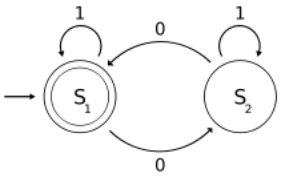
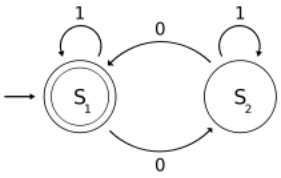


MATLAB

Klasyfikator

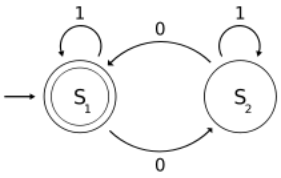
Dokumentacja projektu

Grupa projektowa:
Kamil Żak
Paweł Mitruś
Łukasz Napora
Mateusz Bukowski



MATLAB

Spis treści



MATLAB

1. Cel dokumentu

Celem dokumentu jest przedstawienie użytkownikowi sposobu działania Klasyfikatora. Dokument skupia się na wyjaśnieniu kwestii teoretycznej, metody rozwiązywania poszczególnych problemów oraz implementacji. Zawarte są również testy Klasyfikatora.

2. Środowisko programowania

Klasyfikator został zaimplementowany w środowisku MatLab w wersji R2012b (wersja dostępna w laboratorium komputerowym wydziału Matematyki i Nauki Informacyjnych Politechniki Warszawskiej. Głównym powodem wyboru tego środowiska były liczne operacje na macierzach. W ten sposób, z wiedzą że MatLab dysponuje optymalnymi algorytmami do wyżej wspomnianych działań, staraliśmy się podnieść wydajność programu.

3. Słownik pojęć

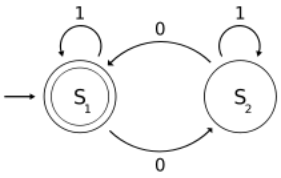
Pojęcie	Znaczenie
Klasa	Finalny obiekt do którego klasyfikujemy symbole (np. litery z języka polskiego)
Symbol	Reprezentant klasy
Cecha	Pewna wielkość matematyczna opisująca jedną z właściwości charakterystycznych dla symbolu z danej klasy
Liczba podziałów przedziału [0, 1]	Liczba, która informuje na ile podprzedziałów musimy dzielić przedział [0, 1]
Etap	Poszczególny etap oddawania projektu. Każdy poszczególny etap odnosi się do dwóch różnych zaimplementowanych automatów (z symbolami obcymi oraz bez symboli obcych)

4. Opis problemu

Problemem jest implementacja automatu klasyfikującego symbole z zadanych zbiorów do odpowiadających im klas na podstawie przedstawionych cech symboli.

Poszczególne etapy: 1, 2 oraz 3 różnią się wzajemnie typem implementowanego automatu. Odpowiednio dla:

1. Automat deterministyczny
    a. Bez symboli obcych
    b. Z symbolami obcymi
2. Automat niedeterministyczny
    a. Bez symboli obcych
    b. Z symbolami obcymi



3. Automat rozmyty
    a. Bez symboli obcych
    b. Z symbolami obcymi

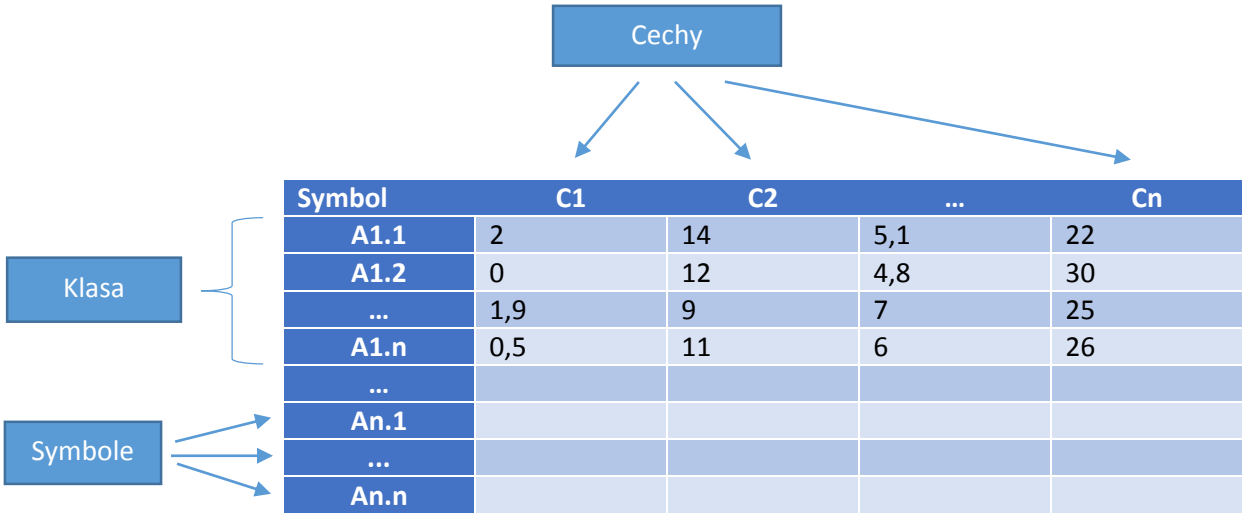
4.1. Reprezentacja i sposób klasyfikowania symboli

Do automatu wraz z każdym zbiorem wprowadzane są następujące wartości:

- Klasy: Zadany jest skończony zbiór klas $A = \{A1, A2, \dots, An\}$, do których finalnie przypisywane są testowane symbole.

- Symbole: To reprezentanci każdej z klas. Zadany jest skończony zbiór symboli $S = \{S1, S2, \dots, Sn\}$. Po próbach rozpoznania symbolu są one przypisywane do odpowiednich klas.

- Cechy: To pewne określone właściwości każdego z symboli. Na ich podstawie decydujemy, czy dany symbol klasyfikuje się do danej klasy. Zadany jest skończony zbiór cech $C = \{C1, C2, \dots, Cn\}$

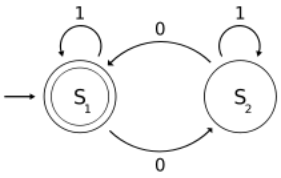


Symbol	C1	C2	…	Cn
A1.1	2	14	5,1	22
A1.2	0	12	4,8	30
…	1,9	9	7	25
A1.n	0,5	11	6	26
…				
An.1				
…				
An.n				

4.2. Konstrukcja i symulacja automatu

Każdy z etapów możemy opisać poniższym, ogólnym algorytmem. Kwestie, pomiędzy którymi poszczególne etapy się różnią, a w zasadzie są kluczowymi, zostaną opisane w punktach 4, 5, 6.

1. Generowanie/wczytywanie zbioru uczącego – każdy ze zbiorów zawiera wartości dla symboli oraz cech.



2. Generowanie/wczytywanie zbioru treningowego – są rozmiaru 1/3 ilości symboli odpowiadających im zbiorów uczących.

3. Generowanie automatu – różne w zależności od etapu, opisane w punktach 5, 6, 7.

4. Symulacja przejścia automatu dla zbioru uczącego – rozwiązanie ogólne. Różnice w zależności od etapu, opisane w punktach 5, 6, 7.

$$W_{n,n} * X_{n,1} = Y_{n,1}$$
$$y_k = \sum_{i}^{n} W_{k_i} * X_i$$

5. Optymalizacja macierzy przejścia automatu – wykonywana jest za pomocą algorytmu PSO bazując na zbiorze uczącym. Dokładny odpis w punkcie 3.3.

6. Obliczanie błędu – różne w zależności od etapu, opisane w punktach 5, 6, 7.

4.3. Optymalizacja macierzy przejścia automatu

Optymalizacji dokonujemy algorytmem PSO (Particle Swarm Optimization, Optymalizacja Rojem Cząstek), który uprzednio zaimplementowaliśmy.

1. Definicja problemu

   Prezentowane algorytmy rozwiązują problem poszukiwania

   minimum globalnego x^* funkcji ciągłej f

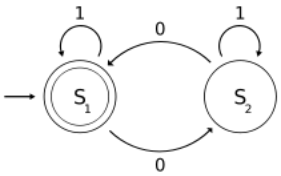
   • $f : R^n \rightarrow R$

   • $x^*, tż. \forall x \in R^n f(x) \geq f(x^*)$.

2. Idea
   - Algorytm opiera swoje działanie na roju cząstek
   - Każda cząsteczka ma swoją prędkość
   - Cząsteczki posiadają pewną bezwładność
   - Cząsteczki mogą być przyciągane przez najlepszy punkt znaleziony przez swoich sąsiadów
   - Cząsteczki są przyciągane przez najlepszy punkt znalezionych przez siebie
   - Cząsteczki mogą być odpychane od innych cząsteczek

3. Algorytm

W modelu PSO każda cząsteczka w kolejnych iteracjach musi wyliczyć swoją nową prędkość



i położenie. Wartość nowy wektor prędkości jest kompromisem między trzema wektorami:

- wektor dotychczasowej prędkości
- wektor ukierunkowany na najlepsze dotychczas znalezione położenie
- wektor ukierunkowany na najlepsze dotychczas znalezione przez sąsiadów położenie

Na podstawie modelu decyzji cząsteczki można wywnioskować ogólny wzór na jej nowe położenie.

$$v = wspol * v + c_1 r_1 (pbest - x) + c_2 r_2 (gbest - x)$$

x = x + v

gdzie

- v - wektor prędkość cząsteczki
- x - wektor położenia cząsteczki
- pbest - najlepsze znalezione przez cząsteczkę położenie
- gbest - najlepsze znalezione przez zbiór sąsiadów położenie
- c_1, c_2 - współczynniki określające wpływ poszczególnych elementów (jak bardzo ufa sobie, swojemu doświadczeniu i doświadczeniu swoich sąsiadów)
- r_1, r_2- wartości losowe z rozkładu jednostajnego U(0, 1)
- pbest−x - wektor odległości między obecnym położeniem, a najlepszym znalezionym
- dotychczas położeniem
- gbest−x - wektor odległości między obecnym położeniem, a najlepszym znalezionym
- dotychczas położeniem przez zbiór sąsiadów
- wspol – współczynnik maksymalnej prędkości poruszania cząsteczki

Wykorzystując wzór na wyliczanie nowej prędkości schemat algorytmu PSO prezentuje się następująco:

Wylosuj początkowe położenia i prędkości roju

while kryterium_zatrzymania

    dla każdej cząsteczki i

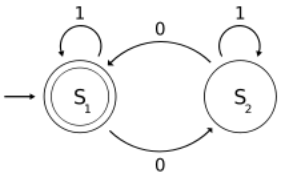
        zaktualizuj prędkość według wzoru na nową prędkość

        zaktualizuj pozycję według wzoru na nową pozycję

        jeżeli f(nowa_pozycja) < f(pbest_i):

            pbest_i = nowa_pozycja

        jeżeli f(nowa_pozycja) < f(gbest_i):



gbest_i = nowa_pozycja

4. Parametry

Algorytm wywołujemy z następującymi parametrami:

s_cnt	liczba klas
d_cnt	liczba podziałów
mtx	macierz testowanego zbioru
maxIterations	maksymalna liczba iteracji
pCount	liczba cząstek roju
wspol	maksymalna prędkość cząstek
u_bnd	ograniczenie górne przestrzeni przeszukiwania
c1	waga własna cząstki
c2	waga sąsiadów cząstki
fcHandle	funkcja błędu na podstawie, której optymalizujemy zbiór wyjściowy

5. Etap I

Różnice w automatach pomiędzy tymi bez symboli obcych, a automatami przyjmującymi zbiory z symbolami obcymi, zostaną opisane jako dodatki do poszczególnych punktów w celu uniknięcia kopiowania dużych ilości tekstu.

