Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

PRACA MAGISTERSKA

TOMASZ PAŁOSZ

SYSTEM AUTOMATYCZNEGO ROZPOZNAWANIA UŻYTKOWNIKA KOMPUTERA NA PODSTAWIE SPOSOBU PISANIA NA KLAWIATURZE I SPOSOBU OPEROWANIA MYSZKĄ

PROMOTOR:
dr Adrian Horzyk

Kraków 2009

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W PRACY.

.....

PODPIS

AGH
University of Science and Technology in Krakow

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

MASTER OF SCIENCE THESIS

TOMASZ PAŁOSZ

AUTOMATIC SYSTEM USER AUTHENTICATION BASED ON KEYSTROKE DYNAMICS AND MOUSE GESTURES

SUPERVISOR:
Adrian Horzyk Ph.D

Krakow 2009

Serdecznie dziękuję promotorowi
dr Adrianowi Horzykowi
za pracę i czas poświęcony na dyskusje
nad algorytmem oraz za możliwość
realizacji ciekawego tematu.

Dziękuję również Aleksandrze, Barbarze, Róży, Jackowi, Łukaszowi, Mateuszowi, Michałowi za czas poświęcony na testy.

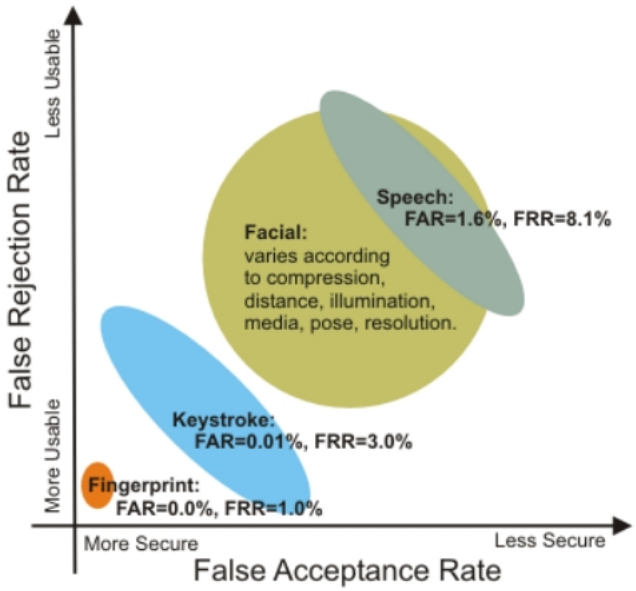
Spis treści

1. Wprowadzenie

Będąc na I roku studiów rozpocząłem działalność w kole naukowym. Zainteresowałem się wtedy, tematem rozpoznawania użytkowników po sposobie pisania. Pod koniec I roku rozpocząłem pisanie aplikacji zbierającej dane. Była to prosta aplikacja napisana w języku C, uruchamiana spod konsoli na linuxie. Analizowała ona tylko czasy pomiędzy wciśnięciami kolejnych przycisków. Baza była małych rozmiarów, a mimo to wyniki rozpoznania były nienajgorsze.

Gdy poznałem bazy danych i trochę więcej technologii chciałbym teraz napisać bardziej rozbudowany system rozpoznawania użytkowników po sposobie pisania na klawiaturze. Obecnie mogę wykorzystać bazy danych, większe środowiska programistyczne, a także języki wyższych poziomów niż C.

Zatem jednym z pierwszych powodów wyboru takiego tematu pracy magisterskiej były własne doświadczenia na obszarze dynamiki pisania oraz systemów bezpieczeństwa. Kolejnym z powodów wyboru tego tematu jest wysoka skuteczność rozpoznawania ludzi podczas badania właśnie tej cechy behawioralnej. Miejsce dynamiki pisania (*keystroke*) na tle innych sposobów rozpoznawania ludzi przedstawia rysunek 1.1 (rysunek zaczerpnięty z [1]).



Rysunek 1.1: Efektywność rozpoznawania po sposobie pisania na klawiaturze na tle innych metod

1.1. Postawienie Problemu

Pierwszym z problemów jest zbieranie danych do bazy. W jaki sposób zbierać dane, aby były jak najmniej zakłócone? W szczególności, aby czas wciśnięcia klawisza oraz czas pomiędzy kolejnymi wciśnięciami był jak najmniej zakłócony.

Kolejnym problemem jest wybór algorytmu analizy danych. Można zastosować prostą analizę statystyczną, licząc odchylenie standardowe. Można także zastosować bardziej złożone algorytmy wykorzystując metody sztucznej inteligencji, np. metodę Support Vector Machine z 26 wymiarami. Najbardziej zaawansowanym rozwiązaniem jest stworzenie sztucznej sieci neuronowej. Takie podejście jest obecnie stosowane przez większość zespołów zajmujących się tym problemem i takie podejście zastosuję w tej pracy.

W przypadku rozpoznawania po zalogowaniu, wiemy jaki jest użytkownik i problem sprowadza się do potwierdzenia tożsamości tego użytkownika - określeniu progu, powyżej którego stwierdzamy autentyczność użytkownika. W przypadku braku informacji o użytkowniku algorytm wymaga znalezienia osoby z bazy danych, dla której prawdopodobieństwo poprawności jest największe.

1.2. Główny Cel

Głównym celem pracy magisterskiej jest napisanie aplikacji, która służyłaby do autentykacji użytkowników podczas codziennej pracy na komputerze. Dodatkowo podczas używania klawiatury przez użytkownika program zbierałby dane do bazy i uczyłby sieć neuronową tymi danymi.

Kolejnym celem jest uzyskanie możliwie jak najniższego wskaźnika błędnego odrzucenia (FRR). Błąd akceptacji jest mniej istotny i może wynosić do 30%. Tak niski błąd odrzucenia zapewni dużą użyteczność programu. W ten sposób interakcja z użytkownikiem zostanie zminimalizowana.

Ważną cechą jest prostota działania algorytmu. Dopuszczalne będą drobne zmiany w istniejących algorytmach. Służące przede wszystkim sprawdzeniu, czy nie zwiększają one skuteczności algorytmu.

2. Istniejące rozwiązania

Idea rozpoznawania ludzi po sposobie pisania zrodziła się podczas II wojny światowej. Potwierdzają się słowa, że motorem rozwoju jest wojna. Wtedy to telegrafiści rozpoznawali się po sposobie nadawania wiadomości, długości kropek i kresek w alfabecie morsa.

W obecnej chwili istnieje kilka patentów rozpoznawania. Numery patentów to, np. 4621344, 5557686, 4805222, 4962530, 4998279 i 5056141. Udało mi się dotrzeć do kilku rozwiązań. W kolejnych rozdziałach zamieszczam tłumaczenia konspektów z poszczególnych rozwiązań wraz z autorami. Na końcu zamieszczam, dodatkowo, tytuły oraz autorów prac, z których korzystałem, a których opis pochłonąłby sporą część pracy magisterskiej.

2.1. Biometric Authentication Tool for User Identification Based on Keystroke Dynamics [3]

2.1.1. Biometryka pisania

Rozpoznanie dynamiki pisania na klawiaturze jest rozwiązaniem wyłącznie programowym. Obejmuje 2 procesy:

1. Proces rejestracji: Proces ten dodaje użytkownika i generuje jego wzór. Użytkownik musi podać login i hasło.

2. Proces weryfikacji: Proces ten będzie weryfikował użytkownika. Będzie dopasowywał wprowadzoną próbkę do wcześniej ustalonych wzorców.

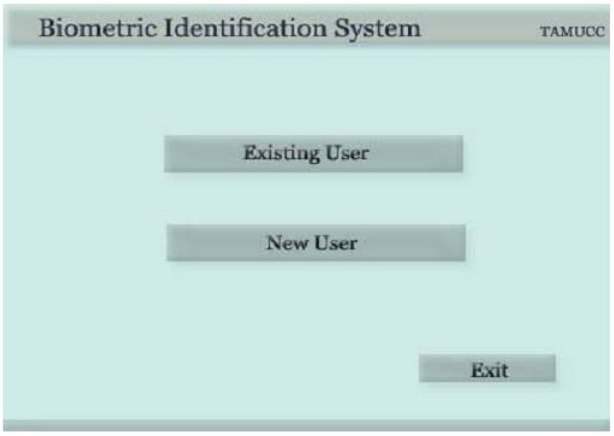
Badania skupiają się na wypracowaniu tanich i nieuciążliwych metod do biometrycznego rozpoznawania na podstawie dynamiki pisania w celu autentykacji użytkowników. Motywacją do użycia tej technologii było całkowicie programowe rozwiązanie do zbudowania systemu autentykacji. System posiada następujące cechy:

1. Nie jest wymagany dodatkowy sprzęt, można użyć posiadaną klawiaturę,
2. Nie jest wymagane dodatkowe szkolenie dla użytkowników,
3. Łatwy w użyciu, szybka konfiguracja. Użycie w systemie operacyjnym lub w sieci,
4. Dane są przechowywane w formie wzorców.

2.1.2. Opis projektu

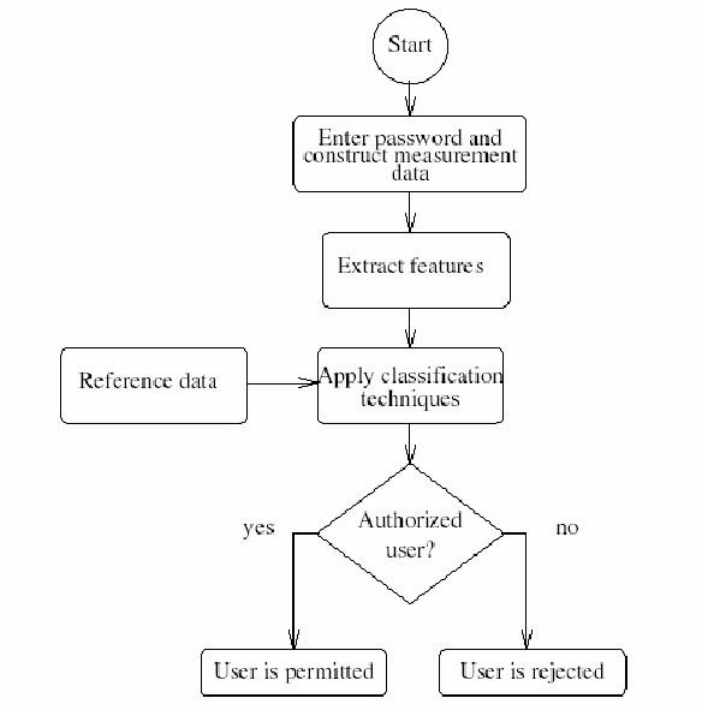
System pyta użytkownika o 3 wyrażenia login, hasło i zadana fraza. Zadana fraza jest stała, dodana w celu wzmocnienia bezpieczeństwa, jest sprawdzana w przypadku każdego użytkownika. Ekran logowania jest zaprezentowany na rysunku poniżej. W przypadku nowego użytkownika, rozpoczynany jest proces rejestracji i system prosi o podanie loginu, hasła i oczywiście frazy. Te informacje są przechowywane w pliku konfiguracyjnym. W celu wejścia do systemu użytkownik jest proszony o podanie kilka razy tych samych danych. Forma logowania widoczna jest poniżej.

Proces logowania jest równocześnie procesem weryfikacji, podczas którego dynamika pisania użytkownika jest porównywana z przechowywanym wzorcem. Jeśli użytkownik zostanie rozpoznany wtedy zostaje poinformowany o poprawnym zalogowaniu. Poszczególne kroki są zaprezentowane na schemacie blokowym.



Rysunek 2.1: Ekran startowy - wejście do systemu

Rysunek 2.2: Ekran logowania



Rysunek 2.3: Schemat blokowy rozpoznawania po sposobie pisania na klawiaturze

2.1.3. Charakterystyka wartości mierzonych przez program

1. Łączna szybkość pisania,

2. Czas pomiędzy wciśnięciami klawiszy,

3. Czas wciśnięcia klawiszy,

4. Częstotliwość użycia poszczególnych klawiszy takich jak numery lub klawisze funkcyjne

5. Indywidualne sekwencje (przyzwyczajenia), np. czy shift zwalniany jest przed zwolnieniem następnego klawisza czy po.

Powyższe wartości tworzą statystykę profilu, a następnie brane są do utworzenia wzorca użytkownika. Wzorzec przechowuje również login i hasło.

2.1.4. Projekt algorytmu autentykacji

Algorytm wykorzystuje każdy z 3 wprowadzonych przez użytkownika łańcuch znaków. Algorytm używa dwóch zmiennych, “test“ oznaczona jako T, zawierająca dane zbierane podczas logowania oraz zmienna ”mean“ oznaczona jako M, gdzie $M = M_{username}, M_{password}, M_{phrase}$. Weryfikacja polega na porównaniu sygnatur T i M oraz na określeniu różnicy pomiędzy nimi. Dane M = m1, m2, ..., mn i T = t1, t2, ..., tn, gdzie n to ilość czasów w próbce. Algorytm wylicza różnicę i pozytywne rozpoznanie następuje, gdy jest większe niż zadany próg. Próg ten wynosi 80%.

2.1.5. Wygląd systemu

Graficzny interfejs użytkownika jest zaprojektowany do autentykacji. Składa się z modułu logowania oraz modułu weryfikacji. Dla nowych użytkowników moduł logowania wywołuje moduł rejestracji. System stworzono używając Microsoft Developer z Visual Studio .Net. Dla użytkownika proces weryfikacji i logowania wygląda identycznie. Poniższe rysunki przedstawiają formę logowania oraz komunikat po poprawnym logowaniu. Weryfikacja może nastąpić po kilkukrotnym zalogowaniu.

Rysunek 2.4: Przykładowe logowanie użytkownika Johna Smitha

Rysunek 2.5: Poprawna weryfikacja użytkownika Johna Smitha

Pierwszym etapem w procesie weryfikacji jest porównanie czasów wciśnięć i zwolnień każdego przycisku. Następnie wyliczane jest standardowe odchylenie dla każdej akcji. Dla zgodności loginu wystarczy 50-procentowa zgodność.

2.1.6. Efektywność dynamiki pisania

Błędne zaakceptowanie (False Acceptance Rate - FAR) określa, jak często nieupoważniona osoba może zostać uwierzytelniona. Mniejsza wartość oznacza większe bezpieczeństwo.

Błędne odrzucenie (False Rejection Rate - FRR) określa, jak często upoważniona osoba nie zostanie zweryfikowana poprawnie. Wysoka wartość zmniejsza użyteczność (zmusza do ponownego logowania).

Przecięcie błędów (cross-over error rate - CER) to punkt, w którym FAR oraz FRR są równe. Najlepsze technologie mają najmniejszy punkt przecięcia FAR oraz FRR.

2.1.7. Wnioski

Dynamika pisania jest ekonomiczną i dobrą metodą do biometrycznej identyfikacji osób. Sposób rozpoznawania jest prosty i efektywny. W tej implementacji istniał problem z błędnym odrzuceniem (FRR) podczas testów. Błędna akceptacja wynosiła 0%, co oznacza, że żaden intruz nie uzyskał dostępu do danych. Podsumowując, projekt odniósł sukces pokazując, że w oparciu o dynamikę pisania można stworzyć system autentykacji użytkowników.

2.2. Password Secured Sites - Stepping Forward With Keystroke Dynamics [2]

2.2.1. Dynamika pisania - wcześniejsza praca

Jak w wielu problemach tak i tutaj pojawiły się 2 podejścia do rozwiązania problemu znalezienia algorytmu, który minimalizowałby CER (cross-over error rate - przecięcie błędów FAR i FRR): algorytmy sztucznej inteligencji oraz deterministyczne algorytmy.

Jako przykład rozwiązania stosującego algorytm sztucznej inteligencji możemy znaleźć pracę prezentowaną przez Chena, który osiągnął przecięcie błędów (CER) mniejsze niż 1% i 0% błędnego zaakceptowania (FAR). ORD i Furnell także testowali tą technologię na 14 osobowej grupie w celu zbadania jej przydatności do zastosowania podczas podawania PINów wpisywanych na klawiaturze numerycznej. Niestety rezultaty sugerują, że zastosowanie technologii dla większej grupy ludzi jest niemożliwe.

Algorytmy deterministyczne były stosowane od późnych lat 70. W 1980 roku Gaines zaprezentował swój raport z pracy badającej wzorce pisania siedmiu osób profesjonalnie piszących na klawiaturze. Mała liczba chętnych oraz fakt, że algorytm został wymyślony na podstawie zebranych danych i nie testowany później na innych osobach świadczą o małej wiarygodności. Mimo to metody użyte do sporządzenia wzorców były przełomem: badanie czasu poświęconego na przepisanie dwa razy tego samego tekstu. Od tego czasu zostało zaprezentowanych wiele algorytmów bazujących na algebrze, probabilistyce oraz statystyce. W 1990 roku Joyce Gupta zaprezentował algorytm, który wyliczał różnicę między czasami w zebranej próbce, a czasami przechowywanymi z wcześniejszych prób. W 1997 Monrose i Rubin użyli odległości euklidesowej i probabilistyki przyjmując, że czasy mają rozkład normalny. W 2000 roku zaprezentowali oni algorytm identyfikacji oparty o model podobieństwa Bayesa, a w 2001 roku zaprezentowali algorytm, który używa wielomianów oraz przestrzeni wektorowych do wygenerowania grupy haseł dla pojedynczej osoby. Użyli przy tym wzorców pisania.

Przytoczone algorytmy są niewielkim przykładem z bogactwa dostępnych posiadających przyzwoity wskaźnik przecięcia błędów (CER). Inne algorytmy posiadają różną liczbę możliwych użytkowników (zazwyczaj z określonym maksymalnym limitem), różną długość próbki potrzebnej przy dodawaniu użytkownika, różną ilość powtórzeń wpisywanej frazy testowej. Taka różnorodność sprawia, że porównywanie algorytmów między sobą jest trudne. Nie jest określona jednoznacznie próbka danych. Te same

algorytmy osiągają różne wyniki na tej samej grupie osób. Jedynym rozwiązaniem porównywania algorytmów jest testowanie ich na tej samej grupie osób. Przewidując szerokie zastosowanie aplikacji do rozpoznawania użytkowników, np. aplikacje internetowe muszą brać pod uwagę dużą liczbę użytkowników. W takiej sytuacji jednym z najważniejszych czynników jest szybkość wykonywanych operacji.

Pomimo to, zgodnie z Peacockiem, określili błędne zaakceptowanie (FAR) od 0% do 50%, błędne odrzucenie (FRR) od powyżej 25% do mniej niż 1%, przy czym ilość użytkowników wahała się między 10, a 100.

2.2.2. Szybki algorytm

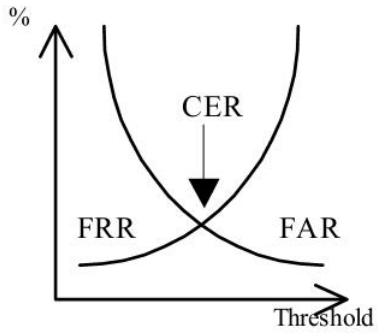
Algorytm zaprezentowany w [2] jest szybki w przypadku dodawani oraz autentykacji użytkownika. Proces dodawania użytkownika, wykonywany raz podczas pierwszego użycia, składa się z wpisania hasła (frazy składającej się na hasło) 12 razy. Dane są przechowywane, wyliczana jest średnia, mediana oraz odchylenie standardowe dla każdych dwóch znaków, która także jest przechowywana. Przechowywana jest także średnia, mediana oraz odchylenie standardowe dla czasu wpisania całego hasła.

Proces autentykacji z punktu widzenia użytkownika to wprowadzenie hasła. Dla każdego wciśnięcia klawisza algorytm zmierzy czas oczekiwania, zdefiniowany jako TLP, i porówna go z przechowywanym czasem oczekiwania. Porównanie będzie trafne, jeśli Minśrednia, mediana * (0.95 - odchylenie/średnia) <= TLP oraz TLP <= Maxśrednia, mediana * (1.05 + odchylenie/średnia).

Te same obliczenia są wykonywane dla każdego hasła i wyniki przechowywane są w tablicy zero-jedynkowej.

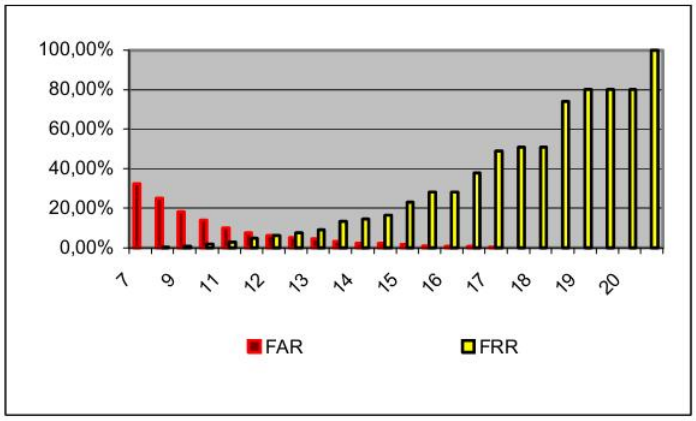
Następnie liczona jest suma A. Wartości nietrafione nie są brane do sumy. Wartość 1 jest dodawana jeśli poprzednia wartość była nietrafiona (lub jeśli jest to pierwsza wartość) lub wartość 1,5 jest dodawana jeśli poprzednia wartość była trafiona. Ostatecznie wartość sumy A decyduje czy proces autentykacji się powiedzie, zgodnie z progiem ustalonym przez administratora. Dla przykładu jeśli próg wynosi 70% wtedy autentykacja się powiedzie, tylko wtedy gdy suma A w stosunku do największej możliwej wartości ((liczba znaków - 1)*1.5+1) będzie większa od 70%. Jeśli próbka zostanie zaakceptowana, to stare wartości czasów zostają zastąpione nowymi. Ta procedura umożliwia ewolucje (zmienność) przechowywanych danych.

Dokładność systemów biometrycznych mierzona jest za pomocą błędu akceptacji (FAR), który mierzy procentową ilość błędnych zalogowań do ilości poprawnych zalogowań; oraz za pomocą błędu odrzucenia (FRR), który mierzy procentową ilość odrzuconych zalogowań. Przecięcie błędów (CER) wyznacza nam punkt, w którym FAR oraz FRR są równe. Im mniejsze FAR w tym punkcie tym lepszy jest algorytm.



Rysunek 2.6: Błędy - FRR, FAR oraz CER

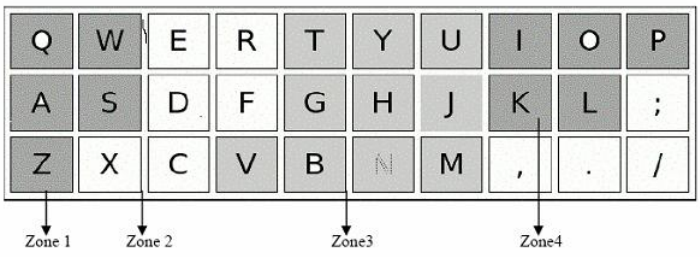
Przedstawiony algorytm osiągnął CER na poziomie 5,58% oraz potrafi osiągnąć, na najmniejszym progu akceptacji, wartość FRR bliską 0%, co zapewnia maksymalny komfort użytkowania. Przy najwyższym progu akceptacji algorytm osiągam wartość FAR bliską 0%, maksymalizując w ten sposób bezpieczeństwo. Wyniki przedstawione są na rysunku poniżej.



Rysunek 2.7: Błędy uzyskane z pomiarów

2.2.3. Wskaźnik wydajności

Revett i Khan [11] stwierdzili, że podział klawiatury na obszary zmniejsza błędy akceptacji (FAR). Zgodnie z wynikami, jeśli litery zawarte w przepisywanym tekście pochodzą z każdego fragmentu klawiatury (innymi słowy tekst jest rozsypany po całej klawiaturze), to trafność określenia poprawnej osoby jest największa. Ponadto wpływ ma także szybkość pisania. Zgodnie z wynikami, najlepsze rezultaty osiąga się, gdy użytkownik nie pisze z maksymalną prędkością.



Rysunek 2.8: Strefy aktywności na klawiaturze zaproponowane przez Revetta i Khana

Nakładanie haseł lub określanie szybkości pisania na użytkownika zmniejsza komfort i powoduje stratę 1% użytkowników w przypadku aplikacji internetowych. W niektórych przypadkach 1% może oznaczać tysiące. Czynnik ten zwiększa rangę utrzymania takich systemów bardziej komfortowymi, mniej narażonymi na ataki i przede wszystkim z mniejszym błędem odrzucenia (FRR). W związku z tym algorytmy dynamiki pisania powinny znaleźć rozwiązania zapewniające zmianę wzorców wraz z przyzwyczajeniami użytkowników.

2.2.4. Nowy algorytm zapamiętujący czasy wciśnięć

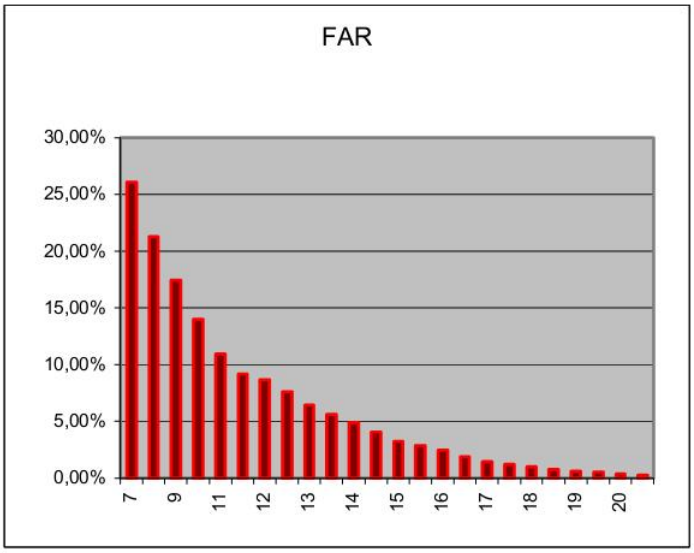
Potrzeba dodania do procesu decyzyjnego określenia wskaźnika wydajności użytkownika (taki sam wskaźnika dla każdego, ale o innej wartości) zrodziła potrzebę zapamiętywania czasów wciśnięć klawiszy. Proponujemy wzbogacić wielowymiarowy wektor wyliczający sumę A o kilka zmiennych, które będą musiały być określone dla każdego użytkownika. Niestety nie możemy przyjąć, że użytkownicy internetowi wiedzą jak są rozpoznawani, zatem musimy zróżnicować zmienne i znaleźć ich stałe wartości (k), które będą maksymalizowały "internetowy"punkt przecięcia błędów (CER) ("internetowy- uwzględniający czasy komunikacji). Zatem teraz

$A = suma_i = 1^n(k_1 k_2 ... k_p f(t_n))$, gdzie n jest ilością zebranych czasów oczekiwania, p jest ilością przyjętych wskaźników wydajności, $f(t_n)$ jest rezultatem zastosowania wcześniej opisanego algorytmu

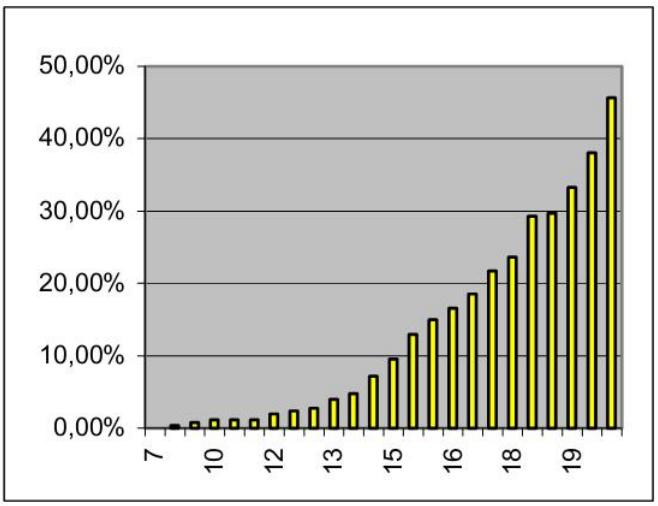
dla n-tego czasu oczekiwania. Praca zaprezentowana w Ęnhancing login security using keystroke hardening and keyboard gridding"[11] sugeruje, że dobrym wskaźnikiem jakości jest średnia szybkość pisania (zebrane dane sugerują, że wynosi ona 2). Intuicja podpowiada nam, że praworęczni będą pisali systematyczniej tekst, gdy litery będą się zmieniały z lewej strony do prawej. Cytując pracę [11] użytkownik pisze regularniej, gdy litery pochodzą z jednego obszaru na klawiaturze. Zatem zdecydowaliśmy dodać określoną wartość do czasów, podczas zmiany liter z lewej części klawiatury na prawą. Także, w tym przypadku najlepszą wartością jest 2. Rezultatem tych zabiegów jest zmniejszenie punktu przecięcia błędów (CER).

2.2.5. Ocena algorytmu

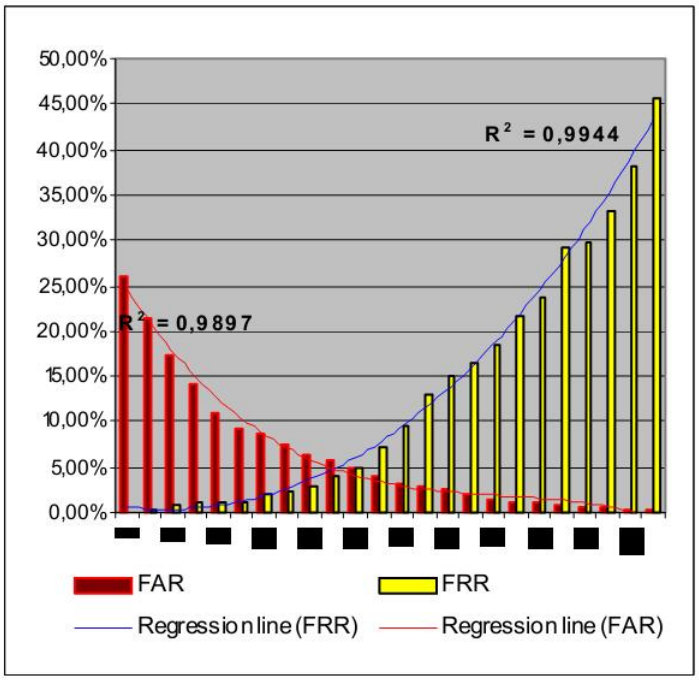
Określenie ilości błędów technologii biometrycznej jest naszym głównym celem. Próbowaliśmy znormalizować sposób oceny, mimo że nasze rezultaty są silnie zorientowane na wyniki pomiaru i silnie zależne od ilości użytkowników systemu i przede wszystkim od ich cech. Oznacza to, że nawet z dużą ilością danych, rezultaty mogą być różne jeśli zmienimy grupę badanych użytkowników. Trudność polega na określeniu reprezentatywnej grupy użytkowników, takiej która pokazywałaby nam przekrój populacji [2]. Dane były zbierane przez aplet javy, zainstalowanym na serwerze na laptopie. W warunkach rzeczywistych przewidujemy różne warunki wpisywania danych oraz różną kondycję użytkownika. W rzeczywistości każda próba logowania jest inna, zatem FAR i FRR powinny być wyliczane zgodnie z liczbą prób, a nie ilością użytkowników. W każdym razie, podczas logowania przez internet nie wiadomo czy obie próby pochodzą od jednego użytkownika czy od dwóch. Zebrane dane są wynikiem 170 391 prób, 143 to błędna akceptacja (FAR), 251 to błędne odrzucenie (FRR). W obu algorytmach, zaprezentowanym w artykule [2] i obecnym, została określona ilość błędów. Rezultaty pokazują, że można osiągnąć błąd akceptacji (FAR) bliski zeru, definiując wysoki próg akceptacji. Podobnie definiując niski próg akceptacji można osiągnąć błąd odrzucenia (FRR) bliski zeru. Porównując oba algorytmy punkt przecięcia (CER) został zmniejszony z 5,58% do mniej niż 5%. Możemy stwierdzić także różne wartości błędów (FRR i FAR) dla różnych progów akceptacji. Jak w przypadku innych algorytmów, balansując progiem akceptacji administrator może osiągnąć błąd akceptacji (FAR) bliski 0% lub błąd odrzucenia (FRR) bliski 0% lub może wybrać kompromis pomiędzy tymi wartościami, zgodnie z potrzebami.



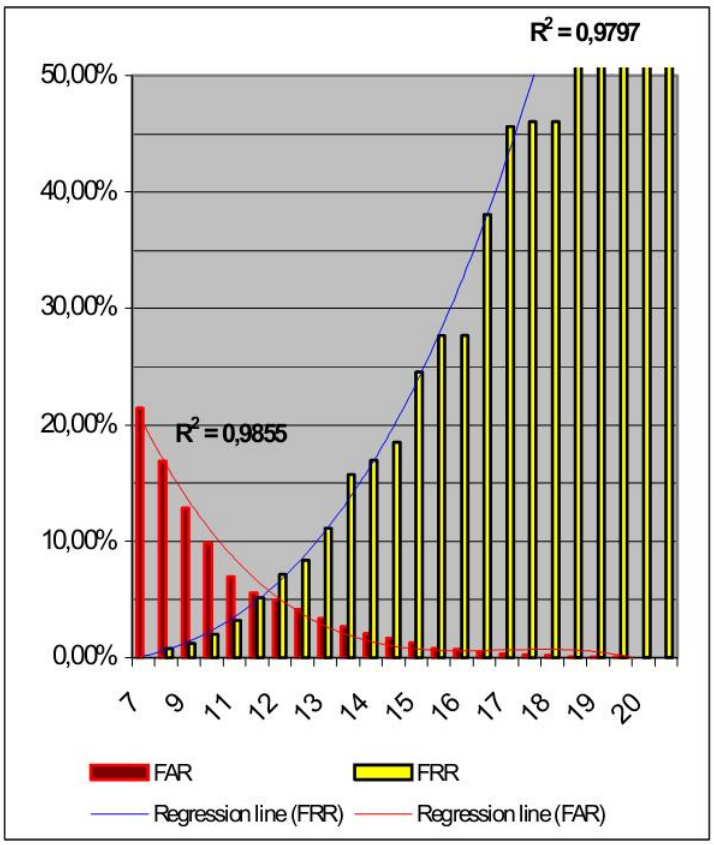
Rysunek 2.9: Błędy akceptacji przy różnych progach, osiągnięte za pomocą nowego algorytmu



Rysunek 2.10: Błędy odrzucenia przy różnych prograch osiągnięte za pomocą nowego algorytmu



Rysunek 2.11: Błąd przecięcia osiągnięty za pomocą nowego algorytmu



Rysunek 2.12: Błąd przecięcia osiągnięty za pomocą poprzedniego algorytmu

2.2.6. Wnioski

Osiągnięte rezultaty sugerują, że dynamika pisania może stać się efektywnym narzędziem zwiększającym bezpieczeństwo logowania. Nasz system, bazujący na dynamice pisania, nie jest uciążliwy dla użytkownika, jest bardzo opłacalny i przede wszystkim efektywny. Udało nam się osiągnąć niskie wskaźniki błędów (FAR/FRR - bliskie 0%) oraz punktu przecięcia błędów (CER - mniej niż 5%). Dodatkowo, rozpoczęliśmy badania nad ulepszeniem dynamiki pisania, np. podział klawiatury. Podział wprowadzi dodatkową warstwę bezpieczeństwa, ale ograniczy możliwość wprowadzanych haseł i loginów. Jeśli bezpieczeństwo jest ważniejsze dla organizacji - prowadzących strony internetowe - to jest to mała cena. Jeden udany atak potrafi sprawić, że firma zbankrutuje. Planujemy rozszerzyć nasz algorytm o metody oparte o długość hasła oraz szybkość pisania w celu zwiększenia bezpieczeństwa systemu. Ostatecznie proponujemy system, który automatycznie zmienia przechowywane wzorce w czasie, wraz ze zmianą sposobu pisania użytkownika.

2.3. Keystroke dynamics based authentication [11]

2.3.1. Przewidywanie zachowań ludzi

Na początku 20 wieku psychologowie i matematycy rozpoczęli eksperymentować z ludzkimi zachowaniami. Psychologowie dowiedli, że zachowania ludzi są przewidywalne obszarach powtarzających się czynności. W 1895 roku obserwacje telegrafistów pokazały, że każdy operator posiada wzorzec pisania. Ponadto operatorzy często rozpoznawali kto nadaje wiadomość, określając rytm kropek i kresek. Od wieków ludzie byli w stanie rozpoznać kto wchodzi do pomieszczenia po dźwięku kroków. Ogólnie rzecz biorąc każdy posiada unikalny sposób chodzenia.

Dzisiaj, mamy do czynienia z klawiaturami i myszkami, które zastępują telegraf. Utarło się, że charakterystyka pisania na klawiaturze jest bogata pod względem poznawczym. Każdy kto siedział obok osoby piszącej był w stanie rozpoznać ją po sposobie pisania na klawiaturze.

Ludzka natura dyktuje nam tempo pisania. Nikt nie siedzi przed komputerem i pisze bez przerwy z tą samą szybkością. Ludzie piszą przez chwilę, zatrzymują się w celu zebrania myśli, zatrzymuje się ponownie żeby odpocząć, kontynuuje pisanie i tak w kółko. W projektowaniu systemu należy wziąć pod uwagę, które klawisze są charakterystyczne dla danej osoby, a które nie. Psychologowie przebadali sposób komunikowania się ludzi z komputerem i zaproponowali kilka modeli opisujących zachowanie ludzi. Jeden z popularnych modeli jest model "keystroke-levelźaproponowany przez Carda. Jego model opisuje interakcje człowiek-maszyna podczas sesji terminalowej. Model "keystroke-levelśumuje sesje terminalowe według wzoru: $T_t = T_a + T_e$, gdzie T_t reprezentuje długość sesji terminalowej; T_a reprezentuje czas potrzebny do oceny zadań, zbudowania reprezentacji przygotowanych funkcji i wybraniu metod do rozwiązania problemu; T_e reprezentuje czas potrzebny do wykonaniu wszystkich funkcji składających się na zadanie.

Trzeba wziąć pod uwagę, że T_a zależy od zadania, doświadczenia użytkownika i zrozumienia przygotowanych funkcji. Wartość ta jest niemożliwa do kwalifikacji. T_a nie może określać cech osoby. Z drugiej strony, T_e opisuje mechaniczne akcje, które mogą być wyrażone za pomocą: $T_e = T_k + T_m$, gdzie T_k jest czasem zebrania informacji i T_m jest czasem potrzebnym do mentalnego przygotowania. Kiedy robimy coś w programie, użytkownik nie dzieli swojego czasu w ten sposób.

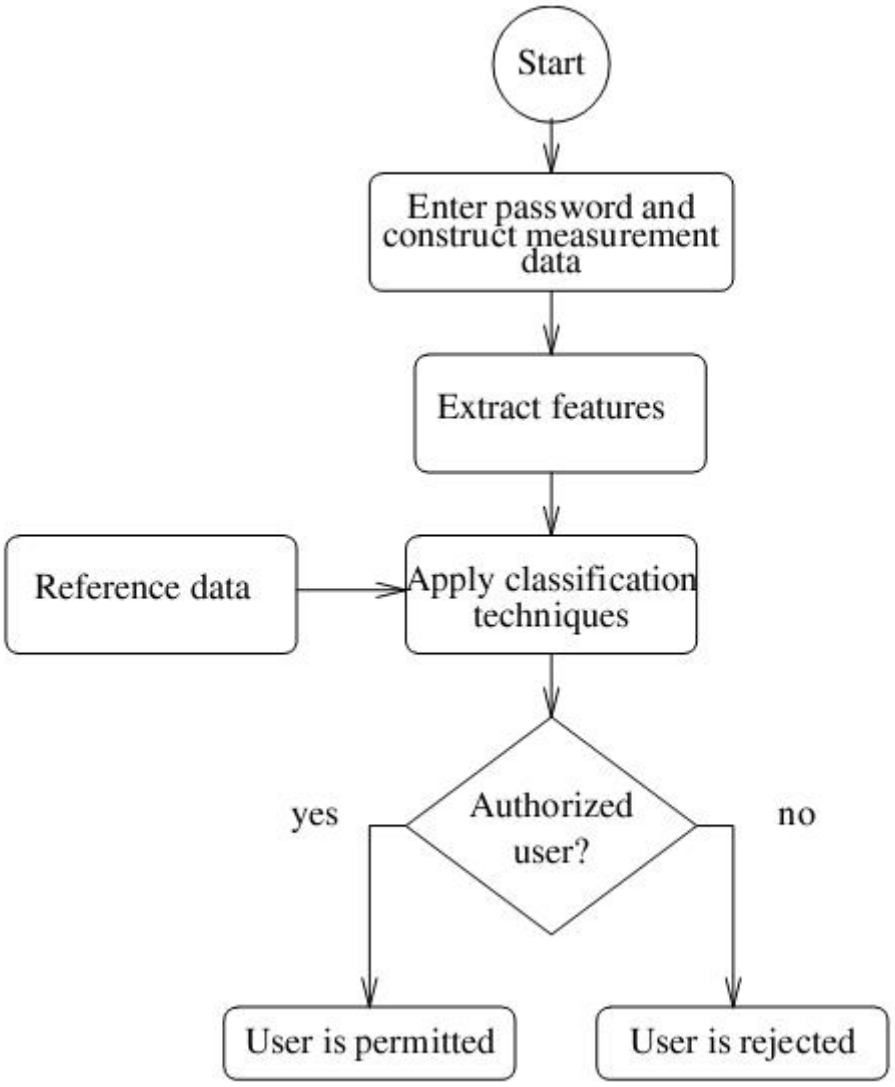
Shaffer pokazał, że kiedy osoba pisząca przepisuje dane, to mózg działa jak bufor, który przetwarza tekst na klawisze. Średnia wielkość bufora wynosi wtedy 6-8 znaków. Z powodu tych limitów, osoby piszące grupują symbole w mniejsze poznawcze jednostki i zatrzymują się pomiędzy tymi jednostkami. Cooper ustalił, że typowe przerwy są pomiędzy słowami tak często jak wewnątrz słów, które są dłuższe niż 6-8 znaków.

2.3.2. Aplikacje dynamiki pisania używające czasów pomiędzy wciśnięciami klawiszy

Pisanie odręczne jak i na klawiaturze są manualnymi umiejętnościami, oba posiadają mierzalne charakterystyki, które są unikalne dla osoby, która wykonuje zadanie. Umphress i Williams przeprowadzili eksperyment charakterystyk pisania. Użyli dwóch parametrów wejściowych do identyfikacji, profilu referencyjnego i profilu testowego. Każde wciśnięcie klawisza było pobierane z dokładnością do setnych części sekundy i zapisywane na dyskietce. Inny program był używany do analizy danych i tworzenia bazy danych dla każdego użytkownika. Trzeci program był używany do porównywania danych z profilu testowego z profilem referencyjnym. 17 osób brało udział w eksperymencie. Każda osoba wykonała dwa testy pisania. Testy te były wykonane w odstępie kilku dni. W pierwszym teście, chętni przepisali 1400 znaków prozy. W drugim teście, profilu testowym, przepisali 300 znaków prozy. Odkryto wtedy, że wysoki stopień korelacji może być osiągnięty jeśli ta sama osoba stworzy referencyjny i testowy profil. Zostało wyznaczonych kilka poziomów pewności w przypadku gdy profile piszących różniły się.

Obaidat opisał metodę identyfikacji użytkowników bazującą na technice pisania. Zmierzone czasy między wciśnięciami klawiszy danego użytkownika znanej sekwencji znaków zostały poddane tradycyjnej analizie wzorca w celu zaklasyfikowania użytkownika uzyskały dobre rezultaty weryfikacji. Lepsze wyniki zostały osiągnięte przez dwukrotne wpisanie sekwencji znaków oraz użycie najmniejszych możliwych jednostek pomiarowych, niż w przypadku jednokrotnego wpisania tekstu. Klasyfikator najmniejszego dystansu osiągnął najlepsze wyniki klasyfikacji. W celu osiągnięcia lepszej klasyfikacji, analiza klasyfikatora doprowadziła do redukcji wymiaru oraz liczby klas w systemie identyfikacji. Wektor pomiarów jest otrzymywany przez pobieranie czasów pomiędzy wprowadzonymi znakami hasła. Poniższy rysunek pokazuje wykaz przepływu danych przez wszystkie kroki systemu rozpoznawania.

Obaidat i Macchiarolo użyli tradycyjnej sieci neuronowej wraz z klasyczną techniką rozpoznawania wzorców w celu klasyfikacji/identyfikacji użytkownika, użyli czasów pomiędzy wciśnięciami klawiszy. Brali pod uwagę 6 użytkowników. Próbka danych użyta do rozpoznania użytkownika to czasy pomiędzy poprawnie wprowadzonymi literami zebrane podczas pisania znanej sekwencji znaków. Ochotnicy eksperymentu byli proszeni o wprowadzenie tej samej frazy, która była niewidoczna podczas pisania;



Rysunek 2.13: Przepływ danych przez wszystkie kroki systemu rozpoznawania

ponadto, ważne jest wyświetlenie komunikatu na monitorze po wprowadzeniu jej. Fraza była wprowadzana ponownie podczas, gdy ochotnik wprowadził ją błędnie. Czas pomiędzy wciśnięciami klawiszy był zbierany przy użyciu bazodanowego systemu akwizycji, który używał języków programowania Fortran oraz asembler. Przy pomocy asemblera obsługiwane były przerwania klawiatury i zaopatrywano główny program w czasy pomiędzy wciśnięciami klawiszy. Na przykład, jeśli hasło ÓBAIDATźostało wprowadzone to program asemblerowy przetwarzał czasy pomiędzy poszczególnymi literami - parami (O, B), (B, A), (A, I), (I, D), (D, A) oraz (A, T). Ochotnikom dawany był dowolny czas na skończenie eksperymentu. To pomagało wyliczyć wartość średnią efektu nieskorelowanych źródeł dźwięku, które mogły być spowodowane przez instrumenty oraz ochotników. Fraza, składająca się z 30 komponentów wektorowych była użyta jako pierwsza; jak również, tylko 15 pierwszych komponentów wektorowych było użytych później do czasu nie wpływania na rezultaty. Dane były zbierane od 6 różnych użytkowników przez okres 6 tygodni. Całkowita liczba wektorów pomiarowych jednego użytkownika wynosiła 40. Surowe dane były organizowane według poniższych zasad:

- każdy wzorzec składający się z 15 wartości, które były czasami w milisekundach pomiędzy poprawnie wprowadzonymi wciśnięciami klawiszy znanej sekwencji znaków;
- było 40 próbek każdego użytkownika (klas)(600 wartości w klasie) i
- było 6 zdefiniowanych klas (3600 wartości w sumie).

Do celów treningu, surowe dane były dzielone na dwie części: wszystkie nieparzyście ponumerowane wzorce każdej klasy oraz wszystkie parzyście ponumerowane wzorce. W każdej symulacji tylko połowa danych była używana do sformułowania treningu. Po każdym treningu sieć była trenowana, całe ustawienia wzorca (24 szablony) były prezentowane do sieci w celu klasyfikacji.

Kilka wersji danych treningowych było stworzonych raczej do przeprowadzenie dochodzenia zdolności generalizacji sieci niż do zapamiętania ustawień treningowych. Różnice w ustawieniach szablonu treningowego to: (a) jeśli wzorce są z nieparzystej lub parzystej połówki surowych danych i (b) rozdrobnienie ustawień treningowych, które są definiowane przez ilość surowych szablonów uśredniona w celu stworzenia każdego wzorca treningowego. Na przykład, jeśli wszystkie surowe wzorce w klasie są uśrednione w celu sformułowania pojedynczego wzorca treningowego, wtedy rozdrobnienie jest małe. Z drugiej strony, jeśli nie zostało użyte uśrednienie tzn. wszystkie wzorce są użyte w treningu, wtedy rozdrobnienie jest duże. Intuicyjnie jeśli duże rozdrobnienie zostało użyte do treningu to powinniśmy osiągnąć lepsze rezultaty klasyfikacji. Tabela 1 pokazuje przykład danych użytych do treningu.

Podczas badania fraz, zostały stworzone różne kombinacje wzorców do testowania zdolności uczenia trzech różnych paradygmatów sieci neuronowej. Po eksperymentach została wybrana najlepsza architektura sieci neuronowej dla przyszłej aplikacji, sieć została wcielona do systemu online, tak że system mógł zbierać czasy pomiędzy wciskanymi klawiszami w czasie rzeczywistym i dokonywać od razu klasyfikacji. Symulatory zostały napisany w języku C. Niektóre krytyczne czasowo fragmenty zostały napisane w asemblerze. System bezpieczeństwa składał się z dalej wymienionych zadań.

2.3.3. Wprowadzanie danych

Funkcje czasowe używały timera 8253, który znajduje się we wszystkich komputerach klasy IBM-PC, do mierzenia czasów pomiędzy wciśnięciami klawiszy. W przypadku komputera z PC-AT była używana funkcja BIOS'u (BIOS microsecond timing) zamiast wyjść timera 8253, gdy były niedostępne. Podobne schematy mogą być użyte dla innych platform. Na wszystkich platformach, kalibracja procedur została wywołana przed pomiarami. Proces kalibracji w pierwszej kolejności ustawia, która metoda pobierania czasu będzie używana (dokonywane to jest na podstawie typu komputera) następnie kalibruje czas używając zegara systemowego (time-of-the day clock). Podczas aktualnego pobierania czasu klawiszy, procedura pobiera każdy klawisz, zapamiętuje go i potem rozpoczyna pomiar dalej czekając na kolejny klawisz. Kiedy pojawi się następny klawisz, zostaje zapamiętany czas pomiędzy wciśnięciami i

wciśnięty klawisz. Proces ten jest powtarzany i druga próbka pomiarów zostaje zapamiętana. To pokazuje, że pobieranie jak najmniejszych wartości każdej przerwy, bazując na dwóch wartościach, poprawia trafność klasyfikacji.

2.3.4. Uczenie

Żeby nauczyć sieć neuronową potrzebny jest zbiór wektorów pomiarowych dla każdej klasy użytkowników. Wektory te są zbierane i zapisywane dla każdego użytkownika. Kiedy zostanie zebrana wymagana ilość wektorów, mogę one zostać uśrednione i znormalizowane, w celu stworzenia wzorca, który zostanie użyty do uczenia sieci. Liczba wektorów wzorców jest zdefiniowana przez użytkownika programu. Użytkownik może opisać konfigurację sieci w programie, pamięć jest alokowana dla jednostek przetwarzających (neuronów), wzorzec treningowy oraz wagi wektora są zapamiętywane. Uczenie składa się z zaaplikowania wektora wzorców na wejście, porównanie uzyskanego wyjścia z przewidzianym wyjście oraz skorygowaniu wag według algorytmu uczącego. Kiedy błąd wektora uczącego zostanie zredukowany do zdefiniowanej wcześniej wartości, która jest całkowitą sumą kwadratów, błąd mniejszy lub równy 0.01 w naszej pracy, uczenie jest przerywane i cała sieć jest zapisywana na dysku.

2.3.5. Klasyfikacja

W celu uruchomienia programu jako klasyfikator/identyfikator online, sieć jest przywoływana z pliku zapisanego po treningu. Alokowana jest potrzebna pamięć oraz wczytywane są wagi wektorów. Użytkownik jest powiadamiany, że może wprowadzić kluczową frazę. Przerwy między wprowadzanymi klawiszami są pobierane, normalizowane przy użyciu wybranej przez użytkownika funkcji normalizującej (procentowa wartość największej wartości, wektor jednostkowy lub bez normalizacji) i podawane na wejście sieci neuronowej. Wartości wejściowe są propagowane w sieci, która posiada tyle wyjść ile jest zdefiniowanych klas użytkowników. Jednostka wyjściowa, która jest aktywna (powyżej granicy zdefiniowanej przez użytkownika) przedstawia klasyfikację wektora wejściowego pomiarów.

2.3.6. Normalizacja, Wykonanie, Włączanie

W teście, sześciu użytkowników pisało 15-znakową frazę 20 razy, każdy przez okres 6 tygodni. Do stworzenia grupy szablonów zostały użyte surowe dane w celu uczenia sieci. Dwa typy normalizacji danych to: jednostkowa długość wektora i część elementu największego. Normalizacja jednostkową długością wektora jest obliczana przez podzielenie każdego elementu z wektora pomiarowego przez całkowitą długość wektora (pierwiastek z sumy kwadratów wszystkich wartości). Zabieg ten jest niewystarczający dla różnych użytkowników, którzy wygenerują wektory o podobnych parametrach i wtedy sieć może nie zauważyć różnicy pomiędzy nimi podczas uczenia. Przy podzieleniu każdego elementu przez największą wartość uzyskujemy wartości z zakresu 0 do 1. Wartości takie są wygodne jako wejścia dla sieci neuronowej, zachowując relatywne różnice w elementach. W celu stworzenia wzorca uczącego, uśredniane są dwa znormalizowane wektory.

Czas uczenia sieci jest może być uzależniony od dopasowanego poziomu uczenia oraz wartości parametrów. Szybkość uczenia jest częścią błędów, która jest używana do wyliczenia ostatecznych wag. Wartość parametrów jest częścią poprzedniego dopasowania, która jest dodawana do aktualnego dopasowania. Po skończeniu uczenia, każdy użytkownik testuje sieć. Całkowita dokładność była 97.8

System może być łatwo wcielony do komputerowego systemu bezpieczeństwa. Początkowo, każdy użytkownik, który posiada dostęp, może zostać poproszony o wprowadzenie przykładowej próbki znanej frazy. Próbki te są zbierane w czasie używania danych wejściowych modułu i zachowywane przez administratora. Administrator generuje zbiór uczący i konfiguruje sieć przy użyciu modułu uczącego. Sieć posiada ilość wejść równą ilości pomiarów w każdym wektorze, liczbę ukrytych jednostek i liczbę wyjść równą ilości użytkowników. Wartości wag są określone podczas uczenia, które może być przeprowadzone offline lub jako proces w tle. Po treningu, wagi są przechowywane i mogą być szybko przywołane do klasyfikacji w trybie on-line. W razie potrzeby usunięcia użytkownika lub dodaniu do listy

autoryzowanych, należy wygenerować ponownie dane uczące, jak również, moduł uczący może automatycznie wygenerować zbiór uczący na podstawie istniejącego lub na podstawie nowej próbki danych. Dodając użytkownika może być wymagane dodanie nowej jednostki wyjściowej do sieci oraz dodanie wag określanych podczas uczenia.

W praktyce użytkownik może zidentyfikować siebie przez użycie numeru przypisanego do jego próbki klasyfikującej. Następnie może zostać poproszony o wpisanie kluczowej frazy. Jego czas pomiędzy wciśniętymi klawiszami może być zebrane oraz zaklasyfikowane. Jeśli numer użytkownika zgadza się z przypisaną klasą, wtedy użytkownikowi zostają przyznane prawa dostępu. Jeśli klasyfikacje nie powiedzie się, kilka rzeczy może się stać:

1. Zostanie odmówiony dostęp użytkownika do danych. Jest to najwyższy poziom bezpieczeństwa.
2. Dostęp do danych zostanie przyznany po wprowadzeniu hasła wysokopoziomowego.
3. Dostęp do danych zostanie przyznany z ograniczeniami.
4. Dostęp do danych zostanie przyznany, ale zostanie wysłane powiadomienie do administratora i akcje użytkownika będą zebrane do późniejszej analizy. To jest najniższy poziom bezpieczeństwa.

Istnieją plusy i minusy, nad którymi należy się zastanowić, w przypadku takich systemów; ryzyko naruszenia równowagi pomiędzy bezpieczeństwem, a wygoda użytkowania.

Bleha i Obaidat eksperymentowali z algorytmami perceptonowymi jako klasyfikatorami do weryfikacji użytkowników. Podczas wykonywania w czasie rzeczywistym pomiarów czasów pomiędzy wprowadzonymi klawiszami podczas wprowadzania hasła użytkownika, dane zostały zebrane z 10 poprawnych użytkowników i 14 niepoprawnych użytkowników przez okres 8 tygodni. Hasłem było imię użytkownika. Funkcje decyzyjne zostały zróżnicowane używając połowy z danych (danych treningowych) do wyliczenia wektorów wagowych. Funkcje decyzyjne zostały użyte do pozostałej połowy danych (danych testowych) w celu weryfikacji użytkownika. Otrzymano następujące wyniki: błąd odrzucenia poprawnych użytkowników na poziomie 9%, błąd akceptacji niepoprawnych użytkowników na poziomie 8%. Algorytm perceptrony został uznany za odporny pod względem wyboru początkowego wektora wag.

Obaidat ocenił wykonanie 5 algorytmów rozpoznawania wzorców wykorzystanych do identyfikacji użytkowników komputera używając czasów pomiędzy poprawnie wprowadzonymi przyciskami wygenerowanych podczas pisania znanej sekwencji znaków. Są to algorytmy: funkcja potencjału, klasyfikator Bayes'a, minimalna odległość i pomiar kosinusa. 100% zgodność została osiągnięta kiedy użyto funkcji potencjału. Najsłabszą funkcją okazał się pomiar kosinusa. Obaidat i Sadoun porównali wykonanie nowo wymyślonych schematów sieci neuronowych, zwanych Hybrid-Sum-Of-Products (HSOP) do weryfikacji użytkowników i innych problemów klasyfikacji. Porównali wykonanie HSOP z Sum-Of-Products i z siecią neuronową z propagacją wstecz. Odkryli oni, że HSOP działa lepiej niż dwa pozostałe paradygmaty. W swojej pracy użyli czasów pomiędzy wciśnięciami klawiszy podczas pisania znanej frazy.

2.3.7. Aplikacja dynamiki pisania używająca czasów wciśnięcia jako podstawa analizy

Obaidat i Sadoun weryfikowali użytkowników komputera używając czasów wciśnięć klawiszy jako podstawę do autentykacji. Ochotnicy eksperymentu zostali poproszeni o wprowadzanie loginu przez okres ośmiu tygodni. Program zbierał czas wciśnięcia i czas zwolnienia klawisza na komputerze klasy PC z dokładnością do 0.1 ms. Program został zaimplementowany jako program rezydentny w DOSie. Standardowy wskaźnik do przerwań klawiatury został zastąpiony takim, który był w stanie wykryć przychodzące kody skanów klawiatury i zapisać jest wraz z czasem przyjścia. Program mierzył czasu pomiędzy wciśnięciem i zwolnieniem klawisza. Taka procedura była wykorzystywana dla każdej litery i dla każdego ochotnika. Kod skanu jest generowany dla wciśnięcia i zwolnienia klawisza.

Procedura logowania została zmodyfikowana w taki sposób, że podczas tworzenia każdej próbki logowania był tworzony wektor czasów wciśnięć do analizy. Taka procedura zwiększała wymiarowość próbki z N znakowej do (2N-1). Taka duża wymiarowość może zapewnić lepszy podział nawet jeśli liczba znaków jest niewielka. Wyniki monitorowania logowania były zbierane dla 15 użytkowników, którym był przyznany dowolny okres czasu prowadzenia eksperymentu. Takie podejście wyeliminowało efekt zmęczenia oraz stresu jak również nieskorelowane źródła dźwięku. Fałszywe próbki 15 ID były zbierane dla każdego z 15 niepoprawnie zalogowanych użytkowników, którzy popełnili błąd 15

razy. Wszystkie fałszywe próby były zbierane w jednej sesji dla każdego błędnego użytkownika. Ochotnicy używali interaktywnie systemu i ich wyniki były zapisywane. Czasy pomiędzy wciśnięciami były zbierane przy użyciu przerwań. Numer identyfikacji użytkownika miał 7 znaków. Zebrane dane zostały podzielone na dwie części: część ucząca i część testowa. Do klasyfikacji procesu został użyty Wzorzec rozpoznawania i sieć neuronowa. Odkryto, że czasy wciśnięć są bardziej efektywne niż czasy pomiędzy wciśnięciami klawiszy oraz najlepszy wynik identyfikacji osiągnięto używając obu metod pomiaru. Trafność identyfikacji 100% (zero błędnych akceptacji i zero błędnych odrzuceń) została osiągnięta kiedy zarówno czasy wciśnięć jak i czasy pomiędzy wciśnięciami były rozważane przy użyciu metod fuzzy ARTMAP, sieć o radialnych funkcjach podstawowych (RBFN) i sieci neuronowej LVQ. Inne sieci neuronowe i klasyczne algorytmy rozpoznawania wzorców takie jak wsteczna propagacja z sigmoidalną funkcją wyjścia (PR, Sig), hybrid-sum-of-products (HSOP), sum-of-products (SOP), funkcja potencjału i decyzyjne reguły Bayesa także dały dobre rezultaty.

Sukces podejścia był mierzony głównie przy użyciu ilości błędnych odrzuceń (błąd typu I) i ilości błędnych akceptacji (błąd typu II), koszt systemu rozpoznawania i czasu potrzebnego do weryfikacji. Za dwa najważniejsze pomiary są uważane ilości błędów typu I i II. Ilość błędnych odrzuceń (błędów typu I) weryfikacji wskazuje jak często osoba autoryzowana zostanie błędnie rozpoznana. Błędy typu II określają jak często nieautoryzowana osoba zostanie błędnie rozpoznana i dopuszczona do systemu. Jest to najlepiej oceniająca wartość mechanizmu. Spowodowane jest to faktem opisywania stopnia, w jakim osoba nieporządana może dostać się do systemu. Błąd typu I jest ważny gdyż opisuje stopień frustracji użytkownika. Wyniki naszych badań pokazały, że najlepszą techniką rozpoznawania wzorca jest funkcja potencjalna wspomagana regułami Bayesa. Najgorszy okazał się algorytm pomiaru kosinusa. Czasy wciśnięć dawały lepszą zgodność niż czasy pomiędzy wciśnięciami klawiszy. Przy używaniu paradygmatów sieci neuronowej podczas procesu klasyfikacji odkryto, że czasy wciśnięć są podstawą dla czasów między wciśnięciami. Ponadto rozwiązanie będące kombinacją czasów wciśnięć i czasów między wciśnięciami zmniejsza błędną klasyfikację. Najlepszymi sieciami neuronowymi okazały się: LVQ, RBFN i Fuzzy ARTMAP. Uzyskały one zerową liczbę błędnych klasyfikacji (błędów klasy I i II).

Średnia długość łańcucha znaków wynosiła 7. W naszej poprzedniej pracy, uzyskaliśmy mniejszą trafność klasyfikacji z hasłami 15 znakowymi. We wszystkich eksperymentach stwierdzaliśmy przewagę czasów wciśnięć nad czasami pomiędzy wciśnięciami klawiszy. Takie rezultaty sugerują, że czasy wciśnięć mogą prowadzić do lepszej charakteryzacji zdolności pisania niż czasy pomiędzy wciśnięciami klawiszy. Odkryliśmy także, że lepsze efekty sieci neuronowej prowadzą do lepszej trafności autentykacji/weryfikacji niż najlepsze klasyczne schematy rozpoznawania wzorców.

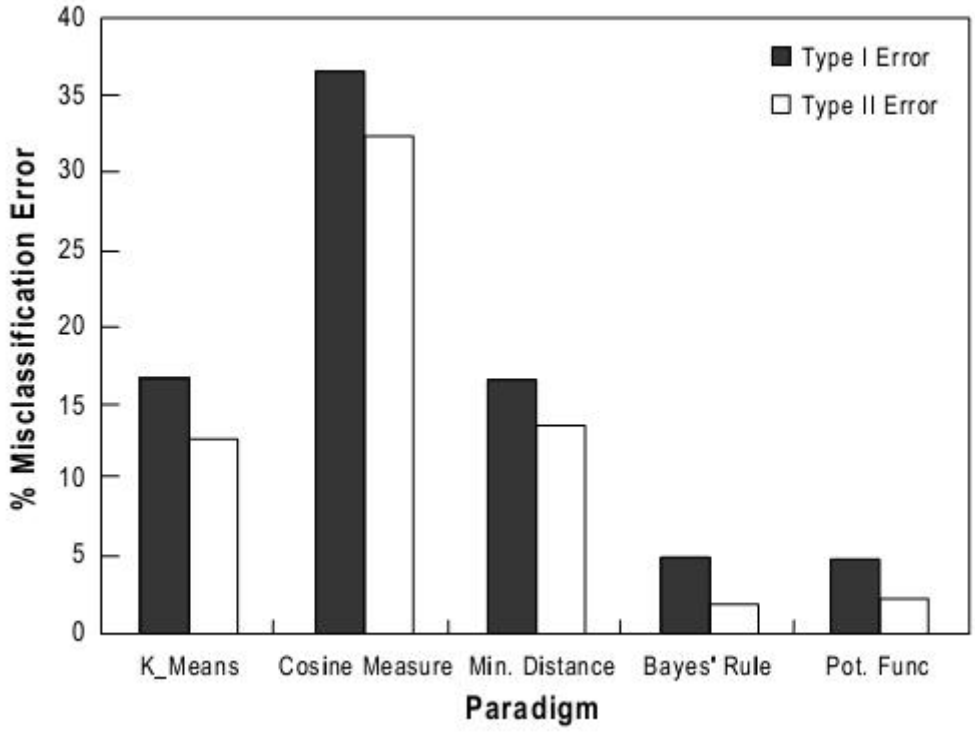
Ostatnia podobna praca prowadzona przez Robinsona potwierdza powyższe stwierdzenia. W pracy tej autorzy używali czasów wciśnięć do określenia stylu pisania. Wykorzystali także tradycyjne schematy rozpoznawania wzorców do procedury klasyfikacji. Użyli czasów wciśnięć i czasów pomiędzy wciśnięciami. Najlepsze wyniki osiągnęli używając indukcyjnej metody uczenia.

2.3.8. Wnioski

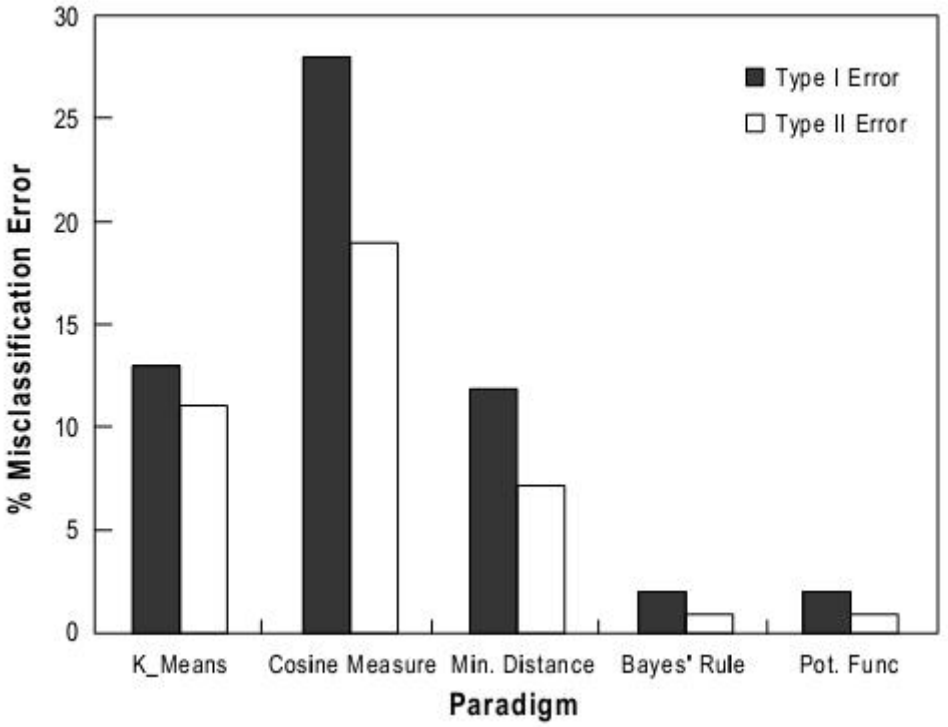
Podsumowując, dynamika pisania jest bogata pod względem sposobów i cech oraz może być użyta do autentykacji/weryfikacji dostępu do systemu komputerowego lub zasobu sieciowego. Dynamika pisania dla logowania użytkownika komputera prowadzi do określenia wzorców, które mogą być użyte do weryfikacji osoby piszącej. Dynamika pisania razem z innymi metodami bezpieczeństwa może prowadzić do silnego i efektywnego systemu autentykacji i weryfikacji. Ani nasza praca ani inne nie poradziły sobie z podstawowymi błędami. Wymagane są dalsze badania w celu stworzenia metod, na których można będzie polegać, które nie będą irytujące dla użytkownika oraz szeroko akceptowalne przez komputery oraz systemy komunikacji internetowej. Ostatecznie, zostało stwierdzone, że paradygmaty sztucznej sieci neuronowej są efektywniejsze niż klasyczne algorytmy rozpoznawania wzorców.

Training Set			
File Name	*Patterns per Class*	*Raw Patterns per Average Pattern*	*Odd/Even Half*
all.pat	40	1	All
Odd20	1	20	Odd
Even20	1	20	Even
Odd5	4	5	Odd
Even5	4	5	Even
Odd2	10	2	Odd
Even2	10	2	Even

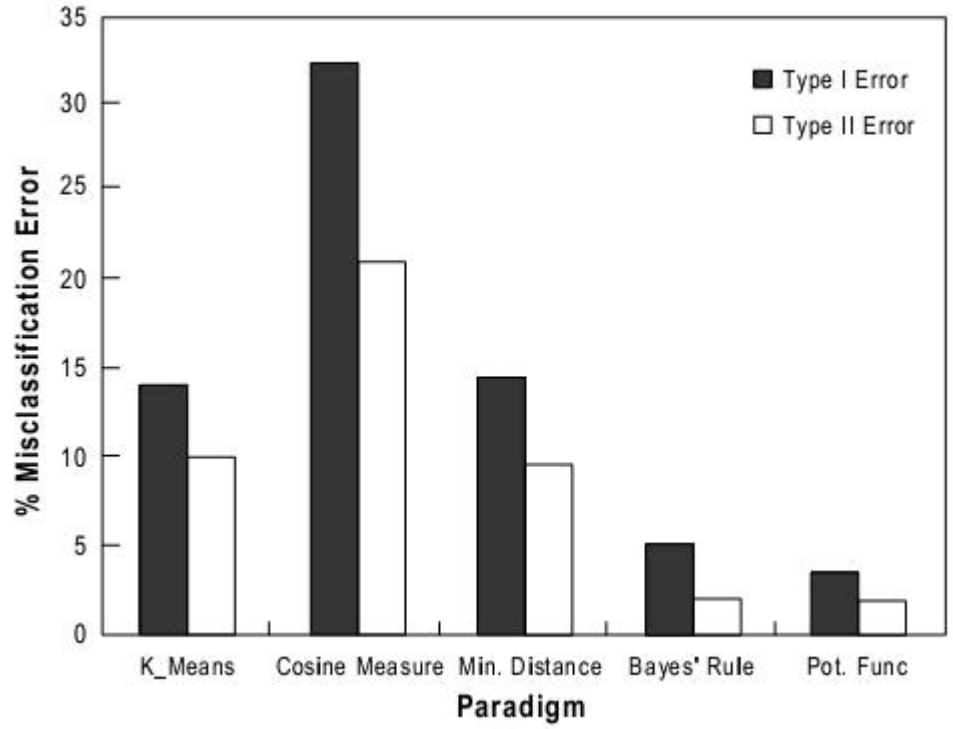
Rysunek 2.14: Przykład użytych danych uczących



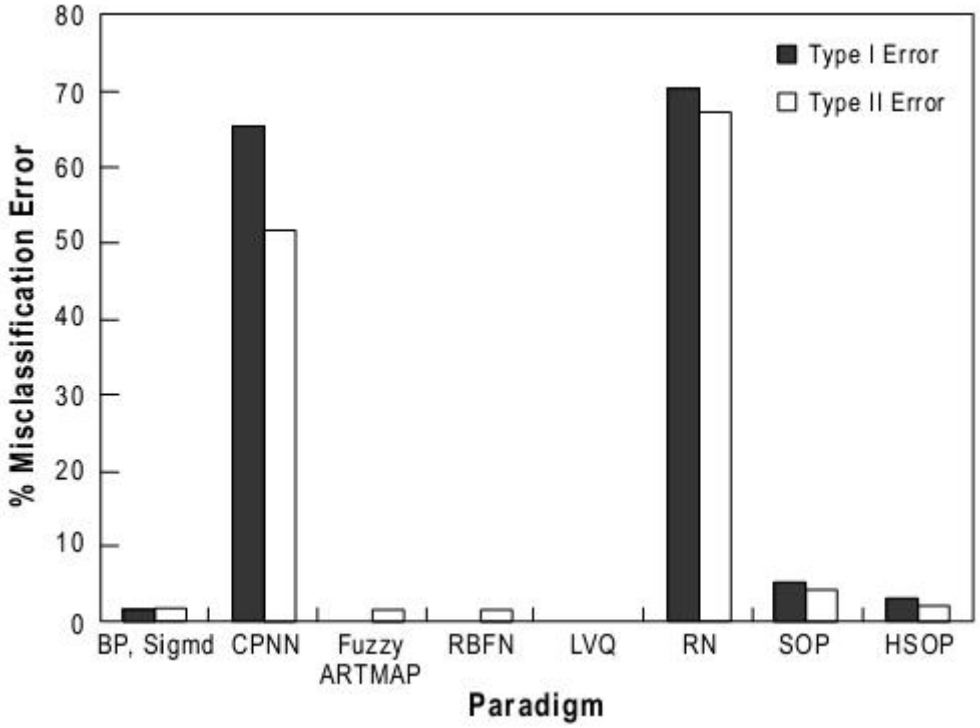
Rysunek 2.15: Wyniki rozpoznawania przy użyciu czasów pomiędzy wciśnięciami i klasycznych metod rozpoznawania



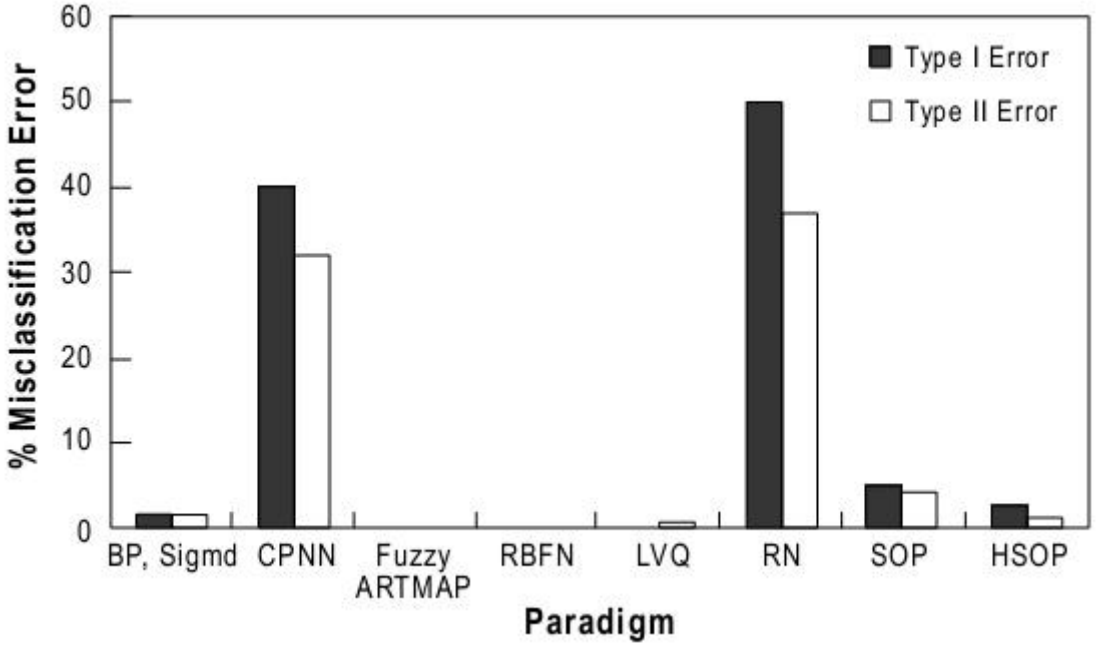
Rysunek 2.16: Wyniki rozpoznawania przy użyciu czasów wciśnięć oraz klasycznych metod rozpoznawania



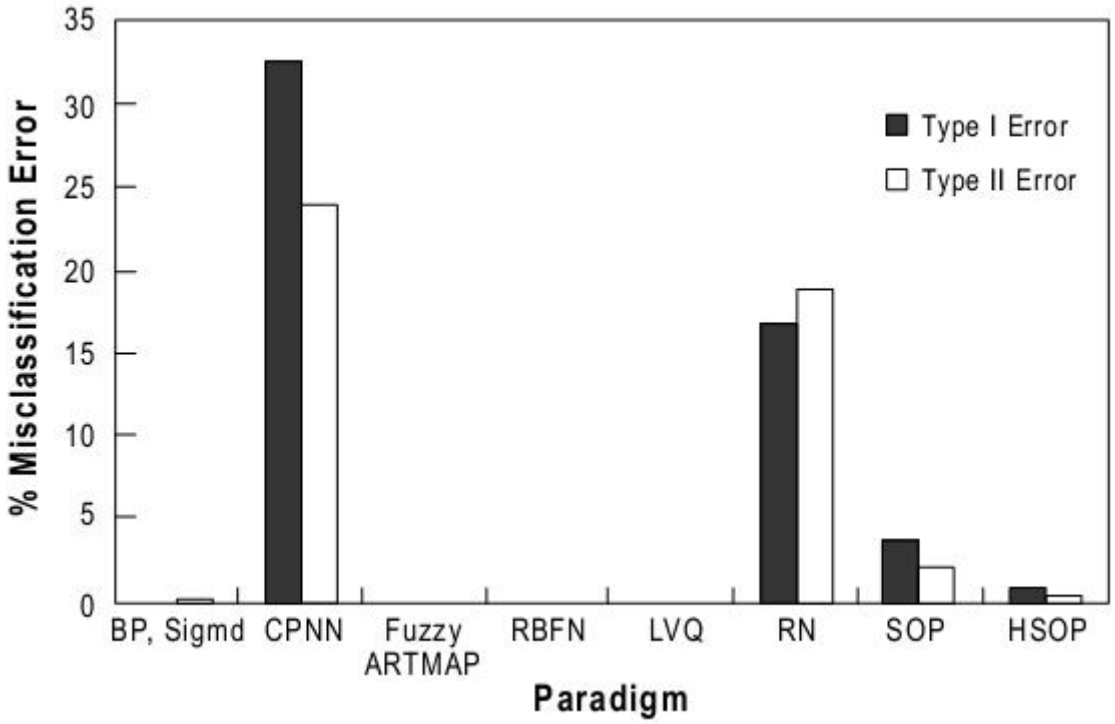
Rysunek 2.17: Wyniki rozpoznawania przy użyciu czasów pomiędzy wciśnięciami oraz czasów wciśnięć oraz klasycznych metod rozpoznawania



Rysunek 2.18: Wyniki rozpoznawania przy użyciu czasów pomiędzy wciśnięciami oraz sieci neuronowej



Rysunek 2.19: Wyniki rozpoznawania przy użyciu czasów wciśnięć oraz sieci neuronowej



Rysunek 2.20: Wyniki rozpoznawania przy użyciu czasów pomiędzy wciśnięciami i czasów wciśnięć oraz sieci neuronowej

3. Podstawy teoretyczne

Rozdział ten zawiera podstawowe informację o sztucznych sieciach neuronowych. Przedstawia budowę biologicznych struktur, na których są one wzorowane. Pokazuje zastosowania, wady oraz zalety sztucznych sieci neuronowych. W szczególności zawiera informacje o sieciach typu RBFN - Radial Basis Function Network.

3.1. Rys historyczny

Pracę opublikowaną przez Warrena McCullocha i Waltera Pittsa w roku 1943 można uznać za przyczynek rozwoju sieci neuronowych. Przedstawili oni matematyczny opis modelu komórki nerwowej oraz powiązania go z problemami przetwarzania danych. Kilka lat później Donald Hebb zaproponował metodę uczenia sztucznych sieci neuronowych, polegającą na zmianie wag połączeń między neuronami, zwaną metodą Hebba.

Lata 50 to rozwój nie tylko sieci neuronowych, ale ogólnie sztucznej inteligencji. Powstają wtedy pierwsze laboratoria AI na Uniwersytecie Carnegie Mellon (założone przez Allena Newella i Herberta Simon) oraz w Massachusetts Institute of Technology (założone przez Johna McCarthy'ego). W roku 1957 powstaje pierwszy perceptron za sprawą Franka Rosenblatta i Charlesa Wightmana.

Kolejny krok to stworzenie sieci Madaline (ang. Many Adaline), składającej się z elementów Adaline (ang. Adaptive linear element). Autorami tej sieci są Bernard Wildrow i Marcian Hoff. Od perceptronu Rosenblatta neuron typu Adaline różnił się funkcją aktywacji. Użyto funkcję liniową zamiast progowej. Do uczenia tej sieci zastosowali nową autorską metodę LMS (ang. Least Mean Square), czyli metoda najmniejszych kwadratów, znana pod nazwą reguła delty.

Lata 70 oznaczały zahamowanie rozwoju sieci neuronowych, głównie za sprawą wąskiego zakresu zastosowań sieci z liniową funkcja wyjścia. Nowe możliwości przyniosły nieliniowe sieci wielowarstwowe, z garścią metod uczących. W tym okresie powstała sieć Avalanche na uniwersytecie w Bostonie za sprawą Stephena Grossberga oraz sieć Cerebellatron na MIT skonstruowana przez Davida Mara, Jamesa Albusa oraz Andresa Pollioneze. Obie służyły od sterowania ramieniem robota. W 1977 James Anderson z Uniwesytetu Browna zbudował sieć Brain State in a Box. Był prekursorem w budowie pamięci asocjacyjnej.

Kolejny przełom to rok 1982, w którym to John Hopfield opisał sieć ze sprzężeniem zwrotnym. Zapoczątkował klasę sieci zwaną sieciami Hopfielda. Stworzył również teorię pamięci asocjacyjnej - podstawę działania sieci rekurencyjnych bazujących na mechanizmach statystycznych.

Rok 1986, za sprawą Rumelharta, Hintona i Williamsa, przyniósł nam metodę uczenia zwaną metodą wstecznej propagacji błędów. Od tego roku wzrasta liczba zastosowań sztucznych sieci neuronowych.

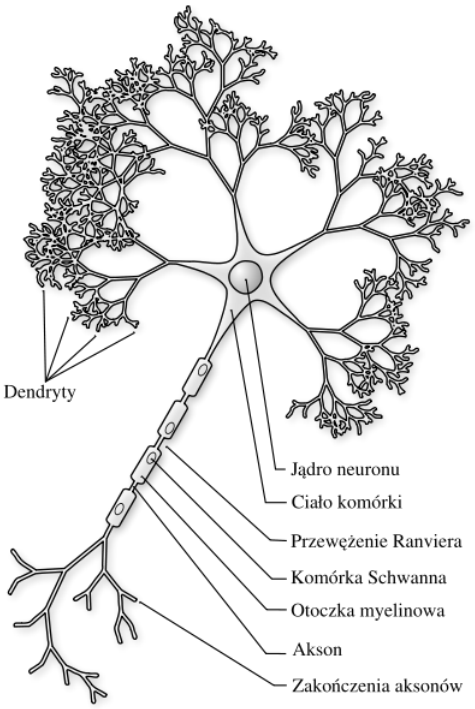
W Polsce sieciami neuronowymi zajmują się Gawroński (1970), Kulikowski (1972), Brodziak (1974), Tadeusiewicz (1992), Korbicz, Obuchowicz, Uciński (1994), Kacprzak, Ślota (1995).

Rys historyczny stworzony na podstawie z pracy magisterskiej [10]

3.2. Pierwowzór z natury

Wzorem sztucznych sieci neuronowych jest mózg ludzki. Masa mózgu ludzkiego wynosi u mężczyzny około 1,375 kg, u kobiety 1,225 kg, podczas gdy masa mózgu niemowlęcia to zwykle około 0,35 kg.

Mózg składa się z komórek nerwowych (neuronów). Komórki nerwowe przyjmują impulsy o częstotliwości 1-100 Hz oraz wysyłają impulsy o napięciu około 100mV przez czas 1-2 ms. Impulsy rozchodzą się z szybkością 1-100 m/s. Szybkość pracy mózgu szacuje się na około 10^18 operacji na sekundę. Neurony składają się z ciała komórki, jądra komórkowego oraz neurytów: dendrytów i aksonu, za pomocą których połączone są z innymi neuronami. Połączenie między komórkami nerwowymi zwane jest synapsą.



Rysunek 3.1: Neuron

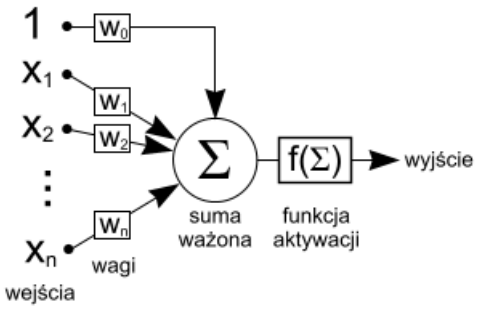
Neuron przetwarza sygnały wejściowe, dostarczane przez dendryty i wytwarza impuls wyjściowy, za pomocą aksonu. Oczywiście przetwarzanie jest nieliniowe, nie tak jak w uproszczonych neuronach. Sygnał wyjściowy emitowany jest po osiągnięciu określonego progu.

Sztuczne sieci neuronowe są daleko za tymi biologicznymi. Są jednak na tyle zbadane, że służą jako potężne narzędzie do rozwiązywania problemów inżynierskich.

Informacje zaczerpnięte z [7].

3.3. Sieci neuronowe RBFN

Sieci neuronowe typu RBF są szczególnym przypadkiem sieci neuronowych. Zaliczane są też do struktur neuropodobnych. W ich skład wchodzą klasyczne elementy sieci, tj. sztuczne neurony. Rysunek poniżej przedstawia sztuczny neuron McCullocha-Pittsa zaczerpnięty z [8].

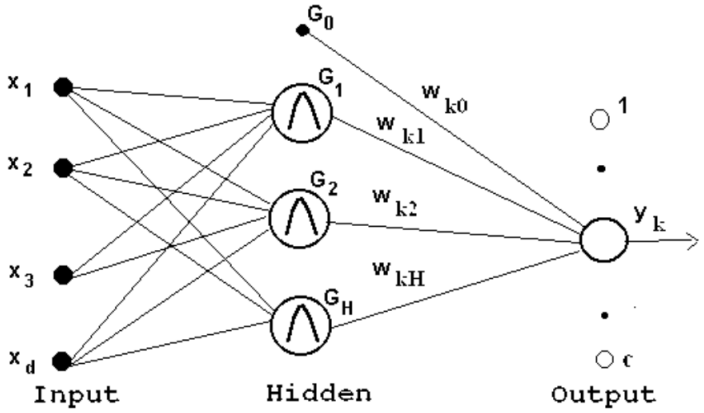


Rysunek 3.2: Sztuczny neuron. Model McCullocha-Pittsa.

Sieci RBF mają budowę typu:

d wejść -> U warstw ukrytych -> k wyjść

Sieci typu RBF często nie posiadają warstwy wejściowej, posiadają warstwę ukrytą oraz warstwę wyjściową. Informację są propagowane do przodu. Nie ma zapętleń ani wymiany informacji między neuronami tej samej warstwy. Przykład sieci typu RBFN przedstawiony jest na rysunku poniżej.



Rysunek 3.3: Przykład sieci typu RBFN

Warstwa wejściowa. Do warstwy wejściowej dane podawane są w postaci macierzy o wymiarach $N \times d$: $X_{N \times d} = [x^1, ..., x^n, ..., x^N]^T$. Kolejne wiersze tablicy interpretowane są jako punkty $x^n = [{x_1}^n, ..., {x_d}^n]^T \in R$.

Warstwa ukryta zawiera U neuronów. Każdy neuron posiada centrum $c_h \in R^d$. Wartość centrum inicjowana jest podczas tworzenia sieci, a później dostosowywana podczas uczenia.

Warstwa wyjściowa. W warstwie wyjściowej każdy z k neuronów jest klasycznym neuronem, z wektorem wag $w_k = [w_{k1}, ..., w_{kU}]^T$ przypisanych do niego oraz z funkcją wyjścia.

3.4. Zastosowania

Mimo,iż sieci neuronowe są narzędziem stosunkowo nowym, to mają one liczne i różnorodne zastosowania. Jako najważniejsze z nich można wymienić:

- Predykcja. Jednym z zastosowań sieci neuronowych jest przewidywanie danych wyjściowych na podstawie pewnych danych wejściowych. Ważną zaletą jest to, że sieć może nauczyć się przewidywania sygnałów wyjściowych bez jawnego definiowania związku między danymi wejściowymi, a wyjściowymi. Przykładami tego typu zastosowań mogą być: prognozy ekonomiczne rozwoju przedsiębiorstw, prognozy zmian rynku, automatyczne sterowanie urządzeń i przewidywania ich zachowania w odpowiedzi na bodźce sterujące.

- Klasyfikacja i rozpoznawanie. Zadanie polega na przypisaniu wzorca wejściowego do jednej z wielu zapamiętanych (wcześniej nauczonych) klas. Przykładem może być wykorzystanie do rozpoznawania pisma, mowy, stawianie diagnozy na podstawie kardiogramu, rozpoznawanie zakażonych komórek, a także przewidywanie na podstawie danych bilingowych sytuacji ekonomicznej przedsiębiorstwa.

- Kojarzenie danych. Sieci neuronowe, dzięki zdolności uczenia się, adaptacji i uogólniania doświadczeń, pozwalają zautomatyzować procesy wnioskowania i pomagają wykrywać istotne powiązania pomiędzy danymi.

- Analiza danych. Zadanie polega na znalezieniu związków pomiędzy danymi. Sieć znajduje podobieństwa pomiędzy danymi wejściowymi i klasyfikuje odpowiednie dane w jednej klasie. Realizacja tego zadania przez sieci neuronowe daje nowe możliwości, np. w zakresie prowadzenia analiz ekonomicznych i pozwala na wykrycie rzeczywistych intencji firm. Analiza danych może również pomóc w ustaleniu przyczyn niepowodzeń.

- Filtracja sygnałów. Technika ta polega na użyciu sieci neuronowych do analizy danych, które są zaszumione, celem usunięcia z nich zakłóceń o charakterze losowym. Własność ta znajduje zastosowanie w technice, a także ekonomii.

- Optymalizacja. Celem optymalizacji jest znalezienie rozwiązania spełniającego narzucone warunki, w szczególności maksymalizujące lub minimalizujące daną funkcję. Dzięki użyciu sieci neuronowych można uzyskać przybliżone rozwiązania zagadnień trudnych obliczeniowo. Typowym przykładem jest problem komiwojażera.

- Pamięć asocjacyjna. Dzięki wykorzystaniu sieci neuronowych można stworzyć pamięć, której zawartość jest dostępna na podstawie częściowej informacji o przechowywanym obiekcie lub w przypadku, gdy ta informacja jest zakłócona. Wyjątkowe własności takich sieci wykorzystuje się m.in. przy budowaniu multimedialnych baz danych (umożliwiających na przykład znalezienie utworu muzycznego w bazie na podstawie jego fragmentu).

Zastosowania zaczerpnięto z pracy magisterskiej [10].

3.5. Wady i zalety

Sztuczne sieci neuronowe są skutecznym narzędziem do rozwiązywania wielu złożonych problemów m. in. problemu komiwojażera. Radzą sobie z problemami, dla których klasyczne algorytmy zawodzą. W trakcie rozwiązywania zadania rola programisty to stworzenie sieci neuronowej i nauczenie jej. Problem stworzenia algorytmu jest zastępowany problemem stworzenia sieci neuronowej i wyboru algorytmu uczącego. Zapewnia to większą elastyczność. Kolejną zaletą jest odporność na szumy oraz błędy pojedynczych neuronów.

Sztuczne sieci neuronowe posiadają także wady. Przede wszystkim dają wyniki przybliżone. W przypadku większych sieci trudne jest określenie wpływu poszczególnych czynników wejścia na wynik. Spory problem stanowi także powolność metod uczących mimo sporej ilości udoskonaleń. Znaczna ilość rodzajów sieci powoduje problem podczas wyboru tej właściwej do danego zastosowania.

