

Raport końcowy projektu z przedmiotu Dedykowane Algorytmy Diagnostyki Medycznej

Katarzyna Kokosza
Olga Janiszewska
Kacper Banach
Jędrzej Chiliński
Joanna Lebica
Piotr Olbrot
Iwona Struglik
Joanna Żuradzka
Aleksandra Rak
Kamelia Niemczyk
Katarzyna Krzywonos
Aleksandra Lubzińska

12 stycznia 2017



Klasyfikacja uderzeń serca

Spis treści

1	Abstrakt	4
2	Linear SVM	5
2.1	Opis algorytmu	5
2.1.1	Metoda wektorów wspierających (SVM)	5
2.1.2	Sekwencyjna minimalna optymalizacja (SMO)	9
2.1.3	Poszukiwanie współczynników Lagrange’a	9
2.1.4	Heurystyka wyboru współczynników do optymalizacji	10
2.1.5	Obliczanie progów	11
2.2	Opis sposobu implementacji algorytmu	11
2.2.1	Schematy blokowe działania algorytmu	11
2.3	Wizualizacja działania algorytmu	15
2.3.1	Wizualizacja dla dwóch cech	15
2.3.2	Wizualizacja dla trzech cech	16
2.4	Opis informatyczny procedur	17
3	Klasyfikator Bayesa	22
3.1	Opis algorytmu	22
3.2	Implementacja	22
3.3	Wykorzystanie klasyfikatora do rozpoznawania uderzeń serca	23
3.4	Podsumowanie	24
4	k-Nearest Neighbours	25
4.1	Założenia wstępne metody	25
4.2	Opis metody	25
4.3	Klasy	25
4.4	Implementacja prototypu algorytmu w środowisku MATLAB	26
5	Extended Nearest Neighbours	26
5.1	Ograniczenia metody k-NN	26
5.2	Opis metody eNN	26
5.3	Implementacja w środowisku MatLab	27
5.4	Podsumowanie	27
6	Radial Basis Function Kernel SVM	28
6.1	Wstęp	28
6.2	Zarys proponowanego rozwiązania	28
6.3	Obliczenie cech kompleksów QRS	28
6.3.1	Matematyczny sposób obliczenia cech	28
6.3.2	Klasyfikacja przy użyciu obliczonych cech	29
6.4	Metoda maszyny wektorów nośnych (SVM)	31
6.4.1	Podstawowe założenia metody	31
6.4.2	Radial Basis Function	32
6.4.3	Matematyczne podstawy implementacji	32
6.4.4	Implementacja w C++	33
6.5	Wyniki	34
6.6	Wnioski	36
7	Liniowa Analiza Dyskryminacyjna	37
7.1	Wyznaczenie klasyfikatora	37
7.2	Matlab - implementacja	38
7.3	C++ - implementacja	38
7.4	Podsumowanie	38

8	Quadratic SVM	40
8.1	SMO	40
8.1.1	Opis ogólny procedury	40
8.1.2	Rozwiązanie analityczne dla dwóch punktów	40
8.1.3	QSVM (ang. quadratic kernel-free non-linear support vector machine)	41
8.2	Implementacja	42
8.3	Wykorzystanie klasyfikatora do rozpoznawania uderzeń serca	43
9	Porównanie algorytmów	44
9.1	Opis zbiorów danych	44
9.1.1	Zbiór danych 1	44
9.1.2	Zbiór danych 2	44
9.2	Skuteczność klasyfikacji	44
9.3	Czas uczenia i klasyfikacji	45
9.4	Czułość i specyficzność klasyfikatorów	45
9.5	Podsumowanie	46

1 Abstrakt

Celem projektu jest detekcja oraz klasyfikacja kardiologicznych stanów pacjentów na podstawie sygnału EKG. Poprawne sklasyfikowanie występujących uderzeń serca jest bardzo ważne i tak naprawdę może być podstawą do stwierdzenia o występowaniu arytmii. Wykrywając niepokojące zmiany w przebiegu EKG u danego pacjenta można odpowiednio wcześniej zastosować leczenie, co na pewno pozwoli uniknąć poważniejszych problemów natury kardiologicznej w przyszłości.

Arytmie serca, czyli wszystkie odchylenia od normy w przebiegu EKG można podzielić na dwie grupy: te, które wymagają natychmiastowej interwencji lekarza oraz takie, w których natychmiastowe leczenie nie jest konieczne. Jednak pominięcie leczenia w drugim przypadku również może w przyszłości prowadzić do poważniejszych stanów, w tym arytmii pierwszej grupy. Klasyfikacja uderzeń, która zostanie przedstawiona w tym projekcie może być pomocą do rozpoznawania arytmii serca należącej do drugiej grupy. Sklasyfikowanie występujących rodzajów bicia serca jest podstawowym krokiem do stwierdzenia wystąpienia odchyleń w tej dziedzinie. Klasyfikacja ta jest bardzo czasochłonna i dlatego podjęto wiele prób w celu zautomatyzowania tego procesu .

Projekt realizuje klasyfikację próbek sygnału EKG pochodzących od pacjenta za pomocą następujących klasyfikatorów: k-Neraest Neighbours (kNN), Extended Nerest Neighbours (eNN), Radial Basis Function Kernel (RBF SVM), klasyfikator Bayesa, Linear SVM, Quadratic SVM. Dla każdego algorytmu przedstawiono ich opisy oraz sposoby implementacji. Algorytmy działają na zasadzie uczenia – po wczytanie zbioru treningowego następuje uczenie, po którym następuje właściwa klasyfikacja dla zbioru testowego. Otrzymane wyniki (klasy dla poszczególnych próbek zbioru testowego) porównano z rzeczywistymi (właściwymi) klasami dla tych próbek, których wartości były znane. Dokonano porównania działania algorytmów na dwóch identycznych zbiorach danych, mierząc skuteczność, czas uczenia i klasyfikacji oraz czułość i specyficzność.

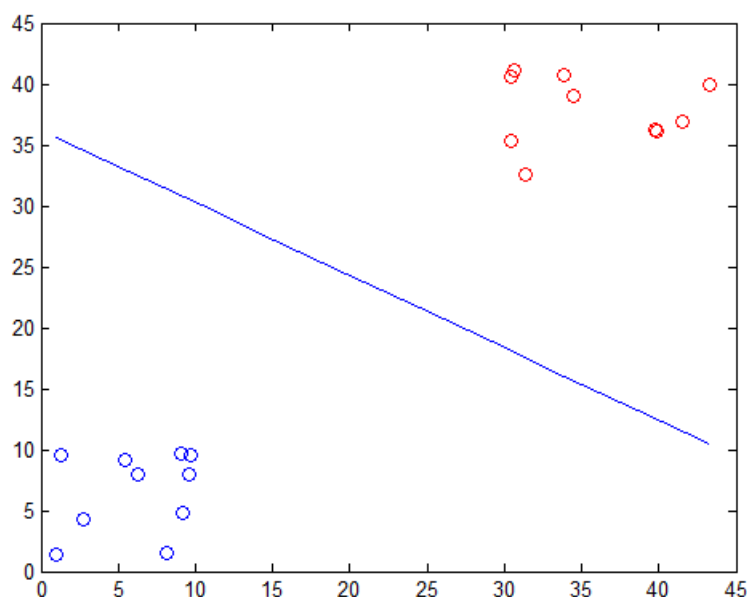
2 Linear SVM

2.1 Opis algorytmu

2.1.1 Metoda wektorów wspierających (SVM)

Maszyny wektorów wspierających (support vector machines) są modelami uczenia nadzorowanego, które analizują dane użyte do klasyfikacji i analizy regresji. Wykorzystując zbiór danych uczących, każdy odpowiednio oznaczony zgodnie z klasą do której przynależy, algorytm SVM tworzy model, który potrafi przyporządkować analizowane dane do jednej z kategorii.

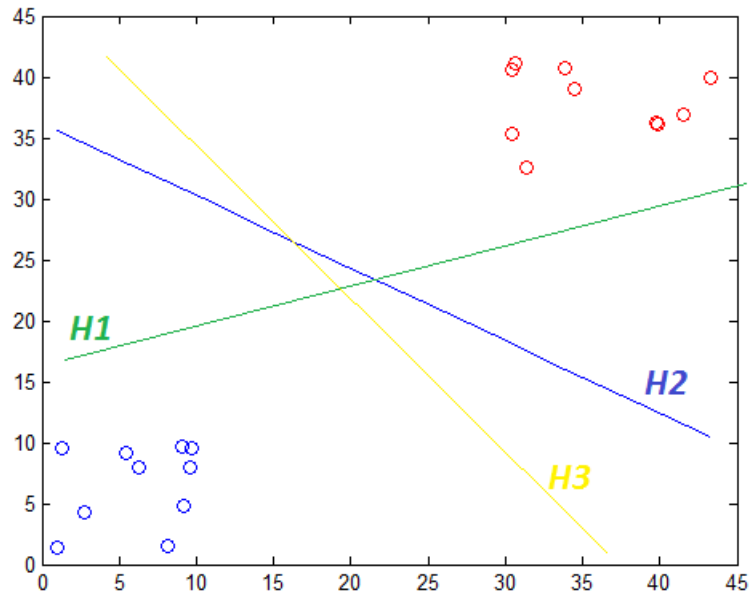
Model SVM jest reprezentacją danych w postaci punktów w przestrzeni. Punkty należące do różnych klas znajdują się w innych obszarach przestrzeni i są rozdzielone za pomocą luki o jak największej możliwej szerokości. Kolejno, analizowany nowy punkt jest odpowiednio przyporządkowywany do jednej z kategorii na podstawie tego, z której strony luki się znajduje. Na rysunku nr 1 został przedstawiony rozkład punktów płaszczyzny należących do dwóch różnych kategorii. W tym wypadku na osiach znajdują się cechy, dzięki którym klasyfikowane są analizowane przypadki.



Rysunek 1: Wizualizacja punktów należących do dwóch klas wraz z rozdzielającą je hiperpłaszczyzną
(opracowanie własne)

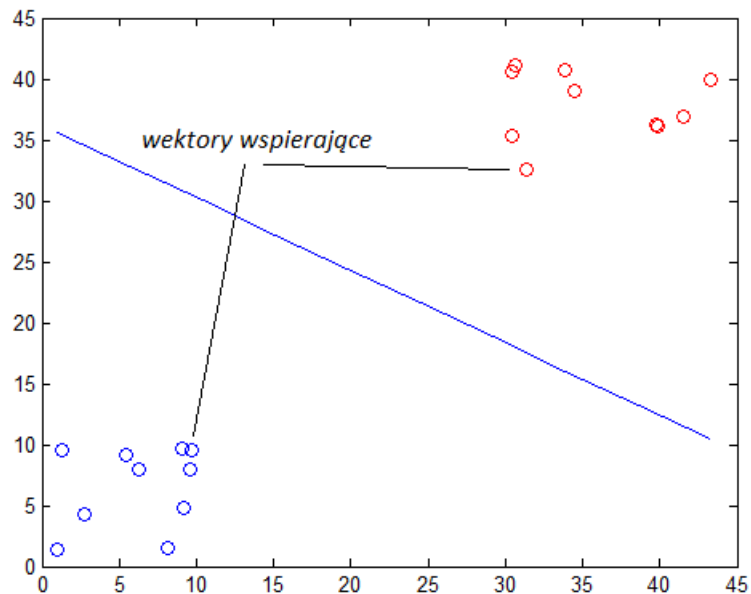
Klasyfikacja danych jest najważniejszym zadaniem uczenia maszynowego. W metodzie wektorów wspierających dane są postaci wektora o rozmiarze p . Celem działania algorytmu jest wyznaczenie optymalnej hiperpłaszczyzny rozmiaru $p - 1$, która rozdzielałaby dane należące do poszczególnych klas. Tego typu podejście jest nazywane klasyfikacją liniową.

Istnieje wiele hiperpłaszczyzn, które mogą rozdzielać dwie klasy. Przykładowo na rysunku 2 przedstawiono dwie klasy rozdzielone trzema różnymi hiperpłaszczyznami H_1, H_2, H_3 . Problem polega na tym, aby wybrać jak najlepszą hiperpłaszczyznę, która w sposób najbardziej optymalny będzie rozdzielać dwie klasy. Taka hiperpłaszczyzna zostaje wybrana w taki sposób, aby była jak najbardziej oddalona zarówno od jednej jak i drugiej klasy.



Rysunek 2: Wizualizacja trzech różnych hiperpłaszczyzn mogących rozdzielać dwie klasy
(opracowanie własne)

Klasyfikator maksymalnego marginesu znajduje hiperpłaszczyznę rozdzielającą dane treningowe na dwie klasy w ten sposób, że maksymalizuje wartość marginesu geometrycznego dla wszystkich punktów treningowych. Marginesem geometrycznym hiperpłaszczyzny jest jej odległość od najbliższych punktów. Punkty położone najbliżej hiperpłaszczyzny są nazywane wektorami wspierającymi (ang. *support vectors*). Wektory wspierające zostały zaznaczone na rysunku 3.



Rysunek 3: Wektory wspierające oznaczone wśród punktów treningowych
(opracowanie własne)

Klasyfikator maksymalnego marginesu jest klasyfikatorem liniowym i może być użyty do klasyfikacji danych, które są liniowo separowalne. W niniejszym projekcie przyjęto założenie, że separowane dane należą do dwóch klas. Klasyfikator maksymalnego marginesu jest klasyfikatorem binarnym. Załóżmy teraz, że mamy zbiór danych uczących składających się z n punktów postaci:

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_n, y_n) \quad (1)$$

gdzie y_i to 1 lub -1 w zależności od klasy, do której należą, czyli od położenia po dodatniej lub ujemnej stronie hiperpłaszczyzny H . Każdy wektor \vec{x}_i jest rozmiaru p . Działanie algorytmu SVM polega na znalezieniu hiperpłaszczyzny o maksymalnym marginesie geometrycznym, która dzieli grupy punktów \vec{x}_i , dla których $y_i = -1$ od grupy punktów \vec{x}_i , dla których $y_i = 1$. Hiperpłaszczyzna H , n -wymiarowa określona jest wzorem:

$$(H)y(\vec{x}) = 0 \quad (2)$$

gdzie $(y(\vec{x}) = w^t + b$, w - wektor wagowy, b - wyraz wolny. Po przedstawionym wcześniej założeniu, że $y(x) = -1$ lub $y(x) = 1$, możemy przedstawić wzór na odległość punktu x od danej hiperpłaszczyzny H . Przedstawia się on następująco:

$$d(x, H) = \frac{|y(x)|}{||w||} \quad (3)$$

margines geometryczny natomiast, będzie dany wzorem:

$$\gamma = \frac{1}{||w||} \quad (4)$$

Klasyfikator maksymalnego marginesu znajduje hiperpłaszczyznę, która maksymalizuje wartość marginesu geometrycznego czyli taką, dla której wartość $||w||$ jest minimalna. Maksymalizacja marginesu może zostać zapisana w postaci poniżej przedstawionego problemu optymalizacyjnego:

$$\text{minimalizacja } ||\vec{w}||^2$$

przy warunkach:

$$y_i(\vec{w} * \vec{x}_i - b) \geq 1$$

gdzie $i \in \{1..N\}$ oraz istnieje założenie o liniowej separowalności wektorów.

Można następnie dla powyższego problemu optymalizacyjnego zapisać lagranżjan postaci:

$$L(w, b, \vec{\alpha}) = \frac{1}{2} \cdot ||\vec{w}||^2 - \sum_{i=1}^N \alpha_i (y_i(\vec{w} * \vec{x}_i + b) - 1) \quad (5)$$

gdzie $\vec{\alpha}$ to wektor mnożników Lagrange'a, o rozmiarze N .

Korzystając z lagranżjanu, przedstawiony problem optymalizacyjny może zostać zamieniony na formę dualną, w której funkcja celu jest wyłącznie zależna od mnożników Lagrange'a:

minimalizacja funkcji :

$$\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j (\vec{x}_i \cdot \vec{x}_j) \alpha_i \alpha_j - \sum_{i=1}^N \alpha_i$$

gdzie N to liczba punktów treningowych

przy warunkach:

$$\alpha_i \geq 0, \forall i$$

$$\sum_{i=1}^N y_i \alpha_i = 0$$

Kiedy współczynniki Lagrange'a zostaną wyznaczone, wektor \vec{w} oraz wyraz wolny b może zostać obliczony przy ich pomocy:

$$\vec{w} = \sum_{i=1}^N y_i \alpha_i \vec{x}_i, \quad (6)$$

$$b = \vec{w} \cdot \vec{x}_i - y_i \quad (7)$$

dla pewnych $\alpha_i > 0$

Oczywiście, nie wszystkie zbiory danych mogą być liniowo separowalne. Może się okazać, że nie istnieje hiperpłaszczyzna, które rozdziela wszystkie punkty należące do jednej klasy od punktów należących do drugiej klasy. W takim właśnie przypadku można skorzystać z pewnej modyfikacji oryginalnego problemu optymalizacyjnego. Prezentuje się ona następująco:

$$\text{minimalizacja } \|\vec{w}\|^2 + C \sum_{i=1}^N \xi_i$$

przy warunkach:

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 - \xi_i, \forall i$$

gdzie ξ_i to tak zwana zmienna luzu, która pozwala na błąd marginesu.

Klasyfikator w tym wypadku bierze pod uwagę możliwe wahania wartości danych. Wektory wspierające w tym wypadku to nie tylko punkty znajdujące się najbliżej hiperpłaszczyzny, ale również dalsze.

Forma dualna powyższego problemu przedstawia się następująco:

minimalizacja funkcji :

$$\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j K(\vec{x}_i \cdot \vec{x}_j) \alpha_i \alpha_j - \sum_{i=1}^N \alpha_i$$

gdzie N to liczba punktów treningowych

przy warunkach:

$$0 \leq \alpha_i \leq C, \forall i$$

$$\sum_{i=1}^N y_i \alpha_i = 0$$

W tym wypadku współczynniki α będą również ograniczone z góry.

