

Uniwersytet Jagielloński w Krakowie
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Pavlo Boidachenko

Nr albumu: 1124969

Aplikacja uczenia maszynowego metodą SVM

Praca licencjacka
na kierunku informatyki

Praca wykonana pod kierunkiem
dr Grzegorz Surówka
Zakład Technologii Informatycznych

Kraków 2019

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

.....

Kraków, dnia

.....

Podpis autora pracy

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

.....

Kraków, dnia

.....

Podpis kierującego pracą

Spis treści

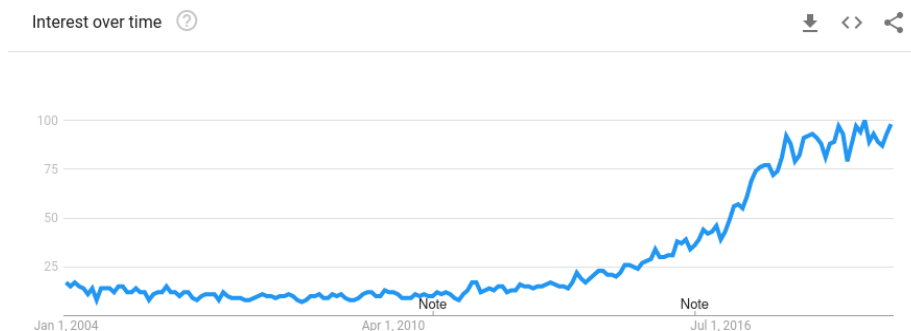
1	Wstęp	5
1.1	Motywacja	5
1.2	Cel	5
1.3	Zakres	5
2	Metoda klasyfikacji SVM	6
2.1	Opis	6
2.2	Formalny opis problemu	6
2.2.1	Przypadek liniowo separowalnych danych	6
2.2.2	Przypadek liniowo nieseparowalnych danych	9
2.3	Typy SVM	12
2.4	Jądra	14
3	Wymagania	15
3.1	Wymagania ogólne	15
3.2	Sterowanie aplikacją LEPSZA NAZWA?	15
3.3	Wejście/Wyjście	15
3.4	Wizualizacja	15
3.5	Przypadki użycia	16
3.5.1	Przypadek użycia 1. Trenowanie modelu	16
3.5.2	Przypadek użycia 2. Testowanie modelu	17
3.5.3	Przypadek użycia 3. Wizualizacja danych	17
3.6	Obsługa błędów	17
3.7	Środowisko wywołania	18
4	Analiza wymagań	19
4.1	Języki programowania	19
4.2	Rysowanie wykresów	19
4.3	Interfejs graficzny	19
4.4	Biblioteki implementujące metodę uczenia SVM	19
4.5	<i>LIBSVM</i>	19
5	Projekt	21
5.1	Model danych	21
5.2	Szkielet kodu	21
5.2.1	Sygnały i gniazda Qt	21
5.2.2	Klasy C++	24
5.2.3	Skrypty Python	25
5.3	Organizacja plików	26
6	Implementacja	27
6.1	Pole tekstowe dla komunikatów	28
6.2	Zarządzanie plikami z danymi	29
6.3	Parametry SVM i funkcji jądrowych	31
6.4	Przyciski trenowania i testowania	31

6.5	Walidacja krzyżowa i holdout	33
6.6	Skalowanie danych	35
6.7	Wizualizacja danych	37
6.8	Wybór cech	41
6.9	Zmiana formatu danych na <i>LIBSVM</i>	41
6.10	Optymalizacja parametrów	43
6.11	Shrinking Kurczenie się????	43
6.12	Estymacja prawdopodobieństwa	44
7	Kompilacja i zależności	45
7.1	Zależności	45
7.2	Kompilacja	45
8	Przykłady działania aplikacji	46
8.1	Przykład 1	46
8.2	Przykład 2	49
8.3	Przykład 3	52
8.4	Przykład 4	56
9	Podsumowanie	58
9.1	Rozwój aplikacji	58

1 Wstęp

1.1 Motywacja

W aktualne czasy temat Uczenia Maszynowego jest popularny^{1.1} jak nigdy do tego. Projekty z użyciem Uczenia Maszynowego pozwalają na tworzenie aplikacji które jeszcze 10 lat temu trudno było wyobrazić.



Rysunek 1.1: Machine Learning trends

Źródło: Google Trends

Rozpowszechnienie Uczenia Maszynowego również spowodowało i moje zainteresowanie tematem. Z tego powodu dla swojej pracy licencjackiej wybrałem temat: Aplikacja uczenia maszynowego metodą SVM. Po zakończeniu pracy spodziewam się podwyższyć swoją kompetencje w dziale Uczenia Maszynowego.

SVM jest atrakcyjną metodą nienadzorowanego Uczenia Maszynowego, specjalną własnością której jest ciągle zmniejszenie błędu i maksymalizacja marginesu, co pozwala nie tylko odseparować klasy, a odseparować ich z maksymalnym marginesem. Z czego wynika większa dokładność modelu na nowych danych.

1.2 Cel

Celem mojej pracy licencjackiej jest stworzenie oprogramowania pozwalającego na generowanie modeli używając Maszyny wektorów wspierających(ang. *Support Vector Machine, SVM*) z graficznym interfejsem użytkownika. Program będą mogli użyć osoby potrzebujące szybko przetrenować kilka modeli, przetestować ich dla różnych parametrów, zwizualizować dane. Program ma na celu ułatwienie pracę z Maszyną wektorów wspierających poprzez graficzny interfejs użytkownika oparty na bibliotece QT. Część funkcjonalna programu jest oparta o bibliotekę LIBSVM[1].

1.3 Zakres

Program powinien móc ustawiać parametry dla wybranej metody oraz jądra(ang. *kernel*), generować wykresy podawanych zbiorów danych, interpretować różne formaty zbiorów danych, wykonywać Sprawdzian krzyżowy (ang. *Cross validation, CV*), mieć metodę do optymalizacji parametrów, pokazywać wyniki trenowania oraz testowania modeli.

2 Metoda klasyfikacji SVM

2.1 Opis

Swój program napisałem w oparciu o bibliotekę LIBSVM[1]. Maszyna Wektorów Wspierających - klasyfikator, nauka którego ma na celu wyznaczenie hiperpłaszczyzny rozdzielającej dwie klasy z maksymalnym marginesem. Zaletą takiego klasyfikatora jest to że po uczeniu margines mówi jak dobrze są odseparowane klasy. LIBSVM implementuje pięć typów Maszyny Wektorów Wspierających C-SVC, ν -SVC, One class SVM, ϵ -SVR, ν -SVR.

2.2 Formalny opis problemu

2.2.1 Przypadek liniowo separowalnych danych

Jest dany trenujący zbiór:

$$(x_1, y_1), (x_2, y_2), \dots, (x_l, y_l) \quad (2.1)$$

Gdzie $y \in \{-1, 1\}$ jest etykietą wektora x_i z przestrzeni wektorowej V (zwykle \mathbf{R}^n), a l rozmiar zbioru trenującego.

Funkcję decyzyjną algorytmu klasyfikującego można przedstawić tak:

$$f(x) : V \rightarrow \{-1, 1\} \quad (2.2)$$

$f(x) = 1$ jeśli klasyfikator przepisuje x_i do pierwszej klasy, a $f(x) = -1$ jeśli do drugiej.

Równanie hiperpłaszczyzny

W przestrzeni \mathbf{R}^n równanie

$$\langle \omega, \mathbf{x} \rangle + b = 0 \quad (2.3)$$

definiuje $(n-1)$ -wymiarowy zbiór wektorów nazywany hiperpłaszczyzną. Gdzie $\omega = (\omega_1, \omega_2, \dots, \omega_n) \in \mathbf{R}^n$ niezerowy wektor, a $b \in \mathbf{R}$ jest skalar. Zbiór wszystkich wektorów $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbf{R}^n$ spełniających równanie 2.3 formuje hiperpłaszczyznę (rysunek 2.1a), dalej oznaczaną jako $\pi(\omega, b)$.

Mówimy że hiperpłaszczyzna $\pi(\omega, b)$ rozdziela dwa zbiory (klasy) C_1 i C_2 jeżeli:

$$\begin{aligned} \langle \omega, \mathbf{x} \rangle + b &> 0, \forall \mathbf{x} \in C_1 \\ \langle \omega, \mathbf{x} \rangle + b &< 0, \forall \mathbf{x} \in C_2 \end{aligned} \quad (2.4)$$

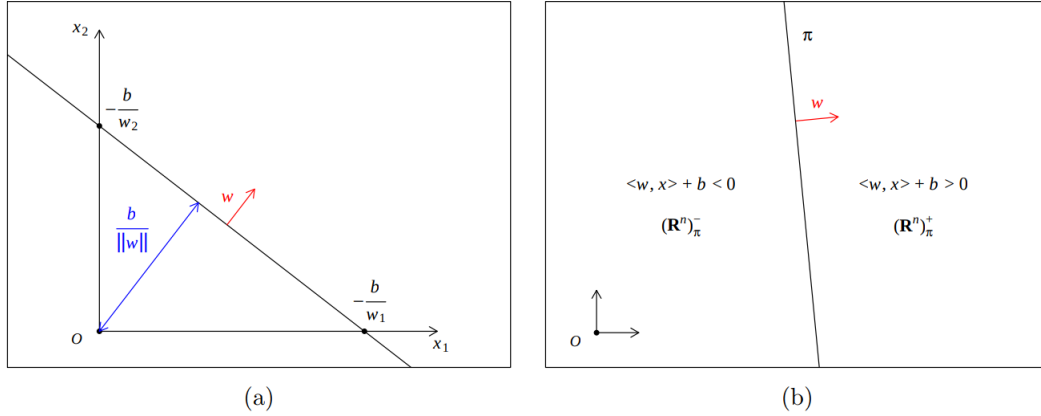
lub

$$\begin{aligned} \langle \omega, \mathbf{x} \rangle + b &< 0, \forall \mathbf{x} \in C_1 \\ \langle \omega, \mathbf{x} \rangle + b &> 0, \forall \mathbf{x} \in C_2 \end{aligned}$$

Dwa zbiory są liniowo separowalne jeżeli istnieje hiperpłaszczyzna $\pi(\omega, b)$ która ich rozdziela [10]. Jeśli hiperpłaszczyzna rozdziela zbiory według 2.4, to funkcja decyzyjna jest taka:

$$f(x) = \text{sign}\{\langle \omega, \mathbf{x} \rangle + b\} = \begin{cases} 1, & \text{if } \langle \omega, \mathbf{x} \rangle + b \geq 0 \\ -1, & \text{if } \langle \omega, \mathbf{x} \rangle + b < 0 \end{cases} \quad (2.5)$$

Funkcja 2.5 potrafi poklasyfikować wszystkie wektory zbiorów C_1 i C_2 .



Rysunek 2.1: (a) hiperpłaszczyznę w dwuwymiarowej przestrzeni jest linia, (b) dodatnia i ujemna półprzestrzeni zdefiniowane hiperpłaszczyznę $\pi(\omega, b)$ [10]

Margines

Odległość pomiędzy wektorem a płaszczyzną można obliczyć za pomocą tego równania:

$$\rho(x, \pi) = \frac{\langle \omega, x \rangle + b}{\|\omega\|} \quad (2.6)$$

Oczywistym jest fakt, że dwa separowalnych zbiory separuje nieskończona liczba hiperpłaszczyzn. SVM wybiera jedną z maksymalnym marginesem. Dla hiperpłaszczyzny π rozdzielającą zbiory C_1 i C_2 marginesem $m(\pi, C_1, C_2)$ jest odległość pomiędzy π i zbiorem C_1 plus odległość pomiędzy π i zbiorem C_2 .

$$m(\pi, C_1, C_2) = \rho(\pi, C_1) + \rho(\pi, C_2) \quad (2.7)$$

odległość pomiędzy zbiorem C a hiperpłaszczyzną π można zdefiniować jako minimalną odległość pomiędzy π i wektorem ze zbioru C :

$$\rho(\pi, C) = \min_{x \in C} |\rho(\pi, x)| \quad (2.8)$$

Weźmy dwa punkty z różnych klas: x_+ i x_- należące do klas 1 i -1 odpowiednio. Wtedy szerokością marginesu jest [12]:

$$\left\langle (x_+ - x_-), \frac{\omega}{\|\omega\|} \right\rangle = \frac{\langle \omega, x_+ \rangle - \langle \omega, x_- \rangle}{\|\omega\|} = \frac{(b+1) - (b-1)}{\|\omega\|} = \frac{2}{\|\omega\|} \quad (2.9)$$

Z czego wynika, że margines jest maksymalny kiedy norma wektora ω jest minimalna.

Jeśli klasy są separowalne, zadanie rozdzielania dwóch zbiorów hiperpłaszczyzną sprowadza się do znalezienia takich ω i b przy których norma wektora ω jest minimalna.

Pierwotny problem optymalizacyjny

Parametry ω i b dla hiperpłaszczyzny SVM są rozwiązaniem dla poniższego problemu:

$$\begin{aligned} & \frac{1}{2} \|\omega\|^2 \rightarrow \min_{\omega, b} \\ \text{Z zastrzeżeniem że } & \langle \omega, \mathbf{x} \rangle + b \geq 1, \forall \mathbf{x} \in C_1 \\ & \langle \omega, \mathbf{x} \rangle + b \leq -1, \forall \mathbf{x} \in C_2 \end{aligned} \quad (2.10)$$

Gdzie C_1 i C_2 dwa zbiory punktów należących do różnych klas [10].

Problem można też zapisać w takiej formie:

$$\begin{aligned} & \frac{1}{2} \sum_{k=1}^n \omega_k^2 \rightarrow \min_{\omega, b} \\ \text{Z zastrzeżeniem że } & y_i \left(\sum_{k=1}^n \omega_k x_{ik} + b \right) \geq 1, i = 1, 2, \dots, l. \end{aligned} \quad (2.11)$$

gdzie $y_i = 1$ jeśli $x_i \in C_1$ i $y_i = -1$ jeśli $x_i \in C_2$.

Problem dualny

Większość implementacji SVM rozwiązują problem dualny, a nie pierwotny, przez to że warunki graniczne są łatwiejsze do obliczeń i jest możliwość użycia funkcji jądrowych.

Problem dualny dla 2.11 wygląda następująco:

$$\begin{aligned} & L_d(\alpha) \rightarrow \max_{\alpha} \\ \text{Z zastrzeżeniem że } & \alpha_i \geq 0, i = 1, 2, \dots, l \end{aligned} \quad (2.12)$$

gdzie $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_l)$, a $L_d(\alpha)$ zdefiniowany jako:

$$L_d(\alpha) = \min_{\omega, b} L(\omega, b, \alpha) \quad (2.13)$$

a $L(\omega, b, \alpha)$ - Lagrange'an zdefiniowany tak:

$$L(\omega, b, \alpha) = \frac{1}{2} \sum_{k=1}^n \omega_k^2 - \sum_{i=1}^l \alpha_i \left(y_i \left(\sum_{k=1}^n \omega_k x_{ik} + b \right) - 1 \right). \quad (2.14)$$

W tym przypadku Lagrange'an ma formę wypukłą, a więc dla każdego α para (ω^*, b^*) jest globalnym minimum dla $L(\omega, b, \alpha)$ wtedy i tylko wtedy gdy:

$$\nabla_{\omega, b} L(\omega^*, b^*, \alpha) = \mathbf{0} \quad (2.15)$$

gdzie $\nabla_{\omega, b} L$ jest gradientem funkcji L , a $\mathbf{0}$ jest wektorem zerowym. A zatem $L_d(\alpha) = L(\omega^*, b^*, \alpha)$ jeśli warunek 2.15 jest spełniony, a więc problem 2.12 może być zapisany następująco [10]:

$$\begin{aligned} & L(\omega, b, \alpha) \rightarrow \max_{\omega, b, \alpha} \\ \text{Z zastrzeżeniem że } & \nabla_{\omega, b} L(\omega^*, b^*, \alpha) = \mathbf{0} \\ & \alpha_i \geq 0, i = 1, 2, \dots, l \end{aligned} \quad (2.16)$$

Pierwszy warunek w 2.2.1 oznacza:

$$\begin{aligned} \frac{\partial L}{\partial \omega_k} &= 0, k = 1, 2, \dots, n \\ \frac{\partial L}{\partial b} &= 0 \end{aligned} \quad (2.17)$$

co jest równoważne do:

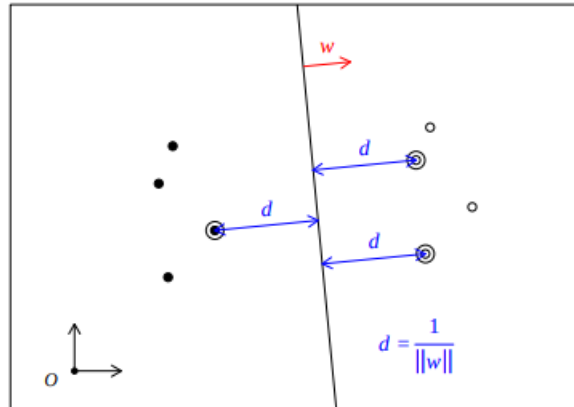
$$\begin{aligned}\omega_k &= \sum_{i=1}^l \alpha_i y_i x_{ik}, k = 1, 2, \dots, n \\ \sum_{i=1}^l \alpha_i y_i &= 0\end{aligned}\tag{2.18}$$

Podstawiając 2.24 w 2.14 problem 2.2.1 można zapisać problem jako:

$$\begin{aligned}\sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle &\rightarrow \max_{\boldsymbol{\alpha}} \\ \text{Z zastrzeżeniem że } \sum_{i=1}^l \alpha_i y_i &= 0 \\ \alpha_i &\geq 0, i = 1, 2, \dots, l\end{aligned}\tag{2.19}$$

Wektory wspierające

Związkiem pomiędzy problemem pierwotnym a dualnym jest 2.24, z czego wynika że wektor znormalizowany $\boldsymbol{\omega}$ jest rozszerzany współczynnikami $\alpha_i y_i$ na wektory trenujące \mathbf{x}_i , gdzie $\alpha \geq 0$ i $y_i = 1$ jeśli $x_i \in C_1$ i $y_i = -1$ jeśli $x_i \in C_2$. Wektory trenujące takie że $\alpha > 0$ są nazywane *wektorami wspierającymi*, i leżą one na marginesie hiperpłaszczyzny rozdzielającej dwie klasy.



Rysunek 2.2: Wektory wspierające (z kółkami) są najbliższymi wektorami do rozdzielającej hiperpłaszczyzny [10]

2.2.2 Przypadek liniowo nieseparowalnych danych

Najczęściej jest tak, że dane nie są liniowo separowalne, w takim przypadku wyżej opisane równania nie mogą być rozwiązane, przez niemożliwość dotrzymania się warunkowi w 2.11.

Problem pierwotny

Możemy rozluźnić warunek w 2.11 wprowadzając błąd ε_i która jest nazywana luźną zmienną (*ang. slack variable*) [10]:

$$y_i \left(\sum_{k=1}^n \omega_k x_{ik} + b \right) \geq 1 - \varepsilon_i, i = 1, 2, \dots, l \quad (2.20)$$

$$\varepsilon_i \geq 0, i = 1, 2, \dots, l$$

Przy minimizacji funkcji

$$\frac{1}{2} \sum_{k=1}^n \omega_k^2 \rightarrow \min_{\omega, b, \varepsilon} \quad (2.21)$$

wyjdzie że przez dowolność wyboru ε_i każdy wektor ω może pasować do równania. Żeby zrobić problem sensownym musimy też minimalizować błędy, co zwykle jest robione przez dodanie ich sum

$$\frac{1}{2} \sum_{k=1}^n \omega_k^2 + C \sum_{i=1}^n \varepsilon_i \rightarrow \min_{\omega, b, \varepsilon}$$

Z zastrzeżeniem że $y_i \left(\sum_{k=1}^n \omega_k x_{ik} + b \right) \geq 1 - \varepsilon_i, i = 1, 2, \dots, l \quad (2.22)$

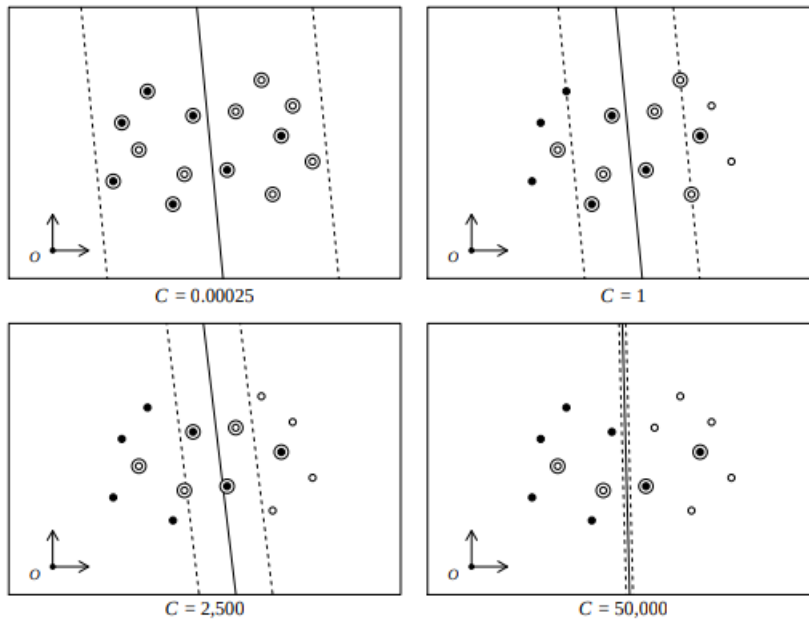
$$\varepsilon_i \geq 0, i = 1, 2, \dots, l$$

gdzie C jest dodatnią stałą która służy dla: maksymalizacji marginesu i minimalizacji liczby błędów na danych trenujących.

Problem 2.22 definiuje tak zwany SVM z miękkim marginesem (*ang. soft margin SVM*). Dla tego typu SVM chcemy znaleźć hiperpłaszczyznę rozdzielającą klasy z maksymalnym marginesem, dopuszczając żeby wektory trenujące leżały wewnątrz marginesu albo były błędnie sklasyfikowane.

Parametr C

Jak opisano wyżej, parametr C ma dwie cele: maksymalizować margines i minimalizować liczbę błędów (punkty znajdujące się wewnątrz marginesu lub błędnie sklasyfikowane). Te dwie cele konfliktują między sobą. Zmieniając parametr C dążymy do jednego z celów i oddalamy się od innego, a więc znalezienie optymalnego C jest zadaniem znalezienia 'złotego środka'. Na rysunku 2.3 jest pokazana zależność marginesu od parametru C . Kiedy C jest małe suma błędów staje się zbędnie mała i głównym celem staje maksymalizacja marginesu, co może powodować, że margines może zawierać wszystkie punkty ze zbioru trenującego. W przeciwieństwie do tego, kiedy parametr C jest duży: suma błędów robi się duża i cel minimalizacji liczby błędów dominuje nad maksymalizacją marginesu. W wyniku może powstać hiperpłaszczyzna margines której w ogóle nie zaeiwra punktów.



Rysunek 2.3: Optymalne hiperpłaszczyzny separujące dane i ich marginesy dla tego samego zbioru trenującego używając różnych parametrów [10]

Problem dualny

Wyprowadzenie problemu dualnego dla 2.22 jest podobne do 2.11. Finalna forma wygląda następująco:

$$\begin{aligned}
 & \sum_{i=1}^l -\frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \rightarrow \max_{\boldsymbol{\alpha}} \\
 & \text{Z zastrzeżeniem że } \sum_{i=1}^l \alpha_i y_i = 0 \\
 & 0 \leq \alpha_i \leq C, i = 1, 2, \dots, l.
 \end{aligned} \tag{2.23}$$

Wektory wspierające

Analogicznie do przypadku z separowalnymi danymi, jeżeli $(\boldsymbol{\omega}, b)$ jest rozwiązaniem pierwotnym, a $\boldsymbol{\alpha}$ jest rozwiązaniem pierwotnym, to:

$$\boldsymbol{\omega} = \sum_{i=1}^l \alpha_i y_i \mathbf{x}_i \tag{2.24}$$

Wektory trenujące \mathbf{x}_i dla których $\alpha_i > 0$ są nazywane wektorami wspierającymi hiperpłaszczyzny rozdzielającej. Wektory trenujące w SVM z miękkim marginesem mogą znajdować się wewnątrz marginesu, poza nim lub w po niepoprawnej stronie hiperpłaszczyzny. Również liczba wektorów wspierających zależy od parametru C : zmniejszając C margines rozszerza się i liczba wektorów wspierających może się zwiększyć [10].

2.3 Typy SVM

1. C-SVC

C-Support Vector Classification - rodzaj klasyfikatora używający C jako parametr regularyzacji. Jeśli jest dany wektor $x_i \in R^n$, $i = 1, \dots, l$ w dwóch klasach i wektor etykiet $y_i \in \{1, -1\}$ to dla C-SVC problem pierwotny wygląda tak:

$$\begin{aligned} \min_{\omega, b, \varepsilon} \quad & \frac{1}{2} \omega^T \omega + C \sum_{i=1}^l \varepsilon_i \\ \text{Z zastrzeżeniem że} \quad & y_i(\omega^T \phi(x_i) + b) \geq 1 - \varepsilon_i \\ & \varepsilon_i \geq 0, i = 1, \dots, l \end{aligned}$$

gdzie ϕ jest funkcją jądrową.

Dobry model dobrze separuje dane i razem z tym ma duży margines. Natomiast w rzeczywistości jedno wyłącza drugie: duży margines włącza punkty z dwóch klas, a dobre separowanie może powodować przeuczanie(ang. Overfitting). Przeuczanie może skutkować tym że model jest dobry na danych treningowych ale jest zły na danych testowych.

2. ν -SVM

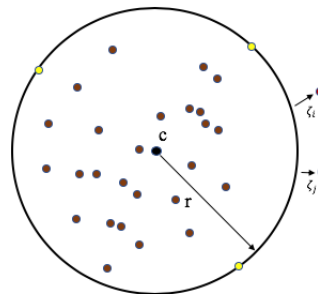
ν -Support Vector Classification - rodzaj klasyfikatora używający ν jako parametr regularyzacji. Jest bardzo podobny do C-SVM, z różnicą że $\nu \in [0, 1]$. Przyjemną właściwością ν jest to że on jest dolną granicą stosunku wektorów wspierających i górną granicą stosunku błędu uczenia.

Jeśli jest dany wektor $x_i \in R^n$, $i = 1, \dots, l$ w dwóch klasach i wektor $y \in R^l$ taki że $y_i \in \{1, -1\}$ to pierwotny problem optymalizacji wygląda następująco:

$$\begin{aligned} \min_{\omega, b, \varepsilon, \rho} \quad & \frac{1}{2} \omega^T \omega - \nu \rho + \frac{1}{l} \sum_{i=1}^l \varepsilon_i \\ \text{Z zastrzeżeniem że} \quad & y_i(\omega^T \phi(x_i) + b) \geq \rho - \varepsilon_i \\ & \varepsilon_i \geq 0, i = 1, \dots, l, \rho \geq 0 \end{aligned}$$

3. One-class SVM

One-class Support Vector Machine - rodzaj klasyfikatora uczenia nienadzorowanego, które zakłada brak etykiet w danych uczących. Ma na celu znalezienie niewiadomych wzorców/anomalii(klastrów) w danych wejściowych.



Rysunek 2.4: Hipersfera zawierająca punkty danych. Ma środek c i promień R . Punkty na krawędzi są wektorami wspierającymi.

Źródło: Wikipedia. https://en.wikipedia.org/wiki/One-class_classification

Jeśli dany jest wektor $x_i \in R^n$, $i = 1, \dots, l$ bez informacji o klasach, to pierwotny problem optymalizacji wygląda następująco:

$$\begin{aligned} \min_{\omega, \varepsilon, \rho} \quad & \frac{1}{2} \omega^T \omega - \rho + \frac{1}{\nu l} \sum_{i=1}^l \varepsilon_i \\ \text{Z zastrzeżeniem że} \quad & \omega^T \phi(x_i) \geq \rho - \varepsilon_i, \\ & \varepsilon \geq 0, i = 1, \dots, l \end{aligned}$$

4. ϵ -SVR

Jeśli wektor etykiet $y_i \in R$ to jest używana metoda regresji. ϵ -Support Vector Classification - używa C i ϵ jako parametrów regularyzacji. Celem jest znalezienie takiej funkcji $f(x)$ że jej wartość odchyła się od y_n na wartość nie większą od ϵ dla każdego punktu z zbioru treningowego.

Jeśli jest dany zbiór danych treningowych $\{(x_1, y_1), \dots, (x_l, y_l)\}$, gdzie $x \in R^n$ jest wektorem cech, a $y \in R^1$ jest wyjściem. Przy danych parametrach $C > 0$ i $\epsilon > 0$, standardowa forma SVR to:

$$\begin{aligned} \min_{\omega, b, \varepsilon, \varepsilon^*} \quad & \frac{1}{2} \omega^T \omega + C \sum_{i=1}^l \varepsilon_i + C \sum_{i=1}^l \varepsilon_i^* \\ \text{Z zastrzeżeniem że} \quad & \omega^T \phi(x_i) + b - y_i \leq \epsilon + \varepsilon_i, \\ & y_i - \omega^T \phi(x_i) - b \leq \epsilon + \varepsilon_i^*, \\ & \varepsilon_i, \varepsilon_i^* \geq 0, i = 1, \dots, l \end{aligned}$$

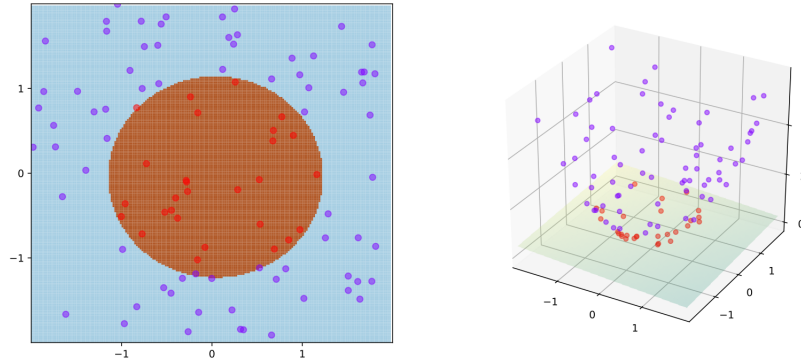
5. ν -SVR

ν -Support Vector Regression - podobnie do ν -SVC, używa parameter $\nu \in (0, 1]$ dla kontroli liczby wektorów wspierających. Również używa parametru ϵ . Z parametrami (C, ν) dla ν -SVR problem pierwotny to:

$$\begin{aligned} \min_{\omega, b, \epsilon, \varepsilon^*, \epsilon} \quad & \frac{1}{2} \omega^T \omega + C(\nu \epsilon + \frac{1}{l} \sum_{i=1}^l (\varepsilon_i + \varepsilon_i^*)) \\ \text{Z zastrzeżeniem że} \quad & (\omega^T \phi(x_i) + b) - y_i \leq \epsilon + \varepsilon_i, \\ & y_i - (\omega^T \phi(x_i) + b) \leq \epsilon + \varepsilon_i^*, \\ & \varepsilon_i, \varepsilon_i^* \geq 0, i = 1, \dots, l, \epsilon \geq 0 \end{aligned}$$

2.4 Jądra

W przypadku kiedy klasy nierozdzielne liniowo to jest używany trik z zastosowaniem funkcji jądrowych(ang. Kernel Trick). Funkcję jądrowe pozwalają odwzorować zbiór danych w przestrzeń z większą liczbą wymiarów, w której klasy danego zbioru będzie można rozdzielić hiperpłaszczyzną.



Rysunek 2.5: Przykład stosowania funkcji jądrowej dla nierozdzielnego liniowo zbioru danych

Źródło: Wikipedia. https://en.wikipedia.org/wiki/Kernel_method

W bibliotece LIBSVM są zaimplementowane następujące funkcje jądrowe:

Liniowa	$K(x, x') = x \cdot x'$
Wielomianowa	$K(x, x') = (\gamma * x \cdot x' + coef0)^{degree}$
RBF	$K(x, x') = -\gamma * (\sqrt{x} + \sqrt{x'} - 2 * x \cdot x')$
Sigmoidalna	$K(x, x') = \tanh(\gamma * x \cdot x' + coef0)$

Autorzy biblioteki LIBSVM polecają [5] że pierwsza funkcja jądrowa którą warto spróbować to RBF. Jak pokazują [7] jeśli używać RBF z optymalizacją hiperparametrów to nie ma potrzeby rozważać liniową funkcję jądrową. W sigmoidalnej funkcji jądrowej macierz jądrowa może nie być pozytywnie określona i generalnie dokładność modelu nie jest lepsza od RBF [9].

3 Wymagania

3.1 Wymagania ogólne

Aplikacja ma implementować uczenie maszynowe metodą SVM. Uczenie ma być dwufazowe: trenowanie i testowanie. Pod czas trenowania aplikacja ma generować model w postaci pliku. Na etapie testowania aplikacja ma mieć możliwość załadowania dowolnego pliku z modelem.

3.2 Sterowanie aplikacją **LEPSZA NAZWA?**

Sterowanie aplikacją musi odbywać się wyłącznie przez Interfejs graficzny. Interfejs graficzny użytkownika ma być zorganizowany w sposób podkreślający to że nauczanie jest dwufazowe. Akcje które użytkownik nie może zrobić przez ich bezsensowność lub niedostępność na danym etapie muszą być zablokowane w sposób oczywisty dla użytkownika.

3.3 Wejście/Wyjście

Program na wejściu otrzymuje co najmniej jeden plik z danymi zawierający cechy i etykiety. Wyjściem jest plik z modelem i wypisywana na interfejsie graficznym precyzja modelu w przypadku testowania.

3.4 Wizualizacja

Aplikacja musi umieć wizualizować dane wejściowe jedno, dwu i trzy wymiarowe. Dla jedno i dwu wymiarowych danych program musi mieć możliwość generowania wykresu gęstości rozkładu. Dla celów dydaktycznych aplikacja musi umieć wizualizować wytrenowany model.

3.5 Przypadki użycia

3.5.1 Przypadek użycia 1. Trenowanie modelu

Nazwa	Wytrenuj model
Inicjator	Użytkownik
Cel	Wygenerowanie pliku modelu
Warunki wstępne	brak
Główny scenariusz	<ol style="list-style-type: none">1. Użytkownik ładuje plik z danymi do trenowania do aplikacji.2. Aplikacja sprawdza czy format danych jest poprawny.3. Użytkownik wprowadza odpowiednie parametry modelu.4. Użytkownik uruchamia trenowanie.5. Aplikacja generuje trenuje model i zapisuje do pliku.
Rozszerzenia	<ol style="list-style-type: none">2a. Wgrany zbiór danych nie jest w formacie <i>LIBSVM</i>.<ol style="list-style-type: none">a. Użytkownik konwertuje swój zbiór danych do formatu <i>LIBSVM</i>.c. Użytkownik kontynuuje scenariusz główny z p.3.3a. Użytkownik wprowadził tekstowy symbol w pole z liczbami.<ol style="list-style-type: none">a. Aplikacja podświetla niepoprawny parametr na czerwono.b. Użytkownik robi korektę błędnego parametru.c. Użytkownik kontynuuje scenariusz główny z p.4.

3.5.2 Przypadek użycia 2. Testowanie modelu

Nazwa	Testuj model
Inicjator	Użytkownik
Cel	Otrzymać informacje o precyzji modelu
Warunki wstępne	Użytkownik wytrenował model.
Główny scenariusz	<ol style="list-style-type: none">1. Użytkownik ładuje plik z testującym zbiorem danych do aplikacji.2. Użytkownik wprowadza procent walidacji(ile rekordów z zbioru trenującego wybrać do testowania)3. Użytkownik uruchamia testowanie.4. Aplikacja przeprowadza testowanie i walidacje modelu.5. Aplikacja wypisuje wyniki testowania i walidacji w polu tekstowym.
Rozszerzenia	<ol style="list-style-type: none">2a. Załadowany plik jest niepoprawny(inna liczba cech/klas niż w pliku trenującym)<ol style="list-style-type: none">a. Aplikacja podświetla ścieżkę do pliku na czerwono.b. Użytkownik ładuje poprawny plik.c. Użytkownik kontynuuje główny scenariusz z p.3.

3.5.3 Przypadek użycia 3. Wizualizacja danych

Nazwa	Wizualizuj dane
Inicjator	Użytkownik
Cel	Przedstawić dane w postaci graficznej
Warunki wstępne	Użytkownik załadował plik z danymi do trenowania
Główny scenariusz	<ol style="list-style-type: none">1. Użytkownik uruchamia wizualizację danych.2. Aplikacja wczytuje dane z pliku trenującego.3. Aplikacja rysuje punkty na płaszczyźnie w nowym oknie.
Rozszerzenia	<ol style="list-style-type: none">1a. Przycisk wizualizacji jest nieaktywny.<ol style="list-style-type: none">a. Użytkownik sprawdza liczbę cech w podanym pliku trenującym.b. Użytkownik ładuje plik z danymi, w którym liczba cech jest mniejsza lub równa 3.c. Użytkownik kontynuuje główny scenariusz z p.1.

3.6 Obsługa błędów

Informacja o błędach występujących w trakcie działania aplikacji będzie przekazywana do użytkownika w polu tekstowym lub w postaci podświetlenia na czerwono elementu interfejsu w którym jest błąd.

Informacja o błędach wewnętrznych(Maszyna wirtualna Python, system operacyjny zamknął apli-

kacje z powodu naruszenia ochrony pamięci) będą od użytkownika ukryte. Zaawansowane użytkownicy będą mogli uruchomić aplikacje z konsoli, w której takie błędy będą wypisywane, by naprawić środowisko w którym uruchamiają aplikacje.

3.7 Środowisko wywołania

Aplikacja jest tworzona dla systemu operacyjnego Linux. Użytkownicy systemu operacyjnego OS X(wcześniej macOS) będą mogli skompilować i uruchomić aplikacje po instalacji wszystkich zależności. Kompilacja i uruchomienie aplikacji na systemie operacyjnym Windows nie są wspierane, chociaż może to być możliwe, ze względu na to, że użyte technologie są wieloplatformowe(ang. cross platform).

4 Analiza wymagań

4.1 Języki programowania

Dla napisania aplikacji wybrałem język programowania *C++*. Wybrałem ten język programowania przez jego wydajność mimo to że jest on językiem wysokiego poziomu. Na wybór *C++* również wpłynęła moja sympatia do tego języka programowania, a napisanie dużego projektu w nim było dla mnie prawdziwym wyzwaniem, które pozwoliło pogłębić swoją wiedzę w programowaniu i używaniu języka *C++*.

Dla niektórych części programu będzie użyty język *Python*, w szczególności dla tych, które dotyczą rysowania wykresów, pracy z plikami i zmiennymi typu *string*. Standardowa biblioteka Python przedstawia sporo narzędzi dla pracy z plikami i zmiennymi typu *string*, co pozwala oszczędzać czas programisty w porównaniu do *C++*, w którym napisanie podobnej funkcjonalności zajęłoby o wiele więcej czasu.

4.2 Rysowanie wykresów

Dla rysowania wykresów wybrałem bibliotekę *matplotlib* [6] która pozwala stosunkowo łatwo rysować jedno, dwu i trzy-wymiarowe wykresy, a w kombinacji z *numpy* i *scipy* generować wykresy gęstości.

4.3 Interfejs graficzny

Dla implementacji interfejsu graficznego wybrałem wieloplatformową bibliotekę *Qt* [3] napisanej w *C++*. Twórcy biblioteki *Qt* stworzyli IDE dedykowane do pracy z biblioteką - *Qt Creator*. Jest ono wyposażone w szczegółową dokumentację do biblioteki, wbudowany program *Qt Designer* pozwala projektować interfejs graficzny metodą przeciągnij i upuść(ang. drag and drop), automatyczne uzupełnienie kodu, prowadzi analizę semantyczną kodu i wskazuje na błędy jeszcze przed kompilacją.

4.4 Biblioteki implementujące metodę uczenia SVM

Istnieje mnóstwo implementacji metody uczenia SVM. Autorzy strony *SVM - Support Vector Machines*[11] zebrali listę oprogramowania implementującego metodę uczenia SVM. Z tej listy, ze względu na to że jako główny język programowania dla swojej aplikacji wybrałem *C++*, pasowali mi tylko te biblioteki: *SVMLight*, *mySVM*, *LIBSVM*, *SVMTorch*.

Przewagę oddałem *LIBSVM* z uwagi na dobrą dokumentację, obecność instrukcji dla nowicjuszy i dobry rozdział FAQ, który wytłumaczył mi wiele pytań o działaniu tej biblioteki.

4.5 *LIBSVM*

LIBSVM napisana w 2011 roku przez Chih-Chung Chang i Chih-Jen Lin w Państwowym Uniwersytecie Tajwańskim. Celem autorów było zrobienie narzędzia z prostym interfejsem umożliwiające używanie Maszyny Wektorów Wspierających użytkownikom które nie zajmują się zawodowo nauczaniem maszynowym. Domyślnie użycie biblioteki *LIBSVM* polega na skompilowaniu i używaniu dwóch programów: *svm-train* i *svm-predict*. *svm-train* służy do trenowania modelu, a *svm-predict* do testowania, chociaż również może być używany do klasyfikacji nowych niegrupowanych danych.

```
Usage: svm-train [options] training_set_file [model_file]
options:
-s svm_type : set type of SVM (default 0)
    0 -- C-SVC                (multi-class classification)
    1 -- nu-SVC                (multi-class classification)
    2 -- one-class SVM
    3 -- epsilon-SVR           (regression)
    4 -- nu-SVR                (regression)
-t kernel_type : set type of kernel function (default 2)
    0 -- linear: u'*v
    1 -- polynomial: (gamma*u'*v + coef0)^degree
    2 -- radial basis function: exp(-gamma*|u-v|^2)
    3 -- sigmoid: tanh(gamma*u'*v + coef0)
    4 -- precomputed kernel (kernel values in training_set_file)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 100)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates : whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
-wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
-v n : n-fold cross validation mode
-q : quiet mode (no outputs)
```

Rysunek 4.1: Użycie programu svm-train

```
Usage: svm-predict [options] test_file model_file output_file
options:
-b probability_estimates: whether to predict probability estimates, 0 or 1 (default 0); for one-class SVM only 0 is supported
-q : quiet mode (no outputs)
```

Rysunek 4.2: Użycie programu svm-predict

5 Projekt

5.1 Model danych

LIBSVM domyślnie używa tak zwany "zadki" format danych z etykietowanymi wektorami. Format danych do trenowania i testowania jest taki:

```
<label> <index1>:<value1> <index2>:<value2> ...  
.  
.  
.
```

Każda linia jest osobną instancją i kończy się znakiem '\n'. Etykiety(ang. label) są używane na różne sposoby: [1]

- Dla klasyfikacji: etykieta jest liczbą naturalną określającą klasę.
- Dla regresji: etykieta jest wartością docelową i może być dowolną liczbą rzeczywistą.
- Dla jednoklasowego SVM(ang. one-class SVM) etykiety nie są używane i mogą być dowolne.

Jeśli w danym wektorze wartość cechy jest 0 to można ją pominąć, np. wektor $[1, 0, 3, 0]$ w formacie *LIBSVM* można zapisać tak:

```
1:1 3:3
```

Aplikacja zachowuje intencję twórców biblioteki *LIBSVM*: rozdzielić uczenie na fazy trenowania i testowania, a więc użycia dwóch plików z danymi. Pliki trnujące i testujące są zapisywane w takim samym formacie i różnią się tylko celem do którego będą użyte.

5.2 Szkielet kodu

W projekcie aplikacji starałem się napisać kod w kształcie modułów, każdy z których odpowiada za swoją część funkcjonalności.

5.2.1 Sygnały i gniazda Qt

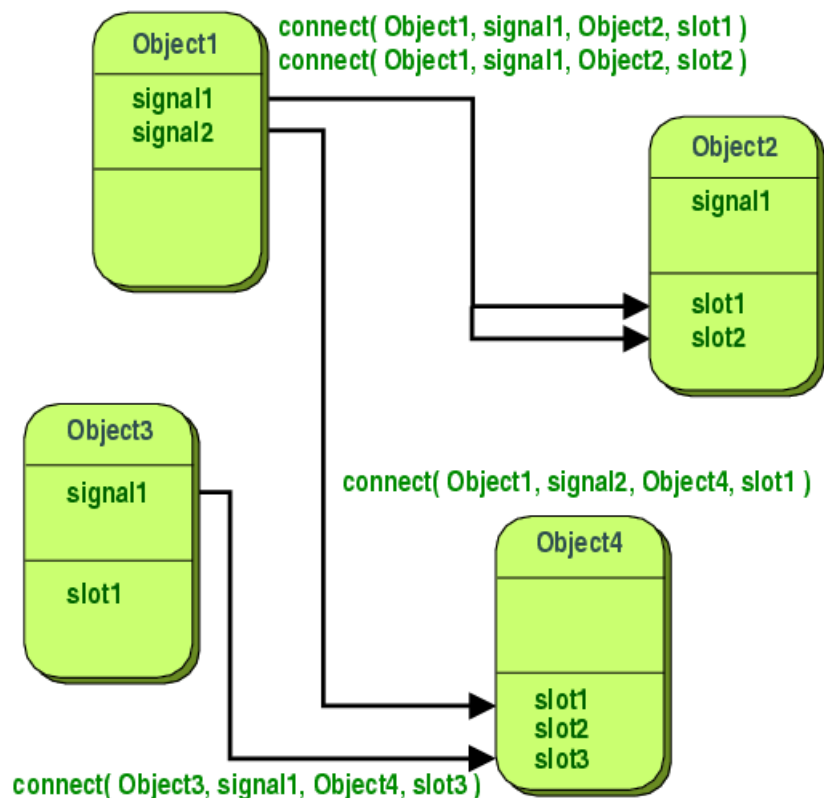
Mechanizm sygnałów i gniazd biblioteki *Qt* [4] jest powszechnie używany w projekcie aplikacji. Sygnały najczęściej są wywoływane przez kontrolki na interfejsie graficznym, a gniazdem jest funkcja do której są przekazywane odpowiednie parametry. Na przykład sygnał *AvailabilityHandler::testEnabled* odpowiadający za dostępność testowania modelu jest połączony z czterema gniazdami:

```

connect(availability_handler , &AvailabilityHandler::testEnabled , ui->test_pushButton ,
        &QPushButton::setEnabled);
connect(availability_handler , &AvailabilityHandler::testEnabled , ui->validation_label
        , &QLabel::setEnabled);
connect(availability_handler , &AvailabilityHandler::testEnabled , ui->
        validation_percent_label , &QLabel::setEnabled);
connect(availability_handler , &AvailabilityHandler::testEnabled , ui->
        validation_lineEdit , &QLineEdit::setEnabled);

```

Co pozwala za pomocą wywołania jednej funkcji *AvailabilityHandler::testEnabled* deaktywować cztery elementy interfejsu.



Rysunek 5.1: Wizualizacja systemu sygnałów i gniazd [4]

Przykład kodu w którym sygnały wywoływane w klasie *AvailabilityHandler* są łączone z gniazdami w kontrolkach z polami do wpisywania parametrów.

```
connect(availability_handler, &AvailabilityHandler::degreeEnabled, ui->degree_spinBox, &QSpinBox::setEnabled);
connect(availability_handler, &AvailabilityHandler::gammaEnabled, ui->gamma_lineEdit, &QLineEdit::setEnabled);
connect(availability_handler, &AvailabilityHandler::coef0Enabled, ui->coef0_lineEdit, &QLineEdit::setEnabled);
connect(availability_handler, &AvailabilityHandler::cEnabled, ui->C_lineEdit, &QLineEdit::setEnabled);
connect(availability_handler, &AvailabilityHandler::nuEnabled, ui->nu_lineEdit, &QLineEdit::setEnabled);
connect(availability_handler, &AvailabilityHandler::pEnabled, ui->P_lineEdit, &QLineEdit::setEnabled);
```

Żeby wyżej wymienione sygnały wywoływały się automatycznie sygnały kontrolek *Combo Box* zmieniających typ SVM lub funkcje jądrową są łączone z gniazdami w klasie *AvailabilityHandler*:

```
connect(ui > svmType_comboBox, SIGNAL(currentIndexChanged(int)), availability_handler, SLOT(filterSVMTypeParams(int)));
connect(ui > kernel_comboBox, SIGNAL(currentIndexChanged(int)), availability_handler, SLOT(filterKernelParams(int)));
```

Z sygnału *currentIndexChanged* jest przekazywany indeks wybranego elementu do gniazda *filterSVMTypeParams* w którym są wywoływany odpowiednie sygnały zarządzające dostępnością parametrów.

5.2.2 Klasy C++

Krótkie opisy najważniejszych klas:

Nazwa klasy	Opis funkcjonalności
AvailabilityHandler	Klasa sterująca dostępnością elementów interfejsu użytkownika. Używa mechanizmu sygnałów i gniazd biblioteki <i>Qt</i> , co pozwala na to, żeby wywołanie metody np. <i>trainButtonEnabled</i> powodowało zmianę stanu na interfejsie graficznym.
FileManager	Klasa zarządza ładowaniem plików do aplikacji, odświeżaniem ścieżek plików i informacji o pliku (takich jak liczba cech, rekordów i klas).
OutputHandler	Klasa odpowiada za wypisywanie tekstu w polu tekstowym. Tworzy potok do którego przekierowuje <i>stdout</i> i uruchamia wątek który czyta z potoku i wszystko co przychodzi na standardowe wyjście wypisuje do interfejsu użytkownika.
ScriptQtManager	Klasa odpowiada za uruchamianie skryptów w języku <i>Python</i> . Przekazuje właściwe parametry do skryptów i zwraca wyjście skryptu do kodu w C++.
SVMController	Klasa pomaga zminimalizować użycie polecenia <i>system()</i> (uruchamia komendę z powłoki <i>sh</i>). Faktycznie klasa opakowuje funkcje biblioteki <i>LIBSVM</i> , żeby wygodnie ich można było używać z kodu C++.
svmscale	Klasa opakowuje funkcjonalność programu <i>svm-scale</i> dostarczanego z biblioteką <i>LIBSVM</i> , pozwala zminimalizować użycie polecenia <i>system()</i> i uruchamiać skalowanie danych bezpośrednio z kodu C++.
MainWindow	Jest jądrem aplikacji, ustawia wszystkie sygnały. Reakcje na kliknięcia w przyciski są zdefiniowane w tej klasie.

5.2.3 Skrypty Python

Krótkie opisy skryptów w języku *Python*:

Nazwa pliku	Opis funkcjonalności
checkdata.py	Skrypt dostarczany z biblioteką <i>LIBSVM</i> , sprawdza czy dane w pliku mają format <i>LIBSVM</i> . Na podstawie wyniku działania tego skryptu aplikacja decyduje czy plik z danymi może być użyty do trenowania lub testowania.
convert2svm.py	Skrypt który stara się skonwertować plik z danymi do formatu <i>LIBSVM</i> . Na wejściu wymaga podania ścieżki do pliku z danymi w formacie innym niż <i>LIBSVM</i> , nazwa pliku wyjściowego z danymi w formacie <i>LIBSVM</i> , separator cech, separator dziesiętny(przecinek lub kropka) i ostatni parametr 1 - jeśli etykieta jest na początku linii i 0 - na końcu linii. Plik wynikowy jest sprawdzany skryptem <i>checkdata.py</i> żeby zapobiec trenowaniu przy pomocy błędnego pliku który może powstać przez błąd użytkownika(np. podanie niepoprawnego separatora cech).
f_select.py	Skrypt pozwala wybrać które cechy są potrzebne w procesie nauczania modelu. Na wejściu wymaga podania pliku z danymi, liczbę cech i jakie cechy należy wybrać. Wyjściem skryptu jest plik z danymi z rozszerzeniem <i>.fselected</i> .
grid.py	Skrypt dostarczany z biblioteką <i>LIBSVM</i> służy do dobierania najlepszych parametrów dla trenowania modelu. Na wejściu wymaga podania początkowych i końcowych wartości dla poszczególnych parametrów oraz krok z którym algorytm "idzie"parametrem.
h4c.py i h4r.py	Holdout dla klasyfikacji i holdout dla regresji. Skrypty pozwalające na podział jednego pliku z danymi na zbiory trenujący i testujący. Skrypty wymagają na wejściu ścieżkę do pliku z danymi oraz procent danych który ma być wybrany dla testowania. <i>h4c.py</i> stara się zachować stosunek klas w zbiorze testującym taki sam jak w pierwotnym pliku, a <i>h4r.py</i> losowo wybiera wskazany procent rekordów. <i>h4r.py</i> jest również używany do walidacji.
plotter.py	Skrypt służy do wizualizacji danych za pomocą biblioteki <i>matplotlib</i> . Umie rysować jedno- dwu- i trzy-wymiarowe wykresy, dla jednego i dwóch wymiarów są również dostępne wykresy gęstości.
pointsgen.py	Skrypt pozwala użytkownikom na ręczne generowanie danych. Po uruchamianiu skryptu otwiera się okno w którym są rejestrowane kliknięcia myszą i na ich miejscu są rysowane punkty. Po zamknięciu okna jest generowany plik z wygenerowanymi danymi.

5.3 Organizacja plików

Kod aplikacji, wszystkie użyte zbiory danych i źródła biblioteki *LIBSVM* znajdują się na repozytorium pod adresem <https://github.com/boidachenkop/svm-app>. Źródła aplikacji znajdują się w katalogu *svm-app*, dla edycji lub przeglądu kodu zaleca się używania IDE(od ang. integrated development environment) *Qt Creator* [2], skoro potrafi ono automatycznie posortować pliki .cpp, .h, .py oraz przedstawić schemat interfejsu graficznego na podstawie pliku *mainwindow.ui*.

Wszystkie zbiory danych używane dla przykładów znajdują się w katalogu *datasets*. Oryginalny kod źródłowy biblioteki *LIBSVM* znajduje się w katalogu *libsvm-3.23*.

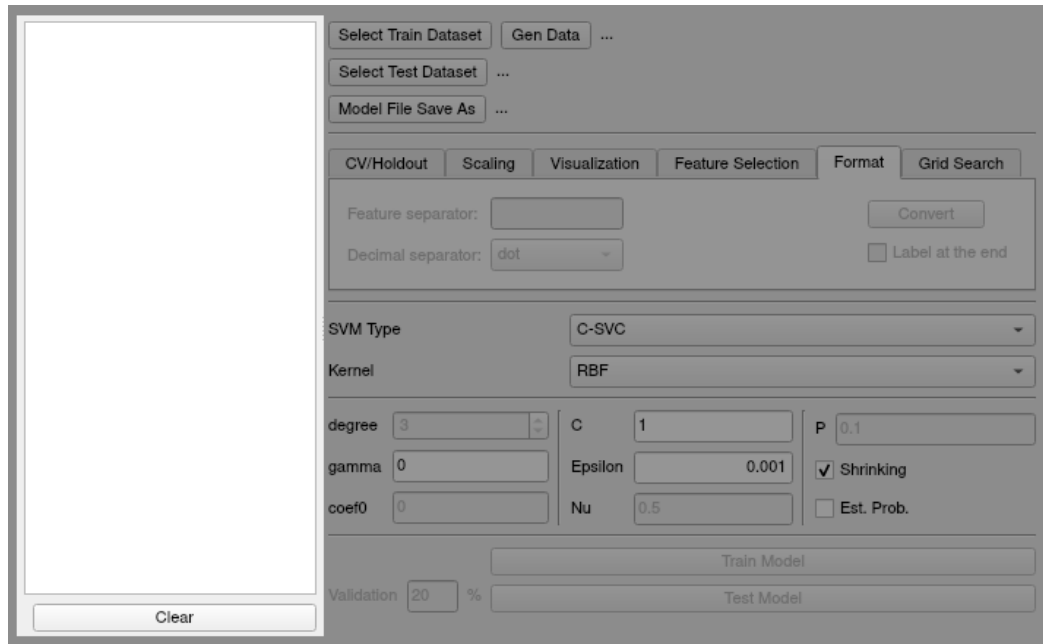
6 Implementacja

Po uruchomieniu programu pierwsze co zobaczy użytkownik - okno główne. Po lewej stronie jest pole tekstowe w którym pojawiają się komunikaty o wynikach trenowania lub testowania modelu. W prawej górnej części okna są kontrolki do wybierania plików z danymi lub zapisywania pliku modelu. Żeby rozdzielić główną funkcjonalność od pobocznej i zrobić okno główne programu bardziej kompaktowym niektóre elementy są zrealizowane w postaci kart, natomiast funkcjonalność która dotyczy trenowania i testowania modelu jest dostępna bezpośrednio z okna głównego.

The screenshot shows the main interface of an SVM application. On the left is a large text area for output, with a 'Clear' button at the bottom. The right side contains a control panel. At the top are buttons for 'Select Train Dataset', 'Gen Data', 'Select Test Dataset', and 'Model File Save As'. Below these are tabs for 'CV/Holdout', 'Scaling', 'Visualization', 'Feature Selection', 'Format', and 'Grid Search'. The 'CV/Holdout' tab is active, showing input fields for 'Feature separator' and 'Decimal separator' (set to 'dot'), a 'Convert' button, and a 'Label at the end' checkbox. Below this are dropdowns for 'SVM Type' (set to 'C-SVC') and 'Kernel' (set to 'RBF'). Further down are input fields for 'degree' (3), 'gamma' (0), 'coef0' (0), 'C' (1), 'Epsilon' (0.001), 'Nu' (0.5), and 'P' (0.1). There are checkboxes for 'Shrinking' (checked) and 'Est. Prob.' (unchecked). At the bottom are 'Train Model' and 'Test Model' buttons, and a 'Validation' field set to 20%.

Rysunek 6.1: Okno główne programu

6.1 Pole tekstowe dla komunikatów.



Rysunek 6.2: Pole dla komunikatów

Pole tekstowe 6.2 w lewej części okna głównego programu przyznaczone dla wypisywania komunikatów z biblioteki *LIBSVM* lub używanych skryptów Python. W zasadzie opisywana kontrolka jest widżetem biblioteki *Qt TextEdit* w którym jest wyłączona możliwość edytowania zawartości. Dla tego żeby nie przerabiać kodu źródłowego biblioteki *LIBSVM* zamieniając każde wywołanie funkcji *printf* na odpowiednią funkcję, któraby dodawała tekst na *TextEdit*, szukałem sposobu na przekierowanie całego wyjścia aplikacji na widżet w interfejsie graficznym. Niestety biblioteka *Qt* nie przedstawia prostej możliwości na zrobienie tego, więc musiałem zaimplementować podobną funkcjonalność samodzielnie. Za tą funkcjonalność odpowiada klasa *OutputHandler*, która w swoim konstruktorze tworzy potok do którego przekierowuje *stdout*.

```
OutputHandler::OutputHandler()
{
    _saved_stdout = dup(STDOUT_FILENO);
    pipe(_stdout_pipe);
    dup2(_stdout_pipe[1], STDOUT_FILENO);
    close(_stdout_pipe[1]);
}
```

Listing 1: Konstruktor klasy *OutputHandler*

Za tym uruchamia wątek który ciągle sprawdza czy coś było zapisane do potoku i jeśli tak to wywołuje sygnał który dopisuje te dane do *TextEdit* na interfejsie graficznym.

```

void OutputHandler::handleOutput()
{
    int bytes_read;
    char buf[1024];
    while((bytes_read = (int)read(_stdout_pipe[0], buf, sizeof(buf)))
    {
        emit updateOutput(QString::fromLatin1(buf, bytes_read));
        if(_cmd_out)
        {
            write(_saved_stdout, buf, bytes_read);
        }
    }
}

```

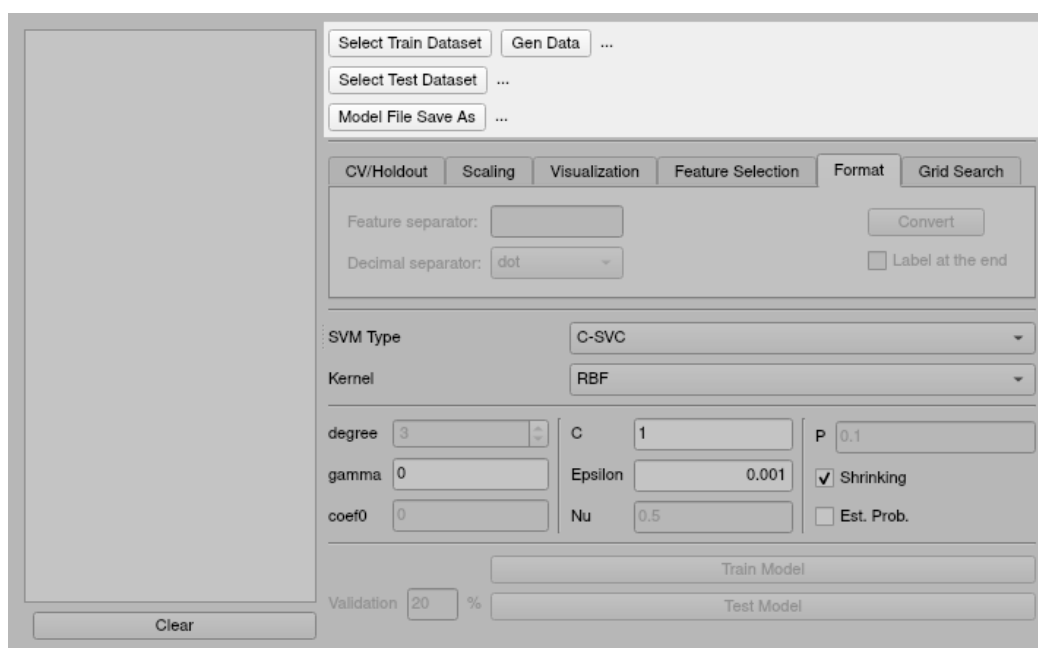
Listing 2: Funkcja uruchamiana w wątku

Takie podejście pozwala używać standardowe sposoby na wypisywanie komunikatów i nie myśleć o tym czy wypisywany komunikat zostanie wyświetlony na interfejsie użytkownika.

Pod omawianym polem tekstowym znajduje się przycisk *Clean*, który usuwa wszystkie wcześniej wypisane komunikaty z pola *TextEdit*.

6.2 Zarządzanie plikami z danymi

Pierwszym krokiem który będzie musiał zrobić użytkownik po uruchamianiu aplikacji - wybrać plik z danymi dla trenowania. Dla tego celu na ekranie głównym są odpowiednie kontrolki 6.3 które pozwalają wybrać plik z danymi do trenowania lub testowania i wybrać katalog w którym należy zapisać plik z wygenerowanym modelem. Domyślnie plik modelu ma nazwę [nazwa pliku z danymi do trenowania + .model]. Przycisk *Gen Data* pozwala wygenerować plik z danymi samodzielnie, jego funkcjonalność będzie omówiona później.



Rysunek 6.3: Zarządzanie plikami

Po wybraniu pliku z danymi program sprawdza czy dane są zapisane w formacie *LIBSVM*. W przypadku poprawności danych program parsuje plik wypisując liczbę rekordów, liczbę klas i liczbę cech jak dla danych trenujących tak i dla danych testowych.

Rysunek 6.4: Przykład wczytanych danych

Format danych *LIBSVM* przewiduje klasę na skrajnej lewej pozycji w linii, cechy są indeksowane, jeśli cecha równa się 0 to informacja o nich jest zbędna. Warto wspomnieć, że wybrana wersja biblioteki *LIBSVM* nie przewiduje użycia danych tekstowych:

```
1 1:3.5 2:2.25 3:1.17 # komentarz
-1 2:3 # pierwsza i trzecia cechy są 0
```

Poprawność danych wejściowych sprawdza skrypt w języku Python udostępniany w ramach biblioteki *LIBSVM*. Jeśli plik mieści niepoprawne dane to ścieżka do wybranego pliku jest farbowana na czerwono i jest wypisywany odpowiedni komunikat. W przypadku 6.5 w pliku z danymi linia 19 mieści cechę wartość której ma liczbę rzeczywistą z przecinkiem, a w formacie danych *LIBSVM* są dopuszczane tylko kropki.

Rysunek 6.5: Przykład wczytanych niepoprawnych danych

Jeśli dane do testowania nie zgadzają się z danymi do trenowania (np. liczbą cech lub liczbą klas) to zbiór do testowania nie jest akceptowany:

Rysunek 6.6: Przykład niezgodności pomiędzy danymi do trenowania a danymi do testowania

Dzięki bibliotece *Qt*, która w miarę możliwości używa natywnych do systemu operacyjnego kontrolek, wygląd selektora plików zależy od systemu operacyjnego. Domyślnie selektor plików otwiera katalog `/home/[username]`, ale w przypadku kiedy pliki z danymi znajdują się głęboko w systemie plików to może być niewygodne, przez to aplikacja przechowuje ostatnio odwiedzony katalog w pliku `/tmp/svmappl-lop` i zawsze otwiera w nim nowy selektor plików.

Niektóre akcje użytkownika mogą nie mieć sensu: zacząć proces trenowania bez wybrania pliku z danymi, zmieniać parametry modelu które nie dotyczą wybranego typu SVM lub wybranej funkcji jądrowej itp. Żeby zapobiec bezsensownym akcjom była zaimplementowana klasa *Availability-Handler*. Klasa odpowiada za deaktywację kontrolek funkcjonalność których jest bezsensowna lub niebezpieczna 6.7

Parameter	C-SVC (RBF)	nu-SVC (Linear)
SVM Type	C-SVC	nu-SVC
Kernel	RBF	Linear
degree	3	3
gamma	0	0
coef0	0	0
C	1	1
Epsilon	0.001	0.001
P	0.1	0.1
Nu	0.5	0.5
Shrinking	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Est. Prob.	<input type="checkbox"/>	<input type="checkbox"/>

Rysunek 6.7: Przykład zmiany dostępności parametrów w zależności od wybranego typu SVM i funkcji jądra

6.3 Parametry SVM i funkcji jądrowych

Wszystkie parametry oprócz *degree* są w postaci kontrolki *Line Edit* ze względu na to że są one liczbami rzeczywistymi, *degree* z kolei jest liczbą naturalną a więc używana jest kontrolka *Spin Box*. Po uruchamianiu trenowania program czyta pola z parametrami i sprawdza czy są poprawne. W pola typu *Line Edit* można wprowadzać litery(ponieważ jest dozwolony zapis liczb w notacji naukowej: $1e-3$). Jeśli wprowadzoną liczbą nie jest w formacie naukowym lub nie jest liczbą rzeczywistą z separatorem kropką, to pole z błędnymi danymi jest obramiane na czerwono a proces treningu jest powstrzymywany.

degree	3	C	0,123	P	0.1
gamma	0	Epsilon	1e-4	<input checked="" type="checkbox"/> Shrinking	
coef0	0	Nu	0.5	<input type="checkbox"/> Est. Prob.	

Rysunek 6.8: Przykład niepoprawnych danych w polu parametru C

6.4 Przyciski trenowania i testowania

Przyciski trenowania i testowania 6.9 są umieszczone w dolnej części okna głównego programu.

Validation	20	%	Train Model
			Test Model

Rysunek 6.9: Przyciski do trenowania i testowania modelu

Po wciśnięciu przycisku *Train Model* aplikacja, jak opisano wyżej, parsuje i sprawdza pola z parametrami i uruchamia proces trenowania dokładnie ten który oferuje program *svm-train* z biblioteki *LIBSVM*. Żeby ściślej powiązać aplikację z biblioteką *LIBSVM* przerobiłem biblioteczne plik źródłowe *svm-train.c* i *svm-predict.c* na klasę *SVMController*, pozwoliło to zapobiec używaniu

polecenia `system()` które służy do uruchamiania programów zewnętrznych. Nie udało się jednak całkiem uniknąć tego - skrypty w języku Python używają `system()`.

Przykład wyjścia algorytmu trenującego LIBSVM C-SVC z funkcją jądrową RBF i parametrami $C = 1$, $gamma = 0$:

```
*
optimization finished, #iter = 318
nu = 0.469505
obj = -383.582676, rho = 0.903354
nSV = 408, nBSV = 401
Total nSV = 408
```

Gdzie *obj* oznacza wartość optymalną dla problemu podwójnego SVM, *rho* jest błędem systematycznym w funkcji decyzyjnej $sgn(w^T x - rho)$, *nSV* jest liczbą wektorów wspierających, *nBSV* - liczbą ograniczonych wektorów wspierających. ν -SVC równoważną formą do C-SVC, *nu* - jest odpowiednikiem podanego parametru C tylko w ν -SVC, czyli jeśli użyć ν -SVC i podać jako parametr ν liczbę 0.469505 to wygenerowany model będzie miał bardzo podobne charakterystyki.

Za przyciskiem *Test Model* poza funkcjonalnością programu *svm-predict* stoi jeszcze jedna - walidacja. Klasa *SVMController* wczytuje plik z danymi do testowania i wypisuje procent dobrych zgadnięć wygenerowanego modelu, po tym w zależności od wartości w polu *Validation* program wyciąga odpowiedni procent danych ze zbioru do trenowania i przeprowadza testowanie na nim. Na ogół testowanie modelu na danych trenujących nie ma sensu ale taka operacja pozwala zidentyfikować czy dany model nie jest przeuczony (np. wynikiem walidacji jest 100%).

Przykład wyjścia algorytmu testującego:

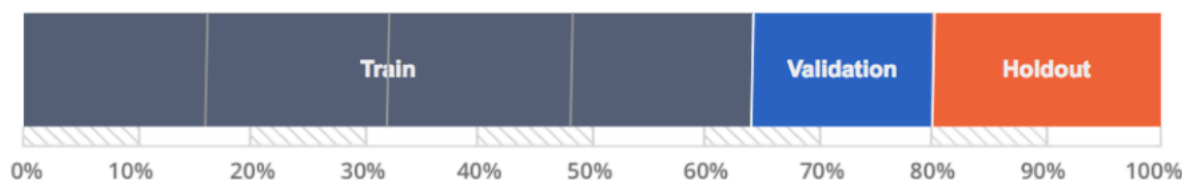
```
Accuracy = 66.925% (2677/4000) (classification)
Validation Accuracy = 99.6769% (617/619) (classification)
```


6.5 Walidacja krzyżowa i holdout

Walidacja krzyżowa(ang. Cross-Validation) i holdout były umieszczone na jednej karcie 6.10 dla zachowania ścisłości interfejsu graficznego, mimo to że ich funkcjonalność nie jest połączona.

