1. SVM (ang. Support Vector Machine)

Opis ogólny

SVM (ang. Support Vector Machine, czyli Maszyny Wektorów Nośnych) jest to technika uczenia maszynowego, która jest używana do klasyfikacji i regresji. SVM wykorzystuje wspomniane wcześniej wektory do określenia granic decyzyjnych, które oddzielają dane w określonych grupach. SVM działa przez znalezienie optymalnego hiperpłaszczyzny, która najlepiej dzieli dane na dwie grupy. Jest to zadanie optymalizacji, które szuka najlepszego rozwiązania w celu zminimalizowania błędu.

SVM została stworzona w celu rozwiązywania problemów klasyfikacji i regresji, których nie można rozwiązać za pomocą innych algorytmów. Została ona również użyta do rozwiązywania problemów związanych z klasyfikacją tekstu i obrazu.

Historia powstania

Support Vector Machine (SVM) została pierwotnie opracowana w latach 90-tych przez dwóch naukowców, Bernharda Boser'a i Vladimira Vapnik'a. Ich celem było opracowanie algorytmu, który byłby w stanie lepiej rozwiązywać problemy klasyfikacji w porównaniu do istniejących już algorytmów. Ich koncepcja polegała na wykorzystaniu wektorów w celu określenia granic decyzyjnych, które oddzielają dane w określonych grupach. Ich rozwiązanie zostało opublikowane po raz pierwszy w 1995 roku i znane jest jako Support Vector Network (SVN). Po opublikowaniu w 1995 roku SVN, czyli poprzedniczki SVM, SVM szybko stał się bardzo popularny wśród naukowców i praktyków. W ciągu ostatnich kilku lat SVM znalazła szerokie zastosowanie w wielu dziedzinach, takich jak: medycyna, biologia, finanse, uczenie maszynowe, analiza danych i wiele innych.

Zadanie optymalizacji dla SVM

Wzór wykorzystywany do minimalizacji błędu w zadaniu SVM:

$$\lambda \cdot ||w||^2 + \left[\frac{1}{n}\sum_{i=1}^{n} max\big(0,1 - y_i(w^T x_i - b)\big)\right] \quad (1)$$

Gdzie:

$\lambda -$ jest parametrem regularyzacji, który kontroluje ilość regularyzacji, która jest wykorzystywana w celu minimalizacji błędu.

$||w||^2 -$ jest wzorem na normę euklidesową wektora w.

$w -$ wektor normalny do płaszczyzny, czyli taki, który wskazuje kierunek prostopadły do płaszczyzny.

$max\big(0,1 - y_i(w^T x_i - b)\big) -$ jest wzorem na funkcję kary, która jest wykorzystywana do minimalizacji błędu.

$y_i(w^T x_i - b) -$ jest wzorem na funkcję liniową, która jest wykorzystywana do określenia granic decyzyjnych.

$w^T x_i - b = 0$ – to równanie hiperpłaszczyzny,

Wzór ten składa się z dwóch części. Pierwsza część wzoru jest wyrażona jako $\lambda \cdot ||w||^2$ i jest wykorzystywana do minimalizacji błędu za pomocą wektorów i funkcji kar. Druga część wzoru jest wyrażona jako $\left[\frac{1}{n}\sum_{i=1}^{n} max\left(0,1 - y_i(w^T x_i - b)\right)\right]$ i jest wykorzystywana do określenia granic decyzyjnych, które oddzielają dane w określonych grupach.

Zadanie optymalizacji w wersji prymalnej dla zadania SVM

Zadanie optymalizacji w wersji prymalnej zadania SVM polega na znalezieniu optymalnych hiperpłaszczyzn, które najlepiej dzielą dane na dwie klasy. W tej wersji problemu optymalizacji wykorzystywane są wektory i funkcje kary, aby zminimalizować błąd.

Problem Prymalny (Pierwotny)

W zadaniu SVM wykorzystuje się następujące wzory:

$$\min_{w,b,\xi} \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{N} \xi_i \tag{2}$$

$$p.o \quad y_i(w^T x_i - b) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \; dla \; każdego \; i$$

gdzie:

$$\xi_i = max\left(0,1 - y_i(w^T x_i - b)\right)$$

ξ_i jest najmniejsza liczbą nieujemną spełniającą nieróność $y_i(w^T x_i - b) \geq 1 - \xi_i$

w – D-wymiarowy wektor wag dla modelu SVM, który jest używany do określenia siły wpływu każdej cechy na wynik,

b – odległość graniczna dla modelu SVM - określa, jak daleko od siebie powinny znajdować się granice oddzielające dane uczące,