Powyższy problem optymalizacyjny może zostać rozwiązany przy pomocy algorytmu *sekwencyjnej minimalnej optymalizacji* który został szczegółowo opisany w kolejnym podrozdziale.

2.1.2 Sekwencyjna minimalna optymalizacja (SMO)

SMO jest jednym z algorytmów pozwalających rozwiązać główny problem w nauczaniu SVM, czyli problem programowania kwadratowego (oznaczany jako QP). SMO pozwala na uniknięcie problemów związanych z optyimizacją numeryczną poprzez rozkład całościowego problemu na podproblemy.

W każdym kroku SMO rozwiązuje najmniejszy możliwy problem optymalizacji, czyli przypadek dwóch współczynników Lagrange'a, które są ograniczone liniowo. Oba współczynniki są jednocześnie optymalizowane, po czym aktualizowana jest cała maszyna wektorów nośnych i następnie algorytm wybiera dwa kolejne współczynniki do optymalizacji.

SMO jest proste w implementacji oraz nie wymaga dużych zasobów pamięci do przechowywania zmiennych.

SMO składa się z dwóch części - analitycznego poszukiwania pary współczynników Lagrange'a oraz heurystyki służącej wybieraniu kolejnych współczynników do optymalizacji.

2.1.3 Poszukiwanie współczynników Lagrange'a

Pierwszą czynnością, którą realizuje SMO, jest obliczenie ograniczeń współczynników. Po ich obliczeniu możliwa jest poszukiwanie minimum w ograniczonej przestrzeni.

Liniowy warunek równości powoduje, że współczynniki Lagrange'a leżą na prostej, przez co minimum optymalizowanej funkcji również musi na niej leżeć. Aby SMO spełniało ten warunek w każdym kroku, konieczne jest użycie dwóch współczynników.

Algorytm oblicza drugi współczynnik Lagrange'a α_2 , po czym oblicza końce odcinka leżącego na prostej spełniającej warunek równości. Jeżeli koniec y_1 nie jest równy końcowi y_2 , wtedy stosuje się następujące ograniczenia dla α_2 :

$$L = \max(0, \alpha_2 - \alpha_1), \quad H = (C, C + \alpha_2 - \alpha_1) \quad (8)$$

Jeżeli koniec y_1 jest równy końcowi y_2 , wtedy ograniczenia α_2 zmieniają się następująco:

$$L = \max(0, \alpha_2 + \alpha_1 - C), \quad H = (C, \alpha_2 + \alpha_1) \quad (9)$$

Drugą pochodną funkcji docelowej wzdłuż odcinka można wyrazić jako:

$$\eta = K(\vec{x}_1, \vec{x}_1) + K(\vec{x}_2, \vec{x}_2) - 2K(\vec{x}_1, \vec{x}_2) \quad (10)$$

W normalnych warunkach funkcja docelowa będzie określona dodatnio, minimum wystąpi wzdłuż prostej określonej liniowym warunkiem równości i η będzie większe od zera. W takim przypadku SMO oblicza nieograniczone minimum wzdłuż całej prostej zgodnie z równaniem:

$$\alpha_2^{nowy} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta}, \quad (11)$$

w którym $E_i = u_i - y_i$ oznacza błąd i -tej próbki ze zbioru uczącego. W kolejnym kroku znajduwane jest ograniczenie minimum poprzez porównanie z obliczonymi wcześniej końcami odcinka.

$$\alpha_2^{nowy, ograniczony} = \begin{cases} H & \text{jeżeli } \alpha_2^{nowy} \geq H \\ \alpha_2^{nowy} & \text{jeżeli } L < \alpha_2^{nowy} < H \\ L & \text{jeżeli } \alpha_2^{nowy} \leq L \end{cases} \quad (12)$$

Używając oznaczenia $s = y_1 y_2$ wartość α_1 można obliczyć wykorzystując nowy, ograniczony współczynnik α_2 :

$$\alpha_1^{nowy} = \alpha_1 + s(\alpha_2 - \alpha_2^{nowy, ograniczony}). \quad (13)$$

W niezwykle przypadkach η nie będzie dodatnie. Ujemne η wystąpi, gdy jądro K nie spełni warunku Mercera, przez co funkcja docelowa może stać się nieoznaczona. Zerowe η może wystąpić nawet z poprawnym jądrem, gdy więcej niż jedna próbka ucząca ma taki sami wektor wejściowy x . SMO zadziała nawet gdy η nie jest dodatnie, w takim przypadku funkcja docelowa Ψ powinna zostać policzona na każdym z końców odcinka:

$$f_1 = y_1(E_1 + b) - \alpha_1 \quad (14)$$

$$f_2 = y_2(E_2 + b) - s\alpha_1 \quad (15)$$

$$K(\vec{x}_1, \vec{x}_2) - \alpha_2 K(\vec{x}_2, \vec{x}_2), \quad (16)$$

$$L_1 = \alpha_1 + s(\alpha_2 - L), \quad (17)$$

$$H_1 = \alpha_1 + s(\alpha_2 - H), \quad (18)$$

$$\Psi_L = L_1 f_1 + L f_2 + \frac{1}{2} L_1^2 K(\vec{x}_1, \vec{x}_1) + \frac{1}{2} L^2 K(\vec{x}_2, \vec{x}_2) + s L L_1 K(\vec{x}_1, \vec{x}_2), \quad (19)$$

$$\Psi_H = H_1 f_1 + H f_2 + \frac{1}{2} H_1^2 K(\vec{x}_1, \vec{x}_1) + \frac{1}{2} H^2 K(\vec{x}_2, \vec{x}_2) + s H H_1 K(\vec{x}_1, \vec{x}_2). \quad (20)$$

SMO przesunie współczynniki Lagrange'a na ten koniec odcinka, w którym wartość funkcji docelowej jest najmniejsza. Jeżeli funkcja docelowa ma takie same wartości na obu końcach odcinka (uwzględniając mały błąd ϵ z powodu błędów zaokrągleń) i jądro spełnia warunki Mercera, to optymalizacja nie może się zakończyć. Ten przypadek opisano poniżej.

2.1.4 Heurystyka wyboru współczynników do optymalizacji

Wartość funkcji docelowej zmniejszy się w każdym kroku działania algorytmu SMO, jeżeli zostanie zoptymalizowana para współczynników Lagrange'a i co najmniej jeden z nich przed optymalizacją łamał warunki KKT. To gwarantuje zbieżność algorytmu, której uzyskanie można przyspieszyć stosując heurystykę do wyboru pary współczynników do jednoczesnej optymalizacji.

Stosowane do tego są dwie różne heurystyki wyboru współczynników. Wybór pierwszego współczynnika gwarantuje zewnętrzną pętlę algorytmu - iteruje po całym zbiorze uczącym sprawdzając które z przypadków naruszają warunki KKT. Jeżeli dana próbka nie spełnia tych warunków, może być zoptymalizowana. Po przejściu przez cały zbiór uczący, zewnętrzna pętla ponownie iteruje po wszystkich próbkach, dla których współczynniki Lagrange'a są różne od 0 i różne od C (przypadki niegraniczne). Ponownie dla każdego z takich przypadków są sprawdzane warunki KKT i optymalizacji mogą podlegać te, które ich nie spełniają. Zewnętrzna pętla powtarza przejścia po wszystkich przypadkach niegranicznych dopóki wszystkie przypadki naruszające warunki KKT nie będą leżały w granicach błędu ϵ . Następnie zewnętrzna pętla cofa się i ponownie iteruje po całym zbiorze uczącym. Pętla ta przełącza się między pojedynczymi przejściami po całym zbiorze uczącym i wielokrotnymi przejściami po podzbiorze niegranicznym do momentu, w którym cały zbiór spełnia warunki KKT w granicach ϵ . W tym momencie algorytm kończy działanie.

Heurystyka pierwszego wyboru skupia się na przypadkach, dla których prawdopodobieństwo złamania warunków KKT jest największe, czyli dla zbioru niegranicznego. Przypadki, które znajdują się na granicy, najprawdopodobniej na niej pozostaną, a te, które nie są na granicy, mogą się przesunąć. SMO optymalizuje więc najpierw podzbiór danych, a następnie przeszukuje cały zbiór w poszukiwaniu punktów, które mogły zacząć łamać warunki KKT w wyniki wcześniejszych zmian.

Typowa wartość błędu ϵ , dla którego warunki KKT są spełnione, to 10^{-3} . Zmniejszenie wartości dopuszczalnego błędu może spowodować wydłużenie czasu potrzebnego na optymalizację, jednak jest to typowe dla wszystkich algorytmów stosowanych do nauki SVM.

Po wyborze pierwszego współczynnika Lagrange'a, wybierany jest drugi współczynnik w taki sposób, aby zmaksymalizować wielkość kroku podczas optymalizacji pary. Obliczanie funkcji jądra K jest czasochłonne, więc SMO przybliża wielkość kroku o wartość bezwzględną licznika w równaniu 11: $|E_1 - E_2|$. SMO zapisuje wartość błędu E dla każdego przypadku niegranicznego w zbiorze uczącym i wybiera błąd tak, aby zmaksymalizować wielkość kroku. Jeżeli E_1 jest dodatnie, SMO wybierze przypadek z najmniejszą wartością błędu E_2 . Jeżeli E_1 jest ujemne, SMO wybierze przypadek z największym błędem E_2 .

W wyjątkowych sytuacjach SMO nie może znaleźć odpowiedniego współczynnika przy wykorzystaniu opisanej heurystyki drugiego wyboru, np. w przypadku, gdy dwie próbki mają takie same wartości cech. W takim przypadku SMO stosuje hierarchię heurystyki drugiego wyboru aż znajdzie parę współczynników Lagrange'a, które mogą być zoptymalizowane. Warunkiem skutecznej optymalizacji jest wykonanie niezerowego kroku. Hierarchię w tym przypadku można przestawić następująco - jeżeli optymalizacja nie jest skuteczna, SMO iteruje po przypadkach niegranicznych, poszukując przypadku, który pozwoli na sukces. Jeżeli żaden z niegranicznych przypadków na to nie pozwala, SMO zaczyna iterować po całym zbiorze uczącym aż zostanie znaleziony przypadek pozwalający na skuteczną optymalizację. Iterowanie zarówno w przypadku podzbioru przypadków niegranicznych jak i całego zbioru rozpoczyna się od losowo wybranego elementu zbioru. Pozwala to uniknąć obciążenia SMO przez elementy znajdujące się na początku zbioru. W najgorszym przypadku żaden z pozostałych przypadków nie będzie się nadawał do optymalizacji. W takiej sytuacji dana próbka jest pomijana i SMO przechodzi do kolejnej wybranej próbki.

2.1.5 Obliczanie progu

Próg b jest obliczany w każdym kroku, dzięki czemu warunki KKT są spełnione dla obu optymalizowanych próbek. Przedstawiony na równaniu próg b_1 jest prawidłowy, gdy nowy współczynnik α_1 nie jest na granicy, ponieważ zmusza SVM, aby wyjściem było y_1 dla wejścia x_1 :

$$b_1 = E_1 + y_1(\alpha_1^{nowy} - \alpha_1)K(\vec{x}_1, \vec{x}_1) + y_2(\alpha_2^{nowy} - \alpha_2)K(\vec{x}_1, \vec{x}_2) + b \quad (21)$$

Opisany poniżej próg b_2 jest prawidłowy, gdy nowy współczynnik α_2 nie leży na granicy, ponieważ zmusza SVM, aby wyjściem było y_2 dla wejścia x_2 :

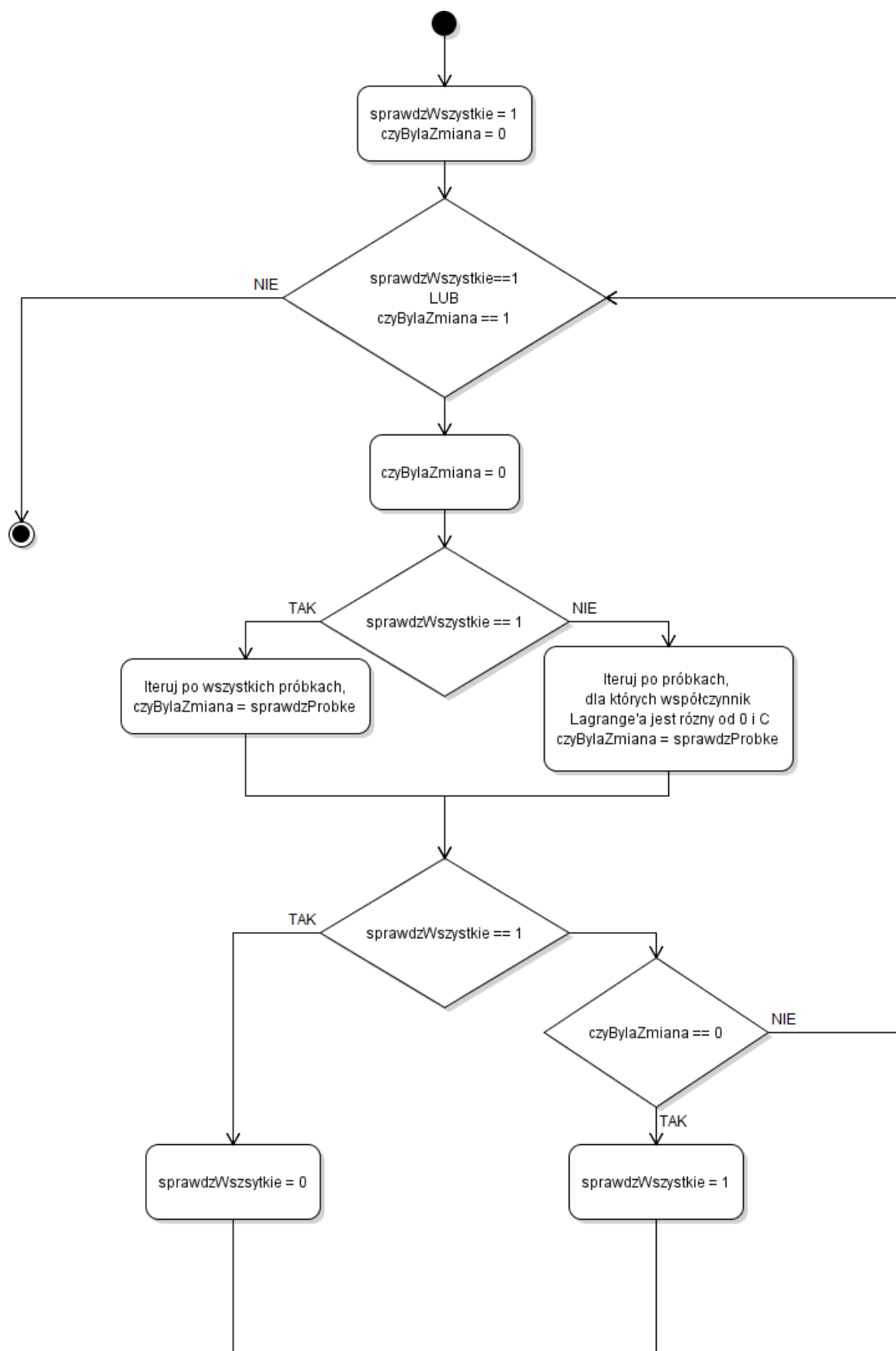
$$b_2 = E_2 + y_1(\alpha_1^{nowy} - \alpha_1)K(\vec{x}_1, \vec{x}_2) + y_2(\alpha_2^{nowy} - \alpha_2)K(\vec{x}_2, \vec{x}_2) + b \quad (22)$$

Jeżeli oba progi b_1 i b_2 są prawidłowe, to są sobie równe. Gdy oba współczynniki Lagrange’a leżą na granicy i gdy L nie jest równe H , wtedy odległością między b_1 i b_2 są wszystkie progi, które są zgodne z warunkami KKT. SMO wybiera wtedy próg leżący pośrodku między b_1 oraz b_2 .