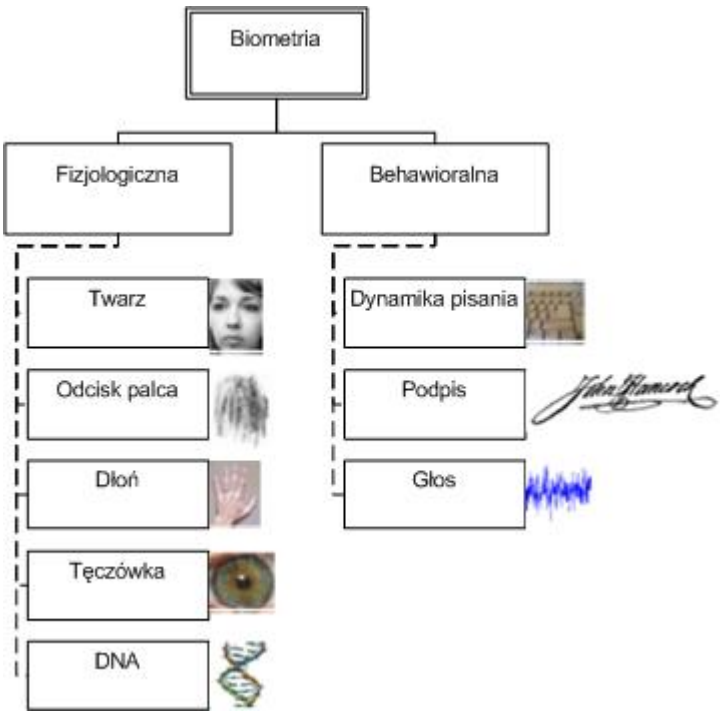
3.6. Biometria

Zgodnie z [14] "**biometria** jest elementem wydobywającym porządek z chaosu, czynnikiem pozwalającym przezwyciężyć podstawową sprzeczność, jaka istnieje pomiędzy naturą zindywidualizowanych osobniczo obserwacji biologicznych i wywodzącymi się z ideałów nauk ścisłych tendencji do formułowania sądów ogólnych i uniwersalnych."

Według [4] biometrię dzielimy na:

- **fizjologiczną** - powiązaną z kształtem ciała, np. odciska palca, rozpoznawanie twarzy, geometrii rąk lub dłoni oraz rozpoznawanie tęczówki.
- **behawioralną** - powiązaną z zachowaniem, np. weryfikacja podpisu, rytm pisania, sposób chodzenia i głos.

Podział przedstawia poniższy rysunek zaczerpnięty z [4].



Rysunek 3.4: Podział biometrii

Miejsce dynamiki pisania na tle innych sposobów rozpoznawania osób przedstawia tabelka porównawcza stworzona przez A. K. Jain. Źródło [4]. Porównane zostały następujące parametry:

- uniwersalność - każda osoba powinna posiadać badaną charakterystykę.
- unikalność - określa jak dobrze badana wartość odróżnia osobę od pozostałych ludzi.
- trwałość - określa jak bardzo parametr odporny jest na starzenie.
- pobieralność - określa łatwość pobrania badanej wartości.
- występowanie - natężenie, prędkość i odporność na zakłócenia.
- akceptowalność - stopień akceptowalności technologii.
- podstępność - łatwość użycia substytutów.

Tablica 3.1: Porównanie cech biometrycznych; H - wysoka, M - średnia, L - niska

Biometria	uniwersalność	unikalność	trwałość	pobieralność	występowanie	akceptowalność	podstępność
Twarz	H	L	M	H	L	H	L
Odcisk palca	M	H	H	M	H	M	H
Geometria ręki	M	M	M	H	M	M	M
Dynamika pisania	L	L	L	M	L	M	M
Żyły na ręce	M	M	M	M	M	M	H
Tęczówka	H	H	H	M	H	L	H
Siatkówka oka	H	H	M	L	H	L	H
Podpis	L	L	L	H	L	H	L
Głos	M	L	L	M	L	H	L
Temperatura twarzy	H	H	L	H	M	H	H
Zapach	H	H	H	L	L	M	L
DNA	H	H	H	L	H	L	L
Chód	M	L	L	H	L	H	M
Kanaliki uszne	M	M	H	M	M	H	M

4. Projekt

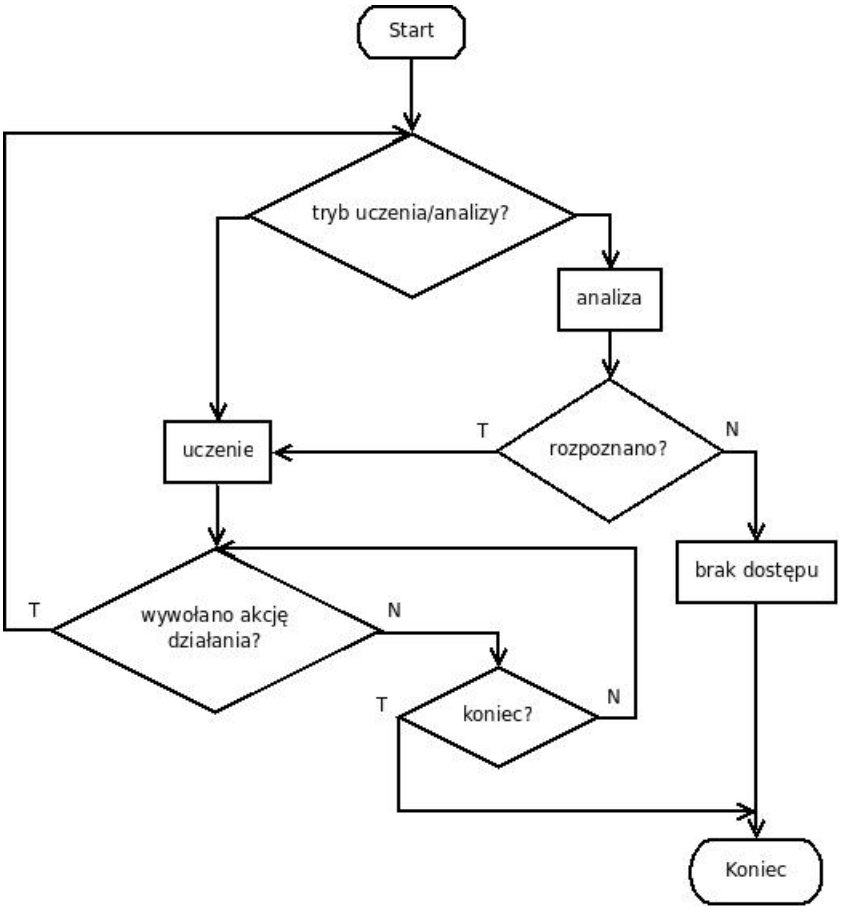
4.1. Wymagania

Aplikacja musi posiadać 2 tryby uruchomienia.

- tryb uczenia,
- tryb rozpoznania.

W trybie uczenia zbierane są dane i sieci neuronowe są uczone. W trybie rozpoznania na podstawie zebranych danych tworzona jest próbka i sieci są testowane (następuje rozpoznanie użytkownika). W przypadku błędnego rozpoznania podejmowana jest akcja zapobiegająca nieuprawniony dostęp. W przypadku poprawnego rozpoznania próbka służy jako zbiór uczący.

Program powinien mieć niewiele opcji i prosty interfejs dla użytkownika. Powinien działać w tle (chować się do paska stanu) i w jak najmniejszym stopniu zakłócać codzienną pracę.



Rysunek 4.1: Schemat blokowy aplikacji

4.2. Projekt rozwiązania

4.2.1. Zbieranie i przechowywanie danych

Pierwszym etapem jest zbieranie danych potrzebnych do analizy. Do tego celu zostały użyte tzw. haki (hooks). Haki to kolejkowy mechanizm, przez który przechodzi każdy komunikat w systemie. W tym także komunikaty pochodzące od klawiatury (WH_KEYBOARD) oraz od myszki (WH_MOUSE). Dane znajdujące się w kolejce mogą być odczytywane oraz zmieniane. Haki klawiatury oraz myszki mogą być włączone w trybie globalny lub dla pojedynczego wątku. Został użyty tryb globalny. W celu uruchomienia haków należy użyć specjalnej funkcji windowsowej:

```
HHOOK SetWindowsHookEx(
    int idHook,        // typ hook'a do założenia
    HOOKPROC lpfn,     // adres funkcji obsługi hook'a
    HINSTANCE hMod,    // hInstance aplikacji / biblioteki dll.
    DWORD dwThreadId   // identyfikator wątku hook?a
);
```

hMod - Identyfikuje DLL zawierający procedurę hook'a, na którą wskazuje parametr lpfn. Parametr hMod musi być ustawiony na NULL jeżeli dwThreadId określa wątek utworzony przez bieżący proces i jeżeli procedura hook'a zaimplementowana jest w kodzie powiązanym z bieżącym procesem.

dwThreadId - musi mieć wartość NULL jeśli chcemy by nasz hook był globalny.

W skonstruowanej aplikacji wywołanie wygląda następująco:

```
g_hLogHook = SetWindowsHookEx(WH_KEYBOARD_LL, (HOOKPROC)KeyboardProc,
     HInstance, 0);
```

Haki usunięto za pomocą funkcji windowsowej:

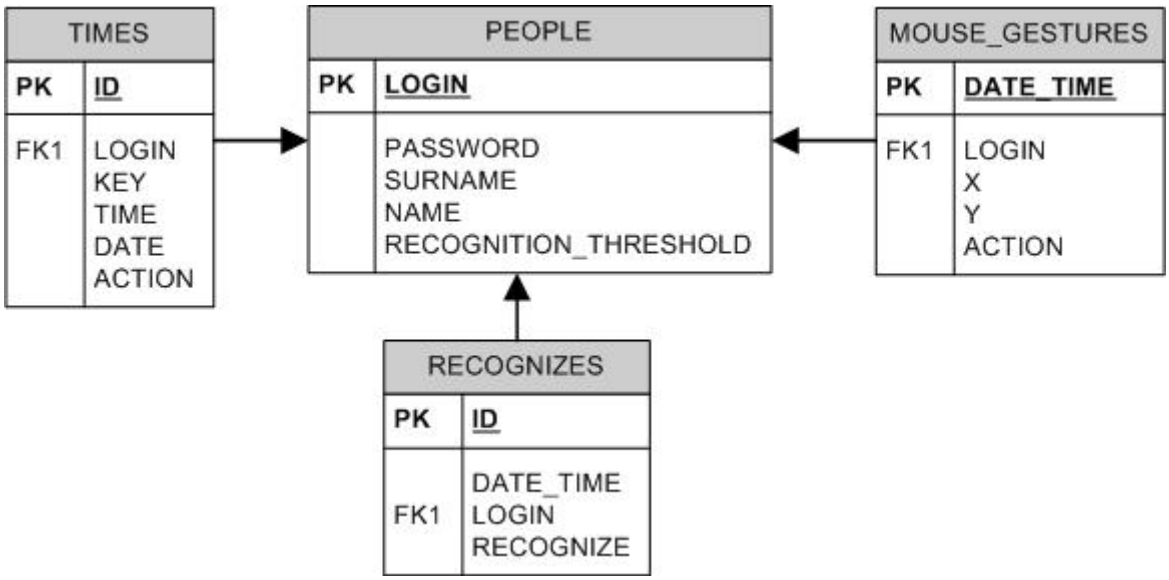
```
BOOL UnhookWindowsHookEx(
    HHOOK hhk  // Uchwyt do naszego wcześniej utworzonego hook'a );
```

Następnie trzeba napisać funkcję obsługującą nasz hak (hooka), czyli funkcję **KeyboardProc**. W ogólnej postaci może ona wyglądać tak:

```
LRESULT CALLBACK KeyboardProc
    (int nCode, WPARAM wParam, LPARAM lParam)
{
     char tab[255];
     if (nCode < 0)
         return CallNextHookEx(hhookKeyb, nCode, wParam, lParam);
     if((lParam & 0x80000000) == 0)
         return CallNextHookEx(hhookKeyb, nCode, wParam, lParam);
     wsprintf(tab, "%x %c", wParam, wParam);
     MessageBox(NULL, tab, "Wirtualny kod wciśniętego klawisza",
                MB_OK);
     return CallNextHookEx(hhookKeyb, nCode, wParam, lParam);
}
```

Wyrażenie $if(nCode < 0)$ sprawdza czy został wczytany poprawny kod klawisza. W przypadku błędu należy przekazać dalej obsługę haków (hooks) za pomocą funkcji **CallNextHookEx**. Wyrażenie $if((lParam \& 0x80000000) == 0)$ służy do rozróżnienia wciśnięcia klawisza od zwolnienia klawisza. Zmienna wParam przechowuje wirtualny kod klawisza. Zmienna lParam przechowuje takie informacje jak: kod klawisza, czas, rodzaj i akcję klawisza (np. systemowy), ilość powtórzeń.

Informację zaczerpnięte ze strony firmy Microsoft [6].



Rysunek 4.2: Schemat ERD bazy danych do zbierania danych

Kolejnym etapem jest zmagazynowanie danych. W tym celu zaprojektowana została baza o strukturze widocznej na rysunku 4.2.

Tabela *PEOPLE* zawiera informacje o użytkowniku oraz jego próg rozpoznania. Tabela *RECOGNIZES* zawiera rozpoznania wykonane o określonej porze. Tabele *TIMES* oraz *MOUSE_GESTURES* zawierają czasy wciśnięć klawiszy oraz gestów myszki.

4.2.2. Wyłączenia systemu i wylogowanie użytkownika

Jednym z założeń aplikacji jest uniemożliwienie dalszego działania intruzowi lub utrudnienie działania. Do tego celu wykorzystano możliwości jakie oferuje funkcja

```
 BOOL WINAPI ExitWindowsEx(
  __in  UINT uFlags,
  __in  DWORD dwReason
);
```

zawarta w API Windowsów. Zgodnie z [5] dostępne są następujące działania

EWX_LOGOFF – Kończy wszystkie uruchomione procesy użytkownika, a następnie wylogowuje go.

EWX_POWEROFF – Wyłącza system.

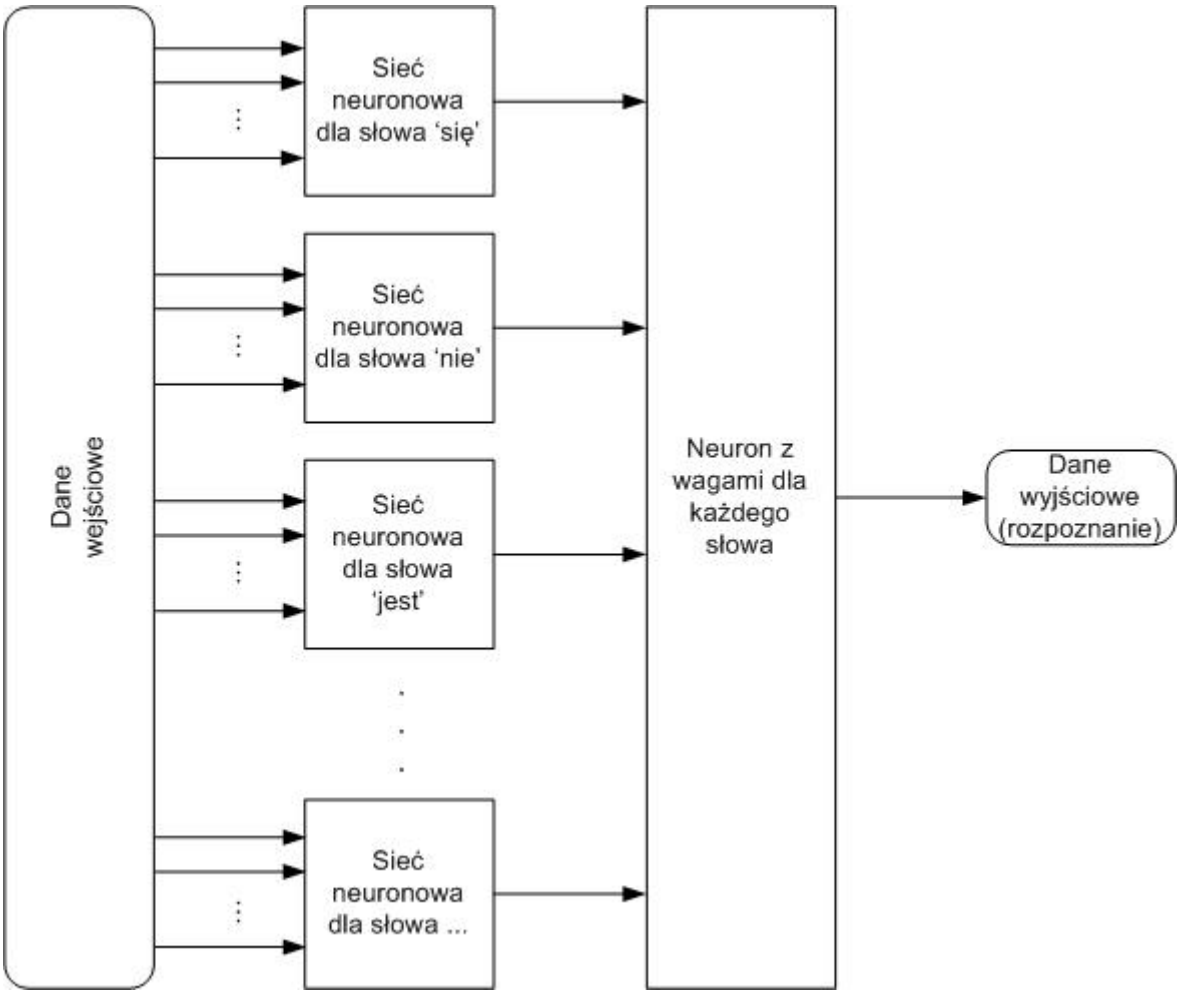
EWX_REBOOT – Restartuje system.

EWX_RESTARTAPPS – Restartuje system, zamykając każdą aplikację zarejestrowaną do restartu za pomocą funkcji RegisterApplicationRestart.

EWX_SHUTDOWN – Wyłącza system do punktów, w którym komputer może zostać bezpiecznie wyłączony.

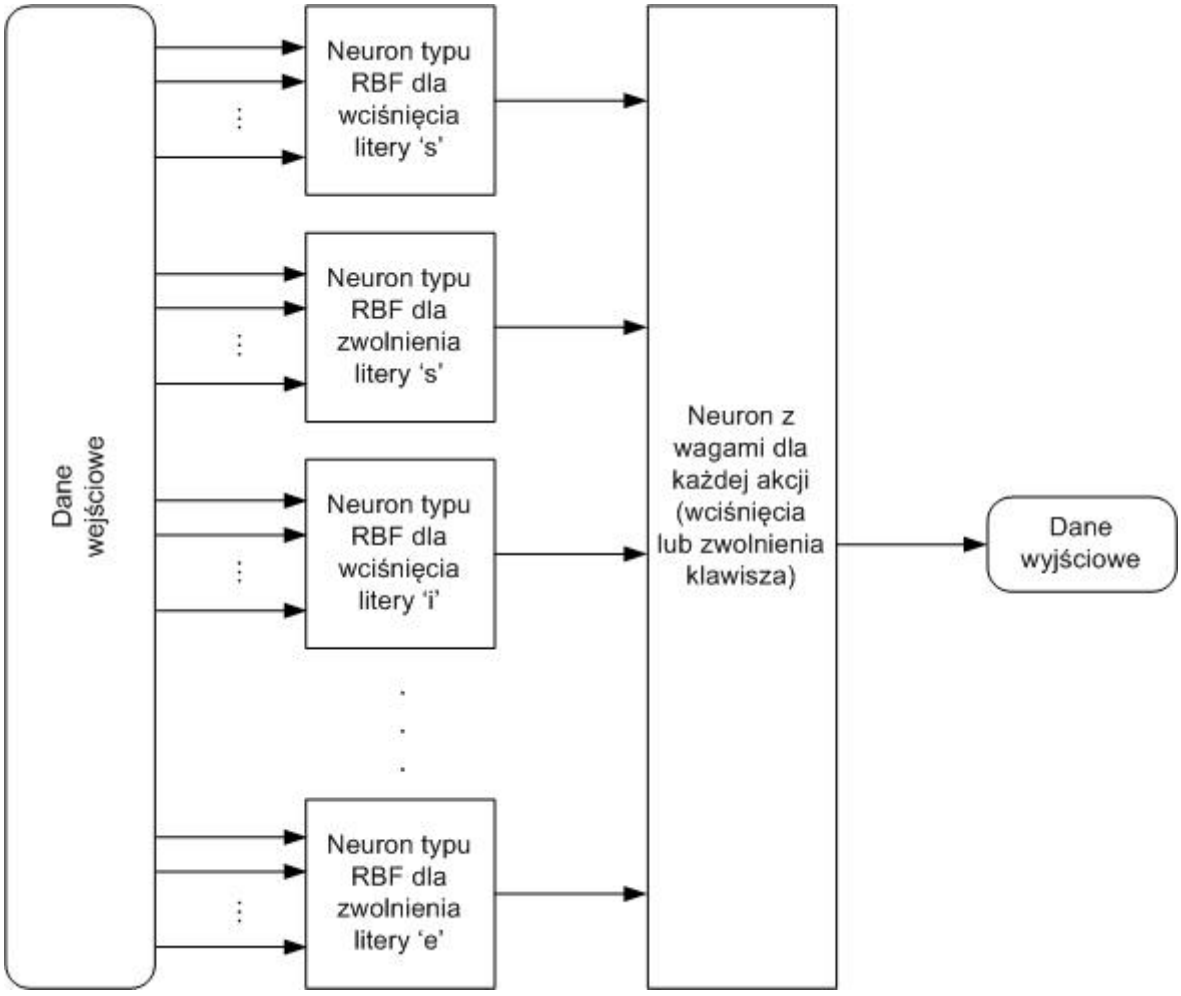
4.2.3. Projekt sieci neuronowej

W oparciu o wiedzę zdobytą podczas analizy dotychczasowych rozwiązań (rozdział 2) oraz wiedzę przekazaną przez promotora została opracowana sieć neuronowa służąca do rozpoznawania osób. Główne założenie sieci wprowadza podział na podsieci według słów (pomysł promotora dr Adriana Horzyka). Pokazane jest to na rysunku poniżej. Wynik z każdej podsieci stanowi wejście neuronu wyliczającego rozpoznanie osoby. Wagi tego neuronu wyliczane są na podstawie częstotliwości używania słów przez użytkownika.



Rysunek 4.3: Schemat sieci neuronowej - poziom 1

Sięgając w głąb podsieci dla danego słowa zobaczyć można neurony typu RBF (opisane w 3.3) oraz neuron obliczający wartość rozpoznania dla tego słowa. W neuronach wejściowych (RBF) centra (μ) ustalane są na podstawie algorytmu. Algorytm wylicza średnią wartość czasu dla danej akcji (wciśnięcia, zwolnienia przycisku). Tutaj została wprowadzona modyfikacja neuronów RBF - zostały dodane wartości promieni w obu kierunkach osi tzw. promienie (ρ), wyliczane na podstawie maksymalnych i minimalnych odchyleń od centrum. Wagi w neuronie wyjściowym obliczane są podczas procesu uczenia z nauczycielem. Wartością oczekiwaną jest oczywiście wartość 1 (rozumiana jako 100%). Wartości wejściowe stanowią dane przepropagowane przez warstwę neuronów typu RBF. Poniższy schemat przedstawia opisaną strukturę.



Rysunek 4.4: Schemat sieci neuronowej - poziom 2

Uczenie nieliniowych neuronów wyjściowych w podsieciach dla poszczególnych słów.

