POLITECHNIKA RZESZOWSKA
im. Ignacego Łukasiewicza
Wydział Elektrotechniki i Informatyki

ROZPRAWA DOKTORSKA

mgr inż. Sylwester Czmil

Algorytmy nadzorowanego inkrementalnego uczenia maszynowego oraz ocena jakości ich klasyfikacji

Promotor
prof. dr hab. inż. Jacek Kluska

Rzeszów 2024

Podziękowania

Składam serdeczne podziękowania mojemu promotorowi, prof. dr. hab. inż. Jackowi Klusce, za nieocenione wsparcie, cenne wskazówki merytoryczne oraz czas poświęcony na konsultacje podczas całego procesu powstawania rozprawy doktorskiej. Profesjonalizm, bogate doświadczenie i pełne zaangażowanie Pana Profesora stanowiły dla mnie źródło inspiracji i motywacji do nieustannego rozwoju, a Jego postawa i etyka pracy będą mi towarzyszyć jako cenny drogowskaz w mojej dalszej karierze.

Pragnę również wyrazić wdzięczność mojej ukochanej Żonie, która była dla mnie nieustającym wsparciem w trakcie pisania rozprawy doktorskiej. Dziękuję za pomoc i motywację w chwilach zwątpienia oraz za naszą wspólną drogę w dążeniu do rozwoju naukowego.

Spis treści

Wykaz symboli, oznaczeń i skrótów

AB – wzmocnienie adaptacyjne (AdaBoost)

ACC – dokładność klasyfikacji (accuracy)

AEE – zespół addytywnych ekspertów (additive expert ensemble)

ARF – adaptacyjny las losowy (adaptive random forest)

ART – teoria rezonansu adaptacyjnego (adaptive resonance theory)

ARTMAP – mapa teorii rezonansu adaptacyjnego (adaptive resonance theory map)

CACP – narzędzie służące do oceny jakości klasyfikacji (Classification Algorithms Comparison Pipeline)

CNN – sieci splotowe (convolutional neural network)

DOB-SCV – stratyfikowany sprawdzian krzyżowy optymalnie zrównoważony pod względem dystrybucji (distribution optimally balanced SCV)

DT – drzewo decyzyjne (decision tree)

DWM – algorytm dynamicznej większości ważonej (dynamic weighted majority)

EFDT – szybkie drzewo decyzyjne (extremely fast decision tree)

EVQ – ewoluująca kwantyzacja wektorowa (evolving vector quantization)

F1 – ważona miara F1 (F1 score)

F-beta – ważona miara F-beta (F-beta score)

FAM – procedura oparta na logice rozmytej i teorii rezonansu adaptacyjnego (fuzzy ARTMAP)

FF – przerzutnik typu D (flip-flop)

GNB – Gaussowski naiwny klasyfikator Bayesa (Gaussian naive Bayes)

HT – drzewo Hoeffdinga (Hoeffding tree)

HAT – drzewo adaptacyjne Hoeffdinga (Hoeffding adaptive tree)

HLS – synteza wysokiego poziomu (high-level synthesis)

KNN – klasyfikator k-najbliższych sąsiadów (k-nearest neighbours)

KNNI – inkrementalna wersja klasyfikatora k-najbliższych sąsiadów (k-nearest neighbours)

LR – regresja logistyczna (logistic regression)

LUT – tablica do wyszukiwań (look-up table)

LUTRAM – look-up table RAM

LVQ – algorytm uczącej się kwantyzacji wektorowej (learning vector quantization)

MLP – perceptron wielowarstwowy (multilayer perceptron)

NB – naiwny klasyfikator Bayesa (naive bayes)

NC – najbliższy centroid (nearest centroid)

NLP – przetwarzanie języka naturalnego (natural language processing)

OB – Oza Bagging

PRE – precyzja ważona (weighted precision)

PYNQ – Python dla Zynq (Python productivity for Zynq)

QDA – kwadratowa analiza dyskryminacyjna (quadratic discriminant analysis)

RF – las losowy (random forest)

ROC – charakterystyka operacyjna odbiornika (receiver operating characteristic)

RTL – poziom transferu rejestru (register transfer level)

SCV – stratyfikowany sprawdzian krzyżowy (standard stratified cross-validation

SEN – czułość ważona (weighted sensitivity, recall)

SEVQ – prosty ewoluujący algorytm kwantyzacji wektorowej (simple evolving vector quantization algorithm)

SFAM – uproszczona procedura oparta na logice rozmytej i teorii rezonansu adaptacyjnego (simplified fuzzy ARTMAP)

SVC – klasyfikacja przy użyciu wektorów wspierających (c-support vector classification)

SVM – maszyna wektorów wspierających (support vector machines)

Tcl – natywny język programowania (tool command language)

VQ – kwantyzacja wektorowa (vector quantization)

XAI – objaśnialna sztuczna inteligencja (explainable artificial intelligence)

XGB – ekstremalne wzmocnienie gradientowe (extreme gradient boosting, XGBoost)

1. Wprowadzenie

W ostatnich latach zyskały na popularności algorytmy głębokie, tj. modele oparte na sztucznych sieciach neuronowych, składające się z dużej liczby neuronów i wielu warstw ukrytych, jak np. sieci splotowe (CNN), rekurencyjne (RNN) czy sieci typu transformer. Jednak algorytmy tradycyjne (płytkie) wciąż są interesujące w wielu zastosowaniach, ponieważ zwykle mają znacznie mniejsze zapotrzebowanie na zasoby obliczeniowe i pamięć niż algorytmy głębokie. Tam, gdzie dostępne zasoby są ograniczone, algorytmy tradycyjne mogą być bardziej praktyczne i wydajne. Ponadto, są one często bardziej transparentne (interpretowalne) niż algorytmy głębokie, co ma duże znaczenie w takich dziedzinach, jak medycyna czy prawo.

Oczywiście wybór między modelem płytkim a głębokim zależy od specyfiki rozwiązywanego problemu i dostępnych zasobów. W dużym uproszczeniu można stwierdzić, że modele głębokie są potężniejsze, ale trudniejsze do zrozumienia. Modele płytkie są z kolei bardziej zrozumiałe (interpretowalne) niż modele głębokie, co jest kluczowe w kontekście objaśnialnej sztucznej inteligencji (XAI). Algorytmy głębokie znajdują zastosowanie w zadaniach, w których kluczowe jest rozpoznawanie wzorców na dużych i złożonych zbiorach danych, np. w analizie obrazów, przetwarzaniu języka naturalnego (NLP) czy automatycznej diagnostyce medycznej. Spore zainteresowanie budzą także metody hybrydowe, które łączą zalety obu podejść — płytkich i głębokich [1].

Niniejsza rozprawa dotyczy problematyki nadzorowanego uczenia klasyfikatorów płytkich na podstawie danych uczących oraz oceny jakości ich klasyfikacji. Klasyfikacja danych jest jednym z najważniejszych i najczęściej występujących zadań w uczeniu maszynowym. Zadaniem algorytmu klasyfikacji danych jest przewidywanie klasy dowolnie wybranych rekordów danych, zwłaszcza tych, które nigdy wcześniej nie były używane w procesie uczenia klasyfikatora. Istnieje wiele nadzorowanych algorytmów klasyfikacyjnych, a wśród nich można wyróżnić algorytmy tradycyjne (nieprzyrostowe) oraz algorytmy inkrementalne (przyrostowe). Główną różnicą między tymi grupami algorytmów jest to, że uczenie inkrementalne pozwala na adaptację modelu do nowych danych w trakcie ich napływu, podczas gdy w tradycyjnym uczeniu maszynowym model jest uczony (trenowany) na całym dostępnym zbiorze danych jednorazowo.

Do najpopularniejszych algorytmów uczenia nadzorowanego rozwiązujących problem klasyfikacji danych należą takie algorytmy, jak las losowy (random forest, RF)

oraz maszyna wektorów wspierających (support vector machines, SVM). Obydwa algorytmy bardzo dobrze sprawdzają się w różnych zastosowaniach i są uważane za bardzo dokładne i niezawodne [2]. Działanie SVM opiera się na poszukiwaniu optymalnej hiperpłaszczyzny, która najlepiej separuje dane należące do różnych klas [3]. Algorytm SVM sprowadza się do rozwiązania zadania minimalizacji funkcji kwadratowej z ograniczeniami. RF to metoda klasyfikacji danych, która konstruuje wiele drzew decyzyjnych na podstawie losowych podzbiorów danych treningowych i losowego doboru cech. Następnie wynik klasyfikacji jest determinowany przez głosowanie lub uśrednienie wyników wszystkich drzew w lesie, co prowadzi do uzyskania modelu o wysokiej dokładności i odporności na nadmierne dopasowanie [4].

Inną metodą klasyfikacji jest algorytm k-najbliższych sąsiadów (k-nearest neighbours, KNN), który dla danego punktu testowego (czyli rekordu danych albo próbki) znajduje k najbliższych punktów w przestrzeni cech na podstawie określonej miary odległości (np. euklidesowej), a następnie przypisuje mu klasę, będącą najczęściej spotykaną klasą wśród k najbliższych punktów [5]. W ostatnich latach dużą popularność zyskał też zaproponowany w 2016 roku algorytm XGBoost (XGB) [6]. Dzięki efektywności i wydajności stał się on algorytmem często wybieranym przez zwycięzców wielu konkursów z dziedziny nauki o danych i uczenia maszynowego, organizowanych głównie na platformie Kaggle [7]. Innymi tradycyjnymi klasyfikatorami uczenia maszynowego stosowanymi w wielu zadaniach klasyfikacyjnych są: naiwny klasyfikator Bayesa (naive bayes, NB), drzewo decyzyjne (decision tree, DT), wzmocnienie adaptacyjne (AdaBoost, AB) czy regresja logistyczna (logistic regression, LR). Każdy z wymienionych algorytmów posiada mocne i słabe strony, a ich skuteczność często zależy od specyfiki zbioru danych oraz charakterystyki problemu. Wybór odpowiedniego algorytmu powinien uwzględniać dostępną ilość danych, liczbę cech, zapotrzebowanie na interpretowalność modelu, a także zasoby obliczeniowe. Często dobrym rozwiązaniem jest testowanie kilku algorytmów i porównanie ich wydajności za pomocą odpowiednich metryk, takich jak dokładność (accuracy, ACC), precyzja (precision, PRE), czułość (sensitivity, SEN) czy pole pod krzywą ROC (area under receiver operating characteristic curve, AUROC, AUC).

Głównym ograniczeniem klasycznych klasyfikatorów jest brak mechanizmu pozwalającego na ciągłe integrowanie nowych danych z już istniejącym modelem [8]. Ten problem nabiera szczególnego znaczenia w kontekście ciągłego wzrostu przetwarzania

dużych i dynamicznie zmieniających się danych. Zwiększanie rozmiaru zbioru danych prowadzi do wydłużenia czasu potrzebnego na ponowne przetrenowanie modelu. Co więcej, po pojawieniu się nowej klasy model nie jest w stanie jej automatycznie uwzględnić bez konieczności ponownego przetrenowywania całego algorytmu.

W związku z powyższym badacze zainteresowali się opracowaniem technik umożliwiających aktualizację modelu klasyfikacji wraz z napływem nowych danych, zamiast rozpoczynania procesu uczenia od początku. Doprowadziło to do pojawienia się klasyfikatorów inkrementalnych, które potrafią uczyć się nowych informacji z nowych danych, zachowując jednocześnie wcześniej nabytą wiedzę, unikając katastroficznego zapominania, identyfikując i aktualizując nowe klasy oraz nie wymagając dostępu do oryginalnych danych wykorzystanych do pierwotnego uczenia istniejącego klasyfikatora [9].

Algorytmy przyrostowe znajdują się w centrum uwagi niniejszej pracy. Cieszą się one dużą popularnością, zwłaszcza w kontekście dużych, zmiennych i różnorodnych zbiorów danych (big data) i uczenia się na strumieniach danych, gdzie dane napływają w czasie rzeczywistym i/lub w dużych ilościach, wymagając częstych aktualizacji modeli lub analiz. W takich przypadkach często zdarza się, że przechowywanie i przetwarzanie całego zbioru danych jednocześnie jest niepraktyczne, a czasem niemożliwe z powodu dużej złożoności obliczeniowej i pamięciowej [10]. Algorytmy te pozwalają na dynamiczne dostosowanie się do zmiennych warunków, bez konieczności ponownego przetwarzania całego zbioru danych. Sprawia to, że są one bardziej efektywne od tradycyjnych algorytmów klasyfikacyjnych w wymienionych wcześniej przypadkach. Ponadto, algorytmy inkrementalne umożliwiają tworzenie skalowalnych systemów uczenia maszynowego, co jest szczególnie istotne w implementacji sprzętowej tego typu modeli, np. w układzie FPGA. Wynika to z faktu, że zazwyczaj są one łatwiejsze do zrównoleglenia w porównaniu z tradycyjnymi algorytmami, gdyż proces aktualizacji modelu może być rozproszony na wiele jednostek obliczeniowych lub rdzeni procesora. Dodatkowo są bardziej efektywne z punktu widzenia zużycia energii i wymagań obliczeniowych. Algorytmy inkrementalne znajdują zastosowanie w analizie sieci, przewidywaniu danych finansowych, kontroli ruchu czy przetwarzaniu pomiarów z czujników [10].

W literaturze zaproponowano szereg algorytmów inkrementalnych [8,11]. Z uwagi na ich dużą liczbę, w niniejszej pracy skupiono się wyłącznie na wybranych algorytmach, które stanowią punkt odniesienia podczas porównywania nowo zaproponowanego algorytmu inkrementalnego oraz oceny jego skuteczności w kontekście klasyfikacji. Są

to między innymi sieci neuronowe oparte na teorii rezonansu adaptacyjnego (adaptive resonance theory (ART) neural networks) zaproponowane w 1987 r. przez Carpenter i Grossberga [12]. Teoria rezonansu adaptacyjnego jest modelem teoretycznym, który opisuje reakcje organizmów i systemów adaptujących się do zmiennych warunków otoczenia. Znajduje zastosowanie w różnych dziedzinach życia. W informatyce może być wykorzystywana do projektowania systemów adaptacyjnych oraz algorytmów, które skutecznie reagują na zmiany w danych czy wymaganiach. Są one szczególnie przydatne w przypadkach, gdy potrzebne jest szybkie, stabilne i przyrostowe uczenie się w złożonym i zmiennym środowisku [13]. Od czasu wprowadzenia teorii rezonansu adaptacyjnego rodzina modeli sieci neuronowych opartych na tej teorii stale się rozszerza. Modele ART stosowane w nadzorowanym uczeniu się zazwyczaj posiadają architekturę ARTMAP (adaptive resonance theory map) [14]. ARTMAP ten składa się z dwóch głównych modułów, zwanych warstwami F1 (moduł uczący) i F2 (moduł decyzyjny). Warstwa F1 odpowiada za adaptacyjne przetwarzanie danych wejściowych, natomiast warstwa F2 przypisuje klasę do danych wejściowych na podstawie informacji uzyskanych z warstwy F1. Rozmyty ARTMAP (fuzzy ARTMAP, FAM) rozszerzył możliwości ARTMAP, umożliwiając przetwarzanie danych o wartościach rzeczywistych i zastępując operator logiczny AND rozmytym operatorem przecięcia AND [15]. Uproszczony rozmyty ARTMAP (simplified fuzzy ARTMAP, SFAM) został opracowany specjalnie do zadań klasyfikacyjnych [16]. W tej metodzie wektory reprezentujące etykiety klas zastąpiły jeden z modułów [17]. Inną uproszczoną architekturę SFAM omówiono w artykule [18].

Kolejnym przykładem klasyfikatora inkrementalnego jest drzewo Hoeffdinga (Hoeffding tree, HT). To algorytm drzewa decyzyjnego, które może być skonstruowane w sposób inkrementalny i umożliwia adaptacyjne uczenie się na podstawie strumieni danych, charakteryzujących się stałym rozkładem [19]. Drzewo adaptacyjne Hoeffdinga (Hoeffding adaptive tree, HAT) stanowi ulepszoną wersję HT, wprowadzając dodatkowe mechanizmy umożliwiające dostosowywanie się do zmiennych warunków danych [20]. Dzięki temu algorytm może lepiej radzić sobie z dynamicznymi strumieniami danych, poprawiając skuteczność klasyfikacji w porównaniu z tradycyjnym HT. Warto tutaj wspomnieć także algorytmy adaptacyjnego lasu losowego (adaptive random forest, ARF) oraz niezwykle szybkie drzewo decyzyjne (extremely fast decision tree, EFDT), które także umożliwiają adaptacyjne uczenie się na podstawie strumieni

danych. W przypadku ARF, drzewa decyzyjne są rozwijane na podstawie strumieni danych, co pozwala na dynamiczną aktualizację modelu w czasie rzeczywistym [21]. EFDT jest natomiast zoptymalizowaną wersją drzewa decyzyjnego, która ma za zadanie szybko i skutecznie radzić sobie z dynamicznymi strumieniami danych, minimalizując jednocześnie zużycie zasobów obliczeniowych [22].

Innym przykładem algorytmu klasyfikacji inkrementalnej jest Additive Expert Ensemble (AEE), który opiera się na zasadzie agregacji predykcji wielu ekspertów w celu uzyskania lepszych wyników klasyfikacji [23]. Algorytm dynamicznej większości ważonej (dynamic weighted majority, DWM) uwzględnia z kolei wagę klasyfikatorów, opierając się na ich skuteczności w przewidywaniu poprawnych klas, co pozwala na dynamiczne dostosowanie do zmieniających się warunków [24]. Innym algorytmem wartym uwagi jest Oza Bagging (OB), który wykorzystuje tworzenie i aktualizację dynamicznego zespołu klasyfikatorów w celu skutecznego klasyfikowania danych w strumieniach [25]. Warto też wspomnieć, że istnieją implementacje niektórych tradycyjnych algorytmów klasyfikacji w wersji inkrementalnej, na przykład KNN czy NB [26].

Niniejsza rozprawa składa się z dziewięciu rozdziałów, dwóch dodatków i spisu literatury. W rozdziale 2. sformułowano hipotezy badawcze, cele i zakres pracy. Rozdział 3. omawia wybrane algorytmy uczenia nadzorowanego do klasyfikacji danych, w tym rodzinę algorytmów uczenia się kwantyzacji wektorowej (LVQ) wraz z jego odmianami, ewoluującą kwantyzację wektorową oraz uproszczoną procedurę opartą na logice rozmytej i teorii rezonansu adaptacyjnego. W rozdziale 4. opisano metryki służące do oceny algorytmów klasyfikacji danych, w tym wybrane metody porównywania modeli klasyfikacyjnych, algorytm hierarchicznego grupowania danych Scotta–Knotta i test Wilcoxona. Nowy klasyfikator inkrementalny (SEVQ) oparty na kwantyzacji wektorowej i adaptacyjnej teorii rezonansu został zaproponowany w rozdziale 5., gdzie omówiono fazy działania tego algorytmu wraz z wizualizacją oraz szczegółowo przedyskutowano podobieństwa i różnice między SEVQ a podobnymi algorytmami. Rozdział 6. jest poświęcony porównaniu wyników SEVQ z innymi algorytmami, wraz z omówieniem zbiorów danych użytych do testowania algorytmów, charakterystyką algorytmów stanowiących punkt odniesienia w badaniach porównawczych i pokazaniem wyników tych badań. Porównano SEVQ z klasyfikatorami inkrementalnymi i tradycyjnymi, podano rankingi porównywanych algorytmów i dokonano analizy rozkładu wartości wyników klasyfikacji. Ponadto, przedstawiono wyniki algorytmu hierarchicz-

nego grupowania danych Scotta–Knotta, jak również wyniki testu Wilcoxona dla par obserwacji. Ze względu na dobre wyniki nowego klasyfikatora dokonano implementacji sprzętowej algorytmu SEVQ w układzie FPGA, która została opisana w rozdziale 7., łącznie z porównaniem implementacji sprzętowej z programową. Ze względu na wagę problemu porównywania nowych algorytmów, które wciąż powstają i będą opracowane w przyszłości, w rozdziale 8. opisano autorskie oprogramowanie służące do oceny jakości klasyfikacji. Podsumowanie i wnioski końcowe zawarto w rozdziale 9.

2. Cel i zakres pracy oraz hipotezy badawcze

Głównym **celem pracy** jest opracowanie i implementacja programowa i sprzętowa nowego inkrementalnego algorytmu klasyfikacji danych, który w klasie algorytmów płytkich okaże się nie gorszy od dotychczas stosowanych. W związku z tym sformułowano hipotezy badawcze.

Hipoteza 1: Możliwe jest opracowanie koncepcyjnie prostego algorytmu inkrementalnego uczenia pod nadzorem, który będzie osiągał wyniki zbliżone lub lepsze od innych powszechnie znanych algorytmów.

Hipoteza 2: Istnieje możliwość implementacji programowej oraz sprzętowej takiego algorytmu.

Hipoteza 3: Możliwe jest zbudowanie narzędzia automatycznie porównującego jakość klasyfikacji algorytmów nadzorowanego uczenia maszynowego, które nie tylko automatycznie wykona porównanie, ale także sprawdzi statystyczną wiarygodność wyników oraz przygotuje te rezultaty bezpośrednio do dzielenia się nimi w publikacjach naukowych.

W związku z postawionymi hipotezami badawczymi, powinny być zrealizowane następujące cele szczegółowe:

1) Zaproponowanie nowego algorytmu przyrostowego klasyfikacji danych, który posiadałby minimalną liczbę parametrów nastrajanych.

2) Stworzenie narzędzia i środowiska programistycznego do wszechstronnego testowania nowego algorytmu klasyfikacji danych (w języku Python), które spełniają istotne wymagania w zakresie badań benchmarkowych.

3) Przeprowadzenie wszechstronnych badań porównawczych nowego klasyfikatora, z uwzględnieniem odpowiednio dużej liczby zbiorów danych do klasyfikacji oraz dużej liczby dotychczas stosowanych algorytmów płytkich i istotnych wskaźników jakości klasyfikacji.

4) Zastosowanie metod statystycznych w celu ulokowania nowego algorytmu na odpowiedniej pozycji wśród algorytmów dotychczas stosowanych pod względem kryteriów stosowanych w klasyfikacji danych.

5) Próba zarówno programowej, jak i sprzętowej implementacji nowego algorytmu.

Zakres prac powinien obejmować przede wszystkim współcześnie stosowane płytkie algorytmy przyrostowe i w miarę możliwości algorytmy nieinkrementalne. Należy dokonać implementacji programowej w języku Python nowego algorytmu klasyfikacji danych i w miarę możliwości implementacji sprzętowej za pomocą układu FPGA.

3. Wybrane algorytmy uczenia nadzorowanego do klasyfikacji danych

Zaproponowany w niniejszej pracy klasyfikator inkrementalny (SEVQ) został szczegółowo opisany w rozdziale 5. W związku z tym, że jest on oparty na kwantyzacji wektorowej i adaptacyjnej teorii rezonansu, posiada on pewne mechanizmy występujące w dotychczas opracowanych algorytmach – EVQ, LVQ i SFAM. W celu pokazania istotnych różnic między tymi algorytmami a SEVQ, w kolejnych podrozdziałach 3.1-3.3, zostaną przedstawione ich krótkie opisy, mające na celu lepsze zrozumienie unikatowych cech proponowanego podejścia oraz jego przewag.

3.1. Uczenie się kwantyzacji wektorowej LVQ

3.1.1. Uczenie się kwantyzacji wektorowej LVQ1

Rodzina algorytmów uczenia się kwantyzacji wektorowej (learning vector quantization, LVQ) zyskała dużą uwagę naukowców jako potencjalne narzędzie do klasyfikacji strumieniowej ze względu na możliwości uczenia się online [27]. W skład rodziny LVQ wchodzą takie algorytmy, jak LVQ1, LVQ2, LVQ3, a także ulepszona wersja LVQ2 – LVQ2.1. Algorytmy te różnią się szczegółami implementacyjnymi, ale bazują na tych samych podstawach teoretycznych.

LVQ został wprowadzony przez Kohonena w 1990 r. jako algorytm klasyfikacji nadzorowanej oparty na kwantyzacji wektorowej, łączący procesy klastrowania i klasyfikacji [28]. Kluczowym elementem algorytmu LVQ jest zastosowanie tzw. książki kodującej (codebook), która przechowuje wektory kodowe (code vectors), zwane również centroidami. Te wektory są optymalizowane w procesie uczenia i służą do reprezentowania różnych klas. Proces klasyfikacji odbywa się w oparciu o paradygmat „zwycięzca bierze wszystko” (winner-takes-it-all), gdzie każda próbka przypisywana jest do najbliższego wektora kodowego, co przypomina działanie algorytmu najbliższych sąsiadów (KNN) [29].

Uczenie się kwantyzacji wektorowej to algorytm klasyfikacji nadzorowanej, który opiera się na prototypach [30]. Jego celem jest dostosowanie ich w taki sposób, aby skutecznie rozróżniały klasy w zbiorze uczącym. Algorytm przechodzi przez kolejne rekordy zbioru danych i dla każdego z nich wybiera najbliższy prototyp. Jeśli dany rekord i prototyp są tej samej klasy, następuje korekcja prototypu w celu lepszego dopasowania się

do rekordu. W przypadku gdy rekord i prototyp należą do różnych klas, następuje ich oddzielenie. Czynność ta jest powtarzana wielokrotnie, w zależności od rozmiaru zbioru uczącego. Aktualizacja wektorów wag odbywa się zgodnie z następującą procedurą [30]. Jeśli $\mathbf{x}$ i $\mathbf{w}_k$ należą do tej samej klasy, wówczas

$$\mathbf{w}_k = \mathbf{w}_k + \alpha(\mathbf{x} - \mathbf{w}_k) \tag{3.1}$$

W przeciwnym przypadku, gdy $\alpha > 0$

$$\mathbf{w}_k = \mathbf{w}_k - \alpha(\mathbf{x} - \mathbf{w}_k) \tag{3.2}$$

gdzie współczynnik uczenia α maleje monotonicznie z czasem

$$\alpha(i) = \alpha_0(1 - i/N_e) \tag{3.3}$$

przy czym i to epoka, N_e – maksymalna liczba epok, α_0 – początkowy współczynnik uczenia ($\alpha_0 = 0.4$ jest wartością domyślną). Hiperparametrami, które można dostosować, są: α_0 oraz k (liczba sąsiadów).

3.1.2. Uczenie się kwantyzacji wektorowej LVQ2

Uczenie się kwantyzacji wektorowej LVQ2 to ulepszona wersja algorytmu LVQ, która w procesie uczenia korzysta z dwóch wektorów wzorcowych: $\mathbf{w}_i$ i $\mathbf{w}_j$, które są najbliższymi reprezentantami $\mathbf{x}$, z których $\mathbf{w}_i$ należy do tej samej klasy co $\mathbf{x}$, a $\mathbf{w}_j$ do innej. Obydwa wektory wzorcowe są jednocześnie modyfikowane. Modyfikacja zostanie przeprowadzona jednak tylko wtedy, gdy $\mathbf{x}$ znajdzie się w przedziale wartości „okna" zdefiniowanego w następujący sposób:

$$\min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > \frac{1-s}{1+s} \tag{3.4}$$

gdzie $d_k = ||\mathbf{x} - \mathbf{w}_k||$ jest odległością euklidesową, s jest typową wartością „okna" (zwykle z przedziału $s \in [0.2, 0.6]$; jeśli $s = 1$, to okno nie zostało zdefiniowane). W algorytmie LVQ2.1 wektor został zdefiniowany w następujący sposób [30]. Jeśli $\mathbf{x}$ i $\mathbf{w}_i$ należą do tej samej klasy, to

$$\mathbf{w}_i = \mathbf{w}_i + \alpha(\mathbf{x} - \mathbf{w}_i) \tag{3.5}$$

Jeśli $\mathbf{x}$ i $\mathbf{w}_j$ należą do różnych klas, to

$$\mathbf{w}_j = \mathbf{w}_j - \alpha(\mathbf{x} - \mathbf{w}_j) \tag{3.6}$$

gdzie $\mathbf{w}_i$ i $\mathbf{w}_j$ to dwa wektory wzorcowe dla $\mathbf{x}$, a $\mathbf{x}$ znajduje się w przedziale wartości „okna". Współczynnik uczenia α maleje monotonicznie z czasem (jak w równaniu (3.3)). Parametry, które można ustawić, to: s – szerokość „okna", α_0, k – liczba sąsiadów oraz liczba epok N_e.

3.1.3. Uczenie się kwantyzacji wektorowej LVQ3

Uczenie się kwantyzacji wektorowej LVQ3 to ulepszona wersja algorytmu LVQ2.1, w którym nie był sprawdzany warunek, czy $\mathbf{w}_j$ w trakcie procesu uczenia nie zmienia swego położenia tak, że przestaje być dobrym aproksymatorem klasy, do której należy $\mathbf{x}$. Potrzeba wprowadzenia takiego zabezpieczenia doprowadziła do powstania kolejnej wersji algorytmu, w której aktualizacja wektora wag odbywa się w następujący sposób [30]:

$$\mathbf{w}_k = \mathbf{w}_k + \epsilon\alpha(\mathbf{x} - \mathbf{w}_k)$$

dla $k \in i, j$, jeśli $\mathbf{x}$, $\mathbf{w}_i$ i $\mathbf{w}_j$ należą do tej samej klasy. Dodatni parametr ϵ oznacza stosunek między dwoma najbliższymi podklasami, powodującymi podwójną aktualizację wag. Współczynnik uczenia α maleje w czasie (jak w równaniu (3.3)). Parametry, które można ustawić, to: s – szerokość „okna", ($s = 0.6$ jest wartością domyślną), α_0 (0.01 domyślnie), k – liczba sąsiadów ($k = 1$ lub 3 domyślnie), parametr $\epsilon \in [0.1, 0.5]$ oraz liczba epok N_e. Optymalna wartość ϵ zależy od szerokości „okna" (wartości s) i jest mniejsza dla węższych „okien". Wprowadzone usprawnienia mają na celu poprawę stabilności i efektywności procesu uczenia.

Istnieją także inne ewoluujące klasyfikatory rozmyte (eClass) zaproponowane w artykułach [31–35], które są również istotnymi algorytmami ze względu na możliwość rozpoczęcia nauki od zera oraz brak konieczności określenia liczby klas i reguł rozmytych. Są one szczególnie przydatne w zadaniach, w których liczba klas nie jest znana a dane mogą być bardzo złożone.

Należy także wspomnieć, że w algorytmach z rodziny LVQ dane wejściowe muszą być znormalizowane [30]. Informacja ta jest istotna w kontekście przeprowadzonych w rozdziale 6. badań porównawczych.

3.2. Ewoluująca kwantyzacja wektorowa

W 2008 roku Lughofer inspirowany podejściem sieci ART rozszerzył konwencjonalną kwantyzację wektorową (VQ). Wprowadził on tzw. parametr czujności (*vigilance parameter*) i zaproponował ewoluującą wersję kwantyzacji wektorowej (EVQ) [36]. Jest to nadzorowana wersja oryginalnej kwantyzacji wektorowej (VQ) oraz opisanego wcześniej algorytmu LVQ. Może ona rozwijać nowe klastry na żądanie, porównując przychodzące próbki z już wygenerowanymi klastrami. Podstawowe różnice między podejściami EVQ a LVQ obejmują: zmodyfikowaną strategię wyboru zwycięzcy, która opiera się na odległościach od powierzchni klastrów, a nie od ich centrów, określoną strategię ewolucji klastrów oraz specyficzny schemat klasyfikacji, który uwzględnia względną pozycję próbki do klasyfikacji w stosunku do granicy zasięgu wpływu dwóch sąsiednich klastrów [30]. Specyficzne dla tego algorytmu jest uwzględnienie etykiety klasy w procesie inkrementalnego klastrowania w celu osiągnięcia nadzorowanej procedury uczenia i klasyfikatora opartego na ewoluującym klastrowaniu. Zamiast przedstawiać pełen opis algorytmu, skupiono się tutaj na jego trzech kluczowych aspektach.

1) W EVQ definiowana jest macierz trafień (hit matrix), zawierająca w i-tym wierszu i j-tej kolumnie liczbę próbek wpadających do klastra i i klasy j. Jeśli nowa klasa jest wprowadzona przez nową próbkę wpadającą do klastra i, dodawana jest kolumna, której wpisy są ustawiane na 0, z wyjątkiem i-tego, który jest ustawiany na 1.

2) Etykieta klasy jest bezpośrednio uwzględniana w wektorach cech (jeśli liczba klas jest znana z góry).

3) Użycie parametru czujności, ponieważ sprawdzany jest warunek, czy odległość między rekordem danych $\mathbf{x}$ i $\mathbf{w}$ jest większa lub równa ρ [37]).

Algorytm wymaga danych znormalizowanych do przedziału $[0, 1]$.

3.3. Uproszczona procedura oparta na logice rozmytej i teorii rezonansu adaptacyjnego

Uproszczona procedura oparta na logice rozmytej i teorii rezonansu adaptacyjnego to prostsza i szybsza wersja FAM, identyfikowana z siecią neuronową [18]. SFAM

został zaprojektowany z myślą o zwiększeniu szybkości działania oraz uproszczeniu procesu uczenia.

Niech N będzie całkowitą liczbą neuronów w drugiej warstwie sieci SFAM („zwycięzca bierze wszystko"), czyli liczbą klas. Znormalizowany wektor wejściowy $\mathbf{x} \in \mathbb{R}^M$ został oznaczony jako $\mathbf{a} \in [0,1]^M$, a wejście do sieci neuronowej SFAM jest równe $\mathbf{I} = [\mathbf{a}, 1-\mathbf{a}] \in [0,1]^{2M}$. Poziom aktywności r-tego neuronu zdefiniowano jako $T_r(\mathbf{I}) = (\alpha + |\mathbf{w}_r|)^{-1} |\min(\mathbf{I}, \mathbf{w}_r)|1$, gdzie norma wektorowa $|\cdot|_1$ wektora $[\mu_1, \ldots, \mu_q] \in [0,1]^n$ jest sumą $|[\mu_1, \ldots, \mu_n]|_1 = \mu_1 + \ldots + \mu_n$ (tzw. sigma-licznością zbioru rozmytego). Algorytm SFAM można zwięźle opisać w następujący sposób:

1) Ustaw bazową wartość parametru czujności ρ.

2) Oblicz wejście sieci $\mathbf{I}$ i oblicz aktywności drugiej warstwy („zwycięzca bierze wszystko") $T_r(\mathbf{I})$ dla $r = 1, \ldots, N-1$, $T_N = T_0$.

3) Znajdź zwycięski neuron jako $K = \arg\max_{1 \leq r \leq N} T_r$. Jeśli zwycięski neuron jest nieprzypisany, przejdź do punktu 7.

4) Jeśli warunek rezonansu
$$\frac{|\min \mathbf{I}, \mathbf{w}_K|_1}{M} \geq \rho \tag{3.7}$$
jest spełniony (tj. wejście jest wystarczająco podobne do prototypu zwycięzcy), przejdź do punktu 5. Jeśli nie, zresetuj zwycięzcę ($T_K = -1$), przejdź do punktu 3. i sprawdź następnego zwycięzcę.

5) Jeśli etykieta klasy zwycięzcy zgadza się z etykietą klasy wejścia, zaktualizuj prototyp zwycięzcy
$$\mathbf{w}_K = \alpha \min(\mathbf{I}, \mathbf{w}_K) + \beta \mathbf{w}_K \tag{3.8}$$
i przejdź do kroku 9. W przeciwnym razie zresetuj zwycięzcę ($TK = -1$) oraz
$$\rho = \frac{|\min \mathbf{I}, \mathbf{w}_K|_1}{M} + \varepsilon \tag{3.9}$$

6) Jeśli $\rho > 1$ (występuje niezgodność danych), przejdź do punktu 9. W przeciwnym razie przejdź do punktu 3.

7) Przypisz dla zwycięskiego neuronu $\mathbf{w}_N = \mathbf{I}$ i ustaw etykietę klasy dla $\mathbf{I}$.

8) Utwórz nowy nieprzypisany neuron, a $N = N + 1$.

9) Przejdź do punktu 1 i powtórz algorytm dla następnego wejścia.

Algorytm wymaga danych znormalizowanych do przedziału $[0, 1]$, podobnie jak wspomniane wcześniej algorytmy z rodziny LVQ oraz algorytm EVQ.

4. Wybrane metody porównywania modeli klasyfikacyjnych

Istnieje wiele metod porównywania różnych modeli klasyfikacji pod względem ich skuteczności. Do najpopularniejszych z nich należą metryki wydajnościowe. Istnieją także inne metody, takie jak: walidacja krzyżowa (cross-validation), testy statystyczne, selekcja cech (feature selection) oraz dostrajanie hiperparametrów, które zostały krótko omówione w niniejszym rozdziale.

4.1. Metryki wydajnościowe

Pomimo rosnącej popularności algorytmów głębokich i dużych modeli językowych, zastosowanie algorytmów tradycyjnych oraz inkrementalnych jest nadal znaczące w wielu dziedzinach, m.in. przetwarzaniu sygnałów, przetwarzaniu obrazów czy analizie danych. Ocena jakości klasyfikatora wymaga zdefiniowania odpowiednich metryk wydajnościowych. Użytkownicy końcowi tych algorytmów oczekują jak najwyższych wartości tych metryk dla różnych zestawów danych oraz ich zrozumiałości, przejrzystości i interpretowalności [38, 39].

Zakładając, że zbiór testowy D' zawiera n par wejście-wyjście, używamy następującej notacji:

$L^n = L \times \ldots \times L$ oznacza iloczyn kartezjański n zbiorów L zawierających etykiety klas, jak w zależności (5.1),

$\mathbf{t} = [t_1, \ldots, t_n]$ oznacza wektor etykiet *rzeczywistych*, $\mathbf{t} \in L^n$,

$\mathbf{p} = [p_1, \ldots, p_n]$ oznacza wektor etykiet *przewidywanych*, $\mathbf{p} \in L^n$. Definiujemy podzbiór T_l zawierający wszystkie indeksy rekordów danych oznaczonych etykietą l, które są współrzędnymi wektora etykiet rzeczywistych

$$T_l = \left\{ u \ i \mid \exists \ \mathbf{x}_i \in \mathbb{R}^j, (\mathbf{x}_i, l) \in D' \text{ oraz } l \in \mathbf{t} \right\} \tag{4.1}$$

oraz podzbiór P_l zawierający wszystkie indeksy rekordów danych oznaczonych etykietą l, które są współrzędnymi wektora etykiet przewidywanych

$$P_l = \left\{ i \mid \exists \ \mathbf{x}_i \in \mathbb{R}^j, (\mathbf{x}_i, l) \in D' \text{ oraz } l \in \mathbf{p} \right\} \tag{4.2}$$

gdzie D' jest podzbiorem D jak w zależności (5.1). Oznaczamy moc zbioru X przez $|X|$. Macierz pomyłek została zdefiniowana w następujący sposób:

$$CM = \begin{array}{c|c|c|c} & P_1 & \dots & P_r \\ \hline T_1 & |T_1 \cap P_1| & \dots & |T_1 \cap P_r| \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline T_r & |T_r \cap P_1| & \dots & |T_r \cap P_r| \\ \hline \end{array}. \tag{4.3}$$

Opierając się na danych zawartych w macierzy pomyłek, można obliczyć wiele metryk wydajnościowych, które pozwalają na ocenę jakości klasyfikacji. Do najpopularniejszych metryk należą: dokładność, precyzja, czułość, miara F-1 oraz pole pod krzywą ROC.

Dokładność klasyfikacji to najczęściej podawany wskaźnik, który pozwala na ocenę jakości klasyfikacji. Określa stosunek liczby poprawnie sklasyfikowanych rekordów danych do całkowitej liczby tych danych w zbiorze testowym. Została zdefiniowana w następujący sposób

$$ACC = \frac{1}{n} \sum\nolimits_{l \in L} |T_l \cap P_l| \tag{4.4}$$

Głównym problemem ACC jest fakt, że może być myląca w przypadku niezrównoważonych danych. Gdy jedna klasa dominuje w zbiorze danych, klasyfikator może osiągnąć wysoką dokładność przez poprawne przewidywanie dominującej klasy, nawet jeśli klasyfikuje źle mniejszość przypadków. Prowadzi to do fałszywego wrażenia, że klasyfikator działa efektywnie. Dlatego, oprócz dokładności, warto obliczyć również inne metryki. Aby zniwelować wpływ niezrównoważenia danych, pozostałe metryki zostały zdefiniowane w sposób ważony, co pozwoli na bardziej wiarygodną ocenę skuteczności klasyfikatorów.

Precyzja ważona (Weighted precision, PRE) jest miarą oceny jakości klasyfikatora, która uwzględnia niezrównoważenie klas przez przypisanie wagi do każdej klasy proporcjonalnej do jej liczności w zbiorze danych. Jest to średnia precyzja dla każdej klasy

$$PRE = \sum\nolimits_{l \in L} v_l \cdot Pre_l \tag{4.5}$$

gdzie *precyzja dla etykiety* $l \in L$ oznaczona jako (Pre_l) rozważa pojedynczą klasę i mierzy liczbę poprawnych przewidywań dla tej klasy, znormalizowaną przez liczbę wystą-

pień tej etykiety w wyniku

$$Pre_l = \frac{|T_l \cap P_l|}{|P_l|} \tag{4.6}$$

przy czym waga (v_l) jest zdefiniowana przez udział danej klasy w całkowitej liczbie przypadków jako

$$v_l = \frac{1}{n} |T_l| \tag{4.7}$$

Czułość ważona (Weighted sensitivity, Recall, SEN) uwzględnia niezbalansowanie klas przez przypisanie wagi do każdej klasy proporcjonalnie do jej liczności w zbiorze danych i została zdefiniowana jako

$$SEN = \sum_{l \in L} v_l \cdot Sen_l \tag{4.8}$$

gdzie *czułość dla etykiety* $l \in L$ oznaczona jako Sen_l rozważa pojedynczą klasę i mierzy liczbę poprawnych przewidywań dla tej klasy, znormalizowaną przez liczbę wystąpień tej etykiety,

$$Sen_l = \frac{|T_l \cap P_l|}{|T_l|} \tag{4.9}$$

natomiast waga v_l jest zdefiniowana wzorem (4.7). Ważona czułość jest miarą zdolności klasyfikatora do poprawnego wykrywania przypadków danej klasy spośród wszystkich przypadków tej klasy.

Ważona miara F-beta (F-beta score, F-beta) łączy PRE i SEN w jedną wartość, uwzględniając niezrównoważenie klas. Waga dla każdej klasy jest proporcjonalna do jej liczności w zbiorze danych. F-beta została zdefiniowana jako

$$F_{\beta,l} = (1+\beta) \sum_{l \in L} v_l \cdot \frac{Pre_l \cdot Sen_l}{\beta^2 Pre_l + Sen_l} \tag{4.10}$$

gdzie waga v_l jest określona zależnością (4.7). W niniejszej pracy używa się *ważonej miary* F_1, zakładając, że $\beta = 1$. Dla Pre_l i Sen_l zdefiniowanych odpowiednio wzorami (4.6) i (4.9), otrzymuje się:

$$F_1 = \frac{2}{n} \sum_{l \in L} \frac{|T_l|}{|T_l| + |P_l|} |T_l \cap P_l| \tag{4.11}$$

Miara F1 uwzględnia zarówno zdolność klasyfikatora do identyfikacji każdej z klas, jak i niezrównoważenie klas w zbiorze danych. Zapobiega to dominacji bardziej licznych

klas i pozwala na obiektywne porównanie skuteczności klasyfikatorów.

W wielu zastosowaniach dokładność, precyzja, czułość czy wynik F_1 nie wystarczają. Jedną z najważniejszych miar efektywności klasyfikatora – według niektórych badań [40] nawet najważniejszą, jest AUC. Mierzy zdolność klasyfikatora do rozróżniania klasy w zależności od wartości progowej. W niniejszej pracy posłużono się ważoną wartością AUC obliczaną zgodnie ze wzorem:

$$AUC = \frac{1}{r(r-1)} \sum_{j=1}^{r} \sum_{\substack{k=1 \\ k \neq j}}^{r} \left(AUC_{j,k} + AUC_{k,j} \right) \tag{4.12}$$

Wartość AUC interpretuje się jako prawdopodobieństwo, że klasyfikator poprawnie sklasyfikuje losowo wybrane pozytywne próbki jako pozytywne. Im wartość AUC jest większa, tym klasyfikator jest lepszy.

4.2. Walidacja krzyżowa

Walidacja krzyżowa jest techniką, w której dane są dzielone na kilka części, a następnie każda część jest używana jako zbiór testowy, podczas gdy pozostałe części są używane jako zbiory uczące. Proces ten jest powtarzany kilka razy, aby uzyskać stabilne i obiektywne oszacowanie skuteczności modelu. Istnieje wiele wariantów tej metody:

- sprawdzian prosty – dzieli się w nim losowo próbę na rozłączne zbiory: uczący i testowy, gdzie zbiór testowy stanowi zwykle mniej niż 1/3 próby,

- k-krotny sprawdzian krzyżowy (k-fold cross-validation) – oryginalna próba jest dzielona na k podzbiorów; każdy z nich jest brany kolejno jako zbiór testowy, a pozostałe razem jako zbiór uczący, następnie wykonuje się analizę (analiza jest wykonywana k razy), po czym uzyskane k rezultaty zwykle uśrednia się w celu uzyskania jednego wyniku,

- „pozostaw jedną poza" (leave-one-out) – jest odmianą sprawdzianu k-krotnego, przy czym elementy podziału są jednoelementowe, tj. n-elementowa próba jest dzielona na n podzbiorów; metoda ta jest stosowana zwykle dla małych zbiorów danych,

- stratyfikowany sprawdzian krzyżowy (standard stratified cross-validation, SCV)

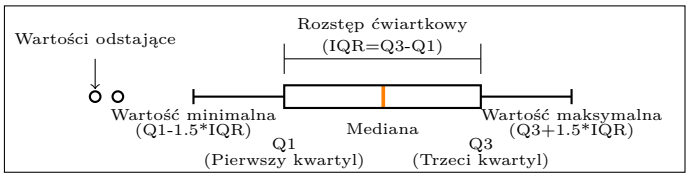
polega na podziale obiektów pomiędzy zbiór uczący i zbiór testowy, aby zachowane były oryginalne proporcje pomiędzy klasami decyzyjnymi,

- stratyfikowany sprawdzian krzyżowy optymalnie zrównoważony pod względem dystrybucji (distribution optimally balanced SCV, DOB-SCV) – metoda podziału danych na zbiór treningowy i testowy, która stara się utrzymać jak najbardziej zbliżony rozkład danych pomiędzy podzbiorami uczącymi i testowymi przez maksymalizację różnorodności pomiędzy podzbiorami i starając się, aby wszystkie podzbiory były jak najbardziej do siebie podobne. Ma to na celu uzyskanie bardziej wiarygodnych wyników wydajności modelu [41].

W niniejszej pracy do obliczeń wykorzystano wariant DOB-SCV, ze względu na wyższą dokładność testowania w porównaniu z innymi wariantami tej metody.

4.3. Analiza rozkładu wartości wyników klasyfikacji

Wykres pudełkowy jest formą reprezentacji graficznej rozkładu cechy statystycznej, która pozwala na wizualizację informacji dotyczących położenia, rozproszenia i kształtu rozkładu empirycznego badanej cechy. Elementy wykresu pudełkowego przedstawiono na rys. 4.1.



Rysunek 4.1: Elementy wykresu pudełkowego

Wykres pudełkowy konstruuje się przez naniesienie na poziomą oś wartości charakterystycznych dla rozkładu danych. Ponad tą osią rysuje się prostokąt, którego lewa krawędź jest zdefiniowana przez wartość pierwszego kwartylu, natomiast prawa przez wartość trzeciego kwartylu. Szerokość tego prostokąta reprezentuje zakres międzykwartylowy. Wewnątrz pudełka umieszczona jest linia pionowa, wskazująca położenie mediany. Do wykresu pudełkowego dodaje się również elementy zwane wąsami, które umieszcza się po obu stronach pudełka. Lewy koniec lewego wąsa reprezentuje najniższą wartość zbioru danych, a prawy koniec prawego wąsa – najwyższą wartość.

Za obserwacje odstające mogą być uznane na przykład obserwacje, które znajdują się o więcej niż 1,5 rozstępu ćwiartkowego poniżej pierwszego kwartyla lub powyżej trzeciego kwartyla [42].

4.4. Algorytm hierarchicznego grupowania danych Scotta–Knotta

Algorytm Scotta i Knotta, znany również jako metoda grupowania Scotta–Knotta [43], jest techniką używaną do hierarchicznego grupowania danych w kontekście analizy wariancji. Jego głównym celem jest identyfikacja istotnych różnic między grupami danych przez podział zbioru na bardziej jednorodne podgrupy.

Podstawowym krokiem w działaniu algorytmu Scotta i Knotta jest podział zbioru danych na dwie podgrupy, zwykle na podstawie pewnej metryki związanej z analizowanymi rezultatami. Następnie każda z uzyskanych podgrup jest oceniana pod kątem istotności różnic. Jeśli różnice są istotne, to podgrupy są dalej dzielone, tworząc hierarchię grup. Proces ten jest kontynuowany iteracyjnie, aż osiągnie się poziom podziału, który pozwala na jednoznaczną interpretację różnic między grupami.

4.5. Test Wilcoxona dla par obserwacji

Statystyczne porównanie wielu algorytmów dla wielu zbiorów danych jest zwykle wykonywane przy użyciu testów średnich rang. Jednak w kilku pracach wykazano, że wynik testu średnich rang zależy od puli pozostałych algorytmów uwzględnionych w eksperymencie. Aby rozwiązać ten problem, w niniejszej pracy proponuje się wykonanie testu oznaczonych rang Wilcoxona, którego wynik zależy tylko od dwóch porównywanych algorytmów. Co więcej, test sumy rang Wilcoxona nie przyjmuje założenia co do rozkładu danych. Jest to nieparametryczny test hipotezy statystycznej, mający na celu określenie, czy istnieje statystycznie istotna różnica między klasyfikatorami [44].

5. Nowy klasyfikator inkrementalny (SEVQ) oparty na kwantyzacji wektorowej i adaptacyjnej teorii rezonansu

W rozdziale omówiono nowy algorytm SEVQ (Simple Evolving Vector Quantization), który jest inkrementalnym klasyfikatorem opartym na kwantyzacji wektorowej i adaptacyjnej teorii rezonansu. Przedstawiono jego definicję oraz wizualizacje sposobu działania na wybranych zbiorach danych. Porównano także SEVQ z innymi algorytmami klasyfikacji: LVQ, EVQ i SFAM, którymi był inspirowany, zwracając szczególną uwagę na podobieństwa i różnice między nimi.

5.1. Definicja algorytmu

Dane dla algorytmu stanowią pary uporządkowane tworzące zbiór:

$$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_z, y_z)\} \subset \mathbb{R}^j \times L \tag{5.1}$$

gdzie $L = \{l_1, \ldots, l_r\}$ jest skończonym zbiorem oryginalnych etykiet (klas), $2 \leq r \leq z$, $\mathbb{R}^j$ – zbiorem j-wymiarowych wektorów rzeczywistych zwanych cechami (atrybutami), $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,j})$ oraz y_i to etykiety klas reprezentowane przez liczby naturalne. Jeśli $r = 2$, mamy do czynienia z klasyfikacją binarną. W przeciwnym przypadku jest to problem klasyfikacji wieloklasowej.

Prosty ewoluujący algorytm kwantyzacji wektorowej (Simple Evolving Vector Quantization Algorithm, SEVQ) jest nadzorowanym algorytmem inkrementalnym opartym na kwantyzacji wektorowej i adaptacyjnej teorii rezonansu. Algorytm generuje *kategorie* $c_1, \ldots, c_k$, $(k \leq r)$, z których każda odpowiada podzbiorowi rekordów przypisanych do danej etykiety. Do każdej *kategorii* c_i jest przypisany jeden *wektor wag* $\mathbf{w}_i \in \mathbb{R}^j$ i dwa skalary n_i i l_i, gdzie n_i to liczba rekordów ze zbioru D biorących udział w obliczeniu kategorii, a l_i to etykieta klasy ze zbioru L, do którego jest przypisana kategoria. Dane wejściowe są na bieżąco wykorzystywane do aktualizacji istniejących wag. Uczenie można prowadzić rekord po rekordzie lub na partii rekordów.

Zaproponowany algorytm składa się z dwóch faz. Faza pierwsza (faza uczenia) została zaprezentowana na Listingu 1. Dane wejściowe tego algorytmu stanowią oznaczony zbiór danych D i liczba epok N_e określona przez użytkownika, natomiast danymi

wyjściowymi są wektory wag $\mathbf{w}_1, \ldots, \mathbf{w}_k$, liczba rekordów partycypujących w tworzeniu danej kategorii $n_1, \ldots, n_k$, i etykiety kategorii $l_1, \ldots, l_k$ $(1 \leq k \leq r)$. Z przeprowadzonych doświadczeń wynika, że najlepiej jest podawać te same dane w różnej kolejności (dane podawane są w pętli „for" w drugiej linii Listingu 1).

Listing 1: Faza uczenia algorytmu SEVQ zapisana w pseudokodzie

Wejścia:

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_z, y_z)$: oznaczony zbiór danych

N_e: liczba epok

Wyjścia:

$\mathbf{w}_1, \ldots, \mathbf{w}_k$: wektory wag

$n_1, \ldots, n_k$: liczba rekordów partycypujących w tworzeniu danej kategorii

$l_1, \ldots, l_k$: etykiety kategorii

```
1  k = 0
2  for 1, 2, ··· , N_e, do
3      for i = 1, 2, ··· , z, do
4          znajdź indeks c = arg min_{s∈{1,...,k}−{i}} ||x_i − w_s||,
5          dla znalezionego indeksu c sprawdź:
6          if l_c = y_i then
7              zaktualizuj instniejące wagi kategorii:
8              w_c = w_c + (1/n_c)(x_i − w_c)
9              n_c = n_c + 1
10         else
11             utwórz nową kategorię:
12             k = k + 1
13             w_k = x_i
14             n_k = 1
15             l_k = y_i
```

Pierwszym krokiem tej fazy jest obliczenie odległości euklidesowych $\|.\|$ pomiędzy aktualnym rekordem danych $\mathbf{x}_i$ a wagą każdej kategorii $\mathbf{w}_1, \ldots, \mathbf{w}_k$ i znalezienie indeksu c takiego wektora $\mathbf{w}_\cdot$ spośród $\mathbf{w}_1, \cdots, \mathbf{w}_k$, który jest najbliższy danemu wektorowi $\mathbf{x}_i$.

Jeśli etykieta kategorii ma tę samą etykietę co rekord danych, istniejąca waga

kategorii jest aktualizowana przez dodanie do niej różnicy między rekordem danych a wagą poprzedniej kategorii podzieloną przez liczbę rekordów poprzednio użytych do wytrenowania tej kategorii. W przypadku gdy etykiety się różnią, tworzona jest nowa kategoria i przypisywany jest do niej aktualny rekord danych.

W fazie drugiej (fazie przewidywania), zaprezentowanej na Listingu 2, algorytm wyszukuje kategorię najbliższą każdemu testowemu rekordowi $\mathbf{x}_i$ i przypisuje mu etykietę tego wektora wag spośród $\mathbf{w}_1, \cdots, \mathbf{w}_k$, który jest najbliższy (w sensie odległości euklidesowej) danemu rekordowi $\mathbf{x}_i$. Ze względu na prostotę fazy przewidywania, szybkość klasyfikacji jest proporcjonalna do liczby kategorii.

Listing 2: Faza przewidywania algorytmu SEVQ zapisana w pseudokodzie

Wejścia:

$\mathbf{x}_1, \ldots, \mathbf{x}_{z'}$: wektory cech danych

Ładowanie wstępnie wytrenowanego modelu:

$\mathbf{w}_1, \ldots, \mathbf{w}_k$: wektory wag

$n_1, \ldots, n_k$: liczba rekordów partycypujących w tworzeniu kategorii

$l_1, \ldots, l_k$: etykiety kategorii

Wyjścia:

$y_1, \ldots, y_{z'}$: etykieta danych

```
1 for i = 1, 2, ..., z', do
2     znajdź indeks c = arg min_{s∈{1,...,k}} ||x_i − w_s||
3     y_i = l_c
```

Rysunek 5.1 stanowi macierzową strukturę zmiennych występujących w algorytmie SEVQ. Prezentuje macierz wag kategorii, wektor etykiet kategorii oraz wektor zawierający liczbę próbek biorących udział w tworzeniu danej kategorii.

Kategoria 1	$w_{1,1}$	$w_{1,2}$	$\cdots$	$w_{1,j}$	l_1	n_1
Kategoria 2	$w_{2,1}$	$w_{2,2}$		$w_{2,j}$	l_2	n_2
$\vdots$	$\cdots$	$\cdots$	$\ddots$	$\cdots$	$\vdots$	$\vdots$
Kategoria k	$w_{k,1}$	$w_{k,2}$	$\cdots$	$w_{k,j}$	l_k	n_k

Rysunek 5.1: Graficzna reprezentacja algorytmu SEVQ, gdzie k oznacza indeks kategorii, j – wymiar wektora wag, $l_1, \ldots, l_k$ – etykiety, $n_1, \ldots, n_k$ – liczbę rekordów partycypujących w tworzeniu danej kategorii

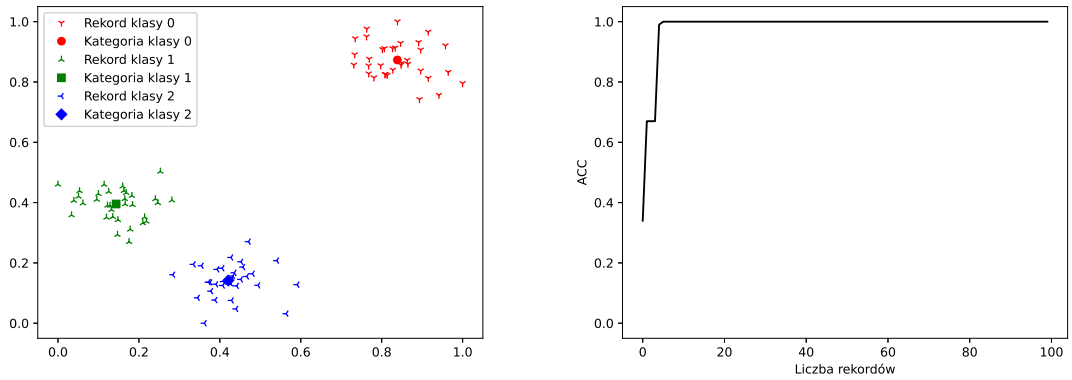
Algorytm SEVQ nie wymaga ręcznego ustawiania żadnych parametrów ani normalizacji danych. Użytkownicy podczas uczenia wsadowego w fazie uczenia mają jednak możliwość ustawienia opcjonalnego parametru, którym jest liczba epok (N_e). Ustawienie liczby epok jako $N_e = 1$ wystarcza, aby algorytm działał dobrze na większości zbiorów danych. Aby poprawić uzyskiwane wyniki, można zwiększyć wartość tego parametru, jednak wartość ta nie powinna być większa niż 10, ponieważ zbyt duża liczba epok może prowadzić do generowania zbyt dużej liczby kategorii. Jest to efekt przypominający zjawisko nadmiernego dopasowania (overfitting) polegającego na idealnym dopasowaniu się modelu do danych treningowych, ale słabej generalizacji nowych danych. Kod źródłowy algorytmu został udostępniony w postaci otwartego oprogramowania pod adresem `https://github.com/sylwekczmil/sevq`.

5.2. Wizualizacje działania algorytmu

Wizualizacje 2D przedstawione na rys. 5.2–5.5 zostały przygotowane w celu zilustrowania zasady działania algorytmu. Każda z wizualizacji zawiera wykres rekordów i przypisanych im kategorii (a), a także odpowiadający im wykres dokładności w zależności od liczby przetworzonych rekordów (b). Wizualizacje zostały przygotowane wyłącznie w celach poglądowych, aby podkreślić przyrostowy charakter działania algorytmu. Proces uczenia i testowania został przeprowadzony na tych samych danych z domyślnymi ustawieniami parametrów algorytmu. Do przygotowania wizualizacji użyto czterech syntetycznych zbiorów danych wygenerowanych za pomocą biblioteki Scikit-learn [45].

Pierwszy z wygenerowanych zbiorów danych zawierał 100 punktów rozłożonych w postaci trzech skupisk (rys. 5.2a). Poszczególne punkty należące do danych klas zaznaczono na wykresie kolorem zielonym, niebieskim i czerwonym. Algorytm wygenerował dla każdej klasy dokładnie jedną kategorię, która została oznaczona na wykresie kropką w kolorze odpowiadającym kolorowi klasy (rys. 5.2a). Dokładność klasyfikacji przedstawiona na rys. 5.2b, która zależy od liczby próbek wykorzystanych do uczenia, ustala się na stałym poziomie równym 1 (100%). Oznacza to zbieżność procesu uczenia oraz osiągnięcie idealnego wyniku pod względem dokładności. Co więcej, wynik ten został uzyskany po podaniu jedynie pięciu próbek. W przypadku przedstawionym na rys. 5.2b, pierwszy losowo wybrany przykład pochodził z klasy „0”, trzy przykłady pochodziły z klasy „1”, a ostatni przykład pochodził z klasy „2”. W optymistycznym

przypadku, gdyby z każdej klasy został wylosowany tylko jeden przypadek, to prawdopodobnie okazałoby się, że do uzyskania 100% dokładności wystarczają jedynie trzy przykłady.



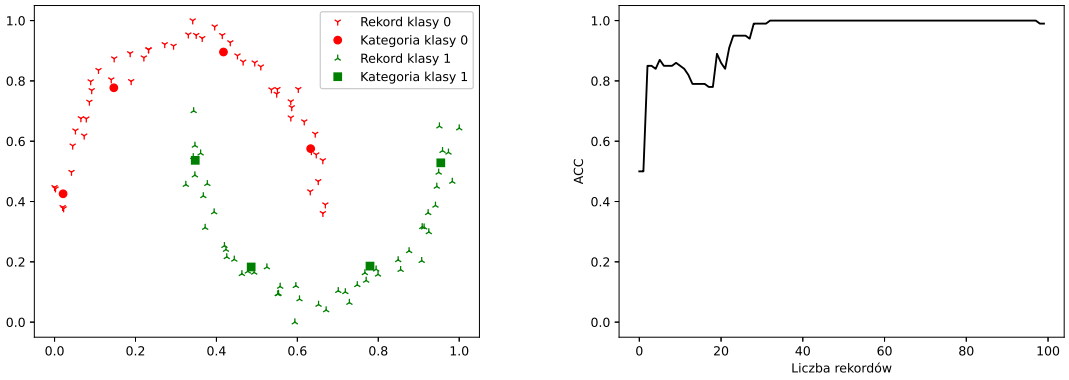
(a) (b)

Rysunek 5.2: Zbiór danych w przestrzeni cech w postaci trzech skupisk odpowiadających trzem klasom ze zbioru 0, 1, 2: (a) wizualizacja dwuwymiarowych danych przed uczeniem i po procesie uczenia, (b) wykres zależności między dokładnością klasyfikacji a liczbą próbek wykorzystanych do uczenia

Jako drugi wygenerowany został zbiór danych przedstawiony na rys. 5.3a, zawierający 100 punktów ułożonych w postaci pary księżyców skierowanych ku sobie. Ten zbiór danych nie jest liniowo separowalny. Dla punktów należących do pierwszej kategorii i oznaczonych kolorem czerwonym algorytm wygenerował trzy kategorie, natomiast dla punktów należących do drugiej kategorii algorytm utworzył cztery kategorie. Ten przypadek był bardziej skomplikowany niż poprzedni, dlatego algorytm potrzebował większej liczby przykładów. Dokładność walidacji na poziomie 100% uzyskano po dostarczeniu 31 losowo wybranych próbek.

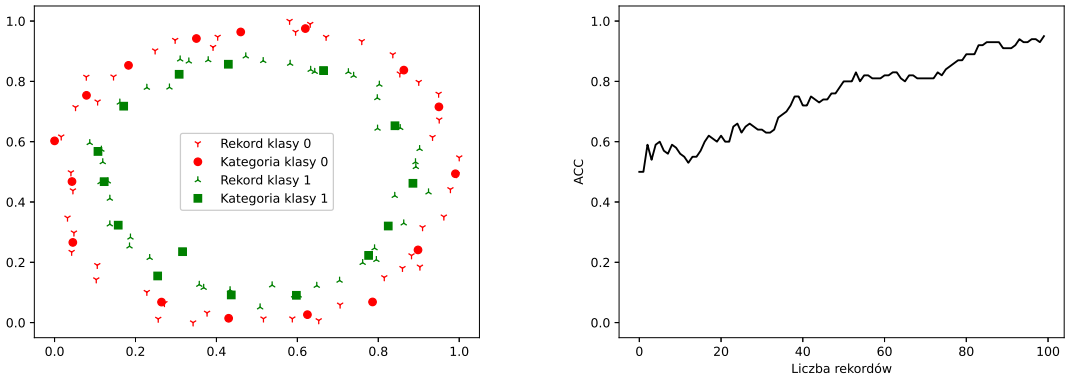
Kolejny zbiór danych stanowiły dwa koncentryczne okręgi przedstawione na rys. 5.4a, z których każdy zawierał 50 punktów. Algorytm SEVQ utworzył 16 klas dla punktów należących do klasy „0” i 14 dla punktów należących do klasy „1”. Po podaniu wszystkich przykładów dokładność klasyfikacji wynosiła 96%.

Ostatni zestaw danych składał się z dwóch spiral przedstawionych na rys. 5.5. Każda ze spiral zawierała 100 punktów. Problem „dwóch spiral” jest uznawany za ogólnie trudny i często stosowany w testach klasyfikatorów [46]. Mimo że jest on prosty do zobrazowania, wiele klasyfikatorów napotyka ogromne trudności w nauczeniu się



(a) (b)

Rysunek 5.3: Zbiór danych w przestrzeni cech w postaci pary księżyców skierowanych ku sobie: (a) wizualizacja danych przed uczeniem i po procesie uczenia, (b) wykres zależności między dokładnością klasyfikacji a liczbą próbek wykorzystanych do uczenia

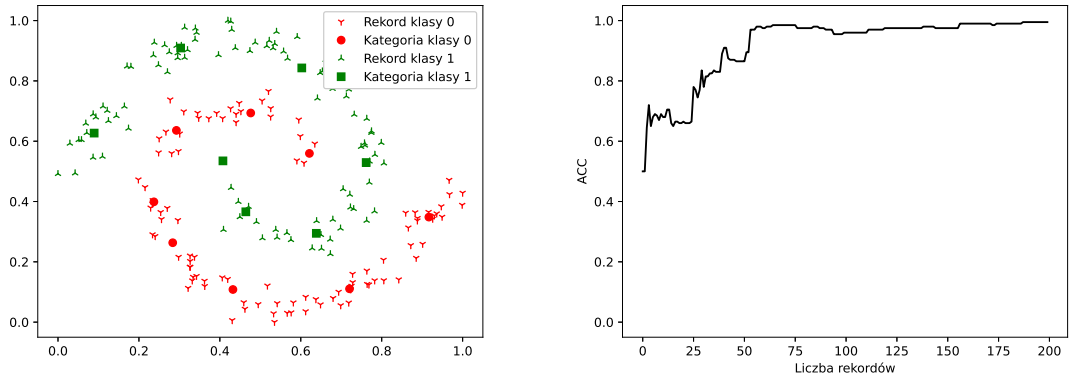


(a) (b)

Rysunek 5.4: Zbiór danych w przestrzeni cech w postaci dwóch koncentrycznych okręgów: (a) wizualizacja danych przed uczeniem i po procesie uczenia, (b) wykres zależności między dokładnością klasyfikacji a liczbą próbek wykorzystanych do uczenia

tego układu ze względu na jego skrajną nieliniowość. SEVQ utworzył 8 kategorii dla każdej klasy. Po podaniu 158 przykładów osiągnął 100% dokładności.

Z przedstawionych eksperymentów wynika, że aby uzyskać najlepsze rezultaty, warto po kolei dostarczać punkty należące do różnych klas. Poprawę wyników działania algorytmu może zapewnić zastosowanie wielu epok uczenia. Na ostateczne wyniki wpływa także kolejność podawania przykładów. Korzystne jest podawanie pró-



(a) (b)

Rysunek 5.5: Zbiór danych w przestrzeni cech w postaci „dwóch spiral”: (a) wizualizacja danych przed uczeniem i po procesie uczenia, (b) wykres zależności między dokładnością klasyfikacji a liczbą próbek wykorzystanych do uczenia

bek w różnej kolejności, ponieważ ma ono wpływ na wygenerowanie innych kategorii. W przypadku stosowania wielu epok warto również za każdym razem przetasować kolejność przykładów.

5.3. Podobieństwa i różnice między SEVQ a podobnymi algorytmami

5.3.1. LVQ

W przeciwieństwie do algorytmu LVQ dokładnie opisanego w podrozdziale 3.1, który charakteryzuje się stałą liczbą prototypów, SEVQ cechuje się dynamiczną zmianą liczby kategorii w trakcie procesu uczenia. Algorytmy różnią się sposobem zmiany tempa douczania. Wybór kategorii przez SEVQ jest prostszy w porównaniu z LVQ. W przypadku SEVQ istotną zaletę stanowi brak konieczności dostrajania parametrów wejściowych w przeciwieństwie do LVQ, który wymaga precyzyjnego określenia wielu parametrów, takich jak liczba prototypów oraz ustawienie początkowych wartości wektorów słownika [30, 47].

5.3.2. EVQ

Główne różnice między SEVQ i EVQ polegają na tym, że w fazie uczenia się EVQ wykorzystuje parametr czujności, na podstawie którego podejmuje decyzję o aktualizacji najbliższego klastra lub utworzeniu nowego. Z kolei SEVQ działa według algorytmu wyjaśnionego w rozdziale 5. i nie używa podobnego parametru. Konsekwen-

cją tych różnic jest wpływ na konstrukcję modeli. W EVQ używana jest macierz trafień (hit matrix), która służy do zapisywania, ile razy klaster został zaktualizowany przez rekord przypisany do określonej klasy. W przypadku EVQ wiele rekordów z różnych klas może zaktualizować ten sam klaster. SEVQ nie korzysta z macierzy trafień, ponieważ każda kategoria zawiera tylko jedną etykietę klasy. Tylko rekordy należące do tej samej klasy mogą aktualizować przynależność do klasy. Dodatkowo algorytmy uczą się z różną szybkością. EVQ wykorzystuje współczynnik uczenia $\alpha = \frac{0,5}{n}$, podczas gdy SEVQ korzysta ze współczynnika $\alpha = \frac{1}{n}$. Szybkość uczenia równa $\frac{1}{n}$ w przypadku SEVQ gwarantuje, że waga kategorii będzie dokładnie pośrodku rekordów danych użytych do jej tworzenia i aktualizacji.

5.3.3. SFAM

Zarówno algorytm SEVQ, jak i SFAM mają zdolność stopniowego generowania nowych kategorii. Warto jednak zaznaczyć, że SEVQ różni się od SFAM w kontekście aktualizacji wag prototypów oraz kryterium dopasowania. SEVQ opiera się na odległości, podczas gdy SFAM wykorzystuje parametr *czujności* do podejmowania decyzji odnośnie do aktualizacji klastrów. W przypadku SFAM wybór czterech dodatkowych parametrów nie jest trywialnym zadaniem, ponieważ wymaga precyzyjnego dostosowania ich do konkretnego zbioru danych. W przeciwieństwie do tego algorytm SEVQ nie posiada takich parametrów, co upraszcza jego implementację.

6. Porównanie wyników SEVQ z innymi algorytmami

W rozdziale omówiono przeprowadzone eksperymenty porównawcze algorytmu SEVQ w zestawieniu z różnymi algorytmami klasyfikacji, zarówno tradycyjnymi, jak i inkrementalnymi. Zaprezentowano metodykę analizy oraz szczegółowe wyniki zaproponowanych eksperymentów. Na koniec wykonano również analizę statystyczną.

6.1. Zbiory danych

W przeprowadzonych eksperymentach wykorzystano 36 standardowych zbiorów danych klasyfikacyjnych o wartościach numerycznych, które zostały wybrane spośród zbiorów dostępnych w repozytorium zbiorów danych KEEL [48]. Wszystkie wykorzystane w badaniu zbiory danych są dostępne publicznie, co umożliwia odtworzenie i weryfikację otrzymanych wyników. Rekordy danych we wszystkich zbiorach składają się z atrybutów liczbowych oraz przypisanych im klas. KEEL umożliwia pobieranie zarówno całych zbiorów danych, jak i zbiorów danych podzielonych za pomocą procedur 5-krotnej i 10-krotnej walidacji krzyżowej, a także 5-krotnej i 10-krotnej procedury DOB-SCV. W przedstawionych badaniach wykorzystano zbiory podzielone za pomocą procedury 10-krotnej procedury DOB-SCV.

W tabeli 6.1 zamieszczono szczegółowe informacje dotyczące wykorzystanych w badaniach zbiorów danych. Zawarto w niej nazwę każdego zbioru danych, liczbę rekordów, liczbę atrybutów, liczbę klas oraz stopień niezrównoważenia każdego zbioru danych. Stopnień niezrównoważenia został wyznaczony jako średnia obliczona odpowiednich par klas.

6.2. Algorytmy stanowiące punkt odniesienia w badaniach porównawczych

Algorytm SEVQ został porównany z dwiema grupami klasyfikatorów. Pierwsza grupa zawierała algorytmy tradycyjne (nieprzyrostowe), które zostały zaimplementowane przy użyciu biblioteki Scikit-learn [45], z wyjątkiem algorytmu XGB, zaimplementowanego przy użyciu biblioteki xgboost [6].

1) AdaBoost (AB) – algorytm, który łączy wyniki wielu słabych klasyfikatorów, aby stworzyć silniejszy klasyfikator. W badaniu wykorzystano implementację

Tabela 6.1: Zbiory danych wykorzystane w eksperymentach (w kolejności alfabetycznej)

Lp.	**Nazwa zbioru danych**	**Liczba rekordów**	**Liczba atrybutów**	**Liczba klas**	**Stopień niezrównoważenia**
1	appendicitis	106	7	2	0,2471
2	australian	690	14	2	0,8016
3	banana	5300	2	2	0,8126
4	bupa	345	6	2	0,7250
5	cleveland	297	13	5	0,4136
6	coil2000	9822	85	2	0,0634
7	contraceptive	1473	9	3	0,6645
8	dermatology	358	34	6	0,5647
9	glass	214	9	6	0,4079
10	hayes-roth	160	4	3	0,6486
11	heart	270	13	2	0,8000
12	hepatitis	80	19	2	0,1940
13	led7digit	500	7	10	0,8739
14	mammographic	830	5	2	0,9438
15	marketing	6876	13	9	0,7003
16	monk-2	432	6	2	0,8947
17	movement libras	360	90	15	1,0000
18	newthyroid	215	5	3	0,4302
19	optdigits	5620	64	10	0,9858
20	page-blocks	5472	10	5	0,2135
21	phoneme	5404	5	2	0,4154
22	ring	7400	20	2	0,9807
23	satimage	6435	36	6	0,6479
24	segment	2310	19	7	1,0000
25	shuttle	57999	9	7	0,1143
26	spambase	4597	57	2	0,6506
27	spectfheart	267	44	2	0,2594
28	tae	151	5	3	0,9613
29	texture	5500	40	11	1,0000
30	thyroid	7200	21	3	0,1771
31	titanic	2201	3	4	0,5380
32	twonorm	7400	20	2	0,9984
33	vowel	990	13	11	1,0000
34	wine	178	13	3	0,7735
35	wisconsin	683	9	2	0,5383
36	zoo	101	16	7	0,4364

AdaBoostClassifier. Maksymalną liczbę estymatorów, przy których kończy się wzmacnianie, ustawiono na 50, a szybkość uczenia była ustawiona na 1.

2) Drzewo decyzyjne (DT) – algorytm klasyfikacji opierający się na drzewie decyzyjnym [49]. Wykorzystano implementację *DecisionTreeClassifier*, a maksymalną głębokość drzewa ustalono na 5.

3) Gaussowski naiwny klasyfikator Bayesa (GNB) – algorytm opierający się na twierdzeniu Bayesa i założeniu naiwności, które przyjmuje, że występowanie jednej cechy nie zależy od występowania innych. Implementacja Gaussowska zakłada dodatkowo, że cechy każdej klasy mają rozkład normalny (Gaussa) [50]. Algorytm *GaussianNB* został użyty z domyślnymi parametrami.

4) K-najbliższych sąsiadów (KNN) – algorytm zaliczający obiekty do tej klasy, która stanowi większość w grupie k najbliższych sąsiadów w przestrzeni cech, gdzie k jest parametrem określającym liczbę sąsiadów branych pod uwagę [51]. Wykorzystano implementację *KNeighborsClassifier*, a liczba sąsiadów została ustawiona na 3.

5) Perceptron wielowarstwowy (MLP) – algorytm klasyfikacji, który używa algorytmu wstecznej propagacji błędów do uczenia sieci neuronowej [52]. Użyto klasyfikatora *MLPClassifier*, parametr *alpha* oznaczający wagę regularyzacji L2 ustawiono na 1, a maksymalną liczbę iteracji na 10 000.

6) Najbliższy centroid (NC) – prosty algorytm klasyfikacji, który reprezentuje każdą klasę za pomocą centroida, a nowe próbki są klasyfikowane na podstawie najbliższego centroida według odległości euklidesowej [53]. Użyto domyślnych ustawień parametrów dla klasyfikatora *NearestCentroid*.

7) Kwadratowa analiza dyskryminacyjna (QDA) – algorytm uczenia maszynowego stosowany w zadaniach klasyfikacji, który opiera się na analizie dyskryminacyjnej [54]. Użyto algorytmu *QuadraticDiscriminantAnalysis* z domyślnymi ustawieniami parametrów.

8) Las losowy (RF) – to algorytm uczenia maszynowego, który tworzy zrównoważony las losowy, składający się z wielu drzew decyzyjnych, gdzie każde drzewo

jest trenowane na losowym podzbiorze danych [4]. Użyto implementacji *RandomForestClassifier*. Liczbę estymatorów ustawiono na 10, maksymalna głębokość drzewa wynosiła 5, a liczba cech branych pod uwagę przy poszukiwaniu najlepszego podziału to 1.

9) C-Support Vector Classification (SVC) – implementacja maszyny wektorów wspierających (SVM) oparta na bibliotece LibSVM [55] z radialną funkcją jądra [56]. Zastosowano domyślne ustawienia parametrów dla klasyfikatora *SVC*.

10) XGBoost (XGB) – algorytm uczenia maszynowego, który wykorzystuje technikę boostingu, sekwencyjnie trenując modele drzew decyzyjnych w celu poprawy predykcji. Wprowadza innowacyjne mechanizmy optymalizacyjne, takie jak regularyzacja, co sprawia, że jest bardzo skuteczny i odporny na zjawisko nadmiernego dopasowania (overfitting) [57]. Użyto współczynnika błędów dla klasyfikacji wieloklasowej, a maksymalną głębokość każdego drzewa ustawiono na 5.

Druga grupa zawiera algorytmy inkrementalne zaimplementowane głównie przy użyciu bibliotek scikit-multiflow [26] i NeuPy [58]. Wyjątki stanowią algorytmy SFAM z implementacją opartą na kodzie AIOpenLab [59] i EVQ, którego implementacja została przygotowana specjalnie na potrzeby niniejszej pracy [60].

1) Additive Expert Ensemble (AEE) – metoda wykorzystania dowolnego ucznia online do modelowania dryfu koncepcji [23]. Zastosowano domyślne ustawienia parametrów dla klasyfikatora *AdditiveExpertEnsembleClassifier*.

2) Adaptacyjny las losowy (ARF) – algorytm klasyfikacji, który poprawia zdolność adaptacji do zmiennych dzięki metodom ponownego próbkowania i operatorom adaptacyjnym, które radzą sobie z różnymi typami dryfów koncepcji [21]. Dla klasyfikatora *ARFClassifier* zastosowano domyślne ustawienia parametrów.

3) Dynamic Weighted Majority (DWM) – algorytm adaptacyjny, który skutecznie radzi sobie z problemem dryfu koncepcji w danych strumieniowych. Wykorzystuje cztery mechanizmy, takie jak uczenie online, przypisywanie wag na podstawie wydajności, usuwanie i dodawanie nowych ekspertów, aby dostosować się do ewoluujących warunków i utrzymać wysoką skuteczność [24]. Klasyfikatora

DynamicWeightedMajorityClassifier użyto z domyślnymi ustawieniami parametrów.

4) Extremely Fast Decision Tree (EFDT) – metoda inkrementalnego konstruowania drzewa decyzyjnego, która wybiera i implementuje podziały, gdy jest to przydatne, systematycznie powracając do tej decyzji i potencjalnie zastępując zaimplementowane już podziały lepszymi, aby dostosować się do zmiennych warunków [22]. Klasyfikator *ExtremelyFastDecisionTreeClassifier* użyto z domyślnymi ustawieniami parametrów.

5) Adaptacyjne drzewo Hoeffdinga (HAT) – wykorzystuje metodę wykrywania dryfu ADWIN do monitorowania wydajności gałęzi w drzewie i zastępowania ich nowymi gałęziami w przypadku spadku dokładności, pod warunkiem, że nowe gałęzie są dokładniejsze [20]. Dla klasyfikatora *HoeffdingTreeClassifier* zastosowano domyślne ustawienia parametrów.

6) Hoeffding Tree (HT) – algorytm inkrementalnego budowania drzewa decyzyjnego, zdolny do uczenia się na dużych strumieniach danych, z założeniem, że rozkład generujący przykłady nie zmienia się w czasie [19]. Wykorzystano implementację opartą na MOA [61]. Dla klasyfikatora *HoeffdingTreeClassifier* zastosowano domyślne ustawienia parametrów.

7) K-najbliższych sąsiadów (KNNI) – inkrementalna wersja klasyfikatora KNN, która śledzi ostatnią maksymalną szerokość „okna" próbek treningowych, a przewidywana klasa dla danej próbki jest uzyskiwana przez znalezienie najbliższych sąsiadów w oknie danych i agregację ich etykiet klas [26]. Zastosowano domyślne ustawienia parametrów dla klasyfikatora *KNNClassifier*.

8) Naiwny Bayes (NB) – klasyfikator, który dokonuje predykcji na podstawie klasycznego podejścia bayesowskiego, zakładając naiwnie niezależność wszystkich wejść [26]. Dla klasyfikatora *NaiveBayes* atrybuty były numeryczne.

9) Oza Bagging (OB) – metoda uczenia zespołowego dostosowana do obsługi strumieni danych, gdzie każda napływająca próbka jest trenowana wielokrotnie zgodnie z rozkładem dwumianowym, co umożliwia efektywne radzenie sobie ze strumieniem danych o nieskończonej wielkości [25]. Zastosowano domyślne ustawienia parametrów dla klasyfikatora *OzaBaggingClassifier*.

10) SFAM – implementacja algorytmu została oparta na kodzie AIOpenLab [59]. Szybkość uczenia ustawiono na 1, waga regularyzacji wynosiła 0,0001, natomiast wartość parametru czujności *vigilance* to 0,75.

11) EVQ – wykorzystano własną implementację opartą na pseudokodzie EVQ-Class (wariant B) zawartym w pracy [37] z wartością parametru czujności *vigilance* ustawioną na 0,2.

12) LVQ, LVQ2, LVQ2.1, LVQ3 – podstawowy algorytm oparty na kwantyzacji wektorowej oraz jego odmiany. Wykorzystano implementację i domyślne ustawienia parametrów zawarte w [58].

Wszystkie parametry konfiguracyjne porównywanych algorytmów są dostępne w dokumentacji pod adresem `https://sevq.readthedocs.io/`.

6.3. Wyniki badań porównawczych

6.3.1. Sposób prezentacji wyników

Wyniki eksperymentów dla algorytmów opisanych w podrozdziale 6.2, na 36 zbiorach danych opisanych w podrozdziale 6.1 zostały zebrane w tab. 6.2-6.5. Wyniki porównań algorytmów przedstawiono w osobnych tabelach dla algorytmów tradycyjnych i inkrementalnych, ze względu na konieczność znormalizowania danych dla metody SFAM oraz algorytmów z rodziny LVQ, z którymi głównie porównywano wyniki. W związku z tym, podczas testowania algorytmów inkrementalnych zastosowano dane znormalizowane, natomiast dla algorytmów tradycyjnych użyto danych nieznormalizowanych.

Tabele 6.2 i 6.3 zawierają wyniki porównania wydajności klasyfikatorów tradycyjnych i inkrementalnych. Dla każdego algorytmu podano średnie wartości oraz odchylenia standardowe dla metryk wydajnościowych, takich jak ACC, PRE, SEN, F1 i AUC. Wyniki obliczono dla wszystkich zbiorów danych na każdej partii danych wydzielonej podczas 10-krotnej walidacji krzyżowej dla każdego algorytmu. Wyniki zostały posortowane malejąco według wartości AUC.

6.3.2. Porównanie SEVQ z klasyfikatorami tradycyjnymi

Algorytm XGB uzyskał najwyższą wartość AUC (0,840±0,149) oraz pozostałych metryk wydajnościowych. MLP również wykazał się wysoką skutecznością, uzyskując

Tabela 6.2: Wyniki porównania tradycyjnych algorytmów

Lp.	Algorytm	AUC	ACC	PRE	SEN	F1
1	XGB	0,840±0,149	0,829±0,173	0,831±0,175	0,829±0,173	0,822±0,182
2	MLP	0,788±0,168	0,799±0,181	0,791±0,193	0,799±0,181	0,783±0,196
3	SEVQ	0,787±0,165	0,764±0,221	0,779±0,211	0,764±0,221	0,763±0,222
4	GNB	0,786±0,141	0,703±0,223	0,774±0,189	0,703±0,223	0,699±0,230
5	DT	0,785±0,140	0,757±0,185	0,758±0,191	0,757±0,185	0,743±0,196
6	RF	0,782±0,149	0,793±0,175	0,783±0,187	0,793±0,175	0,775±0,191
7	KNN	0,779±0,170	0,776±0,205	0,779±0,211	0,776±0,205	0,766±0,213
8	AB	0,751±0,150	0,701±0,246	0,681±0,270	0,701±0,246	0,675±0,271
9	SVC	0,746±0,175	0,763±0,202	0,736±0,237	0,763±0,202	0,733±0,228
10	QDA	0,746±0,176	0,675±0,284	0,698±0,287	0,675±0,284	0,655±0,300
11	NC	0,716±0,145	0,629±0,201	0,700±0,194	0,629±0,201	0,632±0,201

AUC na poziomie 0,788±0,168, co potwierdza jego efektywność na badanych zbiorach danych. SEVQ zajął trzecie miejsce pod względem wartości AUC (0,787±0,165) i piąte miejsce pod względem ACC (0,764±0,221). Pomimo nieco niższych wyników, algorytm ten radzi sobie znacznie lepiej od większości porównywanych algorytmów. Najniższe rezultaty osiągnął klasyfikator NC, którego średnia wartość AUC wyniosła 0,716±0,145, a ACC – 0,629±0,201.

6.3.3. Porównanie SEVQ z klasyfikatorami inkrementalnymi

Należy podkreślić, że SEVQ jest dedykowany dla problemów uczenia inkrementalnego, zatem porównanie go z klasyfikatorami nieinkrementalnymi ma charakter raczej formalny. Najważniejsze jest porównanie SEVQ z algorytmami inkrementalnymi. Wśród algorytmów przyrostowych (tab. 6.3) SEVQ uzyskał najlepsze wyniki dla AUC wynoszącego 0,794±0,166, a następnie NB (0,791±0,144) i EVQ (0,783±0,160). Zajął również drugie miejsce pod względem ACC (0,775 ±0,218), a lepszy od niego był jedynie EVQ (0,785 ±0,193). Najgorsze wartości ze średnimi AUC i ACC osiągnął algorytm LVQ3 – odpowiednio 0,671±0,164 i 0,658±0,213.

W porównaniu algorytmów inkrementalnych przedstawionych w tab. 6.3, algorytm SEVQ uzyskał najwyższe rezultaty, osiągając AUC na poziomie 0,794±0,166. Kolejne miejsca pod względem AUC zajęły algorytmy NB (0,791±0,144) i EVQ (0,783±0,160). Według ACC SEVQ uzyskał drugi najlepszy wynik wynoszący 0,775± 0,218. Lepszy był tylko algorytm EVQ z wartością ACC równą 0,785±0,193. Najniższe średnie wyniki AUC i ACC osiągnął algorytm LVQ3. Wynosiły one odpowiednio 0,671±0,164 i 0,658±0,213.

Tabela 6.3: Wyniki porównania algorytmów inkrementalnych

Lp.	Algorytm	AUC	ACC	PRE	SEN	F1
1	SEVQ	0,794±0,166	0,775±0,218	0,789±0,207	0,775±0,218	0,774±0,219
2	NB	0,791±0,144	0,756±0,192	0,769±0,199	0,756±0,192	0,747±0,205
3	EVQ	0,783±0,160	0,785±0,193	0,791±0,193	0,785±0,193	0,778±0,198
4	HAT	0,780±0,159	0,761±0,193	0,750±0,217	0,761±0,193	0,741±0,213
5	SFAM	0,779±0,155	0,759±0,204	0,779±0,196	0,759±0,204	0,757±0,205
6	HT	0,775±0,159	0,762±0,192	0,748±0,221	0,762±0,192	0,741±0,213
7	AEE	0,765±0,149	0,718±0,218	0,723±0,233	0,718±0,218	0,701±0,234
8	OB	0,762±0,169	0,763±0,217	0,759±0,233	0,763±0,217	0,745±0,235
9	KNNI	0,760±0,168	0,759±0,217	0,756±0,233	0,759±0,217	0,742±0,234
10	EFDT	0,757±0,157	0,736±0,212	0,733±0,233	0,736±0,212	0,713±0,232
11	ARF	0,753±0,165	0,717±0,242	0,683±0,285	0,717±0,242	0,681±0,279
12	LVQ2	0,750±0,160	0,732±0,223	0,726±0,237	0,732±0,223	0,715±0,240
13	LVQ	0,738±0,166	0,725±0,229	0,710±0,254	0,725±0,229	0,699±0,255
14	LVQ2,1	0,736±0,167	0,729±0,227	0,707±0,266	0,729±0,227	0,699±0,260
15	DWM	0,735±0,160	0,674±0,262	0,675±0,288	0,674±0,262	0,656±0,283
16	LVQ3	0,671±0,164	0,658±0,213	0,649±0,226	0,658±0,213	0,626±0,223

6.4. Rankingi porównywanych algorytmów

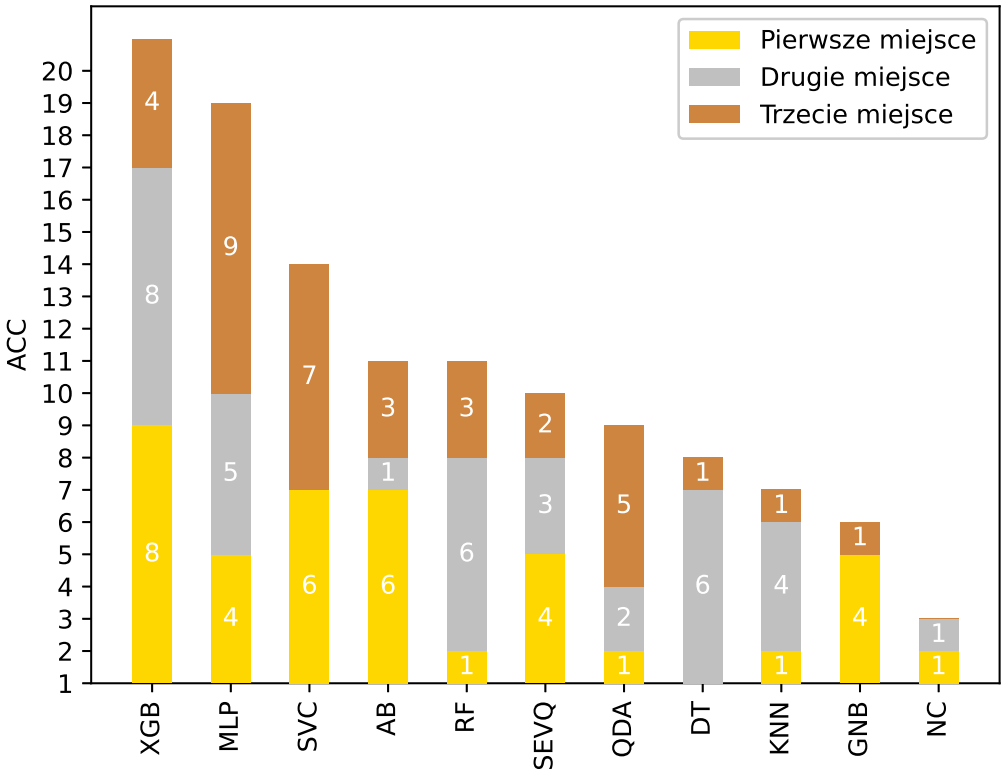
6.4.1. Przygotowanie rankingów

Wykresy zamieszczone na rys. 6.1-6.10 przedstawiają wyniki porównania algorytmów omówionych w podrozdziale 6.2. Rankingi przygotowano na podstawie wyników poszczególnych algorytmów obliczonych dla wszystkich zbiorów danych (tab. 6.1), na każdej partii danych wydzielonej podczas 10-krotnej walidacji krzyżowej dla każdego algorytmu. Rozpatruje się trzy główne kategorie: algorytmy, które osiągnęły najlepszy wynik (pierwsze miejsce), drugi najlepszy wynik i trzeci najlepszy wynik pod względem wybranych metryk wydajnościowych, na podstawie analizy 36 zbiorów danych numerycznych. Każdy słupek na wykresie odpowiada jednemu algorytmowi i jest podzielony na trzy segmenty reprezentujące, ile razy dany model zajął odpowiednio pierwsze, drugie i trzecie miejsce. Porównanie rankingów algorytmów wykonano w dwóch etapach. Najpierw zestawiono wyniki algorytmów tradycyjnych pod względem metryk klasyfikacyjnych, takich jak ACC, AUC, PRE, SEN oraz F1. Następnie oceniono algorytmy inkrementalne pod względem tych samych metryk klasyfikacyjnych.

6.4.2. Ranking algorytmów tradycyjnych

Wykres słupkowy przedstawiony na rys. 6.1 prezentuje liczbę zbiorów danych, na których poszczególne algorytmy tradycyjne osiągnęły najlepsze wyniki pod względem ACC. Algorytm XGB osiągnął najwięcej pierwszych miejsc (7 razy), drugich miejsc (7 razy) i trzecich miejsc (4 razy). Inne modele, takie jak MLP i SVC, również regularnie

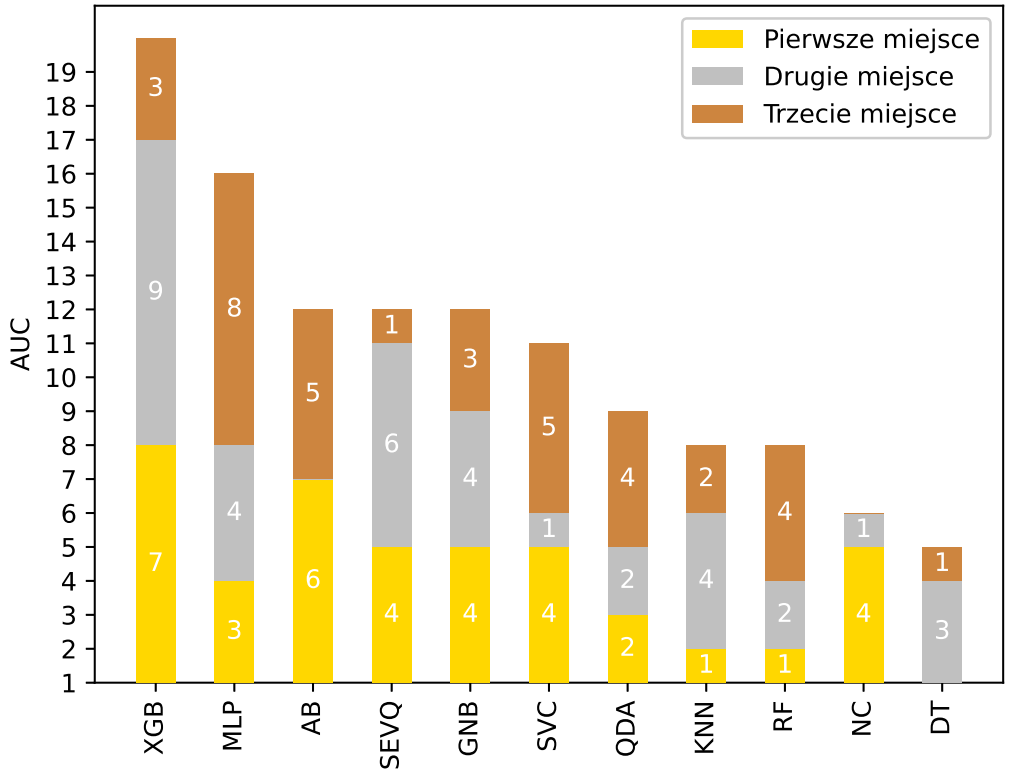
pojawiały się wśród najlepszych. Algorytm SEVQ plasuje się w środkowej części wykresu, z czterema pierwszymi miejscami, trzema drugimi i dwoma trzecimi. Model NC zdobył najmniej punktów, zajmując tylko raz pierwsze i drugie miejsce.



Rysunek 6.1: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy tradycyjne osiągnęły najlepsze wyniki pod względem ACC

Na rysunku 6.2 przedstawiono ranking algorytmów, które najczęściej osiągały najwyższe wyniki AUC podczas klasyfikacji. Liderem ponownie został algorytm XGB, który zdobył najwięcej pierwszych miejsc (7 razy), drugich miejsc (9 razy) i trzecich miejsc (3 razy). Na kolejnych miejscach znalazły się algorytmy MLP, AB i SEVQ (który zdobył 4 pierwsze miejsca, 6 drugich miejsc oraz 1 trzecie miejsce). Na najniższym miejscu w rankingu znalazł się algorytm DT, który zajął trzy razy drugie i raz trzecie miejsce.

Wykres słupkowy na rys. 6.3 prezentuje liczbę zbiorów danych, na których różne algorytmy osiągnęły najwyższe wyniki pod względem metryki PRE. XGB dominuje w rankingu, z największą liczbą pierwszych miejsc (7 razy), drugich miejsc (5 razy) i trzecich miejsc (7 razy). Inne algorytmy, takie jak MLP i GNB, również wykazują

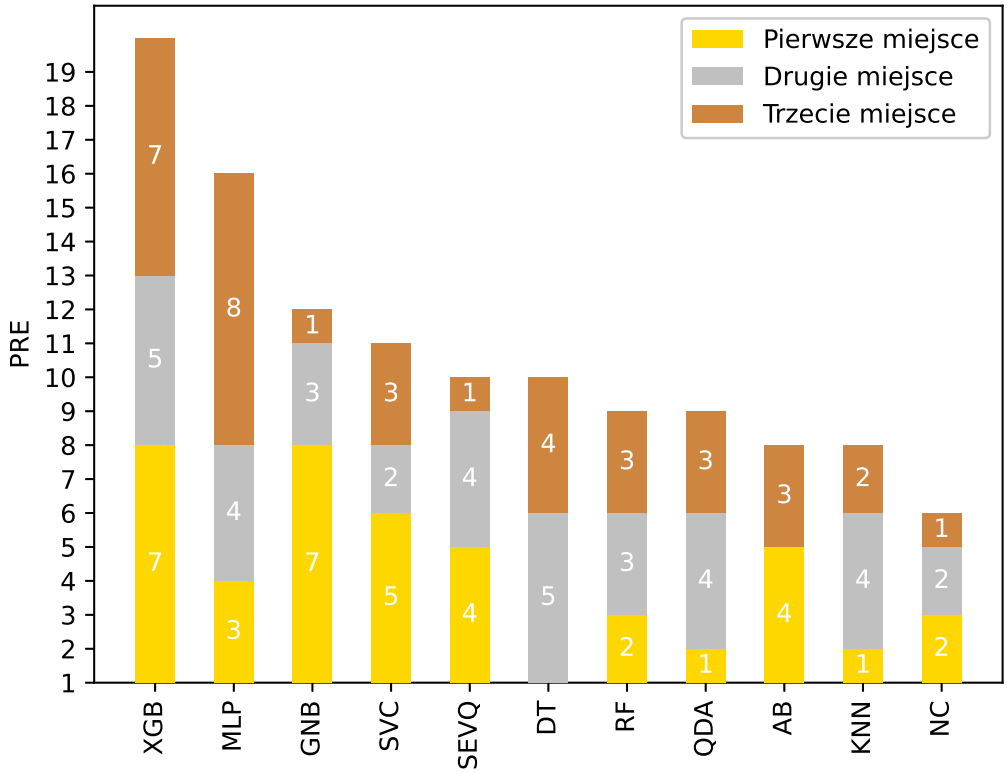


Rysunek 6.2: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy tradycyjne osiągnęły najlepsze wyniki pod względem AUC

bardzo dobre wyniki. Algorytm SEVQ osiągnął 4 razy pierwsze miejsce, co jest czwartym najlepszym wynikiem po XGB, GNB i SVC, i znacząco przewyższa pozostałe modele pod względem liczby zdobytych pierwszych miejsc.

