Quadratic SVM

Katarzyna Krzywonos
Aleksandra Lubzińska

12 luty 2017

Klasyfikacja uderzeń serca klasyfikatorem Quadratic SVM

Spis treści

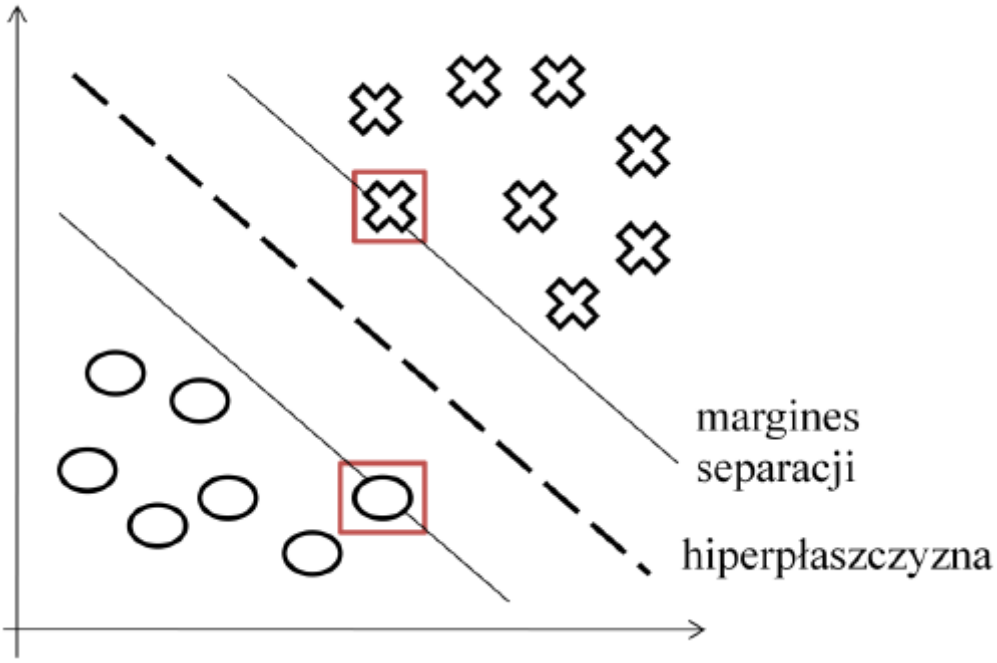
Cel projektu

Celem projektu była klasyfikacja rytmu serca przy użyciu Maszyny wektorów nośnych (ang. Support Vector Machine). Do realizacji tego zadania użyto algorytmu SMO (ang. Sequential Minimal Optimization). Algorytm tez został opracowany przez Johna Platt w 1998 roku i oparty jest na QP (ang. Quadratic Programming).

1 Wstęp teporetyczny

1.1 SVM

SVM (ang. support vector machine), czyli maszyna wektorów nośnych (wspierających), zaliczana jest do technik nadzorowanych, które analizują dane i rozpoznają ich wzorce. Ogólnym celem SVM jest określenie przynależności danego zbioru danych do jednej z dwóch klas przez wyznaczenie granicy między dwoma zbiorami z największym możliwym marginesem.



Rysunek 1:
SVM

Do stworzenia takiej hiperpłaszczyzny brane są pod uwagę tylko niektóre wektory treningowe, czyli wektory nośne, inaczej wektory podtrzymujące (ang. support vectors), określające optymalny margines separacji.

1.2 Quadratic SVM (ang. quadratic kernel-free non-linear support vector machine)

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

1.2.1 Kernel

Klasy $y = \pm 1$ są kojarzone z wzorami x poprzez transformację wzorów do wektora cech $\Phi(x)$
$\widehat{y}(x) = w * \Phi + b$
Parametry w i b są wybierane na podstawie uruchomienia algorytmu na zestawie danych uczących $(x_1, x_2)....(x_i, x_j)$
Paramter Φ jest dobierany indywidualnie do każdego przypadku
$w = \sum_{i=1}^{n} \alpha_i * \Phi_i$
$\widehat{y}(x) = \sum_{i=1}^{n} \alpha_i * K(x, x_i) + b$
Funkcja Kernel K(x,y) rezprezentuje iloczyn skalarny $\Phi(x)'\Phi(y)$. To wyrażenie jest przydatne zwłaszcza wtedy, gdy duża wartość współczynników α_i jest równa 0. Wówczas te współczynniki, które są rózne od zera nazywane są wektorami wspierającymi.

1.2.2 Quadratic Programming

QP opisuje problem optymalizacji (min lub max) kwadratową funkcją wielu zmiennych podlegających liniowym ograniczeniom. Wypukłość funkcji jest wykorzystywana do optymalizacji, ponieważ ma tylko jedno optimum, które jest globalnym optimum.

