Raport końcowy projektu z przedmiotu Dedykowane Algorytmy Diagnostyki Medycznej

¿autorzy¡

12 stycznia 2017

Klasyfikacja uderzeń serca

Spis treści

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 wcześnie 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 .

W tej pracy do rozpoznania typów uderzeń serca wykorzystano różne klasyfikatory. Dla każdego przedstawiono ich opisy oraz metodę implantacji. Następnie w celu znalezienia optymalnego klasyfikatora do rozpoznawania typów uderzeń serca porówna efekty działania każdego z nich.. Porównano je pod względem skuteczności jaki i czasu potrzebnego na wykonanie algorytmu.

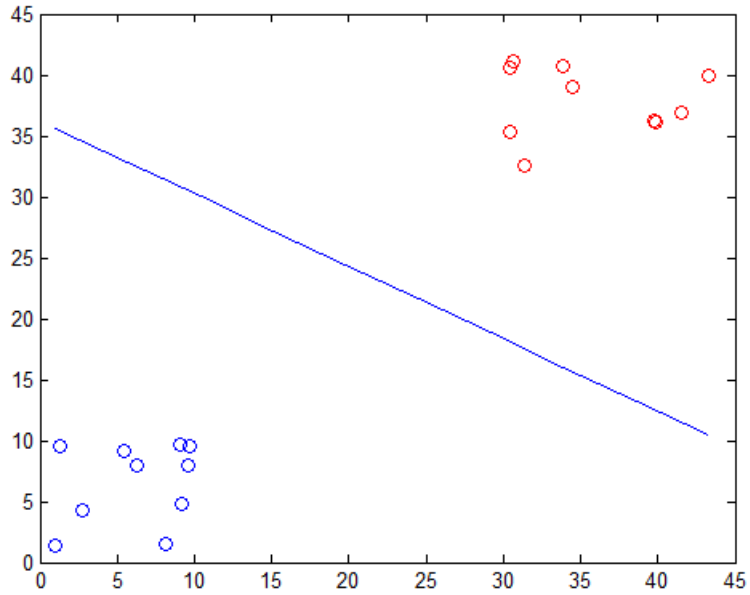
2 Linear SVM

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.

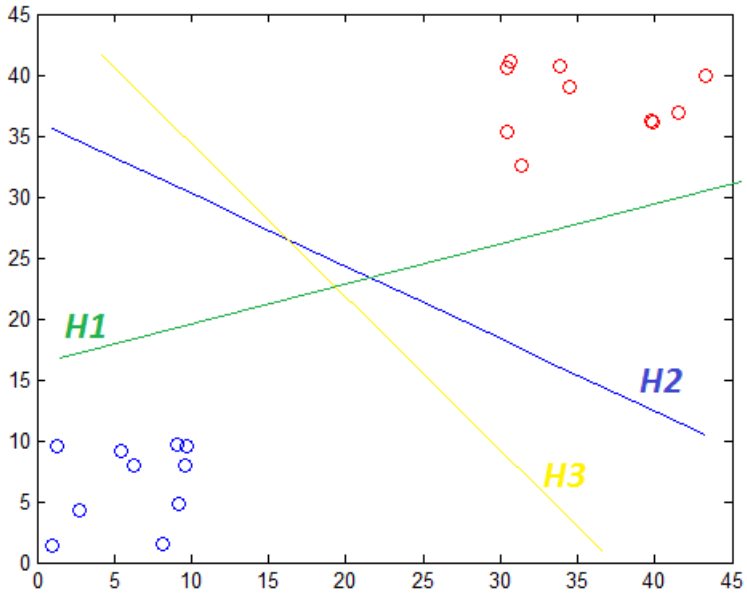
Model SVM jest reprezentacją danych w postaci punktów w przestrzeni. Punkty należące do różnych klas znajdują się w innych obszarach przestrzeni i są rozdzielone za pomocą luki o jak największej możliwej szerokości. Kolejno, analizowany nowy punkt jest odpowiednio przyporządkowywany do jednej z kategorii 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)*