C – współczynnik regularyzacji - określa poziom kary za błędy popełnione przez model. Im większa wartość C, tym mniej błędów model popełnia, ale jest to kosztem niższych wyników docelowych,

y_i – etykieta klasowa dla przykładu uczącego - określa, do której klasy należy dany przykład uczący,

x_i – wektor cech dla przykładu uczącego - określa cechy danego przykładu uczącego, które są używane przez model SVM do klasyfikacji,

ξ_i – błąd dla każdego przykładu uczenia

Zadanie optymalizacji w wersja dualnej dla zadania SVM

Zadanie optymalizacji w wersji dualnej zadania SVM polega na znalezieniu optymalnych hiperpłaszczyzn, które najlepiej dzielą dane na dwie klasy. Problem optymalizacji jest wtedy przedstawiony jako problem dualny, w którym optymalizacja jest wykonywana przy pomocy wektorów Lagrange'a i funkcji kar. W tej wersji problemu optymalizacji wykorzystywane są wektory oraz funkcje kar, aby zminimalizować błąd.

Problem Dualny (Podwójny)

W zadaniu SVM wykorzystuje się następujące wzory:

$$maximize\ f(c_1 \dots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i \left(x_i^T x_j\right) y_j c_j \tag{3}$$

$$p.o \sum_{i=1}^{n} c_i y_i = 0$$

$$0 \le c_i \le \frac{1}{2n\lambda}\ dla\ każdego\ i$$

Zmiana c_i definiowana jest w następując sposób:

$$w = \sum_{i=1}^{n} c_i y x_i$$

Ponadto $c_i = 0$ dokładnie, kiedy x_i leży po właściwej stronie marginesu oraz $0 < c_i < \frac{1}{2n\lambda}$ kiedy x_i leży na granicy marginesu.

Gdzie:

$c_i -$ to wielkości dualne, które wskazują, jak bardzo przykłady uczące wpływają na optymalne wartości w i b,

$y_i -$ etykieta klasowa dla przykładu uczącego,

$x_i -$ wektor cech dla przykładu uczącego w - wektor wag dla modelu SVM,

$b -$ odległość graniczna dla modelu SVM,

$\lambda -$ współczynnik regularyzacji.

2. Cel projektu

Celem projektu jest zapoznanie się z kwadratowym zadaniem optymalizacji w wersji prymalnej i dualnej dla zadania SVM, a następnie przy pomocy odpowiednio wybranej funkcji ze skrzynki narzędziowej OPTYMALIZATION znaleźć najlepszą płaszczyznę rozdzielającą dwie klasy rozwiązując zadanie prymalne i dualne do prymalnego. Rozwiązanie należy przetestować dla danych wygenerowanych losowo oraz dla zestawu wskazanego w treści projektu.

3. Rozwiązanie

Funkcja quadprog

Do wyznaczenia najlepszej płaszczyzny rozdzielającej dane wykorzystano funkcję quadprog z ToolBox 'OPTIMIZATION' programu Matlab.

Funkcja quadprog z pakietu OPTIMIZATION jest wykorzystywana do optymalizacji zadania kwadratowego oraz do zadań optymalizacyjnych o ograniczeniach liniowych i nieliniowych. Funkcja ta jest szeroko stosowana w celu rozwiązywania problemów optymalizacji, takich jak optymalizacja parametrów, optymalizacja zasobów i optymalizacja wielokryterialna. Jest to funkcja wielokryterialna, co oznacza, że wymaga kilku kryteriów, które muszą być spełnione. Funkcja ta jest często stosowana w problemach optymalizacji, takich jak optymalizacja kosztów, optymalizacja czasu, optymalizacja jakości, optymalizacja wydajności i wiele innych.

Składnia funkcji wygląda następująco: $[x, fval] = quadprog(H, c, A, b, Aeq, beq, x0, options)$, gdzie:

$\mathrm{H} -$ macierz symetryczna, która zawiera parametry optymalizacji,

$\mathrm{c} -$ wektor współczynników optymalizacji,

$\mathrm{A} -$ macierz ograniczeń $-\ \mathrm{b} -$ wektor ograniczeń,

$\mathrm{Aeq} -$ macierz równań,

$\mathrm{beq} -$ wektor równań,

$\mathrm{x0} -$ punkt początkowy,

$\mathrm{options} -$ opcje optymalizacji, takie jak długość kroku i metoda optymalizacji.