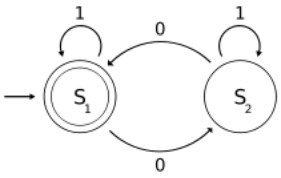
5.1. Generowanie/wczytywanie zbioru uczącego

1. Dla każdej z klas jest generowany losowo dokładnie jeden wektor cech rozkładem jednostajnym. Przedział losowania wartości ograniczony jest z góry i z dołu przez wczytywane parametry.
2. Wylosowany wektor wartości cech dla klasy jest powielany w zależności od wczytanej liczby symboli.
3. Następnie wartości w zbiorze zaburzane są szumem pochodzącym z rozkładu normalnego. Szum generowany jest losowo, a wartość oczekiwana oraz odchylenie standardowe są parametrami wejściowymi dla funkcji generującej.
4. W kolejnym kroku zbiór cech jest normalizowany do przedziału [0,1]
5. W zależności od wczytanej liczby podziałów przedziału [0, 1], dla każdej wartości cechy w zbiorze, przypisywana jest całkowita liczba porządkowa. Nowa wartość jest równa numerowi przedziału, w którym zawierała się poprzednia wartość cechy.

   Dla przykładu: jeżeli liczba podziałów przedziału [0, 1] wynosi 5, otrzymujemy zbiór przedziałów

   $$S = \{S_1 = [0, 0.2), S_2 = [0.2, 0.4), S_3 = [0.4, 0.6), S_4 = [0.6, 0.8), S_5 = [0.8, 1]\}$$

   Jeżeli cecha $C_1 \in S_i$ to zostaje jej przypisana wartość '*i*'.



6. Następnie, tworzona jest kolumna symboli w zależności od wczytanych parametrów (liczba klas, liczba symboli w klasie). Klasy zawsze przedstawiane są jako 1, 2, …, n.

Automat z elementami obcymi:
W przeciwieństwie do punktów 1 i 2 wektory cech dla symboli obcych nie są powielane – dla każdego symbolu losowany jest osobno wektor cech. Kolejno wektory dodawane są do macierzy zbioru uczącego.

Wczytywanie zbioru z pliku:
Wykonujemy tylko kroki opisane w punktach 4 i 5.

5.2. Generowanie/wczytywanie zbioru treningowego

1. Dla każdej z klasy wybieramy losowo 1/3 wektorów cech (symboli z tej klasy) ze zbioru uczącego.
2. Łączymy wylosowane wektory cech w nową macierz zbioru treningowego.

Automat z elementami obcymi:
Wybieramy losowo 1/3 wektorów cech (symboli obcych) z całego zbioru uczącego i dołączamy je do macierzy wylosowanej w 5.2.

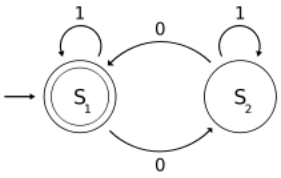
Wczytywanie zbioru z pliku:
Zakładamy, że klasy są takie same (jest ich tyle samo) w obu zbiorach – uczącym i treningowym. Nazwy klas tworzymy automatycznie w programie (potrzebujemy nazw klas od 1, 2, …, n), a następnie jest to tłumaczone na nazwy jakie były w pliku *.xlsx. Zakładamy, że typy danych są takie same jak w przykładowych plikach (liczbowe, tekstowe).