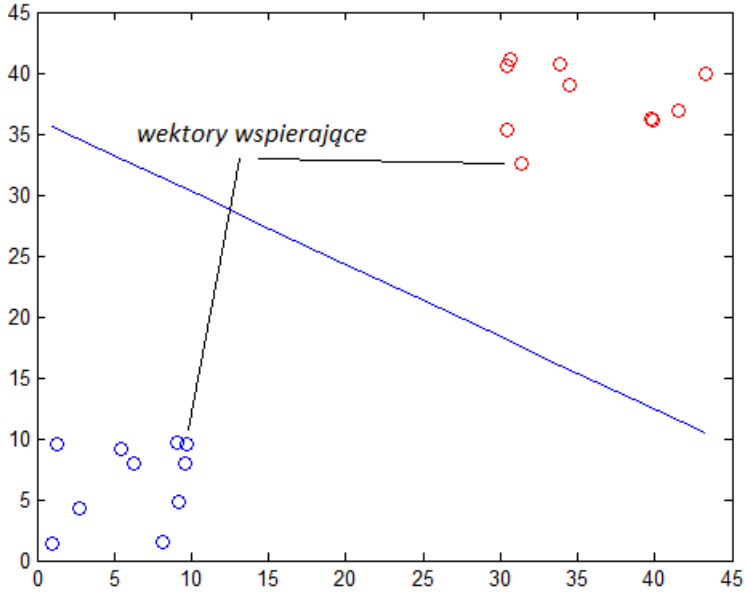
Klasyfikacja danych jest najważniejszym zadaniem uczenia maszynowego. W metodzie wektorów wspierających dane są postaci wektora o rozmiarze p. Celem działania algorytmu jest wyznaczenie optymalnej hiperpłaszczyzny rozmiaru $p-1$, która rozdzielałaby dane należące do poszczególnych klas. Tego typu podejście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

$$(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), ..., (\overrightarrow{x_n}, y_n) \tag{1}$$

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 $\overrightarrow{x_i}$ jest rozmiaru p. Działanie algorytmu SVM polega na znalezieniu hiperpłaszczyzny o maksymalnym marginesie geometrycznym, która dzieli grupy punktów $\overrightarrow{x_i}$, dla których y_i = -1 od grupy punktów $\overrightarrow{x_i}$, dla których $y_i = 1$. Hiperpłaszczyzna H, n-wymiarowa określona jest wzorem:

$$(H)y(\overrightarrow{x}) = 0 \tag{2}$$

gdzie $(y(\overrightarrow{x}) = w^t + b$, 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:

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

margines geometryczny natomiast, będzie dany wzorem:

$$\gamma = \frac{1}{||w||} \tag{4}$$

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:

$$minimalizacja\ ||\overrightarrow{w}||^2$$

przy warunkach:

$$y_i(\overrightarrow{w} * \overrightarrow{x_i} - b) \geq 1$$

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

Można następnie dla powyższego problemu optymalizacyjnego zapisać lagranżjan postaci:

$$L(w, b, \overrightarrow{\alpha}) = \frac{1}{2} \cdot ||\overrightarrow{w}||^2 - \sum_{i=1}^{N} \alpha_i (y_i(\overrightarrow{w} * \overrightarrow{x_i} + b) - 1) \tag{5}$$

gdzie $\overrightarrow{\alpha}$ 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:

$$minimalizacja\ funkcji:$$

$$\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j (\overrightarrow{x_i} \cdot \overrightarrow{x_j}) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i$$

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 $\overrightarrow{w}$ oraz wyraz wolny b może zostać obliczony przy ich pomocy:

$$\overrightarrow{w} = \sum_{i=1}^{N} y_i \alpha_i \overrightarrow{x}_i, \tag{6}$$

$$b = \overrightarrow{w} \cdot \overrightarrow{x}_i - y_i \tag{7}$$

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:

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

przy warunkach:

$$y_i(\overrightarrow{w} \cdot \overrightarrow{x_i} - b) \geq 1 - \xi_i, \forall_i$$

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:

$$minimalizacja \; funkcji :$$

$$\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j K(\overrightarrow{x_i} \cdot \overrightarrow{x_j}) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i$$

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 optymizacją 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:

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

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

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

Drugą pochodną funkcji docelowej wzdłuż odcinka można wyrazić jako:

$$\eta = K(\overrightarrow{x_1}, \overrightarrow{x_1}) + K(\overrightarrow{x_2}, \overrightarrow{x_2}) - 2K(\overrightarrow{x_1}, \overrightarrow{x_2}) \tag{10}$$

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:

$$\alpha_2^n owy = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta}, \tag{11}$$

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

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

Używając oznaczenia $s = y_1 y_2$ wartość α_1 można obliczyć wykorzystując nowy, ograniczony współczynnik $\alpha 2$:

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

W niezwykłych 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:

$$f_1 = y_1(E_1 + b) - \alpha_1 \tag{14}$$

$$f_2 = y_2(E_2 + b) - s\alpha_1 \tag{15}$$

$$K(\overrightarrow{x_1}, \overrightarrow{x_2}) - \alpha_2 K(\overrightarrow{x_2}, \overrightarrow{x_2}), \tag{16}$$

$$L_1 = \alpha_1 + s(\alpha_2 - L), \tag{17}$$

$$H_1 = \alpha_1 + s(\alpha_2 - H), \tag{18}$$

$$\Psi_L = L_1 f_1 + L f_2 + \frac{1}{2} L_1^2 K(\overrightarrow{x_1}, \overrightarrow{x_1}) + \frac{1}{2} L^2 K(\overrightarrow{x_2}, \overrightarrow{x_2}) + sLL_1 K(\overrightarrow{x_1}, \overrightarrow{x_2}), \tag{19}$$

$$\Psi_H = H_1 f_1 + H f_2 + \frac{1}{2} H_1^2 K(\overrightarrow{x_1}, \overrightarrow{x_1}) + \frac{1}{2} H^2 K(\overrightarrow{x_2}, \overrightarrow{x_2}) + sHH_1 K(\overrightarrow{x_1}, \overrightarrow{x_2}). \tag{20}$$

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 zewnętrzna pętla algorytmu - iteruje po całym zbiorze uczącym sprawdzając które z przypadków naruszają warunki KKT. Jeżeli dana próbka nie spełnia tych warunków, może być optymalizowana. 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 optymalizowane. 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:

$$b_1 = E_1 + y_1(\alpha_1^{nowy} - \alpha_1)K(\overrightarrow{x_1}, \overrightarrow{x_1}) + y_2(\alpha_2^{nowy} - \alpha_2)K(\overrightarrow{x_1}, \overrightarrow{x_2}) + b \tag{21}$$

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:

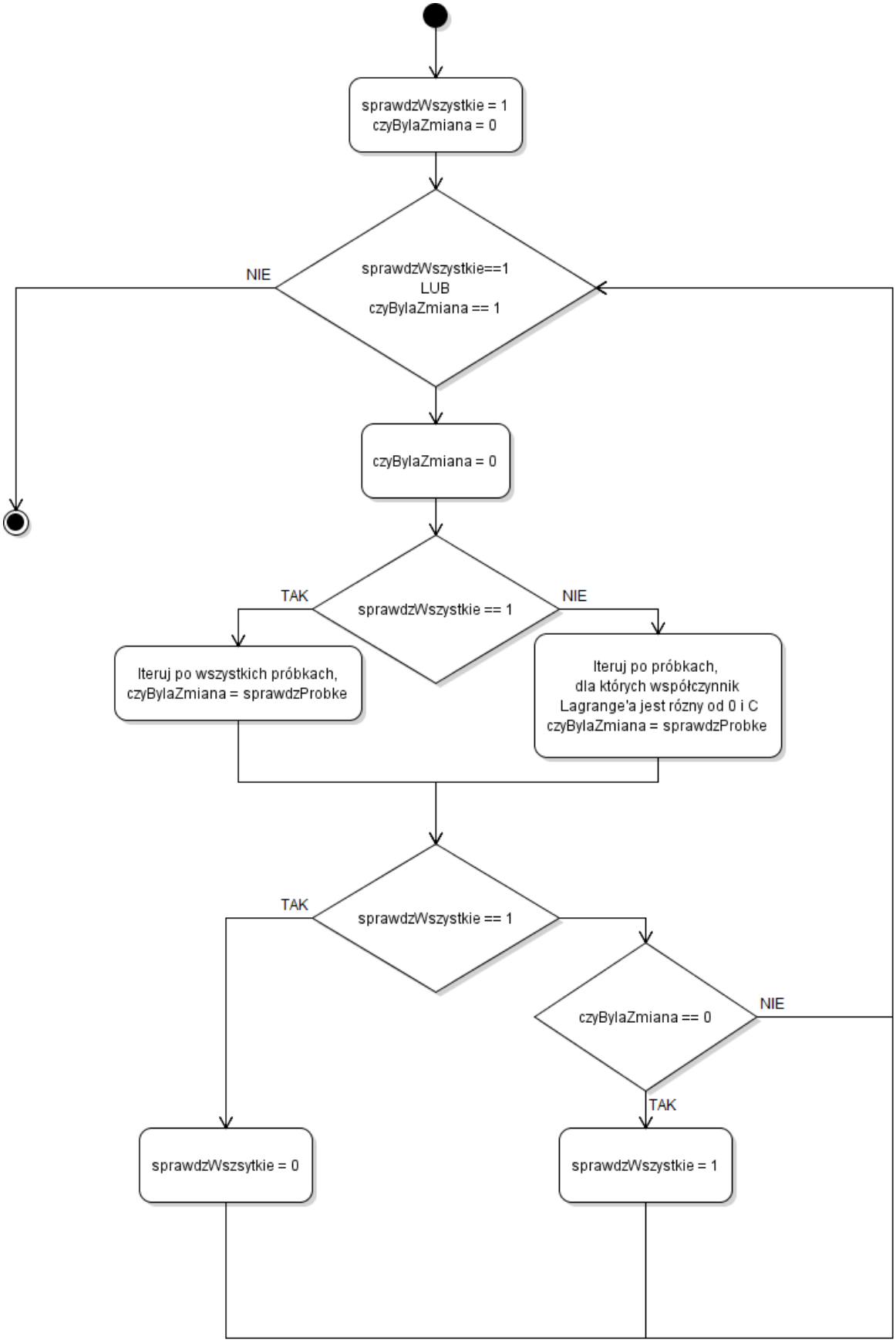
$$b_2 = E_2 + y_1(\alpha_1^{nowy} - \alpha_1)K(\overrightarrow{x_1}, \overrightarrow{x_2}) + y_2(\alpha_2^{nowy} - \alpha_2)K(\overrightarrow{x_2}, \overrightarrow{x_2}) + b \tag{22}$$

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.

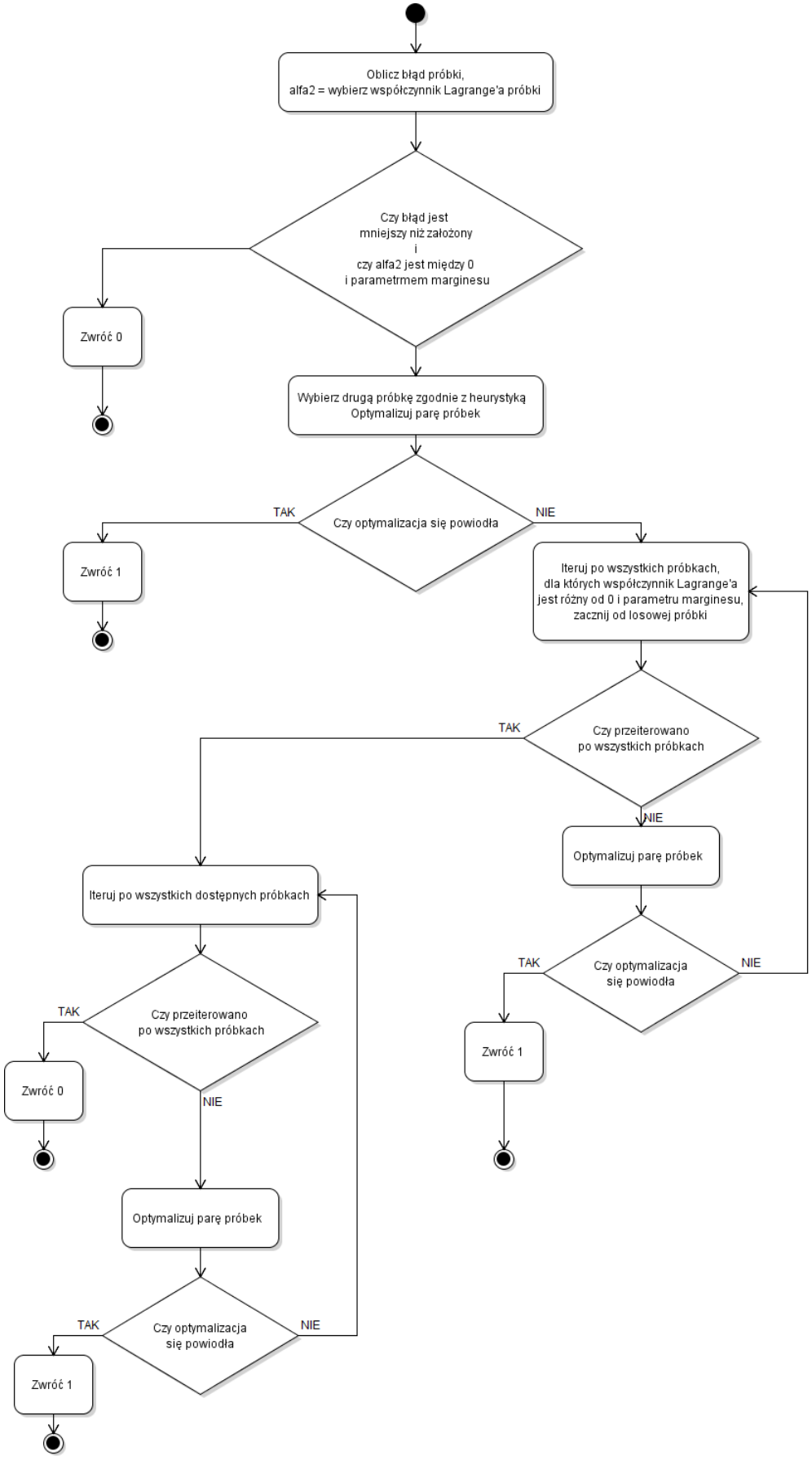
2.2 Opis sposobu implementacji algorytmu

2.2.1 Schematy blokowe działania algorytmu

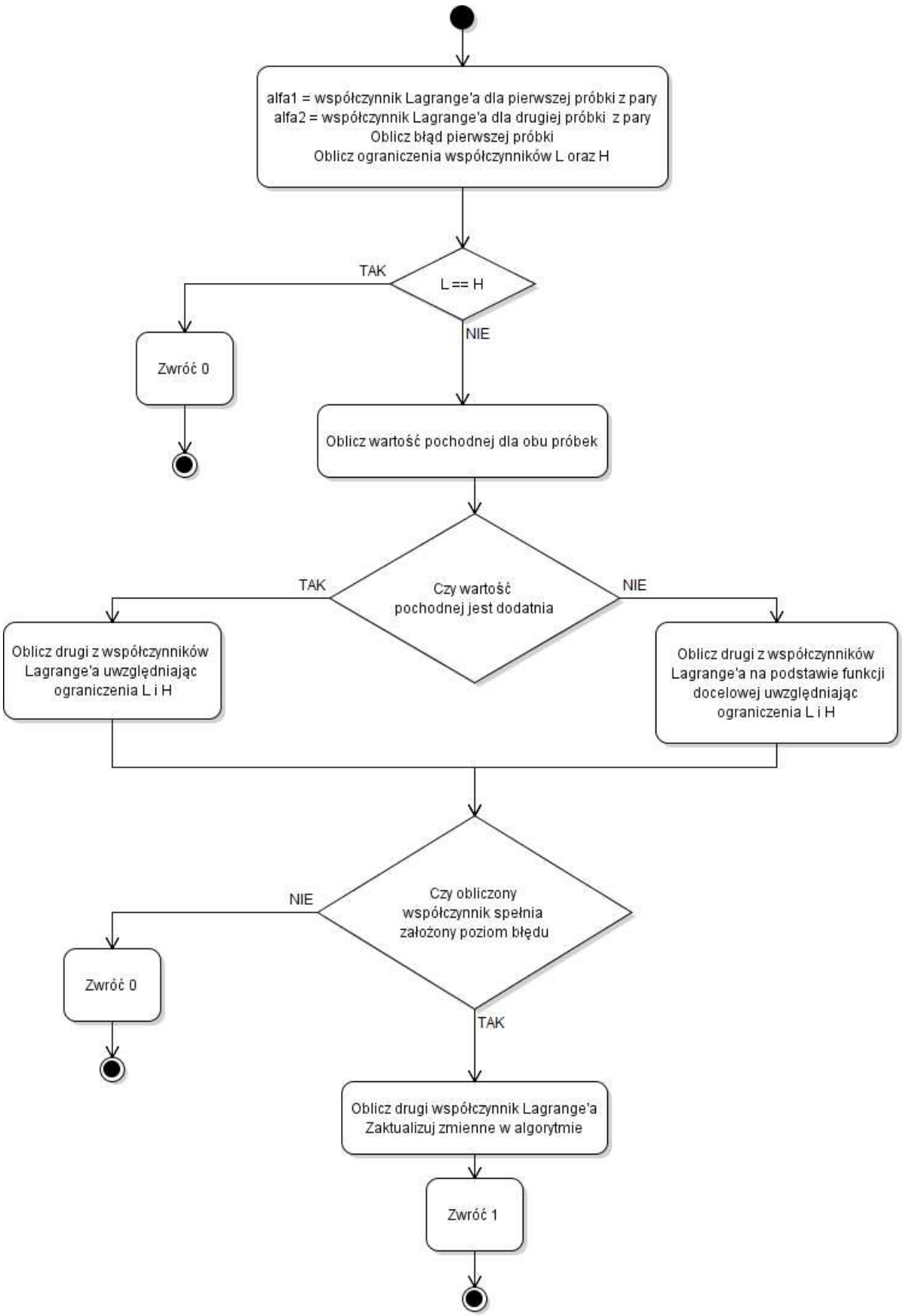
Sposób działania algorytmu przedstawiono za pomocą trzech schematów blokowych, które znajdują się poniżej:



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



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



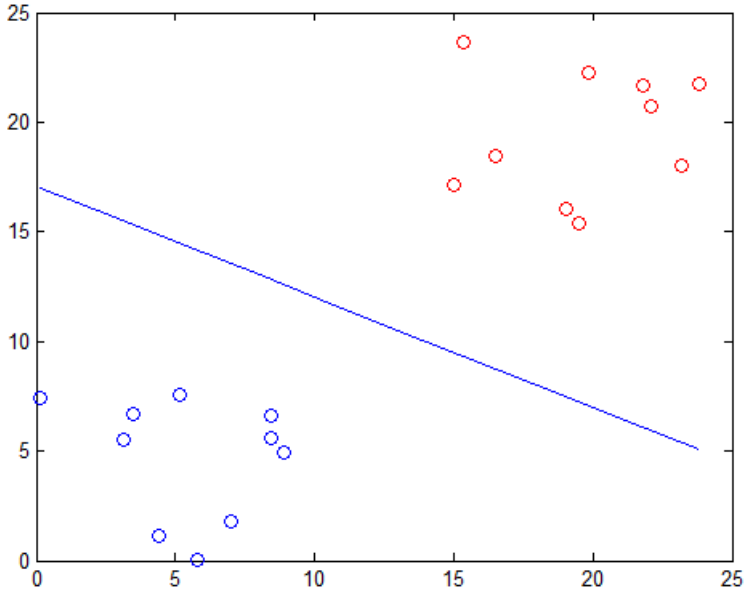
Rysunek 6: Schemat blokowy optymalizacji pary próbek
(opracowanie własne)

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*().

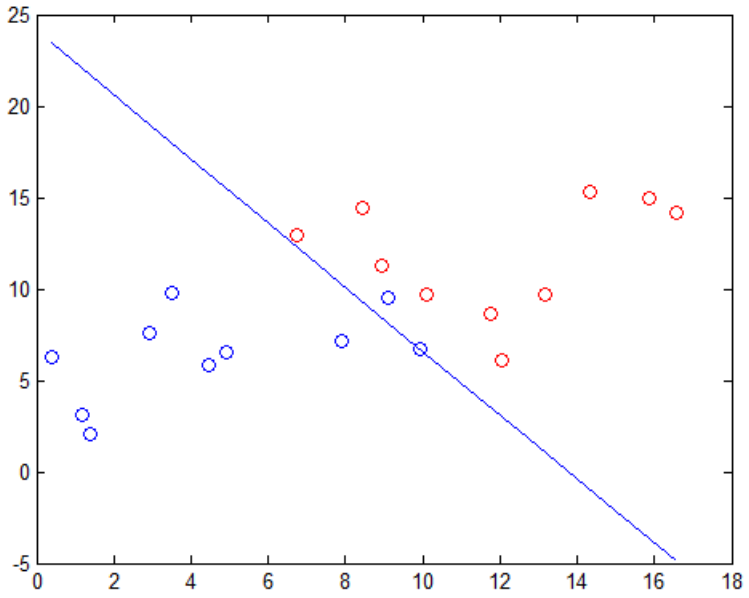
2.3.1 Wizualizacja dla dwóch cech

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)*

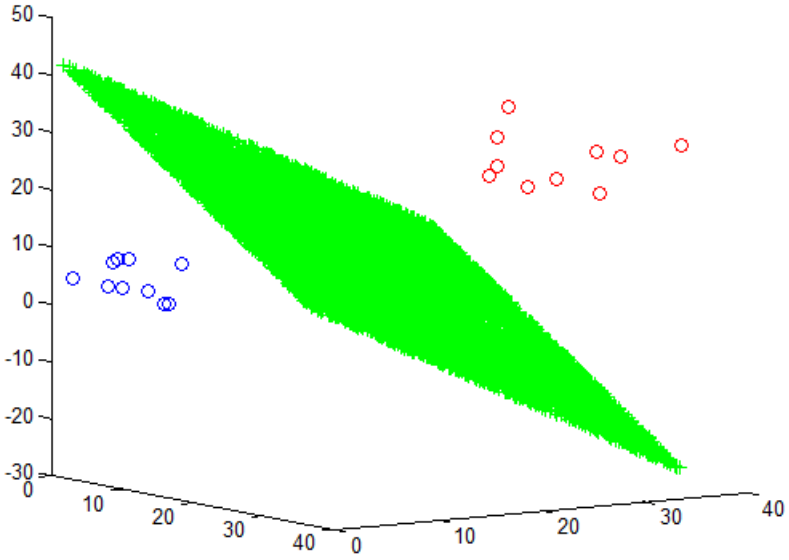
Na rysunku nr 5 przedstawiono dwa zbiory punktów należących do dwóch różnych klas, które zostały rozdzielone przy pomocy obliczonej hiperpłaszczyzny. W tym wypadku użyto dwóch cech, a przedstawione dane nie były liniowo separowalne.