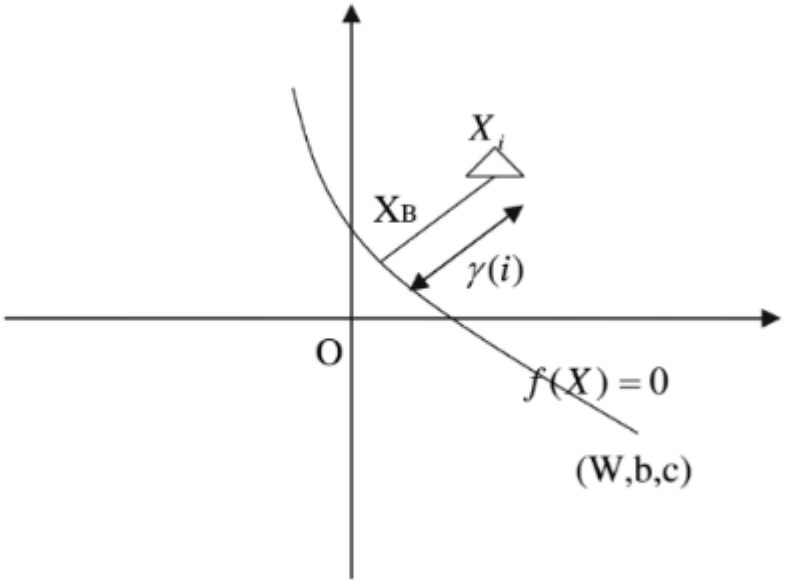
$QSVM(W, b, c)$ $f(X) = \frac{1}{2} * X^T * W * X + b^T + c$ $f(X) = ct$ może przyjąć wszystkie formy hiperpłaszczyzny (hiper-elipsy, hiper-parabole..) f(X) może być rozważana jako suma $fliniowe + fnieliniowa$

1.3 SMO

Minimalna optymalizacja sekwencyjna SMO dekomponuje problem SVM na podproblemy, które są rozwiązywane analitycznie, lecz podproblemy są tylko dwuparametrowe.

1.4 Opis ogólny procedury

Dekompozycja – wyróżnenie 2 podzbiorów parametrów: zbiór aktywny i zbiór pasywny. W każdym kroku iteracyjnym jedynie parametry aktywne są optymalizowane. Ogólną 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.



Rysunek 2:
SMO

Kroki jakię są wykonywane w heurestyce

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 zadane punkty. Gdy $\alpha_i > 0$ to i-ty wektor jest wektorem wspierającym. Gdy $\alpha_i = 0$ to i-ty wektor może być wektorem wspierającym.
Algorytm SMO poszukuje w takim kierunku, w którym wszystkie współczynnki α są równe z 0 z wyjątkiem dla dwóch pojedynczych wartości 1 i -1.

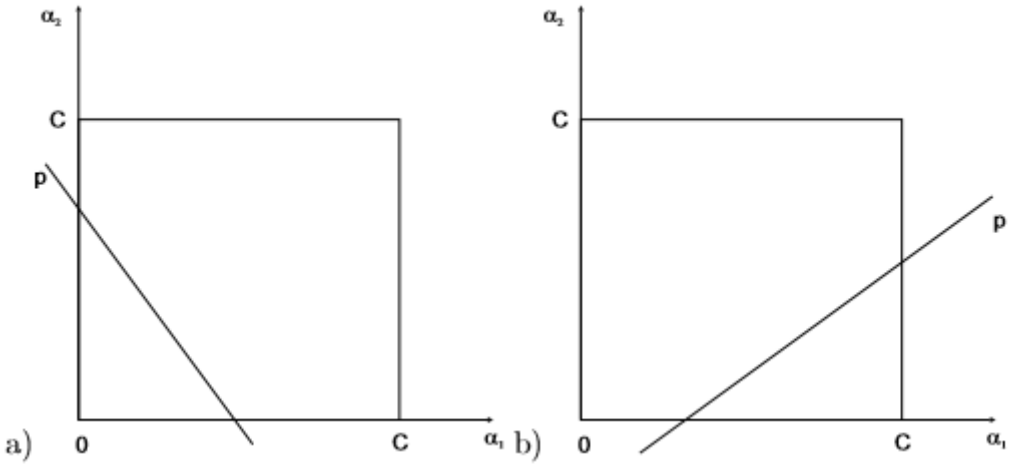
1.5 Rozwiązanie analityczne dla dwóch punktów

1. Dwoma wybranymi parametrami do zbioru aktywnego są parametry α_1 i α_2. Kolejne kroki w algorytmie zaproponowanych przez Platta są nastę-

pujące:

- α_1 wybierany jest spośrod wszytkich paratemtrów, które nie spełniają KKT (warunki konieczne I rzędu), przy czym pierwszęnstwo mają α_1 spełniające $0 < \alpha_1 < C$. Parametr α_1 jest to mnożnik Lagrange'a.
- α_2 (drugi mnożnik Lagrange'a) musi spełniać warunek.
- SMO optymalizuje parę (α_1, α_2) i powtarza to, aż do momentu zbieżności.

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 $\alpha_1 * y_1 + \alpha_2 * y_2 = \alpha_1^{old} * y_1 + \alpha_2^{old} * y_2$
   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 $\alpha_1 = C$



Rysunek 3:
Rysunek przedstawia prostą p w przypadku a)o współczynniku ujemnym, w przypadku b) dodatnim.

Po podstawieniu otrzymuje się (gdy współczynnik kierunkowy jest ujemny (Rysunek 3. a))
$\alpha_2 = \alpha_1^{old} + \alpha_2^{old}$ $\alpha_2 = \alpha_1^{old} + \alpha_2^{old} - C$
Gdy współczynnik kierunkowy jest dodatni (Rysunek 3. b))
$\alpha_2 = -\alpha_1^{old} + \alpha_2^{old}$ $\alpha_2 = \alpha_1^{old} + \alpha_2^{old} - C$
Należy pamiętać, że punkty przecięcia muszą leżeć w obrębie kwadratu.