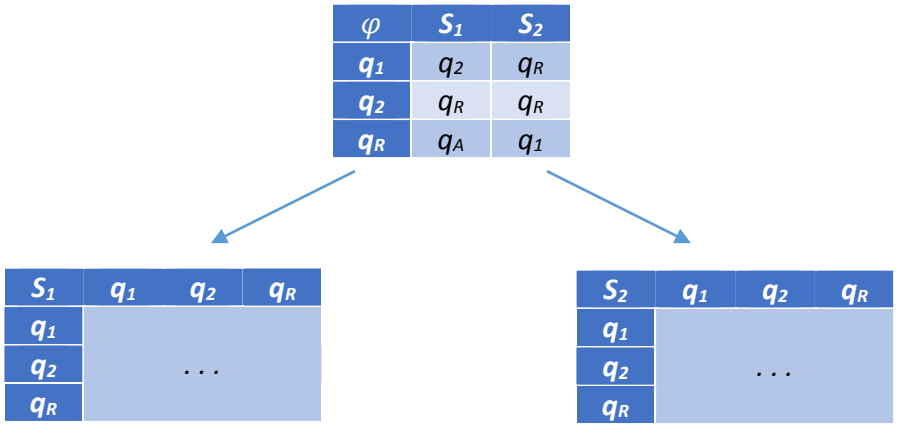
5.3. Generowanie automatu

Tabelę funkcji przejścia automatu zamieniamy na macierz 3-wymiarową, gdzie nowopowstały wymiar równy jest ilości podziałów przedziału [0, 1]. W wyniku tej transformacji otrzymujemy tabele funkcji przejścia dla każdego możliwego symbolu wejściowego. Pojedyncza tabela przejścia określa, z którego stanu możemy przejść w jaki stan, czytając na wejściu symbol z alfabetu.

W tabeli przejścia automatu znajduje się tyko jedna 1 w każdej z kolumn, ponieważ automat jest deterministyczny i w jednym momencie może znajdować się tylko w jednym stanie.

Automat z elementami obcymi:
Dodany jest nowy stan dla wykrywania wszystkich elementów obcych.





5.4. Symulacja przejścia automatu dla zbioru uczącego

Funkcje mnożenia i dodawania z rozwiązania ogólnego są zamienione odpowiednio na funkcje minimum i maksimum.

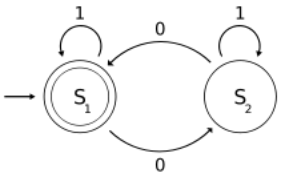
5.5. Obliczanie błędu

1. Otrzymujemy stan w jakim znalazł się automat po symulacji działania automatu
2. Jeżeli symbol nie został rozpoznany, zwiększamy licznik nierozpoznanych symboli, a w przypadku gdy został rozpoznany nie robimy nic.
3. Obliczenia powtarzane są dla każdego elementu z klasy.
4. Liczony jest procentowy błąd w klasyfikowaniu symboli, który wyrażamy jako iloraz elementów nierozpoznanych do liczby wszystkich elementów wszystkich klas.

6. Etap II

6.1. Generowanie\wczytywanie zbioru uczącego

1. Dla każdej z klas jest generowany losowo dokładnie jeden wektor cech rozkładem jednostajnym. Przedział losowania wartości ograniczony jest z góry i z dołu przez wczytywane parametry.
2. Wylosowany wektor wartości cech dla klasy jest powielany w zależności od wczytanej liczby symboli.
3. Następnie wartości w zbiorze zaburzane są szumem pochodzącym z rozkładu normalnego. Szum generowany jest losowo, a wartość oczekiwana oraz odchylenie standardowe są parametrami wejściowymi dla funkcji generującej.
4. W kolejnym kroku zbiór cech jest normalizowany do przedziału [0,1]
5. W zależności od wczytanej liczby podziałów przedziału [0, 1], dla każdej wartości cechy w zbiorze, przypisywana jest całkowita liczba porządkowa. Nowa wartość jest równa numerowi przedziału, w którym zawierała się poprzednia wartość cechy.



Dla przykładu: jeżeli liczba podziałów przedziału [0, 1] wynosi 5, otrzymujemy zbiór przedziałów

$$S = \{S_1 = [0, 0.2), S_2 = [0.2, 0.4), S_3 = [0.4, 0.6), S_4 = [0.6, 0.8), S_5 = [0.8, 1]\}$$

Jeżeli cecha $C_1 \in S_i$ to zostaje jej przypisana wartość '*i*'.

6. Następnie, tworzona jest kolumna symboli w zależności od wczytanych parametrów (liczba klas, liczba symboli w klasie). Klasy zawsze przedstawiane są jako 1, 2, …, n.

Automat z elementami obcymi:
W przeciwieństwie do punktów 1 i 2 wektory cech dla symboli obcych nie są powielane – dla każdego symbolu losowany jest osobno wektor cech. Kolejno wektory dodawane są do macierzy zbioru uczącego.

Wczytywanie zbioru z pliku:
Wykonujemy tylko kroki opisane w punktach 4 i 5.

