Patryk Jankowicz, Jan Walczak
Miłosz Kutyła, Jakub Ossowski

Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych

Sprawozdanie z realizacji projektu WBIO pt. System IDS wykorzystujący AIS, SOM i LVQ wersja 1.0

2025-03-02

Historia zmian

Wersja	Data	Autor	Opis zmian
1.0	20.05.2024	MK, JO, JW, PJ	Opisanie eksperymentów. Poprawki stylistyczne. Utworzenie podsumowania. (MK, JO, JW, PJ)
0.4	14.05.2024	MK, JO, JW, PJ	Opisanie implementacji i wskazanie podstawowej funkcjonalności. (MK, JO, JW, PJ)
0.3	13.05.2024	MK, JO, JW, PJ	Opracowanie sposobów oceny jakości. (MK, JO, JW, PJ)
0.2	08.04.2024	MK, JO, JW, PJ	Opracowanie wstępu teoretycznego do LVQ (JW). Rozwinięcie wstępu teoretycznego dot. algorytmów genetycznych w temacie AIS (MK, JO). Rozpoczęcie prac implementacyjnych (MK, JO, JW, PJ).
0.1	11.03.2024	MK, JO, JW, PJ	Pierwsza wersja raportu – faza Koncepcja. Wskazanie tematu i celu prac, podział obowiązków (MK, JO, JW, PJ). Dodanie pierwszej wersji wstępu teoretycznego dla AIS i SOM (MK).

Opracowane w ramach przedmiotu WBIO, 2024. Patryk Jankowicz, Jan Walczak, Jakub Ossowski, Miłosz Kutyła EiTI, PW.

Spis treści

Wstęp

Niniejszy dokument to sprawozdanie z realizacji projektu w ramach przedmiotu WBIO. Oświadczamy, że ta praca, stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu WBIO, została wykonana przez nas samodzielnie.

Projekt ma na celu stworzenie rozwiązania z zakresu cyberbezpieczeństwa, opartego na podejściu inspirowanym biologicznie.

1. Podział kompetencji

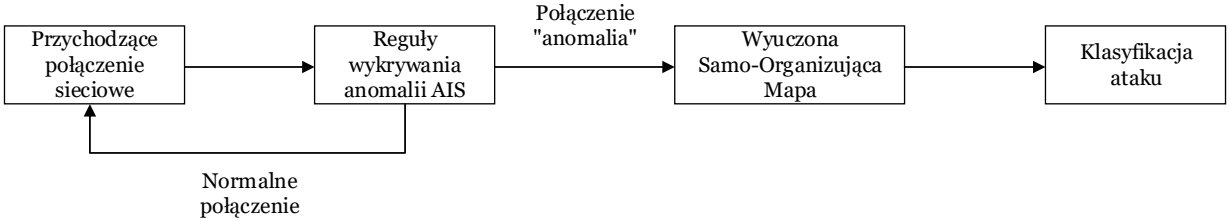
- Miłosz Kutyła: lider zespołu, odpowiedzialny za dokumentację, koordynację projektu oraz monitorowanie postępu prac. Dodatkowo zajmuje się wstępnym opracowaniem zagadnień teoretycznych związanych z AIS oraz ML/AI.
- Jakub Ossowski: udział w burzy mózgów, prowadzenie badań w celu znalezienia tematu i materiałów do projektu, a następnie implementacja rozwiązania.
- Jan Walczak: udział w burzy mózgów, prowadzenie badań w celu znalezienia tematu i materiałów do projektu, a także opracowanie oprogramowania fizycznej formy projektu, bazując na wcześniej przygotowanej dokumentacji teoretycznej i technicznej.
- Patryk Jankowicz: udział w burzy mózgów, prowadzenie badań w celu znalezienia tematu i materiałów do projektu, a także wsparcie zespołu na kolejnych etapach realizacji projektu.

2. Temat i cel projektu

Celem projektu jest zaprojektowanie systemu typu IDS (Intrusion Detection System), bazującego na mechanizmach inspirowanych funkcjonowaniem ludzkiego układu odpornościowego. Tradycyjne systemy IDS opierają się na zestawach reguł i algorytmach, które służą do wykrywania potencjalnych ataków na systemy komputerowe lub sieci. Wyróżnia się dwa główne klasyczne podejścia do implementacji systemów IDS:

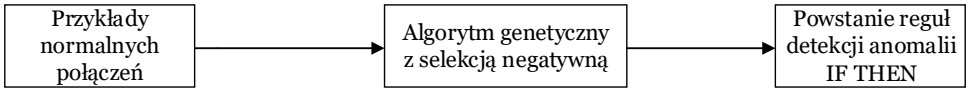
- podejście oparte na sygnaturach: IDS analizuje ruch sieciowy, pliki systemowe lub działania aplikacji w celu identyfikacji określonych wzorców, znanych jako sygnatury ataków. Sygnatury te są tworzone na podstawie wcześniej wykrytych zagrożeń, co uniemożliwia wykrycie nowych rodzajów ataków. Ponadto podejście to wymaga częstych i regularnych aktualizacji bazy sygnatur, aby umożliwić detekcję nowo odkrytych zagrożeń.
- podejście analityczne: IDS uwzględnia różnorodne parametry, na podstawie których analizuje zachowanie systemu w celu wykrycia odstępstw od normalnych wzorców.

Proponowane rozwiązanie będzie oparte na koncepcji Artificial Immune System (AIS), czyli systemach obliczeniowych inspirowanych immunologią i funkcjami ludzkiego układu odpornościowego, których zastosowanie obejmuje różnorodne problemy. Dodatkowo wykorzystana zostanie Samo-Organizująca Mapa (SOM) w celu identyfikacji wykrytych ataków. Schemat działania systemu IDS bazującego na AIS i SOM przedstawiono na rysunku 1.

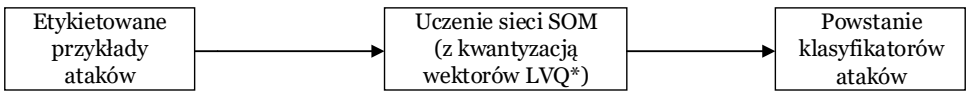


Rys. 1: Schemat działania IDS wykorzystującego AIS i SOM [1]

W systemie IDS mechanizm AIS pełni rolę pierwszej linii reakcji na przychodzące połączenia sieciowe, co wymaga, aby reguły AIS były opracowywane na podstawie analizy prawidłowego ruchu sieciowego. Z kolei zadaniem SOM jest klasyfikacja anomalii, które w domyśle oznaczają potencjalne ataki sieciowe. Aby zapewnić skuteczność, SOM musi być trenowana z wykorzystaniem etykietowanych danych, które reprezentują i klasyfikują różne rodzaje anomalii. Proces uczenia AIS został przedstawiony na rysunku 2, natomiast sposób uczenia SOM zilustrowano na rysunku 3. [1]



Rys. 2: AIS – procedura generowania reguł detekcyjnych



Rys. 3: SOM – procedura nauczania

3. Wstęp teoretyczny

3.1. Artificial Immune System (AIS)

Sztuczne układy odpornościowe (AIS) opierają swoją konstrukcję na mechanizmach inspirowanych działaniem ludzkiego układu odpornościowego. W kontekście systemów IDS szczególną uwagę zwraca zachowanie limfocytów T, które stanowią inspirację dla proponowanego rozwiązania. W uproszczeniu, limfocyty T umożliwiają układowi odpornościowemu identyfikację komórek niebezpiecznych dla organizmu. Ponieważ każdy limfocyt T może wiązać się wyłącznie z określonymi antygenami, konieczna jest duża ich różnorodność, aby skutecznie chronić organizm. Mechanizm ten, oparty na dopasowaniu wzorców, zostanie zastosowany w projektowanym IDS.

W projektowanym systemie IDS antygeny zostaną zastąpione wektorami wejściowymi, które będą reprezentować np. połączenia sieciowe. Wektory te opisują parametry poszczególnych połączeń. Detektory, czyli programistyczna implementacja limfocytów T, reagują na dany wektor wejściowy, jeśli spełnia on odpowiednie warunki dopasowania. Warunki te mogą być wyrażone jako równościowe lub nierównościowe (przedziałowe). W terminologii uczenia maszynowego detektor można rozumieć jako strukturę składającą się z selektorów:

- **Selektor s_i:** Warunek określający zbiór wartości dozwolonych dla danego atrybutu (parametru). Selektor jest spełniony, jeśli wartość atrybutu mieści się w zbiorze dozwolonym. Dopuszczalny jest selektor uniwersalny ("?", dopuszczający dowolną wartość) oraz selektor pusty ($\emptyset$, który nigdy nie jest spełniony).
- **Kompleks $\langle s_1, s_2, \ldots, s_n \rangle$:** Wektor selektorów dla poszczególnych atrybutów. Kompleks jest spełniony, gdy wszystkie selektory są spełnione jednocześnie, co oznacza, że jest to koniunkcja warunków określonych przez selektory.

Detektor aktywuje się w odpowiedzi na wektor wejściowy tylko wtedy, gdy odpowiadający mu kompleks jest spełniony. Jest to analogiczne do sposobu, w jaki limfocyty T reagują na odpowiedni antygen [1] [2].