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

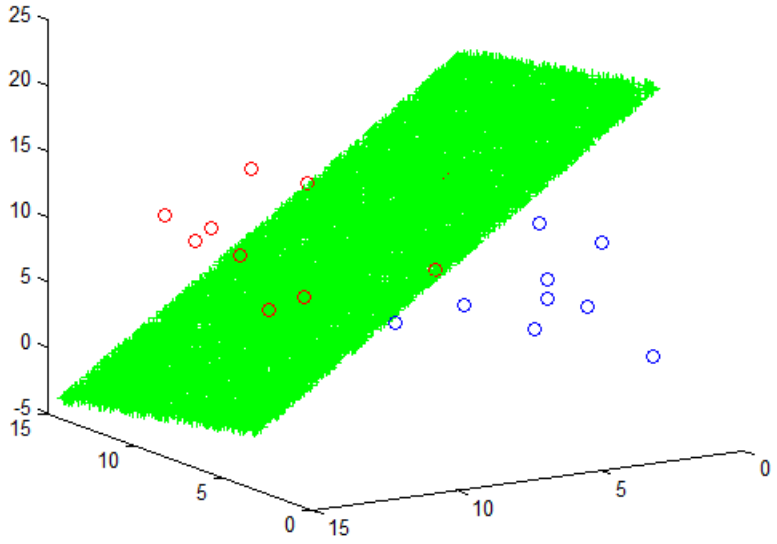
2.3.2 Wizualizacja dla trzech cech

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 nieseperowalnych 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 błędu 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 niepowdzenie 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 błędu 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 niepowdzenie 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 wczy-

tywania 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 zawierającą 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 nauczenie klasyfikatora
classificationTime - czas, którego algorytm potrzebował do klasyfikacji
accurancy - 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 *Understending 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 jakiejkolwiek 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 $X = (x_1, x_2, ..., x_n)$. Reguła Bayesa mówi, że obiekt X należy do klasy C_j, dla której wartość prawdopodobieństwa $P(C_i|X)$ jest największa. $P(C|X)$ to prawdopodobieństwo, że obiekt X należy do klasy C. Aby oszacować prawdopodobieństwa a-posterori $P(C|X)$ należy skorzystać z twierdzenia Bayesa, które ma postać:

$$P(C|X) = P(X|C)P(C)/P(X) \tag{23}$$

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 Ci lub można założyć, że wszystkie klasy mają takie same prawdopodobieństwo.

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

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

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

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

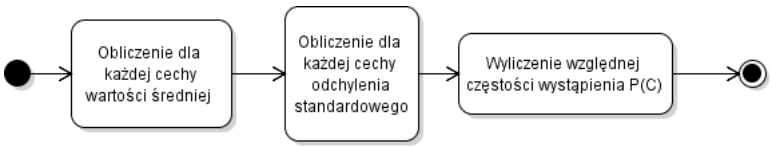
gdzie:
μ - średnia danego atrybutu w klasie,
σ^2 - wariancja atrybutu [1].

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.

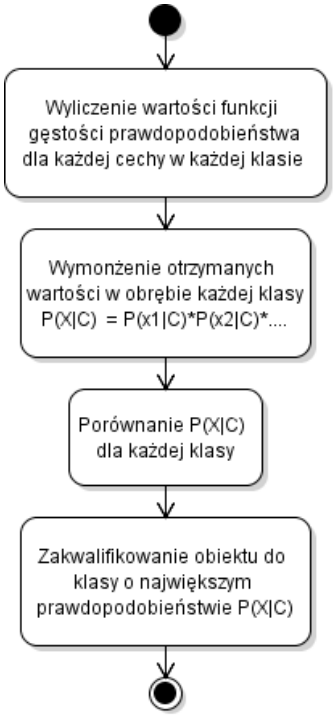
3.2 Implementacja

Patrząc pod kątem implementacji można wyróżnić dwa procesy: ucznia 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(X—Ci) otrzymane wartości wymnaża się. Obiekt zostanie zakwalifikowana do klasy, w której obliczone prawdopodobieństwo $P(X|C_i)$ jest najwyższe. Rysunek 12 przedstawia algorytm przypisana 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 klasyfikacje 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 literą *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 miedzy 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 interwal i post RR interval
- Ratio2 - stosunek per interwalu 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 jakią można było uzyskać z wykorzystaniem tych cechy 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 nadkomorowe 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 skuteczność rozpoznania utrzymuje się na wysokim poziomie. Klasyfikator jest bardzo prosty w implementacji klasyfikacja przebiega bardzo sprawnie. O skuteczności algorytmu nie decyduje ilość danych treningowych tylko ich reprezentatywność. Klasyfikator Bayesa jest tak skonstruowany, że jednie 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.

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 oraz wartością zmiennej objaśnianej
- Dany jest zbiór testowy wraz z wektorem zmiennych objaśniających dla którego prognozuje się wartość zmiennej objaśnianej

Proces uczenia się klasyfikatora k-NN polega po prostu na doborze parametru k. Zbiór uczący i testowy to wektory cech wyodrębnionych na drodze przetwarzania sygnału EKG. Na podstawie zbioru testowego klasyfikator uczy się przyporządkowywania cech do odpowiednich klas, a już podczas analizy zbioru testowego klasyfikator samodzielnie przyporządkowuje cechy do odpowiedniej klasy. Proces klasyfikacji obrazuje Rysunek 3.