Funkcja ta przyjmuje te argumenty i zwraca jako wynik optymalne wartości funkcji celu oraz optymalny punkt rozwiązania.

Zadania znalezienia najlepszej płaszczyzny rozdzielającej dla:

1. Wygenerowanych losowo liczb

Zgodnie z treścią zadania należy wygenerować losowo dwa zbiory punktów w przestrzeni 5-wymiarowej, które leżą po dwóch stronach 4-wymiarowej hiperpłaszczyzny liniowej zawartej w tej przestrzeni (jeden po jednej, drugi po drugiej stronie)

Generowanie losowych liczb

Kod wykorzystany do generacji liczb losowych:

```
%% Generacja losowych dwóch zbiorów punktów w przestrzeni 5-wymiarowej, które leżą
po dwóch stronach 4-wymiarowej hiperpłaszczyzny liniowej zawartej w tej
przestrzeni

clear all

% Ustawienie zmiennej number na wartość, która określa liczbę generowanych punktów
number=1000;

% Ustawienie zmiennych min_number i max_number, które określają zakres danych
min_number = -10000;
max_number = 10000;

% Generowanie losowych 5-wymiarowych danych z zakresu od min do max i
przypisywanie ich do klas
X = min_number + (max_number-min_number)*rand(number,5);
%X = rand(number,5)
Y = sign(X(:,1));

% Rozdzielenie danych losowych na dane uczące i testowe
sizeY = size(Y);
k = round(0.7*sizeY);

% Zbiór treningowy
TrainSetX = X(1:k,:);
TrainSetY = Y(1:k);

% Zbiór testowy
TestSetX = X((k+1):sizeY,:);
TestSetY = Y((k+1):sizeY);
```

Kod generuje losowe 5-wymiarowe dane z zakresu od min_number do max_number i przypisuje je do klas. Dane losowe są następnie dzielone na dane uczące i testowe. Zmienna sizeY jest przypisana do rozmiaru Y. Zmienna k jest ustawiona na 0,7 wielkości sizeY. Dane losowe są następnie zapisywane do zmiennych TrainSetX i TrainSetY odpowiadających pierwszym k punktom losowym – 70% wszystkich danych. Zapisywanie danych losowych do zmiennych TestSetX i TestSetY odpowiada punktom losowym od k+1 do końca – 30% wszystkich danych.

Rozwiązanie prymalne

```
% Wywołanie funkcji svm_primal, która wykorzystuje optymalizację quadprog do
znalezienia optymalnych wag i progu dla SVM
[w, b, time_p, fval_p, exitflag_p, output_p ] = svm_primal(TrainSetX, TrainSetY,
options);

% Wywołanie funkcji svm_pred_primal, aby użyć wag i progu SVM do predykcji danych
testowych
acc_primal = svm_pred_primal(TestSetX, TestSetY, w, b);
```

Pierwsza linia kodu wywołuje funkcję svm_primal, która wykorzystuje optymalizację quadprog do znalezienia optymalnych wag i progu dla SVM. Funkcja ta przyjmuje dane treningowe (TrainSetX i TrainSetY) oraz opcje (options). Wywołuje ona funkcję quadprog, aby znaleźć optymalne wagi i próg SVM. Zwraca ona również czas wykonania, optymalną funkcję celu, flagę wyjścia i informacje o wyjściu.

Druga linia kodu wywołuje funkcję svm_pred_primal, aby użyć wag i progu SVM do predykcji danych testowych. Funkcja ta przyjmuje dane testowe (TestSetX i TestSetY) oraz wagi i próg SVM, które zostały wcześniej wyznaczone. Zwraca ona dokładność predykcji.

Rozwiązanie dualne

```
% Wywołanie funkcji svm_dual, która wykorzystuje optymalizację quadprog do
znalezienia optymalnych wartości alfa dla SVM
[alfa, time_d, fval_d, exitflag_d, output_d ] = svm_dual(TrainSetX, TrainSetY,
options);

% Wywołanie funkcji svm_pred_dual, aby użyć wartości alfa SVM do predykcji danych
testowych
acc_dual = svm_pred_dual( TestSetX, TestSetY, alfa, TrainSetX, TrainSetY );
```

Pierwsza linia kodu wywołuje funkcję svm_dual, która wykorzystuje optymalizację quadprog do znalezienia optymalnych wartości alfa dla SVM. Funkcja ta przyjmuje dane treningowe (TrainSetX i TrainSetY) oraz opcje (options). Wywołuje ona funkcję quadprog, aby znaleźć optymalne wartości alfa SVM. Zwraca ona również czas wykonania, optymalną funkcję celu, flagę wyjścia i informacje o wyjściu.