Podobnie jak limfocyty T, detektory muszą zostać wytworzone w procesie produkcji – do tego celu zastosowany zostanie algorytm genetyczny selekcji negatywnej. Pseudokod tego algorytmu został przedstawiony poniżej [3]:

Algorytm 1 Pseudokod algorytmu selekcji negatywnej

```
1:  function GenerateDetectors(S) returns detectors
2:      Niech S będzie zbiorem próbek self-set
3:      Niech G będzie pustą listą detektorów
4:      while |G| ≠ n do
5:          P ← losujDetektor()
6:          Matched ← false
7:          for each element s z listy S do
8:              if P matches s then
9:                  Matched ← true
10:                 break
11:             end if
12:         end for
13:         if Matched = false then
14:             Add P to G
15:         end if
16:     end while
17: end function
```

gdzie:

- `losujDetektor()` jest funkcją, która generuje i zwraca losowo utworzony detektor. Detektor ten ma możliwość dopasowania do reprezentacji próbki znajdującej się w zbiorze treningowym przestrzeni własnej.

3.2. Self-Organizing Map (SOM)

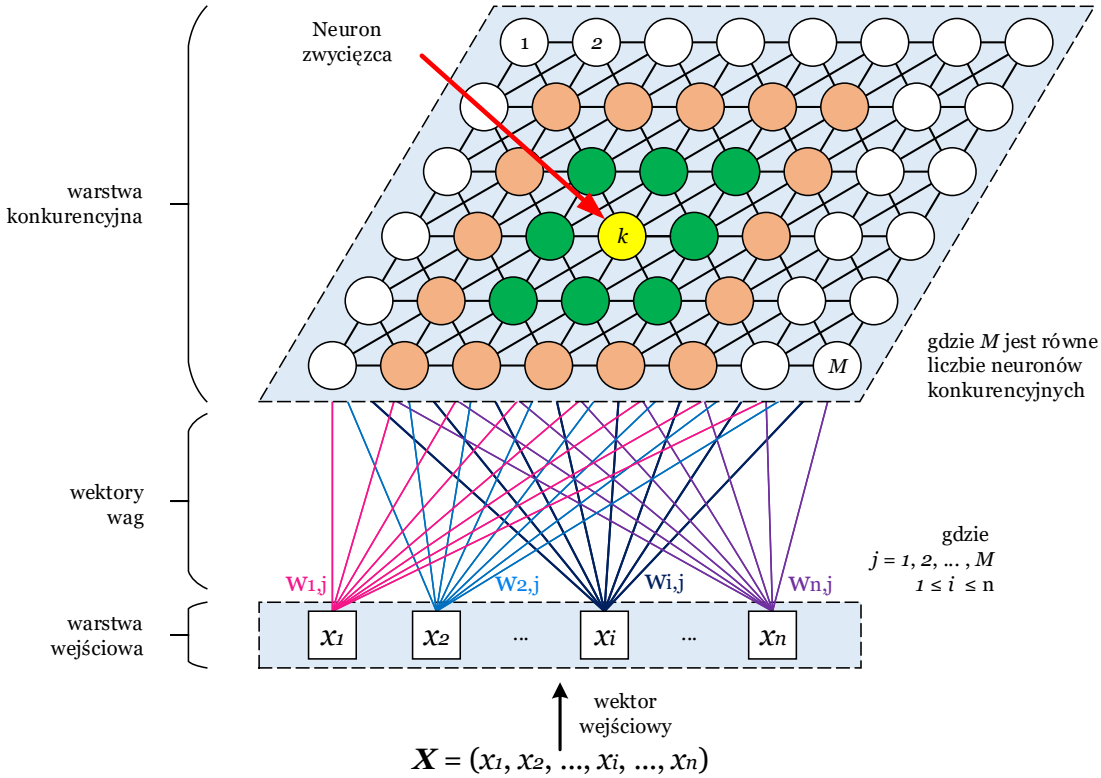
Ważnym elementem AIS jest SOM, czyli Samo-Organizująca Mapa (ang. *Self-Organizing Map*, znana również jako sieć Kohonena). SOM to sieć neuronowa, która jest trenowana przy użyciu reguł uczenia konkurencyjnego (ang. *competitive learning*) w sposób nienadzorowany (ang. *unsupervised learning*). Sieć SOM różni się od tradycyjnych sieci neuronowych, ponieważ składa się jedynie z dwóch warstw – wejściowej i konkurencyjnej (brak warstwy ukrytej, jak np. w modelu perceptronu wielowarstwowego):

- warstwa wejściowa nie wykonuje żadnych przekształceń danych, a jedynie przekazuje dane do neuronów w warstwie konkurencyjnej,
- warstwa konkurencyjna (znana również jako warstwa topologiczna, Kohonena) składa się z neuronów uporządkowanych w siatkę. Każdy z M neuronów w tej warstwie ma przypisany wektor wag $\mathbf{W}_j = (w_{1j}, w_{2j}, ..., w_{ij}, ..., w_{nj})$ dla $j = 1, 2, ..., M$, gdzie n to liczba wejść sieci SOM.

Uczenie konkurencyjne SOM polega na tym, że neurony rywalizują o reakcję na bodziec, którym jest wektor wejściowy. Neuron, który jest najbardziej pobudzony przez bodziec, tzn. którego wektor wag jest najbardziej zbliżony do wektora wejściowego, wygrywa konkurencję. Zwycięski neuron uzyskuje prawo do reagowania na ten bodziec w przyszłości, dostosowując swój wektor wag zgodnie z regułą uczenia. Aktualizacja wzmocnia reakcję zwycięskiego neuronu na ten bodziec, co skutkuje tym, że przy ponownym pojawieniu się tego samego wektora, ten sam neuron będzie pobudzony. Istnieją dwa podejścia do aktualizacji [4]:

- WTA (ang. *Winner Takes All*) – wektor wag jest aktualizowany tylko dla neuronu zwycięskiego,
- WTM (ang. *Winner Takes Most*) – wektory wag są aktualizowane w sąsiedztwie neuronu zwycięskiego, proporcjonalnie do odległości od niego.

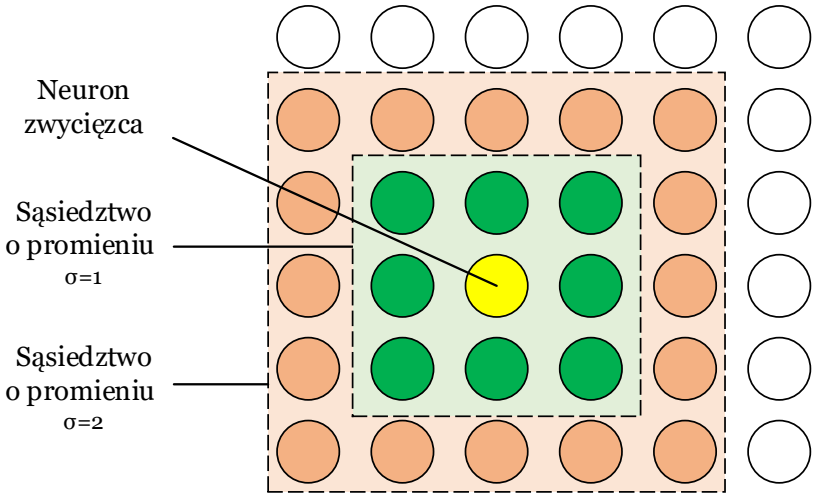
Przykład sieci SOM przedstawia rysunek 4.



Rys. 4: Schemat struktury sieci SOM

Warto zauważyć, że neurony w warstwie konkurencyjnej nie są ze sobą bezpośrednio połączone. Zamiast tego przedstawia się je jako węzły siatki, co ma na celu ułatwienie zobrazowania ich topologii. Przykładową topologią

dla sieci SOM jest topologia prostokątna. Wybór topologii ma wpływ na ważny parametr w kontekście SOM – sąsiedztwo. Sąsiedztwo SOM można interpretować geometrycznie jako obszar o promieniu σ, w którym neurony są uznawane za należące do sąsiedztwa zwycięskiego neuronu. Przykład sąsiedztwa i jego promieni w topologii prostokątnej przedstawia rysunek 7.



Rys. 5: Sąsiedztwo SOM w topologii prostokątnej

Załóżmy, że mamy do czynienia z siecią SOM, która posiada n wejść i M neuronów konkurencyjnych numerowanych od 1 do M (jak pokazano na rysunku 4). Algorytm treningowy sieci SOM, stosujący strategię WTM (choć uproszczony w przypadku WTA), przedstawia się w następujących krokach [1][4]:

- Rozpocznij inicjalizację wektorów wag dla każdego neuronu. Często stosowaną metodą inicjalizacji jest losowanie wartości z rozkładu jednostajnego:
$$\mathcal{U}\left(-\frac{1}{\sqrt{n}};\frac{1}{\sqrt{n}}\right)$$
- Dla każdego wektora $\mathbf{X}$, losowanego bez zwracania ze zbioru uczącego:
  - ◇ Zidentyfikuj zwycięski neuron k, gdzie $1 \leqslant k \leqslant M$, dla którego wektor wag $\mathbf{W}_k$ jest najbardziej zbliżony do wektora wejściowego $\mathbf{X}$. Najczęściej używaną miarą podobieństwa jest odległość euklidesowa. Wówczas neuron zwycięski wyznaczamy poprzez:
$$k = \arg\min_j ||\mathbf{X} - \mathbf{W}_j|| = \arg\min_j \sqrt{\sum_{i=1}^{n}(x_i - w_{i,j})^2}, \qquad \text{dla } j = 1, 2, ..., M \tag{1}$$
  - ◇ Zaktualizuj wektory wag wszystkich neuronów W_j dla $j = 1, 2, ..., M$ zgodnie z następującym wzorem:
$$\mathbf{W}_j := \mathbf{W}_j + \eta(e) \cdot h(j,k) \cdot (\mathbf{X} - \mathbf{W}_j) \tag{2}$$
gdzie $\eta(e)$ to współczynnik uczenia w epoce e (ang. *learning rate*), a $h(j,k)$ to funkcja sąsiedztwa neuronu j względem neuronu zwycięskiego k. Przykład funkcji sąsiedztwa:
$$h(j,k) = \exp\left(\frac{-d(j,k)^2}{2\sigma(e)^2}\right) \tag{3}$$
gdzie $d(j,k)$ to odległość w siatce topologicznej między neuronem j i neuronem zwycięskim k, a $\sigma(e)$ to promień sąsiedztwa w epoce e.
Uwagi implementacyjne: Jeśli traktujemy neurony w topologii prostokątnej jako punkty w układzie kartezjańskim z całkowitymi współrzędnymi, odległość dowolnego neuronu j o współrzędnych (p,q) od neuronu zwycięskiego k o współrzędnych $(0,0)$ (po dokonaniu translacji) jest równa najkrótszej ścieżce

