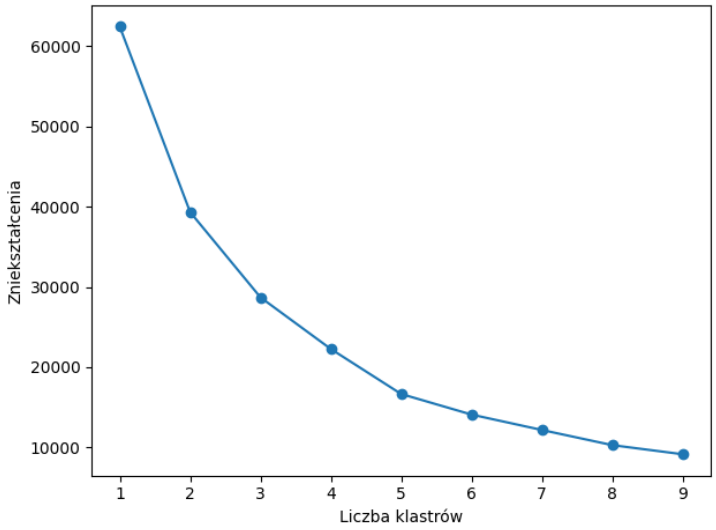


Rysunek 6.10: Analiza sylwetki dla n klastrów = 6.

Przystępujemy do zwizualizowania wykresu metody łokciowej aby oszacować optymalną liczbę klastrów. Jeżeli k się zwiększy, zniekształcenia będą mniejsze, szukamy wartości k, gdzie zniekształcenia będą rosnąć najszybciej.

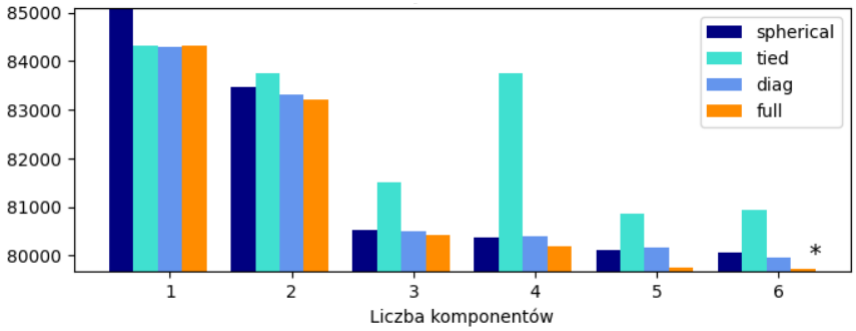
```
zniekształcenia = []
for i in range(1, 10):
    km = KMeans(n_clusters=i,
                init='k-means++',
                n_init=10,
                max_iter=300,
                random_state=0)
    km.fit(X_train_pca)
    zniekształcenia.append(km.inertia_)
plt.plot(range(1,10), zniekształcenia, marker='o')
plt.xlabel('Liczba klastrów')
plt.ylabel('Zniekształcenia')
plt.tight_layout()
plt.show()
```

W przeprowadzonej metodzie łokciowej, zniekształcenie nie jest aż tak dobrze widoczne. Oznacza to, że nie jest to dobra metoda dla tych danych.



Rysunek 6.11: Wykres metody łokciowej.

Przechodzimy jeszcze do wyznaczenia kryterium BIC. Na przedstawionym rysunku 6.12 możemy zauważyć, że kolumna z 6 komponentami - dodatkowo oznaczona gwiazdką.



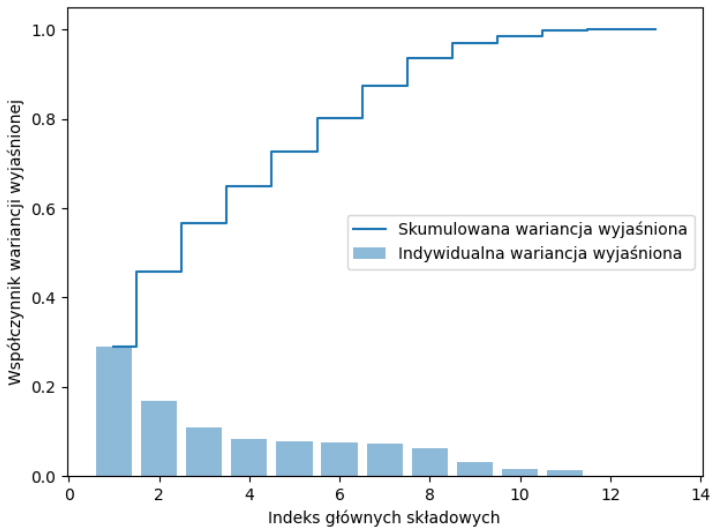
Rysunek 6.12: Kryterium BIC dla przetwarzanych danych.