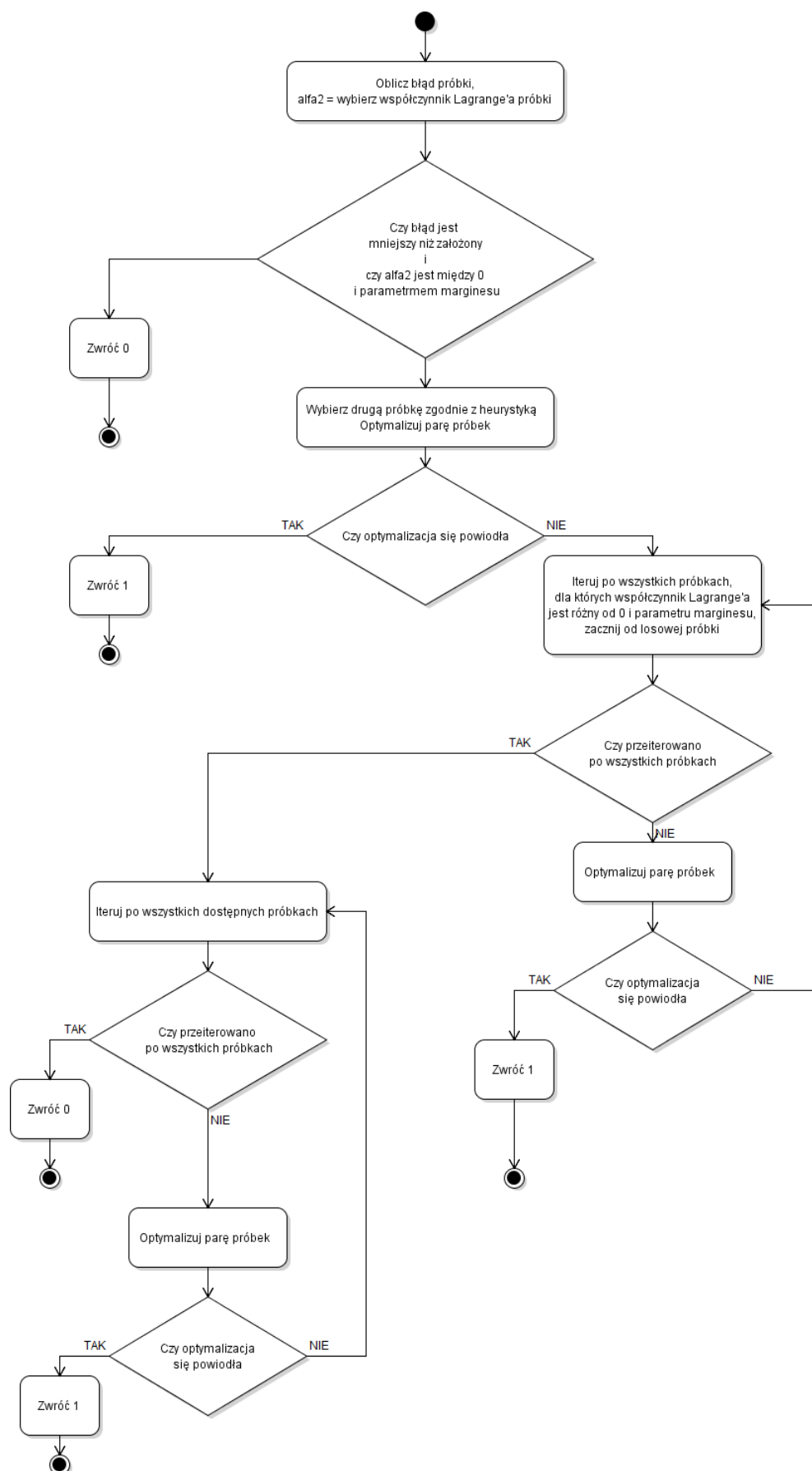
2.2 Opis sposobu implementacji algorytmu

2.2.1 Schematy blokowe działania algorytmu

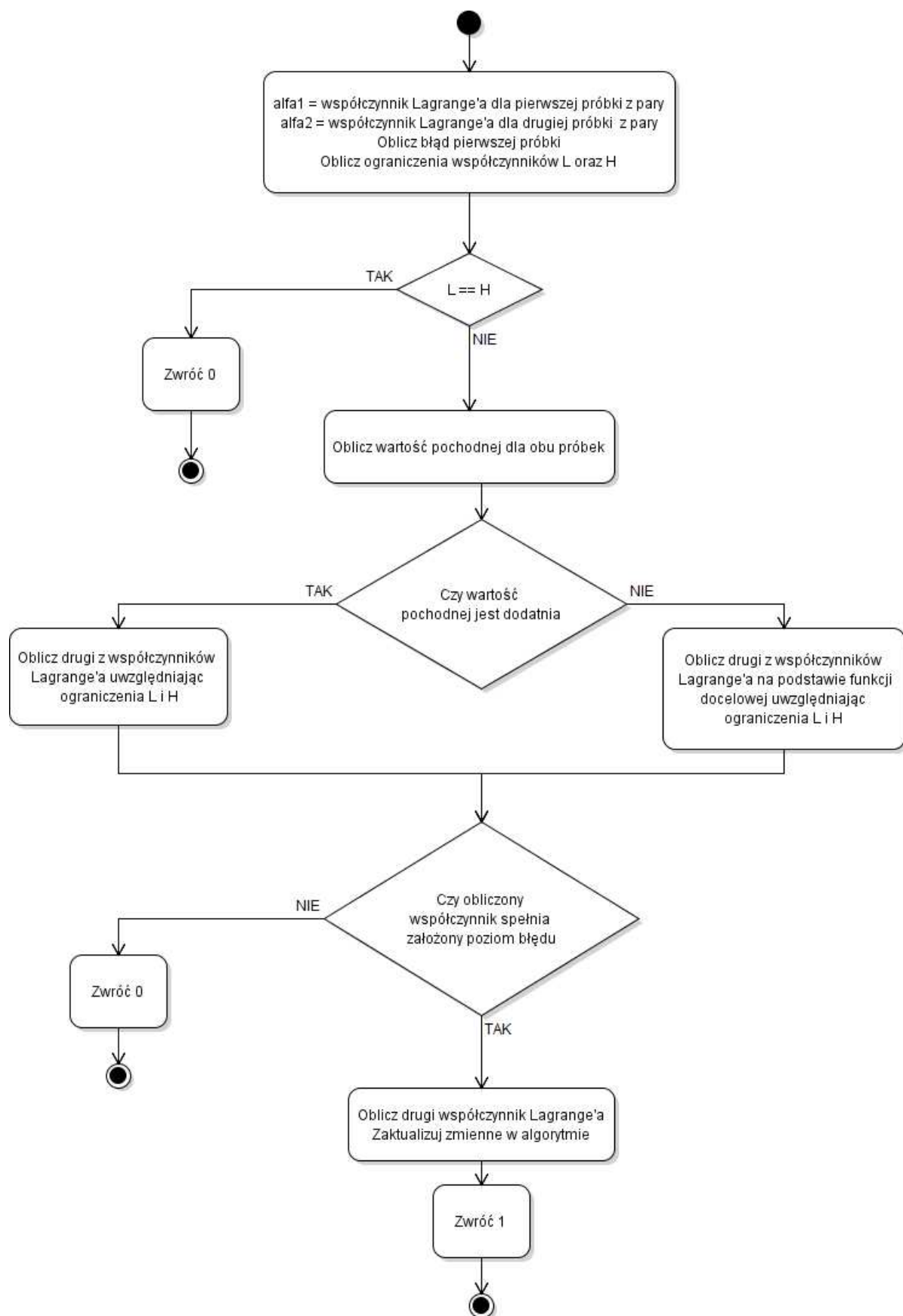
Sposób działania algorytmu przedstawiono za pomocą trzech schematów blokowych, które znajdują się poniżej:



Rysunek 4: Schemat blokowy głównej pętli programu
(opracowanie własne)



Rysunek 5: Schemat blokowy badania pojedynczej obserwacji
(opracowanie własne)



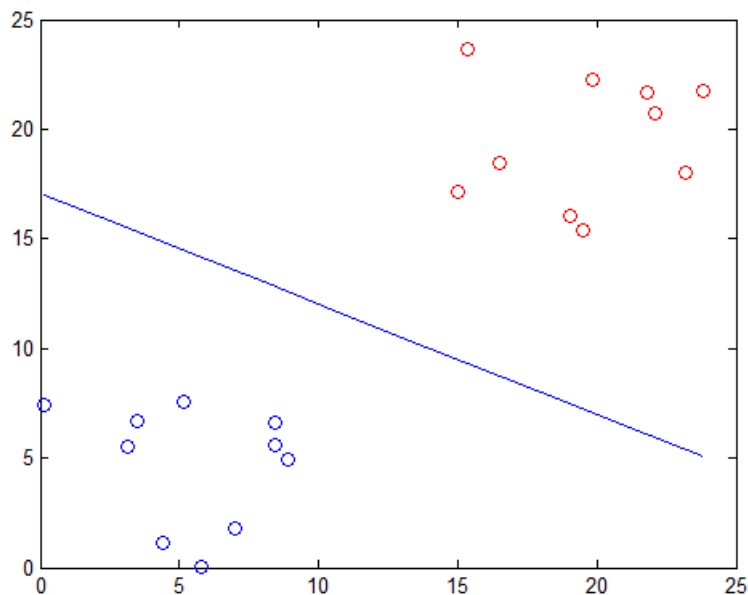
Rysunek 6: Schemat blokowy optymalizacji pary próbek
(opracowanie własne)

2.3 Wizualizacja działania algorytmu

W celu zobrazowania sposobu działania prototypu algorytmu opracowanego w środowisku MATLAB, dokonano wizualizacji wyznaczonej hiperpłaszczyzny. Dane należące do dwóch różnych kategorii zostały wylosowane przy pomocy funkcji *rand()*.

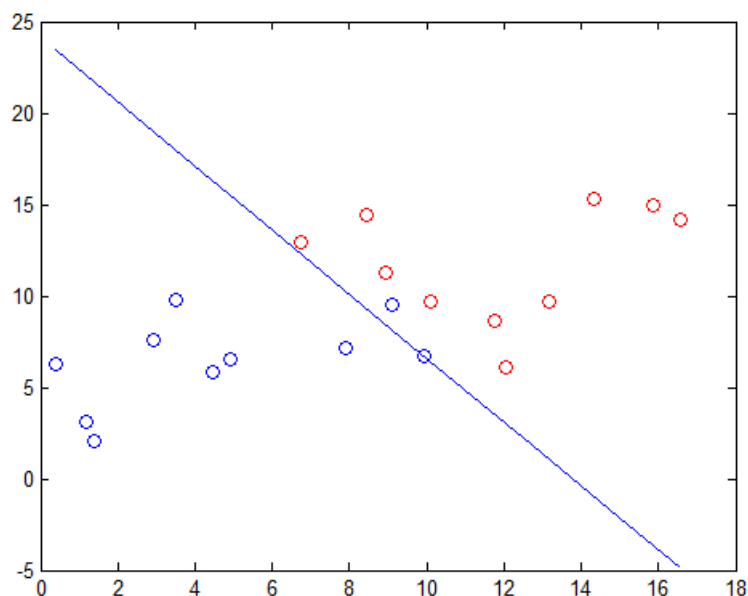
2.3.1 Wizualizacja dla dwóch cech

Na rysunku nr 4 przedstawiono dwa zbiory punktów należących do dwóch różnych klas, które zostały rozdzielone przy pomocy obliczonej hiperpłaszczyzny. W tym wypadku użyto dwóch cech, a przedstawione dane były liniowo separowalne.



Rysunek 7: Rezultat działania algorytmu dla dwóch wymiarów cech i liniowo separowalnych danych
(opracowanie własne)

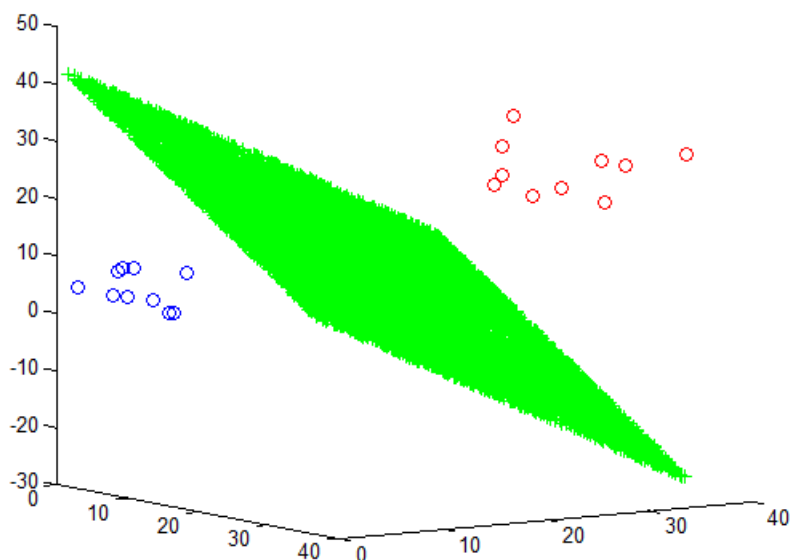
Na rysunku nr 5 przedstawiono dwa zbiory punktów należących do dwóch różnych klas, które zostały rozdzielone przy pomocy obliczonej hiperpłaszczyzny. W tym wypadku użyto dwóch cech, a przedstawione dane nie były liniowo separowalne.



Rysunek 8: Rezultat działania algorytmu dla dwóch wymiarów cech i danych nieseparowalnych liniowo
(*opracowanie własne*)

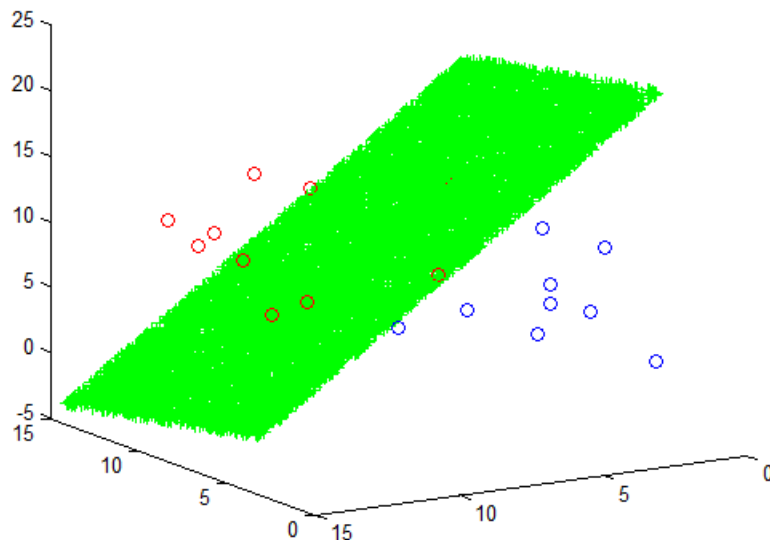
2.3.2 Wizualizacja dla trzech cech

Na rysunku nr 6 przedstawiono dwa zbiory punktów należących do dwóch różnych klas, które zostały rozdzielone przy pomocy obliczonej hiperpłaszczyzny. W tym wypadku użyto trzech cech, a przedstawione dane były liniowo separowalne.



Rysunek 9: Rezultat działania algorytmu dla trzech wymiarów cech i liniowo separowalnych danych
(*opracowanie własne*)

Na rysunku nr 7 przedstawiono dwa zbiory punktów należących do dwóch różnych klas, które zostały rozdzielone przy pomocy obliczonej hiperpłaszczyzny. W tym wypadku użyto trzech cech, a przedstawione dane nie były liniowo separowalne.



Rysunek 10: Rezultat działania algorytmu dla trzech wymiarów cech i danych nieseparowalnych liniowo
(opracowanie własne)

2.4 Opis informatyczny procedur

funkcja `optimizePair`:

```
void optimizePair(MatrixXd labelSet, MatrixXd gramMatrix, int pairFirst, int pairSecond,
MatrixXd sampleError, MatrixXd &lagrangeMultipliers, double const marginParameter,
double &bias, bool &flagOptimizeSuccess)
```

Funkcja pozwala na optymalizację pary współczynników Lagrange'a.

Funkcja przyjmuje:

labelSet - macierz typu MatrixXd z oznaczeniem cech
gramMatrix - macierz typu MatrixXd reprezentującą macierz Gramma
pairFirst - liczba typu int reprezentująca analizowany indeks
pairSecond - liczba typu int reprezentująca analizowany indeks
sampleError - macierz typu MatrixXd zawierająca wartości błędów dla poszczególnych próbek
marginParameter - parametr marginesu

Funkcja przyjmuje w postaci referencji:

lagrangeMultipliers - macierz typu MatrixXd ze współczynnikami Lagrange'a
bias - liczba typu double oznaczająca przesunięcie granicy decyzyjności
flagOptimizeSuccess - wartość logiczna typu bool reprezentująca powodzenie lub niepowodzenie optymalizacji

funkcja `examineSample`:

```
void examineSample(int pairSecond, MatrixXd &sampleError, MatrixXd &lagrangeMultipliers,
MatrixXd gramMatrix, MatrixXd labelSet, double &bias, double const marginParameter,
double const tolerance, bool &flagExamineSuccess)
```

Funkcja pozwala na analizę danej próbki.

Funkcja przyjmuje:

pairSecond - liczba typu int reprezentująca analizowany indeks

gramMatrix - macierz typu MatrixXd reprezentującą macierz Gramma
labelSet - macierz typu MatrixXd z oznaczeniem cech
marginParameter - parametr marginesu
tolerance - liczba typu double reprezentująca tolerancję

Funkcja przyjmuje w postaci referencji:

sampleError macierz typu MatrixXd zawierająca wartości błędów dla poszczególnych próbek
lagrangeMultipliers - macierz typu MatrixXd ze współczynnikami Lagrange'a
bias - liczba typu double oznaczająca przesunięcie granicy decyzyjności
flagExamineSuccess - wartość logiczna typu bool reprezentująca powodzenie lub niepowodzenie analizy

funkcja smosvm:

void smosvm(MatrixXd trainSet, MatrixXd labelSet, double const marginParameter, MatrixXd &w, double &bias, float &trainingTime)

Funkcja pozwala na wyznaczenie optymalnej hiperpłaszczyzny rozdzielającej dwie klasy.

Funkcja przyjmuje:

trainSet - macierz typu MatrixXd zawierającą cechy do trenowania algorytmu
labelSet - macierz typu MatrixXd zawierającą oznaczenia cech
marginParameter - parametr marginesu

Funkcja przyjmuje w postaci referencji:

w - macierz typu MatrixXd zawierającą współczynniki wyznaczonej hiperpłaszczyzny
bias - liczba typu double oznaczająca przesunięcie granicy decyzyjności
trainingTime - liczbę typu float reprezentującą czas trenowania

funkcja loadData:

bool loadData(std::string filename, MatrixXd &dataSet, MatrixXd &labelSet)

Funkcja pozwala na wczytanie danych treningowych oraz danych testowych.

Funkcja przyjmuje:

filename - string z nazwą pliku

Funkcja przyjmuje w postaci referencji:

dataSet - macierz typu MatrixXd z cechami które zostały wczytane
labelSet - macierz typu MatrixXd z oznaczeniem cech

Funkcja zwraca wartość logiczną typu bool reprezentującą powodzenie lub niepowodzenie procesu wczytywania pliku:

funkcja svmclassify:

MatrixXd svmclassify(MatrixXd w, double bias, MatrixXd &dataSet, float &classificationTime)

Funkcja pozwala na klasyfikację zbioru testowego przy użyciu wcześniej obliczonych współczynników

klasyfikatora.