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

5. Po wykonaniu i obliczeniu parametrów α ważne jest by odpowiednio ograniczyć wyniki
   Dla ułatwienia zostanie wprowadzony parametr $s = y_1 * y_2$ oraz y=1 lub y=-1
   $y_1 * \alpha_1 + y_1 * \alpha_1 = constant = \alpha_1 + s * \alpha_2$
   $\alpha_1 = \gamma - s * \alpha_2$

   Dla przypadku $s = 1$
   $\alpha_1 + \alpha_2 = \gamma$

   - $\gamma > C$ max $\alpha_2 = C$, min $\alpha_2 = \gamma - C$
   - $\gamma < C$ max $\alpha_2 = 0$,min $\alpha_2 = \gamma$

   Dla przypadku $s = -1$
   $\alpha_1 - \alpha_2 = \gamma$

   - $\gamma > 0$ min $\alpha_2 = 0$,max $\alpha_2 = C - \gamma$
   - $\gamma < 0$ min $\alpha_2 = -\gamma$,max $\alpha_2 = C$

   W momencie gdy $y_1 \neq y_2$

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

   W momencie gdy $y_1 = y_2$

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

6. Wyliczenie E_i Jest to błąd, który pojawia się w momencie gdy chce się zobaczyć różnice między wyjściem a celem
   $E_i = \sum_{j=1}^{l}(\alpha_i) * y_j * K(x_j, x_i) - y_i$

7. Wyliczenie k_ij
   $\kappa = K(x_1, x_2) + K(x_2, x_2) - 2 * K(x_1, x_2)$

8. Wyliczenie nowych parametrow α
   $\alpha_2^{unc} = \alpha_2^{old} + \frac{y_2 * (E_1 - E_2)}{\kappa}$
   $\alpha_2 =$

   - V, jesli $\alpha_2^{new}unc > V$;
   - $\alpha_2^{new}unc$, jeśli $U \geqslant \alpha_2^{new}unc \geqslant V$;
   - U, jeśli $U < \alpha_2^{new}unc$;

   $\alpha_{new} = \alpha_{old} + y_1 * y_1(\alpha_{old} - \alpha_{new})$

2 Implementacja rozwiązania C++

2.1 Zaproponowane rozwiązanie

Patrząc pod kątem implementacji maszyny wektorów nośnych z jądrem kwadratowym metodą SMO, należy wyróżnić dwa etapy klasyfikatora : uczenie oraz predykcję/klasyfikację. W głównym programie pierwsze tworzony jest obiekt klasyfikatora QSVM, następnie jest on uczony przez wywołanie metody train(), a dopiero później następuje predykcja na próbkach testowych (poniższy pseudokod).

```
#include "svm_additional_fnc.h"
#include "svm_classifier.h"
using std::cerr;
using std::endl;
// główna funkcja main
int main()
{
SVMClassifier *our_classifier = new SVMClassifier(); // tworzenie nowego obiektu klasy SVMClassifier
clock_t begin = clock(); // zapisanie czasu rozpoczęcia treningu w zmiennej begin typy clock_t
our_classifier->train(); // wywołanie metody train na obiekcie
clock_t train = clock(); //zapisanie czasu końca treningu w zmiennej train typu clock_t
our_classifier->predict();// wywołanie metody predict na obiekcie
clock_t predict = clock(); //zapisanie czasu końca predykcji do zmiennej predict typy clock_t
std::cout << "Train time:\t" << (double(train - begin) / CLOCKS_PER_SEC) << endl;
std::cout << "Predict time:\t" << (double(predict - train) / CLOCKS_PER_SEC) << endl;
std::cout << "Total time:\t" << (double(predict - begin) / CLOCKS_PER_SEC) << endl;
int wait;
std::cin >> wait; // zamknięcie przez wpisanie liczby
}
```

Pierwsze w funkcji main wywoływana jest funkcja train. Poniżej przedstawiony jest pseudokod funkcji train().

Pseudokod funkcja train():
numChangedAlpha = 0;
prevNumChangedAlpha = 0;
examineAll = 1;
rounds = 0;
sameRounds = 0;
while numChangedAlpha większe od 0 lub examineAll równe True, oraz rounds < maksymalnej liczby iteracji, oraz sameRounds < maksymalnej liczby iteracji z tym samym błędem
  do
    inkrementacja rounds;
    if examineAll równa się true to
      for dla każdej próbki treningowej
        numChangedAlpha += examineExample(k);
    else
      for pętla dla każdej próbki gdzie alpha nie jest równa 0 oraz nie jest równa C
        numChangedAlpha += examineExample(k);
    if examineAll równa się 1
      examineAll równe 0
    else if numChangedAlpha równe 0
      examineAll równe 1

W funkcji train() w każdej iteracji wywoływana jest funkcja examine-Example.

Pseudokod ExamineExample(i1):

```
definicja zmiennych y1, alpha1, e1, r1, typu float jako 0.0,
y1 -> wartość etykiety klasy o indeksie i1 z arrayY[]
alpha1 -> mnożnik Lagrange'a o indeksie i1 z alpha[]
if alpha[i1] zawiera się w zakresie (0,C)
  e1 -> wartość o indeksie i1 z wektora arrayError
else
  e1 -> wartość wyjściowa SVM dla punktu i1 - y1
r1 -> e1 * y1
if (r1 mniejsze od -TOLERANCE oraz alpha[i1] mniejsza od C) lub (r1
większe od TOLERANCE oraz alpha2>0)
  for pętla przechodząca N razy
    if kolejny mnożnik zawiera się w zakresie (0,C)
      szukanie maksymalnej wartości bezwzględnej (e1-e2), i2 -k,
    if i2 większe lub równe 0
      if wywołanie funkcji takeStep dla i1, i2 zwróci 1
        return 1
  for pętla przez punkty niebrzegowe zaczynając od dowolnej liczby k0
    poprzednio wyznaczone i2 zmieniamy na k0 % N
    if mnożnik[i2] zawiera się w zakresie (0,C)
      if wywołanie funkcji takeStep dla i1 oraz i2 zwróci 1
        return 1
  for pętla przez wszytskie możliwe punkty i2, zaczynając w losowym punk-
cie
    i2 -> zmienna pętli
      if wywołanie funkcji takeStep dla i1 oraz i2 zwróci 1
        return 1
```

Poniżej przedstawiono pseudokod takeStep(i1,i2):

```
if i1 równe i2
   return 0
alpha1 -> alpha[i1]
alpha2 -> alpha[i2]
y1 -> arrayY[i1]
y2 -> arrayY[i2]
e1 -> wynik SVM dla punktu[i1] - y1
e1 -> wynik SVM dla punktu[i2] - y2
s -> y1 * y2;
L -> 0.0;
H -> 0.0;
if y1 == y2
    L -> maksimum w zakresie (0, alpha1 +alpha2 - C)
    H -> maksimum w zakresie (C, alpha1 +alpha2)
else
   L -> maksimum w zakresie (0, alpha2 +alpha1)
   H -> maksimum w zakresie (C,C - alpha1 + alpha2)
if wartość bezwzględna różnicy między L oraz H mieści się w zakresie tole-
rancji
   return 0
k11 -> wynik funkcji kernel(i1, i1)
k12 -> wynik funkcji kernel(i1, i2)
k22 -> wynik funkcji kernel(i2, i2);
eta = 2 * k12 - k11 - k22;
if eta mniejsza od 0
   a2 = alpha2 + y2 * (e2 - e1) / eta;
   if a2 mniejsze od L
      a2 równe L
   else if a2 > H
      a2 = H
else
   Lobj = objective function at a2=L
    Hobj = objective function at a2 = H
   if Lobj większe od Hobj - eps
      a2 = L
   else if Lobj mniejsze od Hobj-eps
      a2 = H
   else
      a2 = alpha2
if (fabs(a2 - alpha2) < epsilon * (a2 + alpha2 + epsilon))
   return 0
a1 = alpha1 + s * (alpha2 - a2);
aktualizacja wartości b, arrayError
umieszczenie a1 oraz a2 w alphaArray
return 1
```

Po zakończeniu czasu treningowego, zwracany jest jego czas, a następnie wywoływana jest funkcja predykcyjna na próbkach testowych. Następnie na obiekcie klasyfikatora wywoływano funkcję predykcyjną predict(),

wyznaczająca przynależność próbek testowych do konkretnej klasy oraz podstawowe statystyki skuteczności klasyfikacji i czas klasyfikacji.

2.2 Dokumentacja

class SVMClassifier
Implementacja rozwiązania w języku C++ wymagała zastosowania programowania obiektowego, więc stworzono klasę SVMClassifier.

Zmienne protected klasy SVMClassifier:
double C - wartość typu double
double epsilon - wartość typu double epsilon ustawiona na 0.0001
char fNameTrain[256] - zmienna typu char, nazwa pliku z próbkami treningowymi, maksymalna długość 256,
char fNameTest[256] - zmienna typu char, nazwa pliku z próbkami testowymi, maksymalna długość 256
char fNameResults[256] - zmienna typu char, nazwa pliku z rezultatami, maksymalna długość 256
int N - zmienna typu int, liczba próbek treningowych, początkowa wartość równa 0,
int NTestSamples - zmienna typu int, liczba próbek testowych, początkowa wartość równa 0,
TVectorArray arrayX - wektor wektorów par obiektów typu int i float, zawiera próbki treningowe,
TFloatArray arrayY - wektor wektorów par obiektów typu int i float, zawiera próbki testowe
TFloatArray alpha - wektor z wartosciami alpha, poczatkowa inicjalizacja zerami
TFloatArray d - macierz pomocnicza w obliczeniach
TFloatArray arrayError - macierz błedów
float b - wartość b
float bDiff - delta b

Metody klasy SVMClassifier:
SVMClassifier() - konstruktor, tworzy obiekt SVMClassifier
SVMClassifier() - destruktor
int train() - funkcja ucząca klasyfikatora,implementuje główna procedure uczenia minimalną optymalizacja sekwencyjna, zapisuje rezultaty do pliku
int examineExample(int i1) - funkcja która przy wywołaniu otrzymuje indeks alphy, sprawdza warunki KKT(herustyka) oraz wyszukuje wartosci alpha2, następnie wywołuje funkcje 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