Wcześniejsze metody zostały wykonane z użyciem zmiennych x (wszystkie kolumny za wyjątkiem Nazwa Obiektu, Klasyfikacja oraz Niebezpieczny) oraz y (kolumna Niebezpieczny, gdzie wartości prawdziwe odpowiadają asteroidom zagrażającym Ziemi, natomiast fałszywe asteroidom, które nie stanowią zagrożenia). Spróbujmy podzielić y wartości prawdziwe i fałszywe.

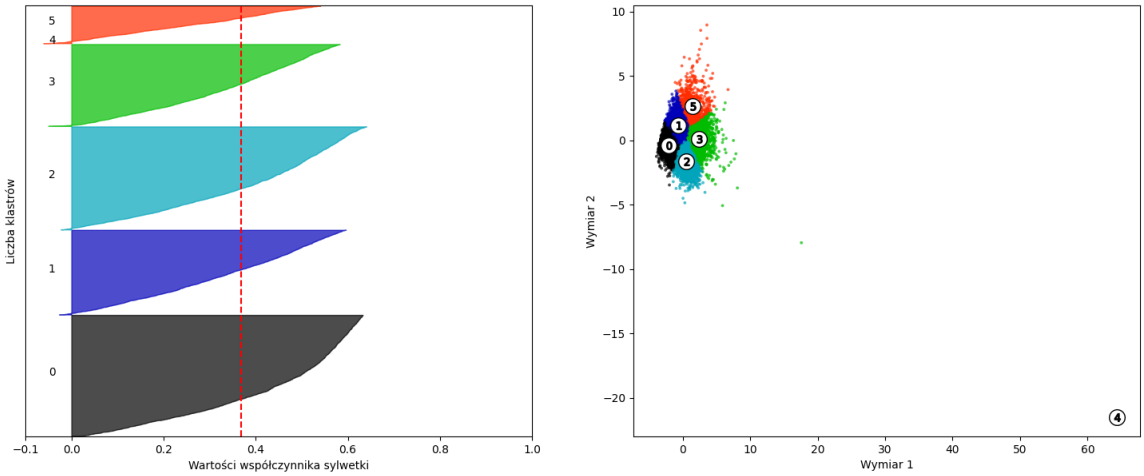
Stworzyliśmy zmienną *dane_y_true*, uwzględniamy wiersze, w których wartości z kolumny Niebezpieczny są prawdziwe oraz analogicznie zmienna dla wartości fałszywych.

```
dane_y_true = dane.loc[dane['Hazardous']==True]

dane_y_false = dane.loc[dane['Hazardous']==False]
```

Wprowadzamy wcześniejsze zmienne x oraz y z wartościami fałszywymi, a następnie wykonujemy analizę dla tych samych metod.

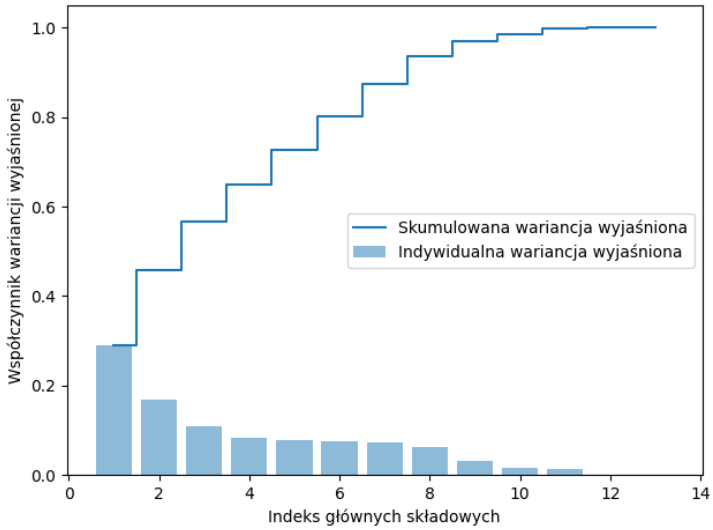


Rysunek 6.13: Analiza głównych składowych dla wartości fałszywych.

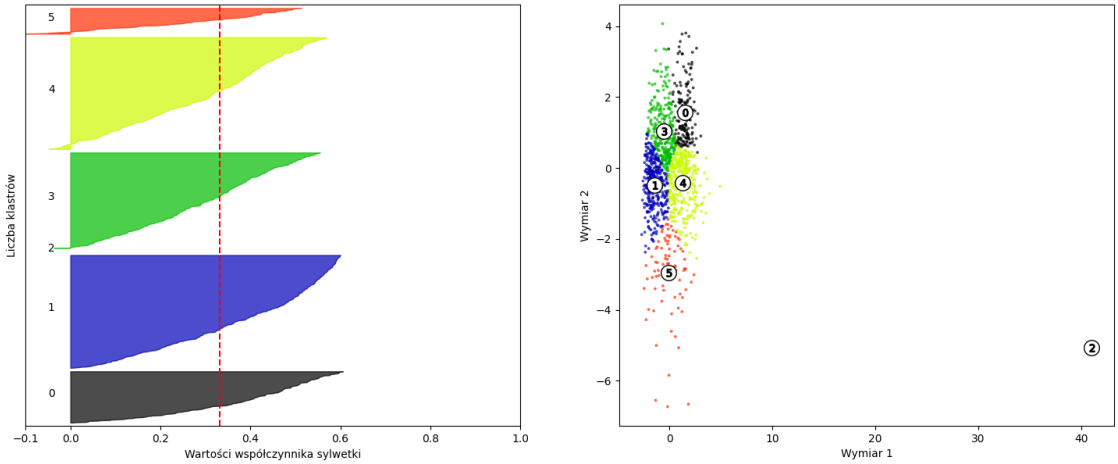


Rysunek 6.14: Analiza sylwetki dla wartości fałszywych.

Przechodzimy do analizy dla wartości prawdziwych.

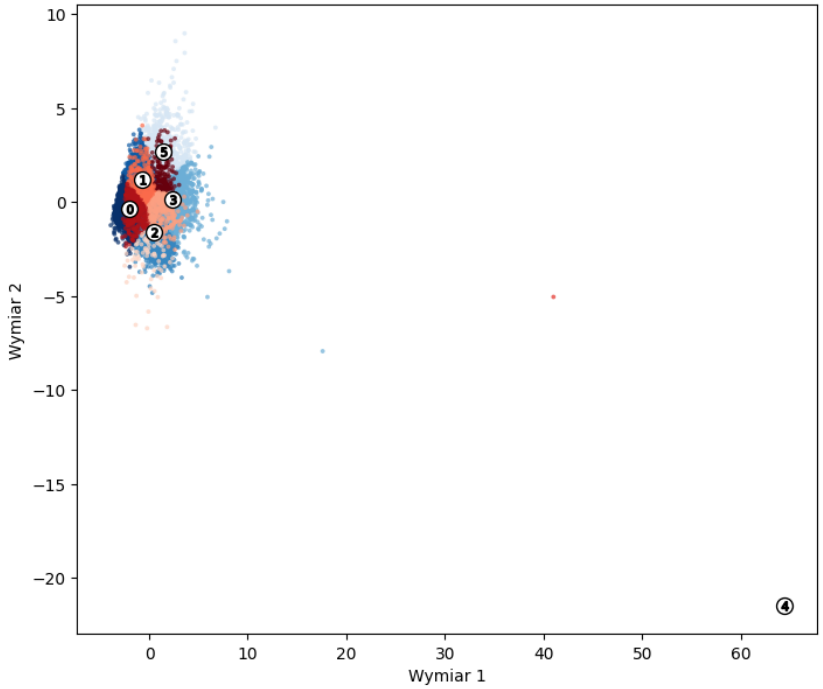


Rysunek 6.15: Analiza głównych składowych dla wartości prawdziwych.

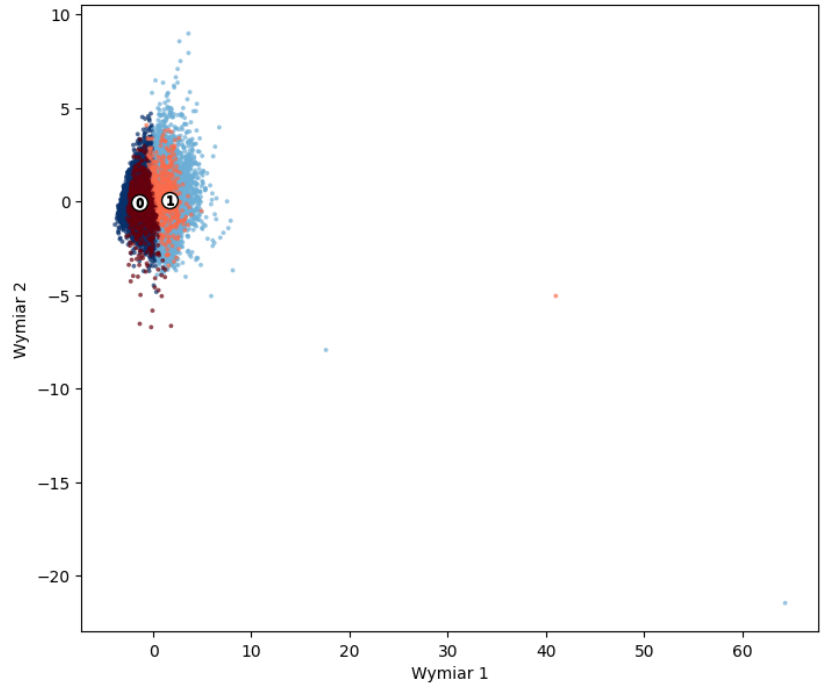


Rysunek 6.16: Analiza sylwetki dla wartości prawdziwych.

Na koniec chcemy wyświetlić wartości prawdziwe na tle fałszywych. Na wykresie 6.17 oraz 6.18 mamy przedstawioną ilość asteroid niebezpiecznych dla Ziemi (wartości prawdziwe, zaznaczone odcieniami koloru czerwonego) oraz asteroidy, które nie zagrażają Ziemi (wartości fałszywe, zaznaczone odcieniami koloru niebieskiego). Liczba asteroid nie zagrażających Ziemi wynosi 13856, natomiast niebezpiecznych 1779. Wykonano również ten sam wykres ale dla n_klastrów = 2. Widzimy, że zastosowane metody uczenia nienadzorowanego nie bardzo radzą sobie z klasyfikacją danych.



Rysunek 6.17: Asteroidy niebezpieczne dla Ziemi dla 6 komponentów - (odcienie koloru czerwonego) na tle asteroid nie zagrażających Ziemi (odcienie koloru niebieskiego), gdzie odcienie oznaczają przynależność do danych klastrów.



Rysunek 6.18: Asteroidy niebezpieczne dla Ziemi dla 2 komponentów - (odcienie koloru czerwonego) na tle asteroid nie zagrażających Ziemi (odcienie koloru niebieskiego), gdzie odcienie oznaczają przynależność do danych klastrów.

6.5 Uczenie nadzorowane

Opisując część uczenia nadzorowanego naszym y będzie kolumna 'Hazardous', natomiast X będą wszystkie kolumny oprócz pierwszej czyli 'Object Name'. Następnie normalizujemy nasze dane przez skaler standardowy. Importujemy biblioteki uczenia maszynowego - SVC, las losowy, drzewo decyzyjne oraz perceptron. Przetwarzamy nasze dane przez wymienione wyżej algorytmy.

```
# importujemy potrzebne biblioteki

from sklearn.metrics import classification_report,precision_score,recall_score,f1_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import Perceptron

# tworzymy cztery puste słowniki, które zostaną wykorzystane do przechowywania
# wskaźników oceny dla każdego modelu

    model_scores = {}
    model_recall = {}
    model_f1 = {}
    model_precision = {}

# nazwa oraz model to klucze, a wartość to obiekt modelu
# do każdego modelu dopasowywane są dane treningowe 'X_train' oraz 'y_train'
# oraz dokonywane są prognozy na danych testowych 'X_test'
# następnie używamy funkcji 'classification_report' do wyświetlenia raportu
# klasyfikacji, który obejmuje precyzję, czułość, F1 oraz nośnik

    for name, model in models.items():
        model.fit(X_train,y_train)
        y_preds = model.predict(X_test)
        print(name)
        print(classification_report(y_test, y_preds))
        print('\n')
        model_scores[name] = model.score(X_test,y_test)
        model_recall[name] = recall_score(y_test, y_preds)
        model_f1[name] = f1_score(y_test, y_preds)
        model_precision[name] = precision_score(y_test, y_preds)

    model_scores = pd.DataFrame(model_scores, index=['Score']).transpose()
    model_scores = model_scores.sort_values('Score')
    model_recall = pd.DataFrame(model_recall, index=['Recall']).transpose()
    model_recall = model_recall.sort_values('Recall')
    model_f1 = pd.DataFrame(model_f1, index=['F1']).transpose()
    model_f1 = model_f1.sort_values('F1')
    model_precision = pd.DataFrame(model_precision, index=['Precision']).transpose()
    model_precision = model_precision.sort_values('Precision')

    return model_scores, model_recall, model_f1, model_precision
```

W naszym przypadku interesuje nas średnia ważona dla wybranych modeli. W tabeli 6.1 znajdują się wyniki analizy - średnie ważone dla precyzji, czułości oraz F1. Parametr nośnik zawiera liczbę danych testowych.

Tabela 6.1: Wyniki analizy dla wybranych modeli bazowych uczenia maszynowego.

	Precyzja	Czułość	F1	Nośnik
perceptron	0.95	0.95	0.95	4691
SVC	0.97	0.97	0.97	4691
las losowy	1.00	1.00	1.00	4691
drzewo decyzyjne	1.00	1.00	1.00	4691

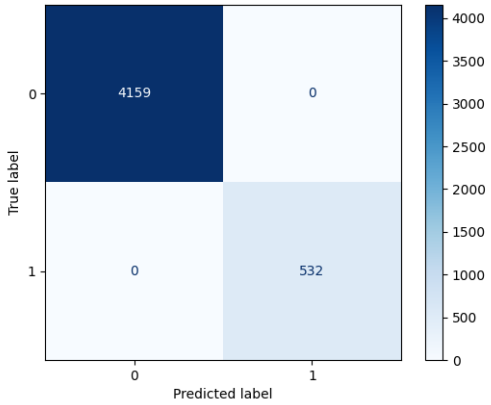
Następnie używamy walidacji krzyżowej aby porównać otrzymane wyniki. W tabeli 6.2 ukazano precyzję oraz odchylenie standardowe wybranych przez nas algorytmów.

Tabela 6.2: Wyniki analizy dla wybranych modeli z użyciem walidacji krzyżowej.

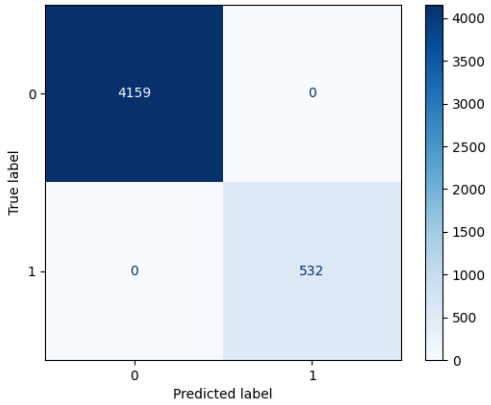
	Precyzja	Odchylenie standardowe
perceptron	0.73	$\pm$ 0.080
SVC	0.84	$\pm$ 0.025
las losowy	1.00	$\pm$ 0.002
drzewo decyzyjne	1.00	$\pm$ 0.002

Przystępujemy do wyznaczenia tablicy pomyłek aby zobaczyć jak sprawdził się dany model.

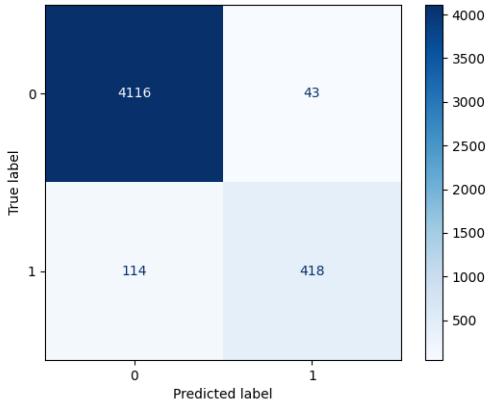
```
# importujemy bibliotekę odpowiedzialną za utworzenie tablicy pomyłek

from sklearn.metrics import plot_confusion_matrix

# wybieramy algorytm oraz dopasowujemy dane

model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_preds = model.predict(X_test)

# wizualizujemy tablicę pomyłek

plot_confusion_matrix(model, X_test,y_test, cmap=plt.cm.Blues)
```



Rysunek 6.19: Tablica pomyłek dla drzewa decyzyjnego.

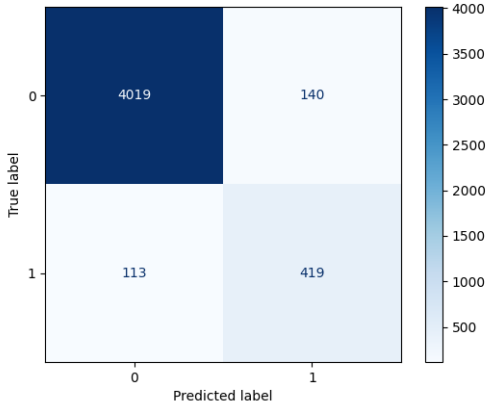


Rysunek 6.20: Tablica pomyłek dla lasu losowego.



Rysunek 6.21: Tablica pomyłek dla SVC.

Tablica pomyłek dla drzewa decyzyjnego oraz lasu losowego potwierdzają największą precyzję modeli. Testowe wartości zostały oznaczone jako przewidywane i wprowadzone do tablicy. W obu algorytmach wartości prawdziwe pozytywne wynoszą 4159, natomiast wartości



Rysunek 6.22: Tablica pomyłek dla perceptronu.

prawdziwe negatywne 532. W przypadku SVC oraz perceptronu niektóre wartości nie są poprawnie sklasyfikowane co może świadczyć o tym, że wybrany model nie jest tak dokładny. Pierwszy model posiada 114 wartości fałszywych pozytywnych oraz 43 wartości fałszywe negatywne. Model perceptronu zawiera 113 wartości FP oraz 140 wartości FN.

Przystępujemy do poprawienia dokładności za pomocą przeszukiwania siatki dla nieprecyzyjnych modeli. Tworzymy obiekt klasy SVC, a następnie tworzymy obiekt klasy przeszukiwania siatki, który będzie szukał najlepszych parametrów dla tego modelu. Podajemy szereg możliwych wartości dla parametrów C i *gamma* oraz *kernel*. Następnie uruchamiamy proces wyszukiwania najlepszych parametrów za pomocą funkcji *fit* a następnie wyświetlamy najlepsze parametry oraz najlepszy wynik. Dodatkowo po analizie zawężamy wartości dla parametru C aby jeszcze bardziej poprawić dokładność modelu. Te same kroki wykonujemy dla perceptronu. Następnie wyniki analizy przedstawiamy w formie tabelki.

```
# importujemy potrzebne biblioteki

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# definiujemy wartości parametrów które mają być wyszukane dla modelu SVC

param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'gamma': ['scale', 'auto'],
              'kernel': ['linear', 'rfb']}

# określamy dany algorytm w zmiennej 'model'

model = SVC()

# tworzymy siatkę wyszukiwania parametrów używając 5-krotnej walidacji krzyżowej

grid = GridSearchCV(model, param_grid, refit=True, verbose=5, cv=5, n_jobs=-1)
```

```
# dopasowujemy model do przeszukiwania siatki

grid.fit(X_train, y_train)

# wyświetlamy najlepsze parametry po przeszukaniu

print(grid.best_params_)
grid_predictions = grid.predict(X_test)

# wyświetlamy raport klasyfikacji

print(classification_report(y_test, grid_predictions))
```

Poniżej przedstawiono wartości parametrów dla modelu perceptronu, które później zamieniam

```
# wartości parametrów dla modelu perceptronu

param_grid = {'alpha': [0.000001,0.00001,0.0001],
              'penalty': ['l1', 'l2']}

model = Perceptron()

grid = GridSearchCV(model, param_grid, refit=True, verbose=5, cv=5, n_jobs=-1)
```

Tabela 6.3: Próba poprawy parametrów najgorszych modeli za pomocą narzędzia przeszukiwania siatki.

	Precyzja	Czułość	F1	Nośnik
Perceptron	0.95	0.95	0.95	4691
SVC	0.95	0.96	0.96	4691

Przedstawione wyniki analizy (tabela 6.4) pokazują, że modele po użyciu narzędzia przeszukiwania siatki, nawet po zawężeniu kryteriów, nie zmieniają swojej dokładności. W przypadku SVC, parametr precyzja pogorszył się o 0.01.

Utworzyliśmy również tablicę pomyłek dla tych modeli aby sprawdzić czy coś się poprawiło względem poprzednich wyników, niestety wartości się nie zmieniły.

Spróbujmy poprawić wynik najlepszych klasyfikatorów. W tym celu używamy tej samej funkcji zmieniając parametry odpowiednio dla drzewa decyzyjnego oraz lasu losowego.

```
# parametry dla drzewa decyzyjnego

param_grid = {'max_depth': [1, 2, 3, 4, 5],
              'min_samples_split': [2, 3, 4, 5],
              'criterion': ['gini', 'entropy']}

model = DecisionTreeClassifier()
```

```
# parametry dla lasu losowego

param_grid = {'n_estimators': [10, 50, 100, 200],
              'max_depth': [None, 2, 4, 6],
              'min_samples_split': [2, 4, 6],
              'criterion': ['gini', 'entropy']}

model = RandomForestClassifier()
```

Tabela 6.4: Próba poprawy parametrów najlepszych modeli za pomocą narzędzia przeszukiwania siatki.

	Precyzja	Czułość	F1	Nośnik
Drzewo Decyzyjne	1.00	1.00	1.00	4691
Las Losowy	1.00	1.00	1.00	4691

Po użyciu narzędzia przeszukiwania siatki, parametry modeli drzewa decyzyjnego oraz lasu losowego nie zmieniły się.

Rozdział 7

Podsumowanie i wnioski

Przeprowadziliśmy przetwarzanie danych w procesie uczenia maszynowego. W uczeniu nienadzorowanym metoda łokciowa okazała się niewłaściwa dla analizowanych danych. Punkt załamania nie jest aż tak dobrze widoczny. Pozostałe metody wskazują na 6 komponentów. Sytuacja wygląda bardzo ciekawie jeśli chodzi o analizę sylwetki, występują tam wartości ujemne. Oznaczają one, że wartości z klastrów nakładają się na siebie. Takie wartości nie oznaczają, że nie ma dla nich klastrów, można powiedzieć, że zastosowany algorytm nie może rozdzielić klastrów i trzeba by było rozważyć dostrojenie algorytmu lub użycie innego. Dodatkowo wykonaliśmy wykres analizy k-średnich niebezpiecznych asteroid na tle asteroid nie zagrażających Ziemi. Uczenie nienadzorowane nie najlepiej sprawdza się w wybranym zbiorze danych.

Jeśli chodzi o uczenie nadzorowane, możemy zauważyć znaczną różnicę w modelach bazowych perceptronu oraz SVC względem tych, które zostały przetworzone z użyciem narzędzia walidacji krzyżowej. Jak widać najlepiej sprawdziły się modele lasu losowego oraz drzewa decyzyjnego. Użyte narzędzie przeszukiwania siatki nie poprawiło dokładności mniej precyzyjnych modeli - perceptronu oraz SVC. Sprawdzono również czy da się poprawić precyzję najlepszych modeli - w obydwu przypadkach nie doszło do zmian parametrów.

Dane, które zostały przetworzone przez wybrane algorytmy uczenia maszynowego nie są tak dokładne jak w Sentry. Autorskie narzędzie NASA zostało przystosowane do zbioru danych.

Analiza danych przez wybrane algorytmy uczenia maszynowego powiodła się w przypadku uczenia nadzorowanego. Możemy więc stwierdzić, że cel pracy został osiągnięty. Projekt może zostać rozwinięty o kolejne metody uczenia maszynowego, które nie zostały ujęte. Niniejsza praca może przyczynić się do poprawy skuteczności wykrywania asteroid niebezpiecznych dla Ziemi i zwiększenia bezpieczeństwa ludzkości.

Bibliografia

[1] `https://jupyter.org/` (Dostęp 24.07.2022)

[2] `https://www.python.org/` (Dostęp 24.07.2022)

[3] `https://github.com/JovianML/opendatasets` (Dostęp 24.07.2022)

[4] `https://numpy.org/` (Dostęp 24.07.2022)

[5] `https://pandas.pydata.org/` (Dostęp 24.07.2022)

[6] `https://matplotlib.org/` (Dostęp 24.07.2022)

[7] `https://scipy.org/` (Dostęp 24.07.2022)

[8] `https://boringowl.io/tag/scikit-learn` (Dostęp 24.07.2022)

[9] `https://seaborn.pydata.org/` (Dostęp 24.07.2022)

[10] H. Karttunen, P. Kröger, H. Oja, M. Poutanen, Karl J. Donner, *Fundamental Astronomy*, wydanie piąte, wydawnictwo Springer, 2014

[11] `https://commons.wikimedia.org/wiki/File:Asteroid-Earth.jpg` (Dostęp 04.01.2023)

[12] `https://www.bryk.pl/artykul/asteroidy-ktore-uderzyly-w-ziemie-czy-to-sie-powtorzy` (Dostęp 04.01.2023)

[13] `https://cneos.jpl.nasa.gov/sentry/intro.html` (Dostęp 18.10.2022)

[14] J. E. Prussing, B. A. Conway, *Orbital mechanics*, wydanie drugie, Oxford university Press, 2012

[15] J. S. Lewis, *Physics and Chemistry of the Solar System*, wydanie drugie, Academic press, 2004

[16] J. M. A. Danby *Fundamentals of Celestial Mechanics*, wydanie drugie, Willmann-Bell, Inc., 1988

[17] `https://en.wikipedia.org/wiki/Longitude_of_the_ascending_node` (Dostęp 25.01.2023)

[18] S. Raschka, V. Mirjalili, *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*, wydanie trzecie, Packt Publishing, 2019

[19] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html` (Dostęp 02.12.2022)

[20] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (Dostęp 02.12.2022)

[21] https://scikit-learn.org/stable/modules/clustering.html (Dostęp 02.12.2022)

[22] https://www.scikit-yb.org/en/latest/api/cluster/elbow.html (Dostęp 02.12.2022)

[23] https://scikit-learn.org/stable/modules/clustering.html#k-means (Dostęp 02.12.2022)

[24] https://www.geeksforgeeks.org/ml-k-means-algorithm/ (Dostęp 18.10.2022)

[25] L. Kaufman, P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, wydanie pierwsze, Wiley-Interscience, 2005

[26] https://scikit-learn.org/stable/modules/mixture.html#gmm (Dostęp 02.12.2022)

[27] https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_selection.html (Dostęp 02.12.2022)

[28] https://numlabs.com/pl/blog/co-to-znaczy-dobrze-ocena-jakości-modelu-część-pierwsza-problem-klasyfikacji (Dostęp 05.01.2023)

[29] https://towardsdatascience.com/support-vector-machines-svm-clearly-explained-a-python-tutorial-for-classification-problems-29c539f3ad8 (Dostęp 05.01.2023)

[30] https://scikit-learn.org/stable/modules/linear_model.html#Perceptron (Dostęp 07.01.2023)

[31] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (Dostęp 07.01.2023)

[32] https://predictivesolutions.pl/jak-udoskonalic-algorytm-drzew-decyzyjnych (Dostęp 07.01.2023)

[33] https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm (Dostęp 07.01.2023)

[34] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (Dostęp 15.12.2022)

[35] https://maelfabien.github.io/machinelearning/Explorium4/#grid-search (Dostęp 07.01.2023)

[36] https://scikit-learn.org/stable/modules/cross_validation.html (Dostęp 02.12.2022)

[37] https://www.kaggle.com/nasa/asteroid-impacts?select=orbits.csv (Dostęp 18.10.2022)

[38] https://cneos.jpl.nasa.gov/ca/ (Dostęp 18.10.2022)

[39] C. M. Bishop, *Pattern Recognition and Machine Learning*, wydanie pierwsze, Springer, 2006

[40] https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py (Dostęp 02.12.2022)

[41] https://scikit-learn.org/stable/modules/cross_validation.html (Dostęp 10.12.2022)

[42] S. Lynch, *Dynamical Systems with Applications using Python*, wydanie pierwsze, Birkhauser, 2018

[43] W. -H Steeb, *The Nonlinear Workbook*, wydanie trzecie, World Scientific Publishing, 2005

[44] W. Richert, L. P. Coelho, *Building Machine Learning Systems with Python*, wydanie pierwsze, Packt Publishing, 2013.

[45] M. Vladimír, A. P. Matwijczuk, A. Niemczynowicz, R. A. Kycia, D. Karcz, B. Gładyszewska, L. Ślusarczyk, P. Burg *Chemometric approach to characterization of the selected grape seed oils based on their fatty acids composition and FTIR spectroscopy.*, Sci Rep 11, 19256 (2021). https://doi.org/10.1038/s41598-021-98763-6 (Dostęp 18.10.2022)

[46] D. Kuhlman, *A Python Book: Beginning Python, Advanced Python, and Python Exercises*, wydanie pierwsze, Platypus Global Media, 2011

[47] https://github.com/rasbt?tab=overview&from=2021-12-01&to=2021-12-31 (Dostęp 02.10.2022)

Spis rysunków

Spis tabel