6.2. Generowanie/wczytywanie zbioru treningowego

1. Dla każdej z klasy wybieramy losowo 1/3 wektorów cech (symboli z tej klasy) ze zbioru uczącego.
2. Łączymy wylosowane wektory cech w nową macierz zbioru treningowego.

Automat z elementami obcymi:
Wybieramy losowo 1/3 wektorów cech (symboli obcych) z całego zbioru uczącego i dołączamy je do macierzy wylosowanej w 6.2.

Wczytywanie zbioru z pliku:
Zakładamy, że klasy są takie same (jest ich tyle samo) w obu zbiorach – uczącym i treningowym. Nazwy klas tworzymy automatycznie w programie (potrzebujemy nazw klas od 1, 2, …, n), a następnie jest to tłumaczone na nazwy jakie były w pliku *.xlsx. Zakładamy, że typy danych są takie same jak w przykładowych plikach (liczbowe, tekstowe).

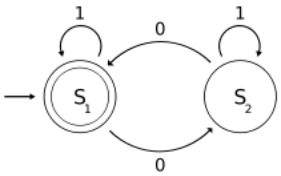
6.3. Generowanie automatu

Tabelę funkcji przejścia automatu zamieniamy na macierz 3-wymiarową, gdzie nowopowstały wymiar równy jest ilości podziałów przedziału [0, 1]. W wyniku tej transformacji otrzymujemy tabele funkcji przejścia dla każdego możliwego symbolu wejściowego. Pojedyncza tabela przejścia określa, z którego stanu możemy przejść w jaki stan, czytając na wejściu symbol z alfabetu.

W tabeli przejścia automatu w każdej z kolumn liczba 1 zawiera się w przedziale od 0 do maksymalnego wypełnienia kolumny (jest ono normowane przez parametr określający procent elementów obcych w stosunku do całości), ponieważ automat jest niedeterministyczny i w jednym momencie może znajdować się w więcej niż jednym stanie.

6.4. Symulacja przejścia automatu dla zbioru uczącego

Funkcje mnożenia i dodawania z rozwiązania ogólnego są zamienione odpowiednio na funkcje minimum i maksimum.



6.5. Obliczanie błędu

1. Otrzymujemy wektor stanów po symulacji działania automatu
2. Jeżeli symbol nie został rozpoznany, zwiększamy licznik nierozpoznanych symboli, a w przypadku gdy został rozpoznany nie robimy nic.
3. Obliczenia powtarzane są dla każdego elementu z klasy.
4. Liczony jest procentowy błąd w klasyfikowaniu symboli, który wyrażamy jako iloraz elementów nierozpoznanych do liczby wszystkich elementów wszystkich klas.

7. Etap III

7.1. Generowanie zbioru uczącego

1. Dla każdej z klas jest generowany losowo dokładnie jeden wektor cech rozkładem normalnym. Wartość oczekiwana oraz odchylenie standardowe określone jest przez wczytywane parametry.
2. Wylosowany wektor wartości cech dla klasy jest powielany w zależności od wczytanej liczby symboli.
3. Następnie wartości w zbiorze zaburzane są szumem pochodzącym z rozkładu normalnego. Szum generowany jest losowo, a wartość oczekiwana oraz odchylenie standardowe są parametrami wejściowymi dla funkcji generującej.
4. W kolejnym kroku zbiór cech jest normalizowany do przedziału [0,1]
5. Następnie każda z cech jest „rozszerzana” do wektora o długości równej ilości podziałów przedziału [0, 1]. W wektorze umieszczamy wartości wg wzoru

$$arr[i] = e^{-((a-b)*(a-b))}$$

gdzie a to wartość cechy ze zbioru postaci w punkcie 4 oraz

$$b = i * \left(\frac{1}{liczba.podzialow}\right) + \frac{1}{2 * liczba.podzialow}$$

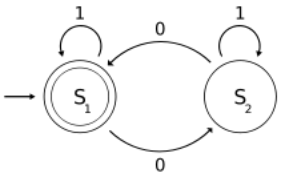
6. Następnie, tworzona jest kolumna symboli w zależności od wczytanych parametrów (liczba klas, liczba symboli w klasie). Klasy zawsze przedstawiane są jako 1, 2, …, n. Kolumna powielana jest w macierz tak aby pasowała do 3-wymiarowej macierzy zbioru uczącego.