2.3 Another functions

typedef std::string TString
typedef std::vector$\langle std :: string \rangle$ **TStringArray** - wektor obiektów typu string,
typedef std::pair $\langle int, float \rangle$ **TVectorDim** - pary obiektów typu int oraz float
typedef std::vector $\langle TVectorDim \rangle$ **TVector** - wektor par obiektów typu int oraz float,
typedef std::vector $\langle TVector \rangle$ **TVectorArray** - wektor wektorów par obiektów typu int oraz float,
int splitCSV(const TString& s, char c, TStringArray& v) - funkcja rozdzielająca plik .csv
int readSample(TString& s, TVector& x, float& y)- funkcja czytająca pojedyńczą próbkę z pliku csv
int writeSample(TString& s, TVector& x, float& y) - funkcja zapisującą próbkę do macierzy z cechami oraz z etykietami klasy
int partReadSample(std::ifstream& is, TVectorArray& arrayX, TFloatArray& arrayY, int& n) - funkcja czytająca cały plik z próbkami
int partWriteSample(std::ofstream& os, TVectorArray& arrayX, TFloatArray& arrayY, int& n) - funkcja zapisująca do pliku
float dotProduct(const TVector& v1, const TVector& v2) - funkcja obliczająca iloczyn skalarny
TVector operator*(const TVector& v, float f) - przeciążenie operatora
TVector operator*(float f, const TVector& v)- zwraca wektor v*f
TVector operator+(const TVector& v1, const TVector& v2) - przeciążenie operatora +

2.4 Aplikacja

Aplikacja „Quadratic Support Vector Machine" (QSVM) napisana jest w języku C++.

2.4.1 Wymagania techniczne

W celu zapewnienia poprawności działania aplikacji należy spełnić poniższe kryteria :

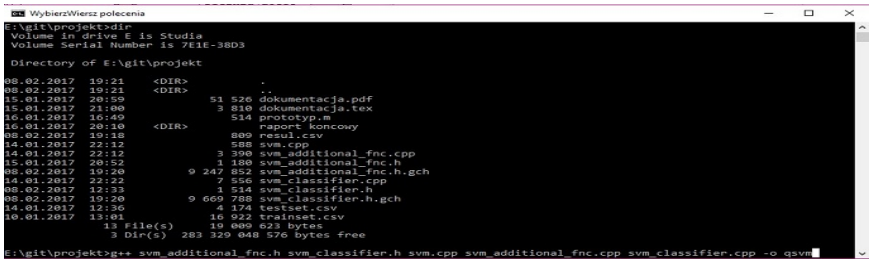
- komputer PC,

- system operacyjny Microsoft Windows (program tworzony na Microsoft Windows 10),

- zainstalowany kompilator g++ (C++) np. pakiet MinGW z GCC.

2.4.2 Uruchamianie aplikacji

Program jest kompilowanych w kompilatorze g++ , jednak w tym celu konieczne jest wypisanie w komendzie wszystkich plików nagłówkowych i źródłowych oraz nazwy pliku wyjściowego qsvm (w przypadku pominięcia zostanie utworzony domyślny plik wyjściowy a.exe). Zastosowanie komendy pokazanej na poniższym obrazku skutkuje utworzeniem pliku wykonywalnego qsvm.exe, który nie wymagania instalowania ani konfigurowania (skompilowana wersja programu została również umieszczona w repozytorium projektu).



Rysunek 4:
Przykład wywoływania programu w kompilatorze g++ na systemie operacyjnym Microsoft Windows 10 (opracowanie własne).

Uruchomienie programu qsvm.exe rozpoczyna on działanie na próbkach trainset.csv oraz testset.csv i wyświetla on numer iteracji programu oraz Error rate podczas poszczególnych iteracji (poniższy rysunek).

Rysunek 5:
Wyświetlenie numeru iteracji oraz Error rate (opracowanie własne).

Efekt klasyfikacji wyświetlany po zakończeniu programu – etykiety próbek dla testset.csv oraz dokładność klasyfikacji i czasy treningu oraz predykcji (Rysunek 6).

Rysunek 6:
Dane zwracane przez program po zakończeniu klasyfikacji (opracowanie własne).

3 Badania eksperymentalne

W tej części przedstawimy rezulataty badań eksperymentalnych nad QSVM metodą SMO.

3.1 Zbiór Iris

Do wstępnego określenia skuteczności klasyfikacji zaimplementowanego rozwiązania zastosowano zbiór próbek „iris" opisujący 150 przykładów należących do jednej z trzech klas będących różnymi typami rośliny irysów. Każdy typ irysa reprezentowany jest przez 50 przykładów opisanych przez 4 atrybuty określające wymiary składowych kwiatów.

Pierwotna wersja klasyfikatora SVM odnosi się do dwóch klas. Istnieją metody pozwalające osiągnąć wieloklasową maszynę wektorów nośnych, jednak jest to związane z użyciem wielokrotnej klasyfikacji. Występują dwa podejścia rozwiązania problemu wieloklasowości maszyny wektorów nośnych:

- *one against all*- zakłada stworzenie liczby klasyfikatorów równą liczbie klas. Każdy klasyfikator ma zbiór uczący podzielony na dwie części. Jedna część składa się ze zbioru uczącego jednej klasy, a druga złożona jest z pozostałych klas. Konkretna analizowana próbka poddawana jest klasyfikacji we wszystkich klasyfikatorach. Za wynik uważana jest klasa zwracająca maksymalny wynik[6].