Zgodnie z [12] proces uczenia opieramy na regule minimalizacji funkcjonału błędu średniokwadratowego:

$$Q = \frac{1}{2}\sum_{j=1}^{N}(z^{(j)} - y^{(j)})^2$$

gdzie $y^{(j)} = f(\sum_{j=1}^{N} w_i^{(j)} x_i^{(j)})$,

$z^{(j)}$ - wartość oczekiwana na wyjściu,

w - wagi,

x - wartości na wejściu sieci.

Funkcjonał błędu możemy rozłożyć na elementy składowe

$$Q = \sum Q(j)$$

gdzie

$$Q(j) = \sum_{j=1}^{N} (z(j) - y(j))^2$$

Przy takich założeniach zmiana wag wynosi:

$$w_i(j+1) - w_i(j) = \Delta w_i(j) = -\eta \frac{\partial Q(j)}{\partial w_i}$$

$$\frac{\partial Q(j)}{\partial w_i} = \frac{\partial Q(j)}{\partial y_i^{(j)}} \frac{\partial y_i^{(j)}}{\partial w_i} = \frac{\partial Q(j)}{\partial y_i^{(j)}} \frac{\partial y_i^{(j)}}{\partial e^{(j)}} \frac{\partial e^{(j)}}{\partial w_i}$$

gdzie można obliczyć:

$$\frac{\partial Q(j)}{\partial y_i^{(j)}} = -(z^{(j)} - y^{(j)})$$

$$\frac{\partial e(j)}{\partial w_i} = x_i^{(j)}$$

natomiast

$$\frac{\partial y_i^{(j)}}{\partial e(j)} = \frac{\partial f(e)}{\partial e^{(j)}}$$

ostatecznie

$$\Delta w_i^{(j)} = -\eta (z^{(j)} - y^{(j)}) \frac{\partial f(e)}{\partial e^{(j)}} x_i^{(j)}$$

W pracy magisterskiej została zastosowana powszechnie używana funkcja

$$y = f(e) = 1/(1 + exp(-\beta e))$$

której pochodna

$$\frac{\partial f(e)}{\partial e(j)} = y^{(j)}(1 - y^{(j)})$$

zatem ostatecznie

$$\Delta w_i^{(j)} = -\eta (z^{(j)}) y^{(j)} (1 - y^{(j)}) x_i^{(j)}$$

Sieć neuronowa tworzona jest tylko dla słów najczęściej występujących w języku polskim. Została stworzona w oparciu o [13].

Cała sieć neuronowa oraz słownik przechowywana jest w oddzielnej bazie danych. Centrum całej struktury jest słownik tj. tabela *DICTIONARY*. Kolumny:

WORD – przechowuje słowo,

FREQ – przechowuje wartość określającą częstotliwość występowania w języku polskim.

Tabela *NN* zawiera przekrój przez sieć neuronową. Kolumny:

NO – index,

WORD – słowo,

LETTER – litera, akcja i poprzednik,

MI – centrum neuronu RBF,

RO_L – odchylenie w lewo od centrum,

RO_R – odchylenie w prawo od centrum,

ITER – ilość przeprowadzonych uczeń danego słowa/litery,

WEIGHTS – waga dla danego słowa.

Tabela *LAST_ATTEMPTS* zawiera *n* ostatnich próbek, którymi była uczona sieć neuronowa. Kolumny:

ATTEMPT – index próbki,

TIME_ID – index czasu/litery,

WORD – słowo,

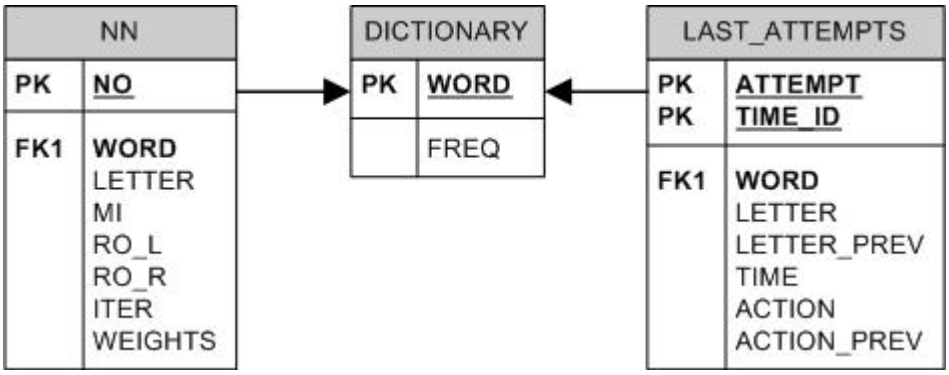
LETTER – litera,

LETTER_PREV – litera poprzedzająca,

TIME – czas (znormalizowany dla danego słowa),

ACTION – akcja (wciśnięcie lub zwolnienie),

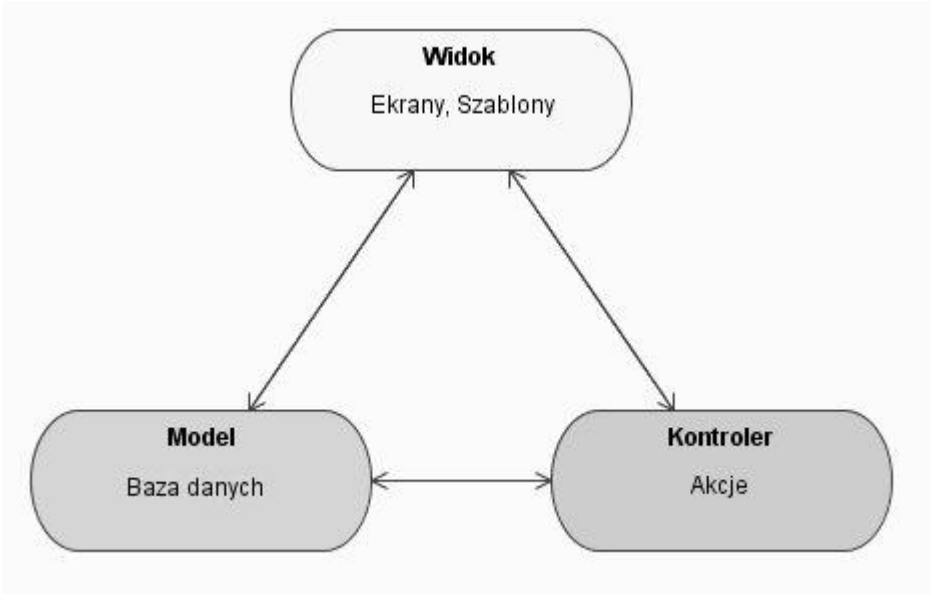
ACTION_PREV – akcja poprzedniej litery (wciśnięcie lub zwolnienie).



Rysunek 4.5: Diadram ERD bazy danych sieci neuronowej

4.2.4. Aplikacja - MVC

Aplikacja stworzona jest w oparciu o klasyczny model MVC - Model Widok Kontroler. Model stanowią dwie bazy danych. Baza przechowująca akcje użytkownika tj. wciśnięcia/zwolnienia klawiszy, gesty myszki. Baza przechowująca sieć neuronową. Widok stanowi forma zawierająca opcje konfiguracji, formy przedstawiające proces uczenia/testowania (progress bary) oraz inne formy pomocnicze. Do kontrolera należy cała sieć neuronowa wraz z operacjami na niej, funkcje odpowiedzialne za zbieranie danych z klawiatury i myszki.



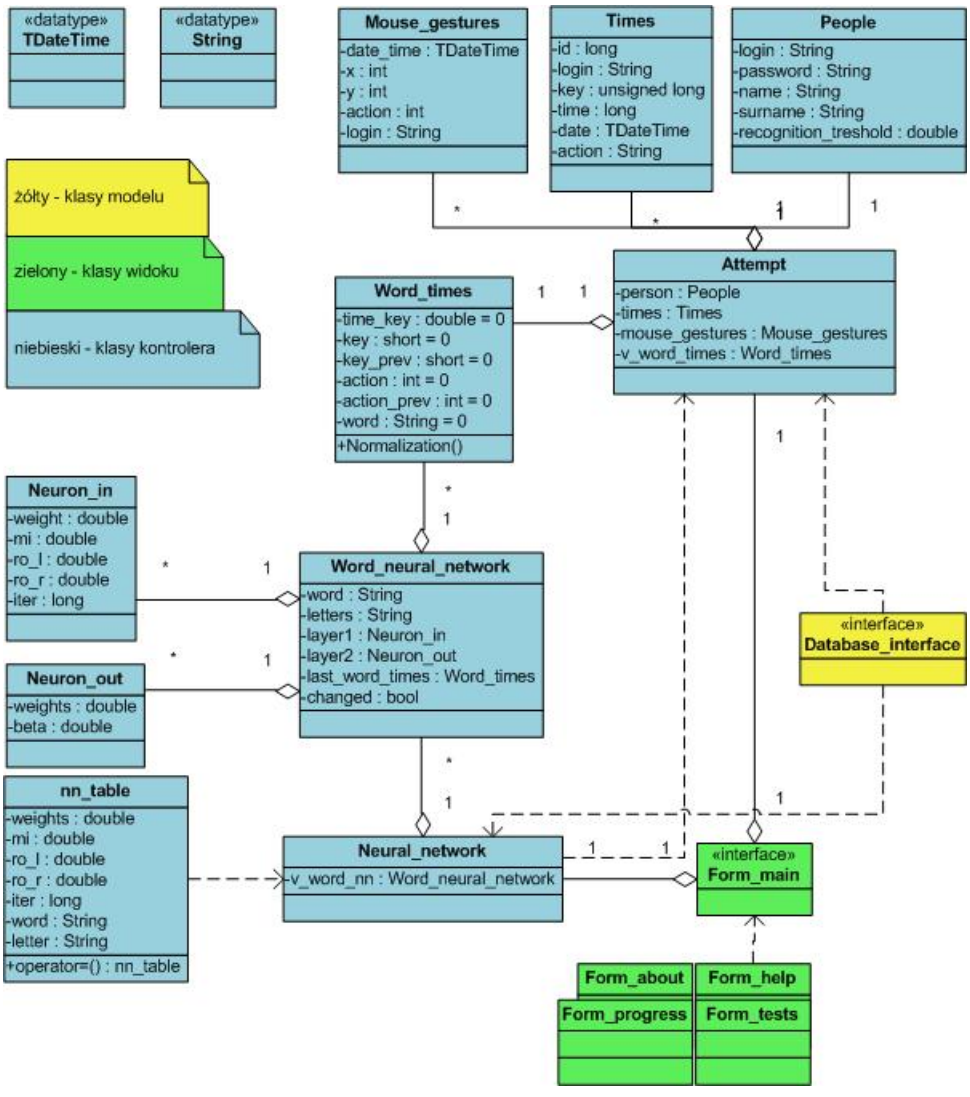
Rysunek 4.6: Model Widok Kontroler

5. Realizacja

5.1. Implementacja

Implementacja została wykonana w języku *C/C++* w środowisku *Borland C++ Builder 5.0*. Użyta baza danych to baza *Firebird* oraz program *IBExpert* do zarządzania bazą danych. Do programu Borland C++ Builder 5.0 zostały doinstalowane komponenty umożliwiające obsługę bazy danych tj. łączenie, wykonywanie zapytań SQL'owych. Więcej informacji o Firebirdzie dostępnych jest na stronie [9].

Poniżej zamieszczam diagram klas obrazujący wszystkie klasy w systemie wraz z atrybutami.



Rysunek 5.1: Diagram klas z podziałem na funkcje modelu MVC

Jak widać na rysunku 5.1 istnieje tylko jedna klasa odpowiedzialna za komunikację z bazą danych. Jest to klasa *database_interface*. Zawiera ona szereg funkcji pogrupowanych według tabel, do których odnoszą się zapytania. Znajdują się tam funkcje:

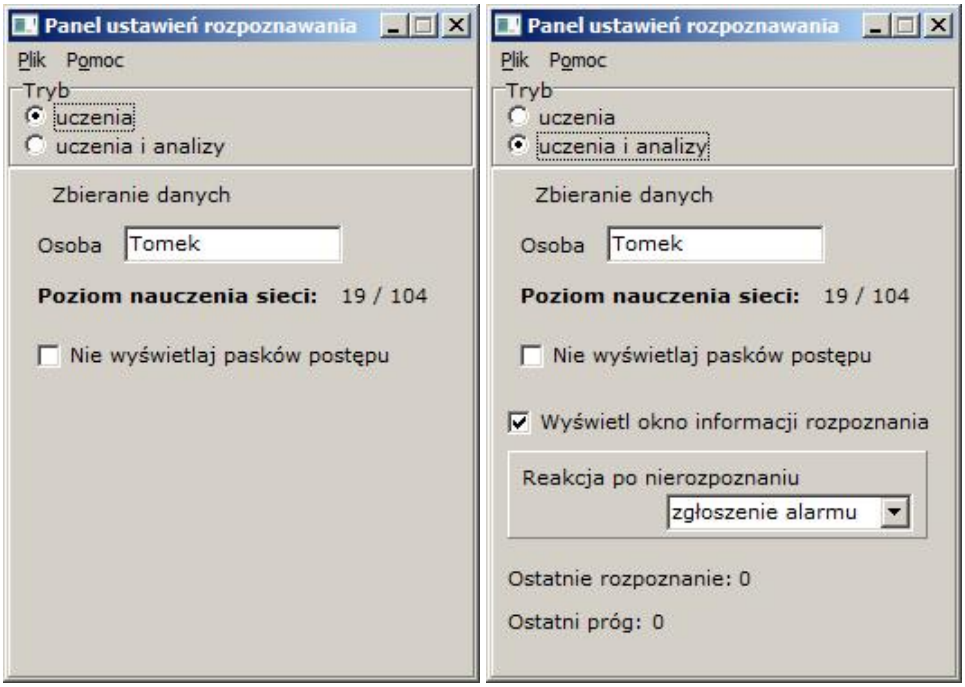
- ustanawiania połączenia z bazą - *set_database*,
- obsługi użytkowników - *check_user*, *check_login*, *add_person*...
- obsługi progu rozpoznania - *get_persons_recognition_threshold*,
- obsługa próbek (czasów wciśnięć klawiszy) - *add_time*.

```
class Database_interface
{
public:
    static int set_database(String path, String uri, String user, String password);
    static int set_database_NN(String path, String uri, String user, String password);


    static int check_user(String login, String password);
    static int check_login(String login);
    static int add_person(People person);
    static int update_person(People person);
    static String get_first_person();
    static String * Database_interface::get_people(int* size);

    static double get_persons_recognition_threshold(String login);
    static int add_recognize(String login, float recognize);
    static double * get_last_n_recognizes(String login, int n);

    static int insert_word_recognition(String login, String word, double recognition);
    static double get_word_threshold(String login, String word);

    static int add_time(Times czas1);
    static vector<Times> * get_times(String login, long key);
    static vector<Times> * get_times(String login, TDateTime date_from, TDateTime date_to);
    static vector< vector<Times > > * get_all_times(String login, int duration);

    static int add_times_between(Times_between czas2);
    static vector<Times_between> get_times_between(String login, char key1, char key2);

    static int add_mouse_gestures(Mouse_gestures mouse_gestures);
    static vector<Mouse_gestures> Database_interface::get_mouse_gestures(
        TDateTime date_time, String login, int x, int y, int action);



    static int insert_NN(long cx, long cy, float mi, float ro_l, float ro_r,
        long iter, float weights);
    static int update_NN(long cx, long cy, float mi, float ro_l, float ro_r,
        long iter, float weights);
    static int select_NN(long cx, long cy);
    static vector<NN_table> * select_all();



    static int insert_NN(String word, String letter, double mi, double ro_l, double ro_r,
        long iter, double weights, long no);
    static int update_NN(String word, char letter, double mi, double ro_l, double ro_r,
        long iter, double weights);
    static int delete_NN();
    static int delete_NN(String word);

    static int count_fully_tought_words();
    static int count_tought_words();



    static long count_words();

    static String get_word(long i);
    static int load_from_file(String filename);
    static int exist_word(String word);



    static int count_last_attempts(String word);
    static int count_last_attempts_letters(String word, int attempt);

    static double get_attempt(String word, int attempt, int index);
    static int get_attempt(String word, int attempt, int index,
        double* rtime, int* rkey, int* rkey_prev,
        short* raction, short* raction_prev);

    static double get_attempt_time(String word, int attempt, int index);
    static int get_attempt_key(String word, int attempt, int index);
    static int get_attempt_key_prev(String word, int attempt, int index);
    static short get_attempt_action(String word, int attempt, int index);
    static short get_attempt_action_prev(String word, int attempt, int index);

    static int insert_last_word_times(Word_times & word_times, int attempt_index);
    static int delete_last_word_times();
    static int delete_last_word_times(String word);
```

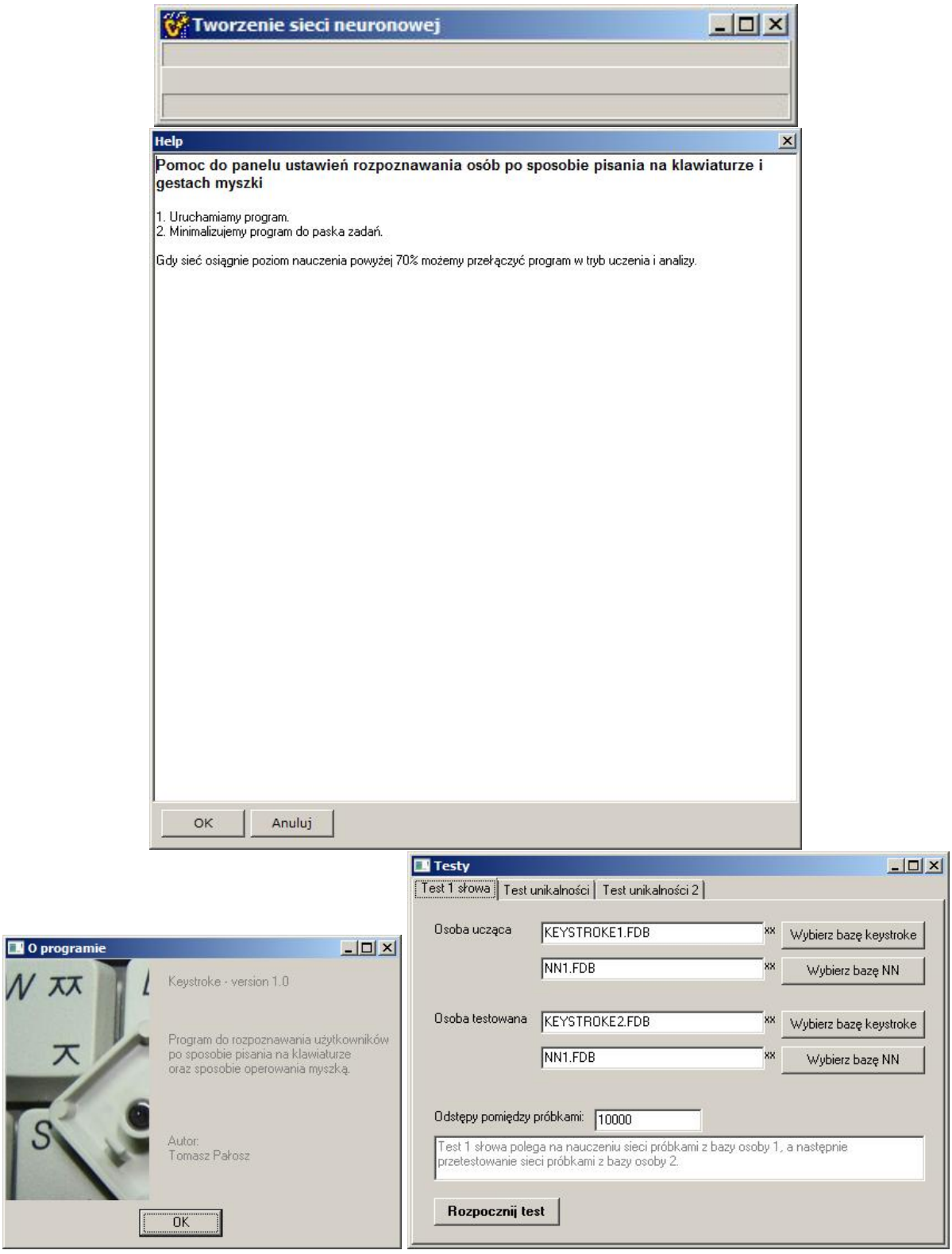
Grupę odrębnych klas stanowią formy odpowiedzialne za prezentację danych. Formy te należą do Widoku w modelu MVC. Są to:

- Form_main - główna forma panelu ustawień (rys. 5.1),
- Form_about - forma wyświetlająca informację o programie,
- Form_help - forma zawierająca pomoc do programu,
- Form_progress - ogólna forma postępu czynności, np. uczenie, testowanie,
- Form_tests - dodatkowa forma do testów sieci neuronowej.

Pozostałe formy przedstawione są na rysunku zbiorczym 5.1.



Rysunek 5.2: Główna forma - panel ustawień programu.



Rysunek 5.3: Formy według kolejności: Form_progress, Form_help, Form_about, Form_tests

Największą grupę stanowią klasy Kontrolera. Składają się na nią klasy sieci neuronowej w pełni zaimplementowanej od postaw oraz klasy zbierania próbek (Attempt). Budowa sieci neuronowej opisana jest w rozdziale 4.2.3.

- **Attempt** - klasa odpowiada za przechowywanie próbek. Zawiera klasy:
  - **People** - klasa mapująca osoby z bazy danych. Osoba rozpoznana lub osoba potencjalnie uważana za osobę piszącą.
  - **Times** - klasa mapująca czasy wciśnięć klawiszy.
  - **Mouse_gestures** - klasa mapująca gesty myszki.
  - **Word_times** - klasa przechowująca przetworzoną próbkę z klawiatury tj. czasy wciśnięć dla słowa.
- **Neural_network** - klasa będąca odwzorowaniem całej sieci neuronowej, realizująca funkcjonalność poziomu 1 w schemacie sieci neuronowej (rysunek 4.3). Klasa wykorzystuje klasę nn_table do szybszego odczytu sieci neuronowej z bazy danych.
- **Word_neural_network** - klasa będąca odwzorowaniem podsieci neuronowej dla pojedynczego słowa, realizująca funkcjonalność poziomu 2 w schemacie sieci neuronowej (rysunek 4.4). Klasa zawiera następujące klasy:
  - **Neuron_in** - klasa odpowiedzialna za funkcjonalność warstwy neuronów typu RBFN.
  - **Neuron_out** - klasa odpowiedzialna za funkcjonalność neuronu wyjściowego z podsieci dla pojedynczego słowa.
  - **Word_times** - klasa przechowująca przetworzoną próbkę z klawiatury tj. czasy wciśnięć dla słowa. W tej klasie wykorzystywana do tymczasowego przechowywania danych potrzebnych do uczenia w kolejnych epokach.