Automat z elementami obcymi:
W przeciwieństwie do punktów 1 i 2 wektory cech dla symboli obcych nie są powielane – dla każdego symbolu losowany jest osobno wektor cech. Kolejno wektory dodawane są do macierzy zbioru uczącego.

Wczytywanie zbioru z pliku:
Wykonujemy tylko kroki opisane w punktach 4 i 5.

7.2. Generowanie/wczytywanie zbioru treningowego

1. Dla każdej z klasy wybieramy losowo 1/3 macierzy cech (symboli z tej klasy) ze zbioru uczącego.



2. Łączymy wylosowane macierze cech w nową macierz 3-wymiarową zbioru treningowego.

Automat z elementami obcymi:
Wybieramy losowo 1/3 macierzy cech (symboli obcych) z całego zbioru uczącego i dołączamy je do macierzy wylosowanej w 7.2.

Wczytywanie zbioru z pliku:
Zakładamy, że klasy są takie same (jest ich tyle samo) w obu zbiorach – uczącym i treningowym. Nazwy klas tworzymy automatycznie w programie (potrzebujemy nazw klas od 1, 2, …, n), a następnie jest to tłumaczone na nazwy jakie były w pliku *.xlsx. Zakładamy, że typy danych są takie same jak w przykładowych plikach (liczbowe, tekstowe).

7.3. Generowanie automatu

Z racji że mamy do czynienia z automatem rozmytym losowane są wartości z przedziału [0 liczba.podzialow].

7.4. Symulacja przejścia automatu dla zbioru uczącego

1. Do automatu zostaje przekazana macierz cech danego symbolu.
2. Zaczynamy od stanu początkowego stan.poczatkowy = [1 0 0 ... 0]
3. Dla każdej z cech iterujemy po wektorze ją reprezentującym
4. Przy użyciu funkcji:

$$arr(a) = 1 - \tanh(\operatorname{atanh}\left(1 - macierz.przejscia(i,a,k) + atanh\left(1 - stan.poczatkowy(a)\right)\right))$$

Iterujemy przez wszystkie wartości w stan.poczatkowy i następnie z użyciem funkcji

$$arr(b-1) = \tanh(\operatorname{atanh}\left(arr(b-1)\right) + \operatorname{atanh}(arr(b))$$

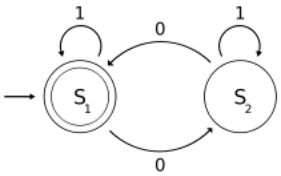
tworzymy stan wejściowy dla kolejnej funkcji przejścia automatu.

5. Zwracamy stworzony wektor stanów

W poprzednich etapach do symulacji automatu przekazywane były reprezentacje symbolu (wektor cech dla symbolu). W tym przypadku przekazujemy macierz cech (wektor różnych cech oraz wektory reprezentujące każdą z pojedynczych cech).

7.5. Obliczanie błędu

1. Otrzymujemy wektor stanów po symulacji działania automatu
2. Jeżeli symbol nie został rozpoznany, nie zwiększamy licznika rozpoznanych symboli, a w przypadku gdy został rozpoznany zwiększamy go o $(1 - \frac{wektor.stanow(i)}{suma(wektor.stanow)})$
3. Obliczenia powtarzane są dla każdego symbolu
4. Liczony jest procentowy błąd w klasyfikowaniu symboli



8. Testy – w załączniku

9. Format uruchomieniowy programu

TAIO2014('etap', 'a1', 'wejscieTyp', 'gen', 'iloscKlas', 6, 'iloscCech', 15, 'iloscPowtorzenWKlasie', 20, 'minLos', 0, 'maxLos', 20, 'zaburzenie', 0.2, 'procRozmTest', 20, 'dyskretyzacja', 5, 'PSOiter', 900, 'PSOs', 19)