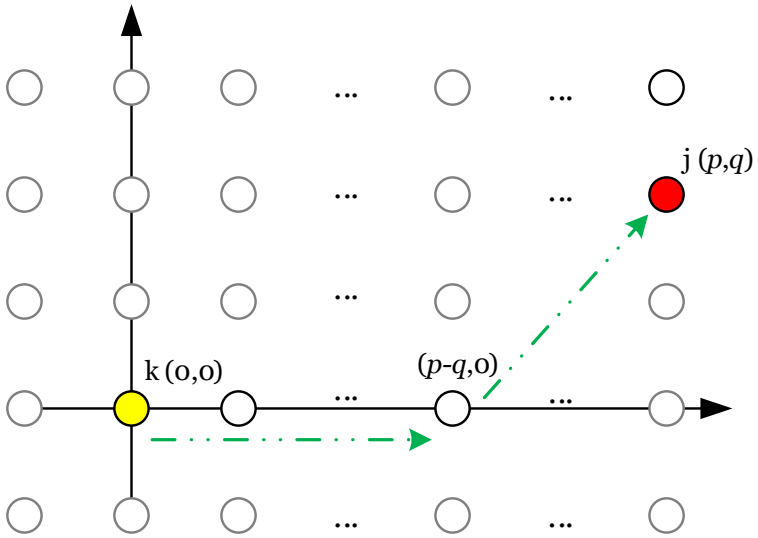
pomiędzy tymi punktami, uwzględniając ruch po neuronach, zarówno wzdłuż osi OX, OY, jak i po przekątnych. Taka odległość wynosi:

$$d(j,k) = \max(p,q) \tag{4}$$

□ *Dowód*: Z uwagi na symetrię względem osi OX, OY oraz punktu $(0,0)$, rozważmy przypadek pokazany na rysunku 6, gdzie neuron zwycięski k znajduje się w punkcie $(0,0)$, a neuron j w punkcie (p,q), przy czym $0 \leqslant p \leqslant q$. Aby znaleźć ścieżkę między tymi punktami, rozważmy następujący sposób przejścia:
— przejdźmy $(p-q)$ jednostek wzdłuż osi OX w kierunku dodatnim,
— przejdźmy q jednostek po przekątnej w kierunku dodatnim zarówno osi OX, jak i OY.
Całkowita długość tej ścieżki wyniesie:

$$q + (p-q) = p \tag{5}$$

Długość najkrótszej ścieżki między punktami (p,q) oraz $(0,0)$ nie może być mniejsza niż długość ścieżki między punktami $(p,0)$ i $(0,0)$, która wynosi właśnie p. Stąd przedstawiona strategia jest optymalna, a obliczona w (5) długość ścieżki jest rzeczywiście najkrótsza. Ponadto, zauważmy, że ta długość p jest równa $\max(p,q)$.
W ten sposób wykazaliśmy, że $d(j,k) = d\Big((p,q),(0,0)\Big) = \max(p,q)$, co kończy dowód. ■



Rys. 6: Przykładowa najkrótsza ścieżka między dwoma neuronami w siatce SOM o topologii prostokątnej

3.3. Learning Vector Quantization (LVQ)

Algorytm LVQ (Learning Vector Quantization), czyli Nauka Kwantyzacji Wektorów, jest ściśle powiązany z siecią SOM. Do tego pojęcia zalicza się różne algorytmy, takie jak LVQ1, LVQ2, LVQ3 oraz OLVQ1. W odróżnieniu od algorytmu Kwantyzacji Wektorów (VQ) oraz klasycznego SOM, LVQ jest algorytmem uczonym w sposób nadzorowany (ang. *supervised learning*), w którym nie definiuje się neuronów w sąsiedztwie zwycięzcy. Ponadto, proces trenowania nie wymaga porządkowania przestrzennego wektorów wejściowych.

LVQ został zaprojektowany przede wszystkim do celów klasyfikacji statystycznej oraz rozpoznawania metod. Jego głównym celem jest podział danych uczących na klasy, w których wektory wejściowe są do siebie podobne. Algorytm przypisuje każdej z tych klas różne elementy wektorów wejściowych, tworząc klasy z granicami liniowymi, w których wektory wejściowe są przypisane do jednej klasy na stałe.

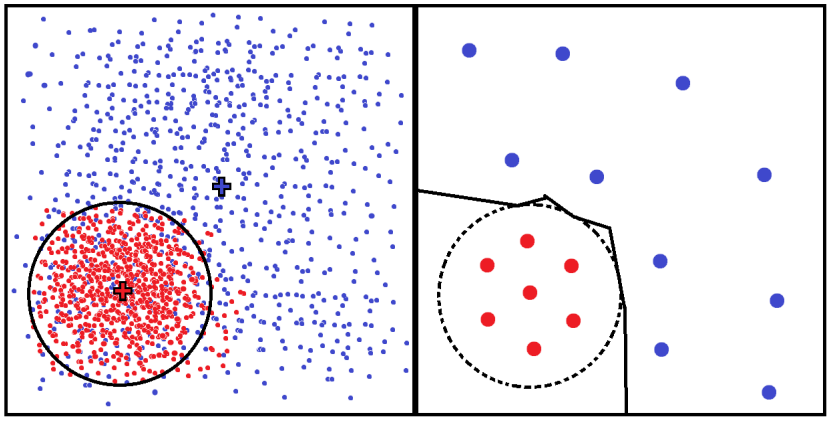
LVQ różni się także od klasycznego podejścia Bayesa w kontekście przestrzeni decyzyjnej. Załóżmy, że wszystkie próbki x pochodzą ze zbioru S klas S_k, gdzie rozkłady klas mogą się nakładać. Przyjmując, że $P(S_k)$ to prawdopodobieństwo *a priori* dla klasy S_k oraz $p(x|x \in S_k)$ to funkcja gęstości prawdopodobieństwa warunkowego, możemy zdefiniować funkcje dyskryminacyjne δ_k:

$$\delta_k(x) = p(x \mid x \in S_k) \cdot P(S_k)$$

Przy założeniu, że klasyfikacja próbki x jest optymalna, tzn. że próbka x jest przypisana do klasy S_c, kiedy:

$$\delta_c(x) = \max_k \{\delta_k(x)\}$$

W klasycznych metodach statystycznego rozpoznawania wzorców, tworzy się przybliżenia dla $p(x|x \in S_k)P(S_k)$, które następnie wykorzystuje się do obliczenia funkcji $\delta_k(x)$. W algorytmie LVQ podejście jest inne. Początkowo dla każdej klasy S_k przypisuje się zbiór wektorów porównawczych. Następnie, gdy pojawia się nowy, nieklasyfikowany wektor x, poszukuje się wektora porównawczego m_i, który ma najmniejszą odległość euklidesową od x. Zakłada się wtedy, że wektor x należy do tej samej klasy co m_i (zasada najbliższego sąsiada). Możliwe jest takie rozmieszczenie wektorów porównawczych, aby wektory z różnych klas były od siebie oddzielone, nie przeplatały się. W tym przypadku algorytm koncentruje się na rozmieszczeniu wektorów w pobliżu granicy decyzyjnej. Oznacza to, że nie jest konieczne przybliżanie wartości $p(x|x \in S_k)P(S_k)$ w całym obszarze, a raczej ważne jest, by rozmieszczenie wektorów porównawczych minimalizowało ryzyko błędnej klasyfikacji, zgodnie z zasadą najbliższego sąsiada.



Rys. 7: Porównanie sposobu klasyfikatora Bayesa (po lewej) oraz modelu LVQ (po prawej)

Na powyższym wykresie przedstawiono różnicę w sposobie wyznaczania granicy decyzyjnej. Na pierwszym wykresie czerwone kropki reprezentują funkcję gęstości Gaussa dla klasy S_1 z jej centrum w czerwonym krzyżu, natomiast niebieskie kropki oznaczają funkcję gęstości Gaussa klasy S_2 z centrum w niebieskim krzyżu. Czarny okrąg zaznacza granicę decyzyjną Bayesa. Na drugim wykresie duże czerwone kropki pokazują wektory porównawcze klasy S_1, a niebieskie kropki wektory porównawcze klasy S_2. Czarna linia łamana reprezentuje granicę decyzyjną w przypadku algorytmu LVQ [5].

LVQ1 to pierwszy i najbardziej podstawowy wariant algorytmu LVQ. Proces jego trenowania przebiega według następującego schematu:

1. Inicjalizacja wartości wag wektorów porównawczych.
2. Określenie liczby wektorów dla każdej klasy.
3. Ustalenie współczynnika uczenia $\alpha \in (0, 1)$ oraz maksymalnej liczby epok.
4. Dla każdego wektora x ze zbioru treningowego:
   - Obliczanie odległości między wektorem wejściowym x a wektorami porównawczymi w:

$$d(x, w) = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i - w_{ij})^2$$

   gdzie:
   - $\diamond$ n — liczba wektorów treningowych,
   - $\diamond$ m — liczba wektorów porównawczych.
5. Określenie zwycięskiego wektora.
6. Aktualizacja wektora:
   $w = w \ + \alpha(x - w)$, jeśli klasy wektorów są takie same,
   $w = w \ - \alpha(x - w)$, jeśli klasy wektorów są różne.
7. Zmniejszenie współczynnika uczenia: $\beta \in (0, 1)$, $\alpha := \beta \cdot \alpha$.

