



**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**

**WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI  
KATEDRA TELEKOMUNIKACJI**

**PRACA DYPLOMOWA INŻYNIERSKA**

**OPRACOWANIE SYSTEMU IDENTYFIKACJI UŻYTKOWNIKÓW SIECI INTERNET  
*SYSTEM ALLOWING TO IDENTIFY THE INTERNET USERS***

Autor:

Adrian Karbowiak

Kierunek studiów:

Teleinformatyka

Opiekun pracy:

Dr inż. Marcin Niemiec

Kraków, 2016

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godność studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwany dalej „sądem koleżeńskim”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście, samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

.....

# **Spis treści**

Spis rysunków .....	1
1. Wstęp.....	4
2. Opis systemu .....	5
2.1.Witryna internetowa.....	6
2.2.Baza danych .....	7
2.3.Aplikacja klasyfikująca użytkowników .....	8
2.3.1. Przygotowanie wejść klasyfikatora .....	10
2.3.2. Trening klasyfikatora .....	11
2.3.3. Test klasyfikatora.....	12
3. Implementacja .....	13
3.1.Witryna internetowa.....	13
3.2.Bazy danych .....	17
3.3.Aplikacja klasyfikująca użytkowników .....	19
3.3.1. Przygotowanie wejść klasyfikatora .....	22
3.3.2. Trening klasyfikatora .....	28
3.3.3. Test klasyfikatora .....	31
4. Weryfikacja aplikacji .....	32
4.1.Porównanie programów rejestrujących ruchy urządzeń wskazujących .....	32
4.2.Analiza możliwości odseparowania próbek.....	34
4.3.Test wydajności aplikacji .....	48
5. Podsumowanie .....	48
Bibliografia .....	50
Dodatek A: Spis zawartości w dołączonej płycie CD.....	52

## Spis rysunków

Rys. 1. Schemat systemu .....	5
Rys. 2. Wygląd strony www systemu .....	6
Rys. 3. Schemat witryny internetowej .....	7
Rys. 4. Tabele bazy danych.....	7
Rys. 5. Schemat aplikacji klasyfikującej użytkowników .....	8
Rys. 6. Klasyfikator binarny .....	9
Rys. 7. Prawidłowa (czarna) i błędna (szara) hiperplaszczyzna .....	9
Rys. 8. Kąt AB .....	11
Rys. 9. Kąt ABC.....	11
Rys. 10. $ B AC / AC $ .....	11
Rys. 11. Uproszczona wizualizacja klasyfikatora o miękkim marginesie .....	12
Rys. 12. Odwzorowanie punktów w wyższym wymiarze w celu lepszego oddzielenia grup .....	12
Rys. 13. Pliki ze skryptami witryny internetowej .....	13
Rys. 14. Fragment kodu init.js włączający licznik czasu i uruchamiający skrypt PHP .....	14
Rys. 15. Obserwatory zdarzeń po stronie klienta.....	15
Rys. 16. Funkcje safeDelay i zwracająca \$.active .....	15
Rys. 17. Działanie bufora i przesyłanie danych z pomocą jQuery AJAX .....	16
Rys. 18. Inicjalizacja użytkownika na serwerze .....	16
Rys. 19. Zapis informacji o użytkowniku do bazy danych .....	17
Rys. 20. Odbiór i zapis do bazy danych ruchów urządzeń wskazujących .....	17
Rys. 21. Tabela general .....	18
Rys. 22. Tabela blocks .....	18
Rys. 23. Tabela mousecurves .....	19
Rys. 24. Tabela curveparameters .....	19
Rys. 25. Struktura klas aplikacji klasyfikującej .....	19
Rys. 26. Graficzny interfejs użytkownika .....	20
Rys. 27. Fragment kodu włączającego funkcję wybraną w Gui .....	21
Rys. 28. Kod wył. automatyczne zatwierdzanie zmian w bazie danych. Dostęp do Gui .....	22
Rys. 29. Akceptowane i odrzucane krzywe .....	23
Rys. 30. Fragment kodu wyznaczającego krzywe .....	23

Rys. 31. Obliczanie kąta ABC .....	24
Rys. 32. Obliczanie stosunku odległości punktu B od prostej AC do długości odcinka AC.....	25
Rys. 33. Prążki kąta AB .....	25
Rys. 34. Prążki kąta ABC.....	25
Rys. 35. Prążki $ B AC / AC $ .....	25
Rys. 36: Główna pętla klasy CalculateBlocks .....	26
Rys. 37. Funkcja obliczająca częstotliwość występowania metryk .....	27
Rys. 38. Konstruowanie dystrybuanty dla kąta AB .....	27
Rys. 39. Przykład próbki przedstawionej w formie histogramu .....	28
Rys. 40. Część kodu z funkcją generującą losowe wektory i pobierającą próbki z bazy danych..	28
Rys. 41. Funkcja selekcjonująca próbki .....	29
Rys. 42. Funkcja formatująca próbki dla LIBSVM .....	29
Rys. 43. Kod zapełniający próbками odpowiednie listy z dowiązaniami.....	29
Rys. 44. Konwersja list z próbками do plików tekstowych .....	30
Rys. 45. Fragment kodu uruchomiającego metodę run obiektu klasy SVM.....	30
Rys. 46. Fragment kodu przedstawiający pętlę metody gridSearch .....	31
Rys. 47. Wczytanie pliku z danymi o wytrenowanym użytkowniku i wybór próbek testowych ..	31
Rys. 48. Fragment kodu uruchamiający główną metodę obiektu klasy SVM dla testu.....	32
Rys. 49. Porównanie wykresów punktów zebranych autorską aplikacją i RUI.....	33
Rys. 50. Liczebność próbek w bazie danych.....	34
Rys. 51. Wykresy uśrednionych dystrybuant.....	35
Rys. 52. Wykresy dystrybuant użytych w treningu klasyfikatora.....	36
Rys. 53. Wyraźnie odseparowane zbiory dystrybuant użytkownika niebieskiego i pomarańczowego.....	36
Rys. 54. Wykresy próbek treningowych użytkowników czarnego i zielonego .....	37
Rys. 55. Wykresy próbek treningowych użytkowników czarnego i pomarańczowego.....	37
Rys. 56. Wykresy próbek treningowych użytkowników czarnego i niebieskiego.....	38
Rys. 57. Wykresy próbek treningowych użytkowników zielonego i niebieskiego.....	38
Rys. 58. Trudno rozróżnialne wykresy użytkowników pomarańczowego i zielonego .....	39
Rys. 59. Trudno odróżnialne wykresy użytkownika pomarańczowego i zielonego.	
Próbki 1 - 24 .....	40
Rys. 60. Trudno odróżnialny wykres użytkownika pomarańczowego i zielonego.	
Próbki 25 - 49 .....	40

Rys. 61. Zbiór treningowy użytkownika pomarańczowego .....	41
Rys. 62. Próbki testowe użytkownika pomarańczowego na jego zbiorze treningowym .....	42
Rys. 63. Zbiór treningowy użytkownika zielonego .....	42
Rys. 64. Próbki testowe użytkownika zielonego na zbiorze treningowym.....	43
Rys. 65. Próbki testowe użytkownika zielonego vs zbiór treningowy użytkownika zielonego ....	43
Rys. 66. Próbki testowe użytkownika pomarańczowego vs zbiór treningowy użytkownika zielonego .....	44
Rys. 67. Próbki testowe użytkownika czarnego vs zbiór treningowy użytkownika zielonego.....	44
Rys. 68. Próbki testowe użytkownika niebieskiego vs zbiór treningowy użytkownika zielonego .....	44
Rys. 69. Próbki testowe użytkownika pomarańczowego vs zbiór treningowy użytkownika pomarańczowego.....	45
Rys. 70. Próbki testowe użytkownika zielonego vs zbiór treningowy użytkownika pomarańczowego.....	45
Rys. 71. Próbki testowe użytkownika czarnego vs zbiór treningowy użytkownika pomarańczowego.....	45
Rys. 72. Próbki testowe użytkownika niebieskiego vs zbiór treningowy użytkownika pomarańczowego.....	46
Rys. 73. Próbki treningowe użytkownika czarnego .....	46
Rys. 74. Próbki testowe użytkownika czarnego na jego zbiorze treningowym .....	46
Rys. 75. Próbki treningowe użytkownika niebieskiego .....	47
Rys. 76. Próbki testowe użytkownika niebieskiego na jego zbiorze treningowym .....	47
Rys. 77. Tabela wyników dopasowania wszystkich próbek test. do zbiorów treningowych.....	47

## 1. Wstęp

W obecnych czasach korzystanie z serwisów internetowych jest powszechną praktyką. Niestety nie każdy użytkownik jest świadomy jak wiele informacji o swojej tożsamości ujawnia w sieci Internet. Niektóre mechanizmy pobierania danych są jawne i użytkownicy są o nich informowani. Przykładem może być rejestracja klientów w sklepie internetowym. Jednak w wielu przypadkach serwisy internetowe nie powiadamiają użytkowników o analizowaniu prywatnych danych. Taką wiedzę można nabyć np. dopiero po analizie kodu źródłowego strony internetowej.

Istnieje bardzo wiele takich mechanizmów, które różnią się między sobą rodzajem zbieranych danych. Jedne koncentrują się na zapisie informacji o używanym sprzęcie, oprogramowaniu oraz czynnościach wykonywanych za ich pomocą jak np. popularny Google Analytics [1]. Z kolei inne analizują i zapisują indywidualne cechy użytkownika.

Rozwój technik tworzenia aplikacji internetowych, w szczególności AJAX (*ang. Asynchronous JavaScript and XML*) pozwolił na efektywne wykorzystanie metod rozróżniania osób, w zależności od sposobów w jaki używają klawiatur, myszek komputerowych czy gładzików [2]. Dane o ruchach kurSORA mogą być szybko gromadzone podczas standardowego przeglądania strony internetowej, posiadającej odpowiedni skrypt. Oznacza to, że istnieje szansa identyfikacji osoby odwiedzającej taką stronę, pomimo prób ukrycia tożsamości, np. używając przeglądarki TOR (*ang. The Onion Router*) lub innego komputera. Co więcej, mechanizmy te mogą zostać skutecznie ukryte, zatem niewiele osób będzie miało świadomość ich istnienia.

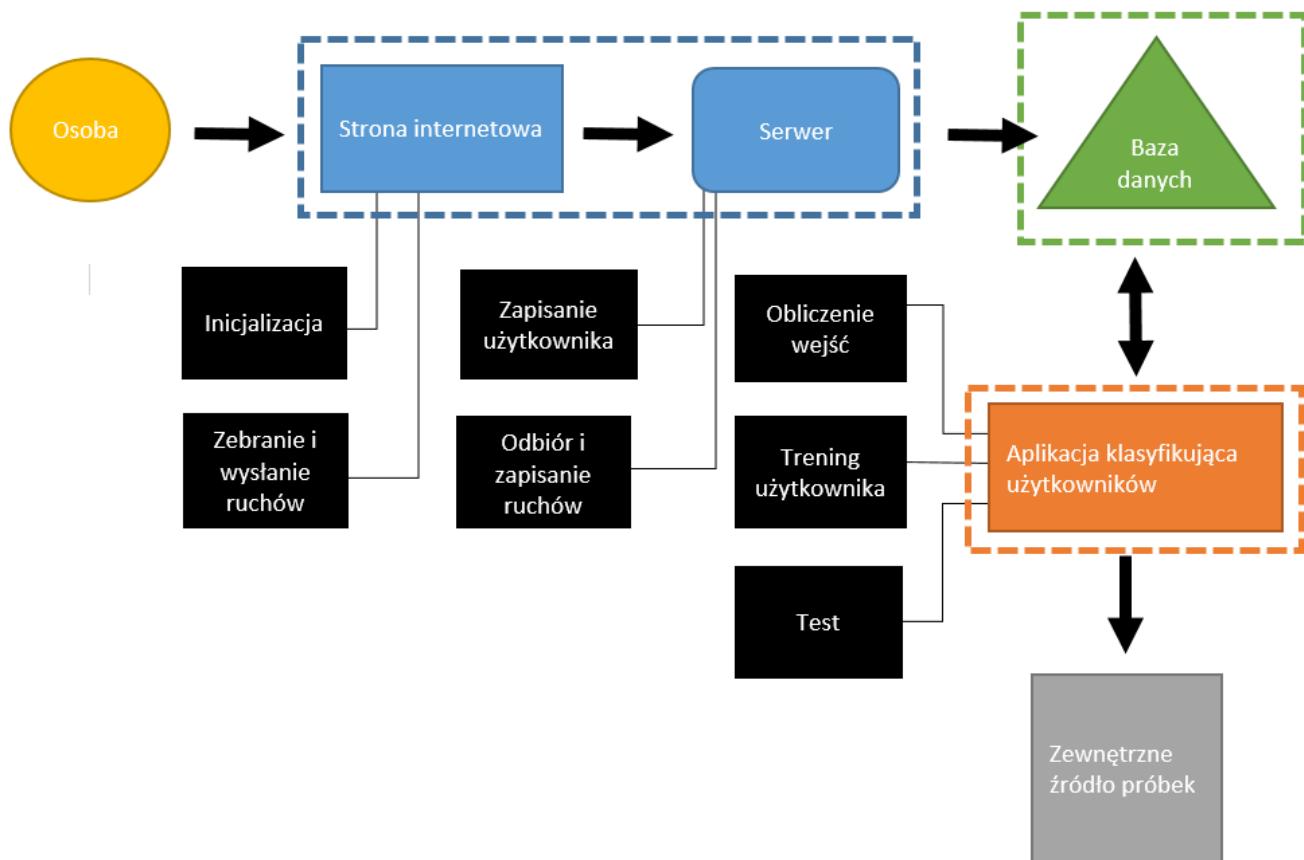
Niniejsza praca opisuje system, którego zadaniem jest identyfikacja użytkownika odwiedzającego specjalnie przygotowaną witrynę. Składa się ze strony internetowej, skryptów pobierających dane o użytkowniku i zapisujących je w bazie danych oraz programu klasyfikującego osoby na podstawie zebranych od nich informacji o położeniu, akcji i czasie akcji kurSORA. Gotowy program miał spełnić trzy najważniejsze cele tj. szybkie działanie, niezawodność rejestracji danych oraz wysoką skuteczność identyfikacji. Zastosowano w tym celu rozwiązania wykorzystujące najnowsze standardy W3C (*ang. World Wide Web Consortium*) oraz potężny algorytm maszyny wektorów wspierających (*ang. support vector machine*), będący techniką uczenia maszynowego.

## 2. Opis Systemu

Rozdział ten zawiera opis systemu identyfikacji użytkowników oraz informacje o technologiach jakie zostały wykorzystane do jego budowy. Ogólny schemat działania systemu został przedstawiony poniżej.

- Użytkownik odwiedza stronę internetową. Skrypty zbierają odpowiednie informacje o użytkowniku. Informacje te są zapisywane w bazie danych.
- Aplikacja klasyfikująca użytkowników pobiera z bazy danych informacje o ruchach urządzeń wskazujących, wyznacza prawidłowe krzywe i tworzy wejścia do maszyny wektorów nośnych. Dodatkowo, aplikacja pozwala wytrenować profil użytkownika oraz sprawdzić zgodność próbek z wytrenowanymi wcześniej profilami.

Kolejne podrozdziały opisują szczegółowo funkcjonalności istotnych części systemu, bez szczegółów implementacji. Rysunek 1 przedstawia jego schemat.



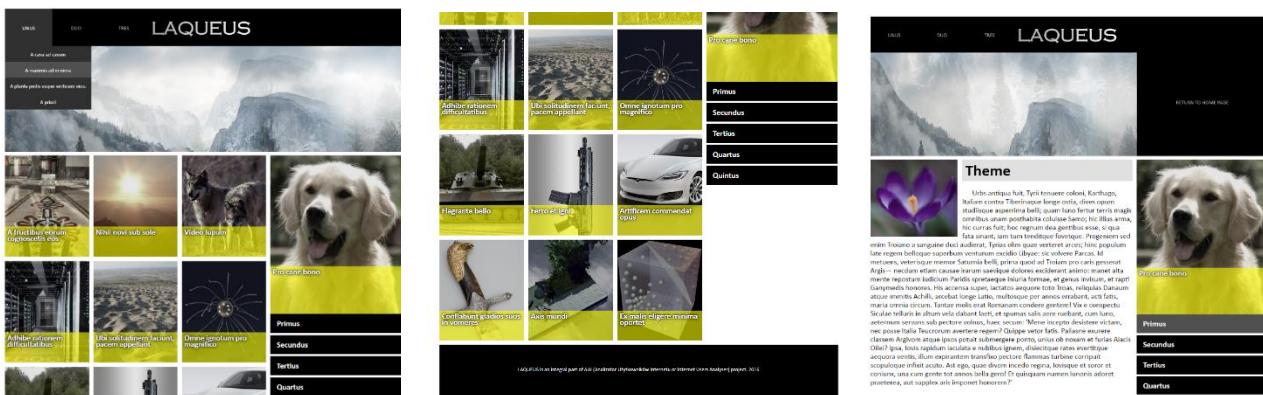
Rys. 1. Schemat systemu.

## 2.1 Witryna internetowa

Prosty serwis internetowy złożony jest ze strony www zawierającej skrypty rejestrujące oraz części serwerowej, mieszczącej skrypt inicjalizujący użytkowników i obsługujący bazę danych. Schemat prezentujący strukturę serwisu umieszczono na rysunku 3. Każdy blok, prócz pierwszego od góry, jest składnikiem odpowiedniego bloku usytuowanego poziom wyżej.

Wygląd witryny zaprojektowano tak, aby upodobnić ją do współczesnych portali informacyjnych (Rysunek 2). W tym celu użyto HTML5 (ang. *HyperText Markup Language 5*) i CSS3 (ang. *Cascading Style Sheets 3*) [3]. Podczas każdorazowych odwiedzin, skrypt inicjalizujący uruchomia licznik czasu oraz część kodu PHP po stronie serwera, która nadaje użytkownikowi unikalny identyfikator sesji. Dzięki niemu możliwe jest dalsze prawidłowe zbieranie danych przy zmianie podstron na witrynie, bez którego rozróżnianie odwiedzających nie byłoby możliwe. Ponadto, skrypt sprawdza czy baza danych zarządzana systemem MariaDB [4] istnieje – jeśli nie, tworzona jest nowa. Dodatkowo gromadzone są informacje o odwiedzającym takie jak czas jego wizyty, adres IP lub wersja przeglądarki.

Skrypt odpowiedzialny za rejestrację ruchów kursora urządzenia wskazującego korzysta z jQuery. Odpowiednie obserwatory zdarzeń (ang. *event listeners*) reagują na ruchy i kliknięcia myszy komputerowej. Każde zarejestrowane zdarzenie jest zapisywane w postaci czteroelementowego zbioru, zawierającego współrzędną x, współrzędną y, czas oraz rodzaj akcji (kliknięcie lub ruch). Zbiory te przechowywane są w buforze, który w momencie przepelenienia jest opróżniany, a zgromadzone dane zostają wysłane do serwera w postaci JSON (ang. *JavaScript Object Notation*). Komunikacja z serwerem odbywa się za pomocą zapytań AJAX, z użyciem jQuery.



Rys. 2. Wygląd strony www.

# Witryna internetowa



Rys. 3. Schemat witryny internetowej.

## 2.2 Baza danych

Baza danych stanowi część wspólną dla witryny internetowej i aplikacji klasyfikującej użytkowników. Za uzupełnienie tabel dot. danych o użytkowniku i punktów zebranych przez rejestrator urządzeń wskazujących odpowiada witryna internetowa lub aplikacja klasyfikująca użytkowników. Pozostałe używane są tylko przez tą drugą. Zdecydowano się na zastosowanie bazy danych MariaDB, gdyż jest alternatywną wersją (*ang. fork*) popularnego i niezawodnego MySQL, opartą o licencję GPL. Schemat na rysunku 4 przedstawia z jakich tabel korzysta. Bloki z pomarańczową obwódką dotyczą tych używanych przez aplikację klasyfikującą, natomiast kolor niebieski właściwy jest dla tabel, które wykorzystuje witryna internetowa.



Rys. 4. Tabele bazy danych.

### 2.3 Aplikacja klasyfikująca użytkowników

Aplikacja została napisana w języku JAVA [5], w środowisku *IntelliJ IDEA 14.1.5/2016.1.1* [6], z użyciem łącza do bazy danych JDBC [7] (*ang. Java DataBase Connectivity*). Wykorzystuje bibliotekę LIBSVM [8] (*ang. A Library for Support Vector Machines*), dzięki której możliwe jest korzystanie z maszyny wektorów wspierających (*ang. Support Vector Machine, w skrócie SVM*). Praca w tym programie jest wygodna i intuicyjna, za sprawą interfejsu graficznego wykonanego z użyciem biblioteki graficznej Swing [9]. Ostatecznie o wyborze tej technologii zadecydowała niezależność wirtualnej maszyny (*ang. Java Virtual Machine*) od platform oraz pomocnicze klasy dla języka JAVA w LIBSVM.

Program posiada trzy główne warianty:

- przygotowanie wejść (czyli zbioru wektorów wejściowych) do maszyny wektorów nośnych,
- trening profilu użytkownika, generujący wzorzec ruchów danego użytkownika,
- test próbek czyli porównanie ich ze wzorcem.

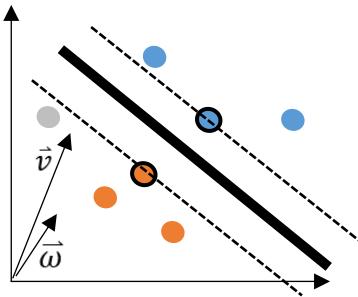
Mniej istotnymi opcjami są załadowanie do bazy danych ruchów urządzeń wskazujących z zewnętrznego źródła oraz usunięcie wszystkich jej rekordów. Istotne komunikaty wyświetlane są w specjalnym oknie, wbudowanym w interfejs graficzny. Schemat na rysunku 5 prezentuje funkcjonalność aplikacji. Każdy blok, prócz pierwszego od góry, jest składnikiem odpowiedniego bloku usytuowanego poziom wyżej.



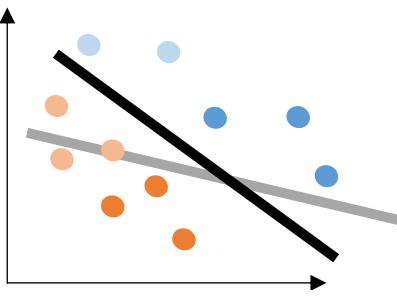
Rys. 5. Schemat aplikacji klasyfikującej użytkowników.

Najważniejszym elementem aplikacji klasyfikującej użytkowników jest kod algorytmu maszyny wektorów wspierających, zaimplementowany w dołączonej bibliotece LIBSVM. To on umożliwia rozróżnianie ruchów urządzeń wskazujących użytkowników. Jest jednym z wielu dostępnych klasyfikatorów, spośród których wymienić można sieci neuronowe, lasy losowe i drzewa klasyfikacyjne. O jego wyborze przesądziło to, że dobrze radzi sobie z małą liczbą próbek uczących, które posiadają dużo atrybutów wejściowych. Pozwala także uzyskać zadowalającą skuteczność klasyfikacji przy stosunkowo niedużej konfiguracji parametrów.

Maszyna wektorów wspierających jest w najprostszym wariantie liniowym klasyfikatorem binarnym. Hiperpłaszczyzna, która w przestrzeni dwuwymiarowej ma postać linii, separuje dwie różne grupy próbek. Na rysunku 6 niebieskie i pomarańczowe punkty symbolizują próbki, a linia hiperpłaszczyznę. Zadaniem klasyfikatora jest podjęcie decyzji, po której ze stron linii powinien znaleźć się szary punkt. Równoznaczne jest to ze stwierdzeniem czy iloczyn skalarny wektorów  $\vec{v} \cdot \vec{\omega}$  jest większy lub równy od pewnej stałej  $C$ , gdzie  $\vec{v}$  to wektor poprowadzony do szarego punktu, a  $\vec{\omega}$  wektor prostopadły do hiperpłaszczyzny.



Rys. 6. Klasyfikator binarny.



Rys. 7. Prawidłowa (czarna) i błędna (szara) hiperpłaszczyzna.

Zanim klasyfikacja będzie możliwa, należy odpowiednio odseparować linią dwie różnice się między sobą grupy próbek. Hiperpłaszczyzna powinna przebiegać możliwie daleko od obu grup próbek. Jak widać na rysunku 7, szara linia dobrze izoluje punkty pomarańczowe i niebieskie, ale niewłaściwie jasnopomarańczowe, które symbolizują nowo dodawane próbki. Należy więc tak wyznaczyć hiperpłaszczyznę, by jej odległość od najbliższych punktów z obu grup była możliwie maksymalna. Na rysunku 6 punkty te (zwane wektorami wspierającymi) mają czarny kontur, natomiast odległość między przechodzącymi przez nie przerywanymi liniami nazywana jest marginesem. Wartości wektora  $\vec{\omega}$  i stałej  $C$  mają więc być ustalone tak, by dla danego zbioru próbek margines był możliwie największy. Wykonując odpowiednie operacje na równaniu hiperpłaszczyzny  $\vec{\omega} \cdot \vec{x} + b = 0$ , gdzie  $\vec{x}$  oznacza próbkę niebieską lub pomarańczową, natomiast

$b = -C$ , można dowieść, że szerokość marginesu wynosi  $\frac{2}{\|\vec{\omega}\|}$ . Maksymalizacja marginesu polega więc na minimalizacji  $\|\vec{\omega}\|$ , co sprowadza się do rozwiązania zadania programowania kwadratowego (*ang. quadratic programming*).

Klasyfikator wymaga poprawnie sformatowanego wejścia czyli zbioru wektorów (próbek), których wartościami są atrybuty wejściowe. Stanowią one liczbową reprezentację danych, które mają być poddane klasyfikacji. W przypadku tej pracy, są to wartości dystrybuant utworzonych z liczb będących częstotliwościami występowania pewnych metryk, opisanych szczegółowo w rozdziale 2.3.1. Dystrybuantą danego rozkładu w pewnym punkcie jest prawdopodobieństwo, że zmienna losowa (funkcja, która przypisuje liczby zdarzeniom elementarnym) przyjmie wartość nie większą niż ten punkt [10]. Istotnymi liczbami dla klasyfikatora są te dotyczące dyskretnych zmiennych losowych. Użyte w niniejszej pracy wykresy dystrybuant nie są więc wykresami funkcji schodkowych – ich punkty połączone są bezpośrednio liniami. Z powodzeniem można by je zastąpić np. histogramami, lecz uznano, że wykresy niemalejących funkcji są czytelniejsze. Utworzone z dystrybuant wektory wejściowe są łatwe do porównania – każdy atrybut wejściowy na danej pozycji dotyczy częstości występowania tej samej metryki. Wektory mają stałą długość, spełniają więc jeden z wymogów implementacji maszyny wektorów wspierających LIBSVM. Wartości dystrybuant mieścią się w przedziale [0,1] więc dane są automatycznie skalowane, zgodnie z zaleceniem w dokumentacji [8]. Prace [11] i [12] opisują systemy w których z powodzeniem wykorzystano histogramy jako wejścia. Ogólny opis klasyfikatora sporządzono w oparciu o materiały [13] i [14].

### 2.3.1 Przygotowanie wejść klasyfikatora

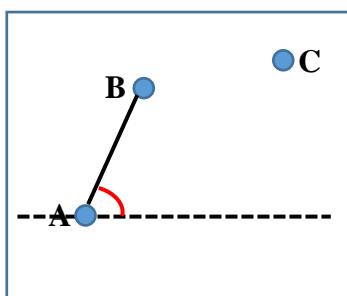
Sposób postępowania z zarejestrowanymi ruchami urządzeń wskazujących jest podobny do rozwiązania przedstawionego w artykule [15]. Najpierw, spośród wszystkich ruchów, wybierane są krzywe o odpowiedniej minimalnej długości. Składają się one z punktów, które zostały zarejestrowane w odstępach czasu mniejszych niż ustalony limit. Każda krzywa kończy się punktem, który zapisany został w wyniku kliknięcia, a nie np. ruchu myszą.

Następnie, dla wszystkich trzech kolejnych punktów A, B, C w każdej krzywej liczone są wartości zwane metrykami:

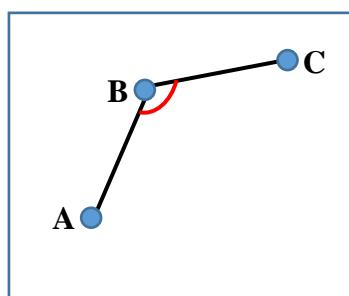
- Kąt AB (Rysunek 8): kąt pomiędzy prostą przechodzącą przez punkt A, równoległą do osi poziomej kartezjańskiego układu współrzędnych, a odcinkiem AB,
- Kąt ABC (Rysunek 9): kąt pomiędzy odcinkami AB i BC,
- Stosunek odległości punktu B do odcinka AC do długości odcinka AC (Rysunek 10).

Metryki są wartościami, które zgromadzone w odpowiednio dużej liczbie, umożliwiają dostrzeżenie różnic między ruchami urządzeń wskazujących różnych użytkowników.

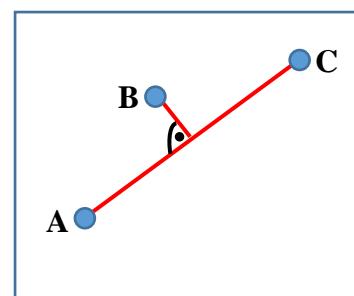
Kolejnym krokiem jest podział wszystkich metryk. Ponieważ każda posiada informację o tym, z której krzywej została policzona, formowane są zbiory składające się z metryk z N różnych krzywych. Aby zapobiec nadmierнемu dopasowaniu, występującego w sytuacji, gdy wektor wejściowy ma zbyt dużo parametrów w stosunku do liczby wszystkich wektorów wejściowych, zdecydowano się połączyć metryki w przedziały (prążki). W każdym zbiorze liczone są więc częstotliwości występowania prążków, a z nich dystrybuanty. Zmiennymi losowymi dystrybuant są prążki, a zborem wartości liczby z przedziału [0,1].



Rys. 8. Kąt AB.



Rys. 9. Kąt ABC.



Rys. 10.  $|B|AC||/|AC|$ .

### 2.3.2 Trening klasyfikatora

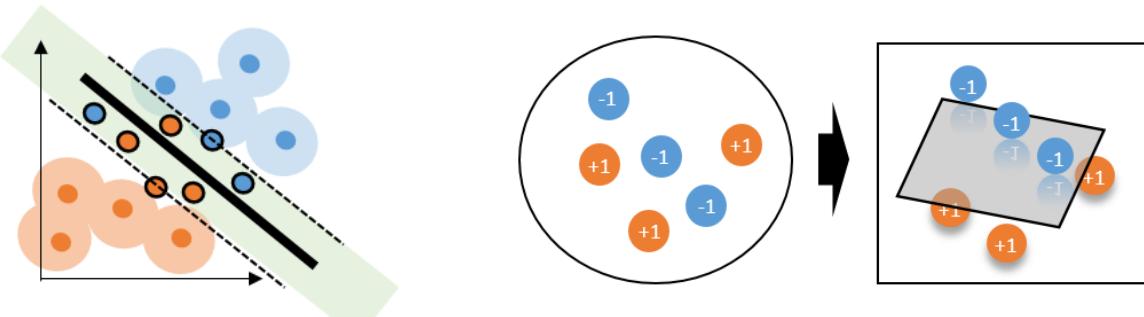
Użyty w pracy klasyfikator wykorzystuje uczenie maszynowe zwane nadzorowanym. Wymusza je wybór strategii `one-vs-all` czyli przyporządkowania podczas treningu próbek jednej osoby do klasy negatywnej, a pozostałych do pozytywnej. Nadzór polega na dodaniu do wektorów wejściowych (próbek) atrybutu wyjściowego będącego liczbą -1 lub 1, oznaczającą odpowiednio klasę negatywną i pozytywną.

Dla uproszczenia, rozpoznany może być jeden spośród czterech użytkowników. Do losowo wybranych N próbek danej osoby, dołączane jest 3N próbek reszty. Maszyna jest dwuklasowym klasyfikatorem w którym użytkownik trenowany ma atrybut wyjściowy -1, a pozostały 1.

Zastosowano popularną funkcję jądra RBF (*ang. Gaussian Radial Basis Function*), która jest co najmniej równie skuteczna, jak liniowe jądro (*ang. linear kernel*), pod warunkiem poprawnej konfiguracji [4]. Jeżeli próbki nie są liniowo separowalne, funkcja jądra transformuje je do takiej postaci, aby były. Jak prezentuje rysunek 12, wiąże się to często ze zmianą wymiaru na wyższy. Niebieskich i pomarańczowych próbek nie da się oddzielić w okręgu, ale szara płaszczyzna bez problemu izoluje próbki w obrysie kwadratu.

Parametry  $C$  i  $\Gamma$  określają odpowiednio rodzaj marginesu i skalę oddziaływanego pojedynczej próbki. Na rysunku 11 zaznaczono efekt parametru  $\Gamma$  kolorami jasnoniebieskim i jasnopomarańczowym, a  $C$  jasnozielonym. Są one wyszukiwane samodzielnie zaimplementowaną metodą przeszukiwania siatki (*ang. grid search*) z zagnieżdżonym sprawdzianem krzyżowym (*ang. cross validation*). Wybierany jest możliwie niski parametr  $C$ , aby uzyskać miękkie marginesy. Podczas wyznaczania hiperpłaszczyzny minimalizowany jest wtedy oprócz  $\|\vec{\omega}\|$  błąd klasyfikatora  $\sum_{i=1}^n \xi_i$  pomnożony przez  $C$ . Pozwala to zignorować próbki nie pasujące do ogólnego profilu danego użytkownika, a wynikające np. z drobnych różnic w ruchach lub chwilowym, rzadkim zachowaniem. Wybór właściwego marginesu zależy więc od położenia przyszłych próbek, co należy przewidzieć. Jak widać na rysunku 11, wektory wspierające mogą znaleźć się wtedy wewnętrz marginesu.

Proces trenowania kończy się wraz z utworzeniem profilu użytkownika, stanowiącego wzorzec z którym porównywane są próbki testowe. Opis zastosowanego rodzaju klasyfikatora sporządzono w oparciu o materiały [16], [17] i [18].



**Rys. 11.** Uproszczona wizualizacja klasyfikatora o miękkim marginesie.

**Rys. 12.** Odwzorowanie punktów w wyższym wymiarze w celu lepszego oddzielenia grup.

### 2.3.3 Test klasyfikatora

Aby sprawdzić czy próbki testowe należą do użytkownika, należy je porównać z jego profilem. Próbki testowe oznaczone są atrybutem wyjściowym o wartości -1, tak samo jak osoba dla której wytrenowano profil. Profil zawiera wzorzec określający jak hiperpłaszczyna rozdziela jego własne próbki oraz pozostałych osób. Próbki testowe muszą zostać sklasyfikowane po jednej z jej stron. Maszyna wektorów wspierających określa na tej podstawie skuteczność dopasowania. Należy jednak pamiętać by poprawnie zinterpretować ten wynik. Jeżeli próbki testowe nie należą w rzeczywistości do wybranego użytkownika, klasyfikator powinien zwrócić, w idealnym przypadku, skuteczność równą 0%. W przeciwnym razie powinno to być 100%.

### 3. Implementacja

Rozdział ten zawiera szczegółowy opis systemu. Przedstawiony został sposób implementacji systemu wraz z fragmentami kodu najbardziej istotnych części aplikacji.

### **3.1 Witryna internetowa**

Strona internetowa nie używa żadnego systemu zarządzania treścią. Istnieje natomiast możliwość skorzystania z części serwerowej i bazy danych poprzez XAMPP (*ang. cross-platform Apache, MySQL, PHP, Perl*). Rysunek 13 przedstawia pliki składające się na witrynę. Folder o nazwie data zawiera mniej istotne elementy, podstronę oraz foldery: ze skryptami CSS i zasobami.

	data	23.05.2016 16:25	Folder plików
	clicksPost	24.05.2016 03:51	Plik JS
	cursorCatcher	24.05.2016 04:34	Plik JS
	index	24.05.2016 02:09	Opera Web Docu...
	index	19.04.2016 19:09	Plik PHP
	init	24.05.2016 01:57	Plik JS
	init	31.05.2016 00:37	Plik PHP
	jquery-1.12.1.min	08.03.2016 02:54	Plik JS
	library	23.03.2016 03:54	Plik PHP
	MouseMovementToDatabase	23.03.2016 00:26	Plik PHP
	SessionHandlerInterface	22.03.2016 01:59	Plik PHP

**Rys. 13.** Pliki ze skryptami witryny internetowej.

Skrypt `init.js` uruchomiany jest podczas każdego wejścia na stronę główną lub podstronę. Rysunek 14 przedstawia kod metody uruchamiający skrypt inicjalizujący użytkownika po stronie serwera. Kluczowe zastosowanie ma metoda `performance.now`, która aktywuje pierwszy znacznik czasu, aktualizowany przy ruchach lub kliknięciach urządzenia wskazującego. Ogromną jej zaletą jest odmierzanie czasu z dokładnością większą niż milisekundy [19] co pozwala sortować zebrane punkty bez ryzyka, że kilku z nich zostanie przyporządkowany ten sam czas.

pomimo różnych współrzędnych. Dzięki temu nie jest konieczne by wpisy do bazy danych odbywały się synchronicznie.

```
var pageLoadedTime;

$(document).ready(function() {
    /**
     * timestamp of the timer
     */
    pageLoadedTime = performance.now();

    /**
     * init the PHP script
     */
    $.ajax({
        type: "POST",
        url: "init.php",
        ...
```

Rys. 14. Fragment kodu init.js włączający licznik czasu i uruchamiający skrypt PHP.

Skrypty CursorCatcher i ClicksPost odpowiadają kolejno za zebranie i wysłanie zarejestrowanych ruchów urządzeń wskazujących do serwera.

CursorCatcher posiada obserwatory zdarzeń wychwytyjące i przekazujące do ClicksPost ruchy oraz kliknięcia urządzeń wskazujących (Rysunek 15). Uruchomienia odsyłaczy potraktowano inaczej, gdyż związane są z opuszczeniem aktualnie przeglądanej strony. Domyślnie, liczba zapytań XMLHttpRequest jaką może skierować przeglądarka klienta do serwera jest bardzo niewielka. Na przykład limit dla Opery 36.0.2130.32 wynosi 6 i nie zmienił się od 2012 roku [20]. Każde zapytanie ponad limit oczekuje w kolejce do czasu odpowiedzi z serwera na któryś z poprzednich. Jeśli użytkownik przełączy w tym czasie stronę lub zamknie przeglądarkę, informacje nie dotrą do serwera. Jednym ze sposobów obejścia tego ograniczenia było utworzenie bufora po stronie klienta, którego kod zaprezentowano na rysunku 17. Każde zapytanie AJAX poprzedza więc zgromadzenie pewnej ilości danych. Drugie rozwiązanie to wyłączenie blokady sesji (*ang. session lock*) na serwerze. Uruchomione skrypty PHP wykonywane są teraz równolegle, a nie kolejkowane.

Opisane powyżej zabezpieczenia nie gwarantują, że w przypadku nienaturalnie szybkiego przeglądania podstron nie dojdzie do pominięcia części danych. Dobrym rozwiązaniem wydaje się być np. zastosowanie akceleratora PHP na serwerze i przechowywanie części informacji w pamięci operacyjnej, zamiast natychmiastowego zapisu do bazy danych. Dla uproszczenia nie wdrożono tego mechanizmu, lecz zastosowano funkcję opóźniającą opuszczenie strony (Rysunek 16), jeżeli

zajdzie taka potrzeba. Domyślna akcja elementu odsyłacza HTML zostaje anulowana, jeżeli liczba aktywnych zapytań AJAX jest większa niż 5. W tym celu, sprawdzana jest okresowo specjalna zmienna `$.active`, której opis działania można znaleźć w kodzie źródłowym jQuery [21]. Jeżeli zmienna jest odpowiednio mała lub upłynie zbyt dużo czasu, użytkownik przenoszony jest do pożądanej strony.

Skrypty zbierają następujące dane o odwiedzającym stronę: współrzędna x kurSORA, współrzędna y kurSORA, względny czas akcji, typ akcji.

Względny czas akcji jest różnicą między aktualnym i pierwszym znacznikiem czasowym (`performance.now() - pageLoadedTime`). Akcjami są ruch urządzenia wskazującego ('Moved'), kliknięcie ('Clicked') oraz kliknięcie odsyłacza ('URLclicked'). Informacje te są automatycznie wysyłane do serwera w formacie JSON, w przypadku przepełnienia się bufora lub kliknięcia przez klienta na element z odnośnikiem do innej strony. Rysunek 17 przedstawia fragment kodu ukazującego działanie funkcji przesyłania danych.

```
$ (document).ready(function() {
    $(document).on({
        mousemove: function(event) {
            if (enableHandler) {
                clicksPost(event.pageX, event.pageY, (performance.now() - pageLoadedTime), 'Moved');
                enableHandler = false;
            }
        },
        click: function(event) {
            if($(event.target).closest('a').is('a')) {
                event.preventDefault();
                clicksPost(event.pageX, event.pageY, (performance.now() - pageLoadedTime), 'URLclicked');
                safeDelay(event);
            }
            else{
                clicksPost(event.pageX, event.pageY, (performance.now() - pageLoadedTime), 'Clicked');
            }
        }
    });
});
```

Rys. 15. Obserwatory zdarzeń po stronie klienta.

```
function safeDelay(e){
    var AJAXSnumber = getActiveAJAXs();
    var numberofChecks = 0;
    var path = $(event.target).closest('a').attr('href');

    var waitForAJAXs = function(){
        if((AJAXSnumber < 5) || (numberofChecks > 50)){
            window.location = path;
        }
        else{
            AJAXSnumber = getActiveAJAXs();
            numberofChecks++;
            setTimeout(waitForAJAXs, 200);
        }
    }
    waitForAJAXs();
}

function getActiveAJAXs(){
    return $.active;
```

Rys. 16. Funkcje `safeDelay` i zwracająca `$.active`.

```

var clicksBuffer = [];
var bufferLimit = 100;
var bufferIterator = 0;

function clicksPost(cursorX, cursorY, moveTime, actionType){
    if(clicksBuffer.length >= bufferLimit || actionType == 'URLclicked'){
        clicksBuffer[bufferIterator] = {x: cursorX, y: cursorY, time: moveTime, action: actionType};
        $.ajax({
            type: "POST",
            url: "MouseMovementToDatabase.php",
            contentType: 'application/json; charset=UTF-8', //pure JSON
            data: JSON.stringify(clicksBuffer),
            complete: function() {
                success: function(msg) {
                error: function(msg) {
                });
                clicksBuffer = [];
                bufferIterator = 0;
            }
        } else{
            clicksBuffer[bufferIterator] = {x: cursorX, y: cursorY, time: moveTime, action: actionType};
            bufferIterator++;
        }
    }
}

```

Rys. 17. Działanie bufora i przesyłanie danych z pomocą jQuery AJAX.

Init.php (Rysunek 18) uruchamia się za pośrednictwem skryptu init.js. Plik konfiguracyjny index.php zawiera zmodyfikowany mechanizm zarządzania sesją. Modyfikacja ta polega na opisany wcześniej wyłączeniu blokady sesji PHP. Wybór funkcji haszującej SHA-1 w wariancie 4 (4 bity na znak) przyznaje identyfikatory sesji o długości 40 znaków heksadecymalnych więc szansa na wygenerowanie tych samych dla różnych użytkowników jest niewielka [22].

```

<?php
include 'index.php';
include 'library.php';

if(session_id() == '' || !isset($_SESSION['currentSessionId']) || session_id() != $_SESSION['currentSessionId']) {
    $dbInit = init($serverName, $userName, $password, $dbName);
}

if(isset($_SESSION['currentSessionId']) && session_id() == $_SESSION['currentSessionId']) {
    userInfo($serverName, $userName, $password, $dbName, $session);
}

function init($serverName, $userName, $password, $dbName) {
}

function userInfo($serverName, $userName, $password, $dbName, $session) {
?>

```

Rys. 18. Inicjalizacja użytkownika na serwerze.

Jeżeli baza danych nie istnieje, automatycznie jest tworzona nowa. Plik library.php zawiera kilka skryptów pobierających dane o odwiedzającym, m.in jego adres IP, system

operacyjny, UAString (ciąg znaków informujący o używanej przeglądarce i posiadanym systemie) oraz czas odwiedzin witryny. Część z nich można wykorzystać do wykrycia próby zatajenia tożsamości przez użytkownika.

Kolejnym krokiem jest zapis zgromadzonych informacji wraz z identyfikatorem sesji do bazy danych. Rysunek 19 prezentuje odpowiedni kod.

```
$sql= "INSERT INTO General
        (sessionID, IPaddress, serverTime, userAgent, agentName, agentVersion,
        platform, agentPattern)
    SELECT * FROM
    (SELECT
        '$session', '$IPaddress', '$serverTime', '$userAgent', '$agentName', '$agentVersion', '$platform', '$agentPattern') as tmp
    WHERE NOT EXISTS (SELECT *
        FROM General
        WHERE sessionID = '$session'
        AND IPaddress = '$IPaddress'
        AND agentName = '$agentName'
        AND userAgent = '$userAgent'
        AND agentVersion = '$agentVersion'
        AND platform = '$platform'
        AND agentPattern = '$agentPattern')
    ";
if ($conn->query($sql) === TRUE) {
```

**Rys. 19.** Zapis informacji o użytkowniku do bazy danych.

Skrypt serwera MouseMovementToDatabase.php odbiera informacje o ruchach urządzeń wskazujących i zapisuje je do bazy danych. Zgodnie z wytycznymi w artykule [23], dane są pobierane z surowego strumienia, a następnie dekodowane standardową funkcją json\_decode. Fragment kodu zawiera rysunek 20.

```
if(isset($_SESSION['currentSessionId']) && session_id() == $_SESSION['currentSessionId']) {
    $conn = new mysqli($serverName, $userName, $password, $dbName);
    $session = mysqli_real_escape_string($conn, $session);
    $bData = json_decode(file_get_contents("php://input"));
    foreach($bData as $key => $value) {
        if($value) {
            $conn->query("INSERT INTO MouseTracks (sessionId, action, time, x, y)
                VALUES ('$session', '$value->action', '$value->time', '$value->x', '$value->y')");
        }
    }
}
```

**Rys. 20.** Odbiór i zapis do bazy danych ruchów urządzeń wskazujących

### 3.2 Baza danych

Baza danych działa w środowisku XAMPP [24] i posiada domyślną konfigurację. Składa się z pięciu tabel.

- General – zawiera podstawowe dane użytkownika.
- Mousetracks – mieści ruchy urządzeń wskazujących wraz z identyfikatorami sesji.

- `Mousecurves` – obejmuje podzbiór zbioru mousteracks, na który składają się ruchy typu „wskaż i kliknij” wraz z dodatkowymi identyfikatorami dla każdego z nich.
- `Curveparameters` – zawiera wyliczone metryki.
- `Blocks` – mieści dystrybuanty, zwane w tej pracy próbками, z pomocniczymi identyfikatorami.

Podczas korzystania ze strony www, tabele `general` i `mousetracks` uzupełniane są przez kod w pliku `init.php`. Aplikacja napisana w języku JAVA umożliwia zapis do nich wszystkich i łączy się z bazą danych przy pomocy metody obiektu klasy `CurveAnalyser`. Na rysunkach 21, 22, 23, 24 przedstawiono fragmenty uzupełnionych tabel `general`, `blocks`, `mousecurves` i `curveparameters`. `Mousetracks` pominięto ze względu na zbyt duże podobieństwo do `mousecurves`.

SELECT * FROM `general` ORDER BY `general`.`sessionID` ASC							
<input type="checkbox"/> Pokaż wszystko   Liczba wierszy: 25   <input type="text"/> Filtrowanie wierszy: Szukaj w tej tabeli							
<a href="#">Edit inline</a>   <a href="#">Edytuj</a>   <a href="#">Wyslij SQL</a>   <a href="#">Create PHP code</a>   <a href="#">Odśw.</a>							
sessionID	IPaddress	serverTime	userAgent	agentName	agentVersion	platform	agentPattern
9728871161433d5568db8f3ba951e3259c0f14f	127.0.0.1	08/06/2016 == 10:31:19	Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36	Google Chrome	50.0.2661.102	windows	#(?<browser>Version Chrome other)/[!]+(>version-[...]
adriafakt	NULL	NULL	NULL	NULL	NULL	NULL	NULL
arturfakt	NULL	NULL	NULL	NULL	NULL	NULL	NULL
ewafakt	NULL	NULL	NULL	NULL	NULL	NULL	NULL
magdafakt	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Rys. 21. Tabela `general`.

SELECT * FROM `blocks`						
<input type="checkbox"/> 1   >   >>            Liczba wierszy: 25   <input type="text"/> Filtrowanie wierszy: Szukaj w tej						
Sortuj wg klucza: <input type="text"/> Zadaj						
+ Opcje	→ ↵	ID	sessionID	blockID	feature	featID
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165760	arturfakt	0	0.078	0
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165761	arturfakt	0	0.082	1
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165762	arturfakt	0	0.090	2
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165763	arturfakt	0	0.096	3
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165764	arturfakt	0	0.099	4
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165765	arturfakt	0	0.101	5
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165766	arturfakt	0	0.110	6
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165767	arturfakt	0	0.111	7
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165768	arturfakt	0	0.115	8
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165769	arturfakt	0	0.172	9
<input type="checkbox"/>	<a href="#">Edytuj</a> <a href="#">Kopij</a> <a href="#">Usuń</a>	3165770	arturfakt	0	0.174	10

Rys. 22. Tabela `blocks`.

SELECT * FROM `mousecurves` ORDER BY `time` ASC							
1	<	>	>>	Liczba wierszy:	25	Filtrowanie wierszy:	Szukaj w tej tabeli
+ Opcje	sessionID	curveID	action	time	▲ 1	x	y
adrianfakt	1979a4d0-c2c1-4d8d-8e72-2505ecdd36a3	Moved	47.000	939	861		
adrianfakt	1979a4d0-c2c1-4d8d-8e72-2505ecdd36a3	Moved	48.000	923	866		
adrianfakt	1979a4d0-c2c1-4d8d-8e72-2505ecdd36a3	Moved	49.000	879	881		
adrianfakt	1979a4d0-c2c1-4d8d-8e72-2505ecdd36a3	Moved	62.000	835	894		

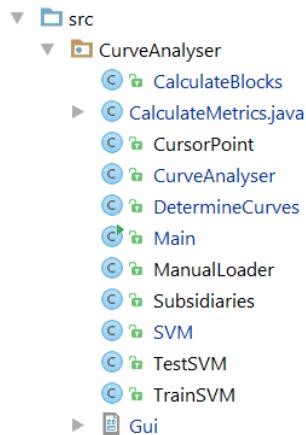
Rys. 23. Tabela mousecurves.

SELECT * FROM `curveparameters` ORDER BY `curveparameters`.`firstPointTime` ASC						
1	<	>	>>	Liczba wierszy:	25	Filtrowanie wierszy:
+ Opcje	sessionID	curveID	angleAB	angleABC	ratioBAC	firstPointTime
adrianfakt	1979a4d0-c2c1-4d8d-8e72-2505ecdd36a3	163.00	179.00	0.005	49.000	
adrianfakt	1979a4d0-c2c1-4d8d-8e72-2505ecdd36a3	161.00	178.00	0.010	62.000	
adrianfakt	1979a4d0-c2c1-4d8d-8e72-2505ecdd36a3	164.00	176.00	0.015	63.000	
adrianfakt	1979a4d0-c2c1-4d8d-8e72-2505ecdd36a3	160.00	176.00	0.019	78.000	
adrianfakt	1979a4d0-c2c1-4d8d-8e72-2505ecdd36a3	156.00	179.00	0.006	79.000	

Rys. 24. Tabela curveparameters.

### 3.3 Aplikacja klasyfikująca użytkowników

Na rysunku 25 przedstawiono strukturę klas programu. CalculateBlocks, CalculateMetrics, CursorPoint, DetermineCurves, ManualLoader odpowiadają za przygotowanie wejść, podczas gdy SVM, testSVM i trainSVM dotyczą maszyny wektorów wspierających. Subsidiaries jest to klasa pomocnicza, zawierająca struktury używane przez klasy główne. CurveAnalyser stanowi logikę aplikacji, której metody uruchamiane są z graficznego interfejsu użytkownika – klasy Gui.

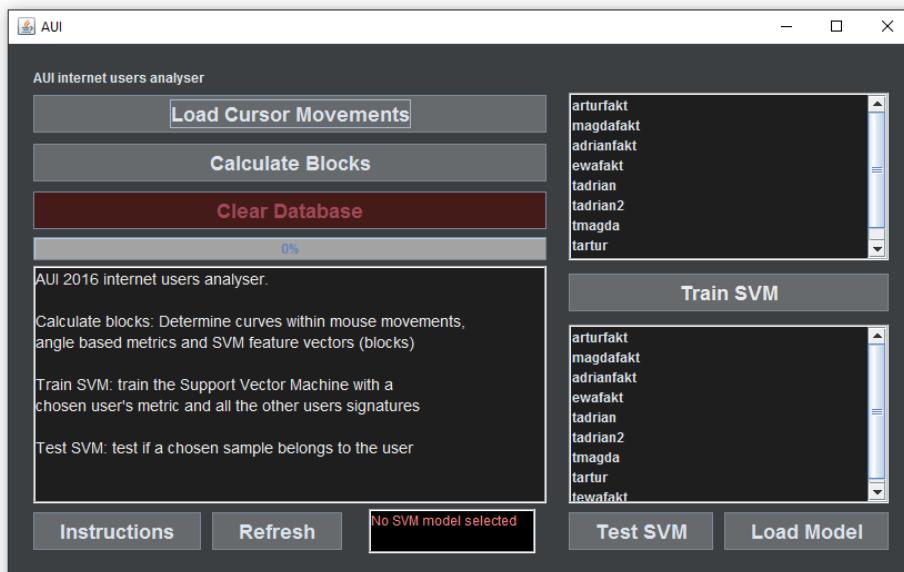


Rys. 25. Struktura klas aplikacji klasyfikującej.

Na rysunku 26 pokazano wygląd okna aplikacji. Za pośrednictwem interfejsu można uruchomić wszystkie istotne funkcje programu.

- Load Cursor Movements: pozwala na wczytanie ruchów urządzeń wskazujących zebranych zewnętrznym programem.
- Calculate Blocks: uruchamia część programu odpowiedzialną za przygotowanie wejścia.
- Clear Database: wymazuje zawartość wszystkich tabeli w bazie danych.
- Instructions: wyświetla instrukcje.
- Refresh: odświeża okna z nazwami próbek użytkowników.
- Train SVM: tworzy profil użytkownika powstający w procesie treningu maszyny wektorów wspierających. Użytkownika należy wybrać w specjalnym oknie (prawy, górny róg).
- Test SVM: Wyświetla w oknie (lewy, dolny róg) wynik porównania profilu danego użytkownika z załadowanymi próbками (okno w prawym, dolnym rogu).
- LoadModel: Ładuje profil użytkownika, którego nazwa wyświetla się w okienku w dolnej części interfejsu.

Duże okno po lewej stronie komunikuje o postępie, błędach i wynikach przeprowadzanych operacji. Wyświetla także informacje o tym, które opcje i parametry zostały wybrane. Znajdujący się ponad nim pasek stanu obrazuje nieustannie progres aktualnie wykonywanej czynności.



Rys. 26. Graficzny interfejs użytkownika.

```

public void startCurveAnalyser(Subsidiaries.RunParams runParams) {
    switch (runParams.caMode) {
        case CALCULATE :
            calculateAllFeatures(runParams.gui);
            break;
        case TRAIN :
            trainSVM(runParams.gui, runParams.svm, runParams.trainOptSel);
            break;
        case TEST :
            testSVM(runParams.gui, runParams.svm, runParams.testOptSel, runParams.modelName);
            break;
    }
}

```

Rys. 27. Fragment kodu włączającego funkcję wybraną w Gui.

Wyselekcjonowane za pośrednictwem GUI parametry przekazywane są do obiektu klasy `CurveAnalyser` w specjalnej strukturze `runParams`, która jest niezbędna do uruchomienia wybranych funkcjonalności. Rysunek 27 prezentuje fragment kodu włączającego metody różnych klas w zależności od zawartości struktury. Na przykład, aby przeprowadzić test próbek, `runParams` musi posiadać informacje o ich nazwie oraz wybranym profilu wytrenowanego użytkownika. Z uwagi na fakt, że obiekt `CurveAnalyser` jest instancją najważniejszej klasy zarządzającej pozostałymi, posiada też niezbędne parametry, aby uzyskać dostęp do bazy danych oraz funkcji połączenia z nią i odłączenia.

Typowy scenariusz zakłada uruchomienie kolejno kodów odpowiedzialnych za przygotowanie wejść (instancje klas `DetermineCurves`, `CalculateMetrics`, `CalculateBlocks`), trening (instancja `TrainSVM`) i test (instancja `TestSVM`). W przypadku tych pierwszych, aby program działał sprawnie i szybko, każde zapytanie do bazy danych wykonywane jest w tej samej transakcji, z wykorzystaniem metody `setAutoCommit(false)` (Rysunek 28). Duża liczba zarejestrowanych ruchów (2643766 w tabeli `mousetracks` w zapełnionej bazie danych) była przyczyną wolnego działania w przypadku wykonywania operacji wstawiania (*ang. insert into*) w osobnych transakcjach.

Na rysunku 28 widoczne są odwołania do obiektu GUI, gdyż interfejs jest na bieżąco aktualizowany – zmienia się m.in. pasek postępu. Ponieważ każda nowa instancja klasy `CurveAnalyser` tworzona jest w osobnym wątku, wątek biblioteki graficznej Swing o nazwie EDT (*ang. Event Dispatch Thread*) nie jest blokowany i uaktualnia wygląd aplikacji.

```

try {
    dbConnect();
    conn.setAutoCommit(false); /*< faster inserts*/
    gui.updateBar(25, Gui.S_RED);
    dc = new DetermineCurves(conn);
    dc.startDetermineCurves();
    gui.updateBar(50, Gui.S_RED);
    cm = new CalculateMetrics(conn);
    cm.startCalculateMetrics();
    gui.updateBar(75, Gui.S_RED);
    cp = new CalculateBlocks(conn);
    cp.startCalculateBlocks();
    gui.updateBar(100, Gui.S_GREEN);
    gui.updateTextArea(CALC_OK, Gui.S_GREEN, false);
    conn.commit(); /*< execute all queries */
    conn.setAutoCommit(true);
    dbDisconnect();
}
catch(SQLException e){System.out.println("Cannot achieve DB connection");}

```

Rys. 28. Kod wyłączający automatyczne zatwierdzanie zmian w bazie danych. Dostęp do Gui.

### 3.3.1 Przygotowanie wejść klasyfikatora

Proces przygotowania wejść zaczyna się w obiekcie klasy DetermineCurves, odpowiadającym za wyznaczenie krzywych. Zdefiniowane są tam następujące parametry:

```

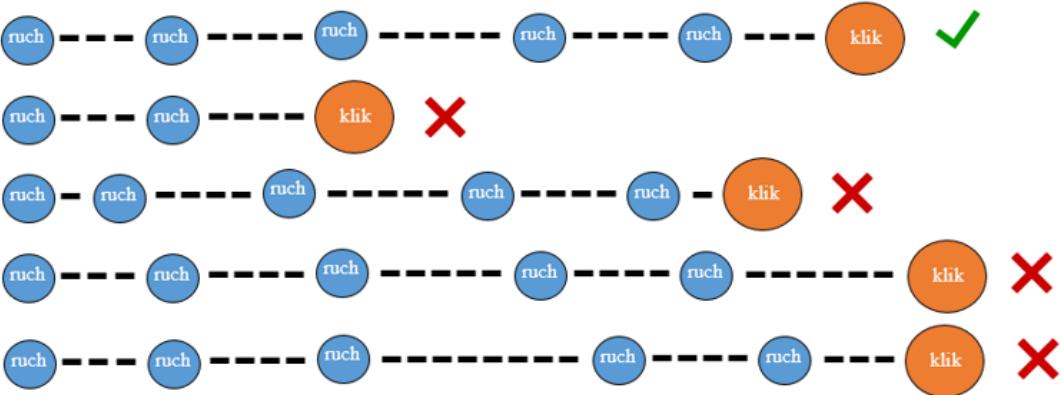
MAX_MOVE_INTERVAL = 200; /*< millisecs */
MAX_CLICK_INTERVAL = 200;
MIN_CURVE_POINTS = 50;
SAMPLING_RATE = 0;

```

Pierwszy z nich określa maksymalny dopuszczalny odstęp czasu między dwoma ruchami urządzenia wskazującego. Kolejny to limit powyżej którego czas między kliknięciem a ostatnim ruchem uznawany jest za zbyt duży. Następny definiuje najmniejszą liczbę punktów jaka musi się znaleźć w każdej krzywej, natomiast ostatni określa minimalną odległość między dwoma punktami. Na rysunku 29 określono przykładowe parametry i przedstawiono zaakceptowane oraz odrzucone krzywe. Pierwsza od góry spełnia wszystkie kryteria, druga posiada za mało punktów, trzecia ma niedomiar jednostek czasu między punktami ruch – ruch, czwarta i piąta mają zbyt dużo jednostek czasu między odpowiednio klik – ruch i ruch – ruch.

Najpierw sprawdza się czy operacja wyznaczenia krzywych nie została już wcześniej wykonana dla danego identyfikatora sesji. Jeżeli nie, wszystkie ruchy urządzeń wskazujących pobierane są z bazy danych. Analogiczny proces weryfikacji wykonuje się dla obiektów klas CalculateMetrics i CalculateBlocks. Na rysunku 30 przedstawiono kod głównej funkcji CheckCursorMovements decydującej o tym, które krzywe spełniają wszystkie warunki i są akceptowane.

— jednostka czasu  
 MAX\_MOVE\_INTERVAL = 7 jednostek czasu  
 MINIMUM\_CURVE\_POINTS = 4 punkty (ruch lub klik)  
 SAMPLING\_RATE = 2 jednostki czasu  
 MAX\_CLICK\_INTERVAL = 5 jednostek czasu



Rys. 29. Akceptowane i odrzucane krzywe.

We fragmencie kodu funkcji `checkCursorMovements`, `RsCcVals` jest obiektem typu `ResultSet`, a `curveList` to lista do której trafiają punkty. W pierwszej kolejności sprawdzana jest akcja. Jeśli ruch zostanie wykryty, weryfikuje się czy różnica w czasie między aktualnym i poprzednim punktem jest mniejsza niż `MAX_MOVE_INTERVAL`. W przypadku gdy punkty te mają identyczne współrzędne, aktualny zajmuje miejsce poprzedniego. Jeśli nie, ustala się, czy odległość między nimi jest większa niż `SAMPLING_RATE`. Spełnienie tego warunku przyłącza punkt do listy.

```

if(rsCcVals.getString("action").equals("Moved")){
  if((rsCcVals.getFloat("time") - curveList.getLast().getTime()) < MAX_MOVE_INTERVAL){
    if((rsCcVals.getInt("x") == curveList.getLast().getX()) && (rsCcVals.getInt("y") == curveList.getLast().getY())){
      curveList.removeLast();
      curveList.addLast(new CursorPoint(rsCcVals.getString("sessionID"), "none", rsCcVals.getString("action"),
        rsCcVals.getFloat("time"), rsCcVals.getInt("x"), rsCcVals.getInt("y")));
    }
  } else if (sampling(rsCcVals.getInt("x") - curveList.getLast().getX(), rsCcVals.getInt("y") - curveList.getLast().getY())){
    curveList.addLast(new CursorPoint(rsCcVals.getString("sessionID"), "none", rsCcVals.getString("action"),
      rsCcVals.getFloat("time"), rsCcVals.getInt("x"), rsCcVals.getInt("y")));
  }
}
  
```

Rys. 30. Fragment kodu wyznaczającego krzywe.

Każda zaakceptowana krzywa otrzymuje własny identyfikator i umieszczana jest w bazie danych w tabeli `mousecurves`. Zachowuje także swój identyfikator sesji. Następnym krokiem jest obliczenie metryk w obiekcie klasy `CalculateMetrics`, gdzie istotnymi funkcjami są `abAngle`,

*abcAngle*, *bACratio*. Aby zapewnić odpowiednią szybkość, zastosowano działania na wektorach i ograniczono użycie czasochłonnych funkcji typu *atan2*.

Na rysunku 31 przedstawiono kod obliczający kąt ABC między dwoma wektorami. Zamiast różnicy dwóch funkcji *arctan* zastosowano jeden *atan2* działający na stosunku iloczynu wektorowego do iloczynu skalarnego czyli sinusa do cosinusa. Potraktowanie wyniku iloczynu wektorowego jako wartości skalarnej jest uproszczeniem, gdyż formalnie jest on zdefiniowany w przestrzeni trójwymiarowej.

```

public double abcAngle(Point pointA, Point pointB, Point pointC) {
    /**
     * Set down AB and CB vectors
     */
    Point vectorAB = new Point(pointB.getX() - pointA.getX(), pointB.getY() - pointA.getY());
    Point vectorCB = new Point(pointB.getX() - pointC.getX(), pointB.getY() - pointC.getY());

    /**
     * Count dot product and pseudo cross product
     */
    double dot = (vectorAB.getX() * vectorCB.getX() + vectorAB.getY() * vectorCB.getY());
    double cross = (vectorAB.getX() * vectorCB.getY() - vectorAB.getY() * vectorCB.getX());

    /**
     * count angle
     */
    double angle = Math.atan2(cross, dot);
    angle = angle >= 0 ? angle : angle * (-1);

    return Math.floor(angle * 180 / Math.PI + 0.5);
}

```

Rys. 31. Obliczanie kąta ABC.

*BACratio* oblicza stosunek odległości punktu B od prostej AC do długości odcinka AC. Ponieważ łatwo to zrobić dzieląc pole równoległoboku *ABCC'* przez długość boku AC, liczony jest iloczyn wektorowy między wektorami AB i AC i dzielony przez długość wektora AB. Jeśli punkty A i C nakładają się, uzyskiwana jest odległość punktu B od A/C. Na rysunku 32 zaprezentowano odpowiedni kod. Metryki zapisywane są do tabeli *curveparameters* i rozróżniane po identyfikatorach sesji i krzywej.

Ostatni etap to wyliczenie dystrybuant przez obiekt klasy *CalculateBlocks*. Najważniejsze parametry zostały przedstawione poniżej:

```

BLOCK_SIZE = 25;
AB_MIN = 0;
AB_BIN_NUM = 36;
AB_BIN_SIZE = 10;
PDF_PRECISION = 1000.0;

```

```

public double bACratio(Point pointA, Point pointB, Point pointC){
    /**
     *Set down AB vector
     */
    Point vectorAB = new Point(pointB.getX() - pointA.getX(), pointB.getY() - pointA.getY());
    Point vectorAC = new Point(pointC.getX() - pointA.getX(), pointC.getY() - pointA.getY());

    /**
     * Count dot product and pseudo cross product
     */
    double cross = (vectorAB.getX() * vectorAC.getY() - vectorAB.getY() * vectorAC.getX());

    /**
     * count vAC module
     */
    double vectorACmod = vectorAC.getX() * vectorAC.getX() + vectorAC.getY() * vectorAC.getY();
    if(vectorACmod == 0){
        return (Math.sqrt(Math.pow(vectorAB.getX(),2) + Math.pow(vectorAB.getY(),2)));
    }
    else{
        return Math.abs(cross)/vectorACmod;
    }
}

```

Rys. 32 Obliczanie stosunku odległości punktu B od prostej AC do długości odcinka AC.

*BLOCK\_SIZE* określa z ilu krzywych należy zebrać metryki, aby wyznaczyć jedną dystrybuantę czyli próbkę. *AB\_MIN* ustala minimalną wartość kąta lub stosunku odległości jaka jest brana pod uwagę, natomiast *AB\_BIN\_NUM* to liczba prążków (*ang. bins*) czyli przedziałów metryk. *AB\_BIN\_SIZE* określa długość prążka czyli szerokość przedziału kątów lub stosunków odległości. W przypadku tak zdefiniowanych parametrów jak powyżej, brany pod uwagę będzie przedział od 0 – 360 stopni, liczony co 10 stopni. Dla dwóch pozostałych metryk postępowano analogicznie. Przedziały użyte w pracy dla każdej z metryk pokazane są na rysunkach 33, 34, 35. *PDF\_PRECISION* określa precyzję liczb będących zbiorem wartości dystrybuant.



0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320 330 340 350 360

Rys. 33. Prążki kąta AB.



0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180

Rys. 34. Prążki kąta ABC.



0 0.18

Rys. 35. Prążki  $|B|AC|/|AC|$ .

Dla danego użytkownika, kolejno pobierane są z bazy danych rzędy z metrykami (ich zbiór zapisany jest w zmiennej `rsCpcall`). W pętli widocznej na rysunku 36, sprawdza się czy w jej danym obiegu nowy rząd ma inny identyfikator krzywej niż poprzedni. Jeśli tak, uruchamiana jest funkcja `histograms` lecz pod warunkiem, że dane z 25 krzywych (wartość `BLOCK_SIZE` w niniejszej pracy) nie zostały jeszcze zebrane (Rysunek 37). Jej działanie jest następujące: utworzone wcześniej tablice (`abInput`, `abcInput`, `bacInput`) dla wszystkich metryk, o rozmiarach zdolnych pomieścić komplet prążków są uzupełniane. Każda z trzech metryk może pasować do jednego z 36 prążków swojego rodzaju lub zostać odrzucona, jeśli wypada poza zakres. W przypadku pozytywnym, wartość pod jednym z 36 indeksów jest podnoszona o 1. Na przykład, kąt AB równy 34 pasuje do czwartego prążka swojego rodzaju bo podłoga (*ang. floor*) z ilorazu  $\frac{34-0}{10}$  wynosi 3. Oznacza to, że wartość tablicy `abInput[3]` wzrośnie o 1.

```

while (rsCpcall.next()) {
    /**
     *
     */
    currCurveID = rsCpcall.getString("curveID");
    if (!currCurveID.equals(prevCurveID)) {
        if (blockIterator == BLOCK_SIZE + 1) {

            /**
             * Create CDFs from histograms
             */
            hist2cdf();
            /**
             * save a block in database
             */
            sendBlock(currSessionID, currBlockID);
            /**
             * clear all blocks' data
             */
            resetBlockData();

            /**
             * iterate the current block's ID
             */
            currBlockID++;
        }
        blockIterator++;
    }
    histograms(rsCpcall);
    prevCurveID = currCurveID;
}

```

Rys. 36. Główna pętla klasy CalculateBlocks.

```

public void histograms(ResultSet rsCpcall) {

    try{
        int ABbin = (int) ((rsCpcall.getInt("angleAB") - AB_MIN) / AB_BIN_SIZE);
        int ABCbin = (int) ((rsCpcall.getInt("angleABC") - ABC_MIN) / ABC_BIN_SIZE);
        int BACbin = (int) ((rsCpcall.getDouble("ratioBAC") - BAC_MIN) / BAC_BIN_SIZE);

        if(match2anyBin(ABbin, AB_BIN_NUM)
            && match2anyBin(ABCbin, ABC_BIN_NUM)
            && match2anyBin(BACbin, BAC_BIN_NUM) ) {
            abInput[ABbin]++;
            abcInput[ABCbin]++;
            bacInput[BACbin]++;
        }
    }
    catch(Exception e){System.out.println("Error");}
}

```

Rys. 37. Funkcja obliczająca częstotliwość występowania metryk.

Jeżeli natomiast zebrane zostaną dane ze zbioru metryk z 25 krzywych, z wartości w tablicach *abInput*, *abcInput* i *bacInput* konstruowana jest dystrybuanta. Rysunek 38 przedstawia proces jej tworzenia dla kąta AB. *AbSum* to suma liczebności wszystkich metryk w tablicy *abInput*.

```

for(int i = 0; i<= AB_BIN_NUM; i++) {
    tempSum = tempSum + abInput[i];
    abInput[i] = (Math.round((tempSum / abSum) * PDF_PRECISION)) / PDF_PRECISION;

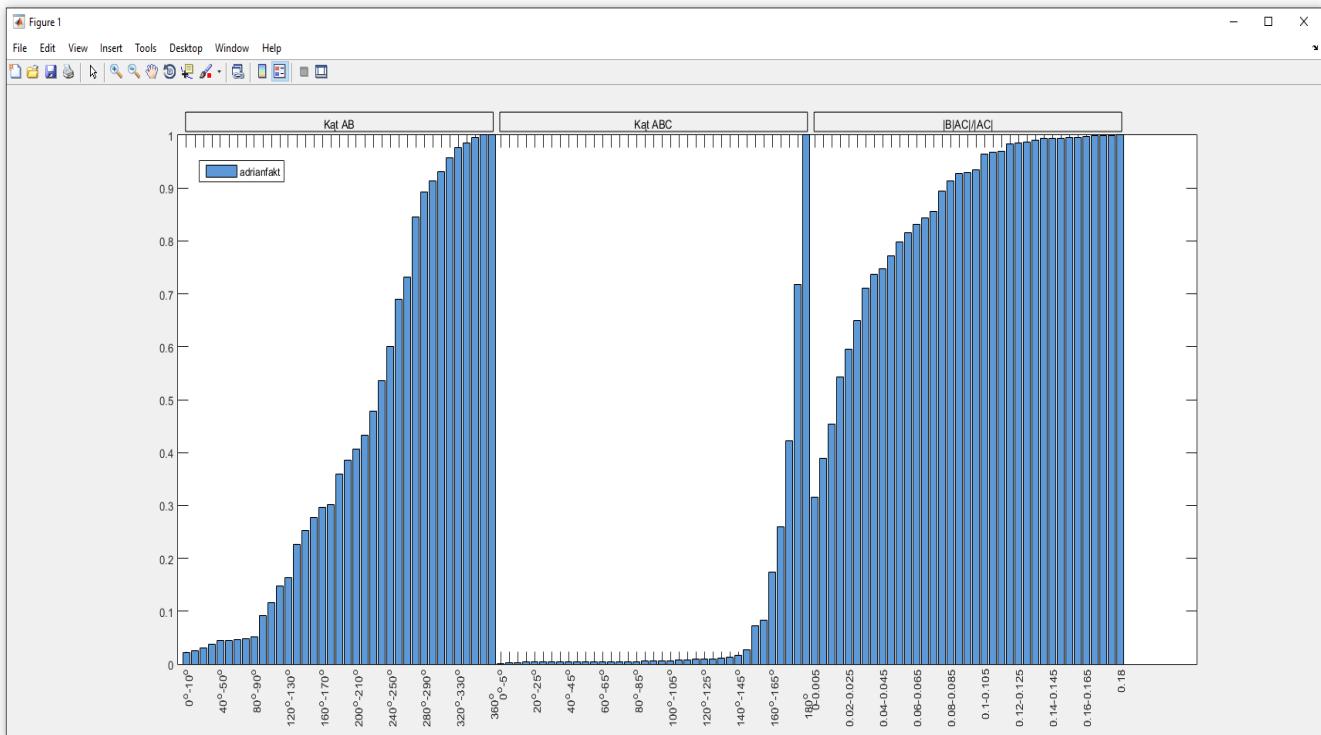
}
tempSum = 0;

```

Rys. 38. Konstruowanie dystrybuanty dla kąta AB.

Wszystkie dane z trzech tablic otrzymują ten sam identyfikator i zostają wysłane do bazy danych jako jedna próbka. W pętli z rysunku 36 odpowiednie zmienne i struktury, m.in. tablice *abInput*, *abcInput* i *bacInput*, są resetowane. Rysunek 39 przedstawia histogram pełnej dystrybuanty, złożonej z wartości z 3 tablic. Na osi poziomej zaznaczono częstotliwości przedziałów metryk. Oś pionowa zawiera odpowiadające im wartości w przedziale [0,1].

Każda próbka posiada identyfikatory sesji, krzywej oraz próbki. Te trzy atrybuty pozwalają jednoznacznie je rozróżnić w zbiorze w bazie danych. Zawierająca je tabela *blocks* jest źródłem z którego są pobierane i formatowe wektory wejściowe dla maszyny wektorów wspierających.



Rys. 39. Przykład próbki przedstawionej w formie histogramu.

### 3.3.2 Trening klasyfikatora

Za trening odpowiadają obiekty klas TrainSVM i SVM. Celem ich działania jest wygenerowanie pliku zawierającego profil wytrenowanego użytkownika. TrainSVM przygotowuje wcześniej dwa pliki tekstowe, które zawierają próbki użytkownika, dla którego utworzony ma być profil i dystrybuanty pozostałych osób. Jeden z nich przeznaczony jest dla sprawdzianu krzyżowego, który wyszukuje odpowiednie parametry funkcji jądra, natomiast drugi służy do treningu z wyznaczonymi dla niej wartościami. Sprawdzian krzyżowy i trening przeprowadza obiekt klasy SVM.

Zanim utworzone zostają pliki tekstowe z próbками osób, dystrybuanty są losowo wybierane z bazy danych. Na rysunku 40 przedstawiono metody, które odpowiadają za ich wybór.

```

int[] negRandomVec = createRandomizationVector(HALF_CV_DATA_SIZE + HALF_TRAINING_DATA_SIZE, negativeBlocks);
makeBlockList(rsBsidN, "-1 ", cvNegativeBlockList, trainNegativeBlockList, negRandomVec, HALF_CV_DATA_SIZE);

int[] posRandomVec = createRandomizationVector((HALF_CV_DATA_SIZE + HALF_TRAINING_DATA_SIZE) * posUserNum , positiveBlocks);
makeBlockList(rsBsidP, "+1 ", cvPositiveBlockList, trainPositiveBlockList, posRandomVec, HALF_CV_DATA_SIZE * posUserNum);

```

Rys. 40. Część kodu z funkcją generującą losowe wektory i pobierającą próbki z bazy danych.

Następnie metoda `createRandomizationVector` generuje tablice z ciągiem niepowtarzających się, losowo ułożonych liczb. Sekwencja ta dopasowana jest do liczności próbek. Funkcja `makeBlockList` posiada pętlę, która iteruje przez wszystkie rzędy w obiekcie klasy `ResultSet`, zawierającym pobrane z bazy danych próbki. Dla tych iteracji, których liczby znajdują się w tablicy, pobierane są dystrybuanty. Widać to w kodzie na rysunku 41.

```

if (Arrays.binarySearch(randomVec, blockIt) >= 0) {
    String currSessionID = rsBsid.getString("sessionID");
    String currBlockID = rsBsid.getString("BlockID");

    String SQL_B_ALL = "SELECT feature FROM `blocks` WHERE SessionID = '" + currSessionID + "' AND BlockID = '" + currBlockID +
    Statement stmtBall = conn.createStatement();
    ResultSet rsBall = stmtBall.executeQuery(SQL_B_ALL);
}

```

**Rys. 41.** Funkcja selekcjonująca próbki.

Metoda `makeBlockList` wywoływana jest dwukrotnie: w pierwszej kolejności w celu pobrania próbek użytkownika, a następnie pozostałych osób. Dystrybuanta zapisywana jest w obiekcie `String`, w formacie wymaganym przez LIBSVM, co pokazuje kod na rysunku 42.

```

while (rsBall.next()) {
    if (rsBall.getDouble("feature") > 0) {
        blockRow = blockRow + rowIt + ":" + rsBall.getDouble("feature") + " ";
    }
    else{
        blockRow = blockRow + rowIt + ":" + rsBall.getDouble("feature") + " ";
    }
    rowIt++;
}

```

**Rys. 42.** Funkcja formatująca próbki dla LIBSVM.

Najpierw zbierane są próbki dla sprawdzianu krzyżowego, a następnie do treningu. Zgromadzone dystrybuanty przechowywane są w odpowiednich listach z dowiązaniami (dla trenowanego użytkownika oraz dla pozostałych). Rysunek 43 zawiera kod realizujący tę funkcję.

```

if(halfCvBlockList.size() < halfDataSize) {
    halfCvBlockList.add(blockRow);
}
else{
    halfTrainBlockList.add(blockRow);
}

```

**Rys. 43.** Kod zapełniający próbami odpowiednie listy z dowiązaniami.

Następnie tworzone są nowe listy, zawierające połączone próbki treningowe i dla sprawdzianu krzyżowego. Na rysunku 44 pokazano kod konwertujący je do plików tekstowych. Nadawana jest im odpowiednia nazwa – przedrostek `4train` lub `4cv` oraz identyfikator sesji trenowanego użytkownika. Jeżeli dana nazwa już istnieje, dodawany jest przyrostek z liczbą w nawiasie.

```
trainInputName = writeToFile("4train_" + selectedNegative, trainBlockList);
```

Rys. 44. Konwersja list z próbками do plików tekstowych.

Sprawdzian krzyżowy i trening na osobnych zbiorach próbek powinien teoretycznie zapewnić większą skuteczność maszyny wektorów wspierających. Jednakże w pracy, z powodu zbyt małej liczby zebranych próbek, wykonywane są na tym samym zbiorze. Testy potwierdziły wystarczająco dobrą skuteczność przy tym rozwiązaniu.

Rysunek 45 przedstawia fragment kodu wywołującego główną metodę `run` obiektu klasy SVM. Funkcja ta jest najpierw uruchamiana z argumentami niezbędnymi do przeprowadzenia sprawdzianu krzyżowego (`-v` – sprawdzian krzyżowy, `10` – jego krotność, nazwa pliku z próbками). Później przesyłana jest sama nazwa pliku z próbками. W obu przypadkach przekazuje się też strukturę `hyperParams` w której zapisane zostaną wyznaczone parametry.

```
/**  
 * Grid search cross validation  
 */  
String[] argv = {"-v", "10", cvInputName};  
  
try {  
    svm.run(argv, hyperParams);  
    gui.updateBar(75, Gui.S_RED);  
    gui.updateTextArea("Grid search cross validation completed. \nSelected parameters: \nC: " + hyperParams.getC());  
}  
catch (IOException e){  
    gui.updateTextArea("Grid search cross validation error", Gui.S_RED, true);  
}  
/**  
 * Train  
 */  
String[] argv2 = {trainInputName};  
  
try {  
    gui.updateBar(99, Gui.S_RED);  
    svm.run(argv2, hyperParams);  
    gui.updateTextArea("SVM model trained and saved to a file successfully", Gui.S_WHITE, true);  
}  
catch (IOException e){gui.updateTextArea("Error training SVM model", Gui.S_RED, true);}
```

Rys. 45. Fragment kodu uruchamiającego metodę `run` obiektu klasy SVM.

Klasa SVM zawiera zmodyfikowane kody dołączone do pakietu LIBSVM – `svm_train.java` i `svm_predict.java`. Korzystaję one z biblioteki dodanej w formie archiwum z rozszerzeniem `libsvm.jar`. Obiekt klasy SVM wczytuje najpierw plik tekstowy z próbami do sprawdzianu krzyżowego. Następnym ważnym krokiem jest wyszukanie parametrów C i gamma. Zastosowano w tym celu metodę `gridSearch`, zalecaną przez autorów dokumentu [8]. Różne pary C i gamma takie, że  $C = 2^{-5}, 2^{-3}, \dots, 2^{-15}$ ,  $\gamma = 2^{-15}, 2^{-13}, \dots, 2^3$  są testowane, a wybierana jest ta, dla której sprawdzań krzyżowy ma największą skuteczność w rozpoznawaniu próbek (Rysunek 46). W przypadku, gdy wiele par ma taką samą, preferowane są te z najniższym C by otrzymać maszynę o miękkim marginesie. Uzupełniona struktura `hyperParams` pozwala na wytrenowanie maszyny z właściwymi parametrami i wygenerowanie końcowego pliku dla trenowanego użytkownika. Nadawana jest mu odpowiednia nazwa – przedrostek `4train` + identyfikator sesji użytkownika + przyrostek `model`.

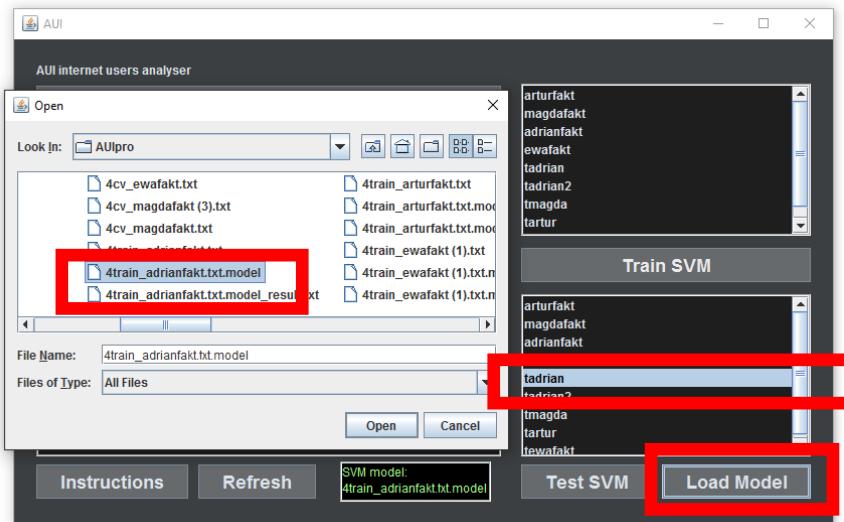
```

for (int i = 0; i < arrayC.length; i++) {
    for (int j = 0; j < arrayGamma.length; j++) {
        param.C = Math.pow(2, arrayC[i]);
        param.gamma = Math.pow(2, arrayGamma[j]);
        arrayAcc[i][j] = do_cross_validation();
    } // gamma
} // C
    
```

Rys. 46. Fragment kodu przedstawiający pętlę metody `gridSearch`.

### 3.3.3 Test klasyfikatora

Test SVM i SVM to klasy, których obiekty odpowiadają za przeprowadzenie testu klasyfikacji próbek. Rysunek 47 pokazuje, które elementy mają zostać wczytane i zaznaczone.



Rys. 47. Wczytanie pliku z danymi o wytrenowanym użytkowniku i wybranie próbek testowych.

Próbki testowe oznaczono w pracy przedrostkiem `t`. Zostały one wyznaczone z ruchów urządzeń wskazujących po skompletowaniu danych użytych do treningu, dzięki czemu są od nich zupełnie niezależne. Nie ingerowano też w ich spójność, tzn. nie były dzielone i mieszane.

Pobrane z bazy danych próbki użytkownika, który ma być poddany porównaniu z wybranym profilem, zostają zapisane do pliku z przedrostkiem `test`, po uprzedniej konwersji do formatu LIBSVM, przeprowadzanej analogicznie jak w części treningowej. Następnie nazwa profilu użytkownika testowanego przekazywana jest jako argument do funkcji `runPredict` w obiekcie klasy SVM. W wyniku jej działania tworzony jest plik o nazwie tego profilu z przyrostkiem `result.txt`. Wynik dopasowania przedstawiany jest także w oknie GUI. Na rysunku 48 zaprezentowano kluczowy fragment kodu odpowiedzialny za wymienione wyżej działania.

```
try {
    double[] prob_estimates = new double[2];
    String output = modelName + "_result.txt";
    String[] argv = {inputName, modelName, output};
    svm.runPredict(argv, svmResult);
    gui.updateBar(99, Gui.S_RED);
    if(svmResult.getAcc() > 90) {
        gui.updateTextArea("Test completed. Result:\n" + "Accuracy: " + svmResult.getAcc() +
                           "\nCorrect:" + svmResult.getVerified() + "\nTotal: " + svmResult.getTotal(), Gui.S_GREEN, true);
    }
    else{
        gui.updateTextArea("Test completed. Result:\n" + "Accuracy: " + svmResult.getAcc() +
                           "\nCorrect:" + svmResult.getVerified() + "\nTotal: " + svmResult.getTotal(), Gui.S_RED, true);
    }
}
catch (IOException e){System.out.println("SVM START ERROR");}
```

Rys. 48. Fragment kodu uruchamiający główną metodę obiektu klasy SVM dla testu.

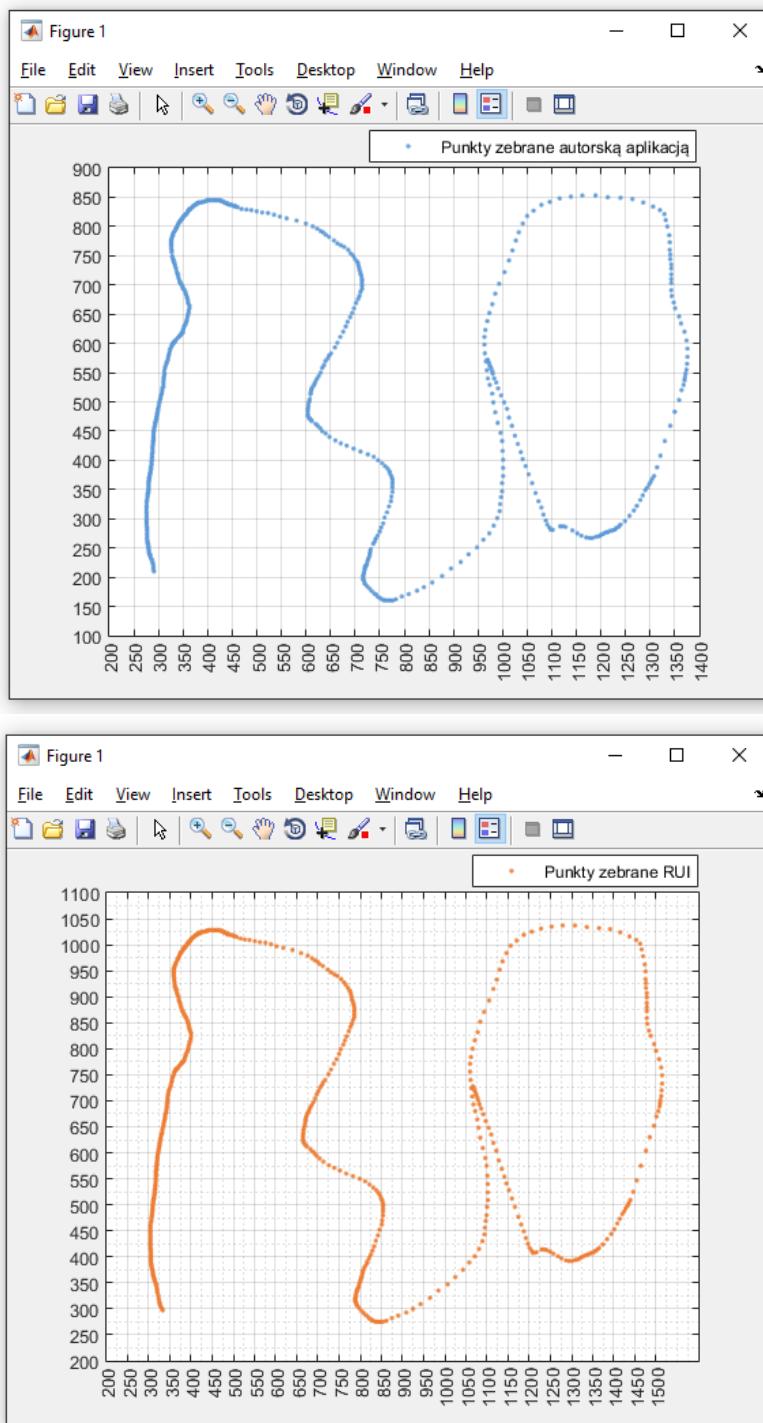
## 4. Weryfikacja aplikacji

Rozdział ten zawiera testy wydajności i rejestratorów ruchu oraz wyjaśnia dlaczego zastosowane metryki i wejścia pozwalają na skuteczną separację przez klasyfikator.

### 4.1 Porównanie programów rejestrujących ruchy urządzeń wskazujących

Z uwagi na konieczność zebrania stosunkowo dużej liczby próbek od użytkowników, zdecydowano się na użycie programu RUI, napisanego w c#. Jego dokładny opis umieszczony jest w dokumencie [25]. Jest to program uruchamiany bezpośrednio w systemie Windows i podobnie jak aplikacja autorska, nagrywa ruchy urządzeń wskazujących. Podczas rejestracji użytkownicy przeglądali portal fakt.pl. Korzystanie z rzeczywistego serwisu internetowego pozwalało zgromadzić naturalne ruchy użytkowników.

Oba programy posiadają bardzo podobną częstotliwość odświeżania (*ang. polling rate*). Rysunek 49 prezentuje wykresy punktów zebranych aplikacjami w tym samym czasie. Wykonano 3 ruchy myszką od lewej strony ekranu, kolejno: wolny, średni i szybki. Niebieski kolor wykresu dotyczy autorskiego programu, pomarańczowy aplikacji RUI.



Rys. 49. Porównanie wykresów punktów zebranych autorską aplikacją i RUI.

## 4.2 Analiza możliwości odseparowania próbek

Do treningu wykorzystano  $4 \cdot 49$  próbek bez przedrostka  $\tau$ . Szczegółowa liczебность z przyporządkowaniem do użytkowników widoczna jest na rysunku 50.

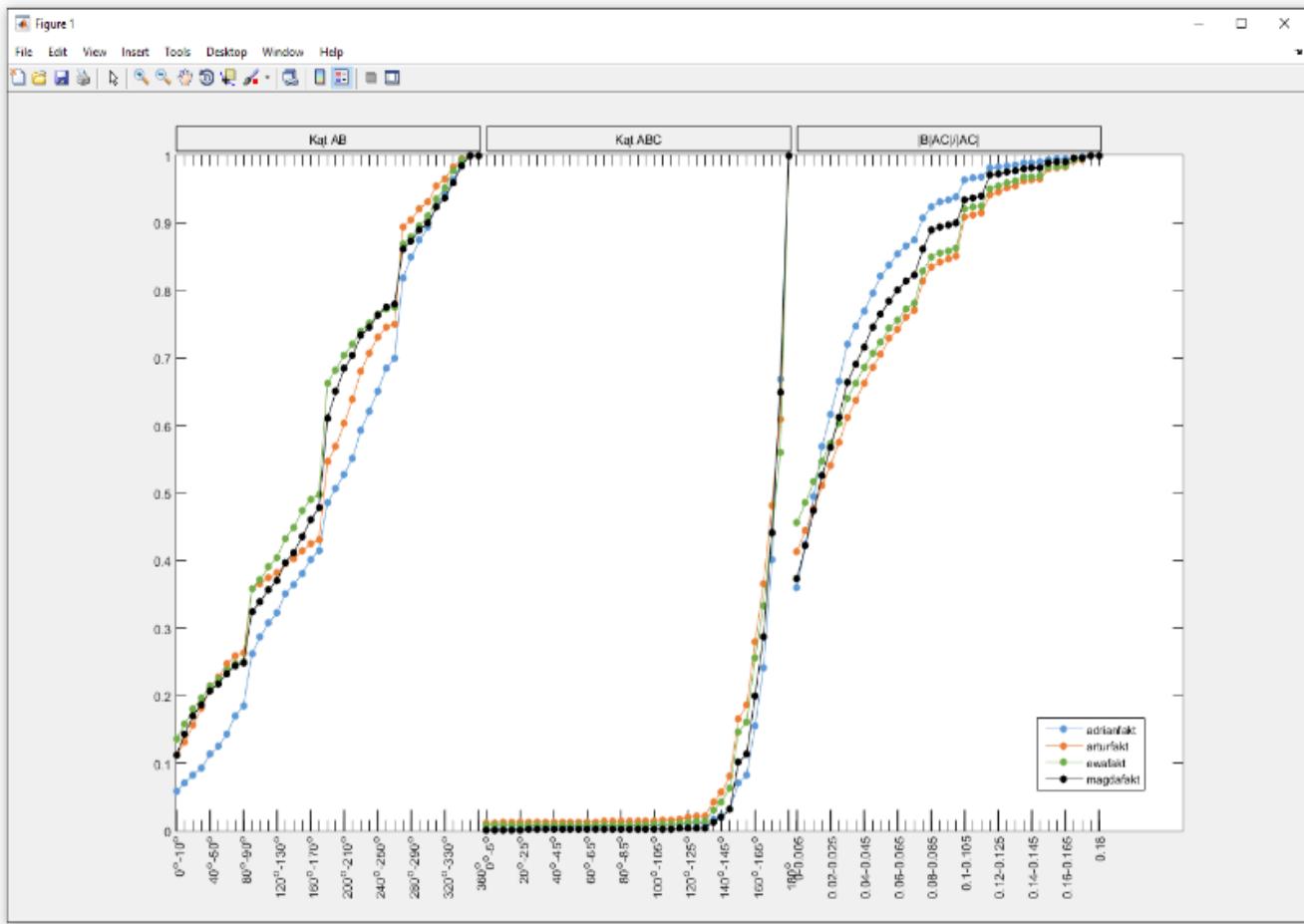
Identyfikator_sesji	Liczba_róbek
adrianfakt	49
arturfakt	52
ewafakt	52
magdafakt	53
tadrian	5
tadrian2	39
tartur	20
tewafakt	60
tmagda	40

Rys. 50. Liczebność próbek w bazie danych.

Rysunek 51 przedstawia wykresy krzywych, należących do czterech użytkowników. Zbiorem wartości każdego z nich są uśrednione wartości 49 dystrybuant. Na osi OX zaznaczono dyskretne zmienne losowe czyli częstości występowania przedziałów kątów i odległości (inaczej prążki metryk). Oś OY zawiera przyjmowane przez nich prawdopodobieństwa wystąpień.

Kąt AB dotyczy kierunku w którym użytkownik porusza kursor. Kształt krzywej wykresu jego dystrybuant jest więc zależny od interfejsu strony internetowej. Różne miejsca i sposoby nawigacji mogą być uznane za bardziej interesujące. Jak widać na rysunku, duże skoki w wartościach występują na przedziałach  $0^\circ - 10^\circ$ ,  $90^\circ - 100^\circ$ ,  $180^\circ - 190^\circ$ ,  $270^\circ - 280^\circ$ . Związane jest to między innymi z rejestracją kątów jedynie o miarach  $0^\circ$ ,  $90^\circ$ ,  $180^\circ$ ,  $270^\circ$  przy wolnych ruchach. Częstotliwość odświeżania pozwala wtedy zebrać wszystkie możliwe punkty. Wykonywanie ruchów poziomych i pionowych wydaje się być też częstsze z powodu naturalnego dążenia do nich podczas nawigacji, czego przykładem może być korzystanie z suwaków do przewijania strony.

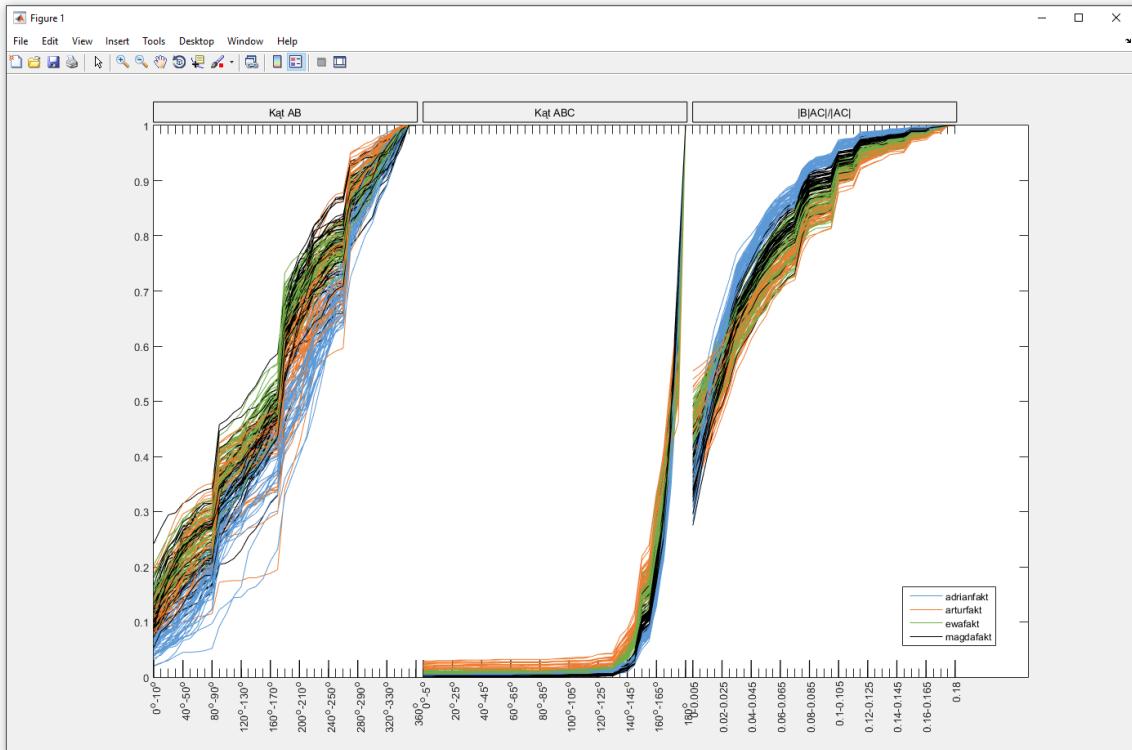
W przypadku kąta ABC oraz ilorazu odległości  $B|AC$  i długości odcinka  $|AC|$ , istotnych jest kilka przedziałów w których występują duże przyrosty wartości. Dla kąta ABC największe wzrosty zaczynają się od przedziału  $135^\circ - 140^\circ$ . Rozwarte kąty krzywizn charakteryzują krzywe znaczenie częściej niż inne. Ostre występują zazwyczaj wtedy, gdy następują gwałtowne zwroty w prowadzeniu kurSORA.



Rys. 51. Wykresy uśrednionych dystrybuant.

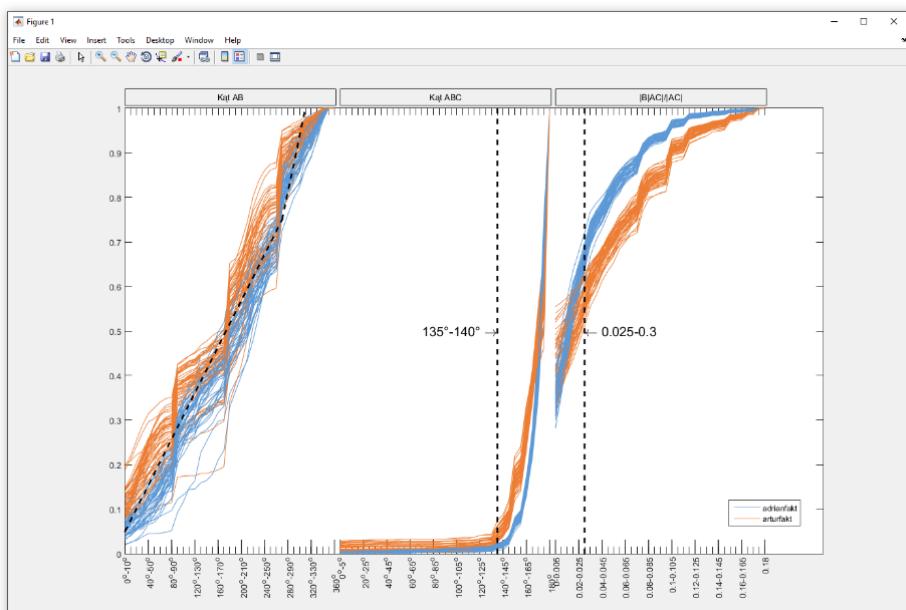
Jak już napisano, każda z dystrybuant przyporządkowywana jest podczas treningu do jednej z dwóch grup czyli klasy pozytywnej lub negatywnej. Skuteczność identyfikacji użytkowników zależy od tego ile próbek znalazło się po właściwych stronach hiperpłaszczyzny separującej oraz jak bardzo reprezentują one wszystkie możliwe zachowania osób podczas ruchów urządzeniami wskazującymi. Im lepiej odizolowane są wykresy dystrybuant, tym większą skuteczność ma klasyfikator.

Rysunek 52 przedstawia wykresy  $4 \times 49$  dystrybuant użytych do treningu. Próbki użytkowników utworzyły dostrzegalnie odseparowane zbiory.



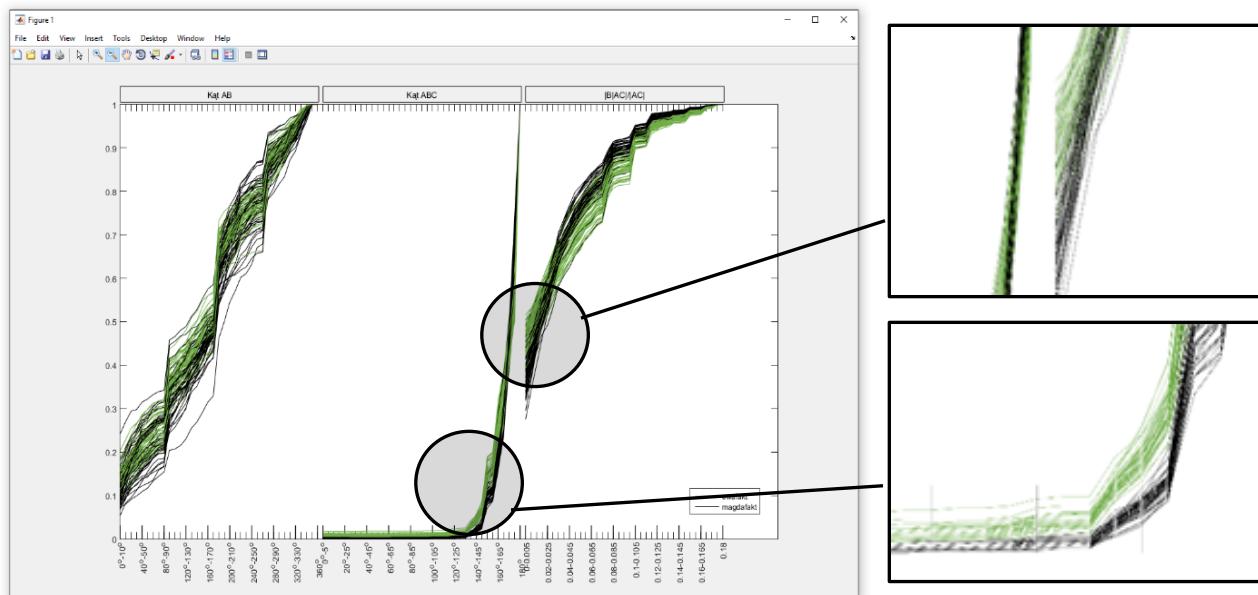
Rys. 52. Wykresy dystrybuant uzytych w treningu klasyfikatora.

W celu oszacowania jakości separacji użytkowników dokonano analizy wykresów odpowiednich zbiorów dystrybuant. Zestawiono ze sobą wszystkie możliwe pary zbiorów treningowych czterech osób. Na rysunku 53 widoczne są dobrze odseparowane zbiory próbek dwóch użytkowników. Największy odstęp dostrzegalny jest od przedziału  $135^\circ - 140^\circ$  dla kąta ABC i  $0,025 - 0,3$  dla stosunku odległości. W izolowaniu grup punktów nie przeszkadza nakładanie się wykresów dystrybuant w przedziale  $0 - 0,025$ .

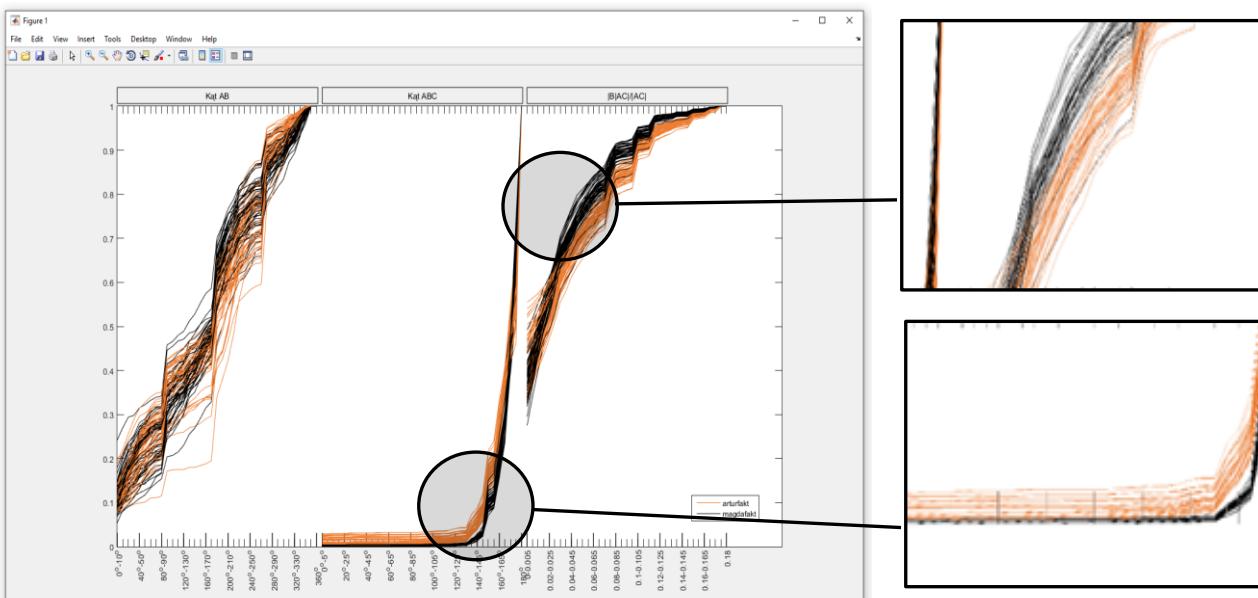


Rys. 53. Wyraźnie odseparowane zbiory dystrybuant uzytkownika niebieskiego i pomarańczowego.

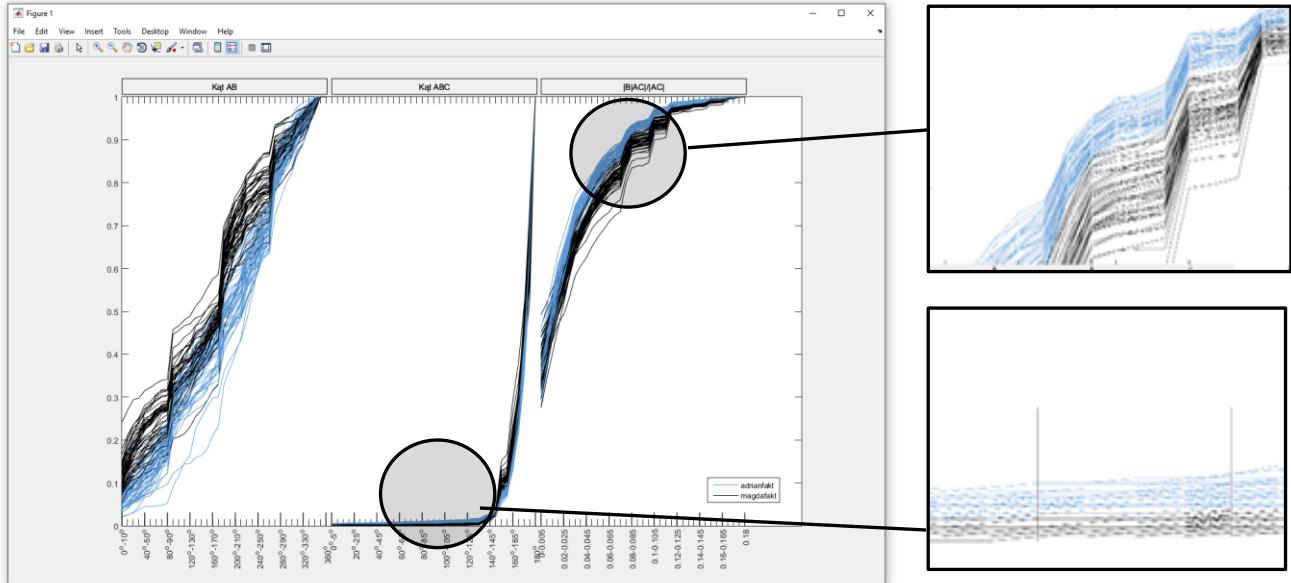
Rysunki 54, 55, 56, 57 prezentują pozostałe pary wykresów dystrybuant. Wyszczególnione zostały fragmenty na których dokładnie widać odseparowanie większości próbek.



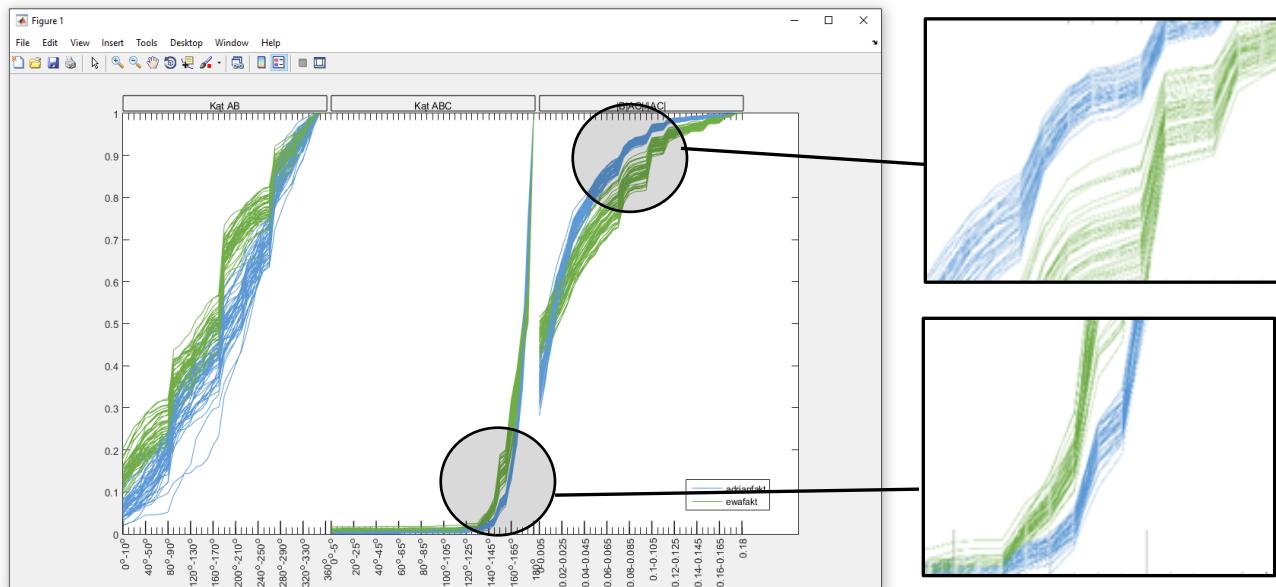
Rys. 54. Wykresy próbek treningowych użytkowników czarnego i zielonego.



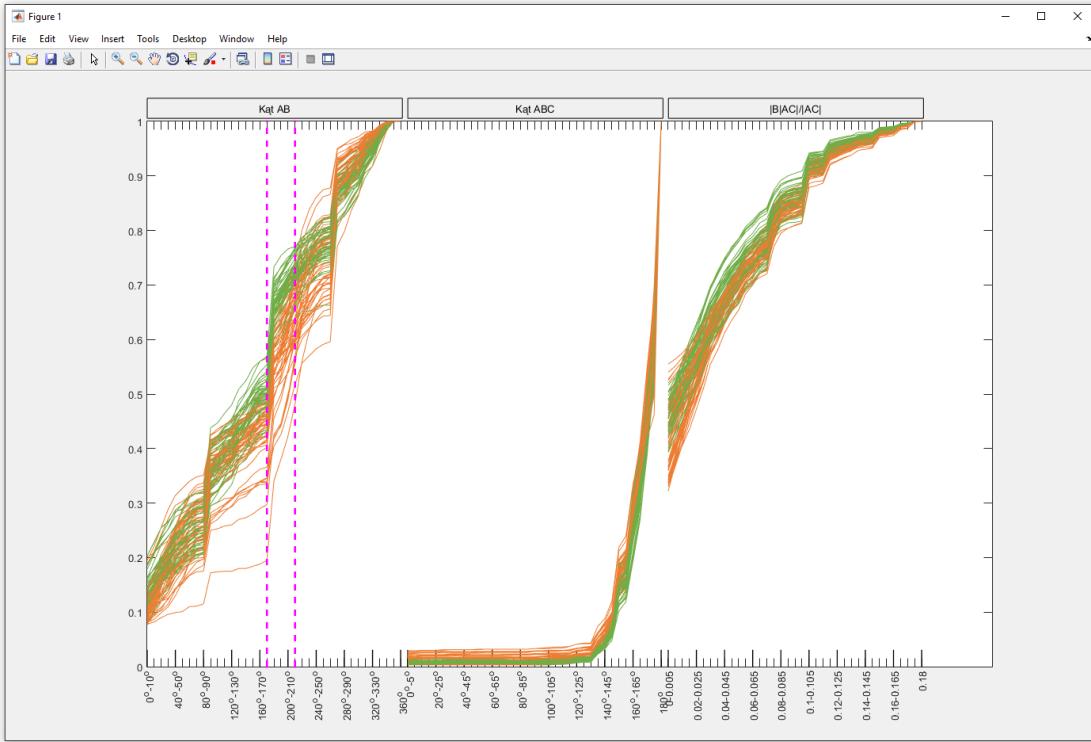
Rys. 55. Wykresy próbek treningowych użytkowników czarnego i pomarańczowego.



Rys. 56. Wykresy próbek treningowych użytkowników czarnego i niebieskiego.



Rys. 57. Wykresy próbek treningowych użytkowników zielonego i niebieskiego.

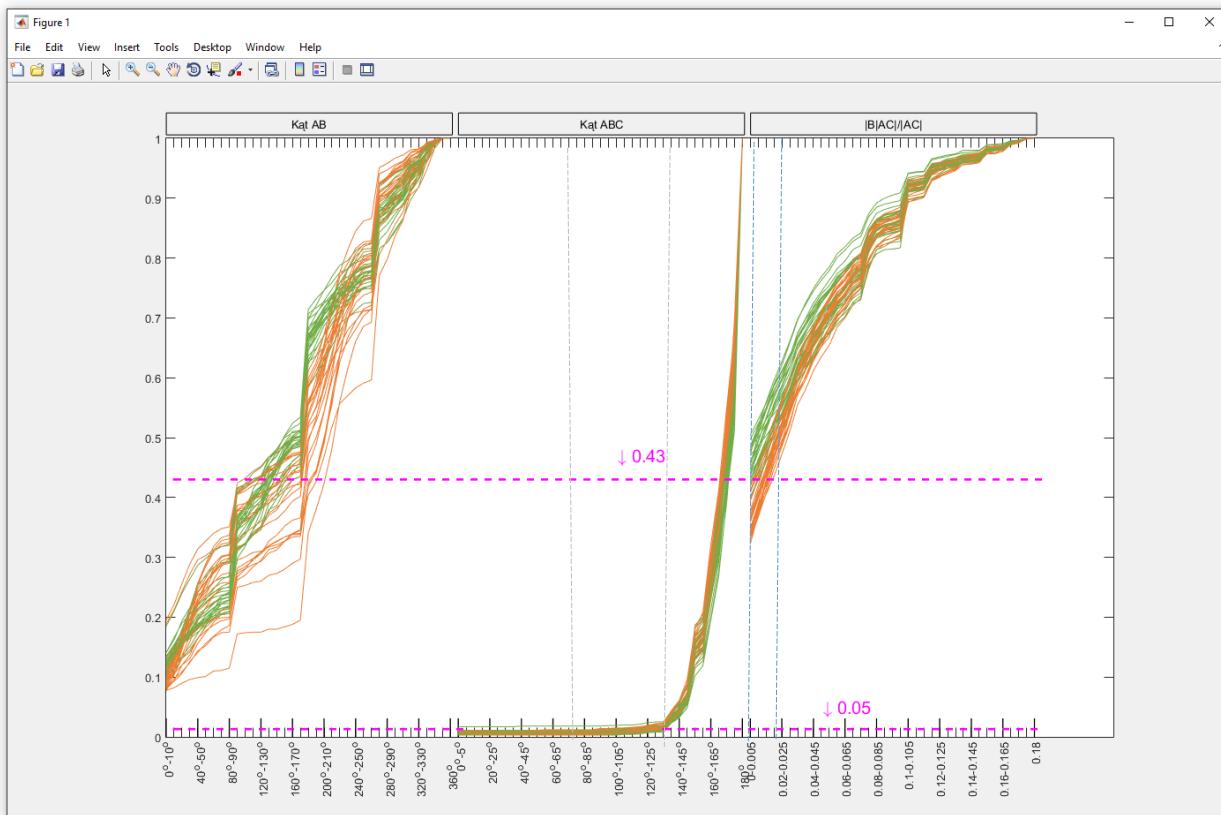


Rys. 58. Trudno rozróżnialne wykresy użytkowników pomarańczowego i zielonego.

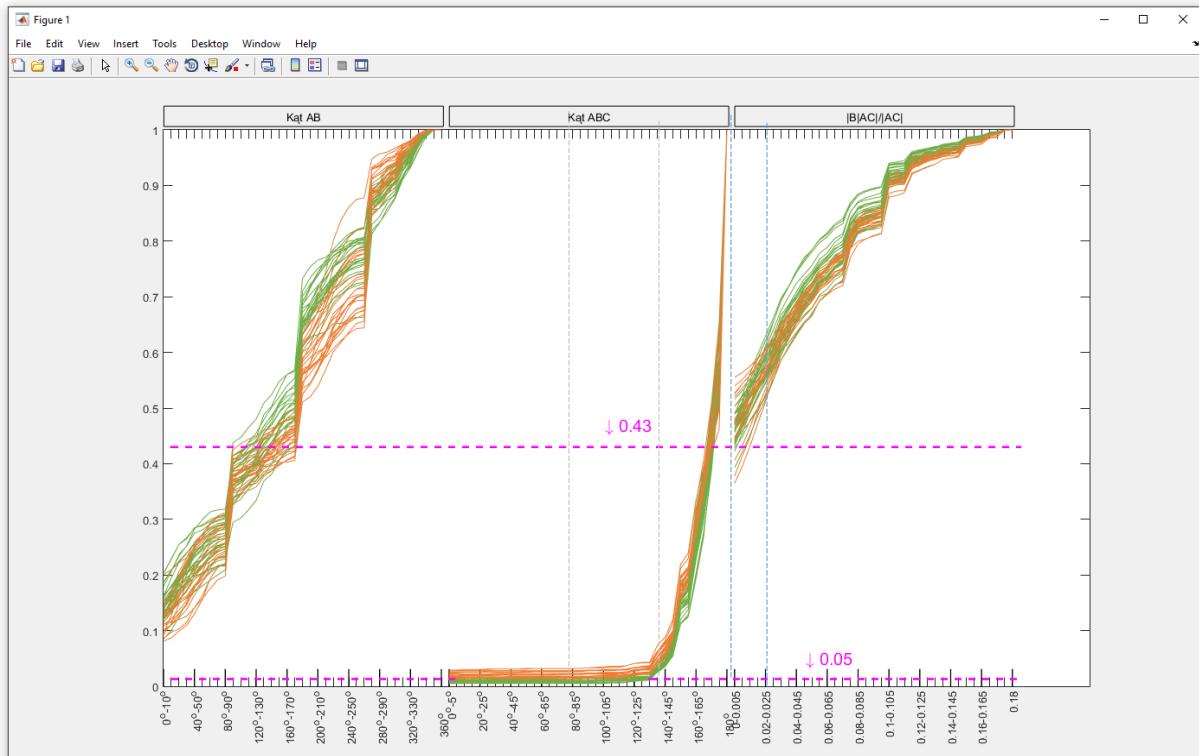
Przedstawione na rysunku 58 wykresy dystrybuant nie są tak łatwo rozróżnialne jak poprzednie. Wyróżnia się kilka przedziałów kąta AB. W innych prążkach natomiast, sporo wykresów próbek nakłada się na siebie. Aby sprawdzić jakość separacji przeanalizowano podzbiory danych z rysunku 58. Zbiór dystrybuant 1 – 24 (Rysunek 59) użytkownika pomarańczowego i zielonego mają nieco inną charakterystykę niż 25 – 49 (Rysunek 60). Analizując je można zauważać, że dla użytkownika oznaczonego kolorem zielonym początkowe wartości prawdopodobieństwa dla kąta ABC prawie nigdy nie osiągają wartości przekraczających 0,5 zarówno dla dystrybuanty 1 – 24 jak i 25 – 49. Jednocześnie wartość stosunku odległości jest niemal zawsze większa od 0,43.

Natomiast wielkości te dla użytkownika oznaczonego kolorem pomarańczowym zmieniają się w zależności od zbioru dystrybuant. Kąt ABC jest początkowo zawsze mniejszy od 0,05 dla pierwszego z nich, podczas gdy wzrasta od tej wartości dla zakresu 25 – 49. Podobnie zachowuje się wykres stosunku odległości; dla dystrybuanty 1 – 24 wartości początkowe nie przekraczają 0,43 oraz są niemal zawsze większe od tej liczby dla drugiego zbioru dystrybuant.

Oznacza to, że komplet 98 dystrybuant jest rzeczywiście rozróżnialny. Można bowiem wyznaczyć wykresy właściwe tylko dla użytkownika pomarańczowego i jedynie dla zielonego.



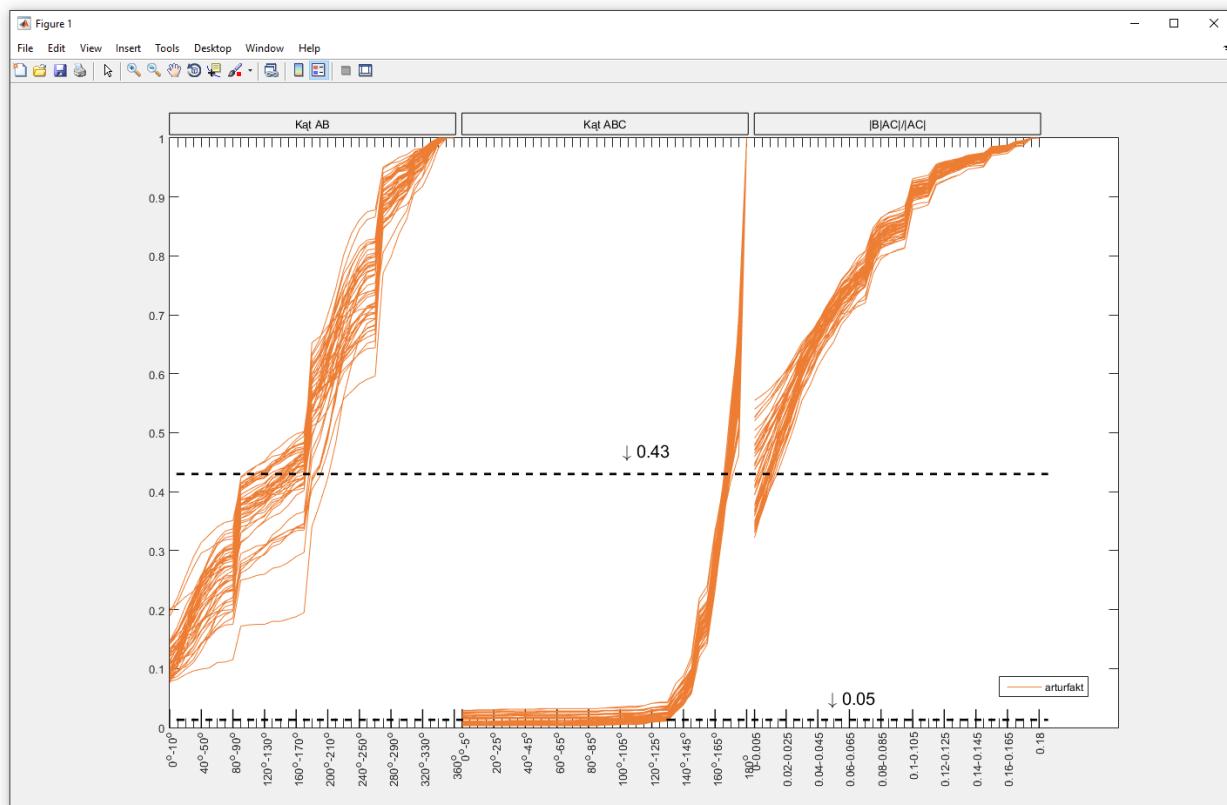
Rys. 59. Trudno odróżnialne wykresy użytkowników pomarańczowego i zielonego. Próbki 1-24.



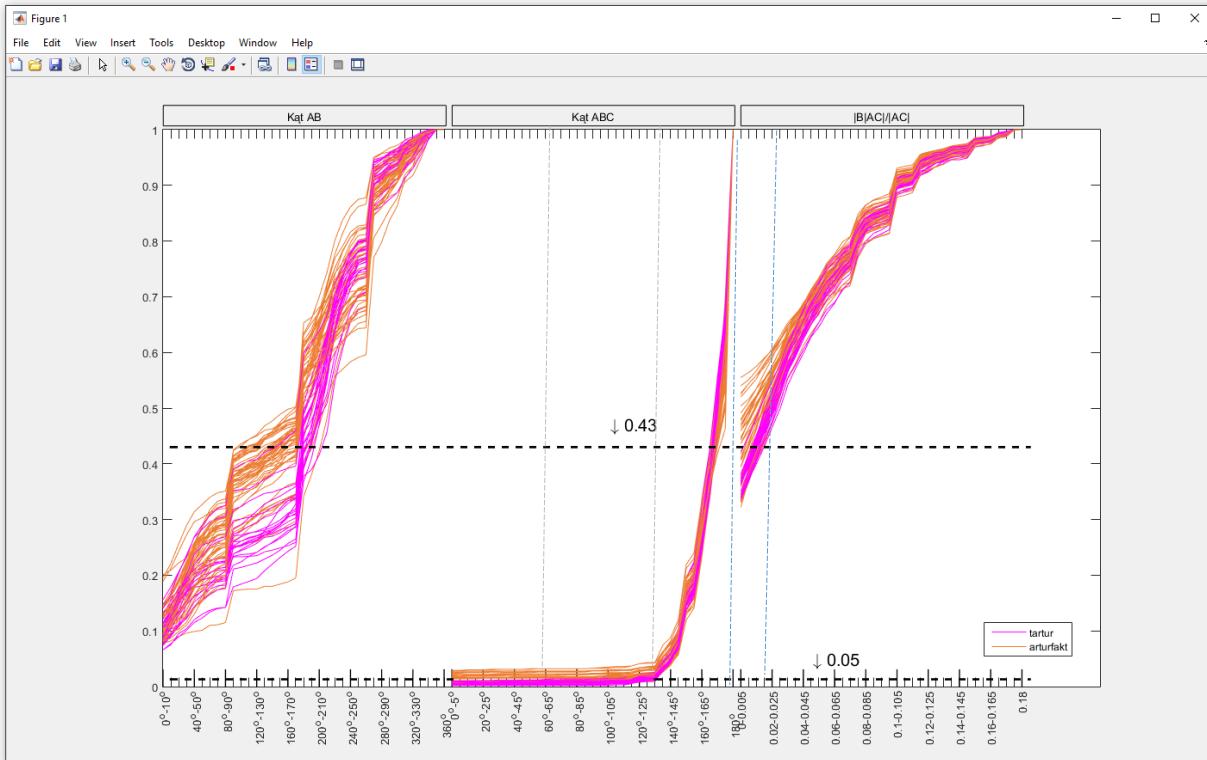
Rys. 60. Trudno odróżnialny wykres użytkowników pomarańczowego i zielonego. Próbki 25-49.

Najważniejszym etapem weryfikacji jakości odseparowania dystrybuant jest porównanie próbek testowych użytkownika do jego zbioru treningowego. Maszyna wektorów wspierających poprawnie klasyfikuje próbki testowe, których wykresy pokrywają się w możliwie najwyższym stopniu z wykresami zbioru treningowego. Próbki treningowe muszą więc reprezentować wszystkie możliwe zachowania osób podczas ruchów urządzeniami wskazującymi. Wykresom dystrybuant testowych przyporządkowano kolor magenty.

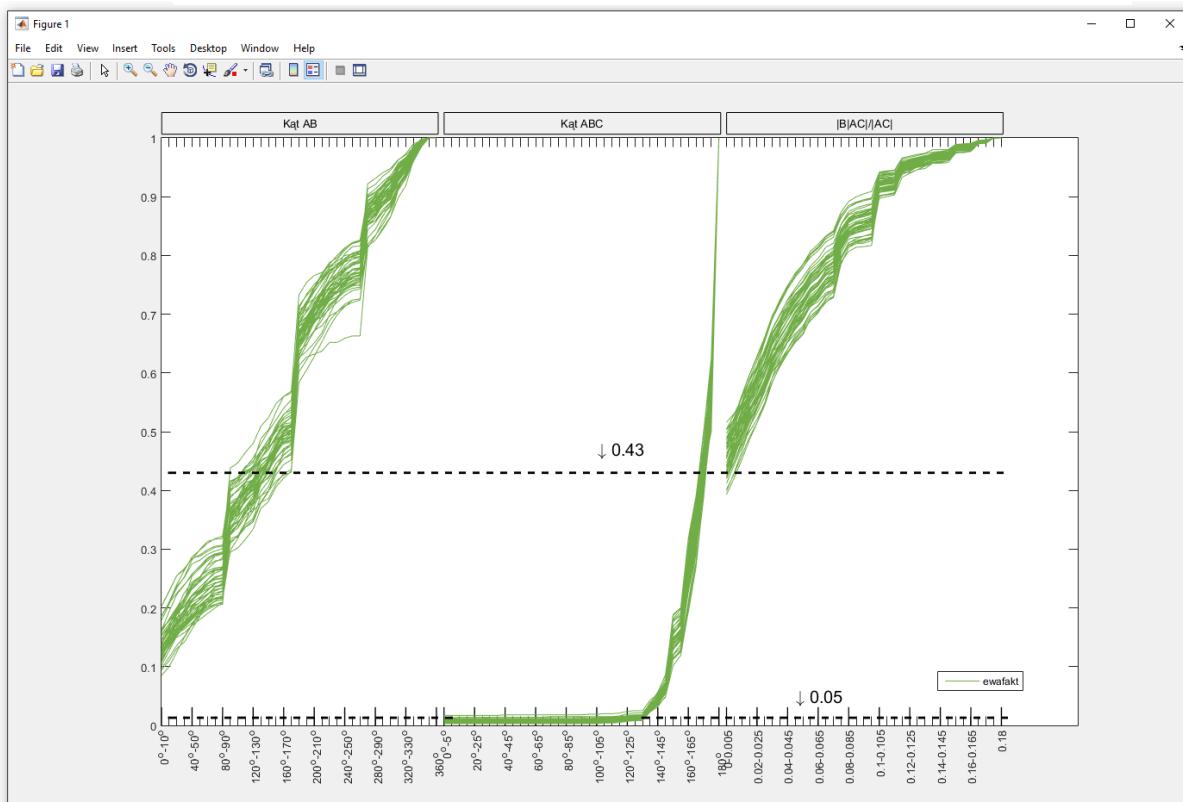
Rysunki 61, 62, 63, 64 potwierdzają wnioski z obserwacji próbek treningowych użytkowników pomarańczowego i zielonego. Dla wartości początkowych kątów  $ABC < 0,43$  i stosunków odległości  $< 0,05$  próbki testowe użytkownika pomarańczowego wpasowują się w odpowiednich miejscach w zakres jego dystrybuant treningowych. Podobnie dla użytkownika zielonego, którego dopuszczalne przedziały wynoszą  $< 0,05$  i  $> 0,43$ .



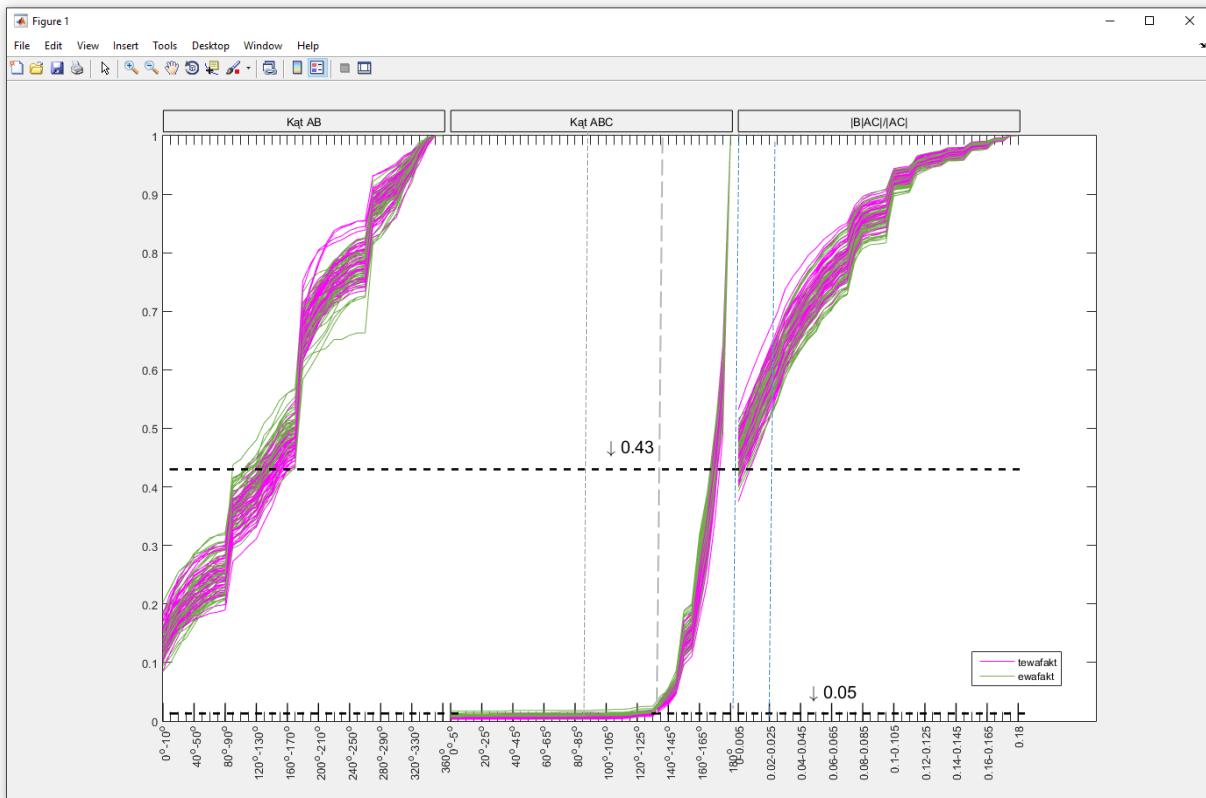
Rys. 61. Zbiór treningowy użytkownika pomarańczowego.



Rys. 62. Próbki testowe użytkownika pomarańczowego na jego zbiorze treningowym.

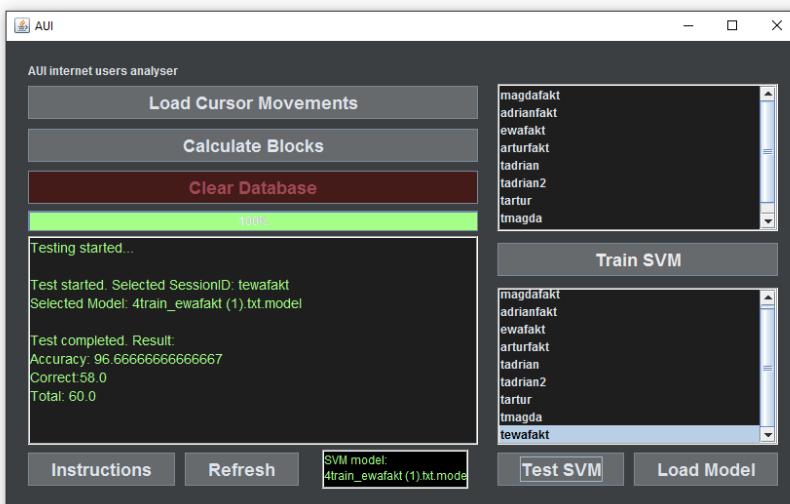


Rys. 63. Zbiór treningowy użytkownika zielonego

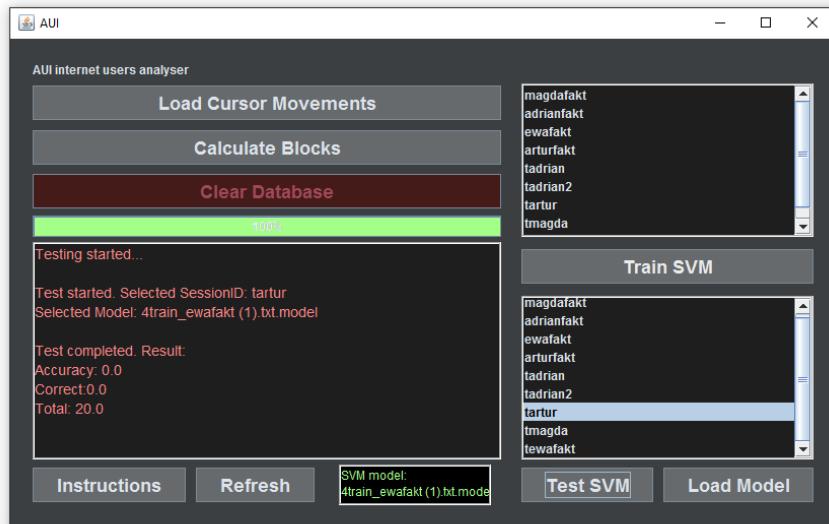


Rys. 64. Próbki testowe użytkownika zielonego na zbiorze treningowym.

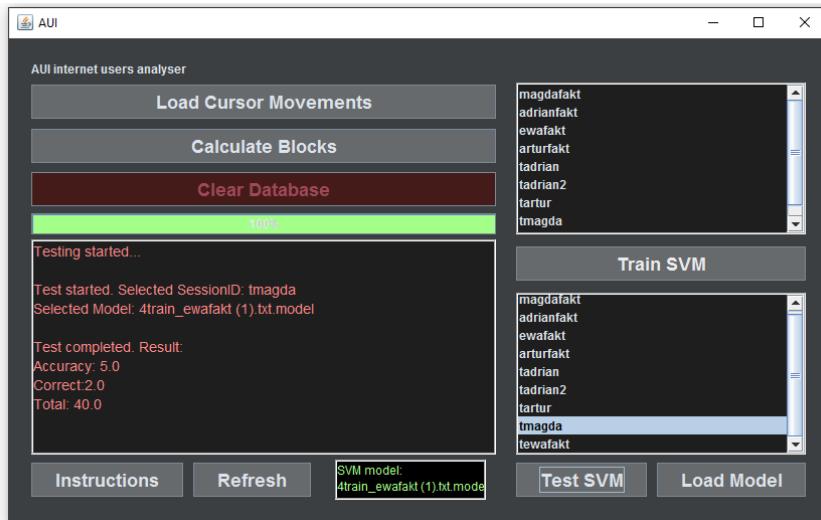
Na rysunkach 65, 66, 67, 68 widać zrzuty ekranowe aplikacji z wyświetlonymi wynikami dopasowania próbek testowych wszystkich osób do zbioru treningowego użytkownika zielonego. Rysunki 69, 70, 71, 72 przedstawiają analogiczne wyniki dla użytkownika pomarańczowego. Pozostałe zbiory testowe i treningowe przedstawiono na rysunkach 73, 74, 75, 76.



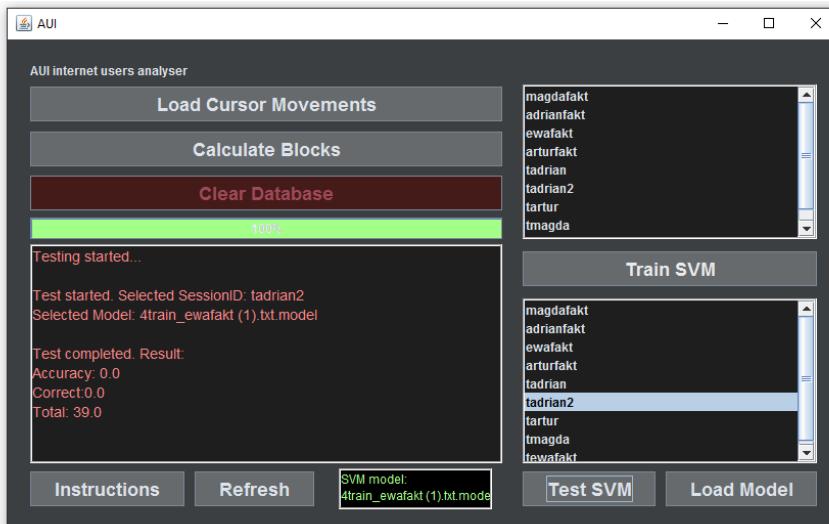
Rys. 65. Próbki testowe użytkownika zielonego vs zbiór treningowy użytkownika zielonego.



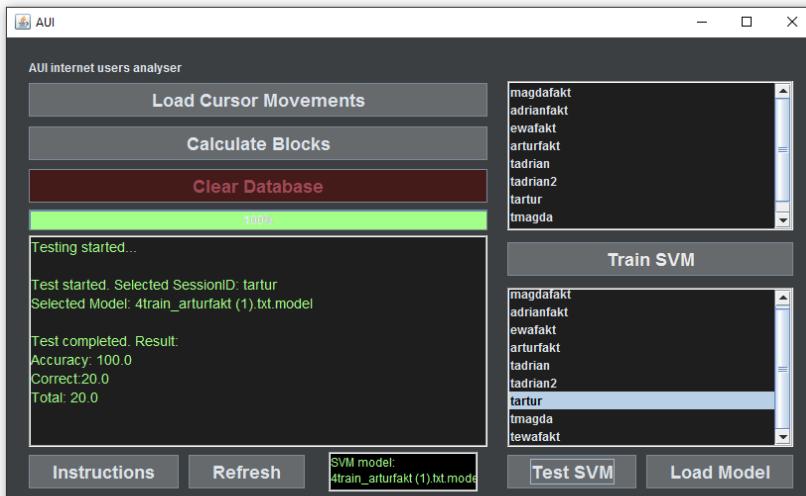
Rys. 66. Próbki testowe użytkownika pomarańczowego vs zbiór treningowy użytkownika zielonego.



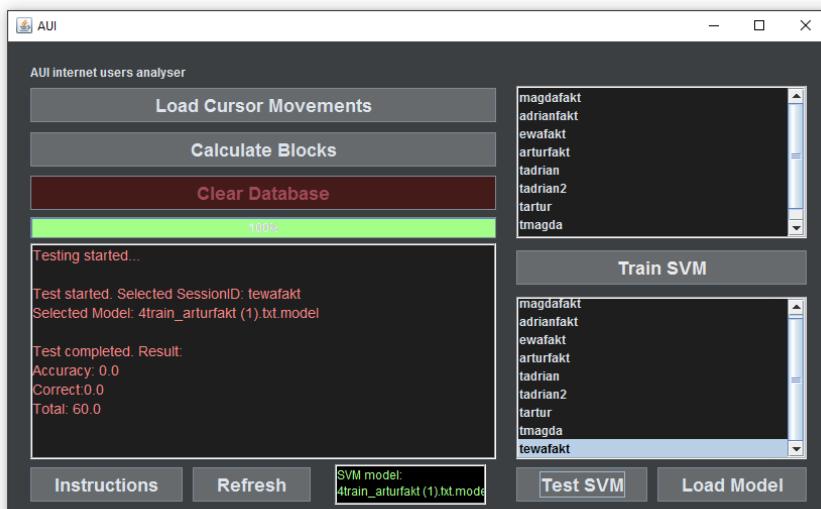
Rys. 67. Próbki testowe użytkownika czarnego vs zbiór treningowy użytkownika zielonego.



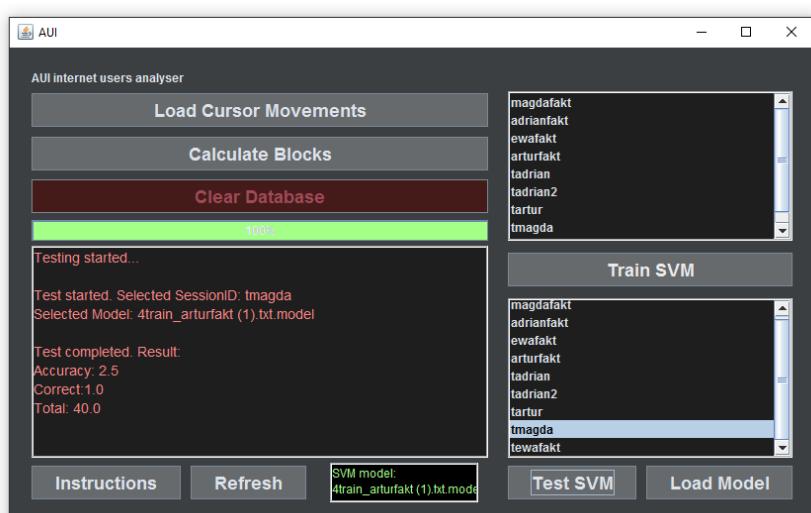
Rys. 68. Próbki testowe użytkownika niebieskiego vs zbiór treningowy użytkownika zielonego.



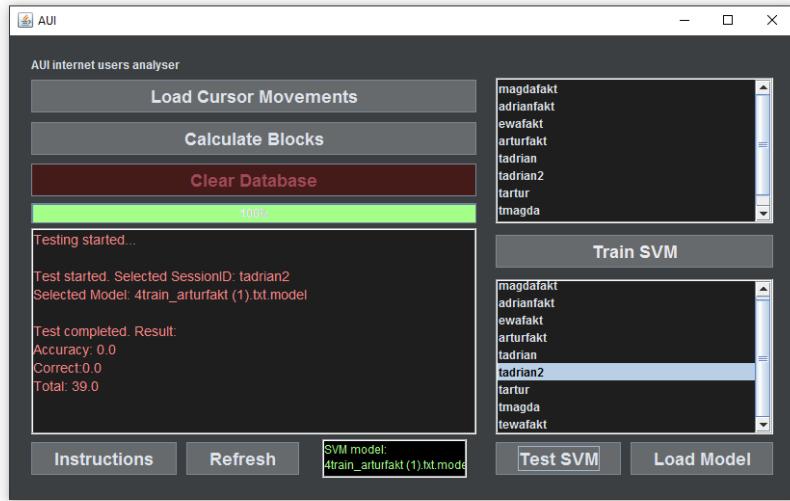
Rys. 69. Próbki testowe użytkownika pomarańczowego vs zbiór treningowy użytkownika pomarańczowego.



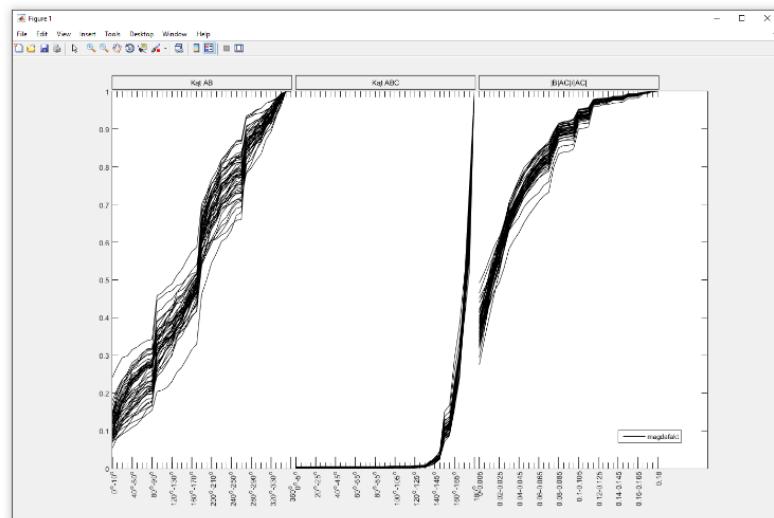
Rys. 70. Próbki testowe użytkownika zielonego vs zbiór treningowy użytkownika pomarańczowego.



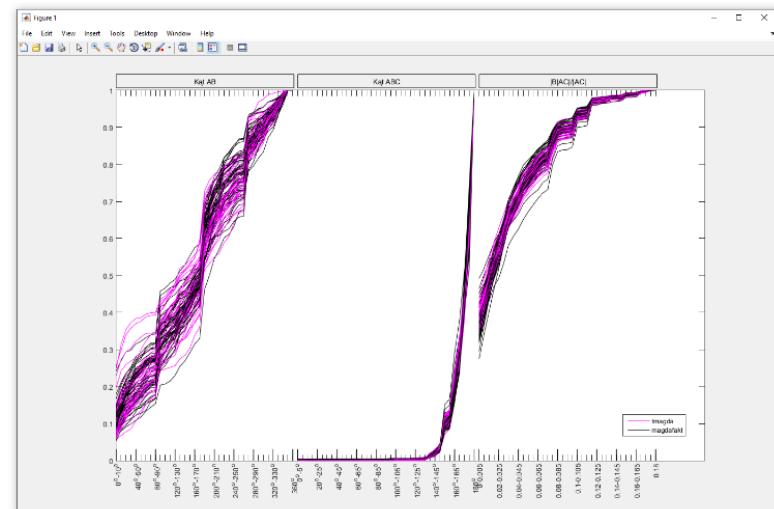
Rys. 71. Próbki testowe użytkownika czarnego vs zbiór treningowy użytkownika pomarańczowego.



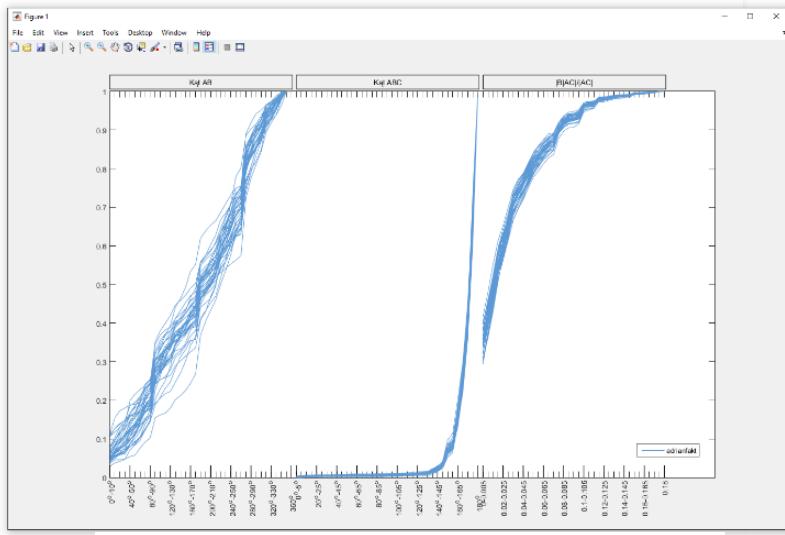
Rys. 72. Próbki testowe użytkownika niebieskiego vs zbiór treningowy użytkownika pomarańczowego.



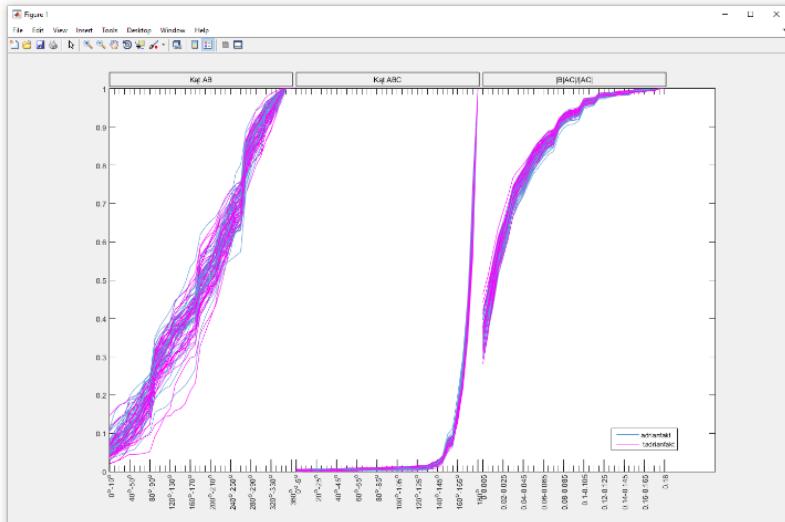
Rys. 73. Próbki treningowe użytkownika czarnego.



Rys. 74. Próbki testowe użytkownika czarnego na jego zbiorze treningowym.



Rys. 75. Próbki treningowe użytkownika niebieskiego.



Rys. 76. Próbki testowe użytkownika niebieskiego na jego zbiorze treningowym.

Tabela zawierająca wyniki dopasowania wszystkich próbek testowych do kompletu zbiorów treningowych użytkowników widoczna jest na rysunku 77. Największy błąd FRR (ang. *False Reject Rate*) wyniósł 4,55%, a FAR (ang. *False Accept Rate*) 5%.

	Użytkownik niebieski – tadrian + tadrian2	Użytkownik pomarańczowy – tartur	Użytkownik zielony – tewafakt	Użytkownik czarny – tmagda
Użytkownik niebieski – adrianfakt	<b>95.45%</b>	<b>0%</b>	<b>0%</b>	<b>2.5%</b>
Użytkownik pomarańczowy – arturfakt	<b>0%</b>	<b>100%</b>	<b>0%</b>	<b>2.5%</b>
Użytkownik zielony – ewafakt	<b>0%</b>	<b>0%</b>	<b>96.67%</b>	<b>5%</b>
Użytkownik czarny – Magdafakt	<b>0%</b>	<b>0%</b>	<b>0%</b>	<b>97.5%</b>

Rys. 77. Tabela wyników dopasowania wszystkich próbek testowych do zbiorów treningowych.

#### **4.3. Test wydajności aplikacji**

Czas załadowania do bazy danych wszystkich ruchów urządzeń wskazujących (Load Cursor Movements) wyniósł 5 min. 44 s. Obliczenie kompletu próbek (Calculate Blocks) zajęło 10 min. 30 s. Mediana czasu trenowania (Train SVM) wszystkich użytkowników osiągnęła wartość 11 sekund. Każdy test (Test SVM) trwał około 1 s. Pomiary wykonano na urządzeniu z procesorem i5 – 6300HQ, 12 GB DDR3 8GB 1600MHz CL11 i dyskiem SSD 540/520

### **5. Podsumowanie**

Przedmiotem niniejszej pracy było opracowanie systemu, którego zadaniem jest identyfikacja użytkownika serwisu internetowego pomimo zmiany jego danych takich jak adres IP, właściwości używanego systemu operacyjnego i przeglądarki. Istotnym aspektem było użycie w tym celu technologii webowej (HTML, AJAX, JavaScript), gromadzącej dane przy korzystaniu z sieci internetowej. Jej zadaniem było zbieranie odpowiednich informacji w sposób transparentny i nieuciążliwy dla użytkownika w ogólnodostępnych miejscach takich jak witryna www.

Przeprowadzone testy potwierdziły, że utworzenie systemu spełniającego postawione założenia, tj. szybkość, niezawodność rejestracji danych oraz skuteczna identyfikacja jest możliwym do zrealizowania. Zapis ruchów urządzeń wskazujących był bezstratny i transparentny dla użytkownika w warunkach standardowego użytkowania strony www. W przypadku nienaturalnie szybkiego opuszczania podstron, wykorzystywano skrypt opóźniający załadowanie nowej strony. Zastosowanie buforowania danych w pamięci operacyjnej po stronie serwera powinno wystarczająco zabezpieczyć system przed utratą informacji. Użycie dedykowanej strony www minimalizuje obliczenia wejść i treningu klasyfikatora o czas załadowania ruchów zewnętrznym programem. Czynność ta ostatecznie trwa około jedenastu minut dla wszystkich czterech użytkowników. Algorytmy odpowiedzialne za przygotowanie wejść zostały zaprojektowane tak, aby umożliwić łatwe debugowanie programu. W celu przyspieszenia jego działania można np. zmniejszyć liczbę serii wpisów i odczytów z bazy danych. Zaznaczyć należy, że nie jest to jedyny sposób zredukowania złożoności czasowej i obliczeniowej. Skuteczność identyfikacji użytkowników wyniosła w najgorszym przypadku 95%. W pracach [15] i [26] użyto

bardziej zaawansowanych strategii SVM, co zaowocowało wynikami FAR i FRR odpowiednio 3,33%, 1,3% i 2,12%, 1,3%. Co więcej, klasyfikatory zaprogramowane zostały do rozpoznawania ponad trzydziestu osób. Warto podkreślić, że aplikacje tego typu były z powodzeniem łączone z innymi mechanizmami rejestrującymi i analizującymi zachowania osób użytkujących klawiatury komputerowe [27].

Wzrost mocy obliczeniowej procesorów oraz rozwój aplikacji wykorzystujących standardy W3C, sprzyja ulepszaniu istniejących i powstawaniu nowych aplikacji, służących do analizy zachowań użytkowników. W niedalekiej przyszłości mogą pojawiać się nowe urządzenia, które w jeszcze większym stopniu będą wykorzystywać interakcję człowieka z takim urządzeniem. Tym samym pozwolą na zebranie jeszcze bardziej indywidualnych cech i zachowań użytkowników. Nietrudno jest wyobrazić sobie jakie możliwości analizy indywidualnych cech ludzi mogą mieć takie rozwiązania jak np. okulary czy hełmy rozszerzonej rzeczywistości. Szczególnie że rozpowszechnienie tego typu innowacji, wydaje się być jedynie kwestią czasu.

## Bibliografia

- [1] Nikiforakis N., Kapravelos A., Joosen W., Kruegel C., Piessens F., and Vigna G., *Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting*, IEEE Symposium on Security and Privacy, 2013, s. 1–15.
- [2] Stanic M., *Continuous User Verification Using Mouse Dynamics*, Proceedings of the ITI 2013 35th International Conference on, Cavtat, Croatia, 2013, s. 251 – 256.
- [3] Hogan B., *HTML5 and CSS3: Develop with Tomorrow's Standards Today*, Pragmatic Programmers, edycja I, wyd. Pragmatic Bookshelf, 17.01.2011r.
- [4] Dokumentacja MariaDB <https://mariadb.com/kb/en/mariadb/documentation/>
- [5] Dokumentacja JAVA <https://docs.oracle.com/javase/specs/jls/se8/html/index.html>
- [6] Dokumentacja IntelliJ IDEA <https://www.jetbrains.com/idea/documentation/>
- [7] Dokumentacja JDBC <http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/>
- [8] Dokumentacja LIBSVM <http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>
- [9] Dokumentacja Swing <http://docs.oracle.com/javase/7/docs/technotes/guides/swing/>
- [10] Ostasiewicz W., *Myslenie statystyczne*, Wolters Kluwer, Warszawa, 2012, s. 56 – 57.
- [11] Schulz D.A., *Mouse Curve Biometrics*, Biometric Consortium Conference, 2006 s. 1–6.
- [12] Hinbarji Z., Albatal R., Gurrin C., *Dynamic user authentication based on mouse movements curves*, 21st International Conference on MultiMedia Modelling (MMM 2015), Sydney, Australia, 2015.
- [13] Wykład MIT o SVM <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-16-learning-support-vector-machines/>
- [14] Prezentacja dot. SVM <http://www.cs.put.poznan.pl/pboinski/files/ED/SVM.pdf>
- [15] Zheng N., Paloski A., Wang H.M., *An efficient user verification system via mouse movements*, Proc. ACM Conf. Computer and Communications Security, Chicago, IL, 2011, s. 139–150.
- [16] Artykuł dot. SVM  
[http://www.pszw.edu.pl/images/publikacje/t019\\_pszw\\_2009\\_budzinski\\_misztal\\_-\\_zastosowanie\\_algorytmu\\_maszyny\\_wektorow\\_wspierajacych\\_do\\_klasyfikacji\\_podatnikow\\_z\\_wykorzystaniem\\_bazy\\_danch\\_oracle\\_11g.pdf](http://www.pszw.edu.pl/images/publikacje/t019_pszw_2009_budzinski_misztal_-_zastosowanie_algorytmu_maszyny_wektorow_wspierajacych_do_klasyfikacji_podatnikow_z_wykorzystaniem_bazy_danch_oracle_11g.pdf)

- [17] Keerthi, S. S. and C.-J. Lin, *Asymptotic behaviors of support vector machines with Gaussian kernel*, Neural Computation 15 (7), s. 1667–1689.
- [18] Artykuł dot. SVM, o miękkim marginesie  
<http://www.cs.put.poznan.pl/jstefanowski/ml/SVM.pdf>
- [19] <https://www.w3.org/TR/hr-time/#sec-DOMHighResTimeStamp>, 11.07.2016
- [20] Smith P., *Professional Website Performance: Optimizing the Front-End and Back-End*, John Wiley & Sons, Indianapolis, s. 20-21.
- [21] Kod źródłowy jQuery <https://github.com/jquery/jquery/blob/master/src/ajax.js#L574>,  
11.07.2016
- [22] Dokumentacja PHP, funkcja haszująca  
<http://php.net/manual/en/session.configuration.php#ini.session.hash-function> , 11.07.2016
- [23] Mitchell L., *PHP Web Services*, 2, O'Reilly Media, Sebastopol, 2016, s. 82
- [24] Strona internetowa XAMPP <https://www.apachefriends.org/index.html>
- [25] Kukreja U., Stevenson W.E., Ritter F.E., *RUI: Recording user input from interfaces under windows and mac os x. Behavior Research Methods*, 2006, s. 38(4):656–659.
- [26] Shen C., Cai Z., Guan X., Du Y., Maxion R.A., *User authentication through mouse dynamics*, IEEE Trans. Inf. Forensics Security, tom 8, 2013, s. 16-30.
- [27] Bailey, K.O., *Computer Based Behavioral Biometric Authentication via MultiModal Fusion*, 2013.

## **Dodatek A: Spis zawartości dołączonej płyty CD**

Tekst pracy inżynierskiej w pliku PDF.

Kod źródłowy aplikacji klasyfikującej użytkowników w formie projektu IntelliJ IDEA wraz ze wszystkimi próbками.

Kody źródłowe witryny internetowej.