Na rysunku 6.4 przedstawiono ranking algorytmów pod względem SEN. XGB ponownie wyróżnia się na tle pozostałych algorytmów, osiągając pierwsze miejsce 8 razy, drugie miejsce 8 razy i trzecie miejsce 4 razy. MLP i SVC wykazują również wysokie wyniki, chociaż w przypadku liczby pierwszych miejsc lepiej od algorytmu MLP wypadł AB. Algorytm SEVQ ponowie zajmuje środkową pozycję w rankingu, z 4 pierwszymi miejscami, 3 drugimi miejscami i 2 trzecimi miejscami, co wskazuje na jego umiarkowaną wydajność wśród analizowanych modeli.

Z wykresu prezentującego ranking algorytmów pod względem F1, przedstawionego na rys. 6.5 wynika, że XGB osiągnął pierwsze miejsce 8 razy (co jest najwyższą wartością na wykresie), drugie miejsce 10 razy i trzecie miejsce 4 razy. Bardzo dobre wyniki osiągają także algorytmy MLP i SVC. Algorytm SEVQ znalazł się na środkowej

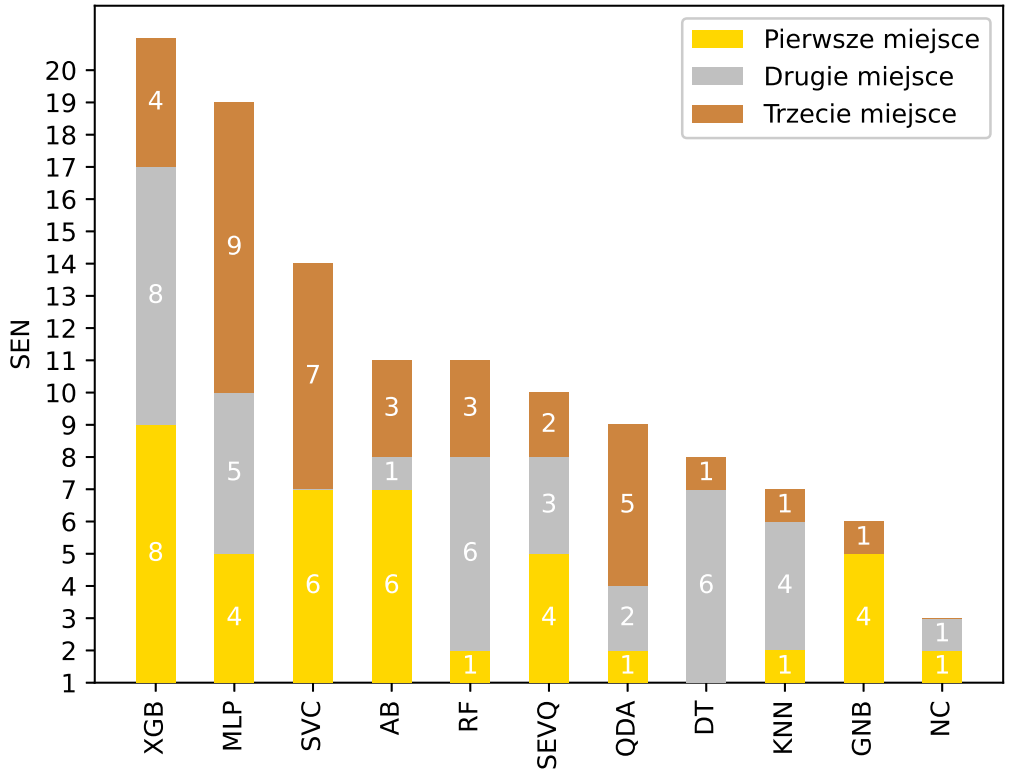


Rysunek 6.3: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy tradycyjne osiągnęły najlepsze wyniki pod względem PRE

pozycji wykresu, osiągając pierwsze miejsce 4 razy, drugie miejsce 4 razy i trzecie miejsce 1 raz, co wskazuje na jego stosunkowo wysoką efektywność w porównaniu z innymi modelami. Jest to szczególnie widoczne w kontekście drugiego miejsca, gdzie znacznie SEVQ wyprzedza inne algorytmy.

6.4.3. Ranking algorytmów inkrementalnych

Rysunek 6.6 prezentuje ranking algorytmów inkrementalnych w kontekście ich efektywności zbadanej na 36 różnych zbiorach danych oceniony za pomocą wskaźnika ACC. Algorytm SEVQ wyróżnia się największą liczbą pierwszych miejsc (5 razy), drugich miejsc (6 razy) i trzecich miejsc (2 razy), co dobrze świadczy na korzyść tego algorytmu w sensie najbardziej intuicyjnego i powszechnie stosowanego wskaźnika jakości klasyfikacji – ACC. Bardzo dobre wyniki osiągnęły też algorytmy EVQ (6 pierwszych miejsc, 3 drugie miejsca oraz 3 trzecie miejsca) i SFAM (3 pierwsze miejsca, 4 drugie miejsca oraz 5 trzecich miejsc). Najsłabiej w tym zestawieniu wypadły algorytmy NB, HT, LVQ2 i LVQ3, ponieważ żaden z nich nie zdołał osiągnąć najlepszego wyniku ACC

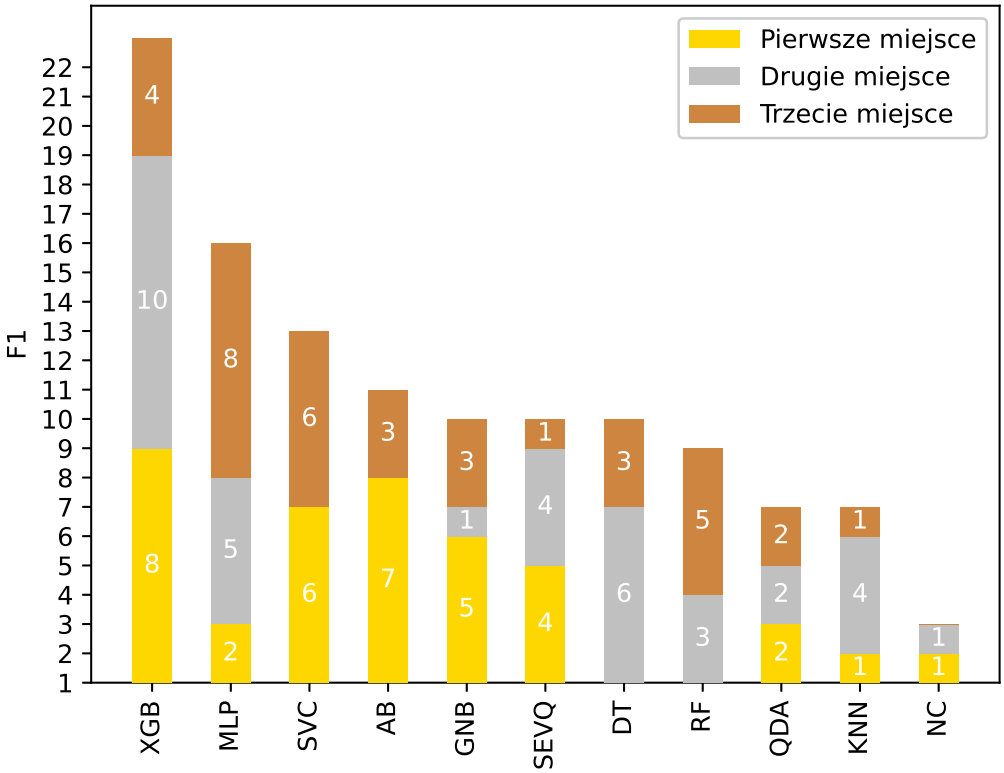


Rysunek 6.4: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy tradycyjne osiągnęły najlepsze wyniki pod względem SEN

na żadnym zbiorze danych.

Na rysunku 6.7 przedstawiono ranking algorytmów inkrementalnych z wykorzystaniem wskaźnika AUC. Algorytm SEVQ odznacza się największą liczbą zdobytych pierwszych miejsc (6 razy), drugich miejsc (6 razy) i trzecich miejsc (2 razy). Algorytmy EVQ i SFAM również notują wysokie pozycje, z odpowiednio czteroma, sześcioma i dwoma miejscami na podium dla EVQ oraz czterema, trzema i pięcioma dla SFAM. W przeciwieństwie do nich, algorytmy LVQ, HT, LVQ3 nie wykazują najlepszych rezultatów w żadnym z testowanych zbiorów, co wskazuje na ich ograniczoną skuteczność w kontekście AUC. Warto w tym miejscu przypomnieć, że większość badań wskazuje na to, że wysokie wartości AUC dla wielu zbiorów danych świadczą na korzyść klasyfikatora SEVQ, jako dobrze nadającego się dla danych niezrównoważonych.

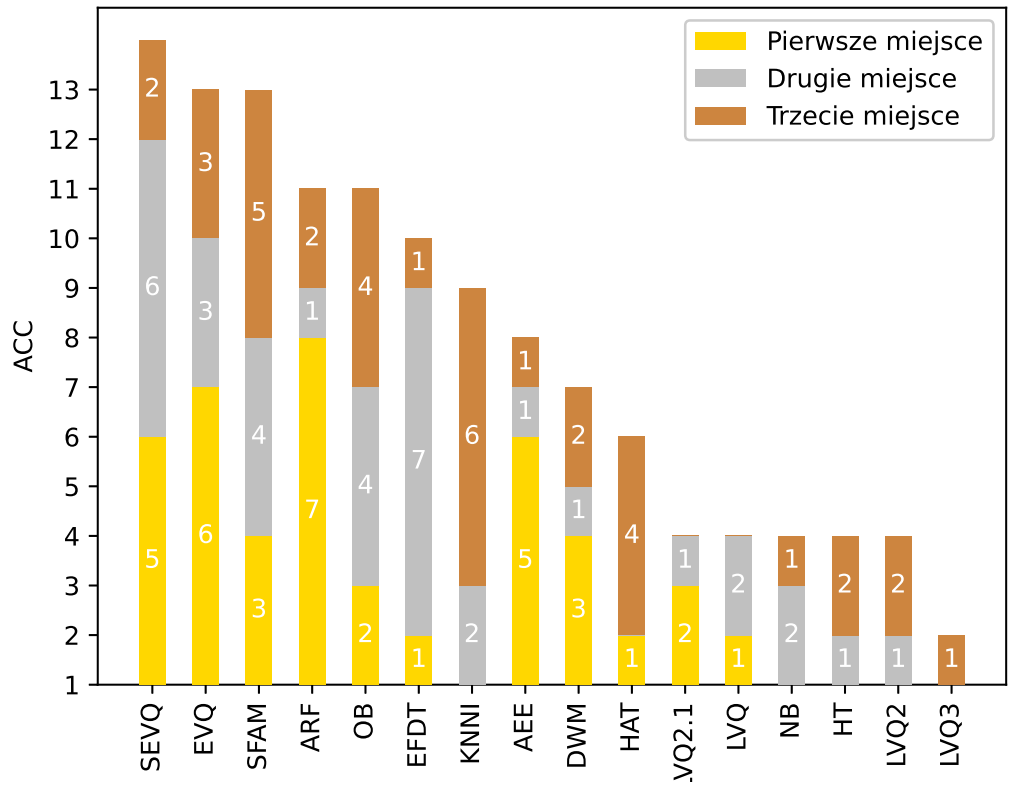
Rysunek 6.8 przedstawia ranking algorytmów inkrementalnych z wykorzystaniem wskaźnika PRE. Największą liczbę pierwszych miejsc w tym zestawieniu osiągnęły algorytmy EVQ i ARF (6 razy), a po nich SEVQ i DWM (po 5 razy). Algorytm SEVQ



Rysunek 6.5: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy tradycyjne osiągnęły najlepsze wyniki pod względem F1

najczęściej znajdował się w czołówce rankingu, ponieważ 7 razy zajął drugie miejsce i 3 razy trzecie miejsce. Dla kontrastu, modele takie jak HAT, LVQ2.1, HT, LVQ, LVQ3 i LVQ2 prezentują niższą efektywność w kontekście PRE, odnotowując najmniejszą liczbę wygranych. Jak widać, algorytm SEVQ wykazuje się lepszą efektywnością pod względem wartości wskaźnika PRE, w porównaniu z innymi algorytmami należącymi do rodziny LVQ.

Na rysunku 6.9 przedstawiono liczbę zbiorów danych, na których poszczególne algorytmy tradycyjne osiągnęły najlepsze wyniki pod względem SEN. Ponownie algorytm SEVQ najczęściej znajdował się w czołówce rankingu. Największą liczbą zdobytych pierwszych miejsc odznaczył się algorytm ARF (7 razy), a po nim EVQ (6 razy) oraz SEVQ i AEE (po 5 razy). Ponownie SEVQ najczęściej znajdował się w czołówce rankingu, ponieważ drugie miejsce zajął 6 razy, a trzecie miejsce – 3 razy. Dla kontrastu, modele takie jak LVQ2.1, LVQ, NB, HT, LVQ2 i LVQ3 najrzadziej osiągały najlepsze wyniki.



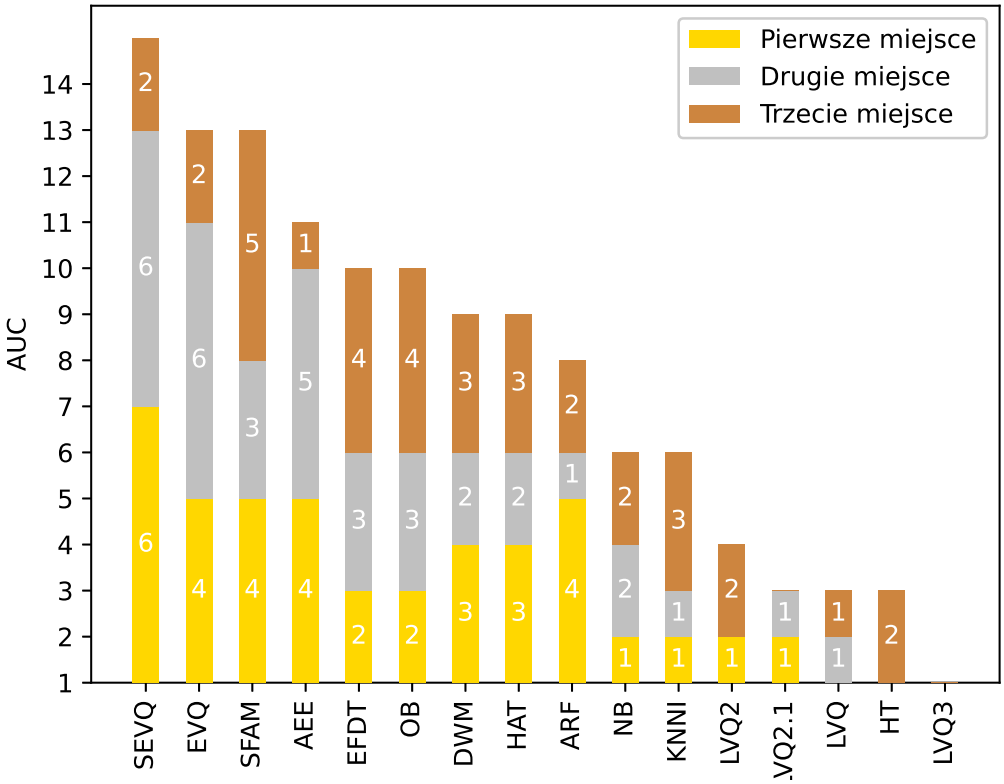
Rysunek 6.6: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy inkrementalne osiągnęły najlepsze wyniki pod względem ACC

W przypadku metryki F1, dla której ranking algorytmów przedstawiono na rys. 6.10, algorytm EVQ wykazał się wyjątkową skutecznością, najczęściej zdobywając pierwsze miejsce (7 razy). Tuż za nim uplasowały się algorytmy ARF (6 razy) oraz SEVQ (5 razy). SEVQ ponownie wykazał silną pozycję, pojawiając się najczęściej wśród liderów zestawienia i zajmując drugie miejsce (6 razy) oraz trzecie miejsce (3 razy). Ponadto znacznie przewyższył on takie modele, jak LVQ2.1, NB, LVQ, HT i LVQ3.

6.5. Analiza rozkładu wartości wyników klasyfikacji

6.5.1. Porównanie rozkładów wartości wskaźników jakości dla metryk klasyfikacyjnych dla algorytmów tradycyjnych

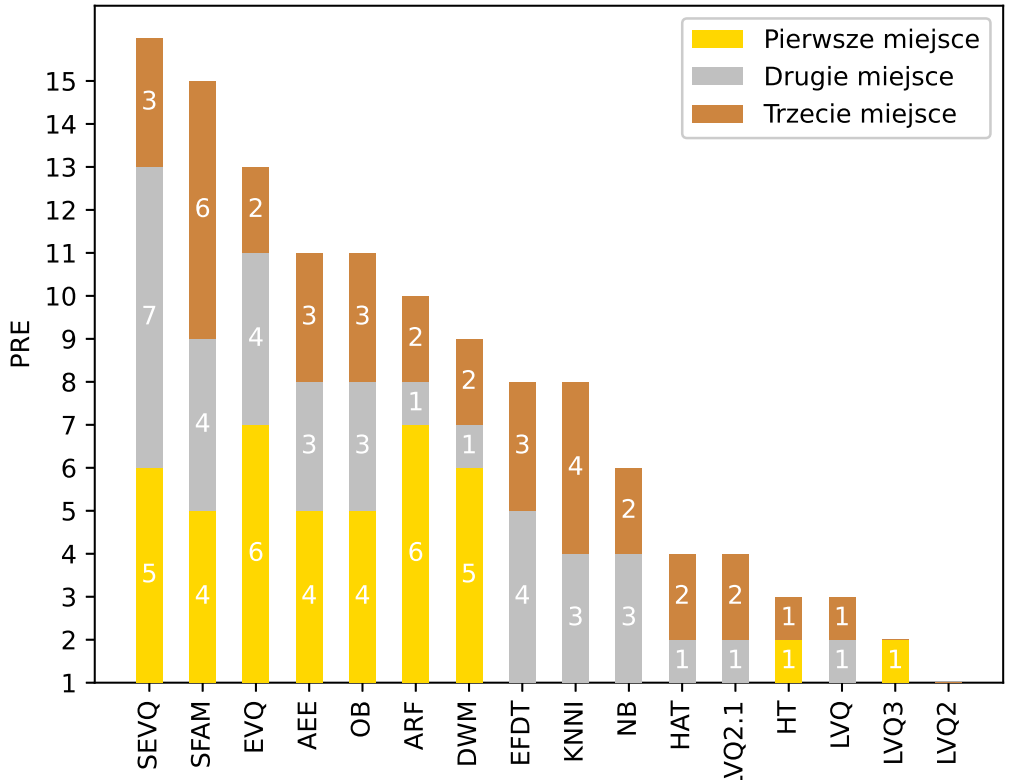
Porównanie rozkładów wartości wyników klasyfikacji przeprowadzono w dwóch etapach. W pierwszym etapie algorytm SEVQ został porównany z dziesięcioma wybranymi algorytmami tradycyjnymi. Na wykresach przedstawionych na rys. 6.11-6.15 przedstawiono rozkłady wartości metryk klasyfikacyjnych: AUC, ACC, PRE, SEN i F1.



Rysunek 6.7: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy inkrementalne osiągnęły najlepsze wyniki pod względem AUC

Analiza rys. 6.11 wskazuje na efektywność algorytmu XGB pod względem wartości AUC. Charakteryzuje się on medianą oscylującą wokół wartości 0,9 przy zachowaniu niewielkiej zmienności zaznaczonej przez najkrótsze wąsy. Można przy nim jednak zaobserwować znaczną liczbę odstających obserwacji o dużo niższych wartościach ACC. Nasuwa to wniosek, że są zbiory danych, na których algorytm radzi sobie gorzej. Algorytmy SEVQ, RF oraz MLP mają bardzo podobne mediany skupione wokół 0,85 i osiągają przeciętne wyniki z zakresu 0,75-0,80. Świadczy to o ich porównywalnej efektywności. Algorytm NC odznacza się wyraźnie najniższą medianą, umiejscowioną w okolicach 0,65, oraz wykazuje znacznie szerszy zakres zmienności, co jest podkreślone przez jego długie wąsy.

Ciekawe wyniki uzyskał algorytm QDA. Mimo braku wartości odstających charakteryzuje się wyjątkowo szerokim zakresem wyników, wskazującym na ich spore rozproszenie. Warto zauważyć, że wszystkie pozostałe algorytmy, z wyjątkiem QDA, prezentują pewien stopień wartości odstających.

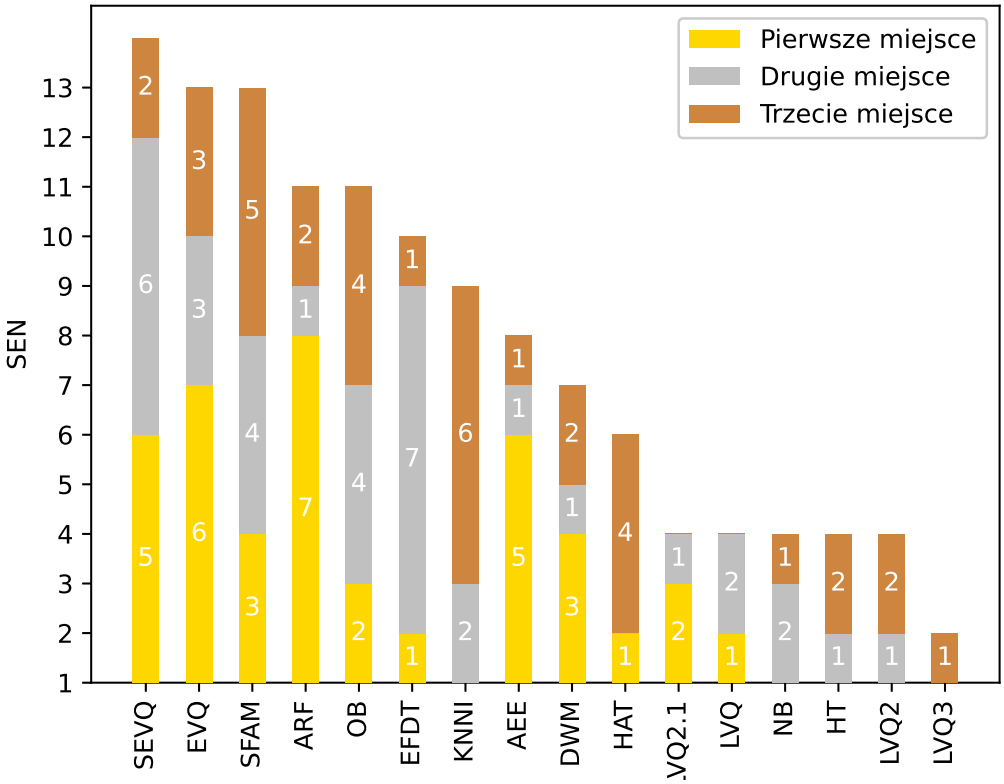


Rysunek 6.8: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy inkrementalne osiągnęły najlepsze wyniki pod względem PRE

Na rysunku 6.12 przedstawiono na wykresie pudełkowym rozkład wartości AUC dla algorytmu SEVQ oraz porównywanych z nim dziesięciu algorytmów tradycyjnych. Najwyższą medianę wykazuje algorytm XGB. Algorytmy SEVQ, GNB, KNN, MLP, RF oraz DT charakteryzują podobne mediany, wynoszące 0,8. Wartości odstające obserwuje się przy algorytmie DT, sugerując nieregularne przypadki znacznie niższej wartości AUC.

Analiza wykresu pudełkowego zamieszczonego na rys. 6.13 pokazuje, że mediana wartości PRE dla większości algorytmów mieści się w przedziale od 0,8 do 0,9, z wyjątkiem algorytmów NC oraz AB, których mediany osiągają wartość 0,75. Jednak, algorytm QDA ma również bardzo szeroki zakres wartości i najdłuższe wąsy, co sugeruje, że osiągane przez niego wartości mogą się różnić w zależności od zbioru danych.

Na rysunku 6.14 przedstawiono rozkład wartości SEN dla algorytmów z pierwszej porównywanej grupy. Analizując otrzymane wyniki zauważa się, że algorytm XGB ponownie wyróżnia najwyższa mediana oraz średnia wyników. Algorytmy SEVQ i RF



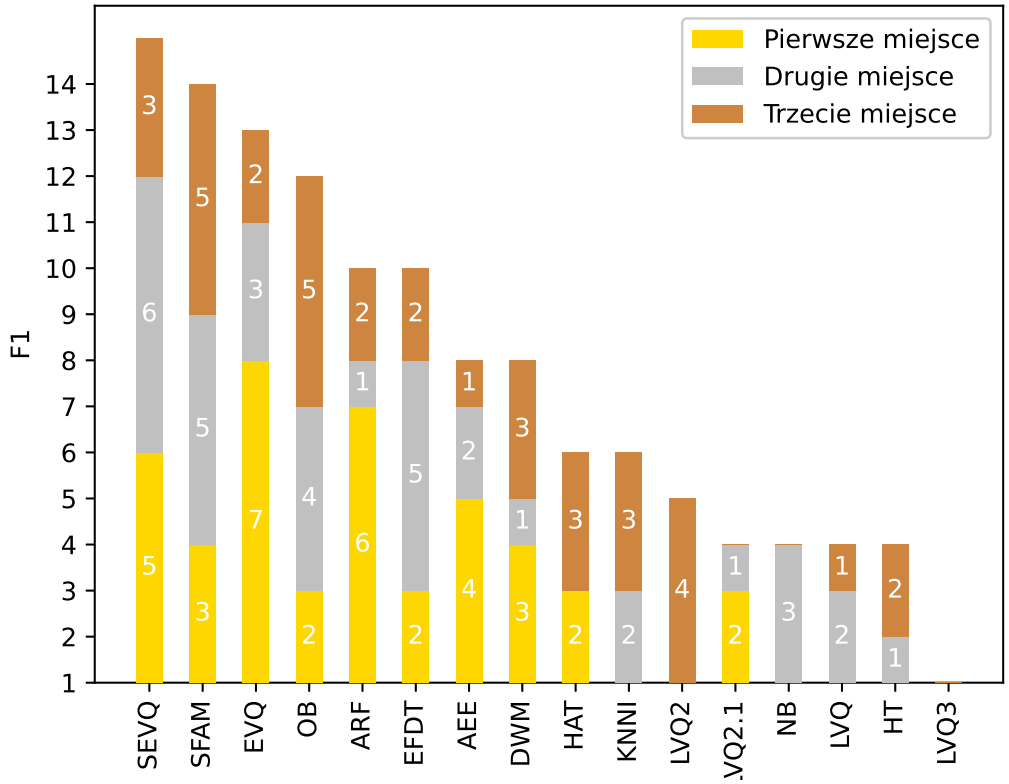
Rysunek 6.9: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy inkrementalne osiągnęły najlepsze wyniki pod względem SEN

osiągają zbliżone średnie wartości oraz mediany około 0,85, podczas gdy reszta algorytmów skupia się wokół mediany 0,75-0,83. Najniższe średnie wyniki oraz najniższą medianę osiągnął ponownie algorytm NC.

Rysunek 6.15 przedstawia rozkład wartości F1, gdzie algorytmy XGB i RF mają najwęższe zakresy kwartylowe. Pod względem mediany najlepsze wyniki osiągnął algorytm XGB, a zaraz po nim SEVQ, RF i MLP. Najniższe wyniki pod względem mediany osiągnął algorytm NC, jednak algorytmy AB i QDA miały najdłuższe wąsy, co świadczy o znaczącym rozproszeniu wyników.

6.5.2. Porównanie rozkładów wartości wskaźników jakości dla algorytmów inkrementalnych

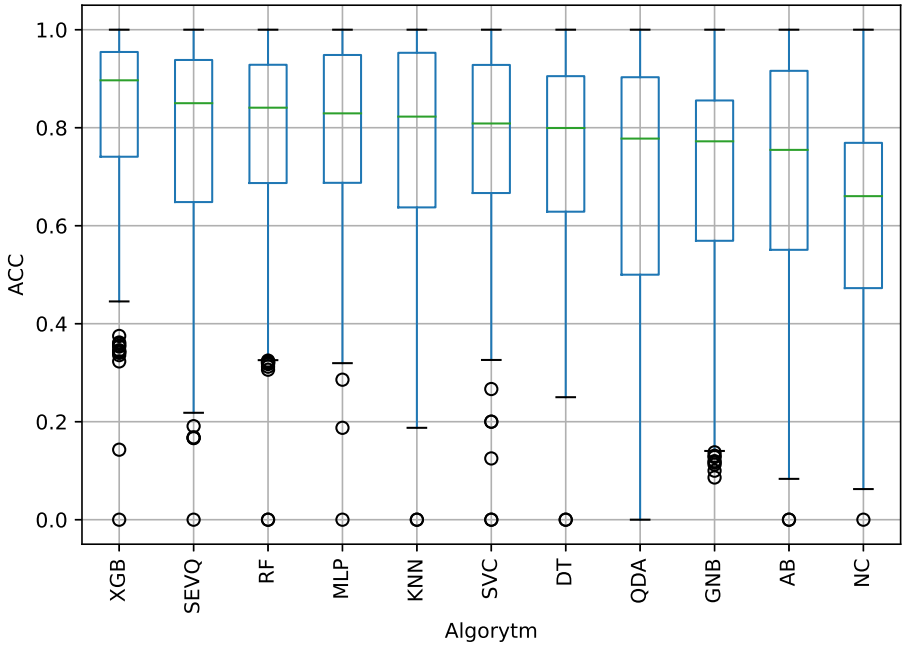
W drugim etapie algorytm SEVQ zestawiono z piętnastoma wybranymi algorytmami inkrementalnymi. Na rysunkach 6.16-6.20 przedstawiono rozkłady wartości metryk klasyfikacyjnych, tj. AUC, ACC, PRE, SEN i F1, dla poszczególnych algorytmów inkrementalnych.



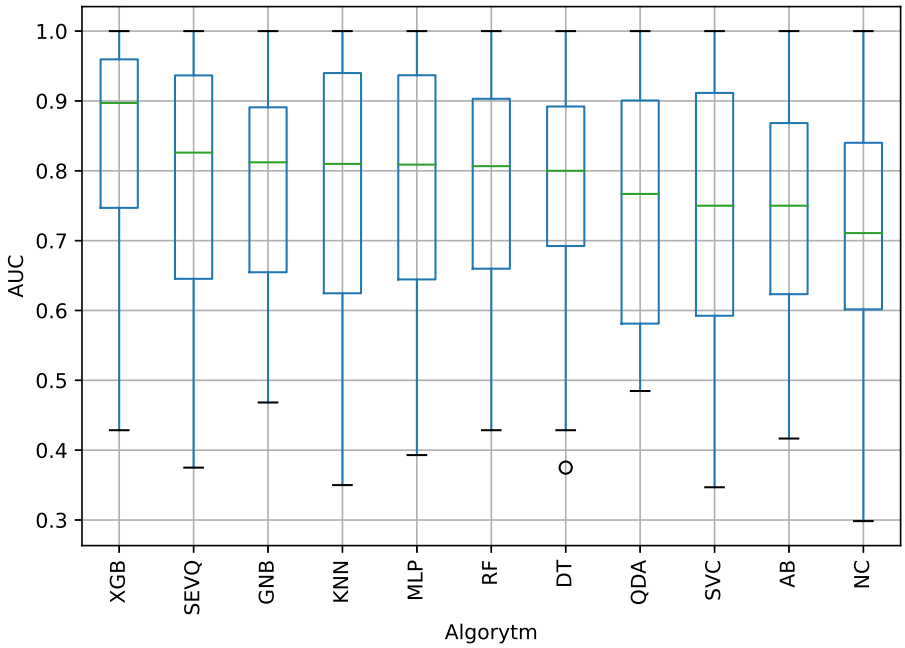
Rysunek 6.10: Zestawienie liczby zbiorów danych, na których poszczególne algorytmy inkrementalne osiągnęły najlepsze wyniki pod względem F1

Rysunek 6.16 przedstawia rozkład wartości ACC dla poszczególnych algorytmów inkrementalnych. W tej grupie algorytmów SEVQ osiągnął najlepsze średnie wyniki ACC oraz najwyższą medianę. Charakteryzował się jednak występowaniem wartości odstających. Algorytmy OB, KNNI oraz EVQ osiągnęły zbliżone wyniki średnich wartości ACC, jednak ich mediany były trochę niższe. Można zauważyć, że SEVQ zarówno pod względem średnich wartości ACC, jak i mediany wartości wyprzedza wszystkie algorytmy z rodziny kwantyzacji wektorowej, tj. EVQ, LVQ2.1, LVQ2, LVQ oraz LVQ3.

Rysunek 6.17 przedstawia wykres pudełkowy wartości AUC dla każdego z algorytmów przyrostowych posortowanych w porządku malejącym median wartości wyników. SEVQ, tak jak w przypadku ACC, znajduje się na pierwszym miejscu, a za nim kolejno algorytmy: HAT, NB, HT i SFAM. W przypadku pierwszej piątki algorytmów wartości ich median przekraczają próg 0,8. Pozostałe algorytmy uzyskały medianę wyższą niż 0,7 i niższą niż 0,8. Wyjątek stanowi algorytm LVQ3, którego mediana wynosiła około 0,65. Osiągnął on też najniższe średnie wyniki AUC i charakteryzował się



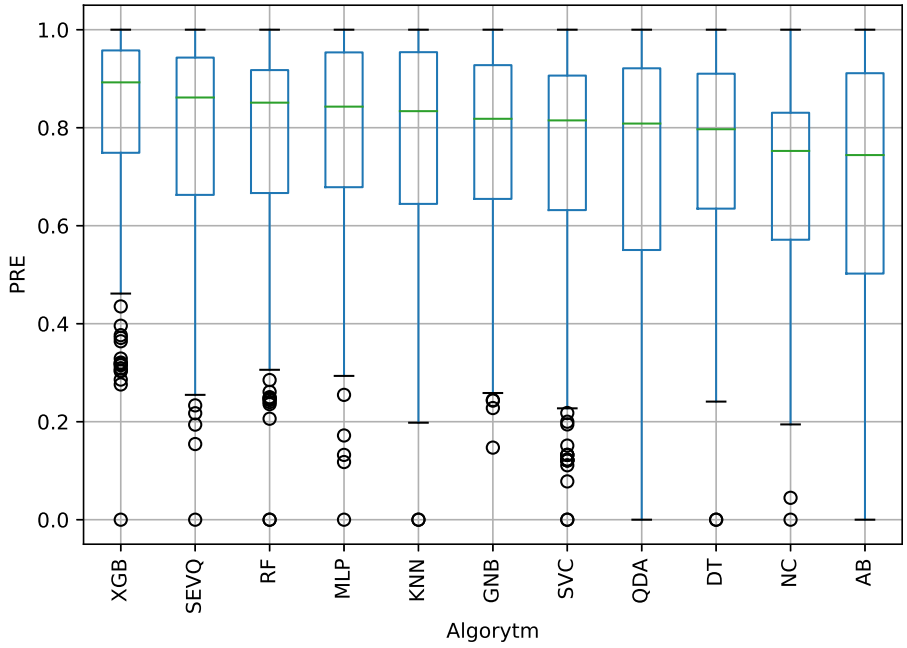
Rysunek 6.11: Rozkład wartości ACC dla poszczególnych algorytmów tradycyjnych



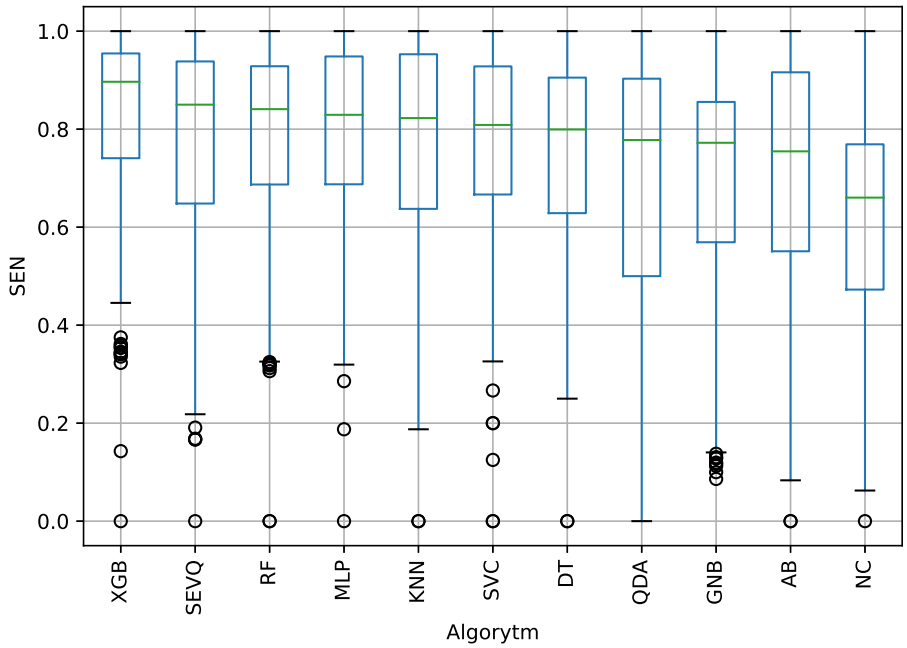
Rysunek 6.12: Rozkład wartości AUC dla poszczególnych algorytmów tradycyjnych

znacznym rozproszeniem wyników.

Analiza wykresu pudełkowego zamieszczonego na rys. 6.18 wskazuje, że mediana wartości PRE dla algorytmów SEVQ, SFAM, OB, EVQ i KNNI znacznie przekracza

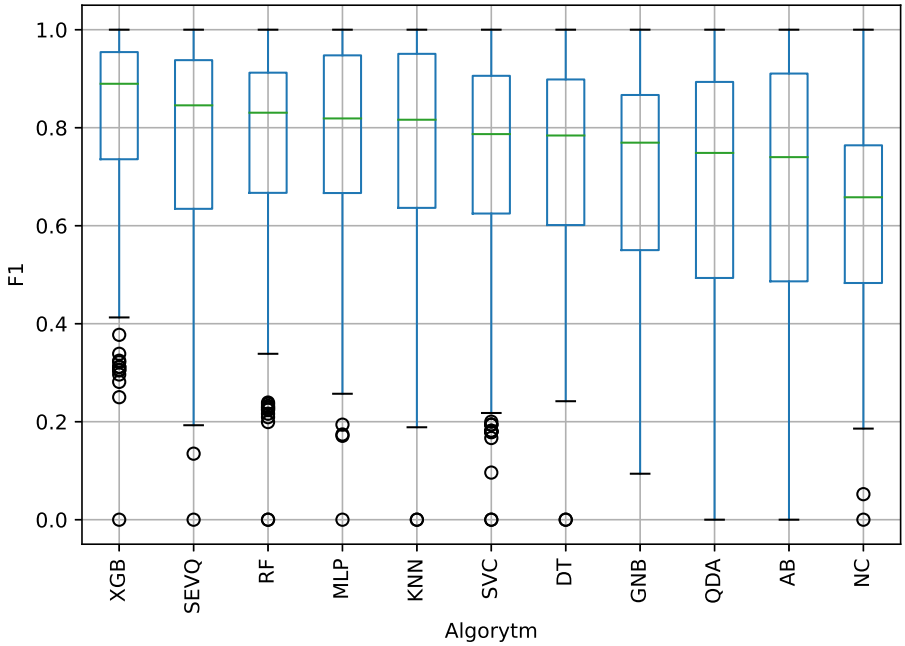


Rysunek 6.13: Rozkład wartości PRE dla poszczególnych algorytmów tradycyjnych

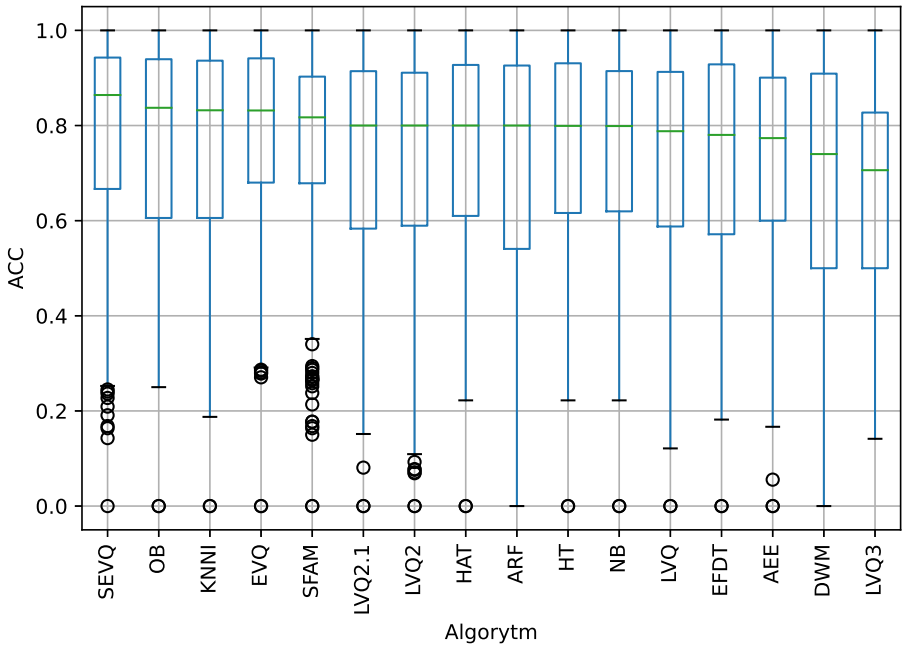


Rysunek 6.14: Rozkład wartości SEN dla poszczególnych algorytmów tradycyjnych

wartość 0,8. Najwyższą z nich osiągnął algorytm SEVQ. W przypadku pozostałych, z wyjątkiem algorytmu LVQ3, wartość mediany wynosi około 0,8. Pod względem średnich wartości PRE najwyższe wyniki osiągnęły algorytmy EVQ i SEVQ, natomiast



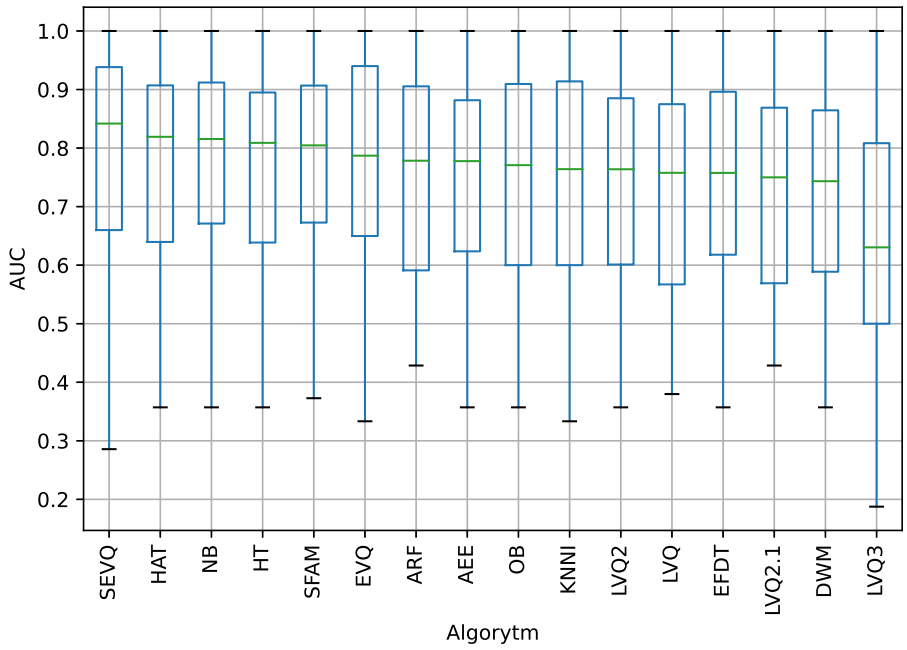
Rysunek 6.15: Rozkład wartości F1 dla poszczególnych algorytmów tradycyjnych



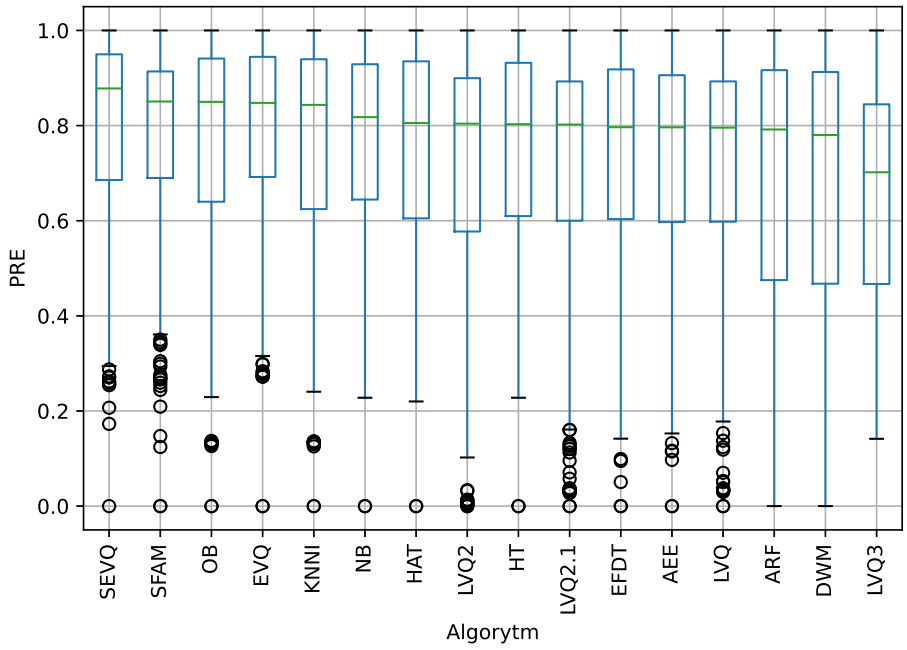
Rysunek 6.16: Rozkład wartości ACC dla algorytmów inkrementalnych

najniższe – ARF, DWM i LVQ3.

Z wykresu pudełkowego zaprezentowanego na rys. 6.19 wynika, że mediana wartości SEN dla algorytmu SEVQ przekracza próg 0,85. Mediany wartości algorytmów

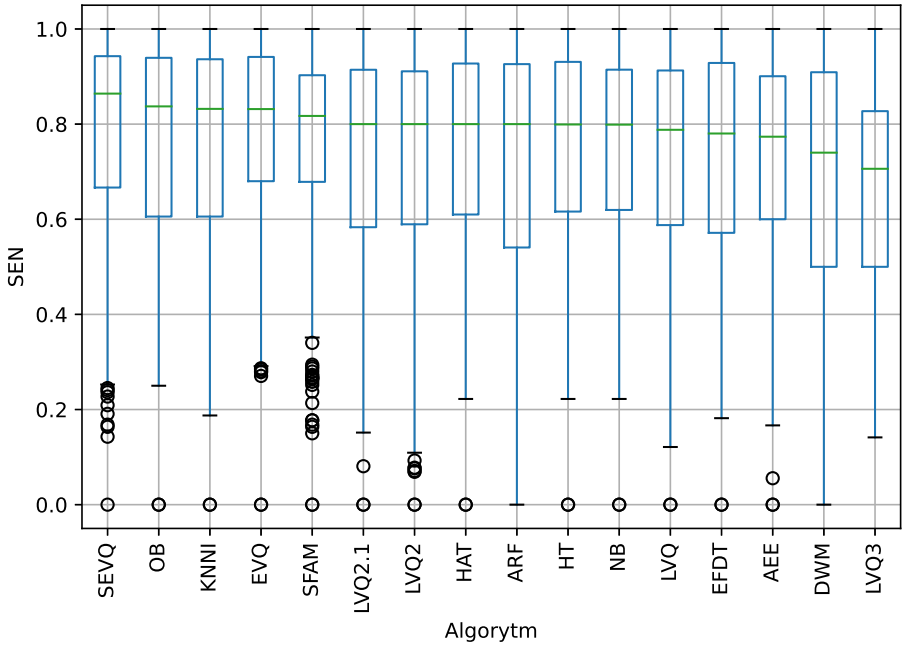


Rysunek 6.17: Rozkład wartości AUC dla algorytmów inkrementalnych



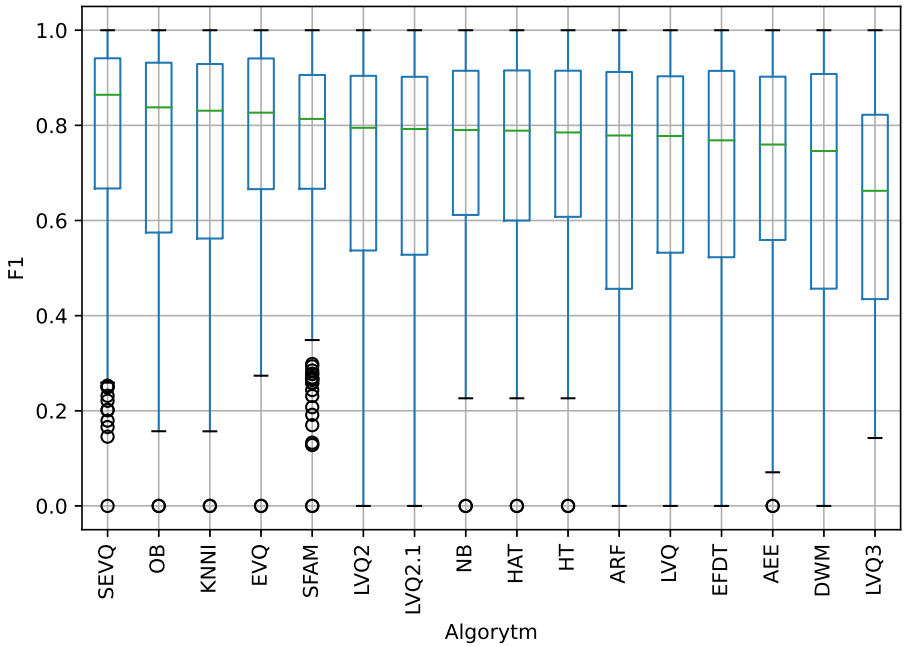
Rysunek 6.18: Rozkład wartości PRE dla algorytmów inkrementalnych

OB, KNNI, EVQ oraz SFAM znajdują się w przedziale 0,80-0,85. W przypadku algorytmów LVQ2.1, LQ2, HAT, ARF, HT i NB ich mediany oscylują dokładnie wokół wartości 0,8. Poniżej tej wartości leżą mediany algorytmów: LVQ, EFDT, AEE, DWM



Rysunek 6.19: Rozkład wartości SEN dla algorytmów inkrementalnych

oraz LVQ3. Najwyższe średnie wartości SEN osiągnęły kolejno: EVQ, SEVQ, OB, HT i HAT, natomiast najniższe średnie wartości – DWM i LVQ3.



Rysunek 6.20: Rozkład wartości F1 dla algorytmów inkrementalnych

Z wykresu pudełkowego przedstawionego na rys. 6.20 wynika, że algorytmy

OB, KNNI oraz EVQ mają podobną efektywność pod względem metryki F1, z medianami skupionymi wokół wartości 0,85. Algorytm SEVQ wyróżnia się najwyższą medianą, przekraczającą wartość 0,85. Z kolei najniższą medianę osiągnął algorytm LVQ3. Pod względem średnich wartości F1 najwyższe wyniki osiągnęły algorytmy EVQ oraz SEVQ, natomiast najniższe osiągnął algorytm LVQ3. Podczas analizy wyników zaobserwowano jednak liczne odstające obserwacje w przypadku algorytmów SEVQ oraz SFAM, które mogą nieco zaburzać interpretację tych wyników.

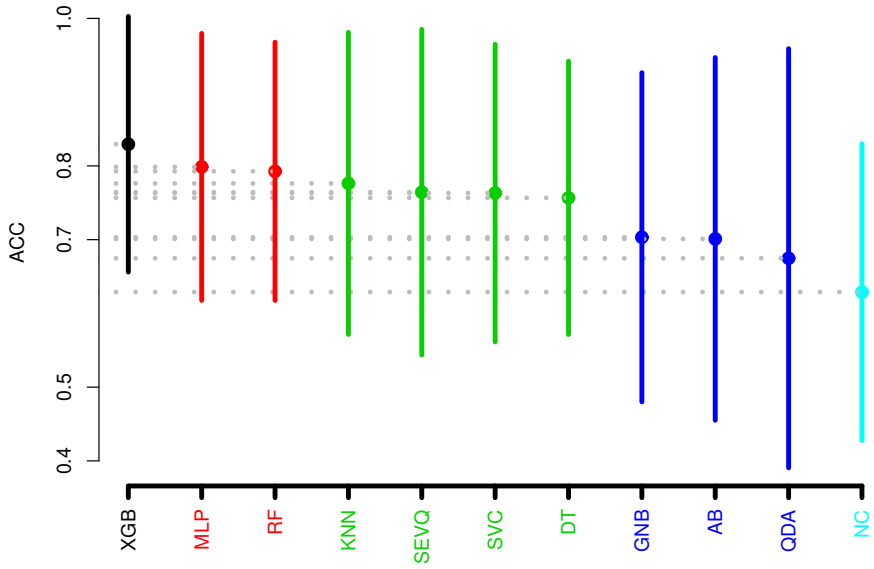
6.6. Wyniki algorytmu hierarchicznego grupowania danych Scotta–Knotta

6.6.1. Porównanie wyników testu Scotta–Knotta dla algorytmów tradycyjnych

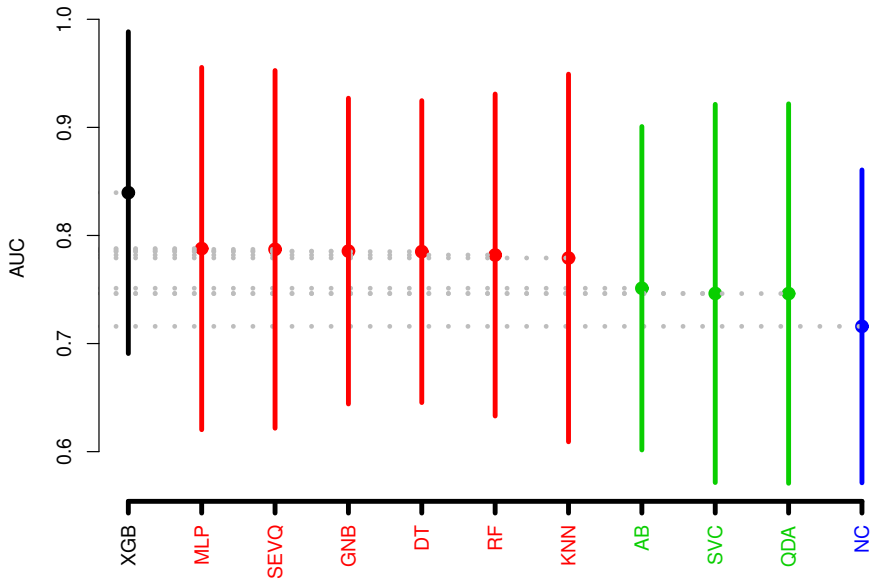
Jak zasygnalizowano w podrozdziale 4.4, algorytm Scotta–Knotta jest narzędziem statystycznym nadającym się do porównywania wyników działania różnych klasyfikatorów na wielu zbiorach danych. Polega on na sortowaniu wyników osiąganych przez klasyfikatory (ACC, PRE itd.), a następnie wyodrębnieniu istotnych statystycznie różnic między grupami klasyfikatorów. Algorytm ten pozwala na identyfikację grup klasyfikatorów, które osiągają istotnie różne wyniki w porównaniu z innymi. Porównanie wyników za pomocą testu Scotta–Knotta również miało charakter dwuetapowy – pierwszy etap polegał na zestawieniu wyników algorytmu SEVQ z tradycyjnymi algorytmami opisanymi w podrozdziale 6.2.

Na rysunkach 6.21-6.25 przedstawiono rezultaty testu Scotta–Knotta według średnich wartości metryk: AUC, ACC, PRE, SEN i F1. Porównanie SEVQ z tradycyjnymi algorytmami rozpoczęto od wykonania analizy Scotta-Knota według średnich miar ACC, a wyniki tej analizy przedstawiono na rys. 6.21. Podczas analizy wygenerowano pięć grup. W najlepszej statystycznie grupie znalazł się algorytm XGB. Do grupy drugiej zostały zaliczone algorytmy MLP oraz RF. Algorytm SEVQ znalazł się w trzeciej grupie wraz z KNN, SVC i DT. Do grupy czwartej zakwalifikowano algorytmy GNB, AB i QDA, natomiast do piątej algorytm NC.

Następnie dokonano analizy według średnich wartości metryki AUC, a jej wyniki zaprezentowano na rys. 6.22. W procesie analizy utworzono cztery odrębne grupy algorytmów. W pierwszej ponownie znalazł się algorytm XGB, wyróżniający się skutecznością i efektywnością w działaniu. Algorytm SEVQ, wraz z algorytmami MLP,



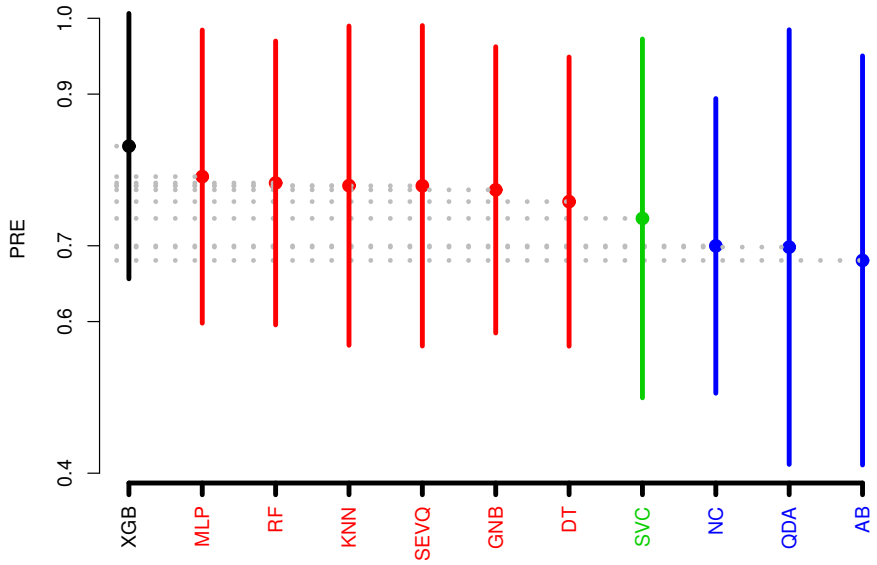
Rysunek 6.21: Wynik analizy metodą Scotta–Knotta według średnich wartości ACC dla algorytmów tradycyjnych przeprowadzonej na wszystkich zbiorach danych



Rysunek 6.22: Wynik analizy metodą Scotta–Knotta według średnich wartości AUC dla algorytmów tradycyjnych przeprowadzonej na wszystkich zbiorach danych

GNB, DT, RF i KNN, został zaklasyfikowany do drugiej grupy. W grupie trzeciej znalazły się algorytmy AB, SVC i QDA, natomiast w grupie czwartej znalazł się algorytm

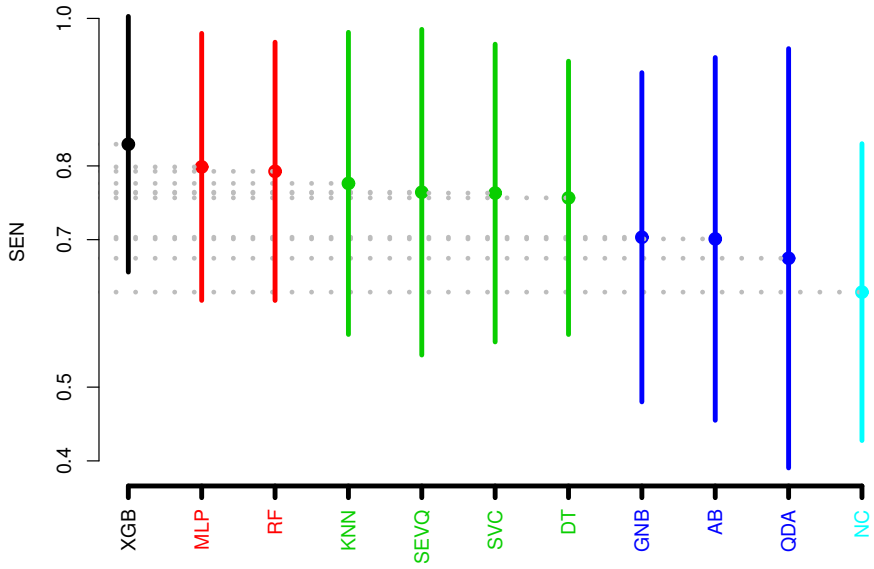
NC, co sugeruje jego najniższą skuteczność w porównaniu z resztą algorytmów.



Rysunek 6.23: Wynik analizy metodą Scotta–Knotta według średnich wartości PRE dla algorytmów tradycyjnych przeprowadzonej na wszystkich zbiorach danych

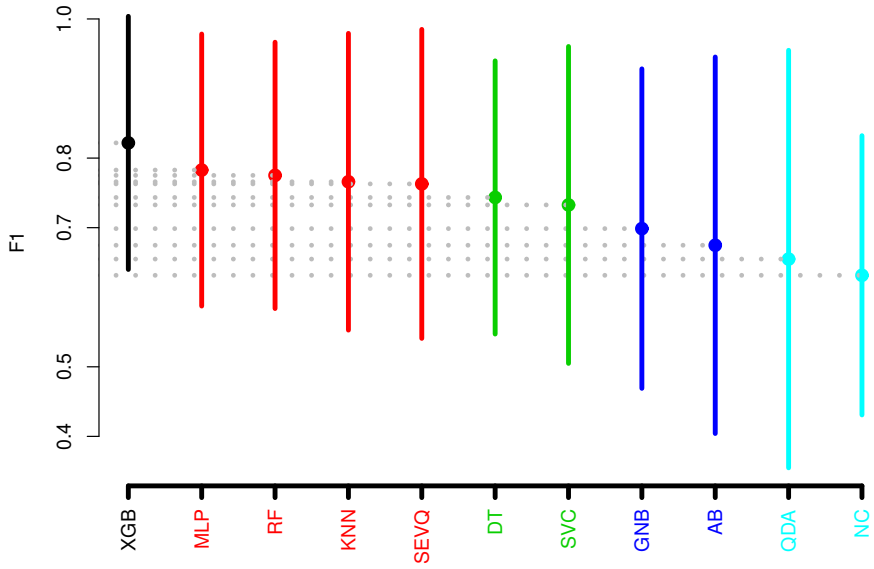
Wyniki analizy przeprowadzonej metodą Scotta–Knotta dla średnich wartości PRE zostały przedstawione na rys. 6.23. W ramach tej analizy wyodrębniono pięć głównych grup algorytmów. W pierwszej grupie, podobnie jak w przypadku metryk AUC i ACC, dominuje algorytm XGB. Drugą grupę tworzy algorytm SEVQ wraz z algorytmami MLP, RF, KNN, GNB i DT. Klasyfikacja ta wskazuje na ich solidną, choć nieco niższą od XGB, efektywność w kontekście precyzji. W trzeciej grupie znalazł się algorytm SVC, a do czwartej zaliczono algorytmy NC, QDA i AB. Ich umieszczenie w tej kategorii sygnalizuje najniższą skuteczność w porównaniu z innymi badanymi algorytmami.

Następnie przeprowadzono analizę poszczególnych algorytmów według średnich wartości SEN, a jej wyniki zilustrowano na rys. 6.24. Podczas analizy wyszczególnione zostało pięć grup algorytmów wskazujących na ich różnorodne poziomy czułości. W pierwszej grupie ponownie został wyróżniony algorytm XGB, co znów podkreśla jego skuteczność i efektywność. Druga grupa skupia w sobie algorytmy MLP i RF, które prezentują solidne, lecz nieco niższe wyniki PRE w porównaniu z algorytmem XGB. Trzecia grupa obejmuje algorytmy KNN, SEVQ, SVC i DT. Do grupy czwartej



Rysunek 6.24: Wynik analizy metodą Scotta–Knotta według średnich wartości SEN dla algorytmów tradycyjnych przeprowadzonej na wszystkich zbiorach danych

zakwalifikowano algorytmy GNB, AB oraz QDA. W grupie piątej znalazł się algorytm NC, którego pozycja na końcu rankingu sugeruje relatywnie niską czułość.



Rysunek 6.25: Wynik analizy metodą Scotta–Knotta według średnich wartości F1 dla algorytmów tradycyjnych przeprowadzonej na wszystkich zbiorach danych

Na koniec porównano SEVQ z tradycyjnymi algorytmami klasyfikacyjnymi w odniesieniu do metryki F1. Wyniki tej analizy zostały przedstawione na rys. 6.25, gdzie algorytmy zostały podzielone na pięć wyraźnych grup w zależności od osiąganych średnich wartości F1. Pierwsza grupa ponownie składała się wyłącznie z algorytmu XGB. W drugiej grupie znalazły się algorytmy MLP, RF, KNN i SEVQ, które nieco odstają od lidera, ale nadal utrzymują wysokie wyniki. Trzecia grupa, składająca się z algorytmów DT i SVC, reprezentuje średni poziom wydajności według wyników F1. Grupa czwarta zawiera algorytmy GNB i AB, natomiast piąta – algorytmy QDA oraz NC, co podkreśla ich relatywnie niższą efektywność w zakresie wyników F1.

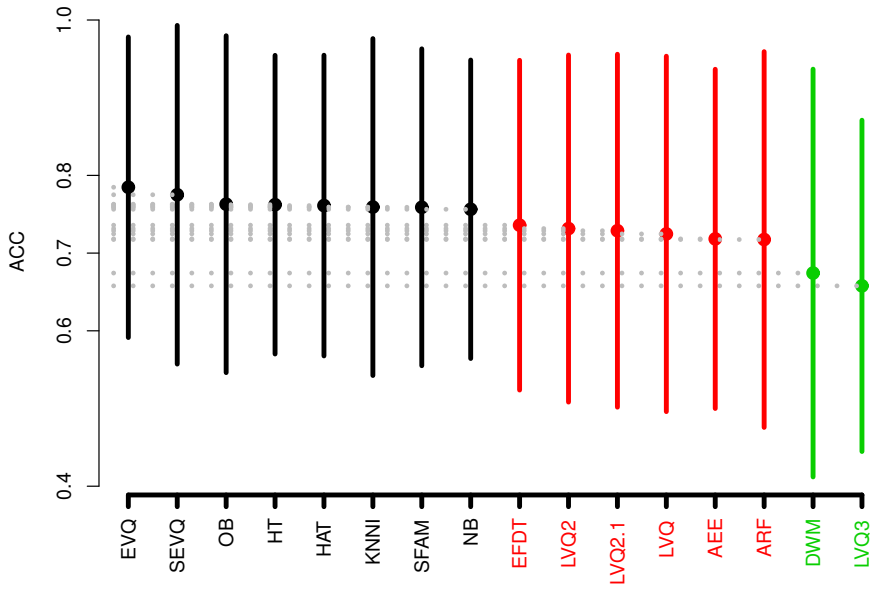
Podsumowując wyniki analiz przeprowadzonych metodą Scotta–Knotta według średnich wartości metryk wydajnościowych, należy stwierdzić, że najlepsze wyniki wśród algorytmów tradycyjnych pod względem AUC, ACC, PRE, SEN i F1 osiągnął algorytm XGB. W każdym przypadku jako jedyny kwalifikował się do pierwszej grupy. Algorytm ten wyróżnia się swoją skutecznością i efektywnością w działaniu. Algorytm SEVQ również wykazał solidną wydajność, plasując się w drugiej grupie dla metryk AUC, PRE oraz F1, oraz w trzeciej grupie dla metryk ACC i SEN. Algorytm NC wykazał się z kolei najniższą efektywnością w ramach tego porównania – był systematycznie zaliczany do ostatniej grupy w każdej z analiz.

6.6.2. Porównanie wyników testu Scotta–Knotta dla algorytmów inkrementalnych

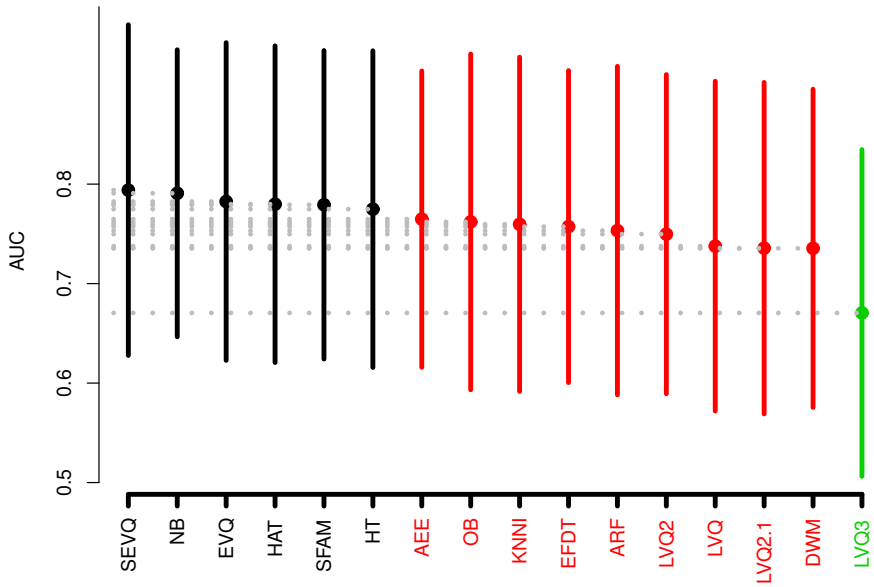
W ramach drugiego etapu analizy algorytm SEVQ porównano z piętnastoma inkrementalnymi algorytmami klasyfikacyjnymi (opisanymi w podrozdziale 6.2). Wykresy przedstawione na rys. 6.26-6.30 prezentują wyniki analiz przeprowadzonych metodą Scotta–Knotta według średnich wartości metryk wydajnościowych, takich jak AUC, ACC, PRE, SEN oraz F1 dla wspomnianych algorytmów.

W wyniku analizy algorytmy inkrementalne zostały podzielone na trzy grupy w zależności od ich średnich wartości ACC, co zostało przedstawione na rys. 6.26. Algorytm SEVQ znalazł się w pierwszej grupie cechującej się najwyższą wydajnością wraz z siedmioma innymi algorytmami: EVQ, SEVQ, SVQ, OB, HT, KNNI, SFAM oraz NB. Do drugiej grupy zaliczono: EFDT, LVQ2, LVQ2.1, LVQ, AEE oraz ARF. W ostatniej grupie znalazły się QDA i NC, które cechuje najniższa efektywność pod względem ACC.

Następnie przeprowadzono analizę według średnich wartości AUC poszczegól-



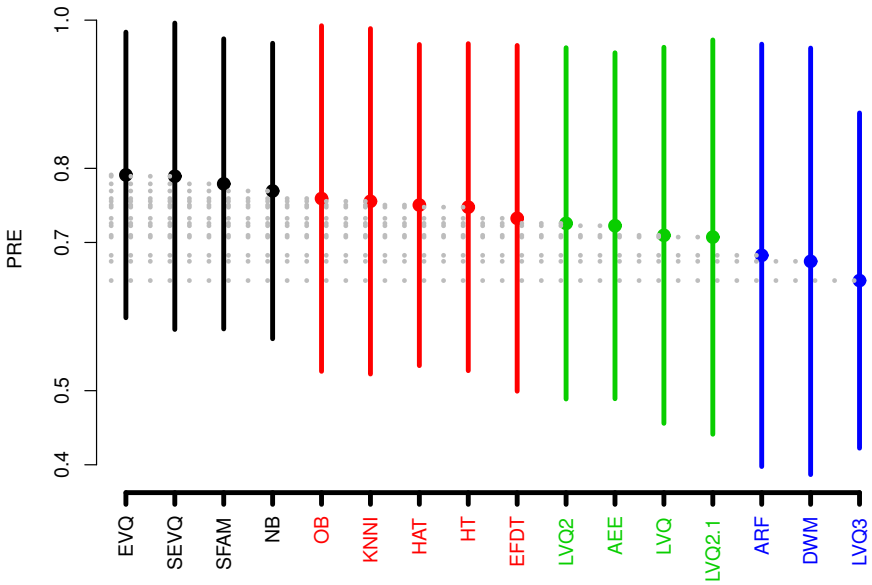
Rysunek 6.26: Wynik analizy metodą Scotta–Knotta według średnich wartości ACC dla algorytmów inkrementalnych przeprowadzonej na wszystkich zbiorach danych



Rysunek 6.27: Wynik analizy metodą Scotta–Knotta według średnich wartości AUC dla algorytmów inkrementalnych przeprowadzonej na wszystkich zbiorach danych

nych algorytmów, której wyniki zilustrowano na rys. 6.27. Podczas analizy ponownie wyszczególniono trzy grupy algorytmów. W pierwszej grupie znalazł się SEVQ, wraz

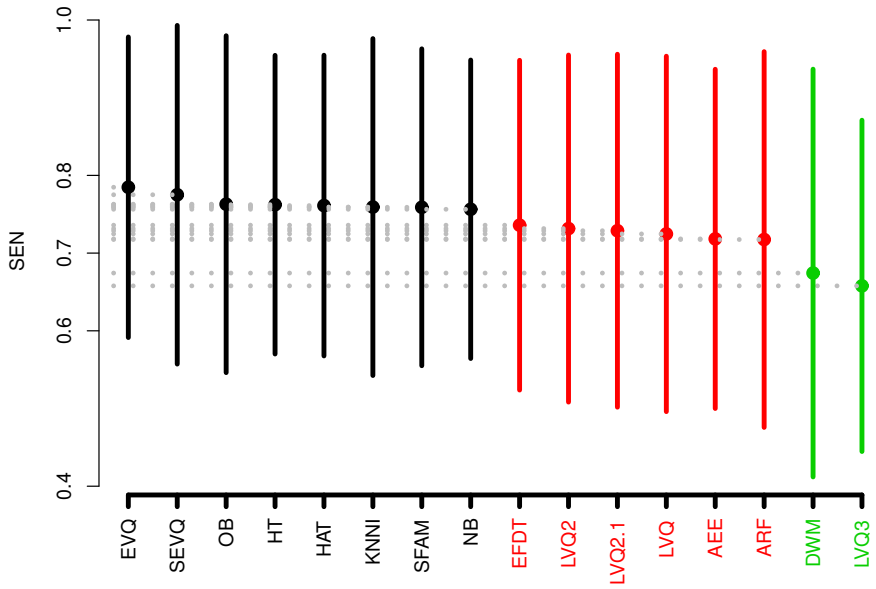
z takimi algorytmami, jak NB, EVQ, HAT, SFAM oraz HT. Do drugiej grupy zaliczyły się algorytmy: AEE, OB, KNNI, EFDT, ARF, LVQ2, LVQ, LVQ2.1 i DWM. W ostatniej grupie znalazł się algorytm LVQ3.



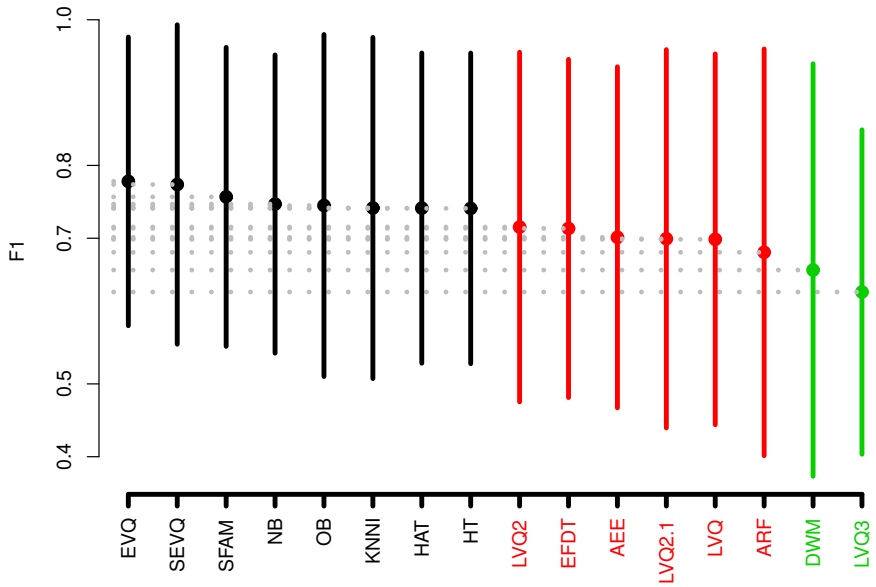
Rysunek 6.28: Wynik analizy metodą Scotta–Knotta według średnich wartości PRE dla algorytmów inkrementalnych przeprowadzonej na wszystkich zbiorach danych

Wyniki analizy według średnich wartości PRE przedstawiono na rys. 6.28. W wyniku analizy algorytmy klasyfikacyjne zostały podzielone na pięć grup w zależności od ich zdolności do precyzyjnego klasyfikowania wyników. W pierwszej znalazły się cztery algorytmy: EVQ, SEVQ, SFAM i NB. Do drugiej grupy zaklasyfikowano OB, KNNI, HAT, HT oraz EFDT. W grupie trzeciej znalazły się algorytmy: LVQ2, AEE, LVQ i LVQ2.1. Do ostatniej grupy zakwalifikowano trzy algorytmy: ARF, DWM i LVQ3.

Wyniki analizy przeprowadzonej metodą Scotta–Knotta dla średnich wartości SEN zostały przedstawione na rys. 6.29. W ramach analizy wyszczególniono trzy grupy algorytmów. W pierwszej grupie, demonstrującej najwyższą czułość, znalazł się algorytm SEVQ wraz z algorytmami: EVQ, OB, HT, HAT, KNNI, SFAM i NB. Ta grupa, już wcześniej wyróżniająca się w analizie dotyczącej ACC, potwierdza swoją skuteczność i uniwersalność. Do drugiej grupy zaliczono: EFDT, LVQ2, LVQ2.1, LVQ, AEE oraz ARF. W trzeciej grupie umieszczono DWM i LVQ3, które prezentują najniższą średnią wartość czułości w porównaniu z pozostałymi grupami.



Rysunek 6.29: Wynik analizy metodą Scotta–Knotta według średnich wartości SEN dla algorytmów inkrementalnych przeprowadzonej na wszystkich zbiorach danych



Rysunek 6.30: Wynik analizy metodą Scotta–Knotta według średnich wartości F1 dla algorytmów inkrementalnych przeprowadzonej na wszystkich zbiorach danych

Ostatnie porównanie SEVQ z tradycyjnymi algorytmami inkrementalnymi odnosiło się do F1. Wyniki tej analizy zostały przedstawione na rys. 6.30. Warto zauważyć,

że grupy zostały wydzielone w taki sam sposób jak w przypadku SEN. Pierwsza grupa ponownie składała się z algorytmów: EVQ, SEVQ, SFAM, NB, OB, KNNI, HAT oraz HT. W drugiej grupie, tak samo jak w przypadku SEN, znalazły się: LVQ2, EFDT, AEE, LVQ2.1, LVQ oraz ARF, natomiast do trzeciej grupy zaliczono DWM i LVQ3.

6.7. Wyniki testu Wilcoxona dla par obserwacji

6.7.1. Sposób przeprowadzenia testu

Jak scharakteryzowane w podrozdziale 4.5, warto zastosować test Wilcoxona dla par obserwacji, ponieważ jest to test nieparametryczny, który nie wymaga założenia normalności rozkładu danych, co jest korzystne w przypadku różnych klasyfikatorów i zbiorów danych. Test ten umożliwia porównanie wyników klasyfikatorów w sposób parowy, oceniając statystyczną istotność różnic między nimi.

W przeprowadzonym porównaniu rozważane są następujące przypadki: SEVQ w porównaniu z tradycyjnymi algorytmami pod względem ACC, AUC, PRE, SEN i F1 (omówiony w podrozdziale 6.7.2) oraz SEVQ w porównaniu z algorytmami inkrementalnymi pod względem ACC, AUC, PRE, SEN i F1 (omówiony w podrozdziale 6.7.3).

Przez $\mathbf{X}$ oznaczmy wektor zawierający elementy będące średnimi wartościami ACC (lub odpowiednio AUC, PRE, SEN i F1) dla algorytmu testowanego na dziesięciu losowych częściach każdego zbioru danych. Przez $\mathbf{Y}_i$ oznaczmy wektor zawierający odpowiednie wartości dla i-tego algorytmu testowanego na tych samych częściach zbiorów. Indeks i oznacza nazwę algorytmu, tj. i należy do zestawu {AB, DT, GNB, KNN, MLP, NC, QDA, RF, SVC, XGB} w przypadku analizy algorytmów tradycyjnych lub i należy do zestawu {AEE, ARF, DWM, EFDT, EVQ, HAT, HT, KNNI, LVQ, LVQ2, LVQ2.1, LVQ3, NB, OB, SFAM}, jeśli analizowane są algorytmy inkrementalne.

6.7.2. Porównanie SEVQ z klasyfikatorami tradycyjnymi

Porównanie wydajności algorytmu SEVQ z algorytmami tradycyjnymi opiera się na średnich wartościach AUC, ACC, PRE, SEN oraz F1, które przedstawiono w tab. 6.2. Tabela 6.4 zawiera wyniki testu Wilcoxona dla hipotezy zerowej H_0, która zakłada brak statystycznie istotnych różnic $(X - Y_i)$ między średnimi wartościami ACC (lub kolejno AUC, PRE, SEN i F1) obliczonymi dla algorytmu SEVQ i kolejno porównywanych algorytmów tradycyjnych. Alternatywna hipoteza H_1 zakłada istnienie takich różnic.

Tabela 6.4: Wyniki testu Wilcoxona użytego do porównania algorytmu SEVQ oraz klasyfikatorów tradycyjnych

	Algorytm	**Wartość p dla ACC**	**Wartość p AUC**	**Wartość p PRE**	**Wartość p dla SEN**	**Wartość p F1**
1	AB	**0,0563**	0,0144	0,0012	**0,0563**	0,0062
2	DT	**0,9889**	**0,5877**	**0,2937**	**0,9889**	**0,5192**
3	GNB	0,0001	**0,5384**	**0,5690**	0,0001	0,0000
4	KNN	0,0002	**0,6423**	0,2622	0,0002	0,0114
5	MLP	0,0000	**0,3201**	0,0438	0,0000	0,0013
6	NC	0,0000	0,0000	0,0000	0,0000	0,0000
7	QDA	0,0008	0,0012	0,0004	0,0008	0,0000
8	RF	0,0001	**0,2342**	**0,6937**	0,0001	**0,0964**
9	SVC	**0,2323**	0,0000	0,0004	**0,2323**	0,0382
10	XGB	0,0000	0,0000	0,0000	0,0000	0,0000

W kontekście analizy średnich wartości ACC wyniki testu Wilcoxona zamieszczone w tab. 6.4 nie wykazały istotnych statystycznie różnic w średnich wartościach ACC pomiędzy algorytmem SEVQ a algorytmami AB, DT i SVC na poziomie istotności $\alpha = 0,05$. Oznacza to, że nie udało się odrzucić hipotezy zerowej, zakładającej brak znaczącej różnicy między porównywanymi algorytmami. Jednocześnie przy porównaniu SEVQ z algorytmami GNB, KNN, MLP, NC, QDA, RF i XGB test Wilcoxona umożliwił odrzucenie hipotezy zerowej na poziomie istotności 5%, co wskazuje na istnienie znaczących różnic w średnich wartościach ACC. W związku z tym przyjęto alternatywną hipotezę H_1, zakładającą znaczącą różnicę w średnich wartościach ACC pomiędzy SEVQ a wymienionymi algorytmami. Dodatkowa analiza wyników testu rang Wilcoxona oraz rozkładu wartości ACC dla poszczególnych tradycyjnych algorytmów, przedstawionych na na rys. 6.11, wskazuje na niższą skuteczność algorytmu SEVQ w porównaniu z algorytmem XGB na przyjętym poziomie istotności $\alpha = 0,05$. Jednocześnie, SEVQ wykazuje lepsze wyniki niż algorytmy RF, MLP, KNN, QDA, GNB i NC. Zgodnie z rankingiem Scotta–Knotta, prezentowanym na rys. 6.21, potwierdza się umiejscowienie SEVQ w trzeciej grupie najlepszych tradycyjnych algorytmów, zaraz za algorytmem KNN, i wykazuje lepszą skuteczność niż algorytmy GNB, QDA i NC, co jest zgodne z analizą pod względem metryki ACC.

W przypadku analizy średnich wartości AUC wyniki testu Wilcoxona zaprezentowane w tab. 6.4, nie wykazują istotnych statystycznie różnic w średnich wartościach AUC pomiędzy algorytmem SEVQ a algorytmami DT, GNB, KNN, MLP i RF przy przyjętym poziomie istotności $\alpha = 0,05$. Oznacza to, że nie ma wystarczających dowodów na odrzucenie hipotezy zerowej, sugerującej brak znaczącej różnicy w efektywności

tych algorytmów pod względem metryki AUC. Ponadto, analiza wartości p dla AUC oraz rozkład wartości AUC dla poszczególnych tradycyjnych algorytmów przedstawiona na rys. 6.12 pozwala na sformułowanie wniosku, że algorytm SEVQ jest mniej efektywny niż XGB, lecz wykazuje lepsze wyniki niż algorytmy QDA, AB, SVC i NC. Wyniki testu Wilcoxona potwierdzają na przyjętym poziomie istotności $\alpha = 0,05$ wyniki rankingu Scotta–Knotta przedstawione na rys. 6.22, zgodnie z którymi SEVQ jest sklasyfikowany jako mniej efektywny niż XGB, lecz wydajniejszy od algorytmów QDA, AB, SVC i NC w kontekście metryki AUC.

Przy rozpatrywaniu wyników testu Wilcoxona (tab. 6.4) pod względem metryki PRE ponownie nie udało się odrzucić hipotezy zerowej, ponieważ nie ma istotnej różnicy w średniej wartości PRE na poziomie istotności $\alpha = 0,05$ między SEVQ a trzema algorytmami: DT, GNB i RF. Jednocześnie przy porównaniu SEVQ z algorytmami AB, KNN, MLP, NC, QDA, SVC i XGB test Wilcoxona umożliwił przyjęcie alternatywnej hipotezy H_1, zakładającej znaczącą różnicę w średnich wartościach ACC pomiędzy SEVQ a wymienionymi algorytmami. Dalsze badania wartości p w kontekście PRE oraz dystrybucji AUC dla różnych tradycyjnych algorytmów przedstawiono na rys. 6.13. Doprowadziły one do stwierdzenia, że algorytm SEVQ jest mniej efektywny niż XGB, ale bardziej efektywny niż MLP, KNN, SVC, QDA, NC i AB. Wyniki testu Wilcoxona potwierdzają na przyjętym poziomie istotności $\alpha = 0,05$ wyniki rankingu Scotta–Knotta przedstawione na rys. 6.23, ponieważ SEVQ został umieszczony w drugiej grupie najlepszych tradycyjnych algorytmów i wykazuje lepszą skuteczność niż algorytmy NC, QDA i AB.

W przypadku kryterium SEN analiza wyników (tab. 6.4) także nie pozwoliła na odrzucenie hipotezy zerowej, zakładającej brak statystycznie istotnych różnic między algorytmem SEVQ a innymi algorytmami. Wyniki wskazują na brak istotnej różnicy w średnich wartościach SEN na ustalonym poziomie istotności $\alpha = 0,05$ pomiędzy SEVQ a trzema algorytmami: AB, DT i SVC. Dalsze rozważania wyników testu rang Wilcoxona i rozkładu wartości SEN wśród tradycyjnych algorytmów (rys. 6.14) prowadzą do konkluzji, że algorytm SEVQ nie jest tak wydajny jak XGB, ale osiąga lepsze rezultaty niż algorytmy RF, MLP, KNN, QDA, GNB oraz NC. Potwierdzenia wyników testu Wilcoxona na przyjętym poziomie istotności $\alpha = 0,05$ dostarczają także wyniki testu Scotta–Knotta (rys. 6.22). W świetle tych danych SEVQ klasyfikuje się poniżej XGB, MLP i RF, jednak przewyższa algorytmy GNB, QDA oraz NC.

Z kolei dla metryki F1 wyniki z tab. 6.4 wskazują, że test nie zdołał odrzucić hipotezy zerowej, gdyż nie ma istotnej różnicy w średniej wartości F1 na poziomie istotności $\alpha = 0,05$ w przypadku SEVQ i dwóch algorytmów: DT oraz RF. Otrzymany rezultat jest również spójny z wynikiem analizy Scotta–Knotta dla miary F1 (rys. 6.25), który ulokował SEVQ w drugiej grupie, wraz z algorytmem RF, a DT w kolejnej (trzeciej) grupie najlepiej działających algorytmów w ocenie metryki F1. Wyniki analizy testu Wilcoxona dla miary F1 wskazują, że można odrzucić hipotezę zerową na poziomie 5%, gdy porówna się algorytm SEVQ z AB, GNB, KNN, MLP, NC, QDA, SVC i XGB. Tym samym odrzucając hipotezę zerową, można przyjąć hipotezę alternatywną H_1, która zakłada, że istnieje istotna różnica w średnich wartościach F1 dla SEVQ w porównaniu z pozostałymi algorytmami (AB, GNB, KNN, MLP, NC, QDA, SVC i XGB). Jest to również w dużej mierze zbieżne z wynikami analizy Scotta–Knotta względem F1 (rys. 6.25, podczas której algorytm SVC został zakwalifikowany do grupy trzeciej, GNB i AB do czwartej, natomiast QDA i NC do grupy piątej.

Porównanie wydajności algorytmu SEVQ z innymi algorytmami tradycyjnymi na podstawie analizy wyników testu Wilcoxona pod względem średnich wartości metryk wydajnościowych, takich jak AUC, ACC, PRE, SEN oraz F1, dostarcza kompleksowego spojrzenia na efektywność algorytmu. Wyniki testu Wilcoxona wskazują na brak statystycznie istotnych różnic pomiędzy SEVQ a niektórymi algorytmami (AB, DT, SVC) pod względem ACC, co sugeruje porównywalną wydajność pod względem dokładności wyników. W przypadku AUC, PRE i F1 algorytm SEVQ wykazuje lepszą wydajność w porównaniu z niektórymi algorytmami, podczas gdy w kontekście SEN wyniki są porównywalne z wybranymi algorytmami. Biorąc pod uwagę wyniki analizy Scotta–Knotta oraz rozkładu wartości metryk na wykresach pudełkowych, należy stwierdzić, że SEVQ klasyfikuje się poniżej XGB, ale przewyższa wiele tradycyjnych metod.

6.7.3. Porównanie SEVQ z klasyfikatorami inkrementalnymi

W ramach porównania efektywności algorytmu SEVQ z algorytmami inkrementalnymi przeprowadzono szczegółową analizę średnich wartości AUC, ACC, PRE, SEN oraz F1, które przedstawiono w tab. 6.3. Tabela 6.5 zawiera wyniki testu Wilcoxona dla hipotezy zerowej H_0, która zakłada brak statystycznie istotnych różnic $(X - Y_i)$ między średnimi wartościami ACC (lub kolejno: AUC, PRE, SEN i F1) obliczonymi dla algorytmu SEVQ i kolejno porównywanych algorytmów inkrementalnych. Alternatywna hipoteza H_1 zakłada odwrotnie.

Tabela 6.5: Wyniki testu Wilcoxona użytego do porównania algorytmu SEVQ oraz klasyfikatorów inkrementalnych

	Algorytm	**Wartość p dla ACC**	**Wartość p AUC**	**Wartość p PRE**	**Wartość p dla SEN**	**Wartość p F1**
1	AEE	0,0001	0,0013	0,0001	0,0001	0,0000
2	ARF	0,0374	0,0001	0,0000	0,0374	0,0000
3	DWM	0,0000	0,0000	0,0000	0,0000	0,0000
4	EFDT	0,0199	0,0000	0,0000	0,0199	0,0001
5	EVQ	**0,3275**	0,0209	**0,4649**	**0,3275**	**0,8284**
6	HAT	**0,1737**	**0,0587**	0,0023	**0,1737**	0,0061
7	HT	**0,2214**	0,0120	0,0012	**0,2214**	0,0050
8	KNNI	**0,9419**	0,0003	**0,0786**	**0,9419**	**0,1796**
9	LVQ	0,0000	0,0000	0,0000	0,0000	0,0000
10	LVQ2	0,0001	0,0000	0,0000	0,0001	0,0000
11	LVQ2.1	0,0000	0,0000	0,0000	0,0000	0,0000
12	LVQ3	0,0000	0,0000	0,0000	0,0000	0,0000
13	NB	0,0189	**0,5311**	**0,1056**	0,0189	0,0079
14	OB	**0,3693**	0,0025	**0,4098**	**0,3693**	**0,7303**
15	SFAM	0,0000	0,0000	0,0003	0,0000	0,0000

Zastosowanie testu Wilcoxona nie wykazało statystycznie istotnych różnic w średnich wartościach ACC między SEVQ a algorytmami inkrementalnymi, takimi jak EVQ, HAT, HT, KNNI i OB. Sugeruje to równoważność wydajności tych metod na poziomie istotności $\alpha = 0,05$. Wartości p dla metryki ACC zamieszczone w tab. 6.5 i rozkład wartości ACC dla algorytmów inkrementalnych przedstawiony na rys. 6.16 prowadzą do stwierdzenia, że SEVQ jest lepszy na poziomie istotności $\alpha = 0,05$ w stosunku do algorytmów: SFAM, LVQ2.1, LVQ2, ARF, NB, LVQ, EFDT, AEE, DWM i LVQ3. Wyniki testu Wilcoxona potwierdzają na tym samym poziomie istotności wynik testu Scotta–Knotta przedstawiony na rys. 6.26, tj. SEVQ przewyższa algorytmy: EFDT, LVQ2, LVQ2.1, LVQ, AEE, ARF, DWM i LVQ3 pod względem średnich wartości ACC.

W zakresie średnich wartości AUC przeprowadzona analiza wyników testu Wilcoxona zamieszczonych w tab. 6.5 nie wykazała istotnych statystycznie różnic w średnich wartościach AUC pomiędzy algorytmem SEVQ a algorytmami HAT oraz NB na poziomie istotności $\alpha = 0,05$. Oznacza to, że nie można odrzucić hipotezy zerowej, zakładającej brak znaczącej różnicy między SEVQ a porównywanymi algorytmami inkrementalnymi. Jednocześnie przy porównaniu SEVQ z algorytmami: AEE, ARF, DWM, EFDT, EVQ, HT, KNNI, LVQ, LVQ2, LVQ2.1, LVQ3, OB i SFAM, test Wilcoxona umożliwił odrzucenie hipotezy zerowej na poziomie istotności 5%, co wskazuje na istnienie znaczących różnic w średnich wartościach AUC. W związku z tym przyjęto alternatywną hipotezę H_1, zakładającą znaczącą różnicę w średnich wartościach AUC

pomiędzy SEVQ a wymienionymi algorytmami. Ponadto, szczegółowa ocena wartości p w kontekście AUC oraz analiza dystrybucji AUC dla różnych algorytmów inkrementalnych (rys. 6.17), prowadzą do wniosku, że algorytm SEVQ przewyższa algorytmy: HT, SFAM, EVQ, ARF, AEE, OB, KNNI, LVQ2, LVQ, EFDT, LVQ2.1, DWM i LVQ3. Wyniki testu Wilcoxona na przyjętym poziomie istotności $\alpha = 0,05$ zyskują potwierdzenie także w wynikach testu Scotta–Knotta (rys. 6.27), ponieważ SEVQ został zakwalifikowany w tej samej grupie co EVQ, SFAM oraz HT i przewyższa AEE, OB, KNNI, EFDT, ARF, LVQ2, LVQ, LVQ2.1, DWM i LVQ3.

W kontekście PRE przeprowadzona analiza wyników z tab. 6.4 także nie pozwoliła na odrzucenie hipotezy zerowej, sugerującej brak znaczących statystycznie różnic między algorytmem SEVQ a pozostałymi metodami. Analiza wykazała, że nie istnieje statystycznie istotna różnica w średnich wartościach PRE między SEVQ a algorytmami EVQ, KNNI, NB oraz OB przy poziomie istotności $\alpha = 0,05$. Ponadto, na podstawie wyników testu rang Wilcoxona oraz rozkładu wartości PRE dla algorytmów inkrementalnych (rys. 6.18) można stwierdzić, że SEVQ przewyższa algorytmy SFAM, HAT, LVQ2, HT, LVQ2.1, EFDT, AEE, LVQ, ARF, DWM i LVQ3. Potwierdzenie tych obserwacji na przyjętym poziomie istotności $\alpha = 0,05$ zapewniają również wyniki analizy metodą Scotta–Knotta, przedstawione na rys. 6.28. Na podstawie tych wyników algorytm SEVQ znalazł się w tej samej grupie co SFAM, lecz wykazuje wyższą skuteczność w porównaniu z metodami: HAT, HT, EFDT, LVQ2, AEE, LVQ, LVQ2.1, ARF, DWM i LVQ3.

Biorąc pod uwagę wyniki testu Wilcoxona (tab. 6.5) pod względem metryki SEN nie można odrzucić hipotezy zerowej, ponieważ nie ma istotnej różnicy w średniej wartości SEN na poziomie istotności $\alpha = 0,05$ między SEVQ a pięcioma algorytmami: EVQ, HAT, HT, KNNI i OB. Jednocześnie przy porównaniu SEVQ z algorytmami AEE, ARF, DWM, EFDT, LVQ, LVQ2, LVQ2.1, LVQ3, NB i SFAM test Wilcoxona umożliwił przyjęcie alternatywnej hipotezy H_1, zakładającej znaczącą różnicę w średnich wartościach SEN pomiędzy SEVQ a wymienionymi algorytmami. Dalsze badania wartości p w kontekście SEN oraz dystrybucji SEN dla różnych algorytmów inkrementalnych (rys. 6.18) prowadzą do stwierdzenia, że algorytm SEVQ przewyższa algorytmy SFAM, LVQ2.1, LVQ2, ARF, NB, LVQ, EFDT, AEE, DWM i LVQ3. Wyniki testu Wilcoxona na przyjętym poziomie istotności $\alpha = 0,05$ potwierdzają wyniki rankingu Scotta–Knotta przedstawione na rys. 6.28, ponieważ SEVQ został umieszczony

w pierwszej grupie najlepszych algorytmów inkrementalnych wraz z algorytmami SFAM i NB, wykazując lepszą skuteczność niż algorytmy EFDT, LVQ2, AEE, LVQ, LVQ2.1, ARF, DWM i LVQ3.

W kontekście metryki F1 (tab. 6.5) wyniki testu Wilcoxona wskazują, że test również nie zdołał odrzucić hipotezy zerowej, gdyż nie ma istotnej różnicy w średniej wartości F1 na poziomie istotności $\alpha = 0,05$ w przypadku SEVQ i trzech algorytmów: EVQ, KNNI oraz OB. Otrzymane wyniki są także zbieżne z wynikami analizy Scotta–Knotta dla miary F1 (rys. 6.30), w której algorytm SEVQ został umieszczony w pierwszej (najlepszej) grupie, wraz z algorytmami SFAM, NB, HAT i HT i przewyższył swoim działaniem algorytmy: LVQ2, EFDT, AEE, LVQ2.1, LVQ, ARF, DWM i LVQ3.

Podsumowując porównanie algorytmu SEVQ z algorytmami inkrementalnymi, należy stwierdzić, że przeprowadzona analiza średnich wartości AUC, ACC, PRE, SEN oraz F1, wraz z zastosowaniem testu Wilcoxona nie wykazała statystycznie istotnych różnic w średnich wartościach ACC między SEVQ a algorytmami EVQ, HAT, HT, KNNI oraz OB. Sugeruje to, że SEVQ jest równie efektywny jak te metody na poziomie istotności $\alpha = 0,05$. W przypadku innych algorytmów, takich jak SFAM, LVQ2.1, LVQ2, ARF, NB, LVQ, EFDT, AEE, DWM i LVQ3, SEVQ wykazał jednak lepszą wydajność, co potwierdza jego przewagę w stosunku do tych metod. Analiza wyników testu Wilcoxona dla metryk AUC, PRE, SEN i F1 również wskazuje na znaczące różnice między SEVQ a większością porównywanych algorytmów inkrementalnych. Pozwoliło to na odrzucenie hipotezy zerowej i przyjęcie alternatywnej hipotezy, zakładającej istnienie znaczących różnic w wydajności. W szczególności SEVQ przewyższa większość porównywanych algorytmów pod względem wszystkich analizowanych metryk, co jest potwierdzone wynikami testu Scotta–Knotta. Ocena wartości p dla różnych metryk oraz analiza rozkładu wartości poszczególnych metryk na wykresach pudełkowych i wyniki testu Scotta–Knotta wskazują na silną pozycję algorytmu SEVQ wśród algorytmów inkrementalnych. SEVQ wykazuje się szczególnie efektywny w porównaniu z algorytmami SFAM, LVQ2.1, LVQ2, ARF, NB, LVQ, EFDT, AEE, DWM i LVQ3. SEVQ jest zatem wartościowym narzędziem uczenia maszynowego, zwłaszcza w zastosowaniach wymagających efektywnego przetwarzania danych w sposób inkrementalny.

7. Implementacja sprzętowa algorytmu SEVQ w układzie FPGA

Prostota algorytmu SEVQ oraz dobre wyniki w stosunku do wielu algorytmów konkurencyjnych stały się motywacją dla autora rozprawy do implementacji sprzętowej tego algorytmu. W rozdziale przedstawiono szczegóły tej implementacji, a także wyniki przeprowadzonych eksperymentów, w tym porównania zużycia zasobów, czasów przetwarzania danych w układzie FPGA i na procesorze ARM oraz wyników jakości klasyfikacji.

7.1. Szczegóły implementacyjne

Implementacja algorytmu SEVQ w układzie FPGA obejmowała kilka etapów. Pierwszym z nich było przepisanie kodu algorytmu SEVQ napisanego w języku programowania Python na kod w języku C++. Następnie, przy użyciu narzędzia Vitis High-Level Synthesis (HLS), kod ten został zsyntetyzowany do kodu Verilog Register Transfer Level (RTL). Proces ten pozwolił na uzyskanie reprezentacji algorytmu w postaci bardziej zbliżonej do struktury układu FPGA.

Kolejnym krokiem było wykorzystanie zintegrowanego środowiska programistycznego Vivado do dalszej implementacji w układzie FPGA. Vivado IDE jest zaawansowanym środowiskiem projektowym firmy Xilinx, służącym do projektowania, implementacji i weryfikacji układów cyfrowych. Oferuje ono zaawansowane funkcje do projektowania układów w FPGA oraz dostarcza intuicyjny interfejs użytkownika. Konfiguracja narzędzi i opcji została przeprowadzona przy użyciu natywnego języka Tool Command Language (Tcl), co umożliwiło elastyczne zarządzanie narzędziami. W środowisku Vivado IDE zaimplementowano oprogramowanie w układzie FPGA, a następnie wygenerowano Bitstream, czyli plik binarny zawierający dane konfiguracyjne dla układu FPGA. W celu przesłania wygenerowanego strumienia bitów do modułu FPGA, skorzystano z interfejsu Pythona dostarczanego przez PYNQ (Python dla Zynq). PYNQ ułatwia efektywną komunikację z układem FPGA, umożliwiając interakcję z zaimplementowanym algorytmem SEVQ przez prosty interfejs programistyczny.

W tabeli 7.1 przedstawiono szczegółowy raport dotyczący zużycia zasobów przez implementację sprzętową algorytmu. Implementacja algorytmu SEVQ na tej płycie FPGA charakteryzuje się wykorzystaniem 10,1% dostępnych komórek LUT (look-

up table), co wskazuje na efektywne wykorzystanie elementów logicznych. LUTRAM (look-up table RAM) stanowi 2,08% dostępnej puli, podczas gdy zużycie przerzutnika typu D (flip-flops, FF) wynosi 6,27%, co sugeruje ich skuteczne wykorzystanie. Jednak wykorzystanie pamięci BRAM wynosi 98,21%. Ta wysoka wartość wynika głównie z konieczności przechowywania wektorów wag i innych struktur danych wykorzystywanych przez SEVQ.

Tabela 7.1: Zużycie zasobów w układzie FPGA

Nazwa zasobu	**Liczba zużytych jednostek**	**Całkowita liczba jednostek**	**Zużycie** [%]
LUT	5374	53200	10,10
LUTRAM	362	17400	2,08
FF	6676	106400	6,27
BRAM	138	140	98,21
BUFG	1	32	3,130

7.2. Porównanie implementacji sprzętowej z programową

W tabeli 7.2 porównano wydajność implementacji w układzie FPGA z realizacją na procesorze ARM pod względem wartości wskaźników AUC, ACC, PRE, SEN i F1. Implementacja algorytmu w układzie FPGA wykazuje wydajność porównywalną z implementacją na procesorze ARM pod względem wyników uzyskanych dla wszystkich metryk. Przykładowo, średnia wartości AUC dla implementacji w układzie FPGA wyniosła 0,747 ±0,176, podczas gdy implementacja na procesorze ARM osiągnęła średnią wartość AUC wynoszącą 0,743 ±0,176. Wyniki te wskazują, że implementacja FPGA nie pogarsza efektywności algorytmu.

Tabela 7.2: Porównanie wyników osiągniętych przez implementacje algorytmu w układzie FPGA i na procesorze ARM na 36 zbiorach danych

Metryka	**Implementacja w układzie FPGA**	**Implementacja na procesorze ARM**
AUC	0,747±0,176	0,743±0,176
ACC	0,700±0,240	0,694±0,244
PRE	0,715±0,235	0,709±0,236
SEN	0,700±0,240	0,694±0,244
F1	0,701±0,239	0,694±0,244

W tabeli 7.3 porównano czasy przetwarzania poszczególnych zbiorów danych przez implementacje algorytmu w układzie FPGA i na procesorze ARM. Rezultat porównania wyraźnie wskazuje na systematyczną przewagę implementacji w układzie

FPGA, charakteryzującej się współczynnikami przyspieszenia z zakresu 0,393-61,822 (w zależności od zbioru danych) w porównaniu z implementacją na procesorze ARM.

Tabela 7.3: Czas przetwarzania poszczególnych zbiorów danych przez implementacje algorytmu w układzie FPGA i na procesorze ARM

Nazwa zbioru danych	**Czas przetwarz. na procesorze ARM [s]**	**Czas przetwarz. w ukł. FPGA [s]**	**Przyspie-szenie**	**Liczba instancji**	**Liczba cech**
marketing	1450,491	23,462	61,822	6876	14
titanic	113,486	3,573	31,759	2201	3
abalone	181,478	10,437	17,388	4174	8
satimage	369,055	22,512	16,394	6435	36
winequality-white	236,955	14,752	16,062	4898	11
thyroid	340,688	23,110	14,742	7200	21
twonorm	185,606	13,429	13,821	7400	20
chess	82,102	6,653	12,341	3196	36
texture	97,250	8,282	11,742	5500	40
led7digit	12,319	1,088	11,321	500	7
winequality-red	26,214	2,348	11,163	1599	11
vehicle	10,090	0,918	10,990	846	18
contraceptive	26,909	2,451	10,979	1473	9
flare	26,475	2,415	10,962	1066	11
yeast	19,785	1,909	10,361	1484	8
phoneme	72,165	7,007	10,299	5404	5
banana	52,404	5,351	9,793	5300	2
german	16,664	1,765	9,441	1000	20
car	15,130	1,762	8,586	1728	6
segment	22,074	2,596	8,504	2310	19
mammographic	7,241	0,921	7,864	830	5
pima	5,927	0,951	6,229	768	8
mushroom	28,185	6,076	4,638	5644	22
tic-tac-toe	7,676	1,760	4,361	958	9
vowel	6,714	1,759	3,816	990	13
balance	3,471	0,917	3,786	625	4
wisconsin	3,112	0,920	3,382	683	9
saheart	3,104	0,940	3,301	462	9
bands	2,713	0,920	2,950	365	19
dermatology	2,422	0,926	2,617	358	34
ionosphere	2,362	0,929	2,541	351	33
cleveland	2,111	0,924	2,285	297	13
bupa	1,915	0,923	2,075	345	6
monk-2	1,793	0,923	1,943	432	6
ecoli	1,720	0,926	1,857	336	7
heart	1,639	0,920	1,781	270	13
breast	1,614	0,921	1,753	277	9
automobile	1,084	0,921	1,177	150	25
glass	1,067	0,935	1,141	214	9
housevotes	1,009	0,933	1,081	232	16
lymphography	0,908	0,939	0,967	148	18

Tabela 7.3 (cd.): Czas przetwarzania poszczególnych zbiorów danych przez implementacje algorytmu w układzie FPGA i na procesorze ARM

Nazwa zbioru danych	**Czas przetwarz. na procesorze ARM [s]**	**Czas przetwarz. w ukł. FPGA [s]**	**Przyspieszenie**	**Liczba instancji**	**Liczba cech**
wine	0,876	0,922	0,950	178	13
newthyroid	0,853	0,933	0,914	215	5
hayes-roth	0,783	0,929	0,843	160	4
tae	0,731	0,917	0,798	151	5
iris	0,554	0,921	0,601	150	4
appendicitis	0,439	0,911	0,482	106	9
post-operative	0,421	0,952	0,442	87	8
zoo	0,402	0,919	0,437	101	17
hepatitis	0,364	0,926	0,393	80	19

Implementacja w układzie FPGA wykazuje szczególnie znaczącą przewagę w przypadku przetwarzania większych zbiorów danych, takich jak zbiór danych marketingowych. W tym konkretnym przypadku czas przetwarzania w układzie FPGA był krótszy o ponad 1427 sekund w porównaniu z implementacją na procesorze ARM, a iloraz czasów wynosił ponad 61 razy, co ilustruje imponującą przewagę i szybkość działania algorytmu w układzie FPGA w porównaniu z implementacją na procesorze ARM. W przypadku 20% zbiorów danych implementacja algorytmu SEVQ w układzie FPGA okazała się wolniejsza od implementacji na procesorze ARM. Przypadki te dotyczyły jednak bardzo małych zbiorów danych, które zawierały 80-215 rekordów opisanych przez 4-19 cech.

Uzyskane wyniki potwierdzają skuteczność implementacji algorytmu w układzie FPGA oraz jej wpływ na przyspieszenie jego działania, szczególnie w kontekście obsługi dużych zbiorów danych. Do innych zysków z implementacji sprzętowej można zaliczyć niższy pobór energii w porównaniu z tradycyjnym procesorem, możliwość zrównoleglania obliczeń, elastyczność i rekonfigurowalność pozwalające na optymalizację działania algorytmu w zależności od konkretnych potrzeb oraz potencjał w zastosowaniach algorytmu w przetwarzaniu w czasie rzeczywistym. Warto również wspomnieć, że pomimo początkowych kosztów związanych z projektowaniem i wdrożeniem układu FPGA, w dłuższej perspektywie może być to bardziej opłacalne rozwiązanie niż skalowanie rozwiązań opartych na procesorach, zwłaszcza przy dużych obciążeniach obliczeniowych.

8. Opracowanie oprogramowania służącego do oceny jakości klasyfikacji

W rozdziale przedstawiono motywację i charakterystykę zaproponowanego oprogramowania przeznaczonego do oceny jakości algorytmów klasyfikacyjnych, ze wskazaniem różnych możliwości przeprowadzania analizy. Zawarto również konkretne przykłady użycia narzędzia do analizy wyników uzyskanych przez wybrane algorytmy na różnych zbiorach danych.

8.1. Motywacja i charakterystyka oprogramowania

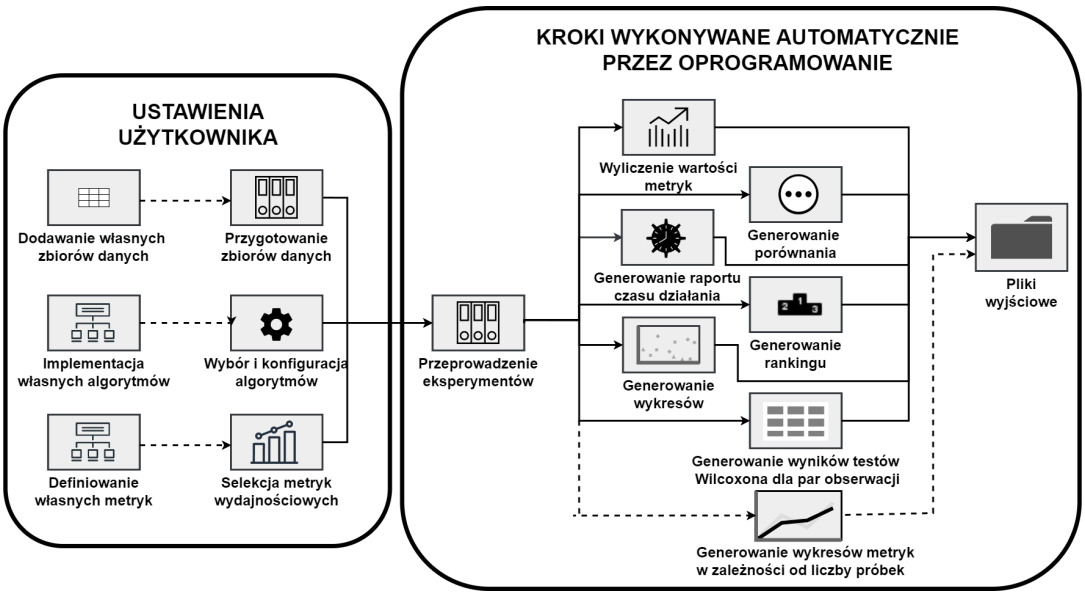
W obszarze sztucznej inteligencji wciąż powstają nowe metody i algorytmy, co wymaga rzetelnego porównania ich z istniejącymi rozwiązaniami [62, 63]. Testowanie wydajności algorytmów na wielu zbiorach danych oraz analiza różnych metryk wydajnościowych są jednak czasochłonne i bardzo absorbujące. Konieczne jest także zastosowanie odpowiednich testów statystycznych do oceny istotności różnic między wynikami algorytmów [64]. Istotne jest również stosowanie tego samego środowiska testowego dla wszystkich porównywanych algorytmów, aby zapewnić wiarygodność i porównywalność otrzymanych wyników, co niestety bywa pomijane przez wielu autorów wykonujących podobne badania porównawcze (benchmarks).

Narzędzia, które pozwalają na automatyczne porównywanie wyników, to na przykład *Classification Learner App* czy *MATLAB Statistics and Machine Learning Toolbox*. Pierwsze z nich jest wbudowanym narzędziem w środowisku MATLAB, umożliwiającym użytkownikom trenowanie i ocenę różnych modeli klasyfikacyjnych. Drugie to Toolbox zawierający implementacje różnych algorytmów klasyfikacyjnych oraz narzędzia do oceny ich wydajności. Inną platformą do analizy danych i uczenia maszynowego, zawierającą narzędzia do budowy, wizualizacji i oceny modeli klasyfikacyjnych jest *RapidMiner*. Są to jednak narzędzia komercyjne, które wymagają zakupu licencji. W grupie narzędzi open-source bardzo popularna jest *Weka* [65], która dostarcza wielu algorytmów klasyfikacji, czy biblioteka *Scikit-learn* [45] napisana w języku Python. Metody te mają jednak pewne ograniczenia i wymagają wciąż sporego nakładu pracy do przeprowadzenia badań porównawczych.

W celu zaadresowania wyżej wymienionych wymagań związanych z porównywaniem nowo powstałych technik klasyfikacji z innymi już istniejącymi pojawiła się

idea stworzenia efektywnego narzędzia, które umożliwiłoby szybkie i dokładne testowanie algorytmów. Narzędzie *Classification Algorithms Comparison Pipeline* (CACP) zaprojektowano w celu uproszczenia procesu porównywania nowo opracowanych algorytmów klasyfikacyjnych z innymi oraz zapewnienia reprodukowalności badań [66]. Automatyzacja tego procesu może pomóc w eliminacji błędów ludzkich i zapewnić rzetelne wyniki, a także znacznie przyspieszyć prowadzenie badań. Automatyczna ocena wydajności algorytmów pozwala na analizę różnych metryk wydajnościowych na wielu zbiorach danych. Schemat działania CACP został przedstawiony na rys. 8.1.

Narzędzie zostało napisane w języku programowania Python, który jest wiodącym językiem w obszarze analizy danych i uczenia maszynowego, z szerokim wsparciem bibliotek, takich jak Scikit-learn, TensorFlow czy PyTorch. Wybór języka Python do implementacji tego narzędzia zapewnia łatwą integrację z istniejącymi narzędziami i dostępność dla szerokiego grona badaczy zainteresowanych tą tematyką.



Rysunek 8.1: Schemat działania CACP

Zadaniem użytkownika jest wybranie algorytmów, zbiorów danych oraz metryk wydajnościowych służących do porównania, natomiast oprogramowanie automatycznie przeprowadza eksperymenty mające na celu porównanie efektywności działania algorytmów. W tym celu są obliczane wartości metryk wydajnościowych. Narzędzie generuje również wykresy rozkładu wartości metryk wydajnościowych oraz tabele z wynikami testu Wilcoxona dla par obserwacji, a także tworzy ranking zwycięstw poszczególnych algorytmów na wybranych do porównania zbiorach danych.

8.2. Założenia dotyczące funkcjonalności

Zrozumienie działania algorytmów tradycyjnych i inkrementalnych jest nadal istotne w celu rozwijania nowych metod i podejść, w tym również dla ulepszania algorytmów głębokich przez ich integrację z tradycyjnymi technikami. W związku z tym porównywanie tych typów algorytmów może pomóc w lepszym zrozumieniu ich mocnych i słabych stron, a także w identyfikacji najlepszych podejść w zależności od konkretnego problemu lub środowiska pracy.

CACP umożliwia przeprowadzanie dwóch rodzajów porównań: dla algorytmów tradycyjnych oraz algorytmów inkrementalnych. Wybór rodzaju eksperymentu determinuje, które klasyfikatory i zbiory danych można później wykorzystać. W Dodatku A omówiono kolejne etapy przygotowywania eksperymentu dla obu rodzajów algorytmów, obejmujące przygotowanie zbioru danych, wybór algorytmów i konfigurację parametrów, selekcję metryk wydajnościowych oraz uruchomienie eksperymentu i prezentację wyników.

W trakcie porównywania algorytmów użytkownik może korzystać z zestawów danych automatycznie pobieranych z repozytorium danych KEEL. Repozytorium to oferuje szeroką gamę zestawów danych z różnych dziedzin, które są odpowiednio przygotowane do przeprowadzania różnorodnych eksperymentów i testów algorytmów. Oprócz tych zestawów istnieje również możliwość wyboru zbiorów danych odpowiednich do testowania algorytmów inkrementalnych spośród zbiorów danych pochodzących z biblioteki River (`https://riverml.xyz/`). Zbiory danych w bibliotece River są dobrze przystosowane do testowania algorytmów inkrementalnych, cechujących się przede wszystkim dostosowywaniem się do nowych danych i uczeniem się na bieżąco. Do porównania użytkownik może wybrać algorytmy dostępne w bibliotece Scikit-learn oraz River, ma także opcję dodania własnej implementacji. Do oceny efektywności algorytmów tradycyjnych użytkownik może skorzystać z metryk wydajnościowych z biblioteki Scikit-learn oraz River z możliwością dodania własnych metryk.

Po wyborze zbiorów danych, metryk wydajnościowych oraz konfiguracji algorytmów użytkownik może rozpocząć analizę porównawczą, której wynikiem są liczne tabele oraz wykresy umożliwiające porównanie otrzymanych wyników.

Narzędzie CACP może być wykorzystywane zarówno z poziomu interfejsu programistycznego, jak i interfejsu graficznego, co pozwala dostosować się do preferencji

i umiejętności użytkowników na różnych poziomach zaawansowania.

CACP został zaimplementowany w języku Python 3 przy użyciu popularnych bibliotek, takich jak Scikit-learn [45], River [67], Numpy [68], Pandas [69] i Matplotlib [70]. Interfejs graficzny został stworzony przy użyciu frameworka Dash. Szczegółowe informacje dotyczące architektury narzędzia oraz implementacji zamieszczono w Dodatku B.

8.3. Przykład zastosowania narzędzia CACP

Na Listingu 3 przedstawiono przykład użycia narzędzia CACP do przygotowania porównania algorytmów inkrementalnych. W tym celu wybrano cztery zbiory danych, pochodzące z biblioteki River (tab. 8.1).

Tabela 8.1: Zbiory danych wykorzystane w przykładowym badaniu przeprowadzonym za pomocą narzędzia CACP

	Dataset	Instances	Features	Classes
1	iris	150	4	3
2	wisconsin	683	9	2
3	phishing	1250	9	2
4	bananas	5300	2	2

Do porównania wybrano cztery klasyfikatory inkrementalne z domyślnymi ustawieniami parametrów i zaprezentowano je w tab. 8.2. Implementacje algorytmów również pochodziły z biblioteki River. Metryki wydajnościowe nie zostały zdefiniowane, dlatego wynikiem eksperymentu będą wartości domyślnych metryk wydajnościowych, tj. AUC, ACC, PRE, SEN i F1.

Tabela 8.2: Algorytmy wybrane do przeprowadzenia przykładowego porównania za pomocą narzędzia CACP

	Name	Class Name	Library	Type
1	ARF	ARFClassifier	river	INCREMENTAL
2	GNB	GaussianNB	river	INCREMENTAL
3	HAT	HoeffdingTreeClassifier	river	INCREMENTAL
4	KNN	KNNClassifier	river	INCREMENTAL

Tabela 8.3 zawiera średnie wartości domyślnie ustawionych metryk wydajnościowych wraz z odchyleniami standardowymi dla poszczególnych algorytmów. Wyniki są posortowane malejąco według AUC. Można zauważyć, że klasyfikator ARF osiągnął najlepsze wyniki według AUC (0,963±0,017), ACC (0,917±0,037) oraz PRE

(0,897±0,030). Pod względem SEN osiągnął on drugi najlepszy wynik (0,897±0,044), zaraz po inkrementalnym klasyfikatorze KNN (0,901±0,034). Pod względem F1 klasyfikator ARF osiągnął taki sam wynik jak KNN (0,897±0,035).

Tabela 8.3: Tabela średnich wartości metryk wydajnościowych wraz z odchyleniami standardowymi obliczonych dla poszczególnych algorytmów

	Algorithm	AUC	Accuracy	Precision	Recall	F1
1	ARF	0.963±0.017	0.917±0.037	0.897±0.030	0.897±0.044	0.897±0.035
2	KNN	0.953±0.026	0.916±0.038	0.894±0.053	0.901±0.034	0.897±0.035
3	HAT	0.882±0.135	0.859±0.149	0.848±0.125	0.788±0.261	0.810±0.208
4	GNB	0.872±0.153	0.854±0.163	0.841±0.132	0.773±0.315	0.791±0.254

W tabeli 8.4 zawarto informacje na temat czasu działania algorytmów klasyfikacyjnych podczas uczenia i testowania. Wynika z niej, że najszybciej uczył się algorytm KNN, natomiast charakteryzował go najdłuższy czas predykcji. Najdłuższego czasu uczenia wymagał ARF, którego czas predykcji był znacznie dłuższy od GNB i HAT.

Tabela 8.4: Czas działania algorytmów klasyfikacyjnych podczas uczenia i testowania

	Algorithm	Train time [s]	Prediction time [s]
1	KNN	0.022659	10.951088
2	GNB	0.078650	0.349685
3	HAT	0.303636	0.135754
4	ARF	5.449630	1.140845

Tabela 8.5 zawiera ranking porównywanych algorytmów inkrementalnych. Poszczególne kolumny tabeli prezentują liczbę zbiorów danych, w których dany algorytm osiągnął odpowiednio najlepszy wynik, drugi najlepszy wynik i trzeci najlepszy wynik pod względem ACC. W przeprowadzonym porównaniu algorytm GNB osiągał najlepsze wyniki ACC na największej liczbie zbiorów danych.

Tabela 8.5: Przykładowy ranking algorytmów porównywanych według ACC

	Algorithm	1st	2nd	3rd
1	GNB	2	0	1
2	ARF	1	1	1
3	KNN	1	1	1
4	HAT	0	2	1

Listing 3: Przykładowe wykorzystanie CACP w porównaniu algorytmów inkrementalnych

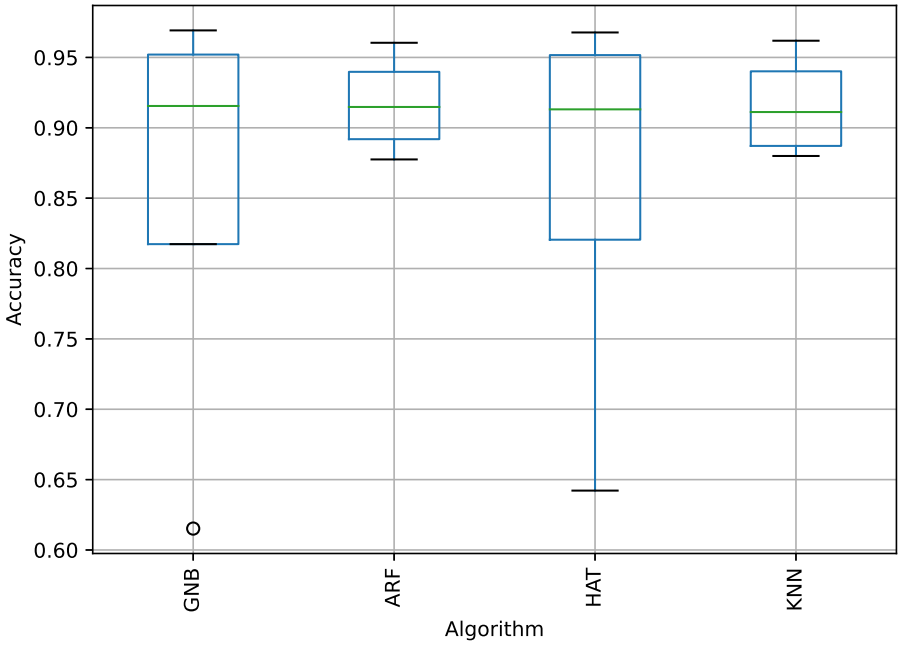
```
import river
from river.forest import ARFClassifier
from river.tree import HoeffdingTreeClassifier
from river.neighbors import KNNClassifier
from river.naive_bayes import GaussianNB
from cacp import run_incremental_experiment, ClassificationDataset

experimental_datasets = [
    ClassificationDataset('iris'),
    ClassificationDataset('wisconsin'),
    # you can use datasets from river
    river.datasets.Phishing(),
    river.datasets.Bananas(),
]

experimental_classifiers = [
    ('ARF', lambda n_inputs, n_classes: ARFClassifier()),
    ('HAT', lambda n_inputs, n_classes: HoeffdingTreeClassifier()),
    ('KNN', lambda n_inputs, n_classes: KNNClassifier()),
    ('GNB', lambda n_inputs, n_classes: GaussianNB()),
]

run_incremental_experiment(
    experimental_datasets,
    experimental_classifiers,
    results_directory='./example_result'
)
```

Tabela 8.6 zawiera wyniki testu Wilcoxona dla algorytmu ARF w porównaniu z resztą algorytmów pod względem wartości ACC. P-wartość (p-value) uwzględniona w tabeli stanowi miarę pomagającą określić istotność statystyczną wyników. Informuje ona o prawdopodobieństwie uzyskania wyników, które są przynajmniej tak skrajne jak obserwowane wyniki, w przypadku gdyby hipoteza zerowa była prawdziwa.

Tabela 8.6: Wyniki testu Wilcoxona dla algorytmu ARF w porównaniu z resztą algorytmów pod względem ACC

	ARF	Algorithm	p-value
1	ARF	GNB	0.875000
2	ARF	HAT	0.625000
3	ARF	KNN	1.000000

Na rysunku 8.2 zaprezentowano przykład wygenerowanego przez CACP wykresu rozkładów wartości ACC dla każdego algorytmu. Rysunek 8.3 przedstawia przykład

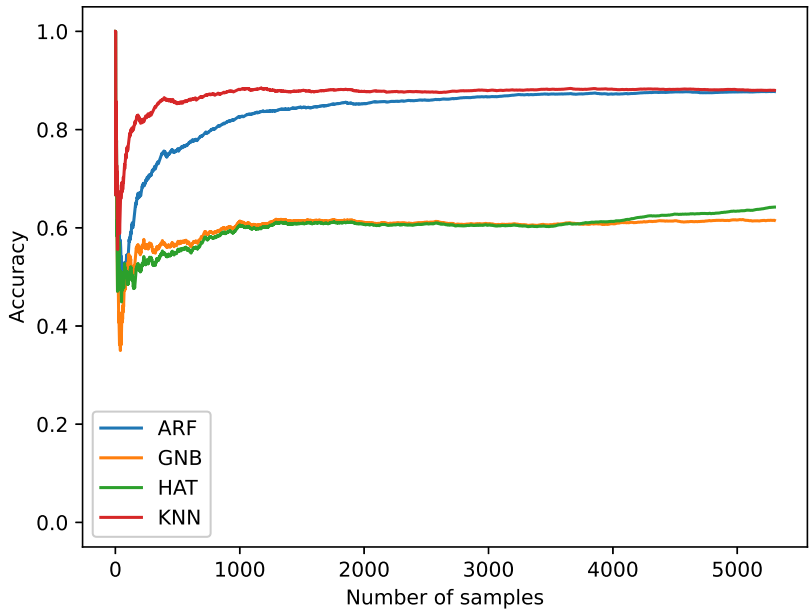


Rysunek 8.2: Przykładowy wykres pudełkowy dla wartości ACC każdego algorytmu

wykresu zależności między wartością ACC a liczbą próbek użytych podczas treningu.

Zaprezentowane tab. 8.1-8.6 oraz wykresy przedstawione na rys. 8.2 i 8.3 stanowią tylko niewielki procent wszystkich rezultatów wygenerowanych przez CACP. Przykłady te można mnożyć, ponieważ CACP generuje pojedyncze tabele dla wszystkich zbiorów danych, algorytmów oraz metryk wydajnościowych, a także tabele zbiorcze tam, gdzie jest to możliwe. Opisy kolumn w tabelach celowo pozostawiono w języku angielskim, natomiast liczby zawierające części dziesiętne zapisano z kropką, ponieważ w taki sposób są one oryginalnie generowane przez oprogramowanie. Można je jednak edytować w bardzo prosty sposób i dostosować opisy tabeli do własnych potrzeb. Narzędzie CACP zostało wykorzystane do porównania nowego klasyfikatora inkrementalnego opartego na kwantyzacji wektorowej i adaptacyjnej teorii rezonansu (SEVQ), który został opracowany jako istotna część niniejszej rozprawy, z popularnymi algorytmami tradycyjnymi oraz inkrementalnymi. Wyniki przedstawiono i omówiono w rozdziale 6.

Kod narzędzia został udostępniony w postaci otwartego oprogramowania pod adresem `https://github.com/sylwekczmil/cacp`, natomiast dokumentacja jest dostępna pod adresem `https://cacp.readthedocs.io/en/latest/`. Ponadto, pod adresem internetowym `https://cacp.czmil.com/` została udostępniona wersja demonstracyjna narzędzia, która pozwala użytkownikom na zapoznanie się z funkcjonalno-



Rysunek 8.3: Przykładowy wykres zależności między wartością ACC a liczbą próbek użytych podczas treningu

ściami oferowanymi przez CACP. Umożliwia ona przejście pełnego procesu konfiguracji wybranego rodzaju eksperymentu oraz podgląd wygenerowanych wyników.

9. Podsumowanie i wnioski końcowe

W niniejszej rozprawie poruszono problematykę nadzorowanego uczenia klasyfikatorów płytkich na podstawie danych uczących, z uwzględnieniem oceny jakości ich klasyfikacji. Zaproponowano nowy, inkrementalny algorytm klasyfikacji danych SEVQ oraz dokonano jego implementacji programowej i sprzętowej. Algorytm porównano z innymi algorytmami klasyfikacyjnymi, zarówno tradycyjnymi, jak i inkrementalnymi. Okazał się on równie efektywny jak tradycyjne algorytmy płytkie, a także przewyższył większość porównywanych algorytmów inkrementalnych pod względem wszystkich analizowanych metryk. Otrzymane wyniki zostały potwierdzone analizą rozkładu wartości poszczególnych metryk na wykresach pudełkowych oraz wynikami testów Scotta-Knotta i Wilcoxona.

Implementacja sprzętowa algorytmu w układzie FPGA znacznie przyspieszyła działanie, zwłaszcza w przypadku dużych zbiorów danych. Przeprowadzono również porównanie obu implementacji pod kątem jakości klasyfikacji, czasu uczenia oraz czasu klasyfikacji.

Opracowano także narzędzie programowe CACP, umożliwiające porównywanie wydajności różnych algorytmów klasyfikacyjnych, w tym tradycyjnych i inkrementalnych. Narzędzie to pozwala na obliczanie metryk wydajnościowych oraz wykonywanie testów statystycznych w celu porównania różnych klasyfikatorów i ułatwienia wyboru najodpowiedniejszego klasyfikatora dla danego zastosowania. Zapewnia również odtwarzalność wyników badań, co jest kwestią kluczową w badaniach naukowych.

Wkład autora rozprawy

Główny wkład autora rozprawy do dyscypliny naukowej informatyka techniczna i telekomunikacja jest następujący:

1) Zaproponowanie nowego algorytmu przyrostowego klasyfikacji danych (SEVQ) z minimalną liczbą parametrów nastrajanych.

2) Przeprowadzenie wszechstronnych badań porównawczych nowego klasyfikatora, z uwzględnieniem:

    - dużej liczby zbiorów danych do klasyfikacji,

- dużej liczby dotychczas stosowanych algorytmów płytkich, zarówno inkrementalnych, jak i nieinkrementalnych,
- wielu wskaźników jakości klasyfikacji (AUC, ACC itd.).

3) Przeprowadzenie grupowania za pomocą algorytmu Scotta–Knotta w celu ulokowania nowego algorytmu SEVQ na odpowiedniej pozycji wśród algorytmów dotychczas stosowanych.

4) Zastosowanie testu Wilcoxona w celu wykazania istotnych statystycznie różnic (bądź ich braku) w średnich wartościach typowych wskaźników jakości klasyfikacji na poziomie istotności $\alpha = 0,05$.

5) Implementacja programowa nowego algorytmu w języku Python i sprzętowa - na układach FPGA.

6) Porównanie implementacji programowej ze sprzętową z uwzględnieniem wskaźników jakości klasyfikacji danych, czasu uczenia i czasu trwania klasyfikacji.

7) Opracowanie uniwersalnego narzędzia do wspomagania procesu wszechstronnego testowania algorytmów klasyfikacji danych (inkrementalnych lub nieinkrementalnych) w języku Python, które spełnia istotne wymagania w zakresie badań benchmarkowych i jest wyposażone w wygodny graficzny interfejs użytkownika.