Funkcja przyjmuje:

w - macierz typu MatrixXd zawierająca współczynniki wyznaczonej hiperpłaszczyzny
bias - liczba typu double oznaczająca przesunięcie granicy decyzyjności

Funkcja przyjmuje w postaci referencji:

dataSet - macierz typu MatrixXd zawierające dane, które mają zostać sklasyfikowane
classificationTime - liczba typu float reprezentująca czas klasyfikacji

Funkcja zwraca macierz typu MatrixXd zawierającą wyniki klasyfikacji uzyskanej przez algorytm:

funkcja checkAccuracy:

float checkAccuracy(MatrixXd const &resultSet, MatrixXd const &trueResultSet)

Funkcja pozwala na sprawdzenie skuteczności klasyfikacji algorytmu.

Funkcja przyjmuje:

resultSet - macierz typu MatrixXd z wynikami klasyfikacji uzyskanymi przez algorytm

Funkcja przyjmuje w postaci referencji:

trueResultSet - macierz typu MatrixXd z poprawnymi wynikami klasyfikacji

Funkcja zwraca liczbę typu float reprezentującą skuteczność klasyfikacji algorytmu:

funkcja saveResult:

void saveResult(float const &trainingTime, float const &classificationTime, float const &accuracy)

Funkcja pozwala na zapisywanie rezultatów treningu oraz klasyfikacji do pliku tekstowego "linear SVM results.txt".

Funkcja przyjmuje w postaci referencji:

trainingTime - czas, którego algorytm potrzebował na nauczanie klasyfikatora

classificationTime - czas, którego algorytm potrzebował do klasyfikacji

accuracy - skuteczność klasyfikacji algorytmu w %

funkcja getBiggerNumber:

double getBiggerNumber(double firstNumber, double secondNumber)

Funkcja pozwala na znalezienie liczby większej spośród dwóch liczb.

Funkcja przyjmuje:

firstNumber - liczbę typu double

secondNumber - liczbę typu double

Funkcja zwraca większą z dwóch analizowanych liczb:

funkcja `getSmallerNumber`:

`double getSmallerNumber(double firstNumber, double secondNumber)`

Funkcja pozwala na znalezienie liczby mniejszej spośród dwóch liczb.

Funkcja przyjmuje:

`firstNumber` - liczbę typu `double`
`secondNumber` - liczbę typu `double`

Funkcja zwraca mniejszą z dwóch analizowanych liczb:

`getNonboundSubset`:

`MatrixXd getNonboundSubset(MatrixXd lagrangeMultipliers, double marginParameter)`

Funkcja przyjmuje:

`lagrangeMultipliers` - macierz typu `MatrixXd` ze współczynnikami Lagrange'a
`marginParameter` - liczba typu `double` z parametrem marginesu

Funkcja zwraca macierz typu `MatrixXd` z indeksami elementów macierzy `lagrangeMultipliers`, których wartość jest różna od 0 lub różna od wartości zmiennej `marginParameter`

funkcja `randomizeIndexOrder`:

`void randomizeIndexOrder(MatrixXd dataSet, MatrixXd &randomizedIndices)`

Funkcja pozwala na losowe ustawienie indeksów.

Funkcja przyjmuje:

`dataSet` - macierz typu `MatrixXd` z indeksami, które mają zostać zrandomizowane

Funkcja przyjmuje w postaci referencji:

`randomizedIndices` - macierz typu `MatrixXd` z indeksami, które zostały zrandomizowane

funkcja `random_data_generator`:

`void random_data_generator(MatrixXd &trainSet, MatrixXd &labelSet)`

Funkcja pozwala na wylosowanie danych do testowania działania algorytmu.

Funkcja przyjmuje w postaci referencji:

`trainSet` - macierz typu `MatrixXd` z cechami do trenowania
`labelSet` - macierz typu `MatrixXd` z oznaczeniem klas

Literatura

- [1] Platt John *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines* Microsoft Research, Technical Report MSR-TR-98-14, 1998
- [2] de Chazal Philip *A Patient-Adapting Heartbeat Classifier Using ECG Morphology and Heartbeat Interval Features*, IEEE Transactions On Biomedical Engineering, Vol. 53, No. 12, 2006
- [3] Stefanowski Jerzy *Metoda wektorów nośnych - slajdy dodatkowe do wykładu*
<http://www.cs.put.poznan.pl/jstefanowski/ml/SVM.pdf> Institute of Computing Sciences, Poznań University of Technology
- [4] SVM Tutorial *Understanding the math*,
<http://www.svm-tutorial.com/2014/11/svm-understanding-math-part-2/>

3 Klasyfikator Bayesa

3.1 Opis algorytmu

Działanie klasyfikatora Bayesa polega na przyporządkowaniu nowego przypadku do wcześniej zdefiniowanej klasy. Podstawowym założeniem tego klasyfikatora jest niezależność każdej cechy występującej w klasie od reszty cech. Innymi słowy użyty klasyfikator bayesowski jest klasyfikatorem probabilistycznym z założeniem niezależności. Oznacza to, że obecność lub brak danej cechy nie łączy się z występowaniem jakiegokolwiek innej. Omawiany klasyfikator jest statystycznym klasyfikatorem opartym na twierdzeniu Bayesa.

Zakłada się, że dany obiekt X jest reprezentowany przez zbiór cech, którego wartości należą do zbioru $X = (x_1, x_2, \dots, x_n)$. Reguła Bayesa mówi, że obiekt X należy do klasy C_j , dla której wartość prawdopodobieństwa $P(C_j|X)$ jest największa. $P(C|X)$ to prawdopodobieństwo, że obiekt X należy do klasy C . Aby oszacować prawdopodobieństwa a-posteriori $P(C|X)$ należy skorzystać z twierdzenia Bayesa, które ma postać:

$$P(C|X) = P(X|C)P(C)/P(X) \quad (23)$$

gdzie:

$P(C)$ - prawdopodobieństwo a-priori wystąpienia klasy C (czyli prawdopodobieństwo, że dowolny przykład należy do klasy C),

$P(X|C)$ - prawdopodobieństwo a-posteriori, że obiekt X należy do klasy C ,

$P(X)$ - prawdopodobieństwo wystąpienia obiektu X .

Prawdopodobieństwo $P(X)$ jest dla wszystkich klas takie same, więc tak naprawdę klasa C_i , dla której wartość $P(C|X)$ jest największa to klasa dla której prawdopodobieństwo $P(X|C_i)P(C_i)$ jest największe.

Wartość $P(C_i)$ zastępuje się względną częstością klasy C_i lub można założyć, że wszystkie klasy mają takie same prawdopodobieństwo.

Prawdopodobieństwo $P(X|C_i)$ to tak naprawdę iloczyn prawdopodobieństw kolejnych atrybutów:

$$P(X|C_i) = \prod_{j=1}^n P(x_j|C_i) \quad (24)$$

W przypadku wystąpienia ciągłego atrybutu prawdopodobieństwo $P(x_j|C_i)$ należy estymować za pomocą funkcji gęstości prawdopodobieństwa przy założeniu normalnego rozkładu wartości atrybutów:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (25)$$

gdzie:

μ - średnia danego atrybutu w klasie,

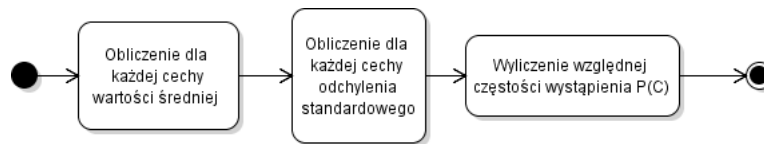
σ^2 - wariancja atrybutu [1].

Naiwny model Bayesa jest łatwy do zbudowania oraz świetnie sprawdza się w przypadku bardzo dużej ilości danych. Dodatkowo zaletą tej metody jest to, że wymaga nie dużej ilości danych trenujących, aby uzyskać potrzebne parametry klasyfikujące.

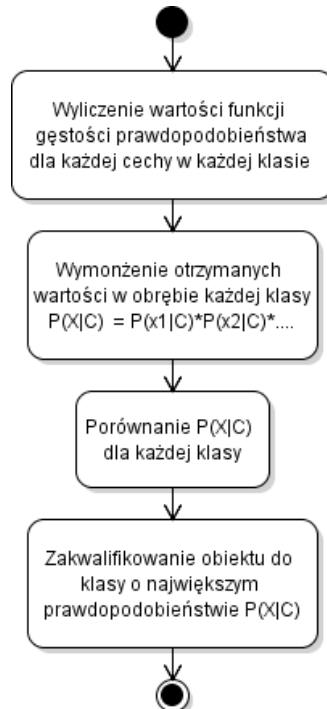
3.2 Implementacja

Patrząc pod kątem implementacji można wyróżnić dwa procesy: uczenia klasyfikatora i testowania. Proces uczenia będzie polegał na wyliczeniu odpowiednich parametrów. Są nimi częstość wystąpienia danej klasy $P(C_i)$, średnia wartość cechy μ w zależności od klasy, oraz odchylenie standardowe wartości cechy σ^2 . Rysunek 11 obrazuje proces uczenia klasyfikatora

Kolejny etap algorytmu to już analiza danych, które mają zostać sklasyfikowane. Polega ona wyliczeniu dla każdej cechy funkcji gęstości prawdopodobieństwa, ponieważ wszystkie cechy które będą wchodziły w skład obiektu będą cechami ciągłymi. Funkcja ta obliczana jest dla każdej klasy. Następnie w celu otrzymania prawdopodobieństwa $P(X|C_i)P(C_i)$ otrzymane wartości wymnaża się. Obiekt zostanie zakwalifikowany do klasy, w której obliczone prawdopodobieństwo $P(X|C_i)$ jest najwyższe. Rysunek 12 przedstawia algorytm przypisania klasy do zestawu cech.



Rysunek 11: Schemat algorytmu uczenia klasyfikatora.



Rysunek 12: Schemat algorytmu przypisania klasy do obiektu.

3.3 Wykorzystanie klasyfikatora do rozpoznawania uderzeń serca

W celu wstępnego wyznaczenia skuteczności algorytmu, przeprowadzono klasyfikację na uderzeniach serca pochodzących z sygnału 100.dat i 228.dat z bazy danych MIT - BIH. Rodzaje uderzeń znajdujące się w tych sygnałach postanowiono podzielić na 3 klasy:

- uderzenia normalne oznaczone literką *N*,
- uderzenia komorowe oznaczone literką *V*,
- inne uderzenia wyróżnione w bazie MIT - BIH.

Zdecydowano się na taki podział ponieważ dwie pierwsze grupy stanowią ponad 90% wszystkich uderzeń serca występujących w całej bazie MIT-BIH.

Cechy jakie zostały zdefiniowane dla każdego obiektu to, zaproponowane na podstawie artykułu [2]:

- Pre interval RR - odległość między aktualnym uderzeniem serca(analizowanym), a poprzednim uderzeniem serca,
- Post inreval RR - odległość między analizowanym uderzenie serca a kolejnym uderzeniem,
- Average RR - średnia długość interwału RR dla całego analizowanego sygnału,
- Ratio1 - stosunek pre RR interwał i post RR interval
- Ratio2 - stosunek per interwału do Avarage RR

Dane zostały podzielone na dwie grupy treningową i testową odpowiednio w stosunku 4:1. Ilość obiektów w obu grupach łącznie wynosiło 4322, w grupie testowej było 864 próbek, a w treningowej 3458. Wynik klasyfikacji z wykorzystaniem wszystkich pięciu wcześniej opisanych cech. prezentuje Tabela 1.

Tabela 1: Wynik klasyfikacji po zastosowaniu klasyfikatora Bayesa.

	Dobrze rozpoznane	Źle rozpoznane	Dobrze rozpoznane/całość klasy
Uderzenia N	776	8	0,99
Uderzenia V	71	1	986
Inne uderzenia	4	4	0,5
Suma	851	13	0,985

Postanowiono sprawdzić jaka będzie skuteczność klasyfikacji, jeżeli uwzględnionych zostanie mniej cech. Na przykład cechy Ratio1 i Ratio2 są powiązane z przednimi cechami, więc istnieje podejrzenie, że skuteczność klasyfikatora z pominięciem tych cech może być wyższa zgodnie z naiwnym założeniem twierdzenia Bayesa. Okazało się jednak, że najlepszą skuteczność jaką można było uzyskać z wykorzystaniem tych cech była w momencie nie uwzględniania ostatniej cechy Ratio2. Wyniki jakie otrzymano korzystający tylko pierwszych 4 cech (post i pre interval RR, average RR i Ratio1) znajdują się w Tabeli 2.

Tabela 2: Wynik klasyfikacji po zastosowaniu klasyfikatora Bayesa z wykorzystaniem 3 cech.

	Dobrze rozpoznane	Źle rozpoznane	Dobrze rozpoznane/całość klasy
Uderzenia N	785	0	1
Uderzenia V	72	0	1
Inne uderzenia	0	7	0
Suma	857	7	0,992

Można zauważyć, że w momencie skorzystania z mniejszej liczby cech, skuteczność całego klasyfikatora wzrosła, jednak skuteczność w obrębie klas jest już bardzo różnorodna. Wszystkie uderzenia normalne i komorowe występujące w grupie testowej zostały właściwie rozpoznane, jednak żadne z innych uderzeń nie zostało poprawnie sklasyfikowane. Powodem może być duża różnorodność wartości w obrębie tej klasy. Natomiast korzystając z wszystkich wyznaczonych cech, skuteczność poprawnego zaklasyfikowania do innych uderzeń jest znacznie większa i wynosi 50%. Jednak w obu przypadkach całkowita skuteczność rozpoznania utrzymuje się na wysokim poziomie.

3.4 Podsumowanie

O skuteczności algorytmu nie decyduje ilość danych treningowych tylko ich reprezentatywność. Klasyfikator Bayesa jest tak skonstruowany, że jedynie czym można manipulować, aby polepszyć wyniki to zbiór treningowy - ilość i rodzaj cech danego uderzenia, natomiast ilość danych w zbiorze treningowym nie jest już tak bardzo istotna. Klasyfikator jest bardzo prosty w implementacji, opiera się na odpowiednim wyliczeniu prawdopodobieństw. Klasyfikacja przebiega bardzo sprawnie. Używając nie dużych zbiorów treningowych można dostać zadowalające rezultaty. Nie występuje zależność im więcej cech zostanie uwzględnionych w klasyfikatorze tym będą lepsze rezultaty, najlepsze wyniki występowały w przypadku skorzystania z 4 cech. Może to być spowodowane, wzajemną korelacją kolejnych cech i założeniem klasyfikatora Bayesa mówiącym o wzajemnej niezależności cech.

Literatura

- [1] Wykład: Klasyfikacja, Naiwny klasyfikator Bayesa
<http://wazniak.mimuw.edu.pl>
- [2] K.M. Senapati: *Automatic Classification of Heartbeats Using ECG Morphology and Heartbeat Interval Features*, Electrical Engineering IIT, 2014

4 k-Nearest Neighbours

4.1 Założenia wstępne metody

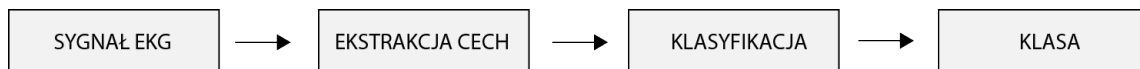
- Dany jest zbiór uczący wraz z wektorem zmiennych objaśniających (cech) oraz wartością zmiennej objaśnianej
- Dany jest zbiór testowy wraz z wektorem zmiennych objaśniających (cech) dla którego prognozuje się wartość zmiennej objaśnianej
- Parametr k jest dobierany przed rozpoczęciem działania algorytmu

4.2 Opis metody

Klasyfikator k-Najbliższych Sąsiadów to nieparametryczna metoda klasyfikacji oznaczana jako k-NN (*k-Nearest Neighbours*). Algorytm kNN nie wykonuje procedury uczenia się klasyfikatora - po prostu zapamiętuje wszystkie próbki ze zbioru treningowego. Ideą metody jest klasyfikacja na bieżąco, czyli wtedy gdy pojawia się potrzeba analizy nowego przypadku. Celem metody jest poszukiwanie klasy, która jest najbliższa analizowanemu przypadkowi. W tym celu próbka testowa jest porównywana z wszystkimi próbkami ze zbioru treningowego. Istnieją różne miary odległości i podobieństwa pomiędzy przypadkami i są one dobierane odpowiednio dla analizowanych danych. - w przypadku niniejszego projektu posłużono się *odległością euklidesową* [1]:

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

Parametr k jest dobierany przez rozpoczęciem działania algorytmu i opisuje ilość najbliższych sąsiadów jakie wyszuka algorytm. Spośród k -sąsiadów jako wynik klasyfikacji wybierana jest klasa najliczniej występująca wśród najbliższych sąsiadów. Odległość od danych sąsiadów na tym etapie nie ma już znaczenia, chyba że liczba sąsiadów z poszczególnych klas będzie identyczna. Proces klasyfikacji obrazuje Rysunek 13.



Rysunek 13: Proces klasyfikowania zespołu QRS do danej klasy

Na jakość klasyfikacji ma wpływ liczba K uwzględnionych sąsiadów oraz rozkład w przestrzeni wielowymiarowej poprzednich próbek. Wybór odpowiedniej liczby sąsiadów K jest podstawowym zadaniem podczas projektowania klasyfikatora. Najczęściej jest to wartość dająca najlepsze rezultaty w fazie treningowej. Zbyt duża liczba K będzie powodowała wysoką złożoność obliczeniową, a zbyt mała sprawi, że klasyfikator nie będzie odporny na szumy.

