Politechnika Łódzka

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

Instytut Informatyki
Kierunek: Informatyka
Specjalność: Sztuczna Inteligencja i Inżynieria Oprogramowania

inż. Michał Sośnicki
nr albumu 207597

Rozpoznawanie mówcy zależne od tekstu z wykorzystaniem sieci neuronowych

Praca magisterska
napisana pod kierunkiem
dr inż. Bartłomieja Stasiaka

Łódź 2018

Spis treści

Rozdział 1

Wstęp

Zakresem niniejszej pracy magisterskiej są systemy rozpoznawania mówcy zależne od tekstu. Celem pracy jest zbadanie możliwości wykorzystania modelu sieci neuronowej do rozwiązania tego problemu.

Rozpoznawanie mówcy polega na analizie nagrań dźwiękowych mowy w celu wydobycia cech biometrycznych charakteryzujących osobę mówiącą. Wyróżnia się dwa praktyczne problemy do rozwiązania z wykorzystaniem tych cech: problem weryfikacji mówcy oraz problem identyfikacji mówcy.

Weryfikacja mówcy polega na stwierdzeniu, czy nagranie pochodzi od pewnej zarejestrowanej wcześniej osoby lub wykryciu, że jest na nim ktoś inny. Ten przypadek występuje na przykład przy uwierzytelnianiu za pomocą głosu do systemu bankowego. Konieczne jest wtedy wykrycie, gdy oszust podaje się za prawdziwego właściciela konta. Identyfikacja mówcy polega na rozpoznaniu, która z zarejestrowanych osób jest na nagraniu lub czy jest na nim ktoś niezarejestrowany. Identyfikacja może być wykorzystana przez służby policyjne do zidentyfikowania osób na zdobytym nagraniu.

Rozpoznawanie mówcy można również podzielić według kryterium, czy treść nagrania jest znana z góry, na rozpoznawanie mówcy zależne od tekstu i niezależne od tekstu. Weryfikacja mówcy zwykle może być przeprowadzona zależnie od tekstu, gdyż w typowych zastosowaniach można oczekiwać od weryfikowanej osoby kooperacji. Znajomość treści jest cenna, gdyż pozwala zwiększyć skuteczność, szczególnie w sytuacji, gdy nagrań rejestrujących jest niewiele[12]. W przypadku, gdy nagranie nie zawiera oczekiwanej treści, wymagane jest odrzucenie weryfikowanego nagrania niezależnie od tożsamości mówcy. Identyfikację z kolei zwykle trzeba przeprowadzać bez założeń co do treści nagrania, gdyż nagrywane osoby mogą być tego nieświadome.

Jako że przedmiotem pracy jest rozpoznawanie mówcy zależne od tekstu, uwaga zostanie poświęcona problemowi weryfikacji, a nie identyfikacji mówcy. Ten problem jest również ciekawy ze względu na potencjalne zastosowania, przypuszczalnie większe niż problem identyfikacji. Zaletą bazowania na nagraniu mowy jest możliwość jego nieinwazyjnego pozyskania. Jednakże niska skuteczność rozpoznawania sprawiała, że takie systemy pełnią tylko rolę drugiej warstwy zabezpieczeń, jako uzupełnienie tradycyjnego hasła.

Systemy bazujące na sieciach neuronowych osiągnęły w ostatnim czasie znakomite wyniki w problemach związanych z przetwarzaniem obrazów, dźwięków czy dokumentów tekstowych. Rekurencyjne sieci neuronowe leżą u podstaw najnowszych systemów do rozpoznawania mowy. Można się spodziewać, iż mogą one osiągnąć również dobre wyniki przy rozpoznawaniu mówcy. Rzeczywiście, miały już miejsce próby wykorzystania głębokich sieci neuronowych. Na przykład zastąpiono nimi `GMM` (*gaussian mixture model*) w celu rozpoznania jaki fonem znajduje się w ramce nagrania, a wektor aktywacji warstwy ukrytej użyto jako tzw. *bottleneck features*[20]. Korporacje takie jak Google również zgromadziły zbiory danych i zaproponowały architekturę bazującą na sieciach neuronowych[6]. Celem tej pracy jest zbadanie tematu i wypróbowanie innego sposobu zaaplikowania sieci neuronowych.

Zainteresowanie tym tematem i potencjalne zyski, które mogą z niego płynąć, sprawiło niestety, że zbiory danych dopasowane do badania tego problemu są albo chronione przez przedsiębiorców, albo dostępne, lecz za opłatą. Na szczęście w 2015 zajęli się tym ograniczeniem naukowcy między innymi z Singapuru. Zgromadzili oni nagrania o znanej treści od ochotników z całego świata i opublikowali zbiór `RedDots`. Trzeba jednak przyznać, że zasoby posiadane przez korporacje w postaci danych i pracowników stawiają ich w pozycji, w której trudno im dorównać. Tym niemniej możliwe jest zbadanie, czy wybrany pomysł jest warty uwagi i w jakich sytuacjach, oraz porównanie go z wybranymi systemami *baseline*.

1.1 Cele pracy

Celem pracy jest stworzenie systemu do rozpoznawania mówcy wykorzystującego sieć neuronową. Zostanie on przetestowany na zbiorze `RedDots` i porównany z innymi stworzonymi systemami. Wyniki pozwolą ocenić sensowność używania wybranej metody oraz wskazać, gdzie może sprawdzić się lepiej od innych metod.

Zostanie stworzony jeden system bazujący na sieciach neuronowych oraz dwa wykorzystujące inne metody. Przetestowane modele to:

- System bazujący na miksturach wielowymiarowych rozkładów normalnych i modelu tła. (`GMM-UBM`, *Gaussian Mixture Model - Universal Background Model*)

- System wykorzystujący ukryty model Markowa z emisjami będącymi miksturami wielowymiarowych rozkładów normalnych. (`HMM-GMM`, *Hidden Markov Model - Gaussian Mixture Model*)

- System wykorzystujący rekurencyjną sieć neuronową do dopasowania ramek do fonemów, a następnie modelujący charakterystyczny sposób generowania określonych fonemów przez mówców za pomocą mikstur wielowymiarowych rozkładów normalnych. (`DNN-GMM`, *Deep Neural Network - Gaussian Mixture Model*)

Modele zostaną przetestowane na zbiorze `RedDots` i porównane między sobą oraz z wynikami innych prac. Osobno zostanie zaprezentowana skuteczność weryfikacji mówcy oraz treści. Uwzględnione zostaną następujące miary jakości:

- Krzywa `ROC` (*Receiver Operating Characteristic*)
- `EER` (*Equal Error Rate*)
- `AUC` (*Area Under Curve*)

1.2 Przegląd literatury

Podstawy teorii rozpoznawania mowy są dobrze opