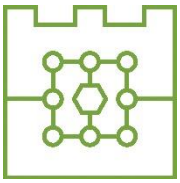
Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Informatyki i Telekomunikacji



Aleksander Kuzmich

Numer albumu: 131957

Optymalizacja kolejności testów regresyjnych poprzez analizę danych historycznych o oprogramowaniu

Optimization of regression tests order by historical software data analysis

Praca inżynierska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem:
dr Radosław Kycia

Kraków 2024

Streszczenie

Niniejsza praca prezentuje zastosowanie algorytmów uczenia maszynowego przy przetwarzaniu, wizualizacji i klasteryzacji danych historycznych o oprogramowaniu w celu wyznaczenia optymalnej puli testów regresyjnych dla kolejnych wersji oprogramowania. W procesie przetwarzania danych wykorzystano metody standaryzacji i normalizacji danych, dla zmniejszenia wymiarowości użyto metodę progu wariancji. Wizualizację danych wykonano używając metodę analizy głównych składowych. Klasteryzację wykonano z wykorzystaniem metody aglomeracyjnej i k-średnich. Opisana w pracy metoda pozwala określić optymalną kolejność testów przy kontroli wykrywalności błędów.

Abstract

The thesis presents the application of machine learning algorithms for the processing, visualization and clustering of historical versions of the software to determine the optimal pool of regression tests for subsequent software versions. Data standardization and normalization methods were used for data preprocessing. Data visualization was performed using the principal component analysis method. Clustering was performed using the agglomerative and k-means methods. Obtained results demonstrate a reduction in the number of tests at the expense of worsening the detection of errors. The method described in the thesis allows determining the optimal order of tests, preserving control over problem detection.

Spis treści

1. Wstęp

W dużych firmach telekomunikacyjnych oprogramowanie jest dynamicznie rozwijane. Ciągła integracja oraz ciągłe wdrażanie podczas rozwoju oprogramowania są szeroko stosowane w dużych przedsiębiorstwach informatycznych. Ten proces zawiera testowanie oprogramowania zapewniające brak powstania krytycznych błędów przy dodaniu nowych funkcjonalności. Każdego dnia powstają kolejne wersje oprogramowania, implementujące nowe funkcjonalności. Wymagane jest ich codzienne testowanie, a produkt jest na tyle duży, że wymaga dużej ilości testów. Kolejka automatycznie wykonywanych testów, które zostaną uruchomione na wybranych wersjach oprogramowania, zwanych kandydatami, jest ciągle rozszerzana. Zdarza się, że dana funkcjonalność jest testowana na każdym kolejnym kandydacie, mimo braku aktualizacji kodu, który implementuje tę funkcjonalność. Naraża to firmę na zbędne koszty. Opisane w pracy badanie sprawdza czy zastosowanie technik nienadzorowanego uczenia maszynowego pomoże skutecznie (z zachowaniem wykrywalności błędów) skrócić kolejkę testów.

1.1. Cel pracy

Celem pracy jest zbadanie skuteczności metod preprocesowania i klasteryzacji danych historycznych o oprogramowaniu przy wyznaczaniu optymalnej kolejki testów regresyjnych. Badanie zostało przeprowadzone w ramach rzeczywistego projektu, wykonanego w firmie Nokia.

1.2. Zakres pracy

W pracy przeanalizowano zbiór okresowo wykonywanych testów regresyjnych, wykonywanych na zbiorze kolejnych wersji oprogramowania oraz zbiór raportów błędów, które zostały wykryte w wyniku wykonania tych testów. Zastosowano próg wariancji dla wyboru odpowiedniego zbioru cech zbioru danych, zawierającego informacje o wersjach oprogramowania. Preprocesowanie cech obejmuje proces standaryzacji albo normalizacji ich wartości. Dla standaryzacji zastosowano StandardScaler, normalizację wartości cech wykonano używając MinMaxScaler. Porównano algorytmy klasteryzacji: AgglomerativeClustering i K-średnich. Dla oceny klasteryzacji zastosowano metryki SilhouetteScore oraz indeks Daviesa-Bouldina. Na podstawie przynależności wersji oprogramowania do któregoś klastra, wyznaczono testy, które mogą mieć większe prawdopodobieństwo odnalezienia błędu.

1.3. Metodyka pracy

Wyznaczenie odpowiedniego zbioru testów, który będzie miał większe prawdopodobieństwo odnalezienia błędów na podanej wersji oprogramowania wymaga wykonania kilku etapów. Pierwszym etapem jest analiza dostępnych danych. Dotyczy to zrozumienia cykli rozwoju oraz wersjonowania oprogramowania, jak również organizacji kolejki okresowo wykonywanych testów oraz ich wyników. Potrzebne jest też zrozumienie cyklu życia raportów wykrytych błędów. Bardziej skomplikowany błąd, wymagający dłuższej analizy i naprawy, może być rejestrowany przez testy w kolejnych wersjach oprogramowania, mimo że został wykryty w jednej konkretnej.

W drugim etapie odbywa się wybór cech, które wyznaczą różnice między kolejnymi kandydatami, ponieważ ilość cech potrafi sięgać kilku tysięcy. Liczebność cech wersji oprogramowania potrafi sięgać kilku tysięcy, co przekłada się na konieczność wyznaczenia zbioru tylko tych, które najbardziej odwzorowują różnice w kolejnych wersjach kodu

Trzeci etap dotyczy zastosowania metod nienadzorowanego uczenia maszynowego – algorytmów klasteryzacji, które podzielą zbiór wersji oprogramowania na klastry.

W czwartym etapie dla każdego kandydata, należącego do testowego zbioru, wyznaczono, do którego klastra on należy oraz przypisano testy, które wykryły błędy na wersjach oprogramowania, należących do tego klastra.

Piąty etap dotyczy porównania skuteczności algorytmu, analizując zbiór wykrytych błędów i skrócenie ilości testów, przypisanych do wykonania.

Badanie zrealizowano za pomocą języka programowania *Python* oraz narzędzia *Jupyter Notebook*, które jest szeroko stosowane przy analizie oraz przetwarzaniu danych. Skorzystano z takich bibliotek jak *Pandas*, *Numpy*, *sci-kit learn*. Do wizualizacji danych użyto bibliotek *Matplotlib* oraz *Seaborn*. Badanie zostało przeprowadzone w ramach współpracy z firmą Nokia, która udostępniła niezbędne dane oraz zdalny serwer, posiadający kartę graficzną NVIDIA GeForce RTX 2070 SUPER o pamięci wewnętrznej 8GB.

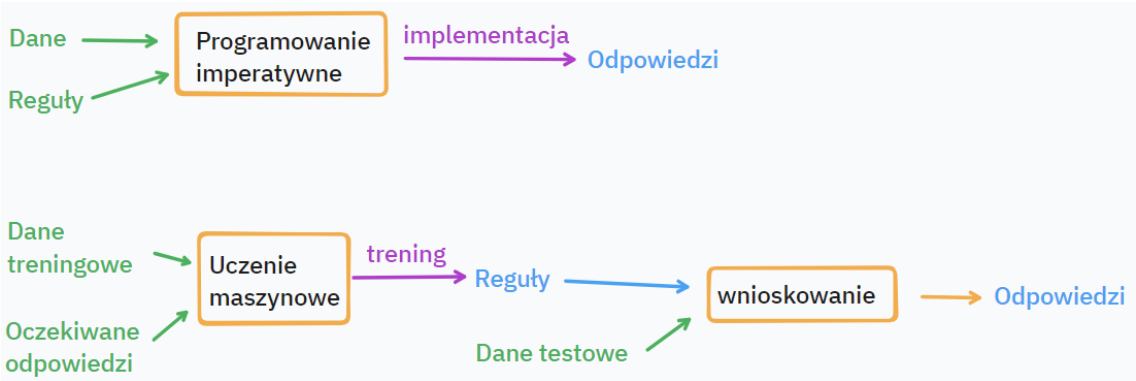
Część teoretyczna składa się z przeglądu metod przetwarzania danych. Przedstawiono algorytmy wyboru cech, standaryzacji wartości oraz klasteryzacji zbioru danych. Omówiono również użyte metryki stosowane dla oceny klasteryzacji. W części praktycznej omówiono kroki, podjęte w celu porównania wybranych algorytmów pod względem skuteczności wyznaczenia odpowiednich testów.

Część teoretyczna

2. Zagadnienia teoretyczne

2.1. Uczenie maszynowe a programowanie imperatywne

Sztuczna inteligencja zyskała ogromną popularność. Jej temat poruszany jest w kręgach naukowych, a media popularnonaukowe opisują jak zmieni świat. Rośnie liczba jej entuzjastów, którzy widząc jej potencjał mają idealistyczne wizje o tym jak wiele problemów ludzkości może ona rozwiązać. [1] Sztuczna inteligencja jest pojęciem bardzo szerokim, dużą część którego stanowi uczenie maszynowe. Podstawowa różnica między programowaniem imperatywnym a uczeniem maszynowym polega na tym, że programowanie imperatywne wymaga od autora określenia wszystkich kroków, których wykonanie prowadzi do rozwiązania problemu, gdy w uczeniu maszynowym model na podstawie dostarczonych danych sam definiuje reguły. Jak przedstawia rys. 2.1, Odpowiedzi są wynikiem zastosowania stworzonych reguł na zbiorze danych testowych.



Rys. 2.1. Schematy rozwiązywania problemu przy użyciu programowania imperatywnego i przy użyciu uczenia maszynowego, opracowanie własne

2.2. Etapy uczenia maszynowego

W dziedzinie uczenia maszynowego istnieją różne metody dostosowane do rozwiązywania zróżnicowanych problemów. Mimo tego, mają one wspólne etapy, które należy wykonać celem uzyskania dobrych wyników. Od zdefiniowania typu problemu zależy wybór algorytmów, które zostaną wybrane dla jego rozwiązania. Najpopularniejszymi typami, do których rozwiązywane problemy zostają sprostowane są: regresja, klasyfikacja oraz klasteryzacja. Jak dobrym nie byłby algorytm, dane uczące o niskiej jakości nie pozwolą na uzyskanie dobrych wyników. Z tego powodu wykorzystywane dane powinny pochodzić z pewnego źródła posiadającego prawdziwe informacje. Potrzebne jest też zrozumienie danych. Nawet wysoka jakość danych uczących nie gwarantuje uzyskania najlepszego wyniku. Trudnym jest przeprowadzić

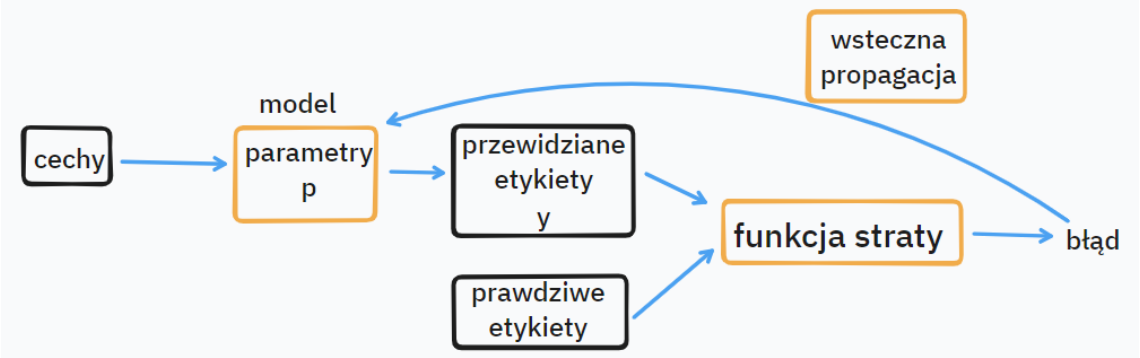
skuteczny trening na nieprzetworzonych danych otrzymanych ze źródła. Dlatego kolejnym bardzo istotnym etapem jest przetwarzanie danych. Obejmuje on wybór istotnych atrybutów, czyszczenie danych, ich transformację i normalizację, a także redukcję. Wybór metod zależy od algorytmu, który będzie zastosowany do rozwiązania problemu [2]. Dla wielu problemów, które mogą zostać rozwiązane z zastosowaniem uczenia maszynowego, powstały algorytmy, stosujące uczenie nadzorowane, nienadzorowane czy przez wzmacnianie. Dobranie odpowiedniego algorytmu i jego parametrów wymaga wielu prób. Często kilka algorytmów i zbiorów parametrów jest stosowano na zbiorze danych celem ich porównania i wybrania najlepszego, dającego zadowalające wyniki przy używaniu specjalnych narzędzi [3]. Określenie, jakie wyniki są zadowalające, jak je mierzyć i porównywać wymaga wybrania metryk – metod pomiaru. Może to być czas treningu i wnioskowania, złożoność pamięciowa, a także metryki oceniające wyniki wnioskowania takie jak dokładność i precyzja. Przy uczeniu nadzorowanym, na etapie treningu, wprowadza się funkcję straty. Jest to metryka, która daje algorytmowi zwrotne informacje o postępach jego uczenia się. [4]. Ostatnim etapem jest etap wnioskowania – wytrenowanemu algorytmowi przedstawia się nowe dane, z dziedziny tego samego problemu i uzyskuje się wyniki. [1] Na rys. 2.2 zaprezentowano etapy uczenia maszynowego.

Rys. 2.2. Etapy uczenia maszynowego, opracowanie własne

2.3. Typy uczenia maszynowego

Nadzorowane uczenie maszynowe polega na trenowaniu modelu przy użyciu zbioru danych, który zawiera pary wejść i odpowiadających im wyjść. Są one nazywane odpowiednio cechami i etykietami. Model, razem z jego parametrami *p,* jest funkcją $f(x, p) = y$, gdzie x to cechy, a y – przewidywania. Trening jest realizowany przez obliczenie funkcji straty i dostosowywanie parametrów modelu. Funkcja straty porównuje przewidziane wyjścia y modelu do prawdziwych etykiet i oblicza wartość

błędu, natomiast mechanizm wstecznej propagacji zmienia parametry modelu tak, aby zminimalizować wartość błędu. Proces ten przedstawiono na rys. 2.3. Wytrenowany model może zostać użyty przy wnioskowaniu etykiet dla nowych danych, należących do tego samego problemu, co zbiór treningowy. [5]



Rys. 2.3. Proces nadzorowanego uczenia maszynowego, opracowanie własne

Przykładowymi metodami uczenia nadzorowanego są regresja oraz klasyfikacja. W przypadku regresji, model przewiduje wartości ciągłe, gdy przy klasyfikacji podanej próbce cech zostaje przypisana klasa, która jest wartością dyskretną, do której próbka należy z najwyższym prawdopodobieństwem.

Przy nienadzorowanym uczeniu maszynowym, model jest trenowany na zbiorze danych wejściowych, które zawierają tylko cechy i nie posiadają etykiet. Algorytm nie posiada dostosowywanych parametrów i stara się odkryć ukryte zależności w danych. Takie metody jak klasteryzacja czy redukcja wymiarowości cech są typowymi przykładami nienadzorowanego uczenia maszynowego. Są one stosowane w celu eksploracji lub wizualizacji danych, ale mogą też stanowić część większego potoku przetwarzania danych.

Istnieją również inne rodzaje uczenia maszynowego. Należą do nich uczenie hybrydowe, które łączy cechy uczenia nadzorowanego i nienadzorowanego oraz uczenie przez wzmacnianie, kiedy algorytm jest nagradzany w przypadku zmniejszenia wartości błędu przy interakcji z otoczeniem.

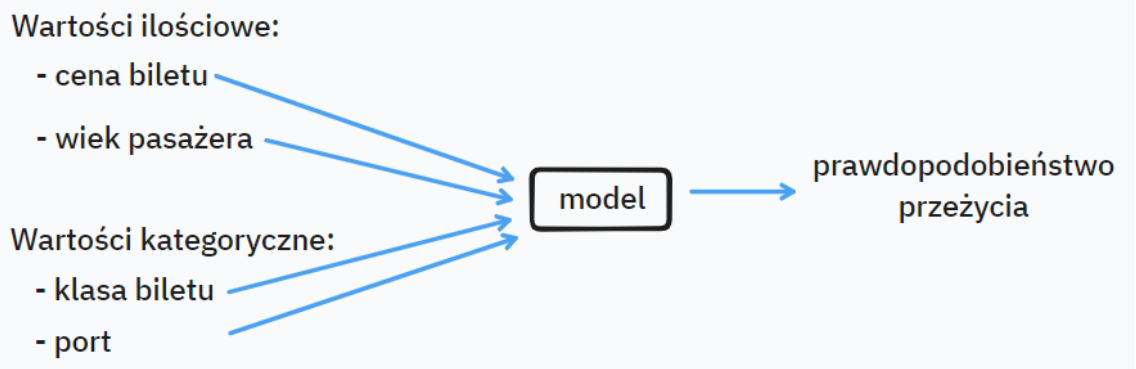
2.4. Rodzaje danych

Dane, używane w uczeniu maszynowym, mogą pochodzić z różnych źródeł. Mogą to być dane obrazowe, wideo czy audio, albo numeryczne – zbierane przy pomiarach z aparatury. Natomiast obecnie dane tekstowe stają się również bardzo popularne ze względu na szeroką dostępność dużych modeli językowych. Niezależnie od

źródła pochodzenia danych, muszą one zostać sprowadzone do postaci liczbowej, którą algorytm będzie w stanie przetworzyć. W uczeniu maszynowym rozróżnia się dwa podstawowe rodzaje danych – dane ilościowe oraz dane kategoryczne.

Dane ilościowe to dane, które mogą być zmierzone. Pochodzą z rozkładu ciągłego albo dyskretnego. Przykładem mogą być wartości ceny biletu i wieku pasażera w zbiorze danych dotyczącym przewidywania prawdopodobieństwa przeżycia na statku RMS Titanic [6], który został wybrany do ilustracji metod opisanych w części teoretycznej metod. Cena biletu przedstawia wartość z rozkładu ciągłego, gdy wiek jest wartością dyskretną. Dane ilościowe często poddaje się normalizacji w przedziale od zera do jeden albo standaryzacji.

Dane kategoryczne cechują się tym, że opisują ograniczoną liczbę potencjalnych wartości. [7] Są kodowane liczbami, gdzie każda liczba przedstawia pewną kategorię. Najczęściej dla danych kategorycznych stosuje się kodowanie za pomocą *Kodu 1 z n* [8], gdzie indeks z wartością 1 koduje kategorię. Przykładem z wyżej wspomnianego zbioru danych mogą być klasa zakupionego biletu, która mogła przyjmować wartości ze zbioru [1, 2, 3] albo litera reprezentująca nazwę portu, w którym dosiadł pasażer: C = Cherbourg, Q = Queenstown albo S = Southampton. Na rys. 2.4 przedstawiono wykorzystanie danych ilościowych i kategorycznych dla wyznaczenia prawdopodobieństwo przeżycia pasażera.

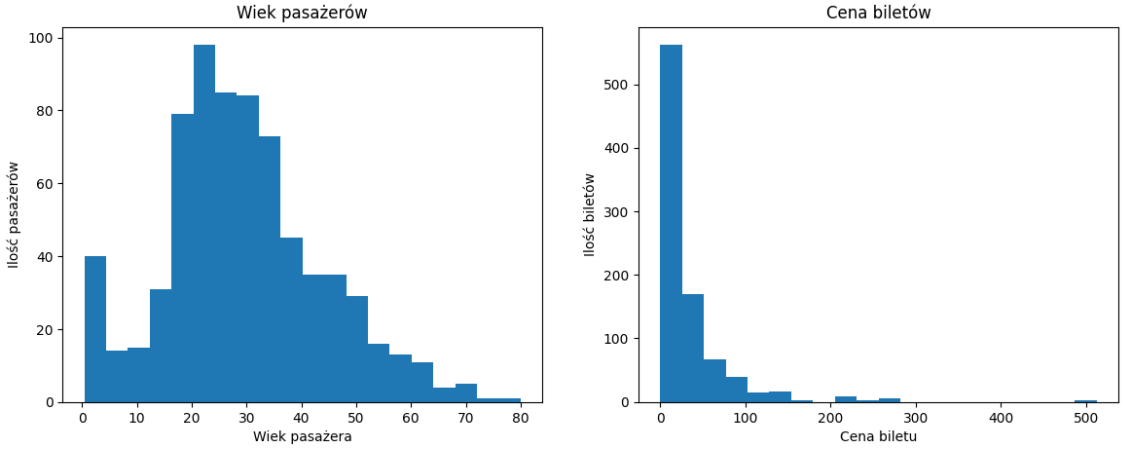


Rys. 2.4. Wykorzystanie danych ilościowych i kategorycznych przez model w celu wyznaczenia prawdopodobieństwa przeżycia pasażera, opracowanie własne

2.5. Przygotowywanie wartości cech

Techniki standaryzacji i normalizacji stanowią jeden z kluczowych kroków w procesie przygotowywania danych. Zastosowanie jednej z nich zapewnia spójność cech i poprawia wydajność algorytmów uczenia maszynowego. Dane ilościowe takie jak wiek czy cena biletu ze wspomnianego wyżej zbioru danych z platformy *Kaggle* [6], posiadają różne zakresy wartości, które przedstawiono na rys. 2.5. Może to prowadzić do tego, że

któraś z cech zdominuje inne ze względu na swoją jednostkę lub zakres wartości. Tym samym, maksymalna wartość ceny biletu może wynosić 512.3292, natomiast maksymalna wartość wieku – 80:



Rys. 2.5. Rozkład ilości pasażerów względem wieku pasażerów i ilości biletów względem ich ceny, opracowanie własne

Standaryzacja pozwoli na dostosowanie cech do podobnej skali i wykluczenie dysproporcji wartości związanych z ich różnymi jednostkami. Zgodnie ze wzorem (2.1), wartości zostaną przekształcone tak, by miały średnią wartość równą zero i odchylenia standardowe równe jeden, co pozwoli na porównanie ich ze sobą w bardziej spójny sposób. [13]

$$z = \frac{x - \mu}{\sigma}, \tag{2.1}$$

gdzie x to zmienna niestandaryzowana, μ to średnia z populacji, σ to odchylenie standardowe populacji.

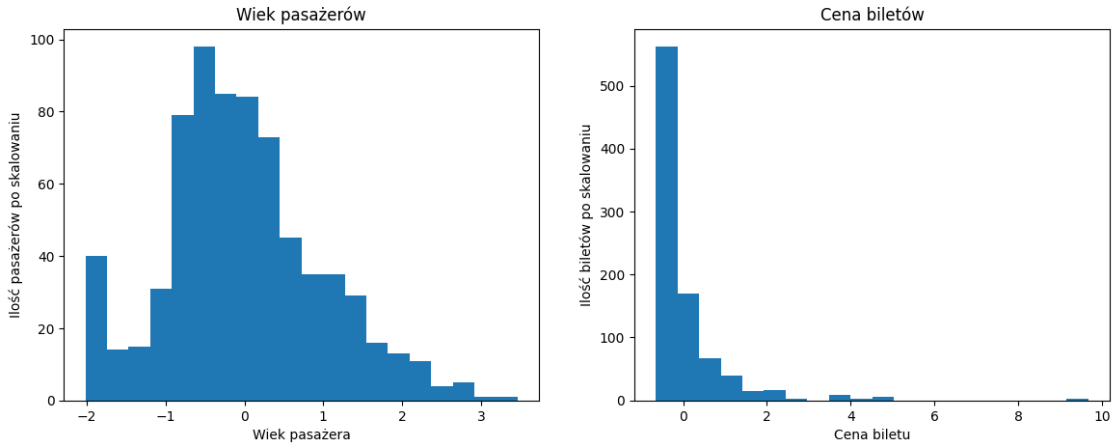
W języku programowania *Python*, do realizacji skalowania można wykorzystać bibliotekę *sci-kit learn*, która implementuje klasę *StandardScaler*. Listing 2.1 przedstawia przekształcenie nieprzetworzonych danych, które zaimportowano do klasy *DataFrame* biblioteki *pandas* i następnie przekazano do metody *fit_transform*.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(titanic[["Wiek", "CenaBiletu"]])

titanic["WiekStandardScaler"] = scaled_data[:, 0]
titanic["CenaBiletuStandardScaler"] = scaled_data[:, 1]
```

Listing 2.1. Kod python skalujący dane wartości wieku pasażerów i ceny biletów, opracowanie własne

StandardScaler posiada kilka ograniczeń względem danych. Dane powinny mieć rozkład zbliżony do rozkładu normalnego oraz nie powinny zawierać dużych wartości odstających, które mogą prowadzić do nieprawidłowego skalowania. Dane muszą posiadać rozkład symetryczny, natomiast skośne rozkłady mogą prowadzić do nieprawidłowego przekształcenia danych. Na rys. 2.6 przedstawiono wyniki skalowania danych o wieku pasażerów i cenach biletów. Odstające wartości ceny nie zostały usunięte, co jest widoczne na wykresie.



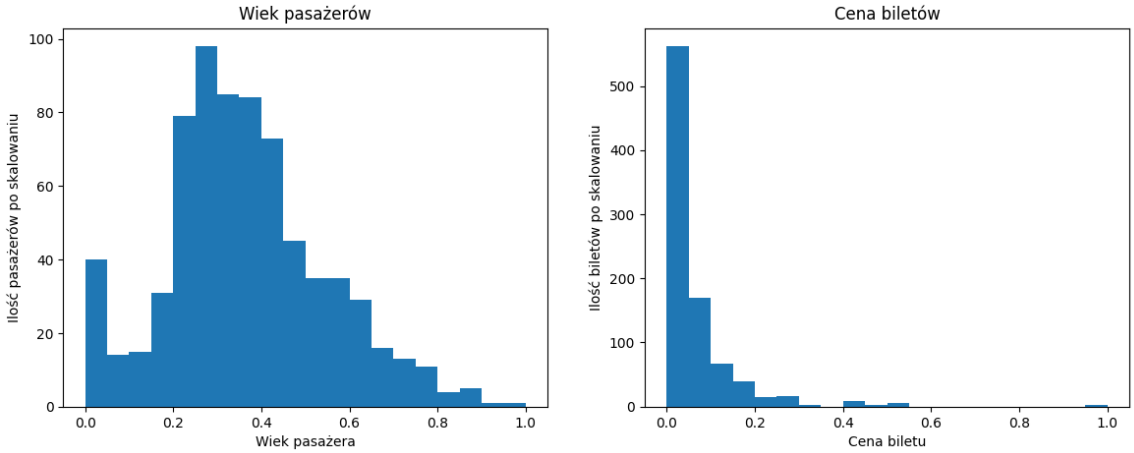
Rys. 2.6. Rozkład ilości pasażerów względem skalowanej wartości wieku pasażerów i ilości biletów względem ich skalowanej ceny, opracowanie własne

Normalizacja jest metodą, która przekształca dane nieprzetworzone do ustalonego zakresu, którym często jest zakres od zera do jeden. Ułatwia to analizę i interpretację danych, szczególnie w przypadku algorytmów opartych na metrykach odległości, gdzie istotne jest zachowanie spójności między cechami. Klasa *MinMaxScaler* z biblioteki *sci-kit learn* implementuje algorytm normalizacji. Przykład wykonania normalizacji nieprzetworzonych danych o wieku pasażerów i cenach biletów jest przedstawiony na listingu 2.2, natomiast przetworzone wartości pokazano na rys. 2.7.

```
from sklearn.preprocessing import MinMaxScaler
normalizer = MinMaxScaler()
normalized_data = normalizer.fit_transform(titanic[["Wiek", "CenaBiletu"]])

titanic["WiekMinMax"] = normalized_data[:, 0]
titanic["CenaBiletuMinMax"] = normalized_data[:, 1]
```

Listing 2.2. Kod python normalizujący dane wartości wieku pasażerów i ceny biletów w zakresie od zera do jeden, opracowanie własne



Rys. 2.7. Rozkład ilości pasażerów względem znormalizowanej wartości wieku pasażerów i ilości biletów względem znormalizowanej wartości ceny biletów, opracowanie własne

2.6. Selekcja cech

Próbka w przygotowanym zbiorze danych jest przedstawiona poprzez wiele wartości, opisujących jej cechy. W uczeniu maszynowym przyjmuje się, że każda cecha przedstawia osobny wymiar, co pozwala zaprezentować próbkę w wielowymiarowej przestrzeni wektorowej. Nie wszystkie cechy są tak samo istotne, niektóre mogą mieć mniejszy wpływ na przynależność próbki do klasy czy zbioru, a inne mogą być nadmiarowe w tym sensie, że ich wartości mogą pochodzić z innych cech: jak przedstawia rys. 2.8, wiek może być obliczony na podstawie daty urodzenia:

	0	1	2	3	4
data_urodzenia	1999-05-24	1994-01-14	2014-03-19	1979-02-27	1989-04-22
wiek	25	30	10	45	35

Rys. 2.8. Przykładowe wartości daty urodzenia i odpowiadające im wartości wieku, opracowanie własne

Wiąże się to z utrudnioną analizą i przetwarzaniem takich danych. Próbki opisane przez tysiące cech często stanowią rzadkie macierzy danych, co potrafi znacząco obniżyć skuteczność wielu algorytmów uczenia maszynowego [9].

Selekcja cech jest procesem wyboru podzbioru istotnych cech ze zbioru danych. Umożliwia skupienie się na najbardziej znaczących informacjach i użycie zbioru danych o mniejszym rozmiarze w procesie uczenia maszynowego.

2.6.1 Metoda progu wariancji

Jedną z kluczowych miar stosowanych w procesie wyboru cech w uczeniu maszynowym jest wariancja. Zgodnie ze wzorem (2.2), wariancja jest wartością

oczekiwaną kwadratu różnic pomiędzy wartością zmiennej losowej a jej wartością oczekiwaną.[10]

$$Var(X) = E[(X - E(X))^2], \tag{2.2}$$

gdzie *Var(X)* to wariancja, *E* to wartość oczekiwana, *X* to zmienna losowa.

Cechy o niskiej wariancji są mało zróżnicowane i mogą nie wnosić istotnych informacji o próbkach dla modelu. Metoda progu wariancji (*VarianceThreshold)* polega na usunięciu cech, których wariancja jest poniżej określonej wartości albo jest zerowa [11]. Dla wyznaczenia progu wariancji mogą być zastosowane percentyle. Percentyl jest miarą, która wskazuje, jaka część danych, wyrażona w procentach, znajduje się poniżej określonej wartości. Na przykład, jeśli dana wartość znajduje się w 70. percentylu, oznacza to, że 70% wszystkich obserwacji w zbiorze danych ma wartość mniejszą lub równą tej wartości. Stosując percentyl 70% wariancji wszystkich cech, jako próg wariancji, uzyska się 30% cech o najwyższej wariancji w zbiorze danych.

2.6.2. PCA

Analiza składowych głównych (Principal Component Analysis, PCA) stawia na celu zmniejszenie wymiarowości zbioru danych przy umożliwieniu kontroli nad ilością zachowanej informacji. Algorytm zmniejsza ilość cech poprzez rzutowanie danych na zbiór wektorów, zwanych głównymi składowymi (Principal Components). Zgodnie ze wzorem (2.3), problem sprowadza się do znalezienia kolejnych zmiennych składowych Z_k, skierowanych wzdłuż wektorów własnych a_k macierzy kowariancji [12].

$$Z_k = a_{k1}X_1 + a_{k2}X_2 + \cdots + a_{kj}X_j = \sum_{j=1}^{d} a_{kj}X_j, \tag{2.3}$$

gdzie Z_k to zmienna składowa, $a_{k1}, a_{k2}, \ldots, a_{kj}$ to elementy wektora własnego macierzy kowariancji, X_j to zmienna losowa cechy, d – ilość cech w zbiorze danych. Macierz kowariancji, przedstawiona wzorem (2.4), reprezentuje wartości wariancji cech na głównej przekątnej oraz wartości kowariancji dla par cech.

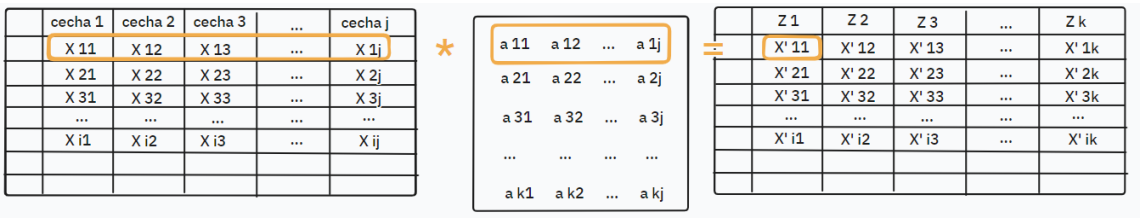
$$C = \begin{bmatrix} var(X_1) & cov(X_1, X_2) & \ldots & cov(X_1, X_n) \\ cov(X_2, X_1) & var(X_2) & \ldots & cov(X_2, X_n) \\ \ldots & \ldots & \ldots & \ldots \\ cov(X_n, X_1) & cov(X_n, X_2) & \ldots & var(X_i) \end{bmatrix}, \tag{2.4}$$

gdzie $var(X_i)$ to wariancja zmiennej losowej X_i, $cov(X_i,X_j)$ to kowariancja między zmiennymi losowymi X_i i X_j. Kowariancja jest miarą, która opisuje jak wartości dwóch cech się wzajemnie zmieniają. Jeżeli zmienne jednocześnie wzrastają, kowariancja jest dodatnia. Gdy jedna zmienna rośnie, a druga maleje, kowariancja posiada wartość

ujemną. Wartość bliska zero oznacza, że zmienne są nieskorelowane, czyli ich zmiany nie są ze sobą związane. Kowariancję przedstawia się wzorem (2.5):

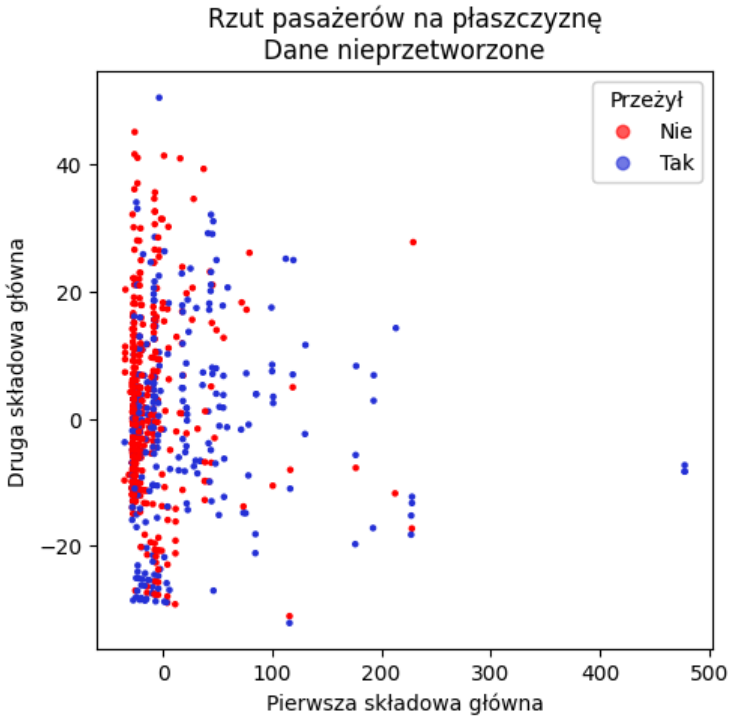
$$cov(X_i, X_j) = E(X_i \cdot X_j) - EX_i \cdot EX_j, \tag{2.5}$$

gdzie X_i, X_j – zmienne losowe, E – wartość oczekiwana. Ilość nowych składowych nie może przekroczyć ilość cech w zbiorze danych, natomiast najczęściej wybiera się możliwie najmniejszą ilość głównych składowych. Analiza składowych głównych stosowana jest nie tylko w celu redukcji wymiaru, ale również w celu kompresji sygnałów czy eksploracji danych i ich klasteryzacji. Należy pamiętać, że PCA jest wrażliwa na zakres wielkości danych, dlatego często stosowanym rozwiązaniem tego problemu stanowi standaryzacja danych. Na rys. 2.9 przedstawiono proces przetwarzania danych z wykorzystaniem obliczonych wektorów *a*:



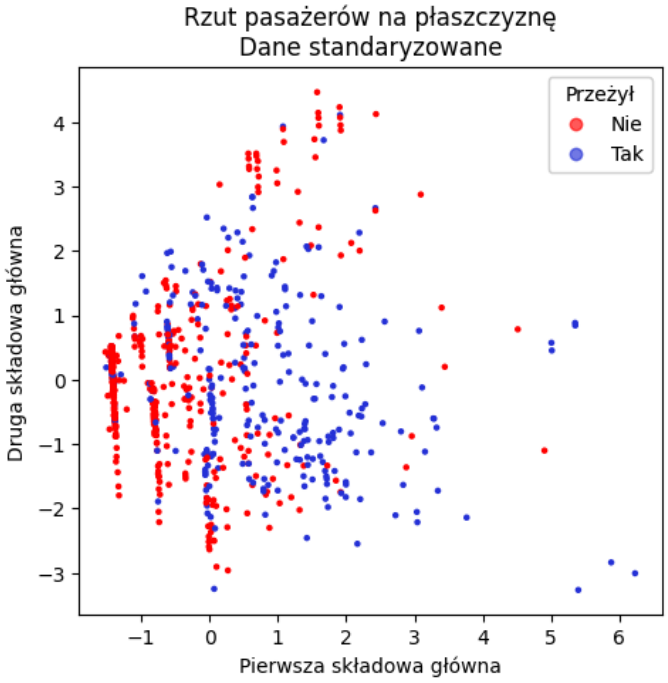
Rys. 2.9. Proces przetwarzania danych z wykorzystaniem obliczonych wektorów składowych głównych, opracowanie własne

Przy stosowaniu metody PCA należy wziąć pod uwagę fakt, że jest ona wrażliwa na zakres wielkości danych. Często stosowanym rozwiązaniem tego problemu jest standaryzacja lub normalizacja danych. Aby zilustrować przeżywalność pasażerów na dwuwymiarowym wykresie, należy wybrać dwie pierwsze składowe główne wygenerowane przez analizę głównych składowych (PCA) i zastosować je, jako osie X i Y. Na rys. 2.10 przedstawiono rozkład pasażerów w sytuacji, gdy dane wejściowe nie były poddane standaryzacji ani normalizacji. Rozkład punktów jest nierównomierny, co wskazuje na różnice w skalach poszczególnych cech. Widoczne są wartości ekstremalne, które mogą wpływać na jakość i interpretację analizy PCA.



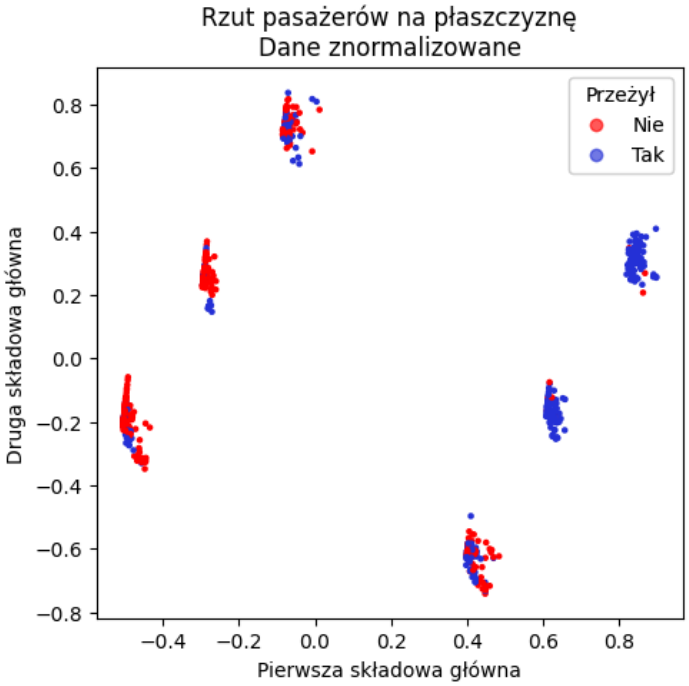
Rys. 2.10. Reprezentacja zbioru danych z zastosowaniem metody PCA bez procesowania wartości cech, opracowanie własne

Na rys. 2.11 jest przedstawiony wynik zastosowania PCA na danych, które zostały poddane standaryzacji. Wartości cech zostały przekształcone tak, aby miały średnią równą zero i odchylenie standardowe równe jeden. Różnice w skalach cech zostały zredukowane, w rezultacie punkty są bardziej skupione wokół środka wykresu.



Rys. 2.11. Reprezentacja zbioru danych z zastosowaniem metody PCA na wartościach cech poddanych standaryzacji, opracowanie własne

Na rys. 2.12 przedstawiono dane znormalizowane, wartości cech zostały przeskalowane do przedziału [0; 1]. Widoczne jest, że punkty są bardziej uporządkowane i znajdują się w ograniczonym zakresie wartości.



Rys. 2.12. Reprezentacja zbioru danych z zastosowaniem metody PCA na wartościach cech poddanych normalizacji, opracowanie własne

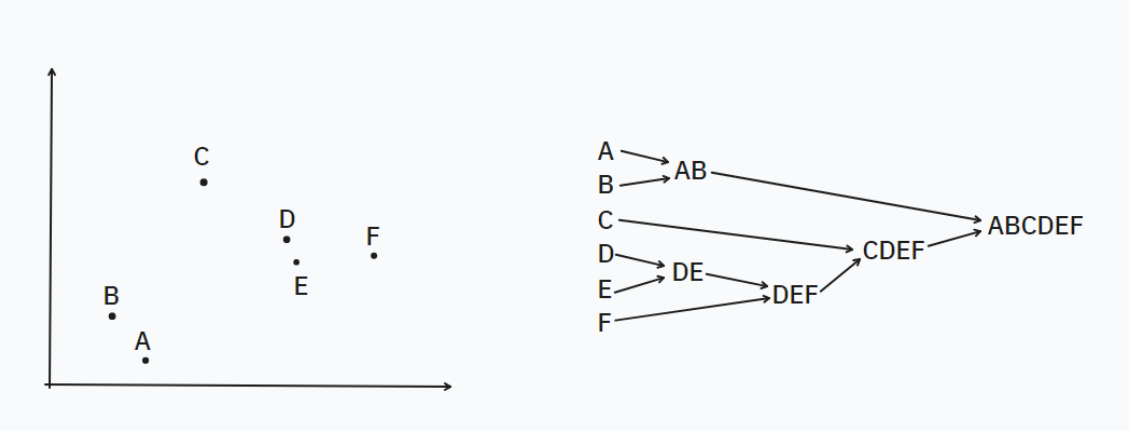
2.7. Klasteryzacja

Przy użyciu klasteryzacji rozwiązywane są problemy dotyczące uczenia nienadzorowanego. W odróżnieniu od klasyfikacji, gdzie zbiór danych treningowych składa się z danych wejściowych i wyjściowych, klasteryzacja polega na podziale zbioru danych wejściowych na mniejsze grupy nazywane klastrami. Podział danych polega na zmniejszeniu odległości pomiędzy elementami należącymi do jednej grupy i zwiększeniu odległości pomiędzy elementami należącymi do różnych grup. Klasteryzacja jest często stosowana przy eksploracji danych, ponieważ pozwala wykrywać ukryte zależności w nich.

2.7.1. Klasteryzacja hierarchiczna

Algorytmy klasteryzacji wykorzystują różne techniki grupowania danych. Klasteryzacja hierarchiczna dzieli się na aglomeracyjną i deaglomeracyjną. Podejście aglomeracyjne przypisuje osobne klastry do każdego elementu zbioru danych, następnie najbliższe elementy są łączone w nowe klastry przy wykorzystaniu wybranej metryki odległości. Na rys. 2.13 przedstawiony jest przebieg łączenia elementów w klastry aż do

uzyskania jednego klastra. Metoda deaglomeracyjna początkowo łączy wszystkie elementy w jedną grupę, następnie ta grupa jest dzielona na mniejsze klastry [15]. Tworzenie nowych grup elementów jest przerywane, gdy dalsze ich łączenie prowadzi do klastrów, które są niepożądane z jednego spośród kilku powodów. Klasteryzacja może zostać zatrzymana przy osiągnięciu określonej liczby klastrów albo przy próbie stworzenia klastrów z punktów, zbyt daleko rozłożonych od siebie w przestrzeni.

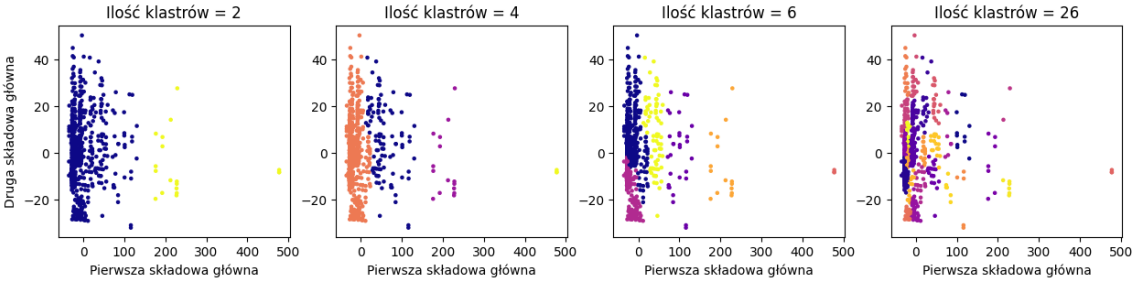


Rys. 2.13. Proces łączenia elementów w klastry z zastosowaniem metody aglomeracyjnej, opracowanie własne

W przestrzeniach euklidesowych o dużej ilości wymiarów może zachodzić zjawisko nazywane „Przekleństwem wymiarowości", gdy większość punktów okazuje się podobnie oddalona od siebie. Rozważmy *d*-wymiarową przestrzeń euklidesową. Punkt w takiej przestrzeni jest przedstawiony wektorem $[x_1, x_2, \ldots, x_d]$, gdzie x_i należy do przedziału $[0; 1]$. Odległość euklidesowa [19] pomiędzy dwoma losowymi punktami $[x_1, x_2, \ldots, x_d]$ and $[y_1, y_2, \ldots, y_d]$ jest przedstawiona wzorem (2.6). Przy zwiększeniu wymiaru *d*, odległość pomiędzy punktami *x* oraz *y* będzie zbliżała się do średniej wartości odległości pomiędzy wszystkimi parami punktów. [14] Brak punktów, które są wyraźnie zbliżone lub oddalone znacząco utrudnia klasteryzacje.

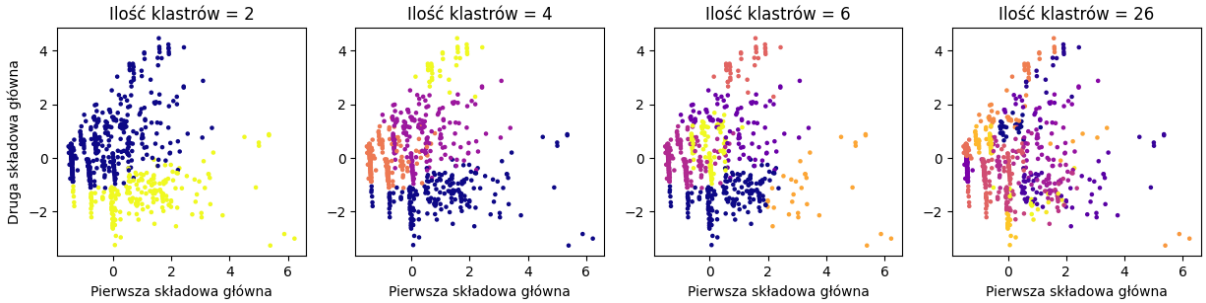
$$\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}, \tag{2.6}$$

gdzie x_i oraz y_i to wartości cech punktów *x* i *y*, *d* to ilość wymiarów (liczba cech). W zbiorze danych o pasażerach RMS Titanic [6], każdy pasażer jest opisany przez kilka cech. Analiza składowych głównych umożliwia wizualizację wyników klasteryzacji aglomeracyjnej tego zbioru na płaszczyźnie utworzonej przez dwie pierwsze składowe główne. Na rys. 2.14 każdy klaster jest przedstawiony innym kolorem.



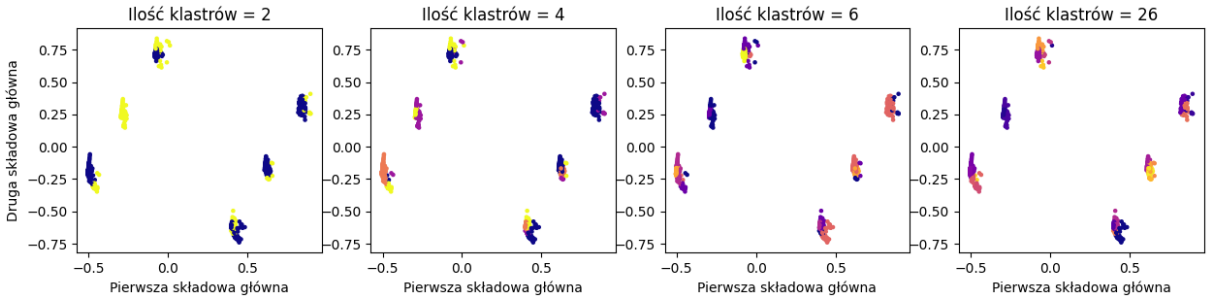
Rys. 2.14. Reprezentacja zbioru danych z zastosowaniem metody PCA bez procesowania wartości cech. Kolory przypisano na podstawie przynależności elementu do klastrów wyznaczonych metodą aglomeracyjną, opracowanie własne

Użycie danych standaryzowanych może mieć istotny wpływ na proces klasteryzacji, szczególnie w przypadku algorytmów, które opierają się na miarach odległości. Każda cecha ma średnią równą zero i odchylenie standardowe równe jeden. Na rys. 2.15 przedstawiono wyniki klasteryzacji danych standaryzowanych.



Rys. 2.15. Reprezentacja zbioru danych z zastosowaniem metody PCA na wartościach cech poddanych standaryzacji. Kolory przypisano na podstawie przynależności elementu do klastrów wyznaczonych metodą aglomeracyjną, opracowanie własne

Normalizacja nieprzetworzonych danych skaluje wartości cech w określonym przedziale, co umożliwia jednakowy wpływ wszystkich cech na proces klasteryzacji. Dzięki temu algorytmy klasteryzacji mogą lepiej wykorzystać informacje zawarte we wszystkich cechach danych. Na rys. 2.16 przedstawiono wyniki klasteryzacji danych znormalizowanych w przedziale [0; 1].



Rys. 2.16. Reprezentacja zbioru danych z zastosowaniem metody PCA na wartościach cech poddanych normalizacji. Kolory przypisano na podstawie przynależności elementu do klastrów wyznaczonych metodą aglomeracyjną, opracowanie własne

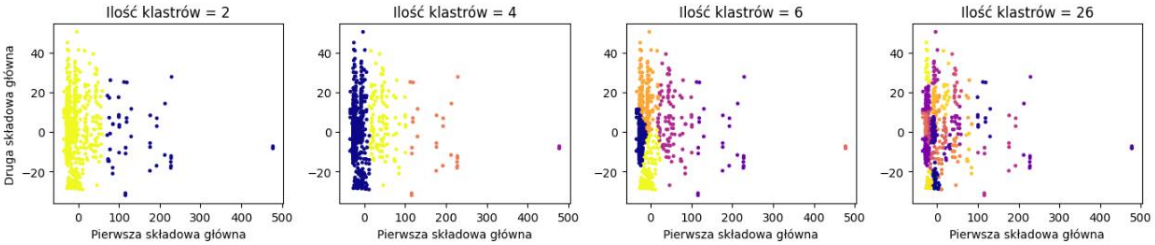
2.7.2. Klasteryzacja metodą k-średnich

Klasteryzacja metodą k-średnich jest jedną z najpopularniejszych i najczęściej stosowanych technik grupowania danych stosowanych do rozpoznawania ukrytych wzorców w danych. [16] Algorytm polega na podziale zbioru danych na określoną ilość klastrów k w taki sposób, aby zminimalizować funkcję J (wzór (2.7)), która jest definiowana, jako suma kwadratów odległości punktów danych od odpowiadających im centroidów [17].

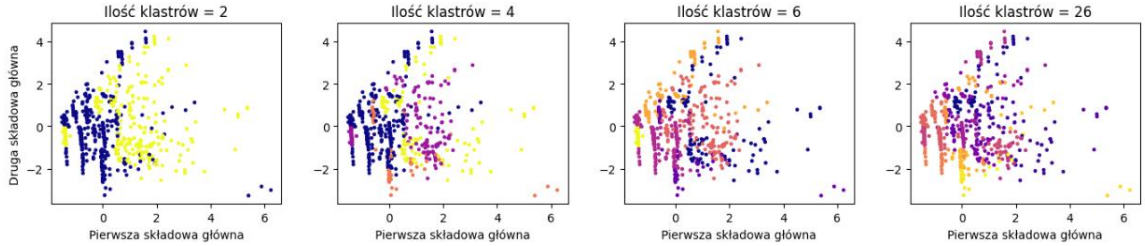
$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} \left\| x_j - \mu_i \right\|^2, \tag{2.7}$$

gdzie C_i to zbiór punktów danych w i-tym klastrze, μ_i to centroid i-tego klastra, a $\left\| x_j - \mu_j \right\|$ to odległości punktów danych od odpowiadających im centroidów.

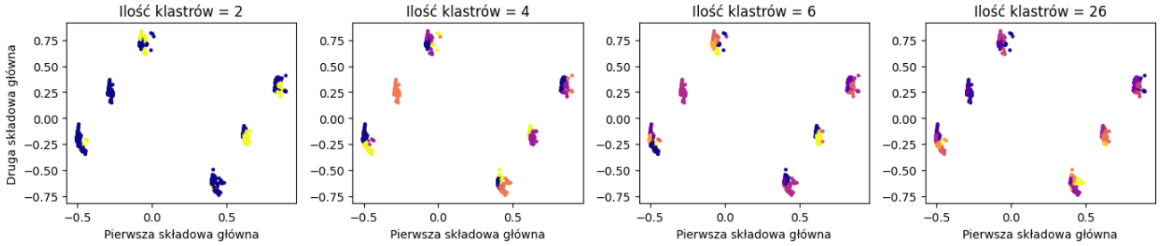
Proces klasteryzacji metodą k-średnich zaczyna się od wyboru k początkowych centroidów, następnie każdy punkt zbioru danych jest przypisywany do najbliższego centroidu na podstawie wybranej miary odległości. Najczęściej jest to odległość euklidesowa. Tak są formowane k początkowych klastrów. Następnie dla każdego klastra jest obliczany nowy centroid, jako średnia punktów należących do tego klastra. Kroki przypisania punktów i aktualizacji centroidów są powtarzane do momentu, gdy centroidy przestają się znacząco zmieniać lub po osiągnięciu maksymalnej liczby iteracji. Wyniki klasteryzacji metodą k-średnich są przedstawione na rysunkach: 2.17 dla danych nieprzetworzonych, 2.18 dla danych standaryzowanych, 2.19 dla danych znormalizowanych. Każdy klaster jest przedstawiony innym kolorem.



Rys. 2.17. Reprezentacja zbioru danych z zastosowaniem metody PCA bez procesowania wartości cech. Kolory przypisano na podstawie przynależności elementu do klastrów wyznaczonych metodą k-średnich, opracowanie własne



Rys. 2.18. Reprezentacja zbioru danych z zastosowaniem metody PCA na wartościach cech poddanych standaryzacji. Kolory przypisano na podstawie przynależności elementu do klastrów wyznaczonych metodą k-średnich, opracowanie własne



Rys. 2.19. Reprezentacja zbioru danych z zastosowaniem metody PCA na wartościach cech poddanych standaryzacji. Kolory przypisano na podstawie przynależności elementu do klastrów wyznaczonych metodą k-średnich, opracowanie własne

2.8. Ocena algorytmów klasteryzacji

2.8.1. Wskaźnik Silhouette

Algorytm obliczenia metryki Silhouette [18] polega na zmierzeniu podobieństwa pojedynczej próbki danych do innych próbek, należących do jej klastra w przeciwieństwie do obiektów w innych klastrach. Każdej próbce jest przypisywana wartość, która zostaje obliczona, jako średnia różnica pomiędzy odległościami do obiektów w jej klastrze a odległościami do obiektów należących do najbliższego sąsiedniego klastra. Wartość wskaźnika Silhouette należy do przedziału [-1; 1], gdzie wartości bliskie 1 oznaczają dobre dopasowanie elementu do klastra. W celu wyznaczenia wartości Silhouette dla zbioru danych oblicza się średnią wartość wskaźników Silhouette wszystkich elementów zbioru. Algorytm wyznaczenia wartości Silhouette dla próbki i należącej do klastra A oblicza średnią odległość a_i do wszystkich elementów należących

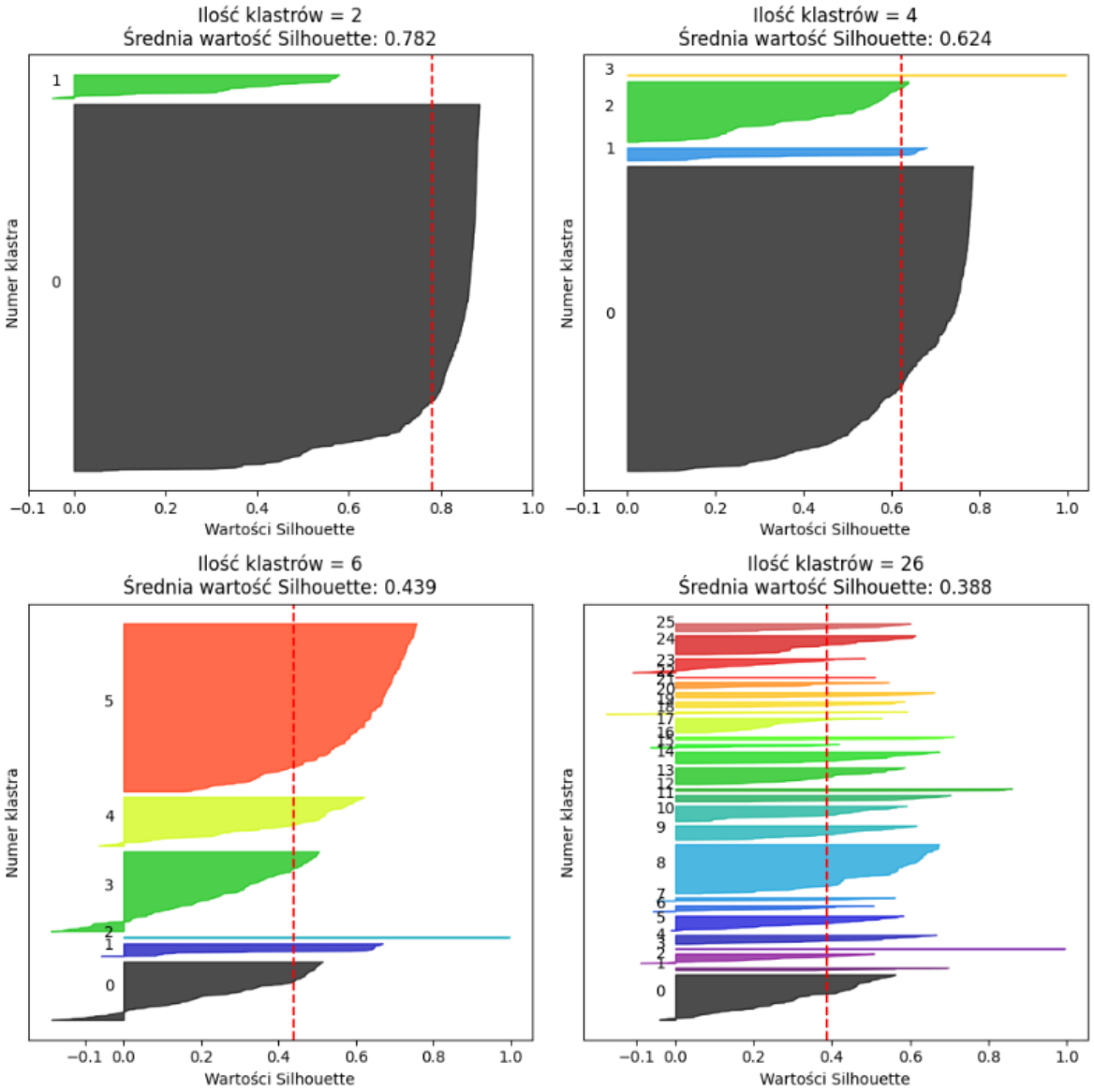
do tego klastra. Dla elementów, należących do innych klastrów niż *A*, zostają obliczone odległości d_i. Zostaje wyznaczony klaster *B,* inny niż *A*, w którym znalazł się element, odległość *d(i)* do którego okazała się najmniejszą. Ten klaster zostaje oznaczony, jako „sąsiedni" do elementu *i*. Następnie obliczona zostaje średnia odległość *b(i)* do wszystkich elementów klastra *B.* Wartość wskaźnika Silhouette dla elementu *i* zostaje obliczona zgodnie ze wzorem (2.8)

$$s(i) = \frac{b_i - a_i}{\max(a_i, b_i)}, \tag{2.8}$$

gdzie *i* to wybrana próbka klastra *A*, a_i to średnia odległość próbki *i* do próbek klastra *A*, b_i to odległości próbki *i* do próbek, nie należących do klastra *A*.

Jednym z zastosowań wskaźnika Silhouette jest analiza wartości wskaźnika dla poszczególnych elementów zbioru danych. Pozwala to wyznaczyć elementy, które zostały źle przypisane do klastrów. [18]

Algorytm wyznaczenia metryki Silhouette w języku programowania Python jest zaimplementowany w bibliotece *sci-kit learn*. Wyniki obliczeń wskaźników Silhouette dla niestandaryzowanych danych o pasażerach dla 2, 4, 6 oraz 26 klastrów są przedstawione na rys. 2.20. Klastry przypisano używając metody k-średnich. Czerwoną linią zaznaczono średnią wartość Silhouette dla zbioru danych. Widoczne jest, że dla 2, 6 oraz 26 klastrów wartość Silhouette jest poniżej 0, co może świadczyć o błędnym przypisaniu elementu do klastra.



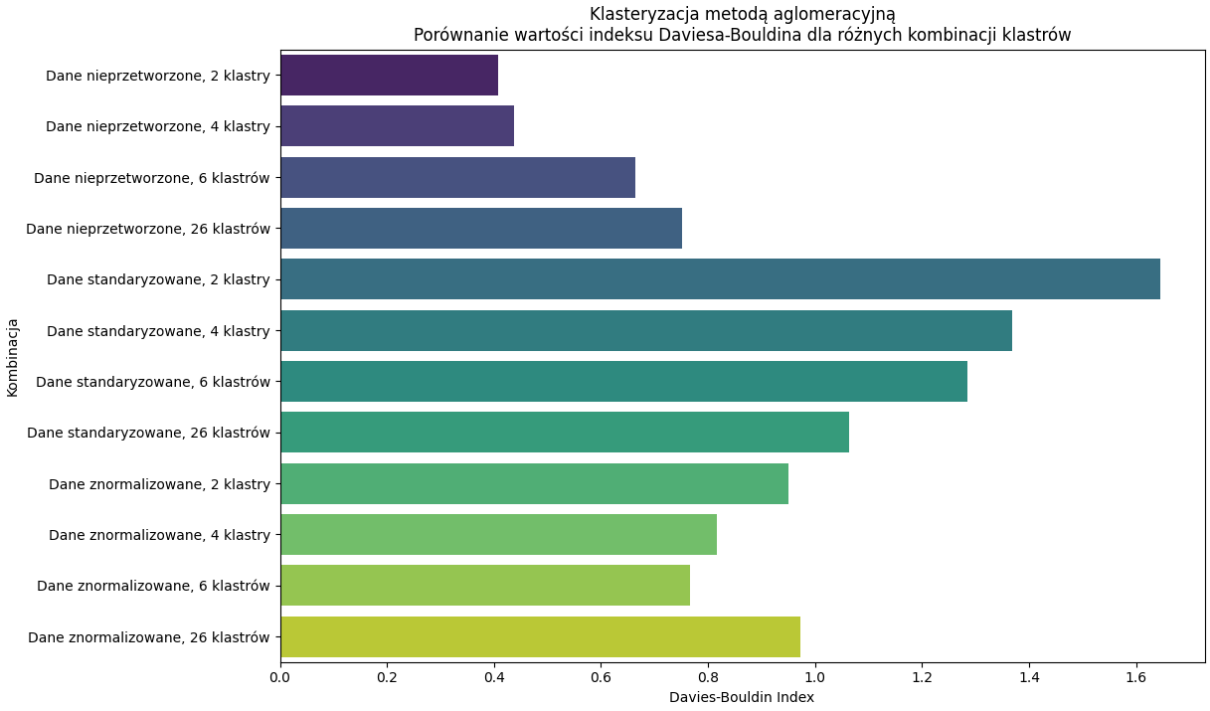
Rys. 2.20. Wartości wskaźnika Silhouette dla każdego elementu zbioru nieprzetworzonych danych przy klasteryzacji metodą k-średnich dla 2, 4, 6, 26 klastrów, opracowanie własne

W celu porównania metod klasteryzacji i znalezienia odpowiedniej ilości klastrów jest używany indeks Daviesa-Bouldina [20]. Jest on oparty na analizie średniej odległości pomiędzy elementami jednego klastra oraz odległości pomiędzy klastrami. Dla zmierzenia odległości jest często stosowana odległość euklidesowa, aczkolwiek inne miary mogą być użyte w przypadku wielowymiarowych danych. Wartość indeksu zmniejsza się, gdy odległości pomiędzy klastrami rosną, a elementy wewnątrz jednego klastra zostają zbliżone do siebie. W ten sposób mniejsza wartość indeksu Daviesa-Bouldina wskazuje na lepszy wynik klasteryzacji. Dla zbioru k klastrów algorytm w pierwszym kroku oblicza średnią odległość ΔX_k między punktami w każdym klastrze. Następnie dla każdej pary klastrów (X_i, X_j) są obliczane odległości $\delta(X_i, X_j)$ pomiędzy

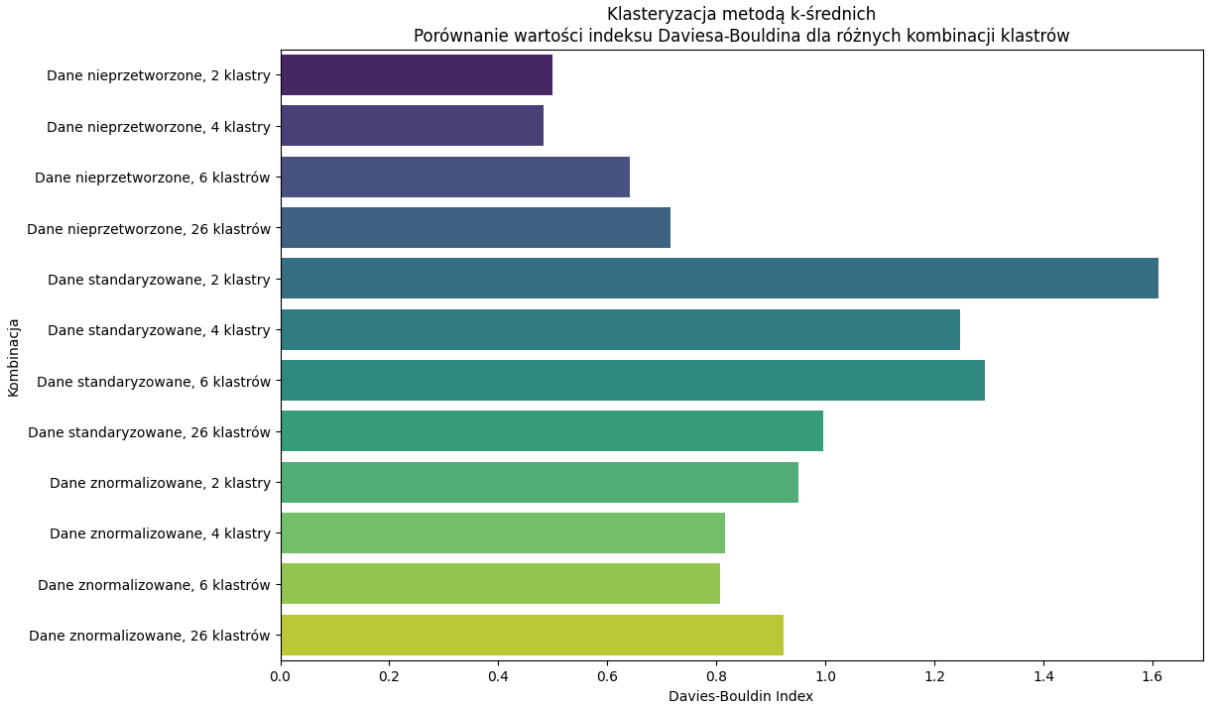
środkami klastrów, które przedstawiają oddzielenie klastrów od siebie. W kolejnym kroku jest obliczany stosunek sumy średnich odległości między punktami w każdym z dwóch klastrów do odległości pomiędzy środkami tych klastrów. Ten wynik przedstawia podobieństwo klastrów. Dla każdego klastra X_k jest wybierana największa wartość podobieństwa. Następnie, zgodnie ze wzorem (2.9) z wybranych wartości jest obliczana średnia, która stanowi indeks Daviesa-Bouldina:

$$\mathrm{DB} = \frac{1}{k}\sum_{i=1}^{k} \max\left(\frac{\Delta X_i + \Delta X_{j\neq i}}{\delta(X_i, X_{j\neq i})}\right), \tag{2.9}$$

gdzie k to ilość klastrów, ΔX_k to średnia odległość pomiędzy punktami w klastrze, $\delta(X_i, X_j)$ to odległość między środkami klastrów i i j. Na rysunkach poniżej przedstawiono, jak wpływ zastosowania standaryzacji lub normalizacji oraz zmiana ilości klastrów wpływa na wartość indeksu Daviesa-Bouldina dla klasteryzacji metodą aglomeracyjną (rys. 2.21) oraz metodą k-średnich (rys. 2.22).



Rys. 2.21. Przedstawienie wartości indeksu Daviesa-Bouldina dla danych nieprzetworzonych, danych standaryzowanych i danych znormalizowanych przy zastosowaniu metody aglomeracyjnej, opracowanie własne



Rys. 2.22. Przedstawienie wartości indeksu Daviesa-Bouldina dla danych nieprzetworzonych, danych standaryzowanych i danych znormalizowanych przy zastosowaniu metody k-średnich, opracowanie własne

Zarówno dla metody aglomeracyjnej, jak i k-średnich, najmniejsza liczba klastrów (2) daje najniższy indeks, wskazując na lepszą separację i spójność klastrów.

Część praktyczna

3. Część implementacyjna

3.1. Analiza problemu

Kluczowym elementem rozwoju oprogramowania są automatyczne testy regresyjne, które zapewniają, że dodanie nowych zmian do kodu nie prowadzi do awarii istniejących funkcjonalności. Przy ciągłej rozbudowie produktu o nowe moduły, ilość testów ulega zwiększeniu. Codzienne wykonywanie pełnej puli testów wymaga coraz większej ilości zasobów sprzętowych, co generuje znaczne koszty finansowe. Dodatkowo, długi czas wykonywania testów opóźnia wykrywanie błędów i wdrażanie poprawek do kodu. W celu usprawnienia testowania prowadzone są różne badania, w tym dotyczące wykrywania podobnych raportów błędów [21]. Narzędzia, wyznaczające optymalną kolejkę testów przy utrzymaniu wysokiego poziomu wykrywania błędów, bazujące na analizie danych i uczeniu maszynowym, mogłyby usprawnić ten proces.

3.2. Przebieg badania

Pracę rozpoczęto od pobrania i przygotowania zbiorów danych. Przy użyciu metod standaryzacji i normalizacji przetworzono wartości, opisujące kolejne wersje oprogramowania. Następnie przy użyciu metody progu wariancji wybrano różne zbiory cech. Z powstałych wersji danych utworzono zbiory danych trenujących, które zaprezentowano na wykresach używając metody analizy głównych składowych. Na danych trenujących zastosowano metody klasteryzacji: metodę aglomeracyjną oraz metodę k-średnich. Na podstawie przynależności wersji oprogramowania do klastrów, wyznaczono testy, stanowiące wynik algorytmu i zmierzono metryki, przedstawiające skuteczność zastosowanego podejścia.

3.3. Opis zbioru danych

Wyznaczenie optymalnej kolejki testów regresyjnych dla nowej wersji oprogramowania opiera się na przygotowaniu odpowiednich zbiorów danych. Dane pozyskano z wewnętrznej bazy wiedzy, przechowującej informacje dotyczące przebiegu testowania produktu. Dane, dotyczące wersji oprogramowania jak i testów, starsze niż osiemnaście miesięcy, uznano za przestarzałe, ponieważ mogą znacząco różnić się od danych obecnych poprzez wprowadzone zmiany w produkt. Ze względu na to, że oprogramowanie jest rozwijane w dwutygodniowych okresach zwanych sprintami, uruchamianie testów regresyjnych również następuje w tych okresach. Informację o dacie

rozpoczęcia i zakończenia każdego sprintu zapisano do tablicy *DataFrame*[22] w postaci przedstawionej na tabeli 3.1.

Tabela 3.1. Dane o sprintach, opracowanie własne

index	start	date	nazwa	evaluation
0	YYYY-MM-DD	YYYY-MM-DD	nazwa sprinta	False
1	YYYY-MM-DD	YYYY-MM-DD	nazwa sprinta	False
2	YYYY-MM-DD	YYYY-MM-DD	nazwa sprinta	False
3	YYYY-MM-DD	YYYY-MM-DD	nazwa sprinta	False
4	YYYY-MM-DD	YYYY-MM-DD	nazwa sprinta	False
…	…	…	…	…
n-1	YYYY-MM-DD	YYYY-MM-DD	nazwa sprinta	False
n	YYYY-MM-DD	YYYY-MM-DD	nazwa sprinta	False

W celu ewaluacji zastosowanych metod uczenia maszynowego, wyznaczono sprinty, do których należące wersje oprogramowania będą stanowiły zbiór testowy. Wybrano wszystkie sprinty z ostatnich sześciu miesięcy od września do lutego 2024 i oznaczono je wartością *True* w dodatkowej kolumnie o nazwie *evaluation*.

Proces rozwoju i testowania oprogramowania jest ciągły, z wielu wersji kodu codziennie zostaje wybrany kandydat, na którym zostaje uruchomiona kolejka testów regresyjnych. Dla odpowiednich sprintów, pobrano listę kandydatów, które należą do okresu ostatnich osiemnastu miesięcy. Na podstawie przynależności do sprintu, oznaczono kandydatów flagami *False* oraz *True*, co odpowiada przynależności do zbioru treningowego i testowego odpowiednio i zapisano tę informacje do kolumny *evaluation* w tablicy *DataFrame*, przedstawiającej kandydatów (tabela 3.2):

Tabela 3.2. Przypisane wersji oprogramowania (kandydatów) do sprintów, opracowanie własne

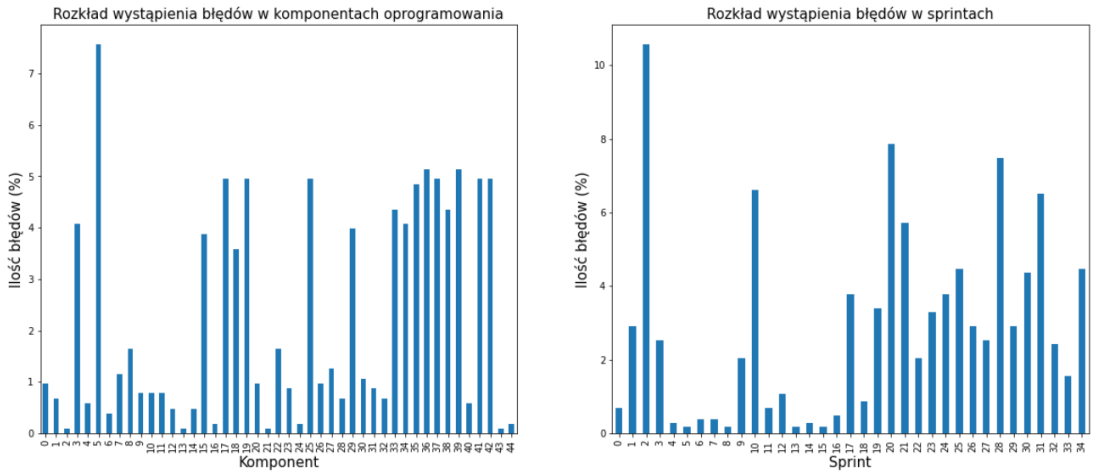
index	wersja	sprint	evaluation
0	wersja kandydata	nazwa sprinta	False
1	wersja kandydata	nazwa sprinta	False
2	wersja kandydata	nazwa sprinta	False
3	wersja kandydata	nazwa sprinta	False
4	wersja kandydata	nazwa sprinta	False
…	…	…	…
n-1	wersja kandydata	nazwa sprinta	False
n	wersja kandydata	nazwa sprinta	False

Z bazy wiedzy pobrano informacje o puli dostępnych testów regresyjnych, wraz z ich parametrami. Każdy test ma przypisaną listę błędów, które zostały wykryte przez ten test w rozpatrywanym okresie czasu. Dodatkowo dane zawierają informacje o dacie utworzenia testu, dacie jego pierwszego wykonania oraz dacie ostatniej modyfikacji tego testu. Nie wszystkie testy regresyjne są uruchamiane automatycznie na wybranym kandydacie. W ramach badania użyto tylko w pełni zautomatyzowanych testów. Listę parametrów, przedstawiających pojedynczy test przedstawiono w tabeli 3.1.

Tabela 3.3. Pola zbioru danych testów regresyjnych, opracowanie własne

test id	int
domena	str
projekt	str
nazwa	str
organizacja	str
data utworzenia	timestamp
data pierwszego wykonania	timestamp
data ostatniej modyfikacji	timestamp
opis funkcjonalności	str
domena testowa	str
domena funkcjonalna	str
lista wykrytych błędów	list(str)
zautomatyzowany	boolean

Dla wersji oprogramowania, będących kandydatami, pobrano dane o uruchomianiach testów regresyjnych, zawierające informację o tym, jakiego kandydata oraz testu dotyczy uruchomienie, jaki jest wynik uruchomienia, kiedy test został uruchomiony oraz kiedy się skończył. Dane te zawierają również informacje o błędach, które zostały wykryte w ramach danego uruchomienia. Ze względu na dużą ilość takich informacji, sięgającej setek tysięcy, wybrano unikatowe uruchomienia pod względem identyfikatora testu, wyniku, numeru kandydata oraz czasu wykonywania testu. Wybrano informacje o uruchomieniach, dotyczące tylko automatycznie wykonywanych testów.
Raporty wykrytych błędów jest ostatnim zbiorem danych, dotyczącym procesu testowania, użytym w badaniu. Zawierają one informację, w jakim komponencie i w jakiej wersji oprogramowania błąd został wykryty, jaki posiadał priorytet i w jaki sposób został rozwiązany. Rozkład występowania błędów w komponentach oprogramowania i sprintach przedstawiono na rys. 3.3:



Rys. 3.1. Rozkład występowania błędów w komponentach oprogramowania i sprintach, opracowanie własne

Zbiór danych, przedstawiający kolejnych kandydatów, zbudowano na podstawie wewnętrznego narzędzia, monitorującego zmiany w kodzie produkcyjnym. Umożliwiło to wyznaczenie dla każdego kolejnego kandydata, ile linijek kodu zostało dodanych lub usuniętych w każdym typie plików oraz w każdym typie plików dla każdego komponentu. Do zbioru danych dodano kolumny, pokazujące czy dany komponent lub typ plików w tej wersji produktu był dodany (flaga „ADDED"), zmieniony (flaga „CHANGED"), był usunięty (flaga „REMOVED") lub nie uległ zmianie (flaga „NOT CHANGED"). Natomiast flaga „-1" wskazuje, że dany element nie istniał w obecnie rozważanym kandydacie. Strukturę zbioru na przykładzie dwóch komponentów i dwóch typów plików przedstawiono na rys. 3.4.