Zaletami tej metody jest prosty proces uczenia się i łatwość implementacji. Do wad można zaliczyć proces klasyfikacji, który musi przeanalizować cały zbiór uczący, żeby obliczyć wszystkie odległości. Prowadzi to, że klasyfikator ma dużą złożoność obliczeniową oraz dużą zajętość pamięci (przechowuje cały zbiór treningowy). Dodatkowo metoda ta jest wrażliwa na dane zaszumione lub błędne.

4.3 Klasy

Podczas implementacji prototypu w środowisku MATLAB w wektorach danych testowych i treningowych wyróżniono pięć klas, do których klasyfikowano zespoły QRS. Są to klasy rekomendowane przez *Association for the Advancement of Medical Instrumentation*:

- **N** (*normal*) - normalne, fizjologiczne uderzenie serca
- **S** (*supraventricular*) - ektopowe pobudzenie nadkomorowe
- **V** (*ventricular*) - ektopowe pobudzenie komorowe
- **F** (*fusion*) - pobudzenie mieszane (jednoczesne pobudzenie komorowe i nadkomorowe)
- **Q** (*unknown beat*) - nierozpoznane pobudzenie

4.4 Implementacja prototypu algorytmu w środowisku MATLAB

W projekcie zaimplementowano funkcję obliczającą k-NN, która po otrzymaniu pojedynczej próbki zawierającej cechy sygnału EKG klasyfikuje ją do odpowiedniej klasy. Algorytm przetestowano na próbkach, których klasa jest znana, po to aby móc zweryfikować poprawność jego działania. Implementacji prototypu dokonano w środowisku MATLAB.

Poniżej przedstawiono zaimplementowaną funkcję, która zrealizowana wykonana dla każdej próbki ze zbioru testowego:

```
function [ out ] = knnClassification( testData , trainData , trainLabels , k)

dist = pdist2( testData , trainData );
for i=1:k
    [ val(i) , idx ] = min( dist );
    out(i) = trainLabels( idx );
    dist( idx ) = max( dist );
end
end
```

Literatura

- [1] Chmielnicki W., *Efektywne metody selekcji cech i rozwiązywania problemu wieloklasowego w nadzorowanej klasyfikacji danych*, Instytut Podstawowych Problemów Techniki Polskiej Akademii Nauk, Kraków 2012

5 Extended Nearest Neighbours

5.1 Ograniczenia metody k-NN

Algorytm k-NN jest szeroko wykorzystywany w wielu różnych dziedzinach. Został nawet zakwalifikowany na konferencji *IEEE International Conference on Data Mining* do dziesięciu najlepszych algorytmów do zastosowań w obrębie *data-mining*. Jednak zastosowanie tego algorytmu wymaga odpowiednio zdefiniowanego parametru k oraz odpowiednio dobranej miary obliczania niepodobieństwa. Ponadto próbki, które nie pasują do żadnej klasy lub są reprezentowane przez klasę o niższej częstości występowania w zbiorze uczącym zostaną *zdominowane* przez klasy o najwyższej gęstości występowania. W takim przypadku liczba sąsiadów dla próbki z klasy drugiej może charakteryzować się większą ilością próbek z klasy pierwszej. Skłoniło to do poszukiwania rozwiązań, które umożliwiłyby jeszcze lepszą skuteczność omawianego algorytmu. Jednym z nich jest algorytm *emphextended nearest neighbour* [1].

5.2 Opis metody eNN

Algorytm ENN (*extended nearest neighbor*) wykorzystuje klasyfikator KNN do odnalezienia k-najbliższych sąsiadów próbki testowej w zbiorze treningowym. Nie wybiera jednak próbki na podstawie liczności sąsiadów z danej klasy, a na podstawie przypisanych do nich wag i wybiera tę klasę, dla której suma wag należących do niej sąsiadów jest największa. Dużą różnicą w stosunku do algorytmu kNN jest brak konieczności wyboru k - algorytm iteruje po wszystkich k od 1 do \sqrt{n} , gdzie n to liczba próbek w zbiorze testowym. Pod uwagę brane są tylko nieparzyste wartości k. Nieparzysta wartość k pozwala uniknąć sytuacji, w której doszłoby do konfliktu oraz zmniejsza złożoność obliczeniową algorytmu, która ze względu na iteracje po k jest duża wyższa w porównaniu z kNN. Dodatkowo algorytm kNN nie tworzy żadnego modelu obliczeniowego - każda próbka testowa jest porównywana z całym zbiorem uczącym, co prowadzi do dłuższego czasu działania w porównaniu z innymi klasyfikatorami. [2] [3]

Waga i-sąsiada w zbiorze k-najbliższych sąsiadów:

$$w(i) = \frac{1}{\log_2(1 + i)}$$

Średnia ważona dla poszczególnej klasy jest obliczana w następujący sposób:

$$WSc = \sum_{k=1}^{\sqrt{n}} \sum_{i=1}^k \begin{cases} w(i), A_i = c \\ 0, otherwise \end{cases}$$

Następnie pod uwagę brana jest ta klasa, która występuje najczęściej:

$$class = \operatorname{argmax}(cWSc)$$

5.3 Implementacja w środowisku MatLab

Danymi wejściowymi algorytmu są dane testowe oraz dane treningowe. Każdy zespół w ramach grupy posłużył się identycznymi danymi, aby wyniki można było porównać. Na początku znormalizowano dane, jednak i bez tej operacji algorytm działa poprawnie.

Algorytm jest przygotowany tak, aby działał dla różnych zbiorów danych (z różną ilością klas oraz cech). Funkcja *knnClassification* zwraca wektor k-najbliższych sąsiadów próbki testowej w kolejności od najbliższego do najdalszego sąsiada. Następnie dla każdego sąsiada obliczana jest jego waga. Wagi są sumowane w obrębie danej klasy. Wynikiem klasyfikacji jest ta klasa, której suma wag jest największa. Następnie, gdy zostanie obliczony rezultat dla każdej wartości k to spośród wektora klas wynikowych algorytm klasyfikuje próbkę testową do tej klasy, która występuje najczęściej.

```
%enn classification for every test sample
for testSample=1:length(testData)
    result = [];
    for k=1:2:sqrt(length(trainData))
        predictedClasses = knnClassification(testData(testSample,:), trainData, trainLabels, k);
        classes = unique(predictedClasses);
        weights = zeros(length(classes),1);
        for i=1:numel(predictedClasses)
            weight = 1/log2(i+1);
            index = find(classes == predictedClasses(i));
            weights(index) = weights(index) + weight;
        end
        [val, idx] = max(weights);
        result(size(result)+1) = classes(idx);
    end
    ennResult(testSample) = mode(result);
end
```

Algorytmy ENN i KNN najpierw zaimplementowano w środowisku MatLab, a później w języku obiektowym C++ w standardzie C++11 używając środowiska programistycznego Eclipse. Podczas ostatecznej implementacji udoskonalono algorytmy, zarówno w C++ jak i w MatLabie.

5.4 Podsumowanie

Podczas testowania działania algorytmu zmierzono czas realizacji klasyfikacji oraz skuteczność właściwego klasyfikowania próbek wyrażoną w procentach. Na zastosowanym zbiorze testowym nie wykryto różnicy w skuteczności dla algorytmu KNN oraz ENN - oba klasyfikowały na poziomie 98% oraz 99%. Oczekiwanym rezultatem była większa skuteczność algorytmu ENN, ponieważ uwzględnia on dalsze sąsiedztwo (a co za tym idzie więcej przypadków). Jednak algorytm ten często jest stosowany dla większej ilości klas - na innym, przetestowanym zbiorze danych z bazy sygnałów biomedycznych PhysioNet wyniki algorytmu rzeczywiście przyniosły oczekiwane rezultaty. Do minusów metody kNN oraz eNN należy duża złożoność obliczeniowa - algorytm nie buduje modelu, a zapamiętuje cały zbiór danych treningowych i z nim porównuje każdą próbkę testową.

Literatura

- [1] Jayalalitha S. Susan D. et al., *K-nearest Neighbour Method of Analysing the ECG Signal (To Find out the Different Disorders Related to Heart)*, Journal of Applied Sciences 2014, p.1682-1632,
- [2] Tang B., He H., *Enn: Extended nearest neighbor Method for Pattern Recognition*, IEEE Computational Intelligence Magazine 2015
- [3] Alkasassbeh M., Altarawneh G. A. et al., *On enhancing the performance of nearest neighbour classifiers using hassanat distance metric*, Canadian Journal of Pure and Applied Sciences 2015

6 Radial Basis Function Kernel SVM

6.1 Wstęp

Od wielu lat, tematyka automatycznej klasyfikacji elektrokardiogramu jest wnikliwie badana przez wiele zespołów. Wynika to z faktu, że pozwala uzyskać użyteczne informacje dotyczące rytmu oraz funkcjonowania serca. Analiza sygnału elektrokardiograficznego jest uważana za skuteczną metodę diagnostyczną w stanach chorobowych układu sercowonaczyniowego.

W ostatnich latach stosuje się coraz bardziej wyrafinowane metody klasyfikacji, do których zaliczają się m.in. która integrują w sobie elementy logiki rozmytej, sieci neuronowych, ukrytych modeli Markowa, transformacji falkowej oraz maszyny wektorów nośnych.

Jednym z kluczowych aspektów projektu systemu automatycznej klasyfikacji jest właśnie dobór metody klasyfikacji. Metoda SVM jest warta uwagi, ponieważ oparta jest na maksymalizacji obszaru pomiędzy hiperpłaszczyznami rozdzielającymi dane z dwóch grup. Znalazła ona zastosowanie w dziedzinach rozpoznawania obiektów 3-D, obrazowania medycznego i kompresji obrazów. [1]

6.2 Zarys proponowanego rozwiązania

Rozwiązanie stworzone w ramach projektu składa się z następujących elementów

- Pobranie sygnału EKG z bazy danych MIT-BIH Arrhythmia Database
- Wstępne przetworzenie sygnału polegające na filtracji pasmowoprzepustowej oraz odjęciu linii izoelektrycznej
- Wyznaczenie przybliżonych miejsc występowania zespołów QRS na podstawie pliku .atr dołączonego każdego z badań
- Dla każdego z zespołów, które zostały sklasyfikowane jako normalny - N lub komorowy - V wyznaczenie cech sygnału
- Stworzenie macierzy zawierającej w kolejnych wierszach zestawu cech dotyczącej każdego z zespołów QRS, w ostatniej kolumnie informacja o poprawnym przyporządkowaniu (wartość +1 dla N, wartość -1 dla V)
- Stworzenie zbioru uczącego i testowego
- Uczenie modelu maszyny wektorów nośnych przy użyciu algorytmu sequential minimal optimization
- Walidacja modelu z użyciem zbioru testowego

6.3 Obliczenie cech kompleksów QRS

Podawanie do algorytmu uczącego całego sygnału bądź też interesujących fragmentów byłoby niepraktyczne z racji dużej złożoności obliczeniowej takiego rozwiązania. Znalezienie prostych do obliczenia cech zespołów QRS, które jednocześnie są w stanie pogrupować dane wejściowe według rodzaju ewolucji serca byłoby najbardziej korzystną sytuacją. [3]

6.3.1 Matematyczny sposób obliczenia cech

Do analizy wybrano 5 prostych do obliczenia parametrów, do których obliczenia wykorzystuje się fragmenty sygnału uznane jako należące do zespołu QRS. [5] Te parametry to:

- Stosunek pola do obwodu (współczynnik Malinowskiej)

$$p_1 = \frac{\sum_{k=1}^N s[k]}{\sum_{k=2}^N s[k] - s[k-1]} \quad (26)$$

- Wartość międzyszczytowa

$$p_2 = \frac{\max_{k \in <1, N>} s[k]}{\min_{k \in <1, N>} s[k]} \quad (27)$$

- Procent próbek sygnału, które są ujemne

$$p_3 = 100\% * \frac{\sum_{k=1}^N u[k]}{N} \quad (28)$$

Gdzie:

$$u[k] = \begin{cases} 1 & \text{gdy } s[k] < 0 \\ 0 & \text{gdy } s[k] \geq 0 \end{cases}$$

- Stosunek maksymalnej prędkości do maksymalnej amplitudy

$$p_4 = \frac{\max_{k \in \langle 3, N \rangle} s[k] - s[k-2] + 2s[k-1]}{|\max_{k \in \langle 1, N \rangle} s[k] - \min_{k \in \langle 1, N \rangle} s[k]|} \quad (29)$$

- Stosunek liczby próbek sygnału, których prędkość przekracza 40% maksymalnej prędkości obserwowanej w sygnale

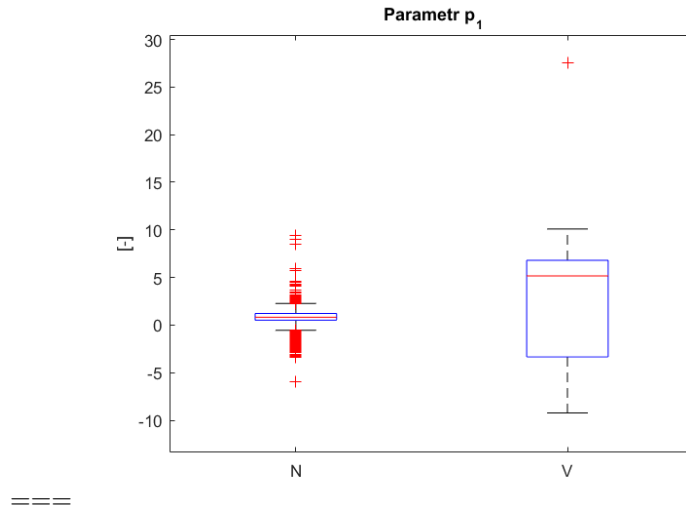
$$p_5 = \frac{\sum_{k=1}^N g[k]}{N} \quad (30)$$

Gdzie:

$$g[k] = \begin{cases} 1 & \text{gdy } |s[k] - s[k-1]| > 0.4 \max_{m \in \langle 2, N \rangle} |s[m] - s[m-1]| \\ 0 & \text{gdy } |s[k] - s[k-1]| \leq 0.4 \max_{m \in \langle 2, N \rangle} |s[m] - s[m-1]| \end{cases}$$

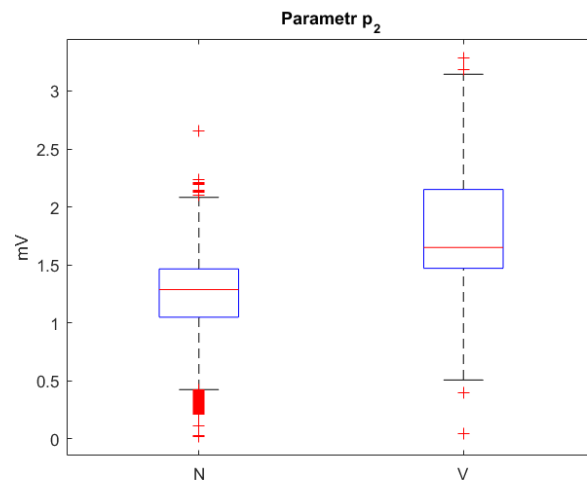
6.3.2 Klasyfikacja przy użyciu obliczonych cech

W celu wstępnej oceny wybranych cech, wybrano losowo 5 sygnałów z bazy danych (100.dat, 107.dat, 114.dat, 200.dat, 208.dat). Wyodrębniono z nich 9306 zespołów QRS, które sklasyfikowano jako normalne (N) bądź komorowe (V). Poniżej przedstawione są wykresy pudełkowe pokazujące zakres zmienności kolejnych cech dla obu grup.