LVQ2 wprowadza modyfikacje procesu uczenia swojego poprzednika, aby zwiększyć efektywność algorytmu. Dodane zostaje okno odległości wymagane do zmiany wartości, a w przypadku spełnienia warunków, zmieniane są dwa najbliższe wektory porównawcze. Dla wersji 2.0 warunki zmiany wag wektorów porównawczych w_i oraz w_j są następujące:

1. Wektor porównawczy w_i ma najmniejszą odległość do wektora testowego, a w_j drugą najmniejszą. Klasa wektora testowego jest różna od klasy wektora w_i, ale taka sama jak klasy wektora w_j,
2. Wektor wejściowy znajduje się w określonej odległości (oknie) od wektora porównawczego. Wielkość okna zwykle ustawia się na 0.2 lub 0.3 ($okno \in (0, 1)$).

Dla wersji 2.1 wprowadzono modyfikację pierwszego warunku:

1a. Wektor porównawczy w_i ma najmniejszą odległość do wektora testowego, a wektor w_j drugą najmniejszą. Klasa wektora testowego jest różna od klasy wektora w_i oraz taka sama jak klasy wektora w_j,
1b. Wektor porównawczy w_i ma najmniejszą odległość do wektora testowego, a wektor w_j drugą najmniejszą. Klasa wektora testowego jest różna od klasy wektora w_j oraz taka sama jak klasy wektora w_i,
2. Wektor wejściowy znajduje się w określonej odległości (oknie) od wektora porównawczego. Wielkość okna zwykle ustawia się na 0.2 lub 0.3 ($okno \in (0, 1)$). Czy wektor testowy spełnia warunki okna, określa się za pomocą równania:

$$\min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > s, \qquad \text{gdzie } s = \frac{1 - okno}{1 + okno}$$

gdzie:
- d_i to odległość między w_i a wektorem testowym,
- d_j to odległość między w_j a wektorem testowym.

Aby spełniony był pierwszy warunek, wystarczy, że tylko jeden z jego podpunktów będzie prawdziwy. Gdy oba warunki zostaną spełnione, wektor porównawczy należący do tej samej klasy co wektor testowy przybliża się do wektora testowego, natomiast wektor porównawczy należący do innej klasy oddala się od wektora testowego:

$$w_i := w_i + \alpha(x - w_i)$$
$$w_j := w_j - \alpha(x - w_j)$$

Algorytm LVQ2 powstał z myślą o różnicowym zbliżaniu granicy decyzyjnej do granic podejścia Bayesa, przy jednoczesnym zignorowaniu tego, co dzieje się z lokalizacją wektora porównawczego w_i w trakcie całego procesu. Z tego powodu konieczne stało się zapewnienie, by w_i cały czas dążył do przybliżenia rozkładu klas. Tak powstał algorytm LVQ3, który zawiera wszystkie operacje z algorytmu LVQ2.1 oraz wprowadza dodatkowe operacje pod warunkiem, że zarówno wektory porównawcze w_i, w_j, jak i wektor testowy x, należą do tej samej klasy. Kiedy oba najbliższe wektory porównawcze oraz wektor testowy należą do tej samej klasy, a dla danego wektora testowego spełniony jest warunek okna, oba wektory porównawcze przybliżają się do wektora testowego:

$$w_i := w_i + \epsilon \cdot \alpha(x - w_i)$$
$$w_j := w_j + \epsilon \cdot \alpha(x - w_j)$$

Zmienna ϵ jest zmienną *stabilizującą*, której standardowa wartość mieści się w przedziale od 0.1 do 0.5, w zależności od wartości okna (zwykle równej 0.2 lub 0.3). Optymalna wartość zmiennej ϵ wydaje się zależna od rozmiaru okna: im mniejsza wartość okna, tym mniejsza optymalna wartość zmiennej ϵ.

Algorytm LVQ2 był pierwszym, który wprowadził metodę aktualizacji wag dwóch wektorów w trakcie jednej epoki, choć zmiany te zachodziły dość rzadko. Algorytmy LVQ2.1 i LVQ3 wprowadziły dodatkowe warunki, które zwiększyły częstotliwość tych zmian, poprawiając dopasowanie algorytmu. Jednak to dopiero LVQX spowodował, że wagi wektorów porównawczych są aktualizowane w każdej epoce. Taki zabieg przyspiesza proces uczenia, zwiększa jego efektywność i skraca czas potrzebny na naukę. W algorytmie LVQX, w każdej epoce wyznaczane są dwa wektory porównawcze, ale w odmienny sposób niż w standardowej metodzie, gdzie wybiera się dwa wektory najbliższe wektorowi testowemu. W LVQX wybierane są zwycięski wektor globalny oraz zwycięski wektor lokalny. Zwycięski wektor globalny w_g jest tym, który ma najmniejszą odległość do wektora testowego, niezależnie od jego klasy, natomiast zwycięski wektor lokalny w_l to wektor o najmniejszej odległości do wektora testowego, wybrany tylko spośród wektorów tej samej klasy. Jeśli oba wektory, lokalny i globalny, są identyczne, wówczas aktualizowana jest tylko waga tego jednego wektora. W przeciwnym przypadku, wektor globalny oddala się od wektora testowego, a wektor lokalny zbliża się do niego. Operacje na wektorach prezentują się następująco:

- Jeżeli $w_l = w_g$, to:

$$w_g := w_g + \alpha(x - w_g)$$

- W przeciwnym przypadku:

$$w_g := w_g - \alpha(x - w_g)$$
$$w_l := w_l + \alpha(x - w_l)$$

Z przeprowadzonych badań [6] wynika, że wprowadzone usprawnienia w LVQX uczyniły go najbardziej efektywnym algorytmem w rodzinie LVQ, zwłaszcza w przypadkach, gdy wagi wektorów są początkowo przydzielone i różne od zera. W naszym przypadku warunek ten zostanie spełniony dzięki etapowi SOM, co stanowi uzasadnienie dla zastosowania tego algorytmu w tworzeniu systemu IDS.

W niektórych przypadkach początkowy rozkład wag dla wektorów porównawczych może stanowić wyzwanie. W takich sytuacjach skuteczną metodą może być zastosowanie klasyfikacji bez nadzoru przy użyciu algorytmu SOM, który początkowo ustali wagi wektorów. Dopiero wtedy ta początkowa mapa wektorów jest przekazywana do algorytmu LVQ, który uwzględnia klasy danych treningowych. Taki proces umożliwia dalsze dostosowanie wartości uzyskanych dzięki SOM za pomocą LVQ, co prowadzi do uzyskania jak najwyższej celności, czyli poprawnej klasyfikacji ataków [7][8][9]

4. Sposoby oceny jakości

4.1. Miary jakości

Podstawową miarą jakości, którą zastosujemy podczas oceny uzyskanych modeli, jest tablica pomyłek (ang. confusion matrix). Tablica pomyłek dla klasyfikacji binarnej pozwala na określenie czterech podstawowych wielkości, które charakteryzują model:

- TP (True Positives): liczba przykładów prawdziwie pozytywnych (w systemie IDS: atak prawidłowo sklasyfikowany jako atak),
- FN (False Negatives): liczba przykładów fałszywie negatywnych (w IDS: atak błędnie sklasyfikowany jako ruch normalny),
- FP (False Positives): liczba przykładów fałszywie pozytywnych (w IDS: ruch normalny błędnie sklasyfikowany jako atak),
- TN (True Negatives): liczba przykładów prawdziwie negatywnych (w IDS: ruch normalny prawidłowo sklasyfikowany jako ruch normalny).

Tablica pomyłek dla klasyfikacji binarnej przyjmuje następującą postać:

		Klasa rzeczywista	
		Pozytywna	Negatywna
Predykcja	Pozytywna	**TP**	**FP**
	Negatywna	**FN**	**TN**

W oparciu o powyższą tabelę pomyłek, oprócz wcześniej podanych wskaźników, możemy wyliczyć również następujące miary:

Czułość (TPR)	$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$
Swoistość (TNR)	$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{FP+TN}}$
Precyzja (PPV)	$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$
Dokładność (ACC)	$\mathrm{ACC} = \frac{\mathrm{TP+TN}}{\mathrm{TP+FP+TN+FN}}$
Miara F1	$\mathrm{F1} = \frac{2\mathrm{TP}}{2\mathrm{TP+FP+FN}}$

W kontekście systemów IDS, kluczową miarą jest FPR (ang. False Positive Ratio), która określa, jaki procent normalnego ruchu zostaje błędnie zakwalifikowany jako atak. Redukcja FPR jest niezbędna, aby zminimalizować liczbę fałszywych alarmów generowanych przez system IDS. Wartość FPR można bezpośrednio obliczyć na podstawie swoistości TNR:

$$\mathrm{FPR} = 1 - \mathrm{TNR}$$

4.2. Sposób oceny jakości

Dla każdego typu modelu (czyli zestawu określonych parametrów) przeprowadzimy wielokrotne próby w celu oceny jego efektywności. W trakcie każdej iteracji analizowane będą odpowiednie metryki, które następnie zostaną uśrednione. Aby rzetelnie ocenić i porównać wpływ poszczególnych parametrów na jakość modelu, zawsze zmieniany będzie tylko jeden parametr. Szczególną uwagę poświęcimy na następujące metryki: ACC, TPR oraz FPR.

5. Implementacja

5.1. Wykorzystane narzędzia

Cały projekt systemu IDS został zaimplementowany w języku Python w wersji 3.12. Do stworzenia systemu wykorzystaliśmy następujące biblioteki:

- `numpy` – umożliwia efektywne operacje na dużych zbiorach danych oraz oferuje funkcje matematyczne, które przyspieszają działanie modułów systemu IDS,
- `numba` – przyspiesza wykonanie kodu, tłumacząc funkcje Pythona na zoptymalizowany kod maszynowy w czasie rzeczywistym,
- `pandas` – służy do obsługi i formatowania dużych zbiorów danych, takich jak dane treningowe i testowe,
- `matplotlib` – umożliwia wizualizację danych na wykresach oraz zdjęciach,
- `scikit-learn` – wykorzystywana do przygotowania danych wejściowych, w tym metod podziału zbiorów.

Dodatkowo, do wersjonowania kodu użyliśmy narzędzia `Git`, co ułatwiło zarządzanie projektem i monitorowanie postępów w jego realizacji.

5.2. Wybrany zbiór danych

Dla modeli, które mają na celu analizę ruchu sieciowego i wspieranie działania systemów IDS, najczęściej wykorzystywanym zbiorem danych jest KDDCUP'99. Zawiera on dane z ruchu sieciowego przechwycone w ramach programu DARPA'98. Składa się z 4 gigabajtów skompresowanych danych binarnych opisujących 7 tygodni ruchu sieciowego, co daje około 5 milionów rekordów połączeń. Zebrany ruch zawiera 41 atrybutów, a dane klasyfikowane są jako ruch normalny lub jako jeden z możliwych ataków z poniższych kategorii:

- **Denial of Service Attack (DoS)** - atakujący nadużywa (przepełnia) zasoby obliczeniowe ofiary, aby uniemożliwić obsługę faktycznego ruchu użytkowników.
- **User to Root Attack (U2R)** - atakujący, będąc zwykłym użytkownikiem, stara się zdobyć uprawnienia administratora systemu.
- **Remote to Local Attack (R2L)** - atakujący, mając jedynie możliwość komunikacji z maszyną ofiary, zdobywa lokalny dostęp do systemu, wykorzystując znalezione luki bezpieczeństwa.
- **Probing Attack** - atakujący próbuje uzyskać informacje o systemie w celu kompromitacji jego zabezpieczeń.

Warto zaznaczyć, że dane testowe pochodzą z innego rozkładu niż dane treningowe, łącznie z typami ataków, które nie występują w zbiorze treningowym. Dzięki temu proces staje się bardziej realistyczny – eksperci ds. cyberbezpieczeństwa twierdzą, że nowe typy ataków są wariantami podstawowych. W związku z tym, sygnatury wypracowane na podstawie danych treningowych powinny wystarczyć do wykrywania nowych wersji ataków. Zbiór danych zawiera w sumie 24 typy ataków używanych do treningu oraz 14 dodatkowych, które służą do testowania.
Opisywany zbiór danych posiada jednak pewne wady, w tym:

- Twórcy dodali sztuczny ruch, symulujący normalny, mający na celu zamaskowanie źródła ataków. Taki ruch nie odzwierciedla rzeczywistego stanu sieci.
- Możliwość wystąpienia zjawiska odrzucania pakietów przez narzędzia podczas przechwytywania ruchu z powodu przeciążenia.
- Niedokładnie zdefiniowane ataki, na przykład próbki (probing) nie są klasyfikowane jako atak, dopóki liczba wystąpień tego zdarzenia nie przekroczy określonego progu.
- Zbiór danych spotkał się z krytyką dotyczącą jego struktury oraz miar wydajności.
- Rozkład ataków w zbiorze danych jest nierównomierny, co utrudniało przeprowadzenie walidacji krzyżowej i wpływało na ostateczną ocenę modelu.
- Wysoka liczba powtarzających się rekordów w zbiorach treningowych i testowych. Powodowało to, że algorytmy były stronnicze względem częściej występujących etykiet, co utrudniało naukę rozpoznawania rzadkich ataków, takich jak U2R czy R2L. Powtarzające się rekordy prowadziły także do stronniczości w metodach wykrywania, które miały lepsze wskaźniki skuteczności oparte na bardziej powszechnych danych. Rozwiązaniem było usunięcie powtarzających się rekordów, co w przypadku zbioru treningowego skutkowało redukcją około 93% krotek.

- Zbiór danych nie był zoptymalizowany pod kątem typowych algorytmów maszynowych, co skutkowało zbyt wysokimi wynikami uzyskanymi przez testowane algorytmy. Ponadto oceny oparte na dokładności, współczynniku wykrywalności i współczynniku fałszywych alarmów nie dawały miarodajnych metryk.

Z powodu powyższych problemów, zbiór danych nie nadawał się do testowania anomalii sieciowych ani uczenia klasyfikatorów, dlatego twórcy postanowili go poprawić. Jednym z omawianych problemów była redundancja rekordów, która została zredukowana w obu zbiorach. Dodatkowo, w celu zbilansowania wyników, utworzono bardziej wymagający zbiór, pobierając próbki z wyników wcześniejszych eksperymentów w taki sposób, że liczba rekordów z każdej grupy była odwrotnie proporcjonalna do procentowego udziału tych rekordów w poprzednich wynikach.

Efektem przeprowadzonej analizy oraz ulepszeń na jej podstawie był zbiór `NSL-KDD` [10][11], który ostatecznie został wykorzystany do procesu trenowania i testowania. Główne cechy oraz poprawki w stosunku do poprzedniego zbioru to:

- Usunięcie redundantnych rekordów, co eliminuje stronniczość wyników wobec częściej występujących wpisów.
- Zbiór testowy nie zawiera duplikujących się krotek, dzięki czemu wyniki nie są zniekształcone przez metody lepiej radzące sobie z częściej występującymi rekordami.
- Dzięki zastosowanej metodzie próbkowania, typy klasyfikatorów różnią się w szerszym zakresie, co pozytywnie wpływa na ich dokładność.
- Redukcja rozmiaru zbiorów pozwoliła na ich stosowanie bez konieczności dzielenia na mniejsze części, co poprawiło spójność i porównywalność uzyskanych wyników.

Typy przykładów wraz z ich liczebnością zostały zaprezentowane w poniższej tabeli.

Typ przykładu	**Liczebność**
normal	9711
neptune	4657
guess_passwd	1231
mscan	996
warezmaster	944
apache2	737
satan	735
processtable	685
smurf	665
back	359
snmpguess	331
saint	319
mailbomb	293
snmpgetattack	178
portsweep	157
ipsweep	141
httptunnel	133
nmap	73
pod	41

Typ przykładu	**Liczebność**
buffer_overflow	20
multihop	18
named	17
ps	15
sendmail	14
xterm	13
rootkit	13
teardrop	12
xlock	9
land	7
xsnoop	4
ftp_write	3
worm	2
phf	2
udpstorm	2
sqlattack	2
perl	2
loadmodule	2
imap	1

5.3. Struktura rozwiązania

Podczas tworzenia naszego rozwiązania zadbaliśmy o rozdzielenie implementacji poszczególnych części algorytmu oraz funkcji i modeli pomocniczych. Ostateczna struktura rozwiązania wygląda następująco:

Folder główny zawiera implementacje algorytmów oraz funkcji pomocniczych. Znajdują się w nim:

- plik `ais.py` – implementacja AIS, zawierająca opcje uruchomienia (proces uczenia i ewaluacji) lub zaimportowania (klasa),
- plik `som_w_lvq.py` – implementacja sieci SOM z wzmocnieniem algorytmem LVQX, z możliwością wykonania (proces uczenia i ewaluacji, interesujące przypadki testowe) lub zaimportowania (klasa),
- plik `ids.py` – implementacja systemu IDS, który wykorzystuje AIS oraz SOM wzmacniane LVQX, plik wykonywalny symulujący działanie oraz ewaluację IDS (oraz jego komponentów: AIS oraz LVQX),

- plik `utilities.py` – zawiera funkcje i stałe pomocnicze.

Folder `dataset/nll-kdd` zawiera pliki z wykorzystywanymi zbiorami danych:

- plik `KDD.names` – zawiera opis zbiorów danych KDD,
- plik `KDDTrain+.txt` – zbiór danych zawierający przykłady połączeń sieciowych sklasyfikowanych jako połączenia normalne lub ataki. Z tego zbioru (ze względu na jego rozmiar) wyodrębniliśmy zbiory treningowe i testowe (stosunek 7:3) do uczenia poszczególnych elementów systemu IDS. Wykorzystany również do ewaluacji całego systemu IDS,
- plik `KDDTest+.txt` – zbiór danych podobny do `KDDTrain+.txt`, zawierający połączenia, których żaden element systemu IDS nie widział podczas nauki. Wykorzystany do ewaluacji całego systemu IDS,
- plik `KDDTest-21.txt` – zbiór danych podobny do `KDDTrain+.txt`, specjalnie przygotowany, zawierający przykłady trudne do klasyfikacji z powodu podobieństwa połączeń normalnych i ataków. Wykorzystany do ewaluacji całego systemu IDS.

Folder `output_images` zawiera przykładowe wizualizacje sieci SOM, które zostały wygenerowane podczas testowania.
Folder `pickles` zawiera zserializowane obiekty AIS oraz SOM w formacie `.pickle`, które są ładowane i wykorzystywane przez system IDS.