index	wersja	komponent_1	komponent_2	.cpp	.py	komponent_1_cpp	komponent_1_py	komponent_2_cpp	komponent_2_py
0	candidate name	CHANGED	-1	150	50	150	50	0	0
1	candidate name	CHANGED	ADDED	200	-120	0	-300	200	180
2	candidate name	REMOVED	NOT CHANGED	-300	-200	-300	-200	0	0
3	candidate name	-1	CHANGED	-100	200	0	0	-100	200

Rys. 3.2. Schemat budowy zbioru danych, przedstawiający zmiany w kodzie kandydatów, opracowanie własne

3.4. Przygotowanie uczącego zbioru danych

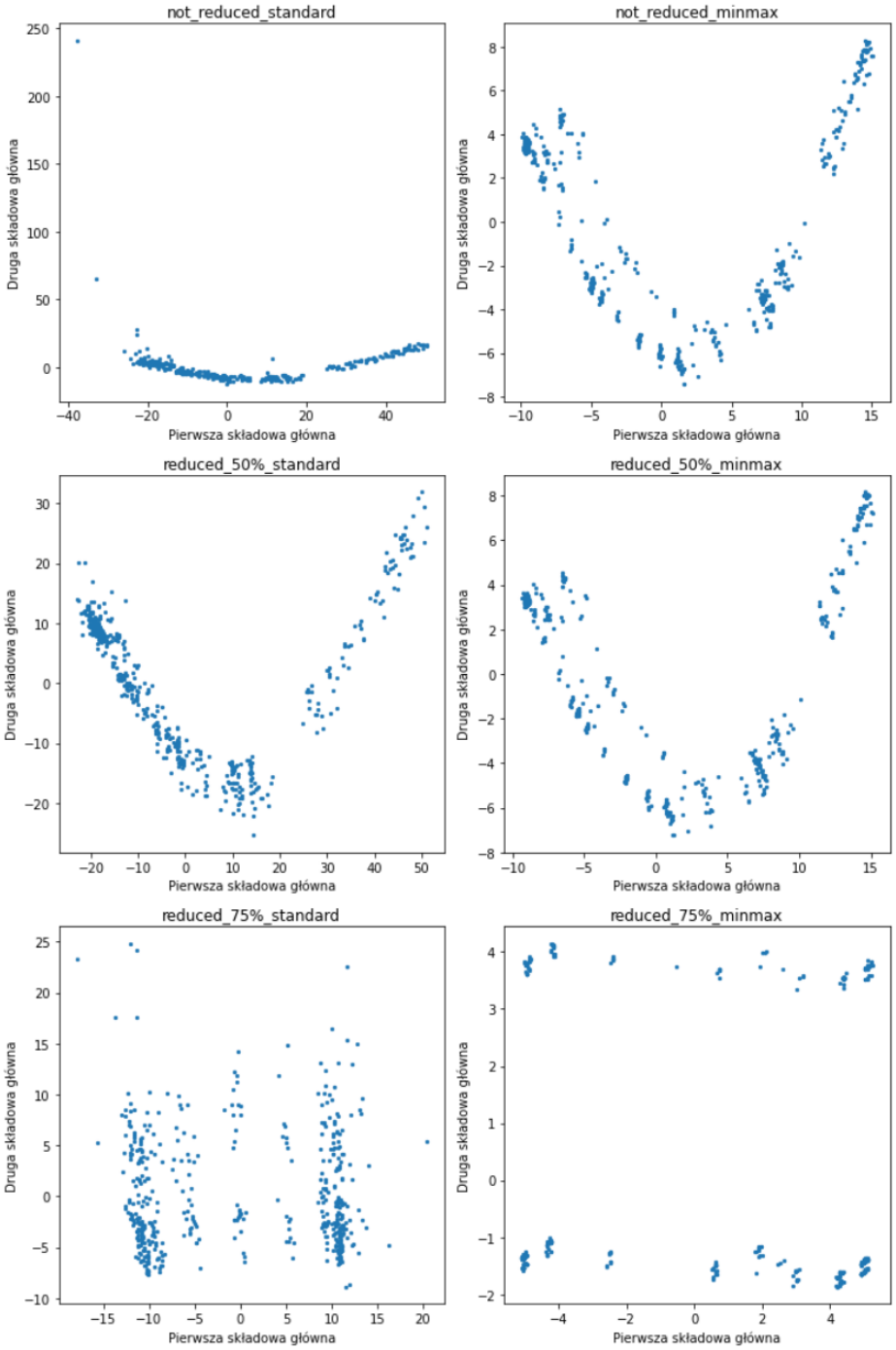
Dla każdego kandydata należącego do zbioru testowego, każdy wcześniejszy kandydat stanowi element zbioru treningowego. Zbiór ten wykorzystano zarówno do przygotowania danych za pomocą metod standaryzacji lub normalizacji oraz progu wariancji, jak i do trenowania modeli klasteryzacyjnych. W ten sposób, bazując na danych o zmianach w kandydatach, utworzono różne wersje zbiorów treningowych na

podstawie wybranych percentyli dla metody VarianceThreshold: 0% - użyto wszystkich cech, 50% - połowa cech o najwyższej wariancji, 75% - jedna czwarta cech o najwyższej wariancji. Zastosowano również dwa algorytmy skalowania i normalizacji: StandardScaler oraz MinMaxScaler. Kombinacje parametrów przedstawiono na rys. 3.5. W ten sposób stworzono sześć zbiorów danych: nieprzetworzone dane poddane standaryzacji i posiadające wszystkie cechy oryginalnego zbioru danych, 50% cech i 75% cech oraz nieprzetworzone dane poddane normalizacji i posiadające wszystkie cechy, 50% cech i 70% cech.

Preprocessing / VarianceThreshold	StandardScaler	MinMaxScaler
0%	not_reduced_standard	not_reduced_minmax
50%	reduced_50%_standard	reduced_50%_minmax
75%	reduced_75%_standard	reduced_75%_minmax

Rys. 3.3. Kombinacje parametrów redukcji cech i metod przetwarzania wartości cech, opracowanie własne

W celu wizualizacji powstałych zbiorów danych treningowych użyto algorytmu analizy głównych składowych. Oś pozioma reprezentuje pierwszą główną składową, a oś pionowa drugą główną składową. Każdy punkt na wykresach odpowiada jednemu kandydatowi, a różne kombinacje skalowania lub normalizacji i parametrów progu wariancji są przedstawione na poszczególnych wykresach na rys. 3.6.



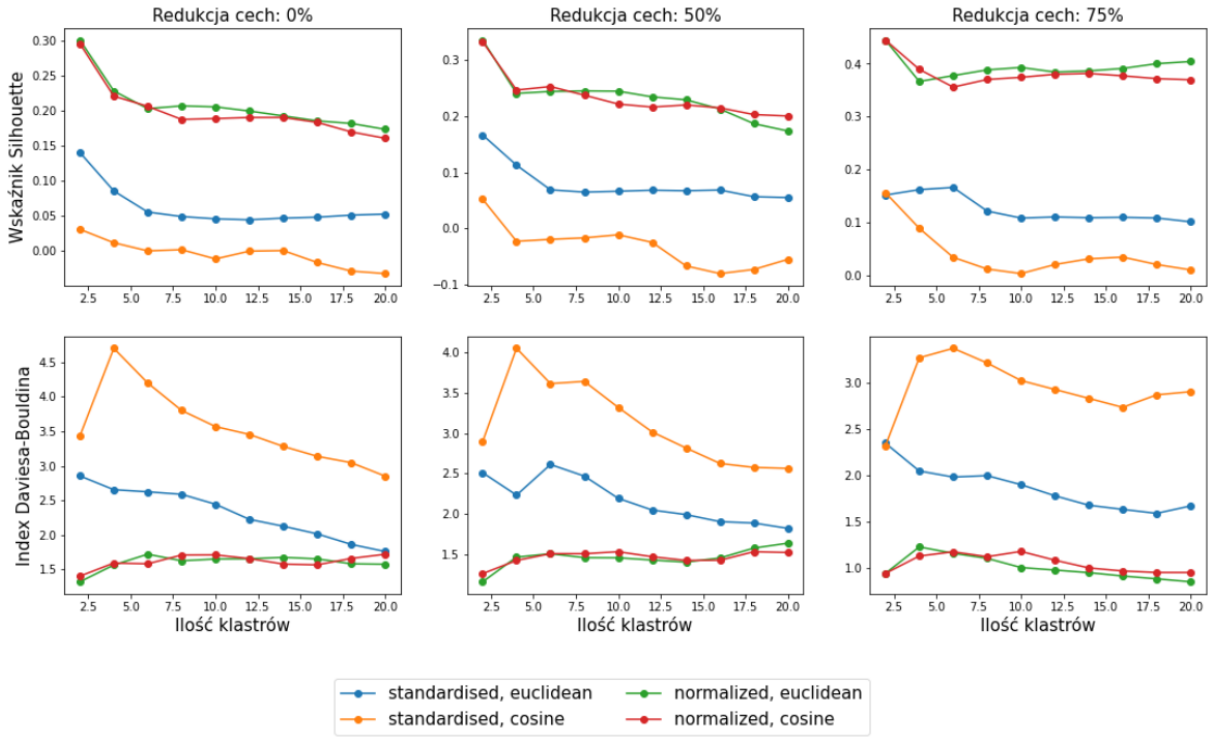
Rys. 3.4. Reprezentacja zbiorów danych treningowych przy użyciu metody PCA. Pierwsza kolumna przedstawia zbiory danych standaryzowanych, druga – zbiory danych znormalizowanych. Wiersze odpowiadają zastosowanej redukcji cech: w pierwszym wierszu przedstawiono zbiór, zawierający wszystkie cechy, w drugim – zbiór, składający się z 50% cech, trzeci – 75% cech, opracowanie własne

3.5. Klasteryzacja i walidacja

W celu odzwierciedlenia procesu biznesowego, gdzie nowy kandydat pojawia się każdego dnia i kolejka wszystkich automatycznych testów regresyjnych jest na nim uruchamiana, dla każdego kandydata, należącego do zbioru testowego, wyznaczono zbiór treningowy, jako wszystkie poprzednie wersje kodu. Pozwala to na przeprowadzenie

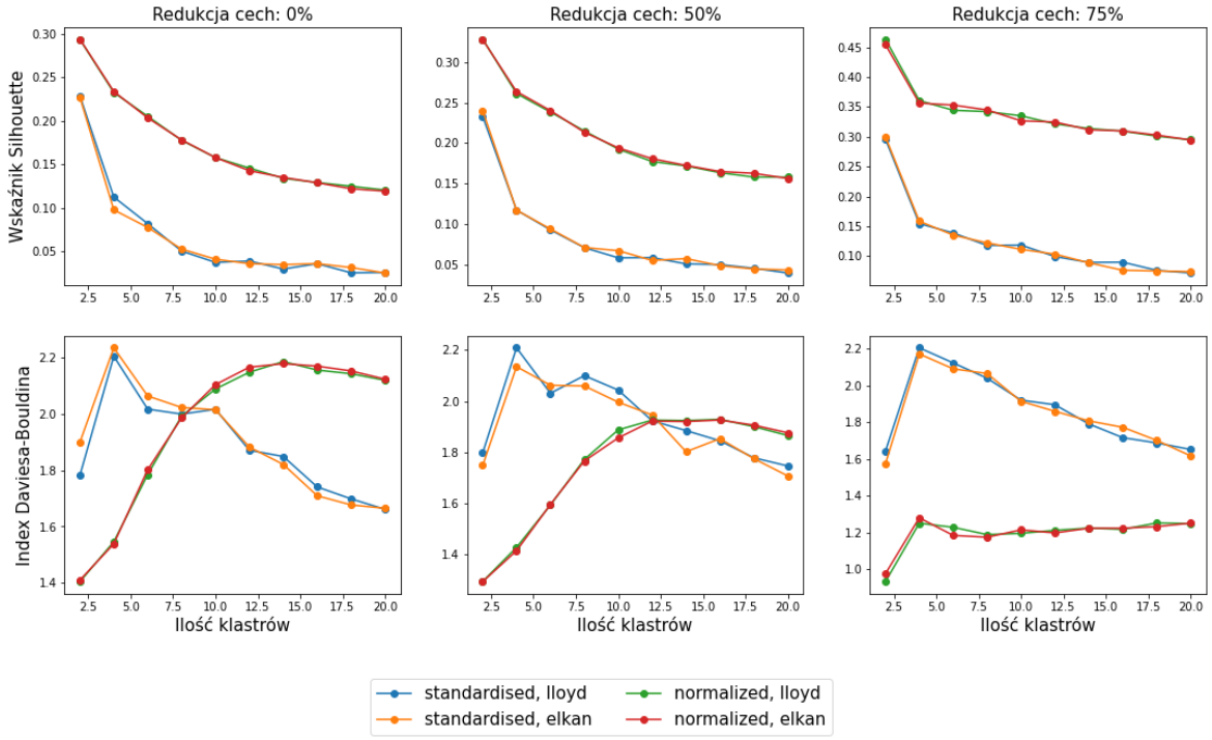
procesu klasteryzacji i predykcji tylko na danych, będących historycznymi dla danego kandydata.

Klasteryzacje wykonano metodą aglomeracyjną z metryką euklidesową i kosinusową oraz metodą k-średnich z algorytmami, opisanymi przez Stuarta Lloyda [23] i Charlesa Elkana [24]. Zakres ilości klastrów ustalono od dwóch do dwudziestu z krokiem dwa dla każdej z metod klasteryzacji. Górna granica ilości klastrów ustalono, jako przybliżoną wartość pierwiastka kwadratowego z liczby próbek w zbiorze danych. Na rys. 3.7 przedstawiono wyniki klasteryzacji metodą aglomeracyjną. Wykresy umieszczono w tablicy, gdzie pierwszy wiersz odpowiada wskaźnikowi Silhouette, drugi – indeksowi Daviesa-Bouldina. W pierwszej kolumnie przedstawiono wyniki dla zbioru danych, w którym ilość cech nie została zmniejszona, w drugiej – ilość cech w zbiorze danych ograniczono do 50% o najwyższej wartości wariancji, w trzeciej – redukcja cech wyniosła 75%. Rysunek przedstawia, że najlepsze wyniki osiągnięto przy użyciu odległości euklidesowej dla zbioru danych, składającego się z 25% cech o najwyższej wariancji, wartości których zostały znormalizowane. Przy klasteryzacji do dwóch i dwudziestu klastrów, wartość wskaźnika Silhouette jest najwyższa, natomiast wartość indeksu Daviesa-Bouldina jest najniższa.



Rys. 3.5. Wartość wskaźnika Silhouette uzyskana przy klasteryzacji metodą aglomeracyjną z użyciem metryki euklidesowej oraz metryki kosinusowej dla zbiorów danych standaryzowanych i zbiorów znormalizowanych, opracowanie własne

Wyniki klasteryzacji metodą k-średnich umieszczono w tablicy wykresów o tej samej strukturze i przedstawiono na rys. 3.8. W tym przypadku najlepsze wyniki osiągnięto dla zbioru danych znormalizowanych, składającego się z 25% cech z zastosowaniem algorytmu Charlesa Elkana. Klasteryzacja do dwóch i sześciu klastrów zwróciła najlepsze wyniki dla wskaźnika Silhouette i indeksu Daviesa-Bouldina.



Rys. 3.6. Wartość wskaźnika Silhouette uzyskana przy klasteryzacji metodą k-średnich z użyciem algorytmu Lloyda i algorytmu Elkana dla zbiorów danych standaryzowanych i zbiorów znormalizowanych, opracowanie własne

3.6. Wyznaczenie kolejki testów

W celu przechowywania predykcji testów dla kandydatów i ułatwienia oceny skuteczności podejścia, dla każdego kandydata, należącego do zbioru treningowego, wyznaczono dane historyczne o wszystkich testach, które były na nim uruchomione. Jak przedstawiono w tabeli 3.2, dane zawierają identyfikator ID instancji testu, flagę, czy został wykryty nowy błąd, listę błędów, wykrytych przy uruchomieniu testu oraz flagę *y_true* wskazującą, że wykonanie testu zakończyło się niepowodzeniem. Może zachodzić sytuacja, gdy wykonanie testu zakończyło się niepowodzeniem bez wykrycia błędu w kodzie. Powodem, między innymi, mogą być błędy środowiskowe albo ukryte błędy w skrypcie testu. Do każdej instancji testu dodano zmienną *y_pred*, która oznacza, że test został przewidziany do uruchomienia na danym kandydacie poprzez algorytm wyznaczania kolejki testów. Na początku zmiennej *y_pred* przypisano wartość „False”.

Tabela 3.4. Pola danych historycznych o przebiegu testów regresyjnych, opracowanie własne

index	ID instancji	czy wykryto błąd	lista ID wykrytych błędów	y_true	y_pred
0	1034	True	[6073]	True	False
1	2335	False	[]	False	False
2	3901	False	[]	True	False