W ramach implementacji została opracowana dodatkowa klasa *Parameters*. Jej celem jest przechowywanie parametrów, od których zależy funkcjonowanie sieci neuronowej oraz programu. Takie zestawienie ułatwia dopracowanie działania algorytmu. Klasa posiada konstruktor, wpisujący wartości domyślne parametrów. Zestawienie parametrów, opis oraz wartości domyślne parametrów znajdują się poniżej.

```
class Parameters
{
public:
    int timer_watchdog; /* dopuszczalny czas bezczynności po którym
                           uruchamiane jest uczenie lub test sieci
                           neuronowej*/
    int last_teach;     /* ilość próbek zapamiętywanych wstecz dla każdego
                           słowa*/
                        /* ---+dodatkowo last_teach_const w pliku
                           word_neural_network.h*/
    int teach_amount;   /* ilość uczeń danym słowem, zwana liczbą epok*/

    int min_recreate;   /* gry ilość uczeń wykonanych na wartwie wejściowej
                           (neuron_in) jest mniejsza od tej wartości
                           i próbka wymusza stworzenie sieci od nowa to
                           ponowne stworzenie sieci nastąpi*/
    double epsilon;     /* minimalna początkowa wartość x*/

    int max_iter;       /* maksymalna ilość uczenia sieci zewnętrznej
                           (neuron_out)*/

    double ni;          /* współczynnik szybkości uczenia sieci*/

    double const_ro;    /* współczynnik poszerzenia przedziału
                           dopuszczalnych wartości x*/
```

```
    double epsilonn;     /* warunek stopu określający minimalną wartość
                            różnicy wyjścia neuronu*/
    double access;       /* próg poprawnego rozpoznania*/

    int last_threshold; /* ilosc ostatnich sprawdzanych rozpoznan sluzacych
                           do wyznaczenia progu*/

    double minus_threshold;/* wartosc odejmowana od minimalnej wartosci
                              progu*/

    double min_good_nn; /* minimalny współczynnik okreslajacy poziom
                           nauczenia sieci*/

    int max_iter_value; /* maksymalna wartosc zmiennej iter w sieci
                           neuron_in zmienna odpowiada za ilosc wartosci
                           skladajacych sie na centrum*/
    int min_words;       /* minimalna ilość słów w próbce */

public:
    Parameters();
};
```

```
Parameters::Parameters()
{
    timer_watchdog = 10000;

    last_teach = 5;

    teach_amount = 3;

    min_recreate = 40;

    epsilon = 0.001;

    max_iter = 10000;

    ni = 0.8;

    const_ro = 0.0;

    epsilonn = 0.001;

    access = 0.7;

    last_threshold = 5;

    minus_threshold = 0.001;

    min_good_nn = 0.7;

    max_iter_value = 1000;

    min_words = 5;
}
```

5.2. Testowanie aplikacji i dokumentacji

Testowanie jest jednym z kluczowych elementów powstawania aplikacji. W modelu spiralnym tworzenia aplikacji po etapie zakończenia testów możliwe jest udostępnienie aplikacji do użytku lub części funkcjonalności, która była testowana.

5.2.1. Testy statyczne - przegląd kodu

Analiza kodu została wykonana po zakończeniu implementacji. W etapie tym zostały uporządkowane niektóre fragmenty kodu, związane z etapami przejścia między badaniami. Wypełnione zostały także funkcje pełniące rolę opcjonalną, a nie używane przeze mnie.

Poszukiwanie typowych błędów, np. underflow dla typów double, było wykonywane kilkukrotnie. Głównie po znalezieniu błędów podczas testów funkcjonalnych lub dynamicznych (5.2.3 oraz 5.2.2).

5.2.2. Testy dynamiczne - testy jednostkowe

Testy dynamiczne wykonywane były ręcznie tj. nie zostało użyte żadne oprogramowanie wspierające. Tworzone były scenariusze testowe, wykonywane ręcznie i wynik oczekiwany porównywany był z wynikiem oczekiwanym. W ten sposób testowane były głównie elementy nie związane z samą siecią neuronową. Za testowanie algorytmu rozpoznawania odpowiadały dodatkowo przeprowadzane testy (5.3).

5.2.3. Testy funkcjonalne

Testy funkcjonalne były przeprowadzane przez kilka osób. Polegały na rozesłaniu aplikacji do osób, oraz wpisywanie określonego wyrażenia. W ten sposób został znalezionych jeden z podstawowych błędów tj. underflow. Dalsze błędy underflow zostały wyeliminowane dzięki analizie kodu (5.2.1).

Do testów funkcjonalnych należało także uruchamianie aplikacji podczas wykonywania codziennych czynności, np. rozmowy za pomocą komunikatorów (gadu-gadu, tlen), pisanie pracy magisterskiej, programowanie. W ten sposób sprawdzana była odporność programu na zakłócenia oraz opóźnienia występujące w kolejce komunikatów pochodzących z klawiatury. Działanie programu wystawiane było na działanie takich programów jak: Borland C++ Builder, Visual Studio 2008, Winamp, tlen, skype, odtwarzacz filmów - subedit, allplayer, ... Badanie miało na celu ogólnego ocenienia działania aplikacji, a nie stwierdzenie poziomu wpływu tychże aplikacji.

5.2.4. Sprawdzanie dokumentacji

Dokumentacja również została poddana sprawdzeniu. Czytały ją osoby z kierunków humanistycznych. Zostały wyłapane drobne błędy merytoryczne, stylistyczne oraz językowe, które nie zostały poprawione przez słownik. Błędy takie są trudno zauważalne przez samego autora.

Po takim dokładnym przeczytaniu dokumentacji przez osoby trzecie, powstawała lista błędów w postaci:

```
str.7. wtedy to zainteresowałem się, także - przecinek przed także,
str.7. po zalogowaniu, wiemy (przecinek)
str.10. Ekran lagowania -> Ekran logowania,
str.14. Przecięcie błędów jest określa - bez "jest".
str.31. zbadane wiec -> zbadane więc
str.33. różnorodne (razem)
...
```

5.3. Testowanie algorytmu

W testach uczestniczyło kilka osób. Dobór osób opierał się na zachowaniu wyraźnych różnic pod względem opisywanych cech. Według mnie istotny może być profil danego użytkownika, jako informacja dająca obraz co można poprawić w algorytmie w przyszłości.

1. **Tomek** - autor pracy, osoba pisząca bardzo szybko, wszystkimi palcami równomiernie, używająca komputera na codzień, login: Tomek

2. **Mateusz** - kolega ze studiów, osoba pisząca szybko, używająca do 3 palców z każdej dłoni, używająca komputera na codzień, login: mati

3. **Michał** - brat użytkownika Tomek, osoba pisząca szybko, używająca do 4 palców z każdej dłoni, używająca komputera na codzień (gracz gier typu Counter Strike), login: michal

4. **Barbara** - osoba pisząca bardzo wolno, używająca 1 palca z każdej dłoni, używająca komputera sporadycznie,

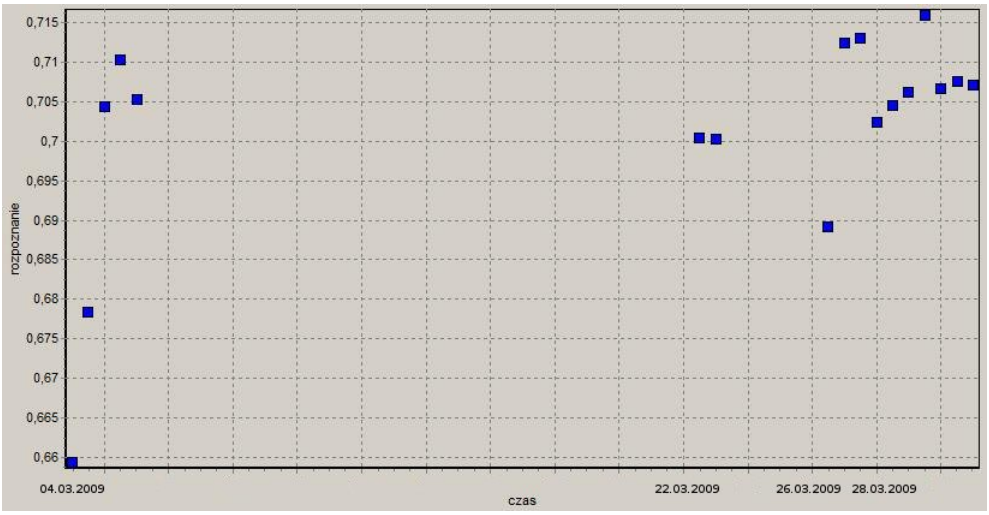
5. **Róża** - osoba pisząca szybko, używająca do 3 palców z każdej dłoni, używająca komputera na codzień w pracy, login ppp

5.3.1. Test jednego słowa

Pierwszym testem sieci neuronowej było sprawdzenie działania dla jednego słowa, czyli przebadanie podsieci (4.2.3). Wybranym słowem było słowo *dobrze*.

Test polegał na nauczeniu sieci neuronowej dużą ilością wpisów badanego słowa. Podczas tego testu został ustalony parametr *próg ponownego tworzenia podsieci - min_recreate* (5.1). Ilość wejść do podsieci zależy od długości słowa, co przekłada się na ilość akcji (wciśnięć lub zwolnień klawiszy). Istnieją przypadki, w którym ta ilość nie jest równa 2 razy ilość klawiszy (w tym Alt'ów, Shift'ów). Przypadek ten występuje m.in. wtedy, gdy dana osoba wciśnie spację po zakończeniu słowa, wcześniej niż zwolni ostatni klawisz. Wtedy akcja zwolnienia tego klawisza jest tracona. Parametr *min_recreate* służy do określenia minimalnej ilości wprowadzeń słowa (o tej samej ilości akcji), wymaganej uznania podsieci za stabilną.

Etapy uczenia sieci zostały przedstawione na rysunku 5.4. Czas określony jest jako dzień + numer rozpoznania. Uczenie zostało wykonane między 3, a 23 marca 2009 roku. Na rysunku widać charakterystyczne dla każdego uczenia dochodzenie do określonej wartości.



Rysunek 5.4: Postęp uczenia sieci neuronowej słowem "dobrze" dla osoby o loginie "Tomek"

Etap testowania sieci wykonany został dla kilku osób. Gdy była to osoba, dla której została stworzona sieć następowało dalsze douczanie sieci. W przypadku pozostałych osób douczanie nie następowało. Wyniki rozpoznania oraz progi rozpoznania zostały zebrane w tabeli poniżej.

Tablica 5.1: Test jednego słowa dla użytkownika “Tomek”

Osoba	słowo	rozpoznanie	próg	rozp. 0-1
Tomek	dobrze	0,700147794511562	0,680700211601176	1
Tomek	dobrze	0,703857945909181	0,680700211601176	1
Tomek	dobrze	0,709192783562767	0,680700211601176	1
Tomek	dobrze	0,719685986011492	0,680700211601176	1
Tomek	dobrze	0,725294551540492	0,680700211601176	1
Tomek	dobrze	0,699806048031234	0,680700211601176	1
Tomek	dobrze	0,706240686577346	0,680700211601176	1
Tomek	dobrze	0,687032243106731	0,680700211601176	1
Tomek	dobrze	0,701110625001857	0,680700211601176	1
Tomek	dobrze	0,702721987705237	0,680700211601176	1
Tomek	dobrze	0,710161257955418	0,680700211601176	1
Tomek	dobrze	0,720862238111913	0,680700211601176	1
Tomek	dobrze	0,700638732967303	0,680700211601176	1
Tomek	dobrze	0,70933095850863	0,680700211601176	1
Tomek	dobrze	0,687438847988748	0,680700211601176	1
Tomek	dobrze	0,698076324780188	0,669668712526206	1
Tomek	dobrze	0,705010523143408	0,671291209714838	1
mati	dobrze	0,669795497897217	0,680700211601176	0
Barbara	dobrze	0,658966954301156	0,688753385670819	0
Barbara	dobrze	0,654652997880535	0,688753385670819	0
Barbara	dobrze	0,662764782624701	0,688753385670819	0
Michał	dobrze	0,590557966074828	0,688753385670819	0
Michał	dobrze	0,705676538830014	0,688753385670819	1
Michał	dobrze	0,696796776357132	0,688753385670819	1
Michał	dobrze	0,689763275635892	0,688753385670819	1
Michał	dobrze	0,709060705005531	0,688753385670819	1

Z tabeli widać, że błędna akceptacja nie ma miejsca. Wszystkie próbki dla użytkownika Tomek zostały poprawnie rozpoznane. Algorytm spisał się również w przypadku użytkowników Mateusz i Barbara. Niestety użytkownik Michał został rozpoznany poprawnie w przypadku 80% próbek. Fakt ten można wytłumaczyć analizą tylko 1 słowa oraz tym, że użytkownicy Tomek oraz Michał są braćmi.

5.3.2. Test wyrażenia

Testy wyrażenia polegał na uczeniu i testowaniu sieci za pomocą określonego wyrażenia. Wyrażenie zostało zaczerpnięte z powieści "Qvo Vadis" Henryka Sienkiewicza:

> "Petroniusz obudził się zaledwie koło południa i jak zwykle, zmęczony bardzo. Poprzedniego dnia był na uczcie u Nerona, która przeciągnęła się do późna w noc. Od pewnego czasu zdrowie jego zaczęło się psuć. Sam mówił, że rankami budzi się jakby zdrętwiały i bez możności zebrania myśli."

Wyniki rozpoznania zostały zebrane z tabeli poniżej.

Tablica 5.2: Test wyrażenia dla użytkownika "Tomek", testowana Róża

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
ppp	zwykle	0,681575511234455	0,683971823046051	0
ppp	bardzo	0,666593576164478	0,671216634120907	0
ppp	do	0,674795457581491	0,643704200376147	1
ppp	jakby	0,65701919288151	0,681876519737353	0
ppp	bez	0,536775059291739	0,653217752143136	0
ppp	jak	0,641764162768517	0,677165183321002	0
ppp	do	0,69415200700499	0,674795457581491	1
ppp	jak, zwykle	0,64546900079993	0,680191751907232	0
ppp	do	0,698107337265308	0,674795457581491	1
ppp	jak, zwykle	0,697527242122138	0,680191751907232	1
ppp	do	0,592528364314161	0,674795457581491	0

Użytkownik ppp to osoba o imieniu Róża (5.3). Błędna akceptacja miała miejsce 4 razy na 11 możliwych. Co daje wskaźnik równy 36,36%.

Tablica 5.3: Test wyrażenia dla użytkownika “Tomek”, testowany Mateusz

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
mati	jak, zwykle	0,642998452529469	0,680191751907232	0
mati	bardzo	0,650850586324403	0,671216634120907	0
mati	do	0,631785830568695	0,674795457581491	0
mati	czasu, jego	0,638015539218089	0,655960197944712	0
mati	jakby	0,672413083563011	0,681876519737353	0
mati	jak, zwykle	0,678394791212021	0,680191751907232	0
mati	do	0,731056228970242	0,674795457581491	1
mati	czasu, jego	0,685200522710073	0,655960197944712	1
mati	jakby	0,663560395591826	0,681876519737353	0
mati	ci	0,672132331430161	0,668691787789757	1
mati	jak, zwykle	0,664998711609274	0,680191751907232	0
mati	bardzo	0,646375009607431	0,671216634120907	0
mati	do	0,724772100993518	0,674795457581491	1
mati	czasu, jego	0,675475937083523	0,655960197944712	1
mati	jakby	0,660025817129808	0,681876519737353	0
mati	ci	0,71022432226886	0,668691787789757	1
mati	jak, zwykle	0,650850586324403	0,680191751907232	0
mati	bardzo	0,629855272898158	0,671216634120907	0
mati	do	0,594358895833971	0,674795457581491	0
mati	czasu, jego	0,674562343890503	0,655960197944712	1
mati	jakby	0,64892772889309	0,681876519737353	0
mati	ci	0,583509668145917	0,668691787789757	0
mati	jak, zwykle	0,64100529518558	0,680191751907232	0
mati	bardzo	0,696188532984125	0,671216634120907	1
mati	do	0,611080327827585	0,674795457581491	0
mati	jakby	0,658095905417359	0,681876519737353	0
mati	ci	0,691678529590762	0,668691787789757	1
mati	jak, zwykle	0,669944076894775	0,680191751907232	0
mati	do	0,605936092529079	0,674795457581491	0
mati	czasu, jego	0,692276068342757	0,655960197944712	1
mati	jakby	0,655122090939399	0,681876519737353	0
mati	ci	0,600277318999578	0,668691787789757	0
mati	jak, zwykle	0,694042622105814	0,680191751907232	1
mati	bardzo	0,643791230022094	0,671216634120907	0
mati	do	0,689403960676676	0,674795457581491	1
mati	czasu, jego	0,648370757304899	0,655960197944712	0
mati	ci	0,694793495748865	0,668691787789757	1

Dla użytkownika “Mateusz“ (login mati) błędna akceptacja miała miejsce 13 razy na 36 możliwych próbek. Daje to wartość tego wskaźnika na poziomie 36,11%. Oczywiście przy zachowaniu błędnego odrzucenia na poziomie 0%.

Tablica 5.4: Test wyrażenia dla użytkownika “Tomek”, testowany Michał

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
michal	jak, zwykle	0,671018414877954	0,680191751907232	0
michal	bardzo	0,673176415561541	0,671216634120907	1
michal	do	0,666360043751978	0,674795457581491	0
michal	pewnego, czasu, jego	0,614839333806527	0,669438774499521	0
michal	jakby	0,653149114472294	0,681876519737353	0
michal	ci	0,71717525461996	0,668691787789757	1
michal	jak, zwykle	0,672378337397577	0,680191751907232	0
michal	do	0,684168516055106	0,674795457581491	1
michal	czasu, jego	0,698763395767224	0,655960197944712	1
michal	jakby	0,659006951771393	0,681876519737353	0
michal	ci	0,672014994553323	0,668691787789757	1
michal	jak, zwykle	0,651411868468487	0,680191751907232	0
michal	bardzo, dnia	0,648552256398094	0,671216634120907	0
michal	do	0,69240072076381	0,674795457581491	1
michal	zdrowie, jego	0,673523297949005	0,655960197944712	1
michal	jakby	0,647325509225035	0,681876519737353	0
michal	jak, zwykle	0,673605009578375	0,680191751907232	0
michal	bardzo	0,670396432273142	0,671216634120907	0
michal	do	0,730134862287422	0,674795457581491	1
michal	czasu, jego	0,60827387829854	0,669438774499521	0
michal	jakby	0,674612176658215	0,681876519737353	0
michal	jak, zwykle	0,686588738238349	0,680191751907232	1
michal	bardzo, dnia	0,665630072812301	0,671216634120907	0
michal	do	0,688633703462158	0,674795457581491	1
michal	jakby	0,623661832376293	0,681876519737353	0
michal	ci	0,678676177125106	0,668691787789757	1
michal	jak, zwykle	0,669377726490317	0,680191751907232	0
michal	bardzo, dnia	0,659925371291294	0,671216634120907	0
michal	do	0,672233394609733	0,674795457581491	0
michal	pewnego, czasu, zdrowie, jego	0,676035081853127	0,655960197944712	1
michal	jakby	0,686271153061426	0,681876519737353	1
michal	jak, zwykle, bardzo, dnia	0,647692594063673	0,677200045978457	0
michal	do	0,658048313702432	0,674795457581491	0
michal	pewnego, czasu, zdrowie	0,626998814413155	0,669438774499521	0
michal	jakby	0,645025153595846	0,681876519737353	0
michal	ci	0,702422298834432	0,668691787789757	1
michal	jak, zwykle	0,638779713822092	0,680191751907232	0
michal	bardzo	0,670393093830569	0,671216634120907	0
michal	do	0,708729603881942	0,674795457581491	1
michal	pewnego, czasu, zdrowie, jego	0,616561468173224	0,655960197944712	0
michal	jakby	0,712498968967272	0,681876519737353	1
michal	jak, zwykle	0,693172806577351	0,680191751907232	1
michal	bardzo, dnia	0,667739893144823	0,671216634120907	0

Tablica 5.5: Test wyrażenia dla użytkownika “Tomek”, testowany Michał c.d.

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
michal	do	0,623104869682814	0,674795457581491	0
michal	pewnego, czasu, zdrowie, jego	0,598403620479458	0,655960197944712	0
michal	jakby	0,673363639707276	0,681876519737353	0
michal	ci	0,697162527238399	0,668691787789757	1
michal	zwykle	0,679043873626341	0,683971823046051	0
michal	na	0,683829808917797	0,650268496724177	1
michal	do	0,677245344126277	0,674795457581491	1
michal	na	0,709444823874547	0,650268496724177	1
michal	pewnego, czasu, zdrowie	0,648458344835157	0,669438774499521	0
michal	jakby	0,6531236622889	0,681876519737353	0
michal	ci	0,67453380845341	0,668691787789757	1
michal	jak	0,702969283853562	0,677165183321002	1
michal	bardzo	0,676468459742784	0,671216634120907	1
michal	do	0,713724102529542	0,674795457581491	1
michal	pewnego, czasu, zdrowie, jego	0,628051756345063	0,655960197944712	0
michal	jakby	0,64021029895032	0,681876519737353	0
michal	ci	0,697162527238399	0,668691787789757	1
michal	ja, zwykle	0,686428778032796	0,683971823046051	1
michal	bardzo	0,663497057775626	0,671216634120907	0
michal	do	0,699322093247473	0,674795457581491	1
michal	jakby	0,651402185967087	0,681876519737353	0
michal	ci	0,720291485160942	0,668691787789757	1
michal	jak, zwykle	0,667443365468076	0,680191751907232	0
michal	bardzo	0,663777647660506	0,671216634120907	0
michal	do	0,694396047564244	0,674795457581491	1
michal	pewnego, czasu, zdrowie, jego	0,594386092973969	0,655960197944712	0

Login michal odpowiada użytkownikowi Michał. Błędne rozpoznanie wystąpiło 30 razy na 79 możliwych. Daje to dosyć wysoki wskaźnik 37,97%. Jest to zarazem najwyższy z uzyskanych wskaźników błędnego rozpoznania.

Z obserwacji widać, że próg jest tak wyliczany, aby zapewnić błędne odrzucenie na poziomie 0%. Jak pokazują przeprowadzone testy zapewnia to wartość błędnej akceptacji na poziomie 30-40% dla badanego wyrażenia. Są to jednak przypadki, dla których próbki testowe były maksymalnie 4 wyrazowe.

Pełny test wyrażenia powinien zawierać w próbce słowa występujące w badanym wyrażeniu i w słowniku najczęściej używanych słów. Są to następujące słowa: 'koło', 'południa', 'jak', 'zwykle', 'bardzo', 'był', 'na', 'do', 'noc', 'pewnego', 'czasu', 'zdrowie', 'jego', 'mówił', 'że', 'budzi', 'jakby', 'bez', 'zebrania', 'myśli'.

Pełne testy dla poszczególnych osób są przedstawione w tabelach poniżej. Przyjmując pewną minimalną ilość słów w próbce można zwiększyć skuteczność algorytmu. Taki próg został przyjęty i wynosi 5.