5.4. AIS

AIS (patrz: 3.1) zostało zaimplementowane w pliku `ais.py` jako klasa `ArtificialImmuneSystem`. Implementacja, oprócz określenia zbioru treningowego składającego się z przykładów pozytywnych (normalny ruch sieciowy, bez ataków), umożliwia określenie liczby generowanych detektorów oraz odległości (tzw. threshold), jaką muszą zachować te detektory od przykładów w zbiorze treningowym. W celu pokrycia jak najszerszej przestrzeni przykładów, wprowadzono dodatkowe usprawnienie, które znacząco wydłuża czas generowania detektorów — detektory muszą zachować odległość (równej połowie threshold) od innych już wygenerowanych detektorów. Odległość między detektorem a przykładem obliczamy jako odległość euklidesową między dwoma wektorami.

Implementacja AIS umożliwia także ewaluację utworzonego modelu oraz jego serializację do pliku w formacie `.pickle`.

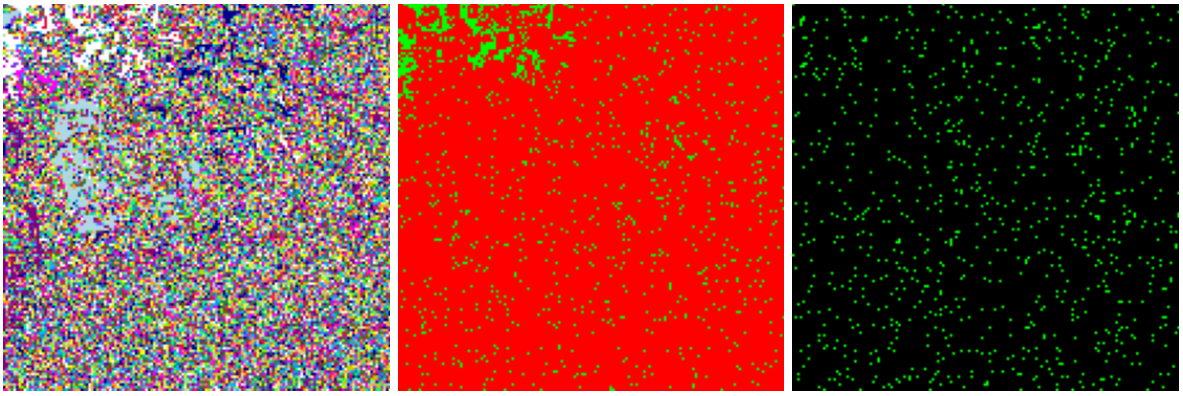
5.5. SOM

Implementacja SOM (patrz: 3.2) znajduje się w pliku `som_w_lvq.py` w formie klasy `SelfOrganizingMap`. Oprócz określenia zbioru treningowego i jego etykiet, umożliwia ona także ustalenie liczby epok, początkowej wartości promienia sąsiedztwa oraz współczynnika uczenia się (oba parametry zmieniają się liniowo w trakcie kolejnych epok), a także konfigurację rozmiaru sieci SOM.

W tym samym pliku zaimplementowano również metody, które umożliwiają filtrowanie zbiorów treningowych i testowych przez usunięcie części normalnego ruchu, co jest symulacją działania systemu AIS.

Dodatkowo klasa `SelfOrganizingMap` oferuje możliwość generowania obrazów o wymiarach `px`, które odpowiadają rozmiarowi sieci SOM. Obrazy te wizualizują klasy neuronów w trzech różnych trybach:

- rozróżniające wszystkie klasy neuronów (klasyfikacja ataków) – wygenerowany obraz posiada sufiks `labels.png`. Przykład obrazu dla jednej z klas można zobaczyć na rysunku 8(a).
- rozróżniające klasę pozytywną (ruch normalny) i negatywną (atak) – plik z obrazem ma sufiks `attack.png`. Przykład tego obrazu dla jednej z klas znajduje się na rysunku 8(b).
- rozróżniające konkretną klasę neuronu od innych – obraz jest zapisany z prefiksem odpowiadającym klasie neuronu (np. `portsweep`) oraz sufiksem `layer.png`. Przykład takiego obrazu dla klasy `nmap` znajduje się na rysunku 8(c).



((a)) Rozróżnienie wszystkich klas neuronów

((b)) Rozróżnienie klas pozytywnych (zielonych) i negatywnych

((c)) Rozróżnienie neuronów klasy `nmap` (zielonych) od pozostałych

Rys. 8: Tryby generowania obrazków wizualizujących klasy neuronów (przykład dla SOM 150x150)

Implementacja SOM oferuje również możliwość oceny stworzonego modelu oraz zapisania go do pliku w formacie `.pickle`. Dodatkowo, możliwe jest wzmocnienie wygenerowanej sieci przy użyciu algorytmu LVQ, co zostało szczegółowo omówione w sekcji 5.6.

5.6. LVQ

W ramach implementacji SOM (zobacz: sekcja 5.5), istnieje możliwość wzmocnienia neuronów sieci konkurencyjnej za pomocą algorytmu LVQX (patrz: sekcja 3.3). Algorytm ten został zaimplementowany w pliku `som_w_lvq.py` i jest uruchamiany poprzez metodę `lvq_enforcement` klasy `SelfOrganizingMap`. Pozwala ona na określenie liczby epok, współczynnika uczenia się (który zmniejsza się liniowo w każdej epoce), a także listy przykładów treningowych wraz z ich etykietami.

Zastosowanie algorytmu LVQ do wzmocnienia sieci SOM wpływa tylko na modyfikację wektorów wag neuronów i nie zmienia funkcjonalności klasy `SelfOrganizingMap`.

5.7. IDS

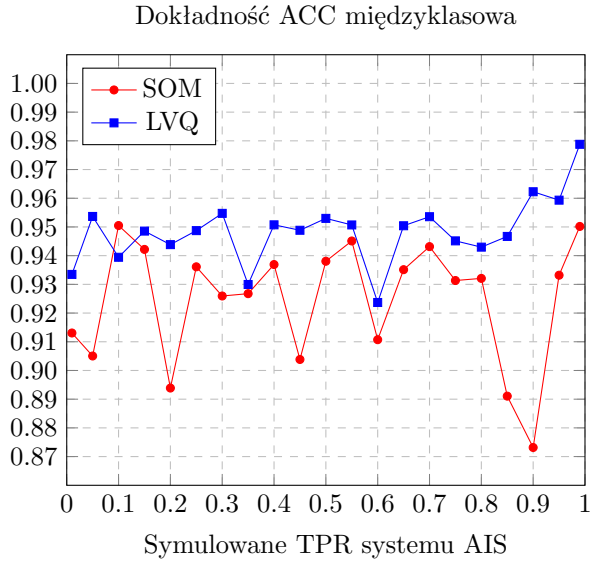
IDS (zobacz: sekcja 2) zostało zaimplementowane w pliku `ids.py` jako program, który ładuje wcześniej wytrenowane modele AIS oraz SOM wzmocnione algorytmem LVQ, zapisane w formacie `.pickle` jako serializowane klasy. Program umożliwia nie tylko wczytanie tych modeli, ale również załadowanie zbioru danych, na którym ma być przeprowadzona detekcja. Zgodnie z założeniami, program przypisuje każdemu z analizowanych przykładów jedną z dwóch klas: `normal` lub `malicious`. Po przeprowadzeniu klasyfikacji na dostarczonym zbiorze, program oblicza metryki takie jak: TPR, FPR i ACC dla implementacji IDS, a także dla poszczególnych modeli działających niezależnie od siebie.

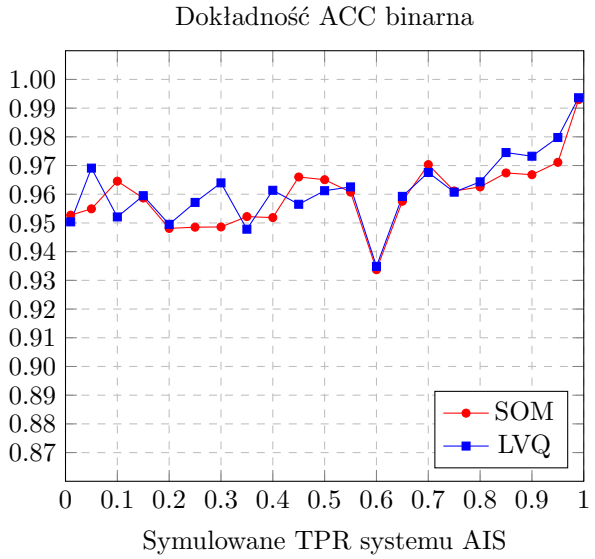
6. Eksperymenty

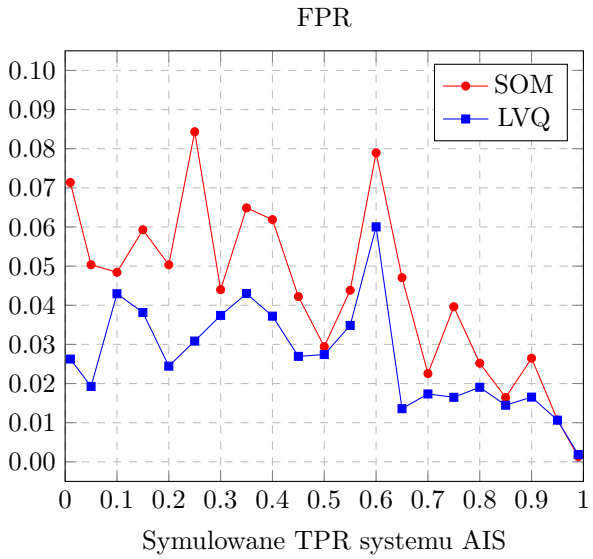
W ramach przeprowadzonych eksperymentów postanowiliśmy zbadać wpływ zakładanej czułości TPR systemu AIS (odfiltrowywania ruchu normalnego) oraz liczby neuronów (rozmiaru sieci konkurencyjnej) na jakość sieci SOM. Obserwowaliśmy również proces klasteryzacji przykładów przez sieć SOM. Na końcu przeanalizowaliśmy jakość stworzonego systemu IDS. Podczas zbierania wyników uwzględniliśmy ocenę SOM zarówno przed, jak i po wzmocnieniu algorytmem LVQ, aby zaobserwować jego wpływ na metryki modelu.