Wyznaczenie kolejki testów dla kandydatów, należących do zbioru testowego, polega na wybraniu poprzednich wersji oprogramowania, na podstawie ich przynależności do wyznaczonych klastrów. Następnie zostają wybranie testy regresyjne, które zakończyły się niepowodzeniem z wykryciem błędu dla tych wersji oprogramowania. Testy, które znalazły się w danych historycznych kandydata, w kolumnie *y_pred* są oznaczane wartością „True”, natomiast nowe instancje testów są dodawane. Istotne jest ustawienie *y_true* dla takich testów na wartość „False”, co oznacza, że nie było ich wśród historycznych testów, uruchomionych na danym kandydacie. Algorytm zastosowano na zbiorze danych znormalizowanych, w którym zostawiono 25% cech o najwyższej wariancji. Dla klasteryzacji metodą aglomeracyjną, jako metryka odległości została wybrana odległość euklidesowa. Dla klasteryzacji metodą k-średnich wybrano algorytm Charlesa Elkana.

3.7. Ocena predykcji

Dla oceny skuteczności zastosowanego algorytmu i metod klasteryzacji użyto dwóch metryk. Pierwsza metryka, którą nazwano „dokładnością” zlicza, jaki procent historycznie wykrytych błędów zostaje wykryty przez wyznaczone testy dla danego kandydata. Druga metryka o nazwie „zysk” przedstawia, jak bardzo skrócona zostaje kolejka historycznie uruchomionych na kandydacie testów względem oryginalnej kolejki testów. Dla każdej ilości klastrów, od dwóch do dwudziestu z krokiem dwa, wyznaczono średnią „dokładność”, obliczaną zgodnie ze wzorem (3.1) oraz „zysk”, obliczany zgodnie ze wzorem (3.2).

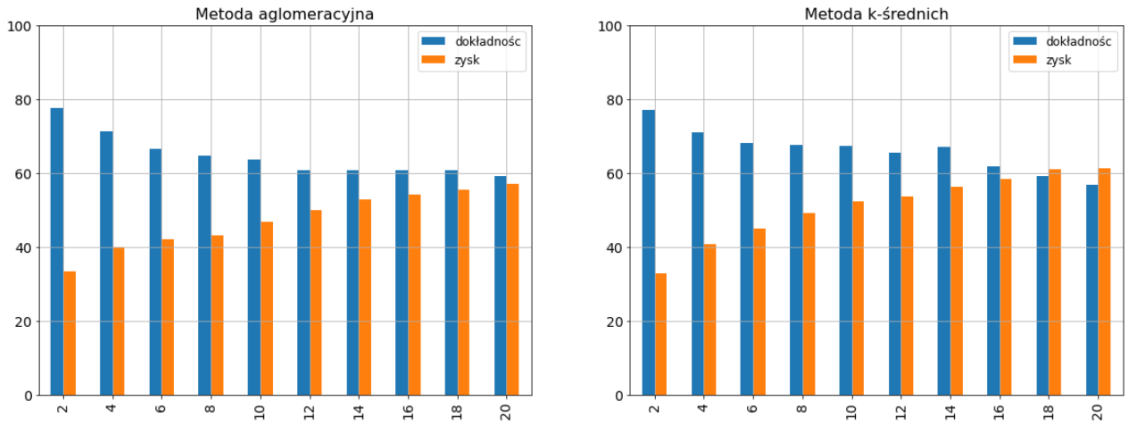
$$A = \frac{1}{n}\sum_{i=0}^{n}\frac{pred_i}{true_i} * 100\%, \tag{3.1}$$

gdzie A – średnia dokładność klasteryzacji, n – ilość kandydatów w zbiorze testowym, i – numer kandydata, $pred_i$ – ilość błędów, które zostały wykryte w kodzie kandydata i przez wyznaczone testy, $true_i$ – ilość wykrytych błędów w kodzie kandydata i przez oryginalną kolejkę testów.

$$P = \frac{1}{n}\sum_{i=0}^{n} 1 - \frac{t\_pred_i}{t\_true_i} * 100\%, \tag{2.1}$$

gdzie P – średni zysk klasteryzacji, n – ilość kandydatów w zbiorze testowym, i – numer kandydata, t_$pred_i$ – ilość wyznaczonych testów dla kandydata i, t_$true_i$ – ilość historycznych testów dla kandydata i.

Wykresy na rys. 3.9 przedstawia zmianę wartości obydwu metryk dla różnej ilości klastrów dla metody aglomeracyjnej oraz k-średnich.



Rys. 3.7. Zmiana wartości metryk dokładności i zysku przy zastosowaniu metody aglomeracyjnej i k-średnich dla różnych ilości klastrów, opracowanie własne

4. Podsumowanie

Cel pracy, którym było zbadanie skuteczności metod procesowania i klasteryzacji danych historycznych o oprogramowaniu w celu wyznaczania optymalnej kolejki testów regresyjnych, został osiągnięty. Uzyskane wyniki demonstrują, że przy ustalonej maksymalnej ilości klastrów udało się osiągnąć zmniejszenie kolejki testów na poziomie 60%. Zastosowanie algorytmu może prowadzić do zmniejszenia obciążenia zasobów sprzętowych, jak i kosztów poniesionych przez firmę. Wiąże się to ze znacznym spadkiem wykrywalności błędów, poniżej 80%, nawet przy najmniejszej wartości metryki „zysku" na poziomie 30%. W przypadku produktów, dotyczących infrastruktury telekomunikacyjnej, niewykryte błędy mogą prowadzić do znacznych strat reputacyjnych i finansowych. Zastosowaniem opisanego algorytmu może być nadanie wyższego priorytetu testom, które zostały zwrócone dla kandydata. Nie zmniejszy to ilości codziennie wykonywanych testów, natomiast może przyśpieszyć wykrywanie błędów i korzystnie wpłynąć na czas ich rozwiązywania.

Bibliografia

[1] Aleksandra Król-Nowak, Katarzyna Kotarba, *Podstawy uczenia maszynowego,* Wydawnictwa AGH, 2022

[2] Canchen Li, Preprocessing Methods and Pipelines of Data Mining: An Overview, https://arxiv.org/pdf/1906.08510, 2019

[3] Sala, Roberto & Zambetti, Michela & Pirola, Fabiana & Pinto, Roberto, How to select the right machine learning algorithm: a feature-based, scope-oriented selection framework, 2018

[4] Statystyka w uczeniu maszynowym, https://www.sztucznainteligencja.org.pl/statystyka-w-uczeniu-maszynowym/

[5] Maciej Ziaja, Przekrojowe wprowadzenie do uczenia maszynowego, https://nowy.kmim.wm.pwr.edu.pl/pl/event/2024/03/przekrojowe-wprowadzenie-do-uczenia-maszynowego/przekrojowe-wprowadzenie-do-uczenia-maszynowego.pdf

[6] Titanic – Machine Learning from Disaster, https://www.kaggle.com/competitions/titanic (data dostępu 24.05.2024)

[7] Jurand Bień, Uczenie maszynowe: przygotowanie danych – wprowadzenie, https://ekordo.pl/uczenie-maszynowe-przygotowanie-danych-wprowadzenie/ (dostęp 24.05.2024)

[8] Jan Pieńkos, Janusz Turczyński, Układy scalone TTL w systemach cyfrowych, Warszawa: Wydawnictwa Komunikacji i Łączności, 1980, s. 53-55

[9] Miroslav Kubat , An Introduction To Machine Learning, Springer International Publishing AG 2015, 2017

[10] Wariancja, Encyklopedia PWN, https://encyklopedia.pwn.pl/haslo/;3993975 (data dostępu 25.05.2024)

[11] Removing features with low variance, https://scikit-learn.org/stable/modules/feature_selection.html (data dostępu 26.05.2024)

[12] Analiza składowych głównych, https://home.agh.edu.pl/~mmd/_media/dydaktyka/adp/analiza_skladowych_glowych.pdf (data dostępu 27.05.2024)

[13] Erwin Otto Kreyszig, Advanced Engineering Mathematics, Wiley, 10 edycja (strona 1037)

[14] Jure Leskovec, Anand Rajaraman, Jeff Ullman, Mining of Massive Datasets, Cambridge University Press 2020

[15] Rokach, Lior, and Oded Maimon, "Clustering methods. Data mining and knowledge discovery handbook", Springer, 2005

[16] Mahamad Omran, Andries Engelbrecht, Ayed A. Salman, An overview of clustering methods, https://www.researchgate.net/publication/220571682_An_overview_of_clustering_methods

[17] Encyclopedia of Mathematics, Centroid, https://encyclopediaofmath.org/wiki/Centroid (data dostępu 03.06.2024)

[18] Peter J. Rousseeuw, Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis, 1987

[19] Mícheál Ó Searcóid, Metric Spaces, Springer, 2006

[20] Davies, David L.; Bouldin, Donald W., A Cluster Separation Measure, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1979

[21] Sebastian Zarębski, Aleksander Kuzmich, Sebastian Sitko, Krzysztof Rusek, Piotr Chołda, Siamese Neural Networks on the Trail of Similarity in Bugs in 5G Mobile Network Base Stations, 2022

[22] DataFrame, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html (data dostępu 05.06.2024)

[23] Stuart P. Lloyd, Least squares quantization in PCM, IEEE Transactions on Information Theory,1982

[24] Charles Elkan, Using the triangle inequality to accelerate k-means, 2003

SPIS RYSUNKÓW

SPIS TABEL