Druga linia kodu wywołuje funkcję svm_pred_dual, aby użyć wartości alfa SVM do predykcji danych testowych. Funkcja ta przyjmuje dane testowe (TestSetX i TestSetY), wartości alfa SVM oraz dane treningowe (TrainSetX i TrainSetY). Zwraca ona dokładność predykcji.

Implementacja metod *svm_primal, svm_pred_primal* oraz *svm_dual, svm_pred_dual* zostanie zaprezentowana w dalszej części pracy.

Wyniki

Wyniki uzyskane dla rozwiązania prymalnego i dualnego dla różnej liczby wygenerowanych punktów znajdujących się w różnych przedziałach wartości prezentuje tabela 1 i tabela 2.

- liczby z przedziału $\langle -100,100 \rangle$

Tabela 1 Wyniki uzyskane dla rozwiązania zadania prymalnego i dualnego – liczby z przedziału $\langle -100,100 \rangle$

	ZDANIE PRYMALNE			**ZADANIE DUALNE**		
Liczba wygenerowanych punktów	dokładność	liczba iteracji	czas wykonania obliczeń	dokładność	liczba iteracji	czas wykonania obliczeń
10	66.6667	5	0.11125	66.6667	5	0.02084
50	86.6667	8	0.08930	86.6667	8	0.01925
100	93.3333	10	0.09682	90	9	0.02450
500	97.3333	44	0.84569	98	37	0.06725
1000	99.66667	57	5.58871	99.3333	49	0.34171

- Liczby z przedziału $\langle -10000,10000 \rangle$

Tabela 2 Wyniki uzyskane dla rozwiązania zadania prymalnego i dualnego – liczby z przedziału $\langle -10000,10000 \rangle$

	ZDANIE PRYMALNE			**ZADANIE DUALNE**		
Liczba wygenerowanych punktów	dokładność	liczba iteracji	czas wykonania obliczeń	dokładność	liczba iteracji	czas wykonania obliczeń
10	66.6667	5	0.09358	66,6667	6	0.01883
50	86.6667	7	0.09633	80	10	0.02022
100	96.6667	15	0.09619	96.6667	15	0.02734
500	98.6667	40	0.76608	99.3333	36	0.07569
1000	99.6667	51	5.21589	99.3333	47	0.34793

Wnioski

W związku z tym, że liczba elementów była generowana losowo za każdym razem przy wywołaniu programu w powyższych tabelach zaprezentowano przykładowe wyniki otrzymane dla liczb z przedziału $\langle -100,100 \rangle$ oraz $\langle -10000,10000 \rangle$ oraz liczby wygenerowanych punktów równej kolejno $\{10, 50, 100, 500, 1000\}$. W obu tabelach widać, że wraz ze wzrostem liczby wygenerowanych punktów dokładność uzyskanych wyników rosła, w większości przypadków była ona nieznacznie lepsza dla zadania prymalnego. Wraz ze wzrostem liczy punktów rośnie również liczba potrzebnych iteracji oraz czasu do wykonywania obliczeń. Liczba operacji w obu problemach różniła się nieznacznie, jednak w przypadku czasu widać znaczną przewagę zadań dualnych - czas potrzebny na wykonanie obliczeń

różnił się o rząd wielkości. Podział liczb, z którego były losowane konkretne wartości nie miał znaczącego wpływu na parametry zawarte w tabeli.