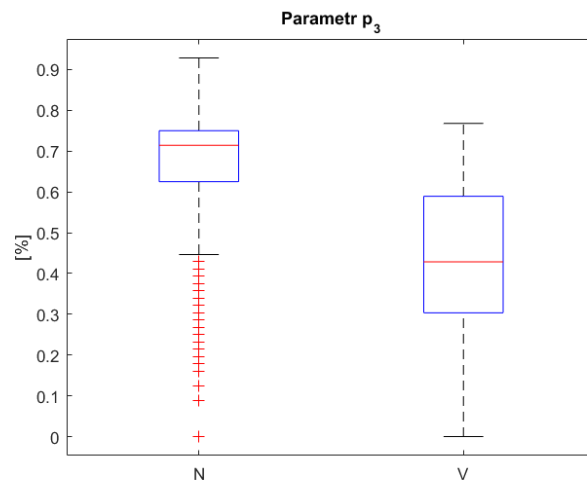


===

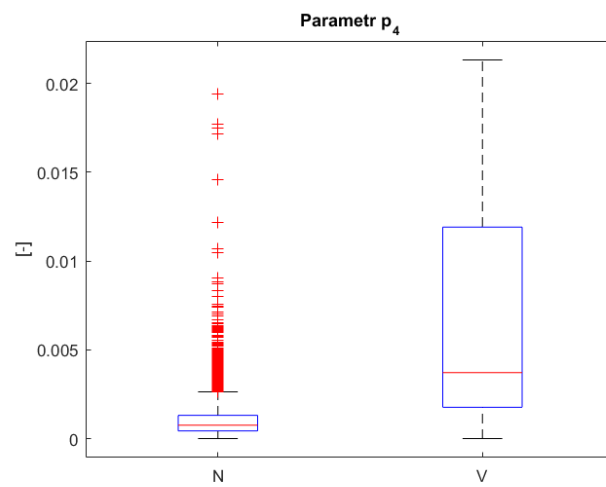
Rysunek 14: Parametr p_1 dla obu grup



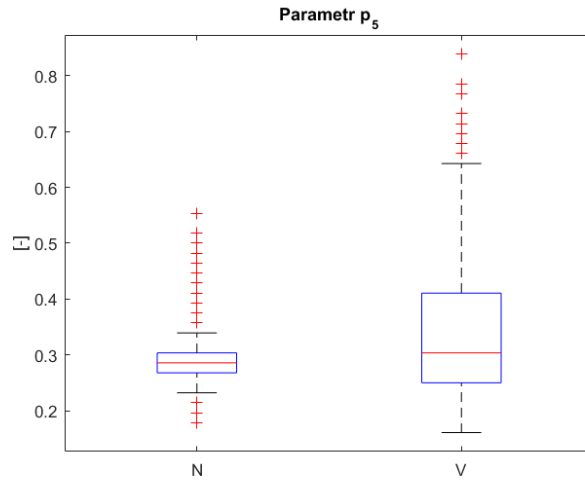
Rysunek 15: Parametr p_2 dla obu grup



Rysunek 16: Parametr p_3 dla obu grup

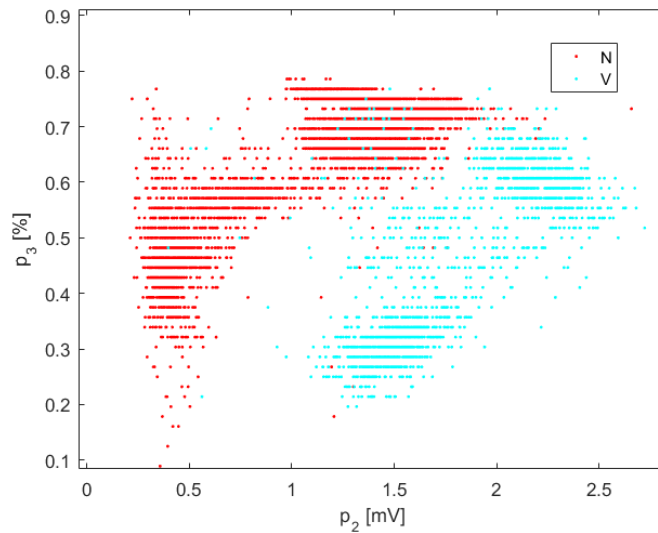


Rysunek 17: Parametr p_4 dla obu grup



Rysunek 18: Parametr p_5 dla obu grup

Jak można zauważyć niektóre cechy (przede wszystkim p_2 i p_3) dobrze dokonują rozróżnienia. W celu potwierdzenia tej obserwacji stworzono wykres obserwacji w przestrzeni $p_2 - p_3$.



Rysunek 19: Zbiór testowy na płaszczyźnie p_2-p_3

Pokazuje on, że klasyfikacja już w przestrzeni tych dwóch cech może dawać satysfakcjonujące rezultaty. Do algorytmu uczącego podano jednak macierz zawierającą wszystkie 5 cech zespołów QRS.

6.4 Metoda maszyny wektorów nośnych (SVM)

6.4.1 Podstawowe założenia metody

Metoda wektorów nośnych dokonuje binarnej (na dwa pozbiory) klasyfikacji z nadzorem. Zbiór uczący składa się z N wektorów

$$x_i \in R^d (i = 1, 2, 3...) \quad (31)$$

z d -wymiarowej przestrzeni cech X . Z każdym wektorem stowarzyszona jest wartość

$$y_i \in \{-1, +1\} \quad (32)$$

Metoda SVM w najprostszym (liniowym) wydaniu szuka hiperpłaszczyzny rozdzielającej dwie klasy w przestrzeni X tak, aby hiperpłaszczyzna była najbardziej oddalona od obserwacji zbioru uczącego (maksymalny odstęp).

W przypadku, gdy zbiory nie są liniowo separowalne, obie klasy są wcześniej mapowane do innej przestrzeni cech (o wyższej wymiarowości). Więcej o tym w kolejnym podrozdziale.

Decyzja dotycząca klasyfikacji polega na obliczeniu wartości funkcji dyskryminacyjnej $f(x)$.

$$f(x) = \omega^* \Phi(x) + b^* \quad (33)$$

Optymalna hiperpłaszczyzna jest zdefiniowana przez wektor wag

$$\omega^* \in R^{d'} \quad (34)$$

i bias

$$b^* \in R \quad (35)$$

Optymalizacja klasyfikatora polega na minimalizacji błędu (tj. sumy kar za źle sklasyfikowane obserwacje) oraz maksymalizacji szerokości odległości między hiperpłaszczyzną rozdzielającą a obserwacjami. Przez odpowiedni dobór parametrów można większą wagę postawić na jedną z tych zależności.

6.4.2 Radial Basis Function

Radial Basis Function (lub radialna funkcja bazowa) jest funkcją zależącą jedynie od odległości od pewnego przyjętego środka. Każda funkcja spełniająca warunek

$$\phi(x) = \phi(\|x\|) \quad (36)$$

jest funkcją radialną.

Wprowadzenie radialnej funkcji bazowej związane jest z tym, że w wielu przypadkach niemożliwy jest skuteczny podział w wyjściowej przestrzeni cech. Funkcje bazowe usuwają tę przeszkodę przenosząc problem klasyfikacji do przestrzeni o wyższej wymiarowości, co schematycznie pokazuje poniższy rysunek.

W algorytmie wykonanym w ramach projektu wykorzystano Gaussowską radialną funkcję bazową opisaną wzorem:

$$k(\vec{x}_i, \vec{x}_j) = \exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2}\right) \quad (37)$$

6.4.3 Matematyczne podstawy implementacji

Zastosowana metoda optymalizacji modelu SVM to Sequential minimal optimization w skrócie SMO. Jest to metoda rozwiązywania zadania optymalizacyjnego polegającego na znalezieniu ekstremum funkcji kwadratowej. Przy implementacji oparto się na opisie z artykułu [4].

SMO jest efektywną metodą rozwiązania problemu optymalizacji, który może być opisany następującym równaniem

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^i y^j \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \quad (38)$$

Gdzie:

$$0 \leq \alpha_i \leq C, \text{ dla } i = 1, 2, 3, \dots, m \quad (39)$$

$$\sum_{i=1}^m y^{(i)} \alpha_i = 0 \quad (40)$$

Warunek znalezienia optymalnego rozwiązania (zwany warunkiem Karusha-Kuhna-Tuckera) dla tego problemu jest zdefiniowany następująco

$$\alpha_i = 0 \implies y^{(i)}(\omega^T x^{(i)} + b) \geq 1 \quad (41)$$

$$\alpha_i = C \implies y^{(i)}(\omega^T x^{(i)} + b) \leq 1 \quad (42)$$

$$0 < \alpha_i < C \implies y^{(i)}(\omega^T x^{(i)} + b) = 1 \quad (43)$$

Algorytm SMO iteruje aż do momentu, gdy te warunki zostaną spełnione przez to zapewniając jego zbieżność do optymalnego rozwiązania.

Składa się on z 3 kroków [4]:

- Wybór parametrów α W zaimplementowanym algorytmie zastosowano najprostszy wybór parametrów, tj. w pętli iteruje się przez wszystkie α_i , a następnie losuje się α_j takie, że $i \neq j$. Istnieje niewielkie prawdopodobieństwo, że któraś z par parametrów, które wprowadziłyby istotną zmianę do ostatecznego wyniku zostanie pominięta. Założono jednak, że algorytm musi wykonać przynajmniej 10 (domyślna wartość) "pustych przebiegów", aby zminimalizować takie prawdopodobieństwo. "Pusty przebieg" polega na tym, że w danej iteracji pętli zewnętrznej nie zostają wprowadzone żadne zmiany do modelu.
- Optymalizacja α_i i α_j Obliczane są ograniczenia, które musi spełniać α_j , a następnie rozwiązywany jest ograniczony problem optymalizacyjny. Następnie obliczana jest α_j optymalna dla danego kroku, która jest następnie ogranicza jeśli posiada wartość spoza założonego zakresu. Na podstawie α_j obliczana jest nowa wartość α_i . Odpowiednie wzory dostępne są [4].
- Obliczenie biasu b Po obliczeniu α_i i α_j , wybierana jest taka wartość b , żeby warunek znalezienia optymalnego rozwiązania był spełniony. Sprawdzane jest, czy obliczone uprzednio α_i i α_j znajdują się na granicach przedziału $< 0, C >$. Jeśli nie, to dane rozwiązanie jest prawidłowe. Jeśli oba nie są na granicach przedziału wartość b jest średnią arytmetyczną b_1 i b_2 obliczoną odpowiednio dla α_i i α_j . Szczegóły: [4].

6.4.4 Implementacja w C++

W przygotowaniu programu w C++ wykorzystano bibliotekę Eigen, która okazała się szczególnie przydatna z racji wbudowanych funkcji obsługujących macierze i wektory. Najważniejsze elementy programu to:

- Klasa SVM - jest to klasa obsługująca proces uczenia modelu (metoda train), a następnie jego testowania (metoda predict), posiada szereg funkcji prywatnych funkcji pomocniczych
- Funkcja parseData - jest to funkcja odpowiadająca za wczytanie odpowiednio zdefiniowanego pliku CSV i stworzenie na jego podstawie struktury zawierającej 2 macierze (treningową, testową) i 2 wektory (poprawne przyporządkowanie dla zbioru treningowego i testowego)
- Funkcje writeDecisionToCSV i appendToOutputFile - odpowiadające za zapis do plików działania modelu
- Struktura SVMOptions - zawierający parametry działania metody maszyny wektorów nośnych oraz domyślny konstruktor przyjmujący wartości domyślne
- Struktura ParsedData - zawierający całość danych potrzebnych do uczenia i testowania modelu

Funkcja główna main zawiera kod wywołujący poszczególne elementy programu:

```
int main(int argc, char** argv){

// wczytaj dane z plikow, ktorych nazwe podano w konsoli
// - zalożono, ze pliki z danymi znajduja sie w tym samym
// folderze co program

ParsedData data = parseData(argv);

// utwórz model, podano domyslne parametry tj. C(1.0),
// tol(0.0001), sigma(0.5), iterLimit(10000),
// passLimit(10)

SVM model(data.trainSet, data.trainSetOut, SVMOptions());

// trenuj model na danych podanych w konstruktorze
model.train();

// dokonaj klasyfikacji danych podanych w pliku
// testowym; zapisz wektor z klasyfikacja do pliku CSV o
// nazwie out.csv

VectorXd decision = model.massPredict(data.testSet);
writeDecisionToCSV(decision);

// ocen dzialanie klasyfikatora (ilosc poprawnie
// sklasyfikowanych ewolucji serca N/V) oraz czas
// uczenia i testow
```

```

evaluateModel(data.testSetOut , decision );
appendToOutputFile(model.trainTime , model.testTime );

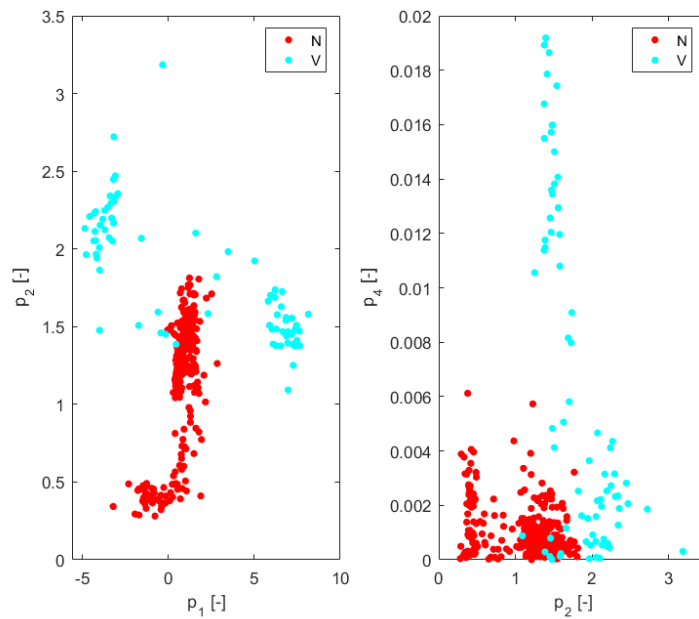
return 0;
}

```

6.5 Wyniki

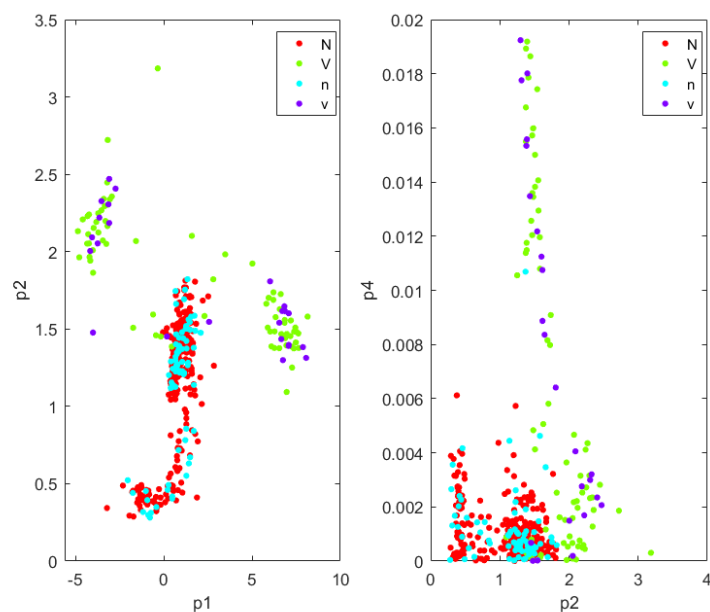
Spośród ponad 9 tysięcy obserwacji do trenowania wybrano losowo 400 (322 zespoły QRS typu N, 76 zespoły typu V, jest to naturalna proporcja w jakiej występowały te zespoły w przebiegach 5 sygnałów, które wybrano losowo z bazy danych), a do testowania 100.

Zbiór uczący pokazano poniżej, w płaszczyźnie p_1 - p_2 (po lewej) i p_2 - p_4 (po prawej).



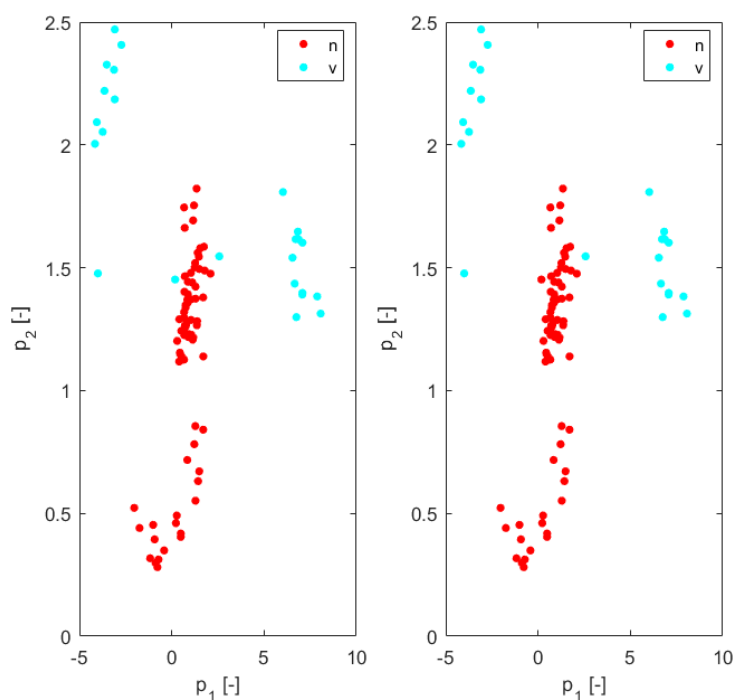
Rysunek 20: Zbiór uczący na płaszczyźnie p_1 - p_2 (po lewej) i p_2 - p_4 (po prawej)

Poniżej zwizualizowano zbiór testowy (n/v) i treningowy (N/V) na jednym wykresie w tych samych płaszczyznach.



Rysunek 21: Zbiór uczący i testowy na płaszczyźnie p_1 - p_2 (po lewej) i p_2 - p_4 (po prawej)

Zaimplementowana metoda klasyfikacji poprawnie przyporządkowała 99 spośród 100 obserwacji. Na wykresie porównawczym (po lewej poprawne przyporządkowanie, po prawej wynik działania algorytmu) oznaczono źle sklasyfikowaną obserwację.



Rysunek 22: Poprawnie sklasyfikowany zbiór uczący na płaszczyźnie p_1 - p_2 (po lewej) i wynik działania algorytmu w tej samej płaszczyźnie (po prawej)

Porównanie działania algorytmu SVM RBF z algorytmami innych grup znajduje się w osobnym raporcie.

6.6 Wnioski

Klasyfikator zespołów QRS stworzony w ramach projektu działa satysfakcjonująco dla wybranych cech fragmentów sygnału i zaimplementowanego algorytmu SVM.

W ramach projektu wykorzystano sprawdzone, ale proste matematycznie parametry sygnału i potwierdziły one swoją zdolność do rozróżniania zespołów QRS o różnym pochodzeniu pobudzenia. W ramach projektu porównywano zespoły sklasyfikowane jako normalne (N) oraz komorowe (V). Należy przyjąć, że gdyby dokonano bardziej szczegółowego podziału działanie klasyfikatora uległoby pogorszeniu. Już w początkowej fazie projektu stwierdzono, że bardzo ciężko byłoby odróżnić zespoły normalne (N) od nadkomorowych (SV) wg klasyfikacji. [2]

Spśród wszystkich metod SVM realizowanych w projekcie metoda SVM RBF zapewnia najlepszą skuteczność rozdziału cech, która w przestrzeni, w której są zdefiniowane nie są liniowo separowalne, co stanowi niewątpliwą zaletę tego algorytmu. [1]

Czas trenowania klasyfikatora tj. ok. 20 sekund (na komputerze jednego z autorów) dla 400 obserwacji po 5 cech każda można uznać za zadowalający, ale z pewnością istnieje wiele metod dalszej optymalizacji kodu.

Literatura

- [1] Bazi Yakoub, Melgani Farid *Classification of Electrocardiogram Signals With Support Vector Machines and Particle Swarm Optimization* IEEE Transactions on Information Technology in Biomedicine 12, 2008
- [2] da Luza E. J. S, et al. *ECG-based Heartbeat Classification for Arrhythmia Detection: A Survey* Computer methods and programs in biomedicine, 2016
- [3] Augustyniak Piotr *Przetwarzanie sygnałów elektrodiagnostycznych* Wydawnictwa AGH, Kraków, Poland, 2001
- [4] Husanbir Singh Pannu *Simplified SMO Algorithm*
<http://math.unt.edu/hsp0009/smo.pdf>
- [5] Augustyniak Piotr *Optymalizacja wyboru reprezentacji zespołów skurczowych dla celów klasyfikacji zapisów holterowskich*
<http://galaxy.uci.agh.edu.pl/august/pub/pdf/p15.pdf>

7 Liniowa Analiza Dyskryminacyjna

Liniowa analiza dyskryminacyjna (ang. Linear Discriminant Analysis, LDA) należy do technik klasyfikacji z nadzorem – wykorzystuje zbiór uczący do stworzenia reguł (klasyfikatorów), które pozwalają na przypisanie nowego obiektu, o nieznanym przynależności, do którejś z klas. Klasyfikatory są liniowymi funkcjami dyskryminacyjnymi, których współczynniki dobrane są tak, by funkcje te jak najlepiej separowały od siebie obiekty zbioru uczącego należące do różnych klas. Reguły klasyfikacyjne dzielą zatem przestrzeń cech na podzbiory odpowiadające klasom (liczba podzbiorów równa jest liczbie klas). W przypadku, gdy liczba klas wynosi dwa, wystarczające jest wyznaczenie jednej funkcji dyskryminacyjnej. W celu przypisania nowego obiektu do klasy należy obliczyć wartość funkcji dyskryminacyjnej dla jego zmiennych.

$$F = a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n \quad (44)$$

gdzie:

$a_0, a_1, a_2, \dots, a_n$ - współczynniki funkcji dyskryminacyjnej

x_0, x_1, \dots, x_n - zmienne

n - liczba zmiennych

7.1 Wyznaczenie klasyfikatora

Metoda LDA opiera się na tym, iż w każdym zbiorze danych (w każdej klasie) wyniki tworzą rozkład normalny. Na tej podstawie zakłada się, że model posiada taką samą macierz kowariancji dla obydwu klas. To, co je odróżnia, to średnia wartość oraz prawdopodobieństwo wystąpienia. W celu utworzenia liniowego klasyfikatora wykonano odpowiadające algorytmowi matematyczne operacje bazujące na [1] oraz [2].

Jako pierwsze zdefiniowano prawdopodobieństwo wystąpienia danej klasy w stosunku do wszystkich próbek na podstawie wzoru

$$\pi = N_i/N_k \quad (45)$$

gdzie:

N_i - liczba próbek klasy i

N_k - liczba wszystkich próbek

Następnie wyznaczono średnią wartość przyjmowaną przez ustalone cechy dla danej klasy. Zebrane średnie utworzyły wektor μ_i , którego kolejne elementy odpowiadały wyznaczonym cechom. Kolejnym krokiem jest obliczenie macierzy kowariancji, która według założeń metody, jest jednakowa dla obydwu klas.

$$E = \sum_{k=1}^K \sum_{i=1}^N \frac{(x_i - \mu_k)(x_i - \mu_k)^T}{N - K} \quad (46)$$

gdzie:

N - liczba próbek klasy i

K - liczba klas

x_i - wartości cech dla kolejnych próbek i

μ - wektor średnich wartości cech klasy k

Wartości macierzy kowariancji, prawdopodobieństw oraz średnich posłużyły do wyznaczenia wyrazu wolnego oraz współczynników funkcji dyskryminacyjnej.

$$a_0 = \log \frac{\pi_1}{\pi_2} - \frac{1}{2}(\mu_1 + \mu_2)^T E^{-1}(\mu_1 - \mu_2) \quad (47)$$

$$(a_1, a_2, \dots, a_n)^T = E^{-1}(\mu_1 - \mu_2) \quad (48)$$

gdzie:

n - liczba cech

Uzyskawszy wartości współczynników możliwe było przeprowadzenie klasyfikacji nieznanego zbioru testowego. Dla kolejnych próbek wyznaczano ich wartości funkcji dyskryminacyjnej oraz przyrównywano do warunku > 0 lub < 0 , co pozwoliło rozdzielić badany zbiór na dwie grupy.

7.2 Matlab - implementacja

Do wykonania modelu klasyfikacji użyto 7 sygnałów z bazy MIT-BIH Arrhythmia Database, które były już odpowiednio przetworzone i pozwoliły na utworzenie wektorów próbek dla załamek R, a następnie zespołów QRS, którym przyporządkowane były zdefiniowane klasy. Próbkę dla Q oraz S zostały oszacowane na podstawie typowego czasu trwania całego zespołu, który w celu zapewnienia detekcji został zawyżony. Różnicę pomiędzy Q a R ustalono o wartości 63 ms, a R – S: 94 ms. Aby stworzyć macierze klas zawierające załamki jednej klasy, lecz pochodzące z różnych sygnałów, napisano funkcję, która grupowała te dane. Dla grup zespołów QRS obliczono cechy charakterystyczne. Tak utworzone wektory cech poddano Liniowej Analizie Dyskryminacyjnej, której celem było uzyskanie odpowiedniego klasyfikatora. Model przetestowano i oceniono na przygotowanym zbiorze testowym.

Tabela 3 przedstawia uzyskane rezultaty klasyfikacji zespołów QRS metodą LDA w założeniu prototypowym. Przyjęta koncepcja rozwiązania zaowocowała poprawną klasyfikacją 540 obserwacji spośród 600, co daje wynik 90% poprawnej klasyfikacji.

Tabela 3: Zestawienie wyników klasyfikacji metodą LDA w środowisku Matlab

Real/Predicted	N	VEB
N	92,33%	7,67%
VEB	12,33%	87,67%

7.3 C++ - implementacja

Implementacja opisanego w przedostatnim rozdziale rozwiązania została wykonana w języku C++. Oprócz przedstawionych wcześniej matematycznych operacji wczytano przygotowane dane oraz odpowiednio je przygotowano. Algorytm rozwiązania został zaprezentowany na schemacie (Rys. 23).

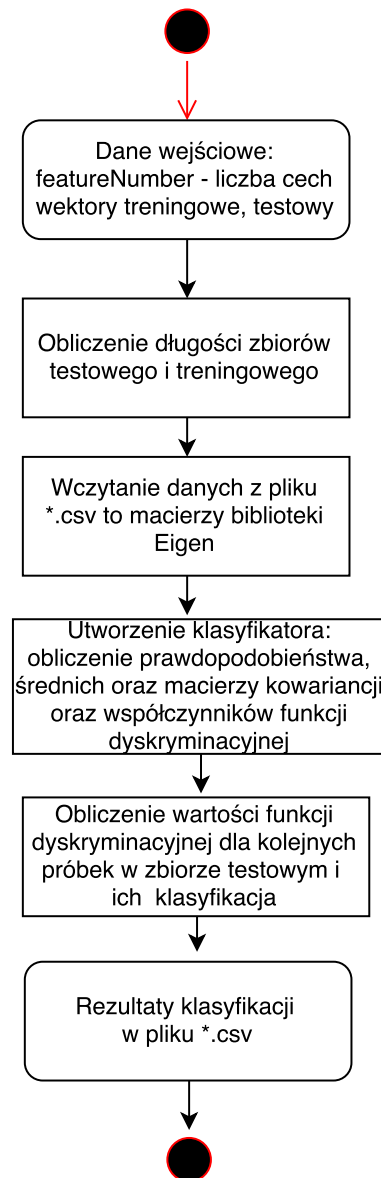
7.4 Podsumowanie

Algorytm liniowej analizy dyskryminacyjnej zaimplementowany w C++ dał podobne wyniki jak analiza przeprowadzona z wykorzystaniem środowiska Matlab. Skuteczność klasyfikacji na poziomie 90% można określić jako dobrą. Przy manualnej analizie oceniono, że klasę przeważającą w liczebności, algorytm klasyfikował z większą poprawnością. LDA można określić jako metodę optymalną do zastosowań w przypadku dwóch klas, ponieważ działa szybko i z zadowalającym wynikiem.

W celu wykonania algorytmu oraz analizy w programie Matlab wykorzystano publikację wymienioną w poniższej bibliografii.

Literatura

- [1] Jia Li *Linear Discriminant Analysis*, Department of Statistics The Pennsylvania State University <http://sites.stat.psu.edu/~jiali/course/stat597e/notes2/lda.pdf> dostęp 12/01/2017
- [2] Mathworks *Discriminant Analysis* <https://www.mathworks.com/help/stats/discriminant-analysis.html> dostęp 12/01/2017
- [3] Augustyniak Piotr *Przetwarzanie sygnałów elektrodagnostycznych* Wydawnictwa AGH, Kraków, Poland, 2001



Rysunek 23: Schemat zaimplementowanego algorytmu LDA

8 Quadratic SVM

8.1 SMO

Jedną z metod implementacji maszyny wektorów nośnych jest zastosowanie minimalnej optymalizacji sekwencyjnej SMO dekomponującej problem SVM na podproblemy, które są rozwiązywane analitycznie, lecz podproblemy są tylko dwuparametrowe.

8.1.1 Opis ogólny procedury

Dekompozycja – wyróżnienie 2 podzbiorów parametrów: zbiór aktywny i zbiór pasywny. W każdym kroku iteracyjnym jedynie parametry aktywne są optymalizowane. Ogólna regułą algorytmów opierających się na dekompozycji jest taka optymalizacja zbioru aktywnego, by aktualne rozwiązanie było jak najbliższe maksimum globalnemu. Sposób wyboru aktywnego zbioru parametrów opiera się na heurystykach.

Kroki jakie są wykonywane w heurystyce

1. Wybór podzbioru parametrów aktywnych.
2. Wyznaczenie rozwiązania optymalnego dla wybranych parametrów.
3. Utworzenie nowego zbioru parametrów aktywnych który składa się z
 - Wszystkich wektorów wspierających otrzymanych w poprzednim rozwiązaniu
 - M wektorów ze zbioru pasywnego spełniającego najgorzej kryterium KKT (M – parametr systemu).
4. W momencie spełnienia kryterium stop proces iteracyjny zostaje zatrzymany.

Gdy C równie jest nieskończoność wówczas mamy do czynienia z idealną separacją – pomiędzy hiperpłaszczyznami nie znajdują się żadne punkty. Gdy $\alpha_i > 0$ to i-ty wektor jest wektorem wspierającym. Gdy $\alpha_i = 0$ to i-ty wektor może być wektorem wspierającym.

8.1.2 Rozwiązanie analityczne dla dwóch punktów

1. Dwoma wybranymi parametrami do zbioru aktywnego są parametry α_1 i α_2 . Przy wyborze parametrów w algorytmie SMO zaproponowanym przez Platt'a odbywa się:

- α_1 wybierany się spośród wszystkich parametrów, które nie spełniają KKT, przy czym pierwszeństwo mają α_1 spełniające $0 < \alpha_1 < C$
- α_2 musi spełniać warunek $\max(E1 - E2)$

2. Wejściowe parametry spełniają warunek równościowy (są równe 0)

$$\sum_{i=1}^l (\alpha_i^{old}) * y_i$$
$$\alpha_1^{old} * y_1 + \alpha_2^{old} * y_2 + \sum_{i=3}^l (\alpha_i^{old}) * y_i = 0$$

3. Nowe wartości parametrów również muszą spełniać warunek nierównościowy $\alpha_1 * y_1 + \alpha_2 * y_2 = \alpha_1^{old} * y_1 + \alpha_2^{old} * y_2$

$$\text{Po przekształceniu } \alpha_2 = \alpha_1^{old} * y_1 * y_2 + \alpha_1^{old} - \alpha_1 * y_1 * y_2$$

Prosta przecina lewy bok kwadratu gdy $\alpha_1 = 0$

Prosta przecina prawy bok kwadratu gdy $\alpha_1 = C$

Po podstawieniu otrzymuje się (gdy współczynnik kierunkowy jest ujemny Rysunek A.1. a) $\alpha_2 = \alpha_1^{old} + \alpha_2^{old}$ $\alpha_2 = \alpha_1^{old} + \alpha_2^{old} - C$ Gdy współczynnik kierunkowy jest dodatni (Rysunek A.1. b))

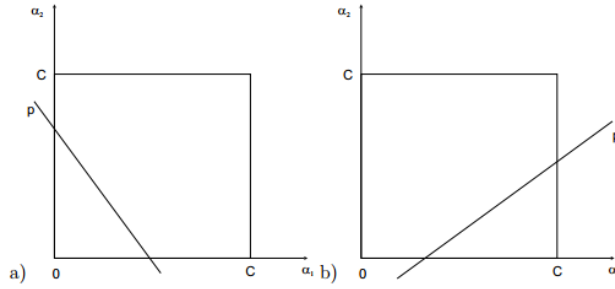
$$\alpha_2 = -\alpha_1^{old} + \alpha_2^{old} \quad \alpha_2 = \alpha_1^{old} + \alpha_2^{old} - C$$

Należy pamiętać, że punkty przecięcia muszą leżeć w obrębie kwadratu.

4. Nowe wartości parametrów muszą również spełniać warunek nierównościowy $0 \geq \alpha_1 \quad C \geq \alpha_2$

5. Ograniczenia dla α_2 $U \geq \alpha_2 \geq V$

- $y_1 \neq y_2$
 $U = \max(0, \alpha_2^{old} - \alpha_1^{old})$
 $V = \max(C, C - \alpha_1^{old} + \alpha_2^{old})$



Rysunek A.1. Rysunek przedstawia prostą p w przypadku a) o współczynniku ujemnym, w przypadku b) dodatnim.

- $y_1 = y_2$
 $U = \max(0, \alpha_2^{old} + \alpha_1^{old} - C)$
 $V = \max(C, \alpha_1^{old} + \alpha_2^{old})$

6. Wyliczenie E_i

$$E_i = \sum_{j=1}^l (\alpha_j) * y_j * K(x_j, x_i) - y_i$$

7. Wyliczenie k_{ij}

$$\kappa = K(x_1, x_2) + K(x_2, x_2) - 2 * K(x_1, x_2)$$

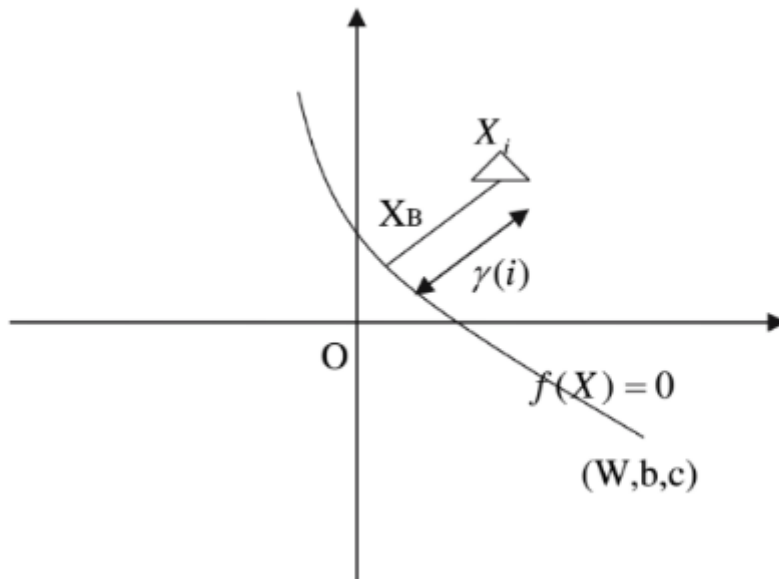
8. Wyliczenie nowych parametrów α

$$\alpha_2^{unc} = \alpha_2^{old} + \frac{y_2 * (E_1 - E_2)}{\kappa}$$

$$\alpha_2 = \begin{cases} V, & \text{jesli } \alpha_2^{\text{new,unc}} > V, \\ \alpha_2^{\text{new,unc}}, & \text{jesli } U \leq \alpha_2^{\text{new,unc}} \leq V, \\ U, & \text{jesli } \alpha_2^{\text{new,unc}} < U, \end{cases}$$

$$\alpha_1^{\text{new}} = \alpha_1^{\text{old}} + y_1 y_2 (\alpha_2^{\text{old}} - \alpha_2^{\text{new}}).$$

8.1.3 QSVM (ang. quadratic kernel-free non-linear support vector machine)



Kwadratowa funkcja decyzyjna jest zdolna do rozdzielania danych nieliniowych. Margines geometryczny jest równy odwrotności gradientu funkcji decyzyjnej. Margines funkcjonalny jest równaniem funkcji kwadratowej.