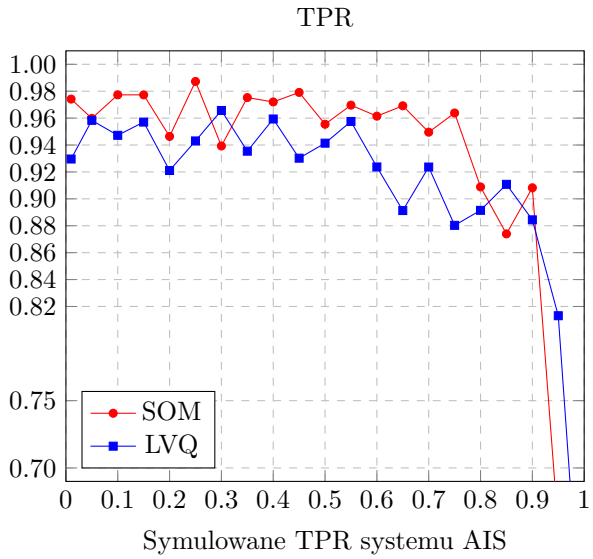
6.1. Wpływ czułości AIS na jakość SOM

Aby zbadać wpływ czułości systemu AIS odfiltrowującego ruch normalny przed klasyfikacją przykładów przez SOM, należy zauważyć, że czułość jest kluczowym parametrem AIS, który ma największy wpływ na późniejsze działanie SOM. Wraz ze wzrostem czułości AIS, binarna dokładność SOM-a poprawiała się, ponieważ do sieci trafiało więcej przykładów poprawnie zaklasyfikowanych jako negatywne. Jednocześnie, wzrost TPR w systemie AIS prowadził do obniżenia FPR SOM-a, co wynikało z mniejszej liczby przykładów normalnego ruchu trafiających do SOM. To z kolei zmniejszało prawdopodobieństwo, że ruch normalny zostanie błędnie sklasyfikowany jako atak. Z drugiej strony, wyższy TPR AIS powodował spadek TPR SOM-a, co było efektem mniejszej liczby przykładów normalnego ruchu przekazanych do SOM. W rezultacie, mniejsza liczba przykładów ruchu normalnego trafiających do SOM sprawiała, że pojedyncze przykłady zaklasyfikowane jako FN miały większy wpływ na obliczony TPR.



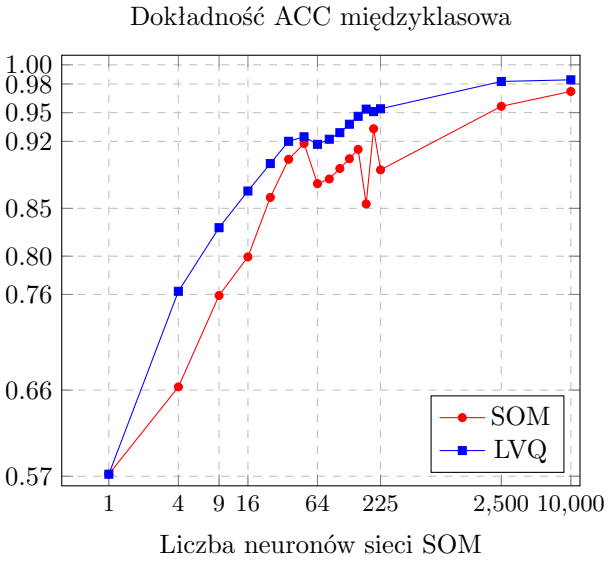


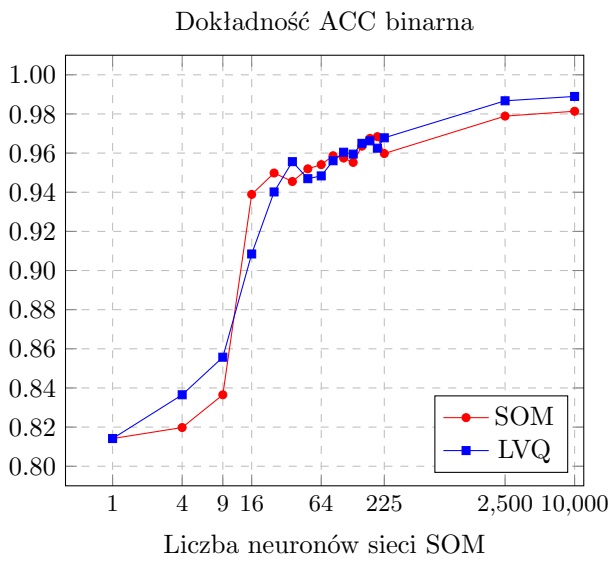


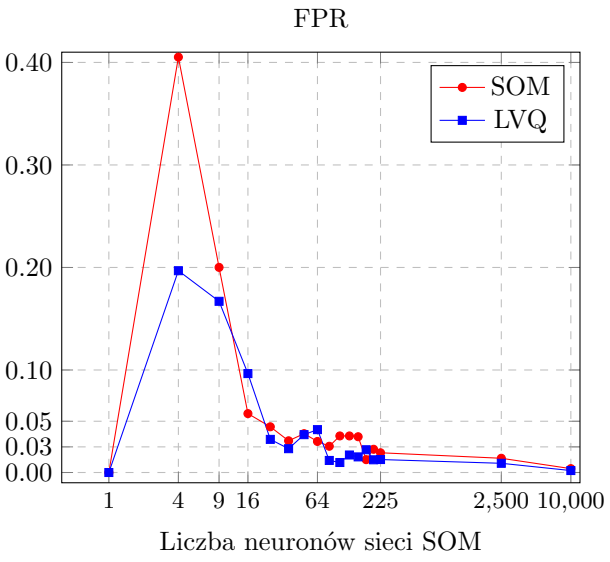


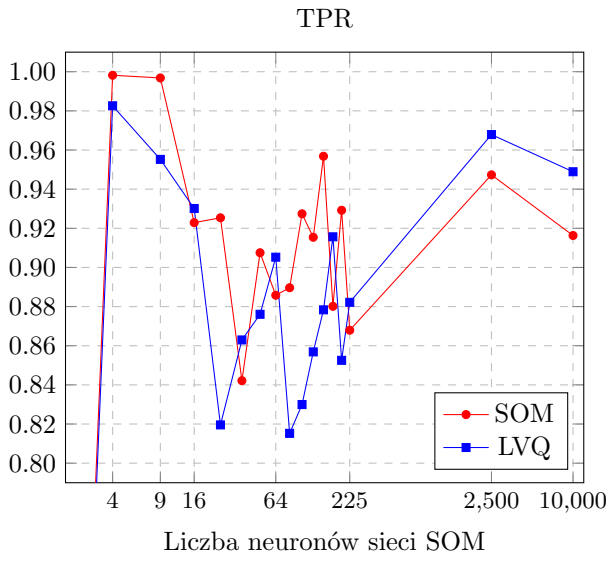
6.2. Wpływ ilości neuronów na jakość SOM

Zbadaliśmy również wpływ liczby neuronów w modelu na jakość klasyfikacji SOM. Wraz ze wzrostem liczby neuronów, poprawiała się zarówno klasyfikacja binarna, jak i wieloklasowa. Dla małej liczby neuronów zaobserwowaliśmy niedeterministyczny charakter współczynników TPR i FPR, które były niestabilne. Jednakże, w miarę zwiększania liczby neuronów, współczynniki te zaczęły się stabilizować: TPR stopniowo wzrastał, a FPR malał, dążąc do wartości zerowej.



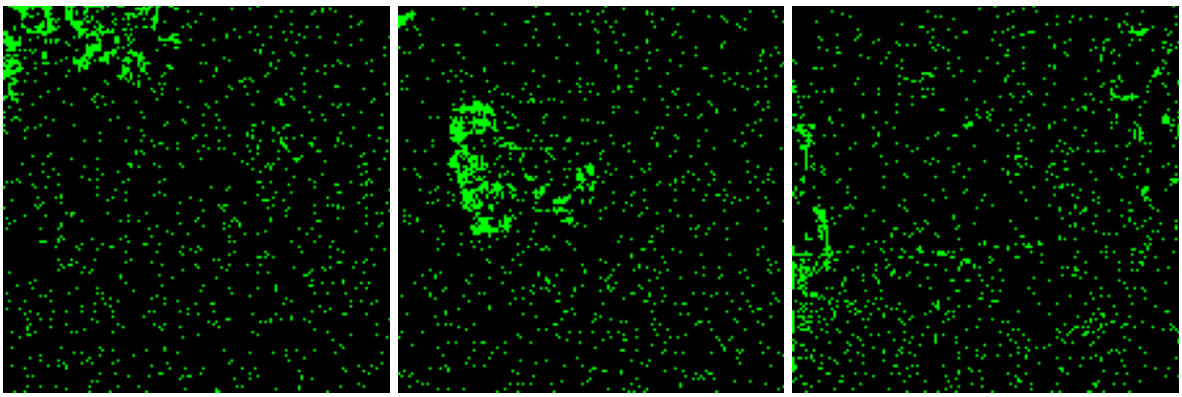






6.3. Klasteryzacja przykładów przez SOM

Jednym z kluczowych zadań sieci SOM jest klasteryzacja klas przykładów. W naszym eksperymencie udało się zaobserwować to zachowanie w zaprojektowanym modelu. Przykład klasteryzacji neuronów w większe skupiska dla klas `normal`, `neptune` oraz `satan` przedstawiono na rysunku 13.