- *one against one* - polega na stworzeniu wszystkich N możliwych kombinacji dwuelementowych wszystkich k klas, czyli . Dane uczące każdego klasyfikatora zawierają dwie klasy. Wynikiem tego algorytmu, czyli określeniem przynależności do klasy jest największa liczba klasyfikacji do danej klasy ze wszystkich klasyfikatorów [7].

W tym przypadku wykorzystano metodę *one against one*, ze wszystkich próbek stworzono po 3 zbiory uczące (80 próbek) oraz testowe (20 próbek). Poniżej przedstawiono osiągnięte wyniki.

Tabela 1. Wyniki klasyfikacji dla zbioru Iris

Zbiór próbek	Osiągnięte wyniki
Setosa + versicolor	Accuracy: 100% (20/20) Train time: 0.059 Predict time: 0.037 Total time: 0.096
Versicolor + virginica	Accuracy: 100% (20/20) Train time: 1.728 Predict time: 0.016 Total time: 1.744
Setosa + virginica	Accuracy: 100% (20/20) Train time: 0.009 Predict time: 0.019 Total time: 0.028

Na podstawie uzyskanych wyników można stwierdzić, że klasyfikator działa prawidłowo, ma on 100 % skuteczności. Łatwo jest również zauważyć, iż dla zestawu próbek versicolor + virginica czas train time jest najdłuższy. Tym samymm predict time jest najkrótszy co świadczy o tym, iż próbki zostały w odpowiedni sposób wytrenowane i dzięki temu klasyfikacja ich jest szybka. Dla zestawu próbek setosa + versicolor czas predict time jest najdłuższy. Moża stwierdzić, że algorytm w sposób najszybszy potraktował zestaw próbek setosa+virginica.

3.2 Wykorzystanie klasyfikatora do rozpoznawania uderzeń serca

Każdy z klasyfikatorów wykonanych w ramach projektów grupy C testowane były przy pomocy dwóch określonych zbiorów danych, zawierających w sobie cechy opisujące uderzenia serca, w celu uwiarygodnienia porównania między sobą klasyfikatorów. W obu grupach uderzenia zostały podzielone na dwie klasy: uderzenia V, określające uderzenia wygenerowane w obrębie komory, zaliczane do patologicznych oraz uderzenia N -> reprezentujące normalna, poprawna prace serca.

3.2.1 Zbiór danych 1

Pierwszy zbiór danych składał się z pięciu cech opisujących uderzenia serca, bazując na interwałach RR (kolejne szczyty zespołów QRS) . Do wyznaczenia zbioru próbek wykorzystano sygnały znajdujące się w bazie danych MIT-BIH (Massachussetts Institute of Technology – Beth Israel Hospital) – standardowa arytmiczna baza danych wykorzystano sygnały 228.dat oraz 100.dat .

Wykorzystane cechy :

(a) **Pre interval** RR - odległość miedzy aktualnym uderzeniem serca(analizowanym), a poprzednim uderzeniem serca,

(b) **Post inreval RR** - odległość miedzy analizowanym uderzenie serca a kolejnym uderzeniem,

(c) **Average RR** - średnia długość interwału RR dla całego analizowanego sygnału,

(d) **Ratio1** - stosunek pre RR interval do post RR interval,

(e) **Ratio2** - stosunek per interwalu do Avarage RR.

Uzyskane próbki podzielono na dwie części w stosunku 4:1, 400 próbek treningowych oraz 100 próbek testowych. Poniżej w tabeli przedstawiono uzyskane wyniki rozpoznania próbek odpowiednio do klas N oraz V.

Tabela 2. Tabela uzyskanych wyników dla zbioru danych 1

	Dobrze rozpoznane	Źle rozpoznane	Całość klasy
Uderzenia N	48	2	0,99
Uderzenia V	49	2	0,98
Skuteczność rozpoznawania [%]:			97

Proces uczenia trwał dla systemu QSVM 24780 ms, natomiast czas klasyfikacji 59 ms, długa ilość uczenia systemu spowodowana jest skomplikowanym algorytmem SMO i konieczności przejścia dużej liczby iteracji do zoptymalizowania dwóch punktów. Można uznać, że skuteczność klasyfikacji uzyskano na wysokim poziomie, 97 %.

Dla uzyskanych wyników również wyliczono czułość (ang. Sensitivity), wyliczona z poniższego wzoru:

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

TP (ang. Truepositive) – liczba poprawnie sklasyfikowanych przykładów wybranej klasy,

FN (ang. Falsenegative) – liczba błędnie sklasyfikowanych przykładów z tej klasy,

Czułość klasyfikacji wynosi 96 %. Wyznaczono również specyficzność klasyfikacji (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 ($ang.falsealarm$)

Specyficzność tej klasyfikacji wynosi również 96 %.

3.2.2 Zbiór danych 2

Drugi zbiór danych również reprezentowała 5 cech sygnału z uderzeniami serca. Próbki podzielone również były w tak samo liczne grupy treningowe i testową. W skład grupy testowej wchodziło 100 próbek, natomiast do treningowej wchodziło 400 próbek.