Implementacja

```
%% Generacja losowych dwóch zbiorów punktów w przestrzeni 5-wymiarowej, które leżą
po dwóch stronach 4-wymiarowej hiperpłaszczyzny liniowej zawartej w tej
przestrzeni

clear all

% Ustawienie zmiennej number na wartość, która określa liczbę generowanych punktów
number=1000;

% Ustawienie zmiennych min_number i max_number, które określają zakres danych
min_number = -10000;
max_number = 10000;

% Generowanie losowych 5-wymiarowych danych z zakresu od min do max i
przypisywanie ich do klas
X = min_number + (max_number-min_number)*rand(number,5);
%X = rand(number,5)
Y = sign(X(:,1));

% Rozdzielenie danych losowych na dane uczące i testowe
sizeY = size(Y);
k = round(0.7*sizeY);

% Zbiór treningowy
TrainSetX = X(1:k,:);
TrainSetY = Y(1:k);

% Zbiór testowy
TestSetX = X((k+1):sizeY,:);
TestSetY = Y((k+1):sizeY);

% Ustawienie opcji optymalizacji dla funkcji quadprog
options = optimoptions(@quadprog,'Algorithm','interior-point-convex');

% Wywołanie funkcji svm_primal, która wykorzystuje optymalizację quadprog do
znalezienia optymalnych wag i progu dla SVM
[w, b, time_p, fval_p, exitflag_p, output_p ] = svm_primal(TrainSetX, TrainSetY,
options);

% Wywołanie funkcji svm_pred_primal, aby użyć wag i progu SVM do predykcji danych
testowych
acc_primal = svm_pred_primal(TestSetX, TestSetY, w, b);

% Wywołanie funkcji svm_dual, która wykorzystuje optymalizację quadprog do
znalezienia optymalnych wartości alfa dla SVM
[alfa, time_d, fval_d, exitflag_d, output_d ] = svm_dual(TrainSetX, TrainSetY,
options);

% Wywołanie funkcji svm_pred_dual, aby użyć wartości alfa SVM do predykcji danych
testowych
acc_dual = svm_pred_dual( TestSetX, TestSetY, alfa, TrainSetX, TrainSetY );

% Wyświetlenie wyników
```

```
disp('zadanie prymalne');
output_p
acc_primal

disp('zadanie dualne');
output_d
acc_dual

% Komentarz
% Kod tworzy losowy zbiór danych 5-wymiarowych, który jest podzielony na dane
uczące i testowe. Następnie wykorzystywana jest funkcja optymalizacji quadprog do
rozwiązania zadania prymalnego i dualnego SVM. Zadania te mają na celu znalezienie
optymalnych wag i progu dla SVM. Na koniec, dokonano predykcji za pomocą zarówno
zadania prymalnego, jak i dualnego, aby sprawdzić dokładność.
```

Implementacja metod *svm_primal, svm_pred_primal* oraz *svm_dual, svm_pred_dual* zostanie zaprezentowana w dalszej części pracy.

2. Zbiór danych tic-tac-toe

Opis przydzielonego zbioru danych

Do realizacji projektu wykorzystano dane – tic-tac-toe.data. Opis zbioru danych został zamieszczony poniżej:

Tic-Tac-Toe Endgame data set

1: Description.

This database encodes the complete set of possible board configurations at the end of tic-tac-toe games, where x is assumed to have played first. The target concept is win for x (i.e., true when x has one of 8 possible ways to create a three-in-a-row).

Attributes description:

1. TopLeft -> top-left-square: {x,o,b}
2. TopMiddle -> top-middle-square: {x,o,b}
3. TopRight -> top-right-square: {x,o,b}
4. MiddleLeft -> middle-left-square: {x,o,b}
5. MiddleMiddle -> middle-middle-square: {x,o,b}
6. MiddleRight -> middle-right-square: {x,o,b}
7. BottomLeft -> bottom-left-square: {x,o,b}
8. BottomMiddle -> bottom-middle-square: {x,o,b}
9. BottomRight -> bottom-right-square: {x,o,b}

2: Type. *Classification*
3: Origin. *Real world*
4: Instances. *958*
5: Features. *9*
6: Classes. *2*
7: Missing values. *No*
8: Header.

@relation tic-tac-toe
@attribute TopLeft {x, o, b}
@attribute TopMiddle {x, o, b}
@attribute TopRight {x, o, b}
@attribute MiddleLeft {x, o, b}
@attribute MiddleMiddle {o, b, x}
@attribute MiddleRight {o, b, x}
@attribute BottomLeft {x, o, b}
@attribute BottomMiddle {o, x, b}
@attribute BottomRight {o, x, b}
@attribute Class {positive, negative}
@inputs TopLeft, TopMiddle, TopRight, MiddleLeft, MiddleMiddle, MiddleRight, BottomLeft, BottomMiddle, BottomRight

Zbiór ten zawiera 10 kolumn i 957 wierszy, wszystkie kolumny mają wartości tekstowe (rys. 1).

TicTacToe =

957×10 table

TopLeft	TopMiddle	TopRight	MiddleLeft	MiddleMiddle	MiddleRight	BottomLeft	BottomMiddle	BottomRight	Class
{'x'}	{'x'}	{'x'}	{'x'}	{'o'}	{'o'}	{'b'}	{'o'}	{'b'}	{'positive'}
{'x'}	{'x'}	{'x'}	{'x'}	{'o'}	{'o'}	{'b'}	{'b'}	{'o'}	{'positive'}
{'x'}	{'x'}	{'x'}	{'x'}	{'o'}	{'b'}	{'o'}	{'o'}	{'b'}	{'positive'}
{'x'}	{'x'}	{'x'}	{'x'}	{'o'}	{'b'}	{'o'}	{'b'}	{'o'}	{'positive'}
{'x'}	{'x'}	{'x'}	{'x'}	{'o'}	{'b'}	{'b'}	{'o'}	{'o'}	{'positive'}
:	:	:	:	:	:	:	:	:	:
{'b'}	{'o'}	{'b'}	{'b'}	{'o'}	{'x'}	{'x'}	{'o'}	{'x'}	{'negative'}
{'b'}	{'b'}	{'o'}	{'x'}	{'b'}	{'o'}	{'x'}	{'x'}	{'o'}	{'negative'}
{'x'}	{'o'}	{'x'}	{'o'}	{'o'}	{'x'}	{'x'}	{'x'}	{'o'}	{'negative'}
{'x'}	{'o'}	{'o'}	{'o'}	{'x'}	{'x'}	{'x'}	{'x'}	{'o'}	{'negative'}
{'o'}	{'x'}	{'o'}	{'o'}	{'x'}	{'x'}	{'x'}	{'o'}	{'x'}	{'negative'}

Rys. 1 Fragment zbioru danych tic-tac-toe.dat w postaci znakowej.

Dane zwarte w tym zbiorze były w postaci znakowej, dlatego też przy pomocy funkcji *categorical* z pakiety *Machine Learning Toolbox* zostały zmienione na wartości numeryczne (rys. 2).

5	5	5	5	3	3	1	3	1	4
5	5	5	5	3	3	1	1	3	4
5	5	5	5	3	1	3	3	1	4
5	5	5	5	3	1	3	1	3	4
5	5	5	5	3	1	1	3	3	4
⋮									
1	3	1	1	3	5	5	3	5	2
1	1	3	5	1	3	5	5	3	2
5	3	5	3	3	5	5	5	3	2
5	3	3	3	5	5	5	5	3	2
3	5	3	3	5	5	5	3	5	2

Rys. 2 Fragment zbioru danych tic-tac-toe.dat w postaci numerycznej..

Klasa ‘positive’ przyjęła wartość 4, a ‘negative’ 2.

Następnie rozdzielono w sposób losowy dane na zbiór treningowy i testowy w proporcji 70/30

```
% Podziałna zbiór uczący i testowy w proporcji 70/30
[m,n] = size(TicTacToe) ;
[m,n]
division = 0.7;
idx = randperm(m);

% Losowo wybrany zbiór uczacy
Training = TicTacToe(idx(1:round(division*m)),:);
```

```
dlmwrite('TicTacToe-numerical-Train.txt', Training, 'delimiter', ','); % zapis
zbioru uczacegodo pliku .txt

% Losowo wybrany zbiór trenujący
Test= TicTacToe(idx(round(division*m)+1:end),:) ;
dlmwrite('TicTacToe-numerical-Test.txt', Test, 'delimiter', ','); % zapis zbioru
testowego pliku .txt
```

Wartości klasy ‘positive’, które po przekształceniu do postaci liczbowej została przyporządkowana wartość 4 przypisano wartość 1, zaś klasie ‘negative’, która była równa 2 wartość -1.

```
% Wczytanie danych uczących -  podział na macierz danych i klas
Train = dlmread('TicTacToe-numerical-Train.txt');

% Zamiana wartosci 'positive'->4->1 oraz 'negative'->2->-1
Training_labels = Train(:,10);
Training_labels( Training_labels== 2)= -1;
Training_labels( Training_labels== 4)= 1;
Train(:,10)=[];
l = length(Training_labels);

% Wczytanie danych testowych -  podział na macierz danych i klas
Test = dlmread('TicTacToe-numerical-Test.txt');

Test_labels = Test(:,10);
Test_labels( Test_labels== 2)= -1;
Test_labels( Test_labels== 4)= 1;
Test(:,10)=[];
l = length(Test_labels);
```

Zabiegu tego dokonujemy, ponieważ większość algorytmów uczenia maszynowego wymaga, aby klasy były reprezentowane przez wartości binarne. Algorytmy uczenia maszynowego są zazwyczaj zaprojektowane do pracy z wartościami binarnymi, dlatego też dane są zamieniane na wartości binarne (1 i -1).

Na koniec wykorzystano już wcześniej zaimplementowanych funkcji służących do rozwiazywania zadania dualnego i prymlnego.

Wyniki

W związku z tym, że zbór treningowy i testowy wybierane są w sposób losowy przy każdorazowym uruchomieniu programu, więc w poniższej tabeli przedstawiono przykładowe wartości uzyskane dla 10 wywołań programu.

Tabela 3 Wyniki uzyskane dla rozwiązania zadania prymalnego i dualnego – dane tic-tac-toe.dat.

	ZDANIE PRYMALNE			**ZADANIE DUALNE**		
Numer wywołania programu	dokładność	liczba iteracji	czas wykonania obliczeń	dokładność	liczba iteracji	czas wykonania obliczeń
1	**68.6411**	**6**	**0.7236**	**57.8397**	**6**	**0.0802**
2	66.5505	6	0.7331	66.5505	5	0.0696
3	**65.1568**	**5**	**0.6710**	**58.1882**	**6**	**0.0687**
4	62.3693	6	0.7094	62.3693	5	0.0724
5	64.1115	5	0.7111	64.1115	5	0.1057
6	68.2927	5	0.6585	68.2927	6	0.0686
7	62.7178	7	0.8258	62.7178	5	0.0770
8	60.2787	4	0.5337	60.2787	5	0.0618
9	70.7317	5	0.6257	70.7317	6	0.0751
10	**67.5958**	**5**	**0.6293**	**61.3240**	**6**	**0.0649**

Wnioski

Na podstawie przeprowadzonych testów, można dojść do wniosku, że rozwiązanie zadania prymalnego i dualnego oscyluje głównie w przedziale od 60% do 70%. W większości przypadków otrzymana dokładność obu zadań była w przybliżeniu jednakowa i różniła się nieznacząco na kolejnych miejscach po przecinko. Jednak w kilku przypadkach (numer 1, 3, 10) mieliśmy do czynienia z widoczną różnicą. W tej sytuacji za każdym razem bardziej dokładny okazywało się rozwiązanie zadania pyrymalnego (różnica ta wynosiła niecałe 10%).

W obu przypadkach zadań liczba iteracji była zbliżona i znajdowała się w przedziale od 5 do 7 iteracji.

Znacząca różnicę da się dostrzec w przypadku czasu wykonania obliczeń. Dla problemu dualnego jest on znacznie krótszy niż w przypadku problemów prymalnych. Dla zadania prymalnego zawiera się on w przedziale $< 0.5337,\ 0.8258 >$, zaś dualnego w $< 0.0618,\ 0.1057 >$.

Implementacja

```
clear all

% Wczytanie danych tic-tac-toe.dat przydzielonych w projekcie
TicTacToe = readtable('tic-tac-toe.dat');
VarNames = {'TopLeft', 'TopMiddle', 'TopRight', 'MiddleLeft', 'MiddleMiddle',
'MiddleRight', 'BottomLeft', 'BottomMiddle', 'BottomRight', 'Class'};
TicTacToe.Properties.VariableNames = VarNames;

% Zmiana danych w postaci znakowej na numeryczne
cats = categorical(TicTacToe{:,:});
TicTacToe = double(cats);
```

```
% Zapisanie danych w formie liczbowej do pliku .txt
dlmwrite('TicTacToe-numerical.txt', TicTacToe, 'delimiter', ',')

% Podziałna zbiór uczący i testowy w proporcji 70/30
[m,n] = size(TicTacToe) ;
[m,n]
division = 0.7;
idx = randperm(m);

% Losowo wybrany zbiór uczacy
Training = TicTacToe(idx(1:round(division*m)),:);
dlmwrite('TicTacToe-numerical-Train.txt', Training, 'delimiter', ','); % zapis
zbioru uczacegodo pliku .txt

% Losowo wybrany zbiór trenujący
Test= TicTacToe(idx(round(division*m)+1:end),:) ;
dlmwrite('TicTacToe-numerical-Test.txt', Test, 'delimiter', ','); % zapis zbioru
testowego pliku .txt

% Wczytanie danych uczących -  podział na macierz danych i klas
Train = dlmread('TicTacToe-numerical-Train.txt');

% Zamiana wartosci 'positive'->4->1 oraz 'negative'->2->-1
Training_labels = Train(:,10);
Training_labels( Training_labels== 2)= -1;
Training_labels( Training_labels== 4)= 1;
Train(:,10)=[];
l = length(Training_labels);

% Wczytanie danych testowych -  podział na macierz danych i klas
Test = dlmread('TicTacToe-numerical-Test.txt');

Test_labels = Test(:,10);
Test_labels( Test_labels== 2)= -1;
Test_labels( Test_labels== 4)= 1;
Test(:,10)=[];
l = length(Test_labels);

%Ustawienie opcji optymalizacji dla funkcji quadprog
options = optimoptions(@quadprog,'Algorithm','interior-point-convex');

% Wywołanie funkcji svm_primal, która wykorzystuje optymalizację quadprog do
znalezienia optymalnych wag i progu dla SVM
[w, b, time_p, fval_p, exitflag_p, output_p ] = svm_primal(Train, Training_labels,
options);

% Wywołanie funkcji svm_pred_primal, aby użyć wag i progu SVM do predykcji danych
testowych
acc_primal = svm_pred_primal(Test, Test_labels, w, b);

% Wywołanie funkcji svm_dual, która wykorzystuje optymalizację quadprog do
znalezienia optymalnych wartości alfa dla SVM
[alfa, time_d, fval_d, exitflag_d, output_d ] = svm_dual(Train, Training_labels,
options);

% Wywołanie funkcji svm_pred_dual, aby użyć wartości alfa SVM do predykcji danych
testowych
acc_dual = svm_pred_dual( Test, Test_labels, alfa, Train, Training_labels );
```

```
% Wyświetlenie wyników
disp('zadanie prymalne');
output_p
acc_primal

disp('zadanie dualne');
output_d
acc_dual
```