$QSVN(W, b, c) f(X) = \frac{1}{2} * X^T * W * X + b^T + c f(X) = ct$ może przyjąć wszystkie formy hiperpłaszczyzny (hiper-elipsy, hiper-parabole..) $f(X)$ może być rozważana jako suma $f_{liniowe} + f_{nieliniowa}$

8.2 Implementacja

```

7  int main(int argc, char *argv[])
8  {
9      SVMClassifier *solver = new SVMClassifier();
10
11     clock_t begin = clock();
12     solver->train();
13     clock_t train = clock();
14     solver->predict();
15     clock_t predict = clock();
16     std::cout << "Train time:\t" << (double(train - begin) / CLOCKS_PER_SEC) << endl;
17     std::cout << "Predict time:\t" << (double(predict - train) / CLOCKS_PER_SEC) << endl;
18     std::cout << "Total time:\t" << (double(predict - begin) / CLOCKS_PER_SEC) << endl;
19     int wait;
20     std::cin >> wait;
21
22 }
23

```

Patrząc pod kątem implementacji maszyny wektorów nośnych z jądrem kwadratowym metodą SMO, należy wyróżnić dwa etapy klasyfikatora : uczenie oraz predykcję/klasyfikację. W c++ stworzono klasę SVMClassify(), obiekt klasyfikatora, która zawierała zmienne: **double C** - wartość C typu double

double epsilon - wartość epsilon ustawiona na 0.0001

char fNameTrain[256] - nazwa pliku z próbkami treningowymi, maksymalna długość 256

char fNameTest[256] - nazwa pliku z próbkami testowymi, maksymalna długość 256

char fNameResults[256] - nazwa pliku z rezultatami, maksymalna długość 256

int N - liczba próbek treningowych początkowa wartość 0

int NTestSamples - liczba próbek testowych początkowa wartość 0

TVectorArray arrayX - macierz próbek treningowych

TFloatArray arrayY - macierz próbek testowych

TFloatArray alpha - wektor z wartościami alpha, początkowa inicjalizacja zerami

TFloatArray d - macierz pomocnicza w obliczeniach

TFloatArray arrayError - macierz błędów

float b - wartość b

float bDiff - delta b

Metody klasy SVMClassifier:

SVMClassifier() - konstruktor, tworzy obiekt SVMClassifier

SVMClassifier() - destruktor

int train() - funkcja ucząca klasyfikatora, implementuje główną procedurę uczenia minimalną optymalizacją sekwencyjną, zapisuje rezultaty do pliku

int examineExample(int i1) - funkcja która przy wywołaniu otrzymuje indeks alphy, sprawdza warunki KKT(herustyka) oraz wyszukuje wartości alpha2, następnie wywołuje funkcję takeStep() - optymalizacja dwóch punktów

int takeStep(int i1, int i2)- optymalizacja dwóch mnożników Lagrange'a, zwraca 1 w przypadku udanej optymalizacji, 0 w przypadku nieudanej, i1, i2 - indexy alph

int predict()- funkcja klasyfikująca próbki testowe

float errorRate() - funkcja zwracająca Error Rate;

int loadResults(std::ifstream is)- funkcja otwierająca/tworząca plik z rezultatami

void writeResultModel(std::ofstream os) - funkcja zapisująca rezultaty do pliku

float kernel(int i1, int i2) - funkcja quadratic kernel zwracająca wartość K

`float learnedFnc(int k)` - wywoływane po zoptymalizowaniu, wylicza wartość nauczonej funkcji w punkcie `k`

8.3 Wykorzystanie klasyfikatora do rozpoznawania uderzeń serca

Stworzy klasyfikator Quadratic SVM wykorzystano do klasyfikacji dwóch zbiorów danych 1 oraz 2 (szczegółowy opis danych w rozdziale 9.1 Opis zbiorów danych). Poniżej w tabelach przedstawiono wyniki, które zgodziły się z wynikami otrzymanymi w programie Matlab.

Tabela 4: Skuteczność klasyfikacji - QSVM

Zbiór danych 1	Zbiór danych 2
98%	97%

Literatura

- [1] Platt John *Sequential Minimal Optimization: A fast Algorithm for Training Support Vector Machines* Microsoft Research, Technical Report, MSR-TR-98-12, 1998
- [2] Marcin Orchel *Klasyfikacja danych wielowymiarowych algorytmami SVM* Praca magisterska, Akademia Górniczo-Hutnicza, Wydział Elektrotechniki Automatyki Informatyki i Inżynierii Biomedycznej
- [3] *The Simplified SMO Algorithm* CS229 Machine Learning, Stanford
- [4] Vasant Honavar *Sequential Minimal Optimization for SVM* Iowa State University, Computer Science Department

9 Porównanie algorytmów

Porównania dokonano korzystając z dwóch zbiorów danych, w których każdy składał się z innego zestawu cech opisujących uderzenia serca. W obu grupach uderzenia zostały podzielone na dwie klasy: uderzenia V , określające uderzenia wygenerowane w obrębie komory, zaliczane do paralogicznych oraz uderzenia N reprezentujące normalną, poprawną pracę serca.

Korzystając z wyznaczonych zbiorów danych klasyfikatory porównano pod kątem skuteczności, czasu uczenia, czasu klasyfikacji, czułości i specyficzności w celu wskazania, który z klasyfikatorów najbardziej sprawdza się w rozpoznaniu uderzeń serca.

9.1 Opis zbiorów danych

9.1.1 Zbiór danych 1

Pierwsza grupa danych opisana jest pięcioma cechami, bazującymi na interwałach RR, są nimi:

- Pre interval RR - odległość między aktualnym uderzeniem serca (analizowanym), a poprzednim uderzeniem serca,
- Post interval RR - odległość między analizowanym uderzeniem serca a kolejnym uderzeniem,
- Average RR - średnia długość interwału RR dla całego analizowanego sygnału,
- Ratio1 - stosunek pre RR interwał do post RR interwał,
- Ratio2 - stosunek pre interwału do Average RR.

Cechy zostały wyznaczone dla sygnałów 228.dat i 100.dat pochodzących z bazy danych MIT-BIH. Z wyznaczonych danych utworzono dwie grupy: treningową oraz testową, które odpowiednio składały się z 400 i 100 próbek.

9.1.2 Zbiór danych 2

Kolejna grupa danych również reprezentowana jest przez 5 cech. Cechy te jednak zdecydowanie różnią się od poprzednich. W ich skład wchodzi takie cechy jak:

- Stosunek pola do obwodu (współczynnik Malinowskiej) [wzór nr 26]
- Wartość międzyszczytowa [wzór nr 27]
- Procent próbek sygnału, które są ujemne [wzór nr 28]
- Stosunek maksymalnej prędkości do maksymalnej amplitudy [wzór nr 29]
- Stosunek liczby próbek sygnału, których prędkość przekracza 40% maksymalnej prędkości obserwowanej w sygnale [wzór nr 30]

Z tych otrzymanych danych również wyznaczono grupę treningową i testową. W skład grupy testowej wchodziło 100 próbek, a treningowej 400 próbek.

9.2 Skuteczność klasyfikacji

Jednym z pierwszych parametrów jakie postanowiono porównać jest skuteczność klasyfikacji, czyli stosunek poprawnie rozpoznanych uderzeń serca do wszystkich jakie występują w zbiorze testowym. Poprawność przyporządkowania klasy do danego obiektu jest istotą działania każdego klasyfikatora i jedną z kluczowych jego cech. Procentową skuteczność każdego klasyfikatora dla obu zestawów danych prezentują Tabele 3 i 4.

Tabela 5: Skuteczność klasyfikacji - zbiór danych 1

kNN	ENN	Linear SVM	SVM + RBF	Naive Bayes	LDA	Quadratic SVM
99%	99%	99.7261%	92.8789%	99.8783%	99.73%	98%

Dla sześciu klasyfikatorów skuteczność poprawnego zaklasyfikowania uderzenia serca jest wysoka i wynosi ponad 95% dla obu zestawów danych. Dla zbioru danych 1, skuteczność pięciu klasyfikatorów przekroczyła wartość 92%. Aż pięć klasyfikatorów w 99%-tach i wyżej poprawnie rozpoznały typ uderzenia

Tabela 6: Skuteczność klasyfikacji - zbiór danych 2

kNN	ENN	Linear SVM	SVM + RBF	Naive Bayes	LDA	Quadratic SVM
98%	98%	76%	99%	96%	98%	97%

serca. Najniższą wartość otrzymano dla klasyfikatora SVM + RBF, która wynosi 92,88%. Wynik ten mimo iż najwyższy jaki otrzymano dla tego zbioru danych i tak jest zadowalający. W przypadku zbioru danych 2, skuteczność nie jest już aż tak wysoka, tylko jeden klasyfikator (SVM + RBF) uzyskał wartość 99%. Klasyfikator Linear SVM nie zbyt dobrze sprawdził się w przypadku tego zestawu danych, skuteczność wyniosła jedynie 76%, co w porównaniu z resztą klasyfikatorów nie jest satysfakcjonującym wynikiem. Powodem tak niskiej wartości skuteczności może być charakter występujących cech w zbiorze danych 2.

9.3 Czas uczenia i klasyfikacji

W celu sprawdzenia złożoności obliczeniowej zaproponowanych algorytmów, zmierzono czasy wykonania się kolejnych algorytmów. Wyróżniono dwa czasy: czas uczenia i czas klasyfikacji. Te parametry odgrywają znaczącą rolę w przypadku dużej ilości danych. Pożądanym jest, aby poprawne wyniki zostały uzyskane w jak krótszym czasie. Trzeba zaznaczyć, że sygnały EKG składają się z bardzo dużej ilości uderzeń serca z tego powodu czas klasyfikacji jest w tym przypadku dosyć istotnym parametrem. Czasy uczenia i klasyfikacji zostały umieszczone w Tabelach 5 i 6. Aby dane były bardziej wiarygodne testy przeprowadzono na tym samym komputerze.

Tabela 7: Czasy uczenia i klasyfikacji - zbiór danych 1

	kNN	ENN	Linear SVM	SVM + RBF	Naive Bayes	LDA	QSVM
Czas uczenia	-	-	25003 ms	7865 ms	22.0936 ms	3 ms	24780 ms
Czas klasyfikacji	3563 ms	36328 ms	26 ms	1961.08 ms	1925.88 ms	13 : ms	59 ms

Tabela 8: Czasy uczenia i klasyfikacji - zbiór danych 2

	kNN	ENN	Linear SVM	SVM + RBF	Naive Bayes	LDA	QSVM
Czas uczenia	-	-	8517 ms	13481 ms	21.9034 ms	15 ms	9400 ms
Czas klasyfikacji	109 ms	1094 ms	1 ms	79 ms	29.2936 ms	< 1 ms	65 ms

Testy wykonano na komputerze wyposażonym w procesor Intel Core i5-4200H o taktowaniu 2.80 GHz oraz 8 Gb RAM w systemie operacyjnym Windows 10 64-bit.

W przypadku klasyfikatorów kNN i ENN, nie występuje w ogóle proces uczenia klasyfikatora, dlatego też nie ma informacji w tabeli. Dla pozostałych pięciu klasyfikatorów proces ten występuje i jak można odczytać z tabeli, czasy uczenia są bardzo zróżnicowane. Uczenie klasyfikatora najkrócej trwa w przypadku klasyfikatora Naive Bayes i LDA, jest to spowodowane tym, że na ten proces składa się tylko wyliczenie kilku wartości statystycznych dla każdej klasy. Zdecydowanie dłużej uczenie zajmuje dla klasyfikatorów Linear SVM, Quadratic SVM i SVM + RBF, gdzie to jest już bardziej złożony proces. Czasy te są kilka set razy wyższe w porównaniu z czasem Naive Bayes i LDA. W przypadku zestawu danych 1, najdłuższy czas uczenia zanotowano dla Linear SVM, który wynosił 25s, natomiast dla zbioru danych 2 najwyższą wartość przypadła dla SVM + RBF i wyniosła 13,5s.

Czasy klasyfikacji podobnie jak czasy uczenia są bardzo zróżnicowane. W obu zestawach danych klasyfikacja najkrócej trwała dla Linear SVM i LDA. Różnica ta jest najbardziej widoczna dla pierwszego zestawu danych, gdzie czas klasyfikacji jest kilkadziesiąt razy krótszy niż dla pozostałych klasyfikatorów. Najgorzej pod tym względem wypadł klasyfikator ENN, gdzie dla zbioru 1 wyniósł 36,33s, a dla zbioru 2 10,94s. Dla porównania czas klasyfikacji dla Linear SVM w zbiorze 2 osiągnął wartość 1ms. Jednak trzeba zwrócić uwagę na brak procesu uczenia dla tego klasyfikatora, co spowodowało wyższy czas klasyfikacji w porównaniu z resztą algorytmów, ponieważ algorytm ten porównuje próbkę wejściową z całym zbiorem uczącym dla różnych wartości k. Tak wiele iteracji skutkuje złożonością obliczeniową rosnącą wraz z ilością próbek w zbiorze treningowym.

9.4 Czułość i specyficzność klasyfikatorów

W celu oceny opracowanych algorytmów pod kątem ich dokładności zdecydowano się na obliczenie ich czułości/wrażliwości oraz specyficzności.

Wrażliwość/czułość (*ang.sensitivity*) może być wyznaczona jako stosunek TP do sumy TP oraz FN .

$$sensitivity = \frac{TP}{TP + FN}$$

gdzie

TP (*ang.truepositive*) - liczba poprawnie sklasyfikowanych przykładów z wybranej klasy (*ang.hit*)

FN (*ang.falsenegative*) - liczba błędnie sklasyfikowanych przykładów z tej klasy, tj. decyzja negatywna podczas gdy przykład w rzeczywistości jest pozytywny (błąd pominięcia - z *ang.miss*)

Specyficzność (*ang.specifity*) może być wyznaczona jako stosunek TN do sumy FP oraz TN .

$$specifity = \frac{TN}{FP + TN}$$

TN (*ang.truenegative*) - liczba przykładów poprawnie nie przydzielonych do wybranej klasy (poprawnie odrzuconych z *ang.correctrejection*)

FP (*ang.falsepositive*) - liczba przykładów błędnie przydzielonych do wybranej klasy, podczas gdy w rzeczywistości do niej nie należą (*ang.falsealarm*)

Tabela 9: Porównanie czułości i specyficzności - zbiór danych 1

	kNN	ENN	Linear SVM	SVM + RBF	Naive Bayes	LDA	QSVM
Czułość	99%	99%	99.7129%	92.5678%	99.9681%	98.68%	100%
Specyficzność	100%	100%	100%	100%	98.0132%	95.83%	91.6%

Tabela 10: Porównanie czułości i specyficzności - zbiór danych 2

	kNN	ENN	Linear SVM	SVM + RBF	Naive Bayes	LDA	QSVM
Czułość	97%	100%	100%	100%	97.3684%	99.78%	96%
Specyficzność	100%	92%	4%	96%	92%	98.67%	96%

Dla wszystkich algorytmów wykonano testy czułości i specyficzności na dwóch zbiorach danych. We wszystkich przypadkach czułość wyniosła ponad 90%, jednak zauważono drobne różnice dla różnych zbiorów danych. Wszystkie algorytmy oprócz kNN i QSVM wykazały lepszą czułość dla zbioru danych 2, a nieco gorsze wyniki otrzymano dla zbioru danych 1. Biorąc pod uwagę obydwa analizowane zbiory danych, zaobserwowano że najwyższą czułość wykazał algorytm ENN i Linear SVM, która nie spadła poniżej 99%. Biorąc pod uwagę drugi zbiór danych, najniższą wartość czułości osiągnął algorytm kNN, a przy testowaniu na zbiorze pierwszym – algorytm –SVM+RBF.

Test porównujący specyficzność klasyfikatorów wykazał różne wyniki dla testowanych dwóch zbiorów danych. Podczas testowania na zbiorze drugim, wyniki były bardzo zróżnicowane, algorytm kNN wykazał pełną specyficzność, podczas gdy Linear SVM jedynie 4%. Porównanie specyficzności dla zbioru danych 1 wykazało 100% wartość parametru dla wszystkich metod za wyjątkiem metody Naive Bayes, która była o niecałe 2% niższa od pozostałych. Analizując rezultaty uzyskane przy testowaniu na obydwu zbiorach danych, wysuwa się wniosek, że najwyższą specyficznością charakteryzuje się algorytm kNN.

9.5 Podsumowanie

Biorąc pod uwagę obydwa zbiory danych najwyższą skutecznością i specyficznością wykazały się algorytmy kNN i ENN. Jednak obydwa algorytmy charakteryzuje dość wysoki czas klasyfikacji. Najniższym czasem charakteryzują się algorytmy kNN oraz Naive Bayes. W czułości przodują ENN, Linear SVM oraz Naive Bayes. Pod względem specyficzności zdecydowanie najlepsze rezultaty uzyskujemy przy wykorzystaniu algorytmu ENN. Podsumowując, biorąc pod uwagę wszystkie mierzone cechy algorytmów oprócz czasu działania (sumarycznego uczenia oraz klasyfikacji), najlepsze rezultaty uzyskujemy dzięki metodzie Extended Nearest Neighbour, jednak ze względu na dużą wartość czasu klasyfikacji najbardziej optymalny na podstawie przeprowadzonych testów wydaje się algorytm Naive Bayes oraz Liniowa Analiza Dyskryminacyjna. Kosztem nieco niższej (ale nie spadającej poniżej 92%) wrażliwości i skuteczności,

w klasyfikacji metodą Naive Bayes osiągnięto najlepszy rezultat w zdecydowanie krótszym czasie. Może to okazać się znaczące przy zdecydowanie bardziej licznych zbiorach danych. Natomiast algorytm LDA przy zachowaniu wysokiej skuteczności, osiąga zdecydowanie najniższe czasy klasyfikacji oraz uczenia. Zwłaszcza przy testowaniu wielokrotnym czy też na dużych zbiorach danych, może okazać się to decydujące podczas wyboru algorytmu do klasyfikacji, ponieważ proces uczenia następuje tylko raz, a czasem wielokrotna klasyfikacja zajmie najmniej czasu (ponad 20 razy mniej niż w przypadku algorytmu Naive Bayes). Wykonana analiza wykazała, że różne algorytmy zachowują się w różnych sposób w zależności od specyfiki oraz liczebności zbioru treningowego i zbioru testowego. Oznacza to, że w zależności od zbioru danych oraz innych czynników (jak konieczność ograniczenia czasu klasyfikacji) należy dobierać odpowiednie metody klasyfikacji. Dobrą metodą jest też dokonanie klasyfikacji za pomocą kilku algorytmów, a następnie porównanie rezultatów.