Rysunek 13: Proces klasyfikowania zespołu QRS do danej klasy

4.2 Opis metody [1]

Klasyfikator k-Najbliższych Sąsiadów to nieparametryczna metoda klasyfikacji oznaczana jako k-NN (*k-Nearest Neighbours*). Rozpoznawany obiekt zalicza się do tej klasy, do której należy większość z jego K najbliższych sąsiadów. Obiekty są analizowane w taki sposób, że oblicza się odległości pomiędzy nimi. Istnieją różne miary podobieństwa - w przypadku niniejszego projektu posłużono się *odległością euklidesową*:

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

Klasyfikator ten odnajduje K najbliższych sąsiadów bieżącej próbki i zlicza próbki przypisane do poszczególnych klas. Bieżąca próbka zostaje przydzielona do klasy, która występuje najczęściej wśród swoich najbliższych K sąsiadów. Odległość od danych próbek na tym etapie nie ma już znaczenia, chyba że liczba sąsiadów z poszczególnych klas będzie identyczna.

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. 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. Powoduje to, że klasyfikator jest tym wolniejszy im obszerniejszy jest zbiór uczący.

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

Podczas implementacji algorytmu w środowisku C++ do testowania algorytmu użyto danych, które klasyfikowały próbki do dwóch klas:

- 1 - uderzenie nadkomorowe (prawidłowe)
- -1 - uderzenie komorowe (nieprawidłowe)

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 została 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
```

4.5 Walidacja algorytmu

Algorytm przetestowano na zbiorze 100 próbek testowych. Wynik walidacji wykazał, że 98 probek ze 100 zostało sklasyfikowanych poprawnie. Pokazuje to, że algorytm klasyfikuje skutecznie, jednak dalsze testy powinny być przeprowadzone na dużo większym zbiorze testowym.

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 [1]

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.

5.2 Opis metody eNN [2] [3]

Algorytm ENN (*extended nearest neighbor*) w uproszczeniu używa klasyfikatora KNN do znalezienia k-najbliższych sąsiadów próbki testowej w zbiorze treningowym, a następnie na podstawie obliczenia średniej ważonej kolejnych sąsiadów wybiera tę klasę, dla której suma średnich ważonych do niej przypisanych wśród najbliższych 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 - pozwala to 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 eNN. 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.

Średnia ważona i-sąsiada w zborze 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:

$$class = argmax(cWSc)$$

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 porownać. Na początku znormalizowano dane, jednak i bez tej operacji algorytm działa poprawnie.

Zaimplementowaną funkcję liczącą skuteczność algorytmu. Algorytm jest przygotowany tak, aby działał dla różnych zbiorów danych. 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 klasyfikuję 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
```

5.4 Podsumowanie

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.

Przeprowadzono test działania programu. Sprawdzono czas realizacji algorytmu oraz skuteczność właściwego klasyfikowania próbek wyrażoną w procentach. Na zastosowanym zbiorze testowym nie wykryto różnicy w skuteczności dla algorytmu KNN oraz ENN - oba poprawnie zaklasyfikowały 98 próbek ze 100. Był to zbiór testowy wybrany przez całą grupę dla porównania wyników uzyskanych w obrębie poszczególnych zespołów. Oczekiwanym rezultatem była większa skuteczność algorytmu ENN, ponieważ uwzględnia on dalsze sąsiedztwo (a co za tym idzie więcej przypadków). Na innym zestawie danych (z bazy danych physionet) rzeczywiście te oczekiwania się potwierdzały. Jednak algorytm ENN ma dużo większą złożoność obliczeniową niż KNN (ponieważ kilkukrotnie wykorzystuje algorytm KNN do własnych obliczeń).

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 Porównanie algorytmów

Tabela 3: Skuteczność klasyfikacji - dane Piotra

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

Tabela 4: Czasy uczenia i klasyfikacji - dane Piotra

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

Tabela 5: Skuteczność klasyfikacji - dane Iwony

kNN	ENN	Linear SVM	SVM + RBF	Naive Bayes	LDA
99%	99%	99.7261%	92.8789%	99.8783%	-

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.

Tabela 6: Czasy uczenia i klasyfikacji - dane Iwony

	kNN	ENN	Linear SVM	SVM + RBF	Naive Bayes	LDA
Czas uczenia	-	-	25003 ms	7865 ms	21.9497 ms	-
Czas klasyfikacji	3563 ms	36328 ms	26 ms	2920 ms	1925.88 ms	-