Tablica 5.6: Pełny test wyrażenia użytkownika "Mateusz"

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
mati	Petroniusz...	0,670012585035118	0,695007205034227	0
mati	Petroniusz...	0,677645942344947	0,694419311448014	0
mati	Petroniusz...	0,675059089721863	0,695703739776509	0
mati	Petroniusz...	0,671340711488431	0,696003905410513	0
mati	Petroniusz...	0,665120072455324	0,6956583550133	0
mati	Petroniusz...	0,664394300435682	0,695477558000813	0
mati	Petroniusz...	0,679179054334261	0,695785761271105	0
mati	Petroniusz...	0,662372614024271	0,692001443168056	0
mati	Petroniusz...	0,659786770477517	0,691064617991576	0
mati	Petroniusz...	0,666631773418179	0,691064617991576	0
mati	Petroniusz...	0,665691725085666	0,691907854957518	0
mati	Petroniusz...	0,662318514082772	0,692488749689671	0
mati	Petroniusz...	0,667917258505982	0,69149511297204	0
mati	Petroniusz...	0,68221808932796	0,692414167726784	0
mati	Petroniusz...	0,67125823128083	0,679412375221532	0
mati	Petroniusz...	0,680643045900868	0,679737183805998	1
mati	Petroniusz...	0,680756984305064	0,681728098972192	0
mati	Petroniusz...	0,673856735835254	0,684173593129482	0
mati	Petroniusz...	0,670186351209159	0,680425351979802	0
mati	Petroniusz...	0,666572712103677	0,684922054509238	0
mati	Petroniusz...	0,680946049405259	0,686779503170452	0

Tablica 5.7: Pełny test wyrażenia użytkownika “Michał”

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
michal	Petroniusz...	0,678063352400129	0,693060584276181	0
michal	Petroniusz...	0,676456391973032	0,69214954307227	0
michal	Petroniusz...	0,666047542641455	0,691907854957518	0
michal	Petroniusz...	0,67653194808846	0,691979670684282	0
michal	Petroniusz...	0,662831424962182	0,68873081657235	0
michal	Petroniusz...	0,664972357912963	0,693163136558354	0
michal	Petroniusz...	0,665198924428522	0,691979670684282	0
michal	Petroniusz...	0,669196531186842	0,691907854957518	0
michal	Petroniusz...	0,667545009705093	0,691907854957518	0
michal	Petroniusz...	0,674887441057946	0,69796488524157	0
michal	Petroniusz...	0,658646416390226	0,691907854957518	0
michal	Petroniusz...	0,663569504151782	0,695444252590613	0
michal	Petroniusz...	0,675145966050459	0,692488749689671	0
michal	Petroniusz...	0,66445708323084	0,691907854957518	0
michal	Petroniusz...	0,681738393051631	0,693446905701537	0
michal	Petroniusz...	0,678063352400129	0,702113839046099	0
michal	Petroniusz...	0,676456391973032	0,6932724118674	0
michal	Petroniusz...	0,666047542641455	0,695703739776509	0
michal	Petroniusz...	0,67653194808846	0,696365788357978	0
michal	Petroniusz...	0,662831424962182	0,693101511891357	0
michal	Petroniusz...	0,664972357912963	0,696689117313059	0
michal	Petroniusz...	0,665198924428522	0,696365788357978	0
michal	Petroniusz...	0,669196531186842	0,695703739776509	0
michal	Petroniusz...	0,667545009705093	0,695703739776509	0
michal	Petroniusz...	0,674887441057946	0,698630909988351	0
michal	Petroniusz...	0,658646416390226	0,695703739776509	0
michal	Petroniusz...	0,663569504151782	0,696052096891462	0
michal	Petroniusz...	0,675145966050459	0,695743332796207	0
michal	Petroniusz...	0,66445708323084	0,695703739776509	0
michal	Petroniusz...	0,652049240604845	0,696457842901558	0
michal	Petroniusz...	0,668629921571171	0,695785761271105	0
michal	Petroniusz...	0,671399981795519	0,695565294606928	0
michal	Petroniusz...	0,666196003340571	0,695703739776509	0
michal	Petroniusz...	0,678499823343428	0,695703739776509	0
michal	Petroniusz...	0,672886442903181	0,680299796338522	0
michal	Petroniusz...	0,676241865568573	0,688827357251194	0
michal	Petroniusz...	0,671142602435266	0,682697365575958	0
michal	Petroniusz...	0,668752315391191	0,681728098972192	0
michal	Petroniusz...	0,680217826023317	0,680184318515373	1
michal	Petroniusz...	0,66603327354525	0,682101375810111	0
michal	Petroniusz...	0,659833602565502	0,68350553058831	0
michal	Petroniusz...	0,667316396445448	0,680184318515373	0
michal	Petroniusz...	0,670164161967183	0,681728098972192	0

Tablica 5.8: Pełny test wyrażenia użytkownika “Michał”

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
michal	Petroniusz...	0,671650400217161	0,681728098972192	0
michal	Petroniusz...	0,674151637168826	0,681608808291121	0
michal	Petroniusz...	0,65781547629148	0,681728098972192	0
michal	Petroniusz...	0,665192341847773	0,680834849771217	0
michal	Petroniusz...	0,677134193336287	0,681715026237822	0
michal	Petroniusz...	0,66441034354028	0,681728098972192	0
michal	Petroniusz...	0,652099126040167	0,681519057802629	0
michal	Petroniusz...	0,670637711945473	0,686779503170452	0
michal	Petroniusz...	0,679064654133392	0,678355256236444	1
michal	Petroniusz...	0,667676870535958	0,681728098972192	0
michal	Petroniusz...	0,679720012744014	0,681728098972192	0

Tablica 5.9: Pełny test wyrażenia użytkownika “Róża”

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
ppp	Petroniusz...	0,677694644938481	0,691907854957518	0
ppp	Petroniusz...	0,67317893417421	0,691650183433102	0
ppp	Petroniusz...	0,661625574870613	0,693134930699063	0
ppp	Petroniusz...	0,690648808382312	0,69273528968859	0
ppp	Petroniusz...	0,660760446044988	0,685250897041216	0
ppp	Petroniusz...	0,670559527502605	0,696083752271669	0
ppp	Petroniusz...	0,680708229201878	0,695513526838439	0
ppp	Petroniusz...	0,678421086492602	0,695878899104933	0
ppp	Petroniusz...	0,674543341896506	0,695951497832195	0
ppp	Petroniusz...	0,695959884700716	0,695951497832195	1
ppp	Petroniusz...	0,660760446044988	0,685250897041216	0
ppp	Petroniusz...	0,670180769481134	0,682355908697161	0
ppp	Petroniusz...	0,683733891011747	0,681534049535166	1
ppp	Petroniusz...	0,681738811513159	0,682220030658108	0
ppp	Petroniusz...	0,6769475078889	0,681659791073698	0
ppp	Petroniusz...	0,698254944757399	0,682816027147458	0

Po przeprowadzeniu badań wyniki zostały zliczone. Dla każdego użytkownika została określona wartość błędnej akceptacji, czyli uznanie go za użytkownika “Tomek”.

Tablica 5.10: Wartości błędnej akceptacji dla poszczególnych użytkowników

Osoba	poziom błędnej akceptacji
Mateusz	4,76%
Michał	2,33%
Róża	12,5%
Średnio	5%

Błędne odrzucenie praktycznie nie występowało podczas badań (domyślnie określone jest jako 0.01%). Związane jest to z automatycznym doborem progu. Aczkolwiek jest możliwa sytuacja, w której nastąpi błędne odrzucenie. Jest to, np. sytuacja drastycznej zmiany sposobu pisania wywołana zmianą nastroju.

5.3.3. Test wyrażenia 2

Został przeprowadzony dodatkowy test wyrażenia. Jego modyfikacja w stosunku do testu 5.3.2 polegała na zmianie algorytmu doboru progu rozpoznania. Próg liczony jest jako średnia wartość *n* ostatnich rozpoznań dla danego słowa. Są to warunki w przybliżeniu zapewniające minimalną wartość błędnej akceptacji. Idealne warunki minimalnej wartości błędnej akceptacji wymagałyby uczenia próbkami oznaczonymi jako niewłaściwe tj. próbkami osób obcych z wartością wyjścia równą 0.

Kilka przykładowych wartości rozpoznania i progu obrazuje poniższa tabelka.

Tablica 5.11: Pełny test wyrażenia nr. 2

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
mati	Petroniusz...	0,67125823128083	0,702568144205707	0
mati	Petroniusz...	0,680643045900868	0,70252469147505	0
mati	Petroniusz...	0,680756984305064	0,701742735646825	0
mati	Petroniusz...	0,673856735835254	0,702784825084765	0
mati	Petroniusz...	0,670186351209159	0,701897746788951	0
mati	Petroniusz...	0,666572712103677	0,702248181249811	0
mati	Petroniusz...	0,680946049405259	0,703574322953134	0
michal	Petroniusz...	0,672886442903181	0,702885276632546	0
michal	Petroniusz...	0,676241865568573	0,703247241453078	0
michal	Petroniusz...	0,671142602435266	0,702391158796817	0
michal	Petroniusz...	0,668752315391191	0,701742735646825	0
michal	Petroniusz...	0,680217826023317	0,70129326873756	0
michal	Petroniusz...	0,66603327354525	0,703196434909277	0
michal	Petroniusz...	0,659833602565502	0,702917989506277	0
michal	Petroniusz...	0,667316396445448	0,70129326873756	0
michal	Petroniusz...	0,670164161967183	0,701742735646825	0
michal	Petroniusz...	0,671650400217161	0,701742735646825	0
michal	Petroniusz...	0,674151637168826	0,702558078690708	0
michal	Petroniusz...	0,65781547629148	0,701742735646825	0
michal	Petroniusz...	0,665192341847773	0,701168872285956	0
michal	Petroniusz...	0,677134193336287	0,701708500385076	0
ppp	Petroniusz...	0,660760446044988	0,707173986927804	0
ppp	Petroniusz...	0,670180769481134	0,702256256482638	0
ppp	Petroniusz...	0,683733891011747	0,701434842983145	0
ppp	Petroniusz...	0,681738811513159	0,701886376585932	0
ppp	Petroniusz...	0,6769475078889	0,701732259231524	0
ppp	Petroniusz...	0,698254944757399	0,702477688747465	0

Jak widać wszystkie wartości są równe 0. Zatem wartość błędnej akceptacji wynosi 0% (w domyśle 0.01%).

Następnie zostały przeprowadzone testy na właścicielu sieci neuronowej.

Tablica 5.12: Pełny test wyrażenia nr.2 dla użytkowniak “Tomek”

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
Tomek	Petroniusz...	0,717992355790392	0,720146710977834	0
Tomek	Petroniusz...	0,672239221727601	0,701268844854864	0
Tomek	Petroniusz...	0,618042149017387	0,706541980340142	0
Tomek	Petroniusz...	0,695962846710586	0,709630402992796	0
Tomek	Petroniusz...	0,660346849859847	0,704088661450933	0
Tomek	Petroniusz...	0,706952701729606	0,705439211009904	1
Tomek	Petroniusz...	0,728881299217639	0,702812742908933	1
Tomek	Petroniusz...	0,708575564815814	0,697394645076035	1
Tomek	Petroniusz...	0,693322735105034	0,706898865633259	0
Tomek	Petroniusz...	0,676018134295856	0,702722073567923	0
Tomek	Petroniusz...	0,704365246646977	0,706304078375979	0
Tomek	Petroniusz...	0,704651236550655	0,71559003280648	0
Tomek	Petroniusz...	0,687161638963135	0,713621649609322	0
Tomek	Petroniusz...	0,682356608600338	0,691104751115926	0
Tomek	Petroniusz...	0,680611950043194	0,713587703489848	0
Tomek	Petroniusz...	0,672886442903181	0,702885276632546	0
Tomek	Petroniusz...	0,676241865568573	0,703247241453078	0
Tomek	Petroniusz...	0,671142602435266	0,702391158796817	0
Tomek	Petroniusz...	0,668752315391191	0,701742735646825	0
Tomek	Petroniusz...	0,680217826023317	0,70129326873756	0
Tomek	Petroniusz...	0,66603327354525	0,703196434909277	0
Tomek	Petroniusz...	0,659833602565502	0,702917989506277	0
Tomek	Petroniusz...	0,667316396445448	0,70129326873756	0
Tomek	Petroniusz...	0,670164161967183	0,701742735646825	0
Tomek	Petroniusz...	0,671650400217161	0,701742735646825	0
Tomek	Petroniusz...	0,675395475508336	0,701863581013111	0
Tomek	Petroniusz...	0,65781547629148	0,701742735646825	0
Tomek	Petroniusz...	0,665192341847773	0,701168872285956	0
Tomek	Petroniusz...	0,677134193336287	0,701708500385076	0
Tomek	Petroniusz...	0,66441034354028	0,701742735646825	0
Tomek	Petroniusz...	0,680168046237656	0,705623824462654	0
Tomek	Petroniusz...	0,701898596155263	0,70389088425284	0
Tomek	Petroniusz...	0,672886442903181	0,702885276632546	0
Tomek	Petroniusz...	0,676241865568573	0,703247241453078	0
Tomek	Petroniusz...	0,671142602435266	0,702391158796817	0
Tomek	Petroniusz...	0,668752315391191	0,701742735646825	0
Tomek	Petroniusz...	0,680217826023317	0,70129326873756	0
Tomek	Petroniusz...	0,66603327354525	0,703196434909277	0
Tomek	Petroniusz...	0,659833602565502	0,702917989506277	0
Tomek	Petroniusz...	0,667316396445448	0,70129326873756	0
Tomek	Petroniusz...	0,670164161967183	0,701742735646825	0
Tomek	Petroniusz...	0,671650400217161	0,701742735646825	0
Tomek	Petroniusz...	0,674151637168826	0,702558078690708	0

Tablica 5.13: Pełny test wyrażenia nr.2 dla użytkowniak “Tomek”

Osoba	słowa	rozpoznanie	próg	rozp. 0-1
Tomek	Petroniusz...	0,637594323342378	0,696083209481737	0
Tomek	Petroniusz...	0,663454879039698	0,703321100020475	0
Tomek	Petroniusz...	0,665192341847773	0,701168872285956	0
Tomek	Petroniusz...	0,677134193336287	0,701708500385076	0
Tomek	Petroniusz...	0,66441034354028	0,701742735646825	0
Tomek	Petroniusz...	0,672886442903181	0,702885276632546	0
Tomek	Petroniusz...	0,719967934894213	0,713852126218535	1
Tomek	Petroniusz...	0,671142602435266	0,702391158796817	0
Tomek	Petroniusz...	0,668752315391191	0,701742735646825	0
Tomek	Petroniusz...	0,680217826023317	0,70129326873756	0
Tomek	Petroniusz...	0,66603327354525	0,713852126218535	0

Niestety ustawienie progu zbyt wysoko powoduje bardzo niską efektywność algorytmu. Co pokazuje powyższa tabelka. Wartość błędnego odrzucenia wynosi 92,59%. Co jest wynikiem nieakceptowalnym.

5.3.4. Test w środowisku Windows

Test w środowisku Windows polegał na uruchamianiu aplikacji w tle podczas codziennej pracy na komputerze. Podczas takiej pracy została zebrana baza danych z dobrze nauczoną siecią neuronową. Tak nauczoną sieć można wykorzystywać w pracy na komputerze w celu zapewnienia sobie dodatkowego bezpieczeństwa komputera.

5.4. Możliwości rozszerzeń

- badanie dłuższych przerw między wprowadzanymi wyrazami,
- badanie przerw w trakcie wprowadzania wyrazu, czyli zatrzymań,
- badanie gwałtownych działań myszki zaraz po dłuższej bezczynności,
- stworzenie wersji działającej pod systemem operacyjnych Windows oraz Linux z wykorzystaniem, np. biblioteki QT do interfejsu,
- alternatywne rozwiązanie: stworzenie wersji działającej przez Internet,
- dopracowanie parametrów z pliku *parameters.h* (5.1).

5.5. Wnioski

Na podstawie przeprowadzonych badań oraz obserwacji dokonanych podczas tworzenia aplikacji można wysunąć kilka wniosków. Zaprezentowany algorytm z wykorzystaniem sieci neuronowych działa poprawnie. Można stworzyć aplikację zwiększającą bezpieczeństwo systemu komputerowego. Podczas tworzenia takiej aplikacji warto pamiętać o istotnym wyborze pomiędzy wygodą, a poziomem bezpieczeństwa (błędną akceptacją intruzów). Wybór ten przekłada się bezpośrednio na politykę doboru progu rozpoznania oraz minimalnej ilości słów w próbce.

Podczas tworzenia takiej aplikacji należy pamiętać o niskiej unikalności i odporności na zakłócenia próbek zebranych podczas pisania na klawiaturze oraz operowania myszką (tabela 3.1). W związku z tym nie należy oczekiwać dużych różnic pomiędzy niektórymi osobami, szczególnie pomiędzy rodzeństwem.

Praca doprowadziła także do utwierdzenia faktu, iż sieci neuronowe są efektywniejsze niż wektorowa analiza matematyczna. W szczególności sieci neuronowe typu RBFN są jednymi z prostszych sprawdzających się w problemach rozpoznawania biometrycznego.

Kolejnym atutem jest prostota działania oraz łatwy sposób zbierania próbek. Panel sterowania zawiera kilka opcji. Pozostałe parametry są dostępne z poziomu kodu, dla zaawansowanych użytkowników. Zbieranie próbek nie stwarza problemu, nie jest wymagany dodatkowy sprzęt. Całość badań można przeprowadzić na jednym komputerze (laptopie).

6. Zakończenie

> "Ludzie uczą się w 25 procentach od mistrza, w 25 procentach słuchając samych siebie, w 25 procentach od przyjaciół, a w 25 procentach uczy ich czas."
> [Paulo Coelho - Czarownica z Portobello]

Praca magisterska pokazuje praktyczne wykorzystanie sztucznych sieci neuronowych do rozwiązania problemu rozpoznawania osób. Najważniejszym osiągnięciem jest uzyskanie błędu akceptacji (False Acceptance Rate - FAR) na poziomie 5%. Błędne odrzucenie (False Rejection Rate - FRR) praktycznie nie występowało, gdy sieć już była dobrze nauczona. Wynik jest porównywalny z innymi pracami wykorzystującymi sztuczne sieci neuronowe.

Aplikacja nie działa nadzwyczaj szybko. Nie jest jednak wymagany rzeczywisty czas obliczeń. Nauka sieci neuronowej jest najwolniejszym etapem. W przypadku dużych próbek może trwać nawet pół minuty. Rozpoznanie, które jest przepropagowaniem sygnału przez sieć neuronową, trwa nie dłużej niż sekundę. Co w systemach tego typu jest wynikiem przyzwoitym.

Zastosowana sieć neuronowa została napisana od podstaw. Co pozwala, w przyszłości, na dowolną jej modyfikację. Najważniejszym jej elementem jest optymalizacja pod względem zajmowanego miejsca w pamięci RAM. W pamięci zaalokowane są tylko te podsieci, które są używane tzn. podsieci dla słów, które są testowane lub uczone.

Opracowany algorytm pokazuje przydatność sieci neuronowych do rozwiązywania złożonych problemów. Stosowanie dodatkowych urozmaiceń, np. podsieci zbudowanych ze słów (zastosowanych w tej pracy magisterskiej), zwiększa skuteczność algorytmu. Niektóre modyfikacje algorytmu mogą prowadzić do błędnego działania. Wyniki zachęcają jednak do podejmowania takich prób i tym samym do stopniowego rozwoju systemów bezpieczeństwa.

Praca przedstawia kolejne podejście do problemu autentykacji użytkowników. Biometria jest rozwijającą się dziedziną jednak dynamika pisania wydaje się być zamkniętym zbiorem rozwiązań. Zauważalna jest tendencja do patentowania rozwiązań tego problemu. Świadczy to o sporej wartości takich elementów systemów bezpieczeństwa.

Rozwój biometrii zapewnia nam coraz większe bezpieczeństwo. Ale czy nie powoduje zarazem coraz większego utrudnienia? Twórcy dziś powstających systemów starają się jak najbardziej je ukryć. Tak, aby użytkownik nie wiedział o ich istnieniu. Tak, aby nie musiał podejmować dodatkowych działań zapewniających bezpieczeństwo, np. podanie hasła, przyłożenie dłoni do czytnika. Myślę, że rozwój systemów pójdzie w tym kierunku i doprowadzi sztukę kamuflażu do perfekcji. Program, który powstał w wyniku tej pracy magisterskiej stara się podążać za tymi trendami. Skromny interfejs, minimalna ilość opcji dostępnych dla użytkownika, możliwość schowania do paska stanu oraz możliwość wyłączenia pasków postępu uczenia zapewniają małą widoczność dla użytkownika.

Podsumowując, założenia postawione na wstępie pracy zostały zrealizowane. Algorytm jest na tyle skuteczny, aby wykorzystywać go w codziennej pracy przy komputerze. Istnieje spora ilość algorytmów służących do rozpoznawania osób. Najstarsze z nich, np. odcisk palca, rozpoznawanie twarzy, zaczynają być widoczne w życiu codziennym, szczególnie w notebookach. Badane są coraz bardziej wyszukane cechy biometryczne, np. zapach, temperatura twarzy lub kanalki uszne. Sądzę, iż w przyszłości będziemy otoczeni przez różnego rodzaju systemy wykorzystujące po kilka algorytmów równocześnie. Tego rodzaju symbioza pomiędzy metodami rozpoznawania zapewni najwyższy poziom bezpieczeństwa.

Spis rysunków

Spis tablic

A. Dodatek B: Zawartość płyty CD-ROM

Dołączona do pracy płyta CD-ROM zawiera następujace katalogi:

- Praca magisterska - LaTeX – katalog zawierajacy tekst pracy magisterskiej w formacie LATEX
  - images – wykorzystane w pracy obrazy.
- Praca magisterska - PDF – katalog zawierajacy tekst pracy magisterskiej w formacie PDF
- Zrealizowany projekt – katalog zawierajacy stworzona aplikację
  - Aplikacja – katalog zawierający skompilowana wersję projektu wraz ze wszystkimi koniecznymi do uruchomienia plikami,
  - Źródła - kod źródłowy w postaci projektu dla Borland C++ Builder,
- Materiały - instalki,
  - Firebird-1.5.3.4870-0-Win32.exe - server bazy danych firebird,
  - ibep_2006.1.29.1_full.exe - program IBExpert do zarządzania bazą danych firebird,
  - ibxbcb504upd.exe - dodatek do programu Borland C++ Builder, zawierający klasy TIBQuery, TIBTransaction, TIBDatabase...
- Materiały - PDF,
  - firebird - pdfy dotyczące bazy danych firebird,
  - inne prace - pdfy z innymi pracami na temat dynamiki pisania.

Bibliografia

[1] J. C. Checco. Keystroke dynamics and corporate security.

[2] S. T. de Magalh'es, K. Revett, and H. M. D. Santos. Password secured sites - stepping forward with keystroke dynamics. 2005.

[3] M. Garcia. Biometric authentication tool for user identifiacation. 2006.

[4] http://en.wikipedia.org/wiki/Biometrics. Wikipedia.

[5] http://msdn.microsoft.com/en us/library/aa376868(VS.85).aspx. Microsoft.

[6] http://msdn.microsoft.com/en us/library/ms632589(VS.85).aspx. Microsoft.

[7] http://pl.wikipedia.org/wiki/Neuron. Wikipedia.

[8] http://pl.wikipedia.org/wiki/Neuron_McCullocha Pittsa. Wikipedia.

[9] http://www.firebirdsql.org/. Firebird.

[10] S. Kochański. Praktyczne wykorzystanie sztucznych sieci neuronowych - implementacja algorytmu rozpoznającego pismo. 2005.

[11] M. S. Obaidat and B. Sadoun. Keystroke dynamics based authentication.

[12] prof. dr hab. Elżbieta Richter-Wąs. Wykład - sieci neuronowe.

[13] prof. Lubaczewski. słownik frekwencyjny języka polskiego.

[14] R. Tadeusiewicz. *Biometria*. Kraków : Wydaw. AGH., 1993.