Zbiór danych numer 2 zawierał w sobie cechy :

(a) Stosunek pola do obwodu (współczynnik Malinowskiej)
$p_1 = \frac{\sum_{k=1}^{N} s[k]}{\sum_{k=2}^{N} s[k]-s[k-1]}$

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

(c) Procent próbek sygnału, które są ujemne
$p_3 = 100\% * \frac{\sum_{k=1}^{N} u[k]}{N}$

(d) Stosunek maksymalnej prędkości do maksymalnej amplitudy
$p_4 = \frac{\max_{k\in<3,N>} s[k]-s[k-2]+2s[k-1]}{|\max_{k\in<1,N>} s[k]-\min_{k\in<1,N>} s[k]|}$

(e) Stosunek liczby próbek sygnału, których prędkość przekracza 40% maksymalnej prędkości obserwowanej
$p_5 = \frac{\sum_{k=1}^{N} g[k]}{N}$

Tabela 3. Wynik klasyfikacji dla zbioru 2

	Dobrze rozpoznane	Źle rozpoznane	Całość klasy
Uderzenia N	76	0	1
Uderzenia V	22	2	0,96
Skuteczność klasyfikacji [%]			98

Dla klasyfikacji tego zbioru danych również wyznaczono wartość czułości i skuteczności klasyfikacji analogiczną metodą przedstawioną w ostatnim podrozdziale. Czułość w przypadku tego zbioru danych osiągnęła maksymalne 100 %, a specyficzność 91,6 %.

Accuracy: 98% (98/100)

Train time: 22.943

Predict time: 0.01

Total time: 22.953

Uczenie klasyfikatora zajęło 22.953 s, natomiast czas klasyfikacji 0.01 s, jest to znacznie niższa liczba prawdopodobnie spowodowana zastosowaniem innych cech charakteryzujących sygnał.

Różna ilość cech wykorzystanych do klasyfikacji.

3.2.3 Porównanie klasyfikacji z różna ilością cech

W celu określenia działania klasyfikacji przetestowano również testowanie klasyfikatora z różną ilością cech określających sygnał. Kolejno z 1, 2, 3, 4 cechami w próbkach uczących i testowych. Poniżej przedstawiono w tabeli uzyskane wyniki.

Tabela 4. Porównanie klasyfikacji dla róznej ilości cech

	1 cecha	**2 cechy**	**3 cechy**	**4 cechy**
Skuteczność rozpoznawania [%]	98 %	98 %	98 %	98 %
Czas uczenia [s]	43.1	10.427	17.613	25.13
Czas klasyfikacji [s]	0.009	0.011	0.009	0.011
Czułość	100	100	100	100
Specyficzność	91,6	91,6	91,6	91,6

Z powyższych uzyskanych wyników można zauważyć, że skuteczność klasyfikacji jest na tym samym poziomie bez względu na ilość zastosowanych cech. Również analizując uzyskane wyniki niepoprawnie zostały sklasyfikowane te same próbki w każdym przypadku. Jednak przypadki między sobą różnią się długością czasu klasyfikacji oraz ilością koniecznych iteracji do nauczenia się programu. Z rosnącą ilością cech opisujących sygnały czas klasyfikacji ma tendencję malejącą do 3 cech, a następnie dla 4 cech i 5 do około 22-25 s. Dla jednej cechy czas uczenia był najdłuższy zajął aż 43.1 sekundy. Najszybszym nauczeniem charakteryzował się zbiór danych z dwoma cechami oraz trzema. Oznacza to, ze dla klasyfikatora należy wybrać pośrednią liczbę cech (najlepiej zachowuje się przy niezbyt niskiej i niezbyt wysokiej liczbie próbek).

4 Porównanie wyników różnych klasyfikatorów

W tym rozdziale przedstawimy różnicę w wynikach uzyskanych dla wszytskich klasyfikatorów: k-Neraest Neighbours (kNN), Extended Nerest Neighbours (eNN), Radial Basis Function Kernel (RBF SVM), klasyfikator Bayesa, Linear SVM, Quadratic SVM.

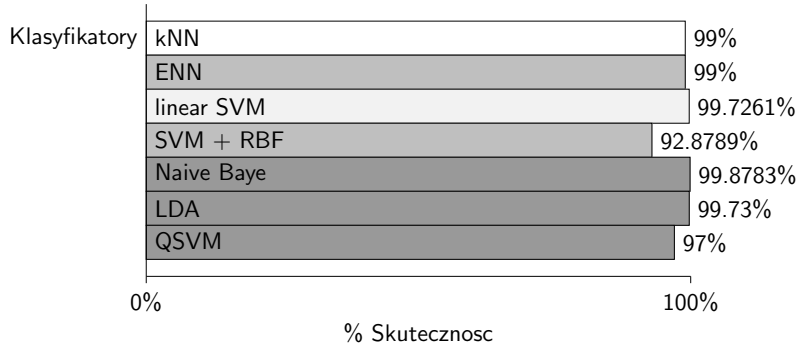
4.1 Zbiór danych 1

Poniżej przedstawiono wyniki uzyskane podczas klasyfikacji zbioru danych nr 1 przez wszystkie klasyfikatory. Wyznaczono takie parametry jak: skuteczność klasyfikacji, czas uczenia, czas klasyfikacji, czułość oraz specyficzność.