Mimo, że opracowany nowy algorytm SEVQ korzysta z idei klasyfikacji danych opartych na kwantyzacji wektorowej (LVQ) i adaptacyjnej teorii rezonansu (ARTMAP), jest on mniej skomplikowany od wszystkich dotychczasowych algorytmów należących do tej rodziny. Nie ma parametrów nastrajalnych w algorytmie SEVQ. Co więcej, klasyfikowane dane nie wymagają normalizacji.

Badania autora wykazały, że prostota danego algorytmu wcale nie oznacza, że jest on gorszy od większości dotąd stosowanych. Aby uwiarygodnić to stwierdzenie, autor rozprawy skonstruował specjalne narzędzie CACP w języku Python. Badania autora zdają się potwierdzać twierdzenie „No Free Lunch” i usprawiedliwiają sens prowadzenia badań polegających na stworzeniu prostych, ale wydajnych algorytmów. Warto dodać, że niewielka jest liczba artykułów naukowych na ten temat.

Należy podkreślić, że dotychczasowe badania koncentrowały się na stworzeniu nowego algorytmu przyrostowego i jego rzetelnej ewaluacji. Jednak uczenie przyrostowe niesie ze sobą wiele wyzwań, które warto podjąć w przyszłych pracach, takich jak: uwzględnienie niestacjonarności danych, zjawiska dryfu koncepcji (concept drift), zmniejszenie wpływu danych historycznych, zmiany w rozkładzie klas, zmiany w rozmiarze zbioru danych i wiele innych.

Dodatek A. Procedura przeprowadzenia eksperymentu z wykorzystaniem CACP

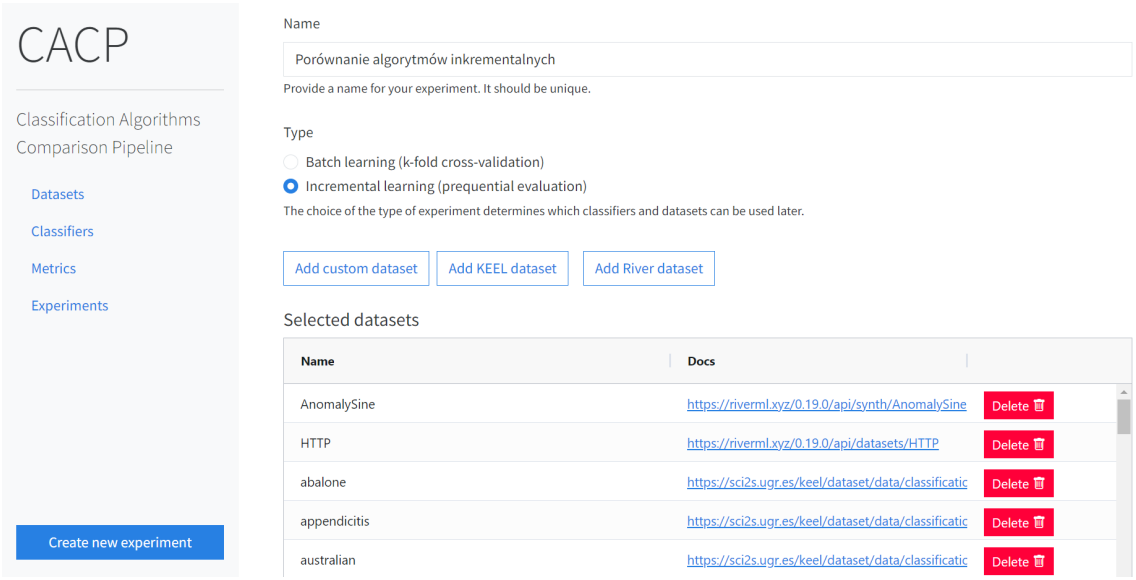
Przygotowanie zbiorów danych

W procesie przygotowywania porównania dla algorytmów tradycyjnych użytkownik może skorzystać z zestawów danych automatycznie pobieranych z repozytorium danych KEEL [48]. Do jego dyspozycji jest 69 różnorodnych zbiorów danych o wartościach numerycznych i zróżnicowanym stopniu zbalansowania klas. Dodatkowo użytkownik ma możliwość dostarczenia własnych zbiorów danych do eksperymentów. Zbiory te mogą być przekazane w formie plików CSV lub DAT, o strukturze analogicznej do plików pobieranych z repozytorium KEEL, lub też mogą być dostarczone w postaci własnej implementacji dziedziczącej po klasie *cacp.dataset. ClassificationDataset*.

W przypadku przygotowywania analizy porównawczej dla algorytmów inkrementalnych użytkownik ma dostęp do rozszerzonego zakresu zbiorów danych. Oprócz możliwości wykorzystania zbiorów danych dostępnych w repozytorium KEEL istnieje również możliwość wyboru spośród 14 zbiorów danych dostępnych w bibliotece River. Na rysunku 9.1 przedstawiono fragment widoku interfejsu użytkownika wyboru algorytmów w procesie porównania algorytmów inkrementalnych. Interfejs umożliwia tworzenie nowych eksperymentów. W tym celu należy podać unikatową nazwę oraz wybrać rodzaj eksperymentu, czy to w kontekście porównania algorytmów tradycyjnych czy inkrementalnych. Po dokonaniu wyboru rodzaju eksperymentu użytkownik może dokonać wyboru zbioru danych z listy, zaznaczając przy jego nazwie pole wyboru, bądź wgrywając i zaznaczając pole wyboru przy własnym zbiorze.

Wybór algorytmów i konfiguracja parametrów

Analogicznie do wyboru zbiorów danych, lista dostępnych algorytmów różni się w zależności od rodzaju przeprowadzanego eksperymentu. W przypadku porównywania algorytmów tradycyjnych użytkownik ma możliwość wyboru algorytmów z biblioteki Scikit-learn [45]. Na liście algorytmów znajduje się ich 34. Rysunek 9.2 przedstawia okno modalne interfejsu użytkownika, umożliwiające wybór algorytmów w procesie porównania algorytmów inkrementalnych. Wybór algorytmu do porównania odbywa się



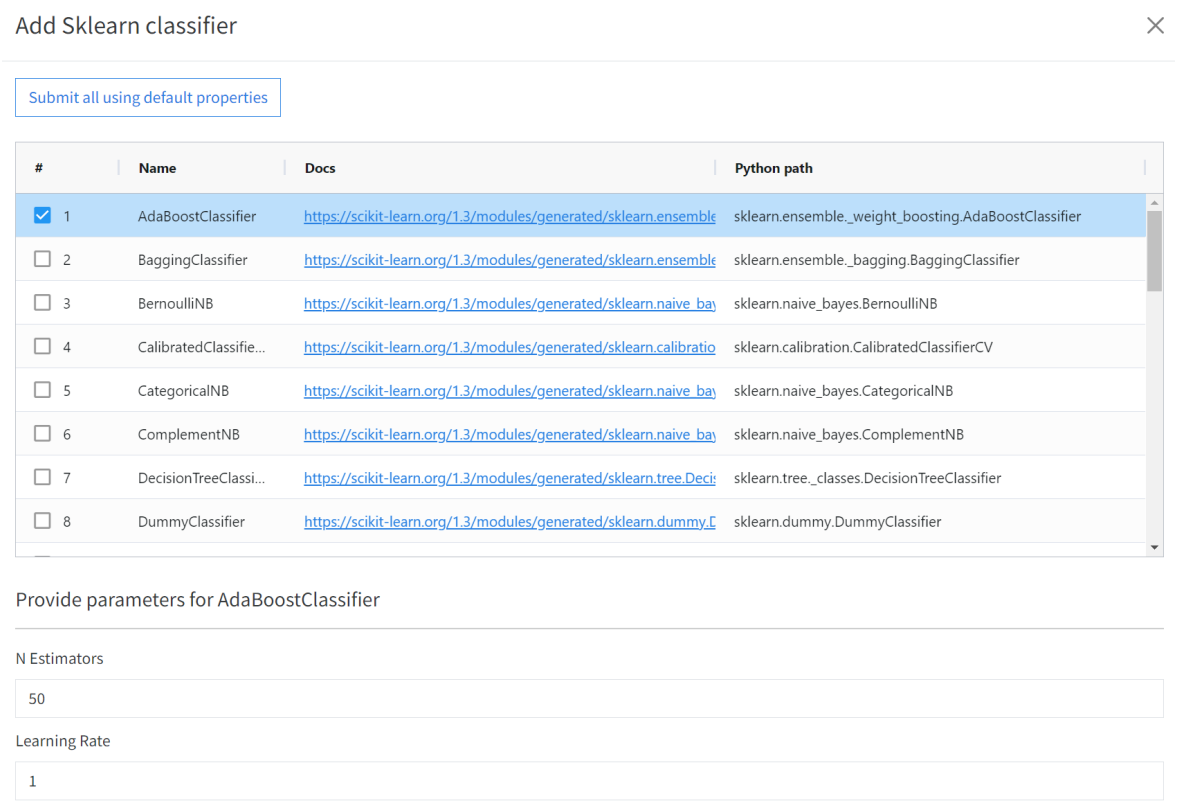
Rysunek 9.1: Fragment widoku interfejsu użytkownika wyboru zbiorów danych w procesie porównania algorytmów inkrementalnych

poprzez zaznaczenie pola wyboru przy nazwie algorytmu, uzupełnienie odpowiednich parametrów konfiguracyjnych (lub skorzystanie z ustawień domyślnych), a następnie zatwierdzenia wprowadzonych ustawień. Dodatkowo użytkownik ma możliwość dodawania własnych algorytmów, ale pod warunkiem, że ich interfejsy są takie same jak algorytmów dostępnych w bibliotece Scikit-learn.

W kontekście porównywania algorytmów inkrementalnych użytkownik ma możliwość selekcji spośród 24 algorytmów tego typu dostępnych w bibliotece River. Dodatkowo istnieje opcja dodania własnej implementacji algorytmu.

Selekcja metryk wydajnościowych

Efektywność działania algorytmów jest oceniana za pomocą wybranych metryk wydajnościowych. W przypadku oceny efektywności algorytmów tradycyjnych użytkownik ma do dyspozycji sześć metryk wydajnościowych, takich jak AUC, ACC, metryka F1, MCC, PRE i SEN pochodzących z biblioteki Scikit-learn. Ma także możliwość dodawania własnych metryk wydajnościowych, jednak pod warunkiem, że mają one takie same interfejsy jak metryki dostępne w bibliotece Scikit-learn. W przypadku algorytmów inkrementalnych użytkownik ma możliwość wyboru z listy 34 różnych metryk zaimplementowanych w bibliotece River bądź dodania własnych.



Rysunek 9.2: Okno modalne interfejsu użytkownika umożliwiające wybór algorytmów w procesie porównania algorytmów tradycyjnych

Uruchamianie eksperymentów i prezentacja wyników

Po dokonaniu wyboru zbiorów danych, metryk wydajnościowych oraz wyborze i konfiguracji algorytmów użytkownik może rozpocząć analizę porównawczą. Jej wynikiem jest folder wyjściowy zawierający:

- tabelę ze średnimi wartościami wybranych metryk wydajnościowych oraz odchyleniami standardowymi dla poszczególnych algorytmów,
- tabelę zawierającą ranking zwycięstw poszczególnych algorytmów na wybranych do porównania zbiorach danych,
- tabelę z czasami działania algorytmów klasyfikacyjnych podczas uczenia i testowania,
- tabele z wynikami testu Wilcoxona dla każdego algorytmu w porównaniu z resztą algorytmów,
- wykresy rozkładów wartości poszczególnych metryk wydajnościowych,

- tabele zawierające informacje o zastosowanych zbiorach danych (z wyszczególnieniem liczby atrybutów, klas oraz rekordów) oraz klasyfikatorach,
- dodatkowo, przy porównywaniu algorytmów inkrementalnych CACP generuje wykresy wartości poszczególnych metryk wydajnościowych w zależności od liczby użytych do treningu próbek.

Tabele wynikowe są eksportowane w formatach CSV i TEX, natomiast wykresy są prezentowane w formatach PNG oraz EPS, co umożliwia elastyczne wykorzystanie tych danych w różnych publikacjach naukowych i raportach. Tabele wygenerowane w formacie CSV można odczytać za pomocą niemal każdego edytora tekstu i mogą być przetwarzane za pomocą innych bibliotek napisanych w języku Python, np. Pandas.

Dodatek B. Architektura narzędzia i szczegóły implementacyjne

CACP został zaimplementowany w języku Python 3, który jest najpopularniejszym językiem używanym do tworzenia oprogramowania opartego na uczeniu maszynowym i sztucznej inteligencji. Implementacja narzędzia wykorzystuje biblioteki Scikit-learn, River, Numpy, Pandas i Matplotlib, które umożliwiają prostą implementację algorytmów uczenia maszynowego, a także przetwarzanie i analizę danych. Interfejs graficzny został zaimplementowany przy użyciu interaktywnego frameworka Dash służącego do budowy aplikacji internetowych opartych na języku Python.

Korzystanie z narzędzia CACP do porównywania algorytmów zaimplementowanych w języku Python jest stosunkowo intuicyjne. Wymagania dotyczące implementacji nowego algorytmu ograniczają się jedynie do posiadania interfejsu zgodnego z tym wykorzystywanym w bibliotece Scikit-learn, co obejmuje implementację metod fit() i predict() w odpowiednim formacie. Te metody pełnią funkcję dostosowywania modeli do podanych danych uczących oraz wykonują klasyfikację na zbiorach testowych. CACP jest skonstruowany z kilku modułów, z których każdy przeznaczony jest do wykonywania specyficznego zadania w ramach procesu analizy porównawczej. Dalej wymieniono najważniejsze moduły:

- *cacp.dataset* – udostępnia klasy i funkcje pozwalające na automatyczne pobieranie wybranych lub wszystkich dostępnych zbiorów danych z repozytorium KEEL-dataset. Do wyboru są zbiory danych automatycznie podzielone za pomocą procedury 5-krotnej i 10-krotnej walidacji krzyżowej i przygotowane do przeprowadzenia stratyfikowanego sprawdzianu krzyżowego (SVC), polegającego na takim podziale obiektów pomiędzy zbiór uczący i zbiór testowy, aby zachowane były oryginalne proporcje pomiędzy klasami decyzyjnymi, a także stratyfikowanego sprawdzianu krzyżowego przy optymalnie zrównoważonym rozkładzie (DOB SCV), zapewniającego równomierną reprezentację wszystkich klas w każdym podziale zbioru danych, aby ocena modelu była bardziej rzetelna i obiektywna [71],

- *cacp.comparison* – udostępnia funkcje pozwalające na przeprowadzanie porównań klasyfikatorów,

- *cacp.info* – zawiera funkcje generujące pliki wynikowe, zawierające listę wszystkich zbiorów danych oraz klasyfikatorów wraz z ich atrybutami, które zostały użyte do przeprowadzenia eksperymentu,

- *cacp.plot* – umożliwia tworzenie wykresów wartości rozkładu metryk wydajnościowych na podstawie wyników porównania,

- *cacp.result* – składa się z funkcji pozwalających na przetwarzanie wyników porównań, np. obliczających średnie wartości dla wszystkich metryk wydajnościowych,

- *cacp.time* – zawiera funkcję obliczającą średni czas działania funkcji fit() i predict(),

- *cacp.util* – zawiera funkcje pomocnicze,

- *cacp.wilcoxon* – służy do przeprowadzania testu Wilcoxona dla każdego algorytmu w porównaniu z resztą algorytmów,

- *cacp.winner* – zawiera funkcje obliczające ranking klasyfikatorów na podstawie wyników uzyskanych na każdym zbiorze danych,

- *cacp.run* – moduł łączący wszystkie pozostałe moduły w jedną całość.

W celu uruchomienia narzędzia CACP użytkownik musi przygotować listę odpowiednich algorytmów wraz z ustawieniami parametrów, a następnie listę wybranych zbiorów danych oraz metryk wydajnościowych, a następnie uruchomić funkcję run_experiment() z modułu *cacp.run*.

Spis rysunków

Spis tabel

Literatura

[1] R. Agharafeie, J. R. C. Ramos, J. M. Mendes, R. Oliveira, From shallow to deep bioprocess hybrid modeling: Advances and future perspectives, Fermentation 9 (10) (2023). doi:10.3390/fermentation9100922.
URL https://www.mdpi.com/2311-5637/9/10/922

[2] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction to Statistical Learning: with Applications in R, Springer US, 2021. doi:10.1007/978-1-0716-1418-1.
URL http://dx.doi.org/10.1007/978-1-0716-1418-1

[3] V. N. Vapnik, The nature of statistical learning theory, Springer-Verlag New York, Inc., 1995.

[4] L. Breiman, Random Forests, Machine Learning 45 (1) (2001) 5–32. doi:10.1023/a:1010933404324.

[5] X. Wu, V. Kumar, J. R. Quinlan, et al., Top 10 algorithms in data mining, Knowledge and Information Systems 14 (1) (2007) 1–37. doi:10.1007/s10115-007-0114-2.

[6] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, ACM, New York, NY, USA, 2016, pp. 785–794. doi:10.1145/2939672.2939785.

[7] X. Shi, Y. Wong, M. Z.-F. Li, et al., A feature learning approach based on XGBoost for driving assessment and risk prediction, Accident Analysis & Prevention 129 (2019) 170–179. doi:10.1016/j.aap.2019.05.005.

[8] Y. Luo, L. Yin, W. Bai, K. Mao, An appraisal of incremental learning methods, Entropy 22 (11) (2020) 1190. doi:10.3390/e22111190.

[9] R. Polikar, L. Upda, S. Upda, V. Honavar, Learn++: an incremental learning algorithm for supervised neural networks, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 31 (4) (2001) 497–508. doi:10.1109/5326.983933.

[10] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, M. Woźniak, Ensemble learning for data stream analysis: A survey, Information Fusion 37 (2017) 132–156. doi:https://doi.org/10.1016/j.inffus.2017.02.004.
URL https://www.sciencedirect.com/science/article/pii/S1566253516302329

[11] V. Losing, B. Hammer, H. Wersing, Incremental on-line learning: A review and comparison of state of the art algorithms, Neurocomputing 275 (2018) 1261–1274. doi:https://doi.org/10.1016/j.neucom.2017.06.084.
URL https://www.sciencedirect.com/science/article/pii/S0925231217315928

[12] G. A. Carpenter, S. Grossberg, A massively parallel architecture for a self-organizing neural pattern recognition machine, Computer Vision, Graphics, and Image Processing 37 (1) (1987) 54–115. doi:10.1016/s0734-189x(87)80014-2.

[13] A. C. Kakas, D. Cohn, S. Dasgupta, et al., Adaptive Resonance Theory, in: Encyclopedia of Machine Learning, Springer US, 2011, pp. 22–35. doi:10.1007/978-0-387-30164-8_11.

[14] G. A. Carpenter, S. Grossberg, J. H. Reynolds, ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network, Neural Networks 4 (5) (1991) 565–588. doi:10.1016/0893-6080(91)90012-t.

[15] G. Carpenter, S. Grossberg, N. Markuzon, et al., Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps, IEEE Transactions on Neural Networks 3 (5) (1992) 698–713. doi:10.1109/72.159059.

[16] T. Kasuba, Simplified fuzzy ARTMAP, AI Expert 8 (1993) 18–25.

[17] L. E. B. da Silva, I. Elnabarawy, D. C. Wunsch, A survey of adaptive resonance theory neural network models for engineering applications, Neural Networks 120 (2019) 167–203. doi:10.1016/j.neunet.2019.09.012.

[18] M. Vakil-Baghmisheh, N. Pavešić, A Fast Simplified Fuzzy ARTMAP network, Neural Processing Letters 17 (3) (2003) 273–316. doi:10.1023/a:1026004816362.

[19] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01, ACM Press, 2001, pp. 97–106. doi:10.1145/502512.502529.

[20] A. Bifet, R. Gavaldà, Adaptive learning from evolving data streams, in: Advances in Intelligent Data Analysis VIII, Springer Berlin Heidelberg, 2009, pp. 249–260. doi:10.1007/978-3-642-03915-7_22.

[21] H. M. Gomes, A. Bifet, J. Read, et al., Adaptive random forests for evolving data stream classification, Machine Learning 106 (9-10) (2017) 1469–1495. doi:10.1007/s10994-017-5642-8.

[22] C. Manapragada, G. I. Webb, M. Salehi, Extremely Fast Decision Tree, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 1953—-1962. doi:10.1145/3219819.3220005.

[23] J. Z. Kolter, M. A. Maloof, Using Additive Expert Ensembles to cope with Concept Drift, in: In Proceedings of the 22nd International Conference on Machine Learning (ICML-2005), ACM Press, 2005, pp. 449–456.

[24] J. Z. Kolter, M. A. Maloof, Dynamic Weighted Majority: An ensemble method for Drifting Concepts, Journal of Machine Learning Research 8 (91) (2007) 2755–2790.

[25] N. C. Oza, S. J. Russell, Online bagging and boosting, in: T. S. Richardson, T. S. Jaakkola (Eds.), Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, Vol. R3 of Proceedings of Machine Learning Research, PMLR, 2001, pp. 229–236, reissued by PMLR on 31 March 2021.

[26] J. Montiel, J. Read, A. Bifet, T. Abdessalem, Scikit-Multiflow: A multi-output streaming framework, Journal of Machine Learning Research 19 (72) (2018) 1–5.

[27] M. Heusinger, C. Raab, F.-M. Schleif, Passive concept drift handling via variations of learning vector quantization, Neural Computing and Applications (Aug. 2020). doi:10.1007/s00521-020-05242-6.

[28] T. Kohonen, Improved versions of learning vector quantization, 1990 IJCNN International Joint Conference on Neural Networks (1990) 545–550 vol.1.

[29] A. M. Elsayad, Classification of ECG arrhythmia using learning vector quantization neural networks, in: 2009 International Conference on Computer Engineering & Systems, IEEE, 2009, pp. 139–144. doi:10.1109/icces.2009.5383295.

[30] T. Kohonen, M. R. Schroeder, T. S. Huang, Self-Organizing Maps, 3rd Edition, Springer-Verlag, Berlin, Heidelberg, 2001.

[31] M. Pratama, S. G. Anavatti, M. Joo, E. D. Lughofer, pclass: An effective classifier for streaming examples, IEEE Transactions on Fuzzy Systems 23 (2) (2015) 369–386. doi:10.1109/TFUZZ.2014.2312983.

[32] M. Pratama, S. G. Anavatti, J. Lu, Recurrent classifier based on an incremental metacognitive-based scaffolding algorithm, IEEE Transactions on Fuzzy Systems 23 (6) (2015) 2048–2066. doi:10.1109/TFUZZ.2015.2402683.

[33] E. Lughofer, E. Weigl, W. Heidl, C. Eitzinger, T. Radauer, Integrating new classes on the fly in evolving fuzzy classifier designs and their application in visual inspection, Applied Soft Computing Journal 35 (2015) 558–582. doi:10.1016/j.asoc.2015.06.038.

[34] M. Pratama, W. Pedrycz, E. Lughofer, Evolving ensemble fuzzy classifier, IEEE Transactions on Fuzzy Systems 26 (5) (2018) 2552–2567. doi:10.1109/tfuzz.2018.2796099.

[35] I. Škrjanc, J. A. Iglesias, A. Sanchis, D. Leite, E. Lughofer, F. Gomide, Evolving fuzzy and neuro-fuzzy approaches in clustering, regression, identification, and classification: A survey, Information Sciences 490 (2019) 344–368. doi:https://doi.org/10.1016/j.ins.2019.03.060.

[36] E. Lughofer, Extensions of vector quantization for incremental clustering, Pattern Recognition 41 (3) (2008) 995–1011, part Special issue: Feature Generation and Machine Learning for Robust Multimodal Biometrics. doi:https://doi.org/10.1016/j.patcog.2007.07.019.

[37] E. Lughofer, Evolving vector quantization for classification of on-line data streams, in: 2008 International Conference on Computational Intelligence for Modelling Control & Automation (CIMCA 2008), IEEE Computer Society, Los Alamitos, CA, USA, 2008, pp. 779–784. doi:10.1109/CIMCA.2008.47.

[38] J. Kluska, M. Madera, Extremely simple classifier based on fuzzy logic and gene expression programming, Information Sciences 571 (2021) 560–579. doi:https://doi.org/10.1016/j.ins.2021.05.041.

[39] Y. Villuendas-Rey, C. F. Rey-Benguría, Ángel Ferreira-Santiago, O. Camacho-Nieto, C. Yáñez-Márquez, The naïve associative classifier (nac): A novel, simple, transparent, and accurate classification model evaluated on financial data, Neurocomputing 265 (2017) 105–115, new Trends for Pattern Recognition: Theory & Applications. doi:https://doi.org/10.1016/j.neucom.2017.03.085.

[40] J. Huang, C. X. Ling, Using AUC and accuracy in evaluating learning algorithms, IEEE Transactions on Knowledge and Data Engineering 17 (3) (2005) 299–310. doi:10.1109/TKDE.2005.50.

[41] J. G. Moreno-Torres, J. A. Saez, F. Herrera, Study on the impact of partition-induced dataset shift on k-fold cross-validation, IEEE Transactions on Neural Networks and Learning Systems 23 (8) (2012) 1304–1312. doi:10.1109/TNNLS.2012.2199516.

[42] A. D. Aczel, J. Sounderpandian, Statystyka w zarządzaniu, Wydawnictwo Naukowe PWN, Warszawa, 2018.

[43] A. J. Scott, M. Knott, A cluster analysis method for grouping means in the analysis of variance, Biometrics 30 (3) (1974) 507–512.

[44] B. Rosner, R. J. Glynn, M. T. Lee, The wilcoxon signed rank test for paired comparisons of clustered data, Biometrics 62 (1) (2005) 185–192. doi:10.1111/j.1541-0420.2005.00389.x.
URL http://dx.doi.org/10.1111/j.1541-0420.2005.00389.x

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, et al., Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[46] K. J. Lang, M. J. Witbrock, Learning to tell two spirals apart, The 1988 Connectionist Models Summer School (1988). doi:10.13140/2.1.3459.2329.

[47] M. Grbovic, S. Vucetic, Learning vector quantization with adaptive prototype addition and removal, in: 2009 International Joint Conference on Neural Networks, IEEE, 2009, pp. 994–1001. doi:10.1109/ijcnn.2009.5178710.

[48] J. Alcalá-Fdez, A. Fernandez, J. Luengo, et al., KEEL Data-Mining Software Tool: Data set repository, integration of algorithms and experimental analysis framework, Journal of Multiple-Valued Logic and Soft Computing 17:2-3 (2011) 255–287.

[49] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees, Wadsworth International Group, Belmont, CA, 1984.

[50] T. F. Chan, G. H. Golub, R. J. LeVeque, Updating formulae and an pairwise algorithm for computing sample variances, Stanford working paper STAN-CS-79-773 (1979) 1—-22.

[51] N. S. Altman, An introduction to Kernel and Nearest-Neighbor nonparametric regression, The American Statistician 46 (3) (1992) 175–185. doi:10.1080/00031305.1992.10475879.

[52] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2017). arXiv:1412.6980.

[53] R. Tibshirani, T. Hastie, B. Narasimhan, G. Chu, Diagnosis of multiple cancer types by shrunken centroids of gene expression, Proceedings of the National Academy of Sciences 99 (10) (2002) 6567–6572. doi:10.1073/pnas.082099299.

[54] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer, USA, 2008.

[55] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machine, ACM Transactions on Intelligent Systems and Technology 2 (3) (2011) 1–27. doi:10.1145/1961189.1961199.

[56] J. Novakovic, A. Veljovic, C-Support Vector Classification: Selection of kernel and parameters in medical diagnosis, in: 2011 IEEE 9th International Symposium on Intelligent Systems and Informatics, 2011, pp. 465–470. doi:10.1109/SISY.2011.6034373.

[57] J. H. Friedman, Greedy function approximation: A gradient boosting machine, The Annals of Statistics 29 (5) (2001) 1189–1232. doi:10.1214/aos/1013203451.

[58] Y. Shevchuk, et al., Neupy (Version 1.18.5) [Computer software] (2015).
URL http://neupy.com/

[59] B. Galbraith, Adaptive Resonance Theory models [Computer software] (May 2017).
URL https://github.com/AIOpenLab/art

[60] S. Czmil, Python implementation of evolving Vector Quantization for classification of on-line data streams (Version 0.0.2) [Computer software] (2021).
URL https://github.com/sylwekczmil/evq

[61] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive Online Analysis, Journal of Machine Learning Research 11 (52) (2010) 1601–1604.

[62] A. Singh, N. Thakur, A. Sharma, A review of supervised machine learning algorithms, in: 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 1310–1315.

[63] V. Nasteski, An overview of the supervised machine learning methods, HORIZONS.B 4 (2017) 51–62. doi:10.20544/horizons.b.04.1.17.p05.
URL http://dx.doi.org/10.20544/HORIZONS.B.04.1.17.P05

[64] K. Stapor, Evaluating and comparing classifiers: Review, some recommendations and limitations, in: M. Kurzynski, M. Wozniak, R. Burduk (Eds.), Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017, Springer International Publishing, Cham, 2018, pp. 12–21.

[65] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques, 4th Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2016.

[66] S. Czmil, J. Kluska, A. Czmil, CACP: Classification algorithms comparison pipeline, SoftwareX 19 (2022) 101134. doi:https://doi.org/10.1016/j.softx.2022.101134.

[67] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, A. Bifet, River: machine learning for streaming data in python, Journal of Machine Learning Research 22 (110)

(2021) 1–8.
URL http://jmlr.org/papers/v22/20-1380.html

[68] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant, Array programming with NumPy, Nature 585 (2020) 357–362. doi:10.1038/s41586-020-2649-2.

[69] The pandas development team, pandas-dev/pandas: Pandas (Feb. 2020). doi:10.5281/zenodo.3509134.
URL https://doi.org/10.5281/zenodo.3509134

[70] J. D. Hunter, Matplotlib: A 2d graphics environment, Computing in Science & Engineering 9 (3) (2007) 90–95. doi:10.1109/MCSE.2007.55.

[71] J. G. Moreno-Torres, J. A. Saez, F. Herrera, Study on the impact of partition-induced dataset shift on k-fold cross-validation, IEEE Transactions on Neural Networks and Learning Systems 23 (8) (2012) 1304–1312. doi:10.1109/tnnls.2012.2199516.
URL https://doi.org/10.1109/tnnls.2012.2199516

Streszczenie

Niniejsza rozprawa dotyczy problematyki algorytmów nadzorowanego inkrementalnego uczenia maszynowego oraz oceny jakości ich klasyfikacji. Głównym celem pracy było opracowanie i implementacja programowa i sprzętowa nowego inkrementalnego algorytmu klasyfikacji danych, który w klasie algorytmów płytkich okazał się nie gorszy od dotychczas stosowanych.

W ramach pracy zaproponowano nowy algorytm przyrostowy klasyfikacji danych (SEVQ) z minimalną liczbą parametrów nastrajanych. Przeprowadzono wszechstronne badania porównawcze nowego klasyfikatora z uwzględnieniem dużej liczby zbiorów danych, wielu dotychczas stosowanych algorytmów płytkich (zarówno inkrementalnych, jak i nieinkrementalnych) oraz wielu wskaźników jakości klasyfikacji. Zastosowano grupowanie za pomocą algorytmu Scotta–Knotta oraz test Wilcoxona w celu ulokowania nowego algorytmu SEVQ na odpowiedniej pozycji wśród algorytmów dotychczas stosowanych.

Dokonano implementacji programowej nowego algorytmu w języku Python oraz sprzętowej na układach FPGA. Porównano obie implementacje z uwzględnieniem wskaźników jakości klasyfikacji danych, czasu uczenia i czasu trwania klasyfikacji. Opracowano także uniwersalne narzędzie do wspomagania procesu wszechstronnego testowania algorytmów klasyfikacji danych, spełniające istotne wymagania w zakresie badań benchmarkowych, wyposażone w graficzny interfejs użytkownika.

Przeprowadzone badania wykazały, że zaproponowany algorytm SEVQ, pomimo swojej prostoty, nie jest gorszy od większości dotąd stosowanych algorytmów płytkich. Implementacja sprzętowa algorytmu w układzie FPGA pozwoliła na znaczne przyspieszenie jego działania, szczególnie w kontekście obsługi dużych zbiorów danych, przy jednoczesnym zachowaniu porównywalnej efektywności klasyfikacji.

Słowa kluczowe: uczenie nadzorowane, klasyfikatory płytkie, algorytm inkrementalny, implementacja sprzętowa, ocena jakości klasyfikacji

Abstract

This dissertation focuses on the supervised incremental machine learning algorithms and the evaluation of their classification quality. The main objective of the work was to develop and implement a new incremental data classification algorithm, both in software and hardware, which proved to be no worse than the existing algorithms in the class of shallow algorithms.

As part of the work, a new incremental data classification algorithm (SEVQ) with a minimal number of tunable parameters was proposed. Comprehensive comparative studies of the new classifier were conducted, considering a large number of datasets, many previously used shallow algorithms (both incremental and non-incremental), and various classification quality indicators. Grouping using the Scott-Knott algorithm and the Wilcoxon test was applied to rank the new SEVQ algorithm among the previously used algorithms.

The new algorithm was implemented in software using the Python language and in hardware on FPGA devices. Both implementations were compared, taking into account data classification quality indicators, learning time, and classification duration. A universal tool for supporting the process of comprehensive testing of data classification algorithms was also developed, meeting essential requirements for benchmark research and equipped with a graphical user interface.

The conducted research demonstrated that the proposed SEVQ algorithm, despite its simplicity, is no worse than most of the previously used shallow algorithms. The hardware implementation of the algorithm on an FPGA allowed for a significant acceleration of its operation, particularly in the context of handling large datasets, while maintaining comparable classification effectiveness.

Keywords: supervised learning, shallow classifiers, incremental algorithm, hardware implementation, classification quality evaluation