Rysunek 6.10: Karta z kontrolkami do walidacji krzyżowej i holdout

Holdout polega na rozdzielaniu zbioru na dane trenujące i dane testujące. Najczęściej zbiór danych jest w jednym pliku, natomiast biblioteka *LIBSVM* jest zorganizowana tak, że potrzebuje plik z danymi trenującymi i plik z danymi testującymi. Za poprawne rozdzielanie danych odpowiadają dwa skrypty w języku Python: *h4c.py* i *h4r.py*. Pierwszy służy do klasyfikacji i stara się zachować stosunek klas w zbiorze testującym, drugi, z kolei, służy do regresji i po prostu wybiera podany procent rekordów tworząc plik z danymi do testowania. Na karcie jest umieszczona kontrolka *Combo Box* która pozwala wybrać pierwsze (*Classification*) lub drugie (*Regression*) podejścia.



Rysunek 6.11: Przykład podziału zbioru danych. *Train* jest do trenowania, a *Holdout* i *Validation* służą do testowania modelu

Źródło: <https://www.datarobot.com/wiki/training-validation-holdout/>

Skrypt holdout dla regresji jest również używany w opisanej wyżej walidacji: on wybiera losowe rekordy z danych trenujących i tworzy z nich tymczasowy plik testujący na którym dalej działa algorytm testowania z biblioteki *LIBSVM*, przez to wynik walidacji może się różnić przy takich

samych parametrach modelu.

Walidacja krzyżowa jest bardzo ważnym narzędziem w nauczaniu maszynowym. Jeśli danych jest mało to rozdzielanie zbioru na dane trenujące i testujące może spowodować niedostateczne nauczanie się modelu, w takim przypadku używamy walidacji krzyżowej. N-krotna walidacja krzyżowa polega na rozdzielaniu zbioru na N równych części jedna z których służy to testowania a wszystkie inne do trenowania. Proces jest iteracyjny, więc każda część w którejś iteracji będzie służyć do testowania.



Rysunek 6.12: Ilustracja procesu 5-krotnej walidacji krzyżowej

Źródło: <https://towardsdatascience.com/>

cross-validation-explained-evaluating-estimator-performance-e51e5430ff85

Również jest dostępna walidacja krzyżowa typu *Leave-one-out*, która jest rodzajem N-krotnej walidacji, tylko zbiór testujący zawsze jest jednoelementowy, czyli jest to równoważne podaniu jako N liczby $\text{rozmiar_zbioru} - 1$.

Warto wspomnieć, że walidacja krzyżowa nie generuje modelu, a służy tylko do określania jak dobry jest model z podanymi parametrami. Kiedy użytkownik uzna dokładność modelu akceptującą on wyłączy walidację krzyżową i wygeneruje model na całym zbiorze trenującym.

6.6 Skalowanie danych

Karta *Scaling* 6.13 służy do skalowania danych. Główną zaletą skalowania danych jest unikanie dominacji cech z większych zakresów liczbowych nad cechami z mniejszych zakresów liczbowych. Skalowanie również zmniejsza obciążenie numeryczne dla algorytmu, ponieważ funkcje jądrowe często zależą od iloczynu wektorowego lub skalarnego cech np. liniowa i wielomianowa funkcje jądrowe, dla których duże wartości w wektorach cech mogą powodować problemy numeryczne.

Rysunek 6.13: Karta z kontrolkami dla skalowania danych

Twórcy biblioteki *LIBSVM* proponują wszystkim początkującym zaczynać swoją pracę z generacją modelu właśnie od skalowania, ponieważ tak prosty krok może mieć bardzo wysoki wpływ na dokładność modelu.

Przykład wyjścia algorytmu trenującego na tym samym zbiorze danych z takimi samymi parametrami do i po skalowaniu danych:

```
...*...*
optimization finished, #iter = 5371
nu = 0.606150
obj = -1061.528918, rho = -0.495266
nSV = 3053, nBSV = 722
Total nSV = 3053
Accuracy = 66.925% (2677/4000) (classification)
Validation Accuracy = 99.6769% (617/619) (classification)
```

```
/* moment w którym by_ło przeprowadzone skalowanie danych */
```

```
*
optimization finished, #iter = 496
nu = 0.202599
obj = -507.307046, rho = 2.627039
nSV = 630, nBSV = 621
Total nSV = 630
Accuracy = 96.15% (3846/4000) (classification)
Validation Accuracy = 95.7997% (593/619) (classification)
```

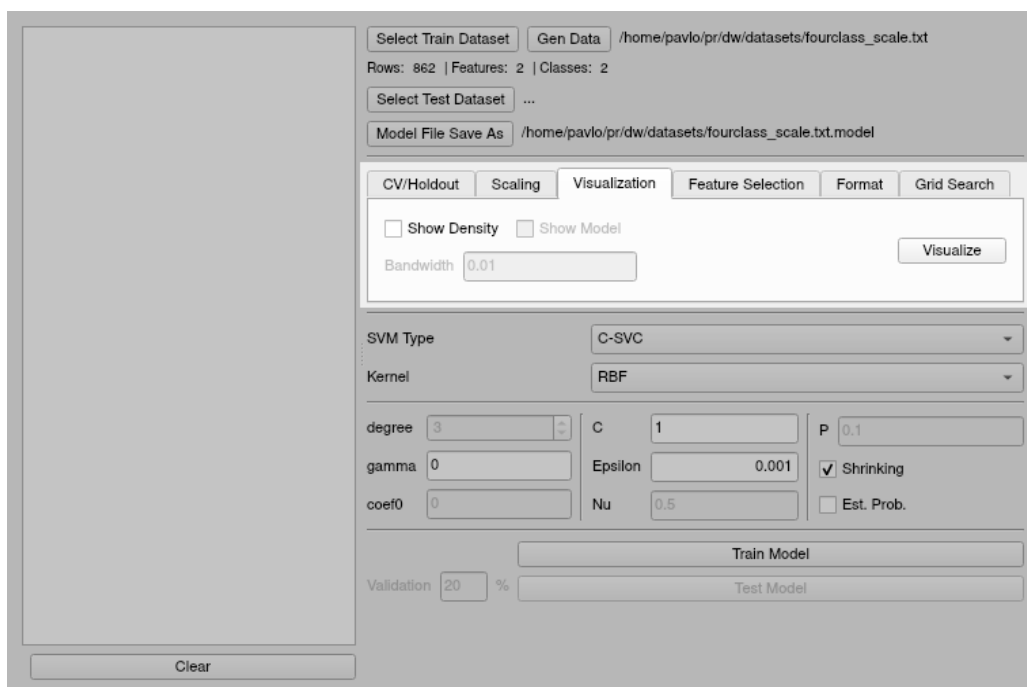
Na tym przykładzie widać, że dokładność modelu wzrosła na 30% przy dziesięciokrotnym zmniejszeniu liczby iteracji algorytmu.

Biblioteka *LIBSVM* zawiera w sobie program *svm-scale* który służy do poprawnego skalowania danych. W ramach projektu aplikacji kod tego programu został przerobiony i zawarty w klasie *svmscale*. Po wybraniu zbioru trenującego kontrolki skalowania danych robią się aktywne. Po skalowaniu aplikacja tworzy nowe pliki z danymi które kończą się na *.scale*. Warto zauważyć że zbiór trenujący i testujący muszą być skalowane razem, a więc przypadek w którym użytkownik wybiera zbiór trenujący, skaluję go, wybiera zbiór testujący i przeprowadza skalowanie ponownie może powodować nieprzewidywalne konsekwencje. Jeśli użytkownik planuje skalować dane i testować swój model to poprawnym podejściem jest wybranie zbiorów danych, a już za tym skalowanie danych.

Również jest dostępne skalowanie klas, które aktywuje się ptaszkiem w polu *Scale y*. Może to być przydatne w regresji, kiedy liczby w kolumnie *Y* (kolumna klas) nie są dyskretne. Za realizacją skalowania klas stoi wyżej wymieniony program *svm-scale*.

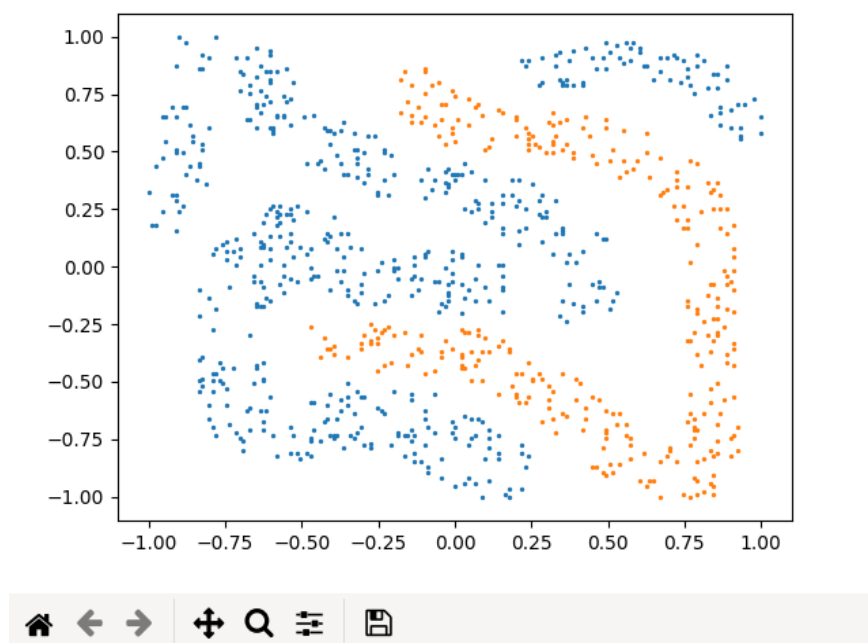
6.7 Wizualizacja danych

Karta *Visualization* 6.15 służy do wizualizacji jedno-, dwu- i trzy-wymiarowych danych.



Rysunek 6.14: Karta z kontrolkami dla wizualizacji danych

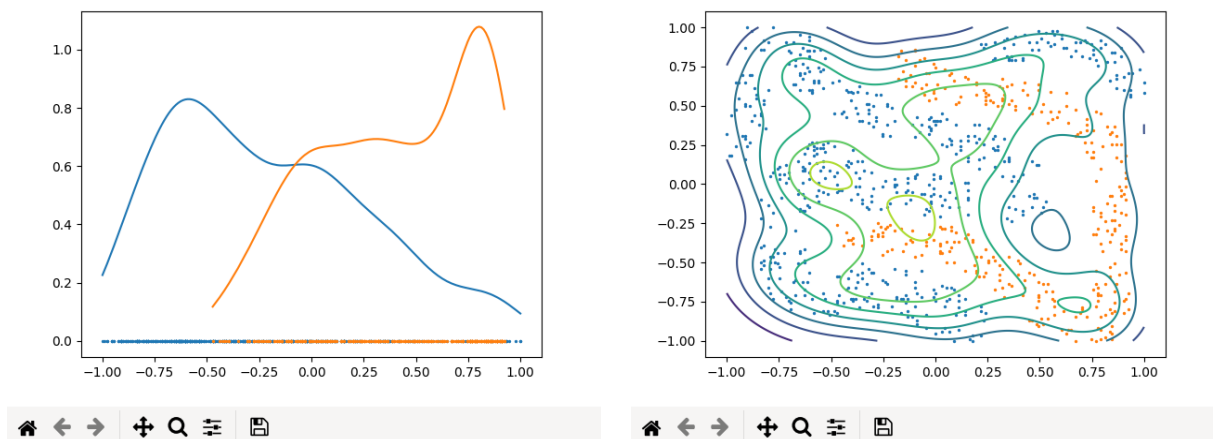
Kontrolki na karcie są dostępne tylko wtedy gdy jest wybrany zbiór trenujący z liczbą cech od 1 do 3.



Rysunek 6.15: Przykład wizualizacji dwuwymiarowych danych

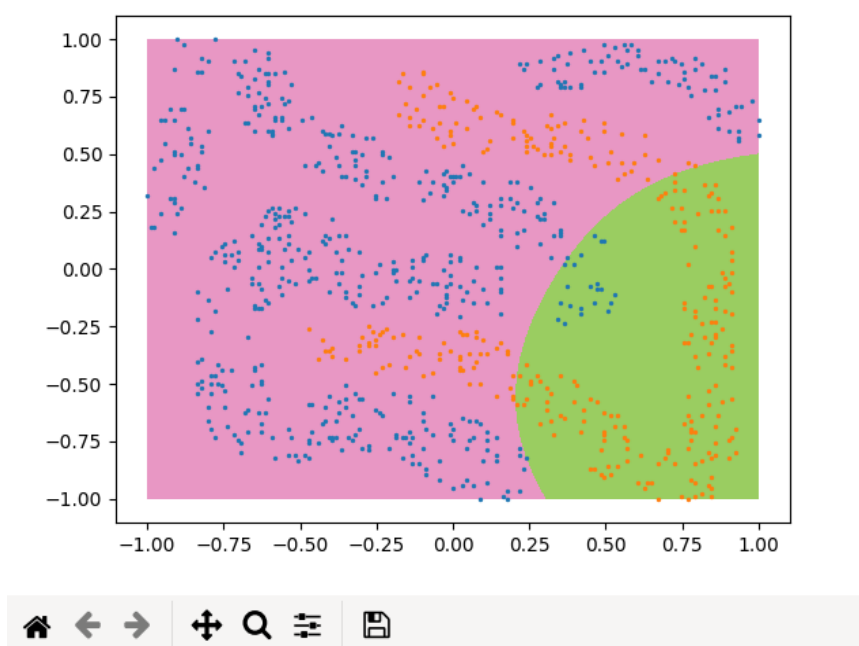
Dla jedno- i dwu-wymiarowych danych jest dostępne rysowanie gęstości danych. Algorytm rysujący

dzieli pole wykresu na kwadraty i zlicza liczbę punktów w każdym z nich. Na podstawie tego jest generowany konturowy wykres gęstości. Za liczbę kwadratów odpowiada wartość *Bandwidth*, która może być ustawiana przez użytkownika na interfejsie graficznym. Im mniejszy jest *Bandwidth* tym bardziej gładki wychodzi wykres, ale razem z tym zwiększa się obciążenie algorytmu rysującego.



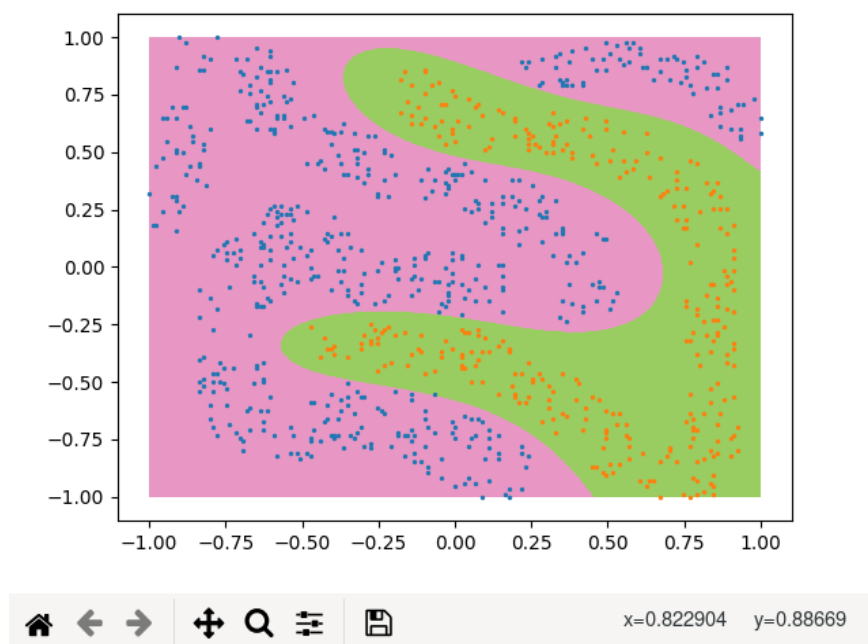
Rysunek 6.16: Przykład rysowania gęstości jedno- i dwu-wymiarowych danych

Dla dwuwymiarowych danych jest dostępne rysowanie modelu.



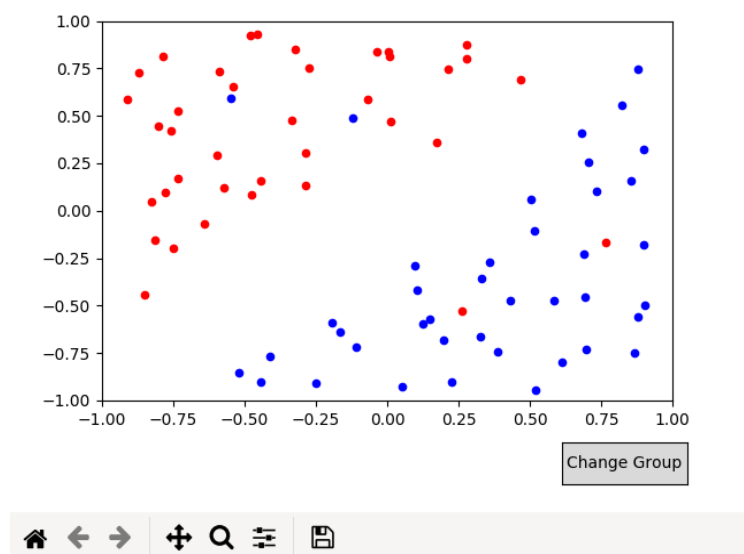
Rysunek 6.17: Przykład rysowania modelu

Dla wyżej narysowanego modelu 6.17 już z wykresu widać że parametry dobrane kiepsko. Dobierając odpowiednie parametry i skalując dane można osiągnąć lepsze wyniki:



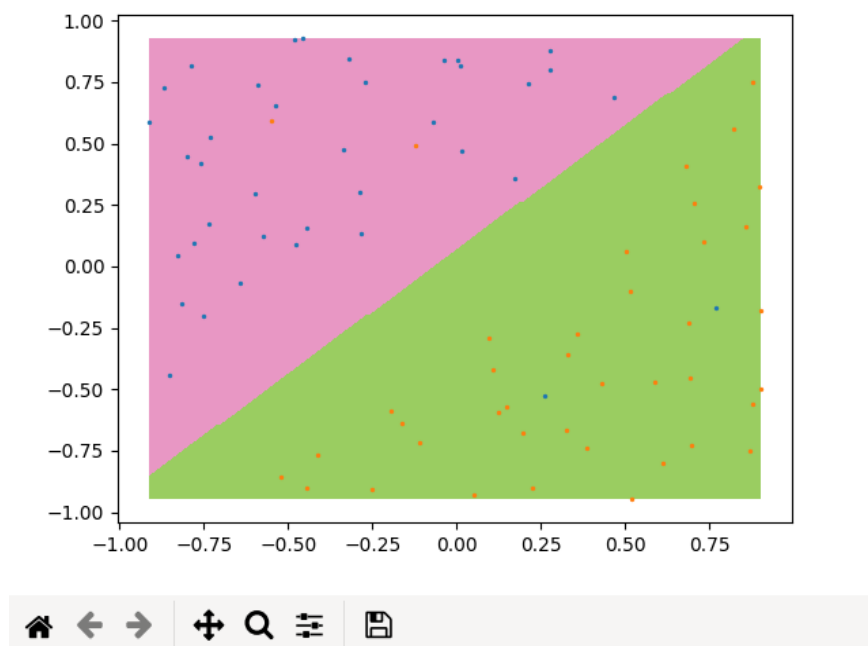
Rysunek 6.18: Przykład rysowania modelu

Przycisk *Gen Points* obok kontrolki do wybierania plików z danymi pozwala samodzielnie generować zbiór danych. Najpierw trzeba będzie wybrać nazwę pliku i katalog w którym należy go umieścić. Za tym pojawi się okno z pustym wykresem, klikając po nim użytkownik będzie generował punkty na tej płaszczyźnie, przycisk *Change Group* pozwala zmienić klasę aktualnie umieszczanych punktów.



Rysunek 6.19: Przykład ręcznego generowania danych

Wygenerowany model z liniową funkcją jądrową:



Rysunek 6.20: Model z ręcznie wygenerowanych danych

Dla rysowania wykresów był napisany skrypt w języku Python *plotter.py*. Wykresy gęstości pomaga rysować funkcja *gaussian_kde* z biblioteki *scipy*. Żeby narysować model skrypt generuje siatkę punktów na całym wykresie zapisując ich do tymczasowego pliku i przekazuje go do programu *svm-predict* z biblioteki *LIBSVM*, który w wyniku swojego działania zwraca swoje zgadnięcia odnośnie podanych punktów, co pozwala kolorować tło wykresu. Skutkiem takiego podejścia jest 'kąciastość' linii rozdzielającej klasy. Wartość *Bandwidth* pozwala kontrolować liczbę punktów w generowanej siatce i robić bardziej gładką linię.

6.8 Wybór cech

Czasami jest potrzeba wybrać tylko niektóre cechy z zbioru danych. Dla takich potrzeb w aplikacji istnieje funkcjonalność wyboru cech na karcie *Feature Selection* 6.21.

Rysunek 6.21: Karta z kontrolkami do wyboru cech

Na karcie jest umieszczone pole typu *Text Edit*, w które użytkownik musi wprowadzić które cechy on chce wybrać. Domyślnie w polu jest wpisany przedział cech w wczytanym pliku. Użytkownik może zmienić ten przedział lub wskazać konkretne cechy do wyboru, założmy że wczytany zbiór danych ma 30 cech, wtedy:

```
1-10           # wybór cech od 1 do 10
1-5,10-25      # wybór cech od 1 do 5 i od 10 do 25
1-5,7,10,15-20,25 # wybór cech od 1 do 5, 7 i 10, od 15 do 20 i 25
```

Przycisk *Select Features* uruchamia skrypt w języku Python *f_select.py* który z wybranych zbiorów danych wybiera cechy i zapisuje ich w formacie *LIBSVM* do plików z nazwą kończąca się na *.fseleced*. Analogicznie do skalowania jeśli użytkownik chce swoje dane testować to przy wyborze cech musi być wybrany odpowiedni plik z danymi testującymi.

6.9 Zmiana formatu danych na *LIBSVM*

W Internecie trudno znaleźć dane przygotowane w formacie *LIBSVM*, większość z nich jest w formacie CSV. Natomiast algorytm trenujący biblioteki *LIBSVM* nie przyjmuje danych w innych formatach. Żeby dać swobodę użytkownikom i pozbawić ich od potrzeby pisania własnych programów dla konwersji została dodana prosta funkcjonalność umieszczona na karcie *Format* 6.22 pozwalająca konwertować dane do formatu *LIBSVM*. Użytkownik musi podać separator pomiędzy

cechami i określić czy cecha jest na początku lub końcu rekordu. Niestety nie ma opcji żeby liczba określająca klasę znajdowała się pomiędzy liczbami oznaczające cechy. Również trzeba określić separator dziesiętny liczb rzeczywistych(kropka lub przecinek).

line 1: label ?
63,1,3,145,233,1,0,150,0,2.3,0,0,1,1 is not a valid multi-label form
Found 1 lines with error.

Select Train Dataset Gen Data /home/pavlo/pr/dw/datasets/heart.csv

Select Test Dataset ...

Model File Save As /home/pavlo/pr/dw/datasets/svmguide1.txt.model

CV/Holdout Scaling Visualization Feature Selection **Format** Grid Search

Feature separator: Convert

Decimal separator: dot ☐ Label at the end

SVM Type C-SVC

Kernel RBF

degree 3 C 1 P 0.1

gamma 0 Epsilon 0.001 ☒ Shrinking

coef0 0 Nu 0.5 ☐ Est. Prob.

Validation 20 %

Train Model

Test Model

Clear

Rysunek 6.22: Karta z kontrolkami do formatowania danych

Po ustawieniu wszystkich parametrów formatowania i kliknięciu przycisku *Convert* aplikacja uruchamia skrypt w języku Python *convert2svm.py* który dzięki podanemu separatorowi analizuje podany plik i uклада dane w formacie *LIBSVM*.

Na przykład to są 5 pierwszych linii ze zbioru danych w formacie CSV:

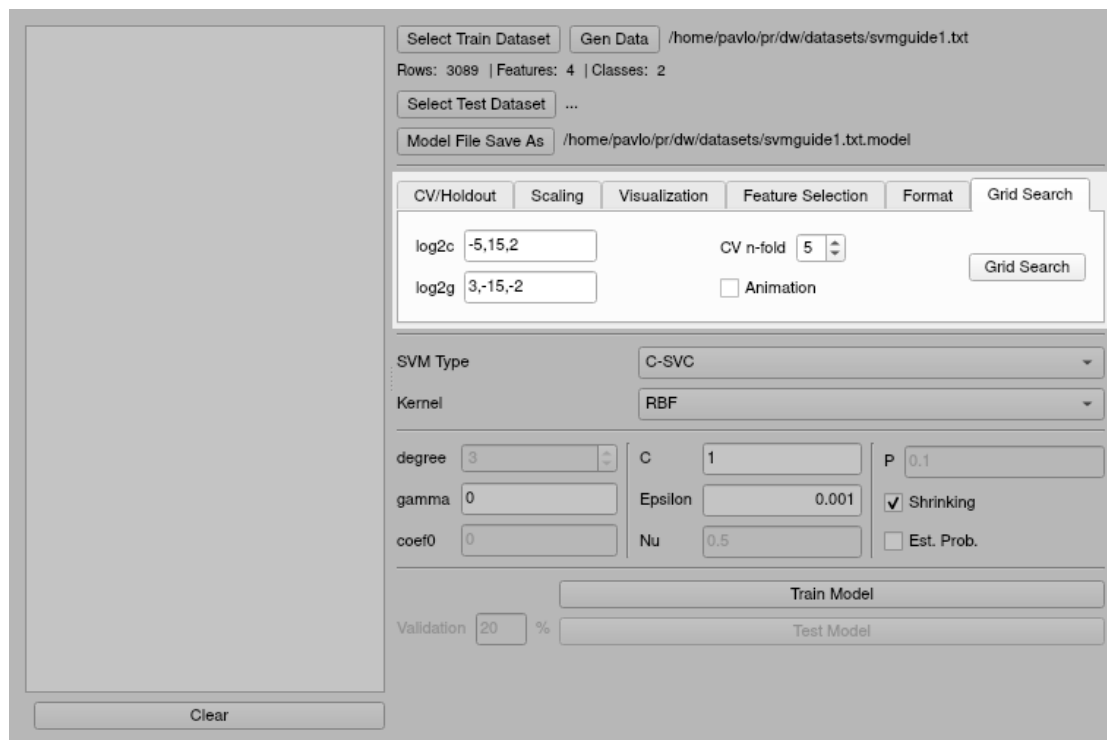
```
63,1,3,145,233,1,0,150,0,2.3,0,0,1,1
37,1,2,130,250,0,1,187,0,3.5,0,0,2,1
41,0,1,130,204,0,0,172,0,1.4,2,0,2,1
56,1,1,120,236,0,1,178,0,0.8,2,0,2,1
57,0,0,120,354,0,1,163,1,0.6,2,0,2,1
```

Te same dane po formatowaniu(ustawiono flagę że liczby określające klasy są na końcach linii):

```
1 1:63 2:1 3:3 4:145 5:233 6:1 7:0 8:150 9:0 10:2.3 11:0 12:0 13:1
1 1:37 2:1 3:2 4:130 5:250 6:0 7:1 8:187 9:0 10:3.5 11:0 12:0 13:2
1 1:41 2:0 3:1 4:130 5:204 6:0 7:0 8:172 9:0 10:1.4 11:2 12:0 13:2
1 1:56 2:1 3:1 4:120 5:236 6:0 7:1 8:178 9:0 10:0.8 11:2 12:0 13:2
1 1:57 2:0 3:0 4:120 5:354 6:0 7:1 8:163 9:1 10:0.6 11:2 12:0 13:2
```

6.10 Optymalizacja parametrów

Przeszukiwanie siatką(ang. grid search) jest algorytmem pozwalającym na dobór parametrów modelu. Biblioteka *LIBSVM* dostarcza skrypt w języku Python *grid-search* dla parametrów C i γ który był zintegrowany z aplikacją. Probuje on różne kombinacje (C, γ) (działa w oparciu o funkcje jądrową RBF) i wypisuje tę dla której walidacja krzyżowa dała najlepszy wynik.



Rysunek 6.23: Karta z kontrolkami do przeszukiwania siatką

Na karcie *Grid Search* znajdują się dwa pola tekstowych pozwalających na ustawienie zakresów parametrów i kroków algorytmu. W pole należy wpisać trzy liczby: początek, koniec i krok. Zakres próbowanych parametrów więc będzie wyglądał tak: $2^{begin, \dots, begin+k*step, \dots, end}$. Przeszukiwanie siatką pomaga wybrać potrzebne parametry nie zgadując i nie wprowadzając ręcznie każdej pary, wadą jednak jest że proces jest dość wolny.

Na karcie *Grid Search* znajduje się kontrolka typu *Check Box* pozwalająca włączyć animację procesu przeszukiwania jeśli na używanym systemie jest skonfigurowana aplikacja *gnuplot*. Również można wybrać krotność walidacji krzyżowej używanej dla testowania każdej pary parametrów.

6.11 Shrinking Kurczenie się????

W sekcji wyboru parametrów trenowania jest umieszczona kontrolka typu *Check Box* która pozwala włączyć kurczenie się(ang. shrinking) **dobrze tłumaczenie na polski?** dla algorytmu trenującego. Metoda polega na wyeliminowaniu elementów ograniczonych(np. $\alpha_i = 0$ lub C) dla zmniejszenia rozwiązywanego problemu [1].

Kurczenie się nie ma wpływu na dokładność ale pomaga zmniejszyć czas trenowania modelu, efekt jest bardziej zauważalny przy dużej liczbie iteracji [8].

6.12 Estymacja prawdopodobieństwa

W sekcji wyboru parametrów jest umieszczona kontrolka typu *Check Box* która pozwala estymować prawdopodobieństwo(ang. Probability estimate) z którą dany rekord należy do wybranej przez algorytm testujący klasy. Jeśli opcja była zaznaczona pod czas trenowania to wytrenowany model ma możliwość estymacji wiarygodności pod czas testowania. Po testowaniu aplikacja stworzy plik o nazwie *[plik testujący].prob_est* zawierający informacje o przydzielonych klasach i prawdopodobieństwach.

Założmy że model jest już wytrenowany a plik z danymi do testów wygląda następująco:

```
0 1:1.474250e+01 2:3.138960e+01 3:5.913974e-01 4:1.532451e+02
0 1:1.683200e+01 2:2.764799e+01 3:4.166706e-01 4:4.596750e+01
0 1:1.653200e+01 2:4.938600e+01 3:-2.416226e-01 4:3.655207e+01
1 1:7.794800e+01 2:1.936780e+02 3:1.584834e-01 4:1.222632e+02
1 1:5.024301e+01 2:3.121110e+02 3:-1.666690e-01 4:1.799808e+02
1 1:3.479100e+01 2:1.253760e+02 3:1.451816e-01 4:1.062559e+02
```

W pierwszej kolumnie są rzeczywiste klasy dla tych rekordów, następne są cechami. Po testowaniu z włączoną opcją estymacji prawdopodobieństwa plik *.prob_est* może mieć taki wygląd:

```
labels 1 0
1 0.772481 0.227519
0 1e-07 1
0 0.0431718 0.956828
1 0.816153 0.183847
1 0.816032 0.183968
1 0.816686 0.183314
```

W pierwszej kolumnie jest klasa przydzielona przez model odpowiedniemu rekordowi z pliku testującego. W drugiej kolumnie jest prawdopodobieństwo tego że dany rekord należy do klasy 1, a w trzeciej że do klasy 0.

Warto wspomnieć że opcja estymacji prawdopodobieństwa wymaga dodatkowych obliczeń, *LIB-SVM* wewnętrznie przeprowadza walidację krzyżową, więc zaleca się unikać używania estymacji prawdopodobieństwa razem z walidacją krzyżową [8].

7 Kompilacja i zależności

7.1 Zależności

Aplikacja była testowana na trzech systemach operacyjnych: *OS X*, *Debian 10* i *Manjaro Linux 19.0.2*. Lista zależności jest dla systemu *Debian 10* skoro jest on najbardziej dostępny i najpopularniejszy z wyżej wymienionych.

Zależności systemowe: *git*, *qt5-default*, *python3-pip* i opcjonalnie *gnuplot* (udostępnia animacje przy przeszukiwaniu siatki). Paczki można zainstalować poleceniem:

```
apt-get install git qt5-default python3-pip gnuplot
```

lub za pomocą wybranego menedżera paczek.

Zależności dla *Python 3*: *numpy*, *scipy*, *matplotlib*. Paczki można zainstalować poleceniem:

```
pip3 install numpy scipy matplotlib
```

7.2 Kompilacja

1. Ściągnąć kod projektu z repozytorium:

```
git clone https://github.com/boidachenkop/svm-app.git
```

2. Przejść do katalogu z kodem źródłowym:

```
cd svm-app/svm-app
```

3. Uruchomić kompilację:

```
make
```

4. Skompilować osobno pliki *svm-train.c* i *svm-predict.c*:

```
g++ -Wall -Wconversion -O3 -fPIC svm-predict.c svm.o -o svm-predict -lm  
g++ -Wall -Wconversion -O3 -fPIC svm-train.c svm.o -o svm-train -lm
```

8 Przykłady działania aplikacji

8.1 Przykład 1

Przykład pokazujący pracę z plikiem z nieprzeskalowanymi danymi w formacie *LIBSVM*

Krok 1: Wczytać plik z danymi

The screenshot displays the SVM application interface. On the left is a large empty plot area with a 'Clear' button at the bottom. The right panel contains the following elements:

- Dataset Selection:** 'Select Train Dataset' and 'Select Test Dataset' buttons are highlighted. The 'Gen Data' button is also visible. Below them, the file paths and statistics are shown: Train Dataset: /home/pavlo/pr/dw/datasets/svmguide1.txt (Rows: 3089, Features: 4, Classes: 2); Test Dataset: /home/pavlo/pr/dw/datasets/svmguide1.test (Rows: 4000, Features: 4, Classes: 2). A 'Model File Save As' button is set to /home/pavlo/pr/dw/datasets/svmguide1.txt.model.
- Navigation Tabs:** 'CV/Holdout', 'Scaling', 'Visualization', 'Feature Selection', 'Format', and 'Grid Search' are present. 'Scaling' is the active tab.
- Scaling Parameters:** Includes 'Lower Limit' (-1.000), 'Upper Limit' (1.000), 'Scale' button, 'y Lower Limit' (-1.000), 'y Upper Limit' (1.000), and a 'Scale y' checkbox.
- SVM Configuration:** 'SVM Type' is set to 'C-SVC' and 'Kernel' is set to 'RBF'. Parameters include 'degree' (3), 'gamma' (0), 'coef0' (0), 'C' (1), 'Epsilon' (0.001), 'Nu' (0.5), and 'p' (0.1). The 'Shrinking' checkbox is checked, and 'Est. Prob.' is unchecked.
- Buttons:** 'Train Model' and 'Test Model' buttons are at the bottom right. A 'Validation' dropdown is set to '20 %'.