Tabel 6. Wyniki klasyfikacji wszystkich użytych klasyfikatorów dla zbioru 1

	kNN	**ENN**	**Linear SVM**	**SVM + RBF**	**Naive Baye**	**LDA**	**QSVM**
Skuteczność klasyfikacji [%]	99	99	99.7261	92.8789	99.8783	99.73	97
Czas uczenia [ms]	brak	brak	25003	7865	22.0936	3	24780
Czas klasyfikacji [ms]	3563	36328	26	1961	1925	13	59
Czułość	99	99	99.7129	92.5678	99.9681	98.68	96
Specyficzność	100	100	100	100	98.0132	95.83	96

Badany klasyfikator QSVM w porównaniu do innych badanych klasyfikatorów na tym zbiorze danych zajął 6 miejsce mimo osiągnięcia dosyć wysokiego wyniku poprawnego sklasyfikowania uderzenia serca, 97 procent. Z tabeli wynika, że QSVM zalicza się do klasyfikatorów długo uczących się, osiągnęło zbliżone wyniki do pozostałych padanych SVM z innymi jądrami, więc QSVM pod względem czasu uczenia jest jednym z gorszych klasyfikatorów. Jednak zaletą SVM jest szybki czas klasyfikacji, co też widać z powyższych danych, że QSVM jest trzeci najszybszy w klasyfikacji.



Wykres 1. Wykres procentowy przedstawiający skuteczność klasyfikacji wszyst-

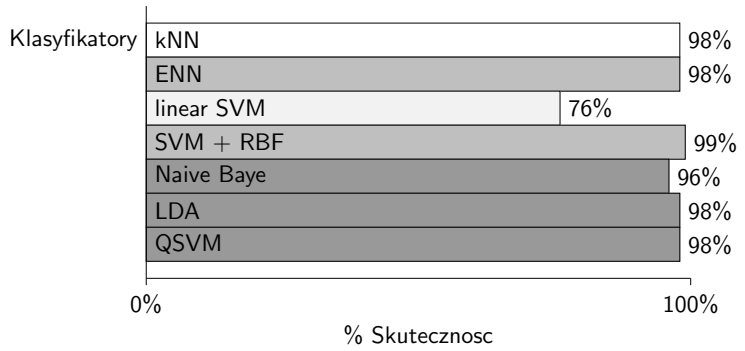
kich użytych klasyfikatorów dla zbioru 1

4.2 Zbiór danych 2

Tabel 5. Wyniki klasyfikacji wszystkich użytych klasyfikatorów dla zbioru 2

	kNN	**ENN**	**Linear SVM**	**SVM + RBF**	**Naive Baye**	**LDA**	**QSVM**
Skuteczność klasyfikacji [%]	98	98	76	99	96	98	98
Czas uczenia [ms]	brak	brak	8517	13481	21.9	15	22953
Czas klasyfikacji [ms]	109	1094	1	79	29.29	1	10
Czułość	97	100	100	100	97.36	99.78	100
Specyficzność	100	92	4	96	92	98.67	91.6

Biorąc pod uwagę skuteczność klasyfikacji dla zbioru 2 jedynie klasyfikator SVM + RBF wykazał lepszy wynik. QSVM razem z kNN, ENN i LDA rozpoznaje na poziomie 98. Jak już wspomniano czas uczenia dla tego klasyfikatora jest długi. Warto jednak tu podkreślić, że czas klasyfikacji dzięki dobremu wytreniowaniu zbioru jest stosunkowo krótki.



Wykres 2. Wykres procentowy przedstawiający skuteczność klasyfikacji wszystkich użytych klasyfikatorów dla zbioru 2

4.3 Wnioski

QSVM na tle innych użytych klasyfikatorów kNN, ENN, linear SVM, SVM + RBF, Naive Baye i LDA wykazał się stosunkowo dobrą skutecznościa. Dla zbioru danych 1 wynik ten jest znacznie gorszy w porównaniu do liniowego odpowiednika SVM, który to osiągnął skuteczność prawie stu procentową. Czas uczenia był porównywalny do QSVM jednak czas klasyfikacji był już krótszy. Warto tu podkreślić, że metoda SVM dobrze sobie radzi z nadmiernym dopasowaniem danych. Dla zbioru 2 skuteczność dla większości klasyfikatorów była porównywalna.
QSVM charakteryzuje stosunkowo dlugi czas uczenia. Związane jest to z faktem, iż metoda ta wymaga użycia Quadratic Programming co dla dużej ilości danych powoduje problem z szybkością działania algorytmu. W celu uniknięcia tego stosuję się opisaną na początku pracy metodę dekompozycji dzięki, której zbiór rozpatrywany jest w kilku etapach.

Literatura

[1] Platt John *Sequential Minimal Optimalization: A fast Algorithm for Ftaining Support Vector Machines* Microsoft Reasearch, 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 MinimalOptimalization for SVM* Iowa State University, Computer Science Department

[5] Juan Miguel *[SVM Matlab code implementation] SMO (Sequential Minimal Optimization) and Quadratic Programming explained* http://laid.delanover.com/svm-matlab-code-implementation-smo-sequential-minimal-optimization-and-quadratic-programming-explained/

[6] Marcin Kmieć *Wykrywanie niebezpiecznych przedmiotów w automatycznie analizowanych sekwencjach wideo* Akademia Górniczo-Hutnicza, Kraków

[7] Piotr Walendowski *Zastosowanie sieci neuronowych typu SVM do rozpoznawania mowy.* Politechnika Wrocławska, Wydział Elektroniki, Wrocław