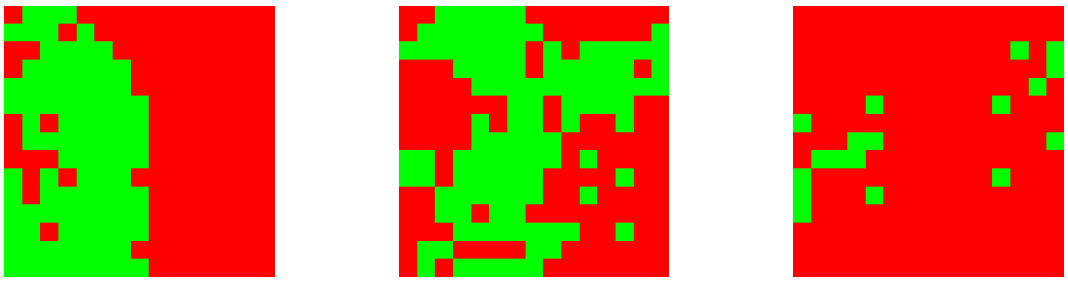
((a)) Neurony `normal` ((b)) Neurony `neptune` ((c)) Neurony `satan`

Rys. 13: Przykład klasteryzacji neuronów danych klas (ozn. na zielono, przykład dla SOM 150x150)

Zwiększając TPR symulowanego systemu AIS w trakcie uczenia się sieci SOM zwróciliśmy również uwagę na wzrost liczby różnych klas odwzorowywanych w sieci SOM (rys. 14) i jednoczesny spadek liczebności neuronów klasyfikującej ruch normalny (rys. 15).

((a)) Dla TPR AIS = 0.01 ((b)) Dla TPR AIS = 0.50 ((c)) Dla TPR AIS = 0.99

Rys. 14: Liczebność różnych klas (kolorów) dla różnych TPR systemu AIS (przykład dla SOM 15x15)



((a)) Dla TPR AIS = 0.01 ((b)) Dla TPR AIS = 0.50 ((c)) Dla TPR AIS = 0.99

Rys. 15: Liczebność neuronów pozytywnych (zielony) oraz negatywnych (czerwony) dla różnych TPR systemu AIS (przykład dla SOM 15x15)

6.4. Badanie jakości systemu IDS

Jakość zaprojektowanego systemu IDS oceniliśmy na dwóch zestawach danych: `KDD-Train+` (obejmujących przykłady, które były znane AIS i SOM podczas procesu uczenia) oraz `KDD-Test+` (zawierających przykłady, które były nieznane tym algorytmom w trakcie nauki). Do stworzenia systemu IDS wykorzystano AIS składający się z 500 detektorów (trenując je przy progu równym 4.1) oraz SOM o rozmiarach 150x150, wspomagane przez LVQX.

Oprócz oceny jakości całego systemu IDS, analizowaliśmy również efektywność klasyfikacji poszczególnych jego komponentów. Wartości metryk dla AIS i SOM zbierano niezależnie od drugiego elementu, przy założeniu, że przykłady są bezpośrednio przekazywane do odpowiedniego komponentu. Wyniki eksperymentów dla różnych zestawów testowych zaprezentowano w tabelach 2, 3 oraz 4.

6.4.1. Zbiór `KDD-Train+`

Dla zbioru `KDD-Train+` system osiąga wysoką dokładność we wszystkich komponentach. Jedną z głównych zalet AIS jest bliska idealna wartość TPR, wynosząca 1.0, co przekłada się na znakomite wyniki TPR całego systemu IDS.

Najwyższą, ale wciąż stosunkowo wysoką wartością FPR charakteryzuje się AIS, co sugeruje, że detektory nie zostały dostatecznie dobrze dopasowane do normalnego ruchu. SOM z kolei wykazuje niski poziom FPR, wynoszący około 2%, co jest wynikiem dużej liczby neuronów w warstwie konkurencyjnej.

Metryka	AIS	SOM	IDS
ACC	0.897	0.964	0.987
FPR	0.220	0.017	0.221
TPR	0.999	0.947	0.999

Tabela 2: Wyniki dla zbioru `KDD-Train+`

6.4.2. Zbiór `KDD-Test+`

Dla zbioru `KDD-Test+` system osiąga niższą dokładność w przypadku każdego z jego komponentów. Największy wpływ na jakość całego systemu IDS ma AIS. Spora część przykładów atakujących jest przez AIS klasyfikowana jako ruch normalny, przez co nie trafiają one do SOM, co widać po wysokich wartościach FPR dla AIS i IDS. FPR dla SOM jest niższe, ale nadal pozostaje stosunkowo wysokie. Wartości TPR pozostały na wysokim poziomie.

Metryka	AIS	SOM	IDS
ACC	0.691	0.751	0.672
FPR	0.500	0.359	0.539
TPR	0.943	0.896	0.950

Tabela 3: Wyniki dla zbioru `KDD-Test+`

6.4.3. Zbiór `KDD-Test-21`

Dla zbioru `KDD-Test-21` system osiąga znacznie niższą dokładność w przypadku każdego z jego elementów. Wciąż dużą zaletą AIS jest podniesienie wartości TPR całego systemu do relatywnie wysokiego poziomu, wyższego niż TPR uzyskany samodzielnie przez SOM. To przyczynia się do zadowalającej efektywności w wykrywaniu ataków. Niestety, dla tego zestawu danych współczynnik FPR całego systemu znacznie wzrósł, co skutkuje większą liczbą niepotrzebnych alertów, w których normalny ruch jest błędnie klasyfikowany jako potencjalny atak.

Metryka	AIS	SOM	IDS
ACC	0.465	0.593	0.465
FPR	0.590	0.430	0.623
TPR	0.850	0.697	0.861

Tabela 4: Wyniki dla zbioru `KDD-Test-21`

7. Podsumowanie

Połączenie AIS i SOM okazało się skutecznym podejściem do stworzenia precyzyjnego oraz adaptacyjnego systemu IDS.

Zgodnie z oczekiwaniami, najlepsze wyniki dokładności osiągnięto przy największym rozmiarze sieci SOM oraz dużej liczbie detektorów AIS.

Wysoka dokładność uzyskana na zbiorze `KDD-Train+` potwierdza prawidłowe działanie systemu dla przykładów podobnych do tych, na których system był trenowany. Jednak w przypadku nowych, wcześniej nieznanych przykładów o odmiennej charakterystyce (zbiory `KDD-Test+` i `KDD-Test-21`), jakość klasyfikacji systemu znacznie spadła. Wysokie wartości FPR uniemożliwiają zastosowanie tego modelu jako jedynej metody oceny ruchu sieciowego w rzeczywistych systemach IDS.

Kolejnym krokiem w pracy nad systemem powinno być wytrenowanie większych modeli, szczególnie zwiększenie liczby detektorów AIS, które pełnią rolę pierwszej linii obrony w systemie IDS.

Zrealizowanie tego tematu wymagało zgłębienia dostępnej teorii, co stanowiło fundament dla opracowania wstępu teoretycznego. Dzięki temu proces implementacji przebiegł zgodnie z założeniami.

Literatura

[1] Simon T Powers and Jun He. A hybrid artificial immune system and Self Organising Map for network intrusion detection. *Information Sciences*, 178(15):3024–3042, 2008.

[2] Tarek S Sobh and Wael M Mostafa. A cooperative immunological approach for detecting network anomaly. *applied soft computing*, 11(1):1275–1283, 2011.

[3] Ismaila Idris, Ali Selamat, and Sigeru Omatu. Hybrid email spam detection model with negative selection algorithm and differential evolution. *Engineering Applications of Artificial Intelligence*, 28:97–110, 2014.

[4] Karolina Bartos. Sieć SOM jako przykład sieci samoorganizującej się. *Ekonometria*, (36):65–74, 2012.

[5] Teuvo Kohonen and Teuvo Kohonen. Learning vector quantization. *Self-organizing maps*, pages 203–217, 1997.

[6] Rıfat AŞLIYAN. Examining variants of learning vector quantizations according to normalization and initialization of vector positions. *Avrupa Bilim ve Teknoloji Dergisi*, (45):8–13, 2022. `https://dergipark.org.tr/en/download/article-file/2844839`.

[7] Jianli Liu, Baoqi Zuo, Xianyi Zeng, Philippe Vroman, Besoa Rabenasolo, and Guangming Zhang. A comparison of robust bayesian and lvq neural network for visual uniformity recognition of nonwovens. *Textile research journal*, 81(8):763–777, 2011. `https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=969a198bf23fc6aaa8f9553063fc6a263604e443`.

[8] Atsushi Sato and Keiji Yamada. Generalized learning vector quantization. *Advances in neural information processing systems*, 8, 1995. `https://proceedings.neurips.cc/paper_files/paper/1995/file/9c3b1830513cc3b8fc4b76635d32e692-Paper.pdf`.

[9] Simon T Powers and Jun He. A hybrid artificial immune system and self organising map for network intrusion detection. *Information Sciences*, 178(15):3024–3042, 2008. `https://www.sciencedirect.com/science/article/pii/S0020025507005531?ref=pdf_download&fr=RR-2&rr=86a13f403ee3fc83`.

[10] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. Ieee, 2009.

[11] M. Hassan Zaib. NSL-KDD. *Kaggle*. `https://www.kaggle.com/datasets/hassan06/nslkdd` [Dostęp: 20.05.2024].