Pojawiły się ścieżki z plikami, a pod nimi informacje o wczytanych zbiorach danych, więc dane wczytane poprawnie.

Krok 2: Wyskalować dane do przedziału $[-1, 1]$

Select Train Dataset: /home/pavlo/pr/dw/datasets/svmguide1.txt.scale
Rows: 3089 | Features: 4 | Classes: 2

Select Test Dataset: /home/pavlo/pr/dw/datasets/svmguide1.test.scale
Rows: 4000 | Features: 4 | Classes: 2

Model File Save As: /home/pavlo/pr/dw/datasets/svmguide1.txt.scale.model

CV/Holdout | **Scaling** | Visualization | Feature Selection | Format | Grid Search

Lower Limit: -1.000 | Upper Limit: 1.000 | Scale

y Lower Limit: -1.000 | y Upper Limit: 1.000 | ☐ Scale y

SVM Type: C-SVC
Kernel: RBF

degree: 3 | C: 1 | P: 0.1
gamma: 0 | Epsilon: 0.001 | ☒ Shrinking
coef0: 0 | Nu: 0.5 | ☐ Est. Prob.

Validation: 20 % | Train Model | Test Model

Clear

Do nazw plików z danymi dodano rozszerzenie *.scale*, więc dane są wyskalowane.

Krok 3: Przeprowadzić przeszukiwanie siatką, żeby dobrać pasujące parametry.

[local] 9.0 -3.0 96.7627 (best c=2.0, g=2.0, rate=96.9893)
[local] 3.0 -3.0 96.2447 (best c=2.0, g=2.0, rate=96.9893)
[local] 15.0 -3.0 96.9893 (best c=2.0, g=2.0, rate=96.9893)
[local] -5.0 -3.0 84.7847 (best c=2.0, g=2.0, rate=96.9893)
[local] 7.0 -3.0 96.6656 (best c=2.0, g=2.0, rate=96.9893)
[local] 1.0 -3.0 95.9858 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 -7.0 96.3095 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 -1.0 96.3742 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 -13.0 95.4678 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 1.0 96.2771 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 -11.0 95.5002 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 -5.0 96.7303 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 -15.0 95.2736 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 3.0 94.6585 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 -9.0 96.1476 (best c=2.0, g=2.0, rate=96.9893)
[local] 13.0 -3.0 96.7951 (best c=2.0, g=2.0, rate=96.9893)
2.0 2.0 96.9893

Select Train Dataset: /home/pavlo/pr/dw/datasets/svmguide1.txt.scale
Rows: 3089 | Features: 4 | Classes: 2

Select Test Dataset: /home/pavlo/pr/dw/datasets/svmguide1.test.scale
Rows: 4000 | Features: 4 | Classes: 2

Model File Save As: /home/pavlo/pr/dw/datasets/svmguide1.txt.scale.model

CV/Holdout | Scaling | **Visualization** | Feature Selection | Format | Grid Search

log2c: -5,15,2 | CV n-fold: 5 | Grid Search

log2g: 3,-15,-2 | ☐ Animation

SVM Type: C-SVC
Kernel: RBF

degree: 3 | C: 1 | P: 0.1
gamma: 0 | Epsilon: 0.001 | ☒ Shrinking
coef0: 0 | Nu: 0.5 | ☐ Est. Prob.

Validation: 20 % | Train Model | Test Model

Clear

Algorytm przeszukiwania siatką zwrócił że przy parametrach $C = 2$, $gamma = 2$ precyzja modelu będzie około 96%.

Krok 4: Wytrenować i przetestować model dla parametrów zwróconych algorytmem optymalizującym.

The screenshot displays the SVM application interface. On the left, a log window shows the results of the optimization process, including the best parameters found (c=2.0, g=2.0) and the final accuracy (96.875%). The main panel on the right contains various controls for dataset selection, model configuration, and training/testing actions.

Dataset Selection:

- Select Train Dataset:** /home/pavlo/pr/dw/datasets/svmguide1.txt.scale
Rows: 3089 | Features: 4 | Classes: 2
- Select Test Dataset:** /home/pavlo/pr/dw/datasets/svmguide1.test.scale
Rows: 4000 | Features: 4 | Classes: 2
- Model File Save As:** /home/pavlo/pr/dw/datasets/svmguide1.txt.scale.model

Grid Search Parameters:

- log2c:** -5,15,2
- log2g:** 3,-15,-2
- CV n-fold:** 5
- Animation:** ☐

SVM Configuration:

- SVM Type:** C-SVC
- Kernel:** RBF
- degree:** 3
- gamma:** 2
- coef0:** 0
- C:** 2
- Epsilon:** 0.001
- Nu:** 0.5
- P:** 0.1
- Shrinking:** ☒
- Est. Prob.:** ☐

Buttons:

- Train Model** (yellow button)
- Test Model** (yellow button)
- Clear** (button at the bottom left)

Validation: 20 %

Wytrenowany model znajduje się w ścieżce podanej naprzeciwko przycisku *Model File Save as*.

8.2 Przykład 2

Przykład z danymi w formacie *CSV*.

Krok 1: Wczytać plik z danymi.

line 1: label ?
63,1,3,145,233,1,0,150,0,2,3,0,0,1,1 is not a valid multi-label form
Found 1 lines with error.

Select Train Dataset Gen Data /home/pavlo/pr/dw/datasets/heart.csv

Select Test Dataset ...

Model File Save As ...

CV/Holdout Scaling Visualization Feature Selection Format Grid Search

log2c -5,15,2 CV n-fold 5

log2g 3,-15,-2 Animation

Grid Search

SVM Type C-SVC

Kernel RBF

degree 3 C 1 P 0.1

gamma 0 Epsilon 0.001 Shrinking

coef0 0 Nu 0.5 Est. Prob.

Validation 20 %

Train Model

Test Model

Clear

Ścieżka do wczytanego pliku jest czerwonego koloru, co sygnalizuje o błędzie, komunikat w polu tekstowym mówi że wczytane dane nie są w formacie *LIBSVM*.

Krok 2: Skonwertować dane do formatu *LIBSVM*.

line 1: label ?
63,1,3,145,233,1,0,150,0,2,3,0,0,1,1 is not a valid multi-label form
Found 1 lines with error.

Select Train Dataset Gen Data /home/pavlo/pr/dw/datasets/heart.csv

Select Test Dataset ...

Model File Save As ...

CV/Holdout Scaling Visualization Feature Selection Format Grid Search

Feature separator: , Convert

Decimal separator: dot Label at the end

SVM Type C-SVC

Kernel RBF

degree 3 C 1 P 0.1

gamma 0 Epsilon 0.001 Shrinking

coef0 0 Nu 0.5 Est. Prob.

Validation 20 %

Train Model

Test Model

Clear

Należy wprowadzić separator danych i zaznaczyć ptaszkiem czy liczba wskazująca klasę jest na początku lub końcu linii. W danym przypadku separatorem jest przecinek, a kolumna klas jest na końcu. Po kliknięciu na przycisk *Convert* program automatycznie wczyta plik z skonwertowanymi danymi i wypisze informację o nich.

Krok 3: Wyskalować dane do przedziału $[-1, 1]$.

The screenshot shows the SVM application interface. On the left, a text area displays an error message: "line 1: label ? 63,1,3,145,233,1,0,150,0,2,3,0,0,1,1 is not a valid multi-label form Found 1 lines with error." The top bar shows the "Select Train Dataset" button, "Gen Data" button, and the file path "/home/pavlo/pr/dw/datasets/heart.csv.libsvm.scale". Below this, it indicates "Rows: 303 | Features: 13 | Classes: 2". The "Select Test Dataset" button is also present. The "Model File Save As" field shows the path "/home/pavlo/pr/dw/datasets/heart.csv.libsvm.scale.model".

The main configuration panel has tabs for "CV/Holdout", "Scaling", "Visualization", "Feature Selection", "Format", and "Grid Search". The "Scaling" tab is active, showing "Lower Limit" set to -1.000 and "Upper Limit" set to 1.000. A "Scale" button is highlighted. Below these, there are fields for "y Lower Limit" (-1.000) and "y Upper Limit" (1.000), along with a "Scale y" checkbox. The "SVM Type" is set to "C-SVC" and the "Kernel" is set to "RBF". Other parameters include "degree" (3), "gamma" (0), "coef0" (0), "C" (1), "Epsilon" (0.001), "Nu" (0.5), and "P" (0.1). The "Shrinking" checkbox is checked, and the "Est. Prob." checkbox is unchecked. At the bottom, there are "Train Model" and "Test Model" buttons, and a "Validation" field set to 20%.

Skoro w danym zbiorze jest tylko 303 rekordy nie będziemy dzielić go na zbiór testujący i trenujący, a użyjemy walidacji krzyżowej. Do nazwy pliku z danymi było dodane *.scale*, więc dane są wyskalowane.

Krok 4: Uruchomić przeszukiwanie siatką, żeby dobrać odpowiednie parametry.

The screenshot shows the SVM application interface. On the left, a list of parameter combinations and their corresponding accuracy rates is displayed. The right panel shows the configuration for the grid search, including SVM Type (C-SVC), Kernel (RBF), and various hyperparameters.

Left Panel (Log Output):

```
[local] -15.0 -1.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 7.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 -9.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 3.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 -13.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 13.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 -3.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 5.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 -11.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 9.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 1.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 -15.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
[local] -15.0 15.0 54.4554 (best c=32.0, g=0.001953125, rate=83.1683)
32.0 0.001953125 83.1683
```

Right Panel (Configuration):

- Select Train Dataset: /home/pavlo/pr/dw/datasets/heart.csv.libsvm.scale
- Rows: 303 | Features: 13 | Classes: 2
- Select Test Dataset: ...
- Model File Save As: /home/pavlo/pr/dw/datasets/heart.csv.libsvm.scale.model
- CV/Holdout: CV n-fold 5
- log2c: -15,15,2
- log2g: 15,-15,-2
- Grid Search: [Grid Search]
- SVM Type: C-SVC
- Kernel: RBF
- degree: 3
- gamma: 0
- coef0: 0
- C: 1
- Epsilon: 0.001
- Nu: 0.5
- P: 0.1
- Shrinking: ☒
- Est. Prob.: ☐
- Validation: 20 %
- Train Model: [Train Model]
- Test Model: [Test Model]

Algorytm zwrócił parametry $C = 32$, $gamma = 0.001953$ na których 5-krotna walidacja krzyżowa pokazuje precyzję modelu w 83%. Mając bardziej szczegółową wiedzę o tym zbiorze danych np. co reprezentują cechy można było by spróbować wyrzucić te które nie dotyczą rozwiązywanego zadania i uzyskać większą dokładność modelu.

Krok 5: Wygenerować model.

The screenshot shows the SVM application interface after training. The left panel shows the optimization results. The right panel shows the configuration for the trained model.

Left Panel (Log Output):

```
**
optimization finished, #iter = 436
nu = 0.457971
obj = -4045.810685, rho = -0.559599
nSV = 148, nBSV = 132
Total nSV = 148
```

Right Panel (Configuration):

- Select Train Dataset: /home/pavlo/pr/dw/datasets/heart.csv.libsvm.scale
- Rows: 303 | Features: 13 | Classes: 2
- Select Test Dataset: ...
- Model File Save As: /home/pavlo/pr/dw/datasets/heart.csv.libsvm.scale.model
- CV/Holdout: CV n-fold 5
- log2c: -15,15,2
- log2g: 15,-15,-2
- Grid Search: [Grid Search]
- SVM Type: C-SVC
- Kernel: RBF
- degree: 3
- gamma: 19.53e-4
- coef0: 0
- C: 32
- Epsilon: 0.001
- Nu: 0.5
- P: 0.1
- Shrinking: ☒
- Est. Prob.: ☐
- Validation: 20 %
- Train Model: [Train Model]
- Test Model: [Test Model]

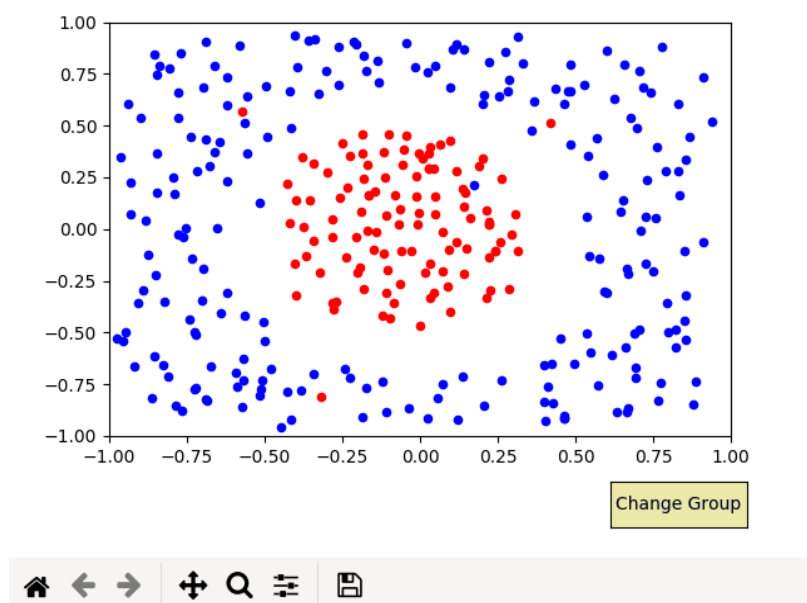
8.3 Przykład 3

Przykład reprezentujący pracę z ręcznie wygenerowanymi danymi.

Krok 1: Wcisnąć przycisk *Gen Data* i wybrać lokalizację i nazwę dla pliku z ręcznie wygenerowanymi danymi.

The screenshot shows the SVM application interface. On the left is a large empty white box. To its right is a control panel with several sections. At the top, there are three buttons: 'Select Train Dataset', 'Gen Data' (highlighted in yellow), and '...'. Below these are 'Select Test Dataset ...' and 'Model File Save As ...'. A tabbed interface follows, with tabs for 'CV/Holdout', 'Scaling', 'Visualization', 'Feature Selection', 'Format', and 'Grid Search'. The 'CV/Holdout' tab is active, showing input fields for 'log2c' (value: -5,15,2), 'log2g' (value: 3,-15,-2), 'CV n-fold' (value: 5), and an 'Animation' checkbox. A 'Grid Search' button is also present. Below the tabs, 'SVM Type' is set to 'C-SVC' and 'Kernel' is set to 'RBF'. A group of parameters includes 'degree' (3), 'gamma' (0), 'coef0' (0), 'C' (1), 'Epsilon' (0.001), 'Nu' (0.5), and 'P' (0.1). There are checkboxes for 'Shrinking' (checked) and 'Est. Prob.' (unchecked). At the bottom, there is a 'Validation' field set to 20%, and two buttons: 'Train Model' and 'Test Model'. A 'Clear' button is located at the bottom left of the interface.

Krok 2: Klikając na płaszczyźnie generować punkty. Dla zmiany klasy należy kliknąć w przycisk *Change Group*.



Staram się imitować klasyczny przykład na którym jest pokazywane działanie funkcji jądrowych. W kolejnym kroku nie ma potrzeby skalować dane skoro granice płaszczyzny na której generujemy dane są w przedziałach $x \in [-1, 1]$ i $y \in [-1, 1]$

Krok 3: W celu zadowolenia interesu użyjmy liniowej funkcji jądrowej z domyślnymi parametrami.

*
 optimization finished, #iter = 113
 nu = 0.683077
 obj = -222.000001, rho = -0.999998
 nSV = 224, nBSV = 220
 Total nSV = 224

Clear

Select Train Dataset

Gen Data

/home/pavlo/pr/dw/datasets/mydata.txt

Rows: 325 | Features: 2 | Classes: 2

Select Test Dataset ...

Model File Save As /home/pavlo/pr/dw/datasets/mydata.txt.model

CV/Holdout

Scaling

Visualization

Feature Selection

Format

Grid Search

log2c -5,15,2

CV n-fold 5

Grid Search

log2g 3,-15,-2

Animation

SVM Type C-SVC

Kernel Linear

degree 3

C 1

P 0.1

gamma 0

Epsilon 0.001

Shrinking

coef0 0

Nu 0.5

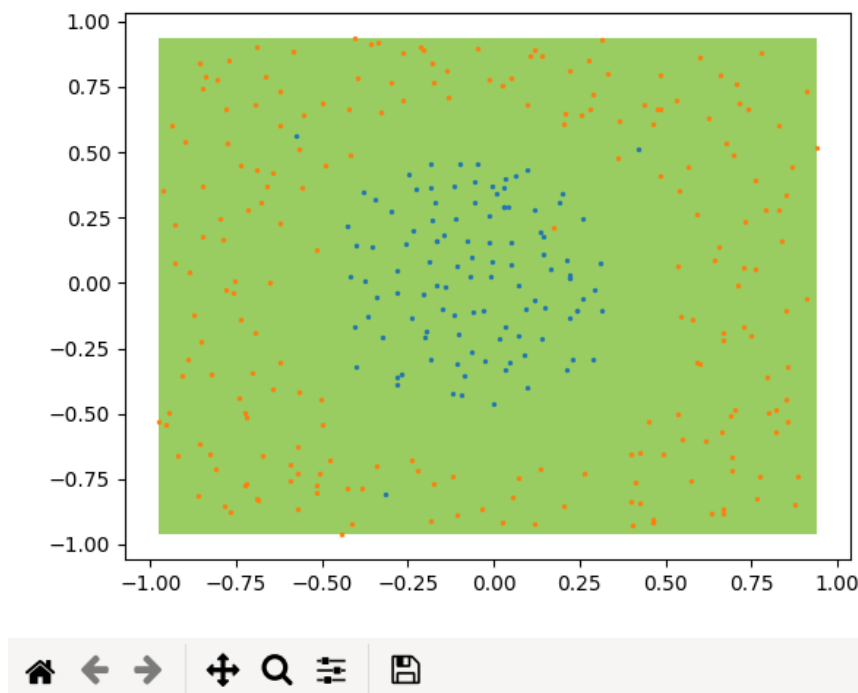
Est. Prob.

Train Model

Validation 20 %

Test Model

Krok 4: Po wygenerowaniu modelu można zwizualizować go.



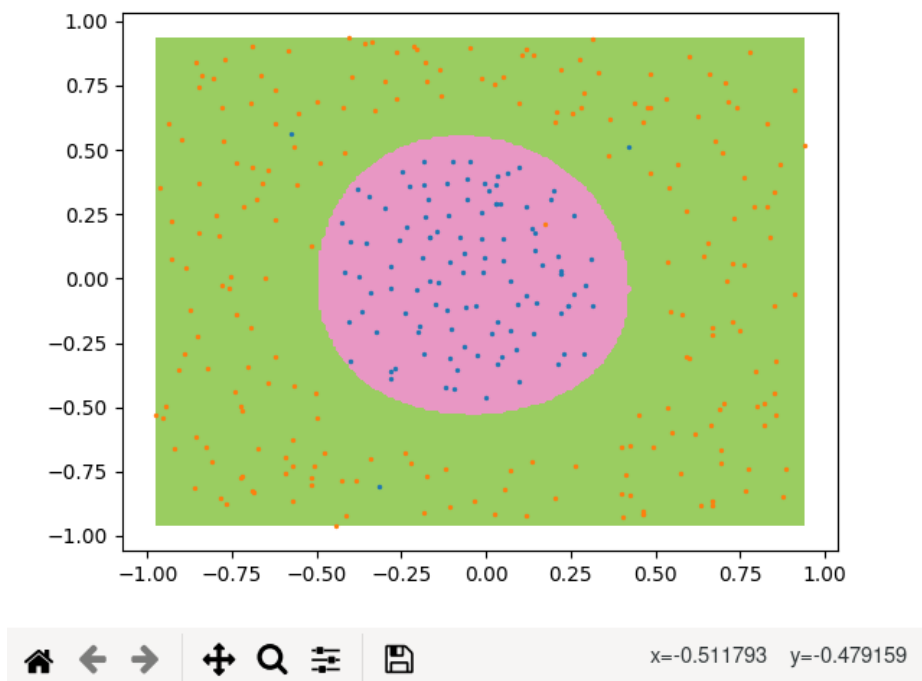
Algorytm *SVM* przydzielił wszystkie punkty do jednej klasy i według 5-krotnej walidacji krzyżowej uzyskał precyzję 65.84%. Jest to, oczywiście, kiepski model.

Krok 5: Wróćmy do funkcji jądrowej RBF i spróbujmy uruchomić przeszukiwanie siatką.

The screenshot shows the SVM software interface. On the left, a list of parameter combinations and their corresponding accuracy rates is displayed. The right pane shows the configuration for the Grid Search, including the SVM Type (C-SVC), Kernel (RBF), and various parameters like degree, gamma, C, Epsilon, and Nu. The Grid Search button is highlighted.

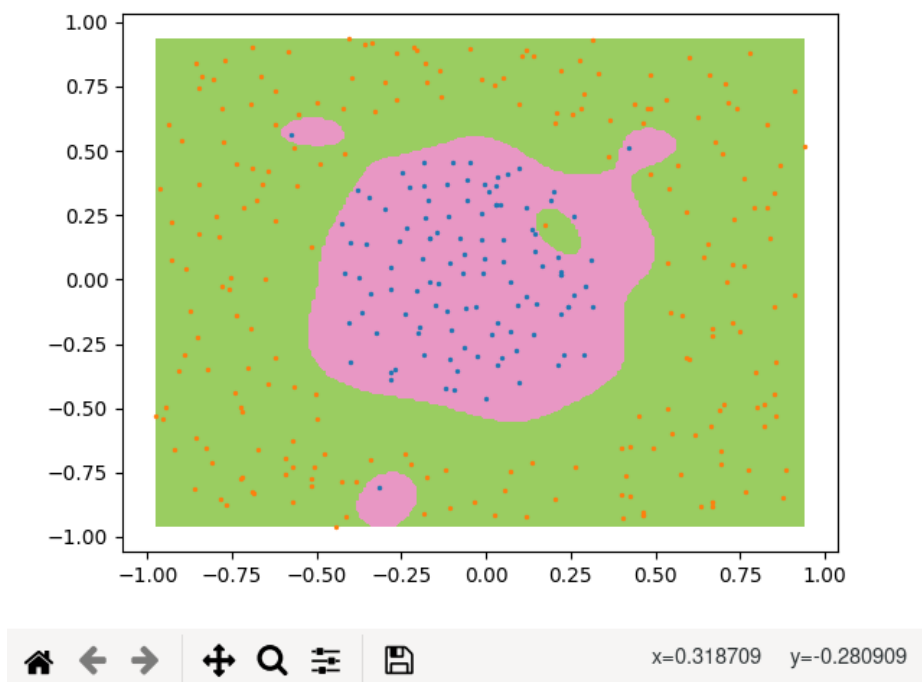
Algorytm proponuje użyć parametrów $C = 0.125$ i $\gamma = 8$ dzięki którym potrafimy uzyskać znacznie lepszą precyzję: 98.76%!

Krok 6: Po wygenerowaniu modelu można go zwizualizować.



Z wykresu widać że ten model jest znacznie lepszy. Algorytm poprawnie zidentyfikował szumy (punkty znajdujące się pomiędzy punktami z innej klasy) i nie widać przeuczania.

Krok 7: W celach zadowolenia interesu spróbujmy sztucznie wywołać zjawisko przeuczania, np. podając jako parametry $C = 500$ i $\gamma = 20$. Po wygenerowaniu modelu będzie on miał taki wygląd.



W tym przypadku model bierze pod uwagę punkty które były wprowadzone specjalnie w roli szumów, 5-krotna walidacja krzyżowa zwraca gorszy wynik: 95.69%

8.4 Przykład 4

Przykład opisujący pracę z zbiorem danych pokazanym na rysunkach 6.17 i 6.18.

Krok 1: Wczytać plik *fourclas_scale.txt* i wytrenować model z domyślnymi parametrami.

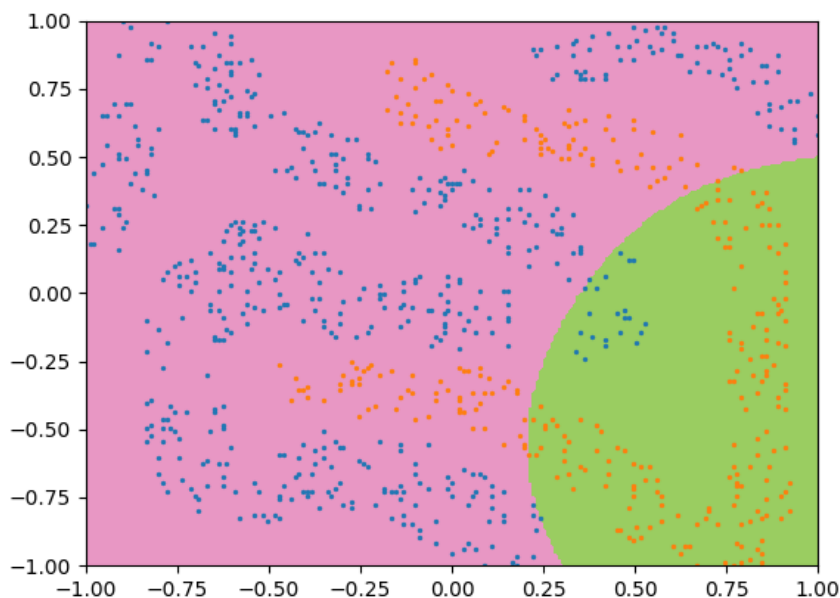
The screenshot shows the SVM application interface. On the left, a text box displays training results:


```

    *
    optimization finished, #iter = 318
    nu = 0.469505
    obj = -383.582676, rho =
    0.903354
    nSV = 408, nBSV = 401
    Total nSV = 408
    
```

 Below this is a 'Clear' button. The main panel has tabs for 'CV/Holdout', 'Scaling', 'Visualization', 'Feature Selection', 'Format', and 'Grid Search'. The 'Visualization' tab is active, showing 'Show Density' (unchecked) and 'Show Model' (checked). A 'Bandwidth' input field is set to 0.01, and a 'Visualize' button is present. Below the tabs, 'SVM Type' is set to 'C-SVC' and 'Kernel' is set to 'RBF'. Parameters include 'degree' (3), 'gamma' (0), 'coef0' (0), 'C' (1), 'Epsilon' (0.001), 'Nu' (0.5), and 'P' (0.1). Checkboxes for 'Shrinking' (checked) and 'Est. Prob.' (unchecked) are also visible. At the bottom, there is a 'Train Model' button, a 'Validation' dropdown set to 20%, and a 'Test Model' button.

Krok 2: Zwizualizować model.



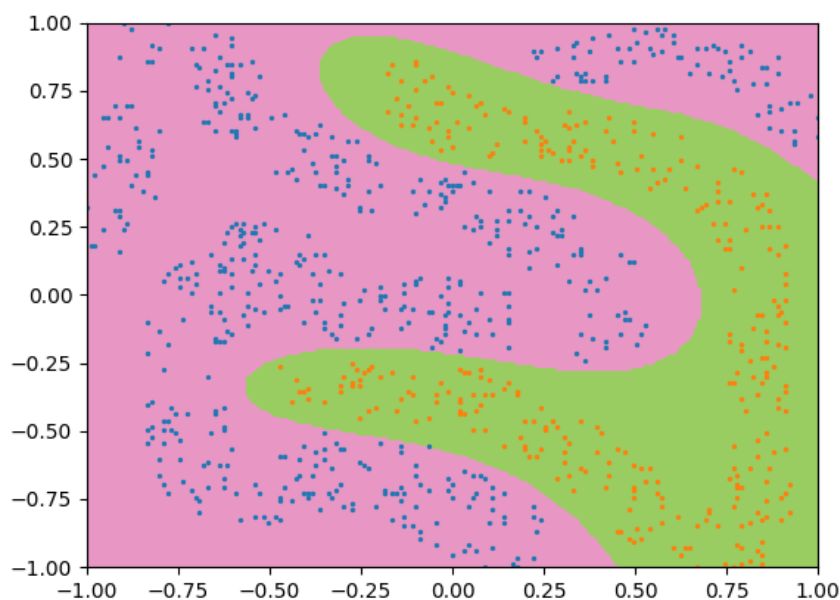
Widać że parametry dobrane kiepsko, trzeba znaleźć lepsze...

Krok 3: Uruchomić przeszukiwanie siatką(ang. Grid Search) żeby znaleźć lepsze parametry dla trenowania.

The screenshot shows the SVM application interface. On the left, a list of parameter combinations is displayed, each followed by its cross-validation score and the best parameters found. The central panel has tabs for CV/Holdout, Scaling, Visualization, Feature Selection, Format, and Grid Search. The Grid Search tab is active, showing a list of parameter combinations and a 'Grid Search' button. The bottom section contains fields for SVM Type (C-SVC), Kernel (RBF), degree (3), gamma (0), coef0 (0), C (1), Epsilon (0.001), Nu (0.5), P (0.1), and Shrinkage (checked). There are also buttons for 'Train Model' and 'Test Model'.

Algorytm zwrócił parametry $C = 32.0$, $gamma = 2.0$ dla których walidacja krzyżowa dała wynik w 100%.

Krok 4: Wytrenować i zwizualizować model dla znalezionych parametrów.



Wizualnie widać że klasy są odseparowane lepiej.

9 Podsumowanie

9.1 Rozwój aplikacji

Jeśli do aplikacji w przyszłości będzie dodawana nowa funkcjonalność to w każdym przypadku będzie konieczna zmiana schematu interfejsu graficznego. W trakcie pracy dodając nową funkcjonalność tworzyłem nową kartę na której umieszczałem odpowiednie kontrolki, kontynuacja tego może spowodować zaśmiecenie interfejsu przez co użytkownikom będzie trudno znaleźć kontrolki i meni których potrzebują. Obecnie aplikacja używa jednego okna za wyjątkiem okien systemowych dla wyboru plików i okien do wizualizacji danych, rozwiązaniem zaśmiecenia interfejsu może być użycie wielu okien. Na przykład osobne okno do wizualizacji danych, okno dla pracy z plikami i danymi, okno do trenowania i testowania modelu itp. Innym podejściem które wydaje się sensownym - praca w jednym oknie, ale zmiana jego treści w zależności od akcji użytkownika. Wadą, oczywiście, będzie to że użytkownik zobaczy dostępne kontrolki dopiero po akcji(np kontrolki do wizualizacji nie są umieszczane na ekranie do wczytania plików z danymi), przez co trudno będzie planować swoją pracę.

W przyszłości warto dodać wstępnie obliczone funkcje jądrowe(ang. *Precomputed kernel*). Biblioteka *LIBSVM* ma taką funkcjonalność, natomiast nie została ona zaimplementowana w obecnej wersji aplikacji. Polega ona na tym, że wartości funkcji jądrowej znajdują się w pliku trenującym.

W nauczaniu maszynowym dane często nie są zbalansowane czyli jedna klasa występuje częściej od innej. W przyszłości warto dodać funkcjonalność którą oferuje *LIBSVM*: ustawianie wag dla poszczególnych klas, co pozwala walczyć z niezbalansowanymi danymi.

Podział na pliki trenujące i testujące był zaimplementowany żeby odwzorować działanie biblioteki *LIBSVM*, natomiast w rzeczywistości użytkownik najczęściej ma jeden plik z danymi. Aplikacja może działać z jednym plikiem, używając walidacji krzyżowej lub wyboru danych do testowania z pliku trenującego(ang. *Holdout*), ale wydaje się sensowniejszym zrezygnować z założenia że użytkownik ma dwa pliki(testujący i trenujący) na korzyść założenia że użytkownik po prostu ma jakiś zbiór danych, nie koniecznie w formacie *LIBSVM*.

Istniejący konwerter danych do formatu *LIBSVM* jest bardzo naiwny, i zakłada że dane mają separator, a liczba oznaczająca klasę jest na początku lub końcu linii. W przyszłości warto zbadać najpopularniejsze formaty do zapisu danych i upewnić się że zaimplementowany konwerter sobie z nimi poradzi.

Na obecną chwilę aplikacja nie umie pracować z danymi w tekstowej postaci, w przyszłości warto to dodać.

Warto przerobić system wątków i uruchamiania skryptów. Teraz jest tak, że skrypty w języku *Python* uruchamiają się na tym samym wątku na którym jest obliczany interfejs graficzny, czyli w trakcie działania skryptu interfejs jest zamrażany. W przypadku skryptów które działają szybko(np. sprawdzenie poprawności danych) to nie ma znaczenia, użytkownik po prostu nie zauważy tego, natomiast skrypt przeszukiwania siatką może trwać dość długo(około 30-40 sekund). Tak długa nieaktywność interfejsu może naprowadzić użytkownika na myśl, że program się zawiesił i przestał działać. Rozwiązaniem może być uruchamianie skryptów w osobnych wątkach, a na interfejsie graficznym umieszczenie animacji oznaczającej że program coś liczy i nie trzeba go przerywać.

Można pomyśleć nad dodaniem innych bibliotek w oparciu o które może działać aplikacja, pozwoliło by to użytkownikom porównywanie wydajności różnych implementacji *SVM*.

Program ma dość sporo zależności dla uruchamianych skryptów. Należałoby zaimplementować system sprawdzający czy wszystkie zależności są zainstalowane i czy wszystkie skrypty są w katalogu

roboczym.

Źródła

- 1 Chih-Chung Chang i Chih-Jen Lin. „LIBSVM: A library for support vector machines”. W: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm> (cytowane na stronach 5, 6, 21, 43).
- 2 The Qt Company. *Qt Creator*. 2020. URL: <https://www.qt.io/development-tools> (cytowane na stronie 26).
- 3 The Qt Company. *Qt library*. 2020. URL: <https://www.qt.io/> (cytowane na stronie 19).
- 4 The Qt Company. *Qt Signals and Slots*. 2020. URL: <https://doc.qt.io/qt-5/signalsandslots.html> (cytowane na stronach 21, 22).
- 5 Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin i inni. „A practical guide to support vector classification”. W: (2003) (cytowane na stronie 14).
- 6 John D. Hunter. *Matplotlib: Python plotting*. URL: <https://matplotlib.org/> (cytowane na stronie 19).
- 7 S Sathiya Keerthi i Chih-Jen Lin. „Asymptotic behaviors of support vector machines with Gaussian kernel”. W: *Neural computation* 15.7 (2003) (cytowane na stronie 14).
- 8 *LIBSVM FAQ*. 2020. URL: <https://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html> (cytowane na stronach 43, 44).
- 9 Hsuan-Tien Lin i Chih-Jen Lin. „A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods”. W: *submitted to Neural Computation* 3 (2003) (cytowane na stronie 14).
- 10 Alexey Nefedov. „Support Vector Machines: A Simple Tutorial”. W: (2016). URL: <https://svmtutorial.online> (cytowane na stronach 6–9, 11).
- 11 *SVM - Support Vector Machines Software*. 2020. URL: http://www.support-vector-machines.org/SVM_soft.html (cytowane na stronie 19).
- 12 KB Воронцов. «Лекции по методу опорных векторов». русский. в: (2007). URL: <http://www.ccas.ru/voron/download/SVM.pdf> (цитируется на странице 7).