4. Wnioski końcowe

Na podstawie obu rozwiązań można uznać, że przy pomocy zadań prymalnych w większości przypadków jesteśmy w stanie uzyska dokładniejsze wyniki, jednak czas ich uzyskania jest znacznie większy niż w przypadku problemów dualnych.

5. Implementacja funkcji wykorzystywanych w obu zadaniach

svm_primal.m

```
function [ w, b, time, fval, exitflag, output ] = svm_primal( trainX, trainY,
options )

    C = 1;
    [N, D] = size(trainX);
    H = zeros(D+1+N);
    H(1:D, 1:D) = eye(D);
    f = [ zeros(D+1,1); C*ones(N,1) ];
    A = [ -(trainY*ones(1,D)).*trainX, -trainY, -eye(N) ];
    b = -1*ones(N,1);
    LB = [ -Inf*ones(D+1,1); zeros(N,1) ];
    UB = [ Inf*ones(D+1+N, 1) ];

    tic
    if nargin == 3
        [X,fval,exitflag,output] = quadprog(H, f, A, b, [], [], LB, UB, [],
options);
    end
    if nargin ==2
        [X,fval,exitflag,output] = quadprog(H, f, A, b, [], [], LB, UB, []);
    end

    time = toc;
    w = X(1:D);
    b = X(D+1);
end
```

svm_pred_primal.m

```
function [ acc ] = svm_pred_primal( testX, testY, w, b )

    N = length(testY);
    pred = sign(testX*w + b);
```

```
    acc = sum(pred==testY)/N * 100;

end
```

svm_dual.m

```
function [ alpha, time, fval, exitflag, output  ] = svm_dual( trainX, trainY,
options )

    C = 1;
    N = length(trainY);
    K = trainX*trainX';
    H = 2*diag(trainY)*K*diag(trainY);
    f = -ones(N,1);
    Aeq = trainY';
    beq = [0];
    LB = zeros(N,1);
    UB = C*ones(N,1);

    tic;
    if nargin == 3
        [alpha,fval,exitflag,output] = quadprog(H, f, [], [], Aeq, beq, LB, UB,
[], options);
    end
    if nargin ==2
        [alpha,fval,exitflag,output] = quadprog(H, f, [], [], Aeq, beq, LB, UB,
[]);
    end

    time = toc;

end
```

svm_pred_dual.m

```
function [ acc ] = svm_pred_dual( testX, testY, alpha, trainX, trainY )

    N = length(testY);
    predict = sign(sum(((alpha.*trainY)*ones(1,N)).*(trainX*testX'), 1));
    acc = sum(testY==predict')/N*100;

end
```

6. Bibliografia

https://en.wikipedia.org/wiki/Support_vector_machine
https://pl.wikipedia.org/wiki/Maszyna_wektor%C3%B3w_no%C5%9Bnych
https://uk.mathworks.com/help/optim/ug/quadprog.html
https://sci2s.ugr.es/keel/category.php?cat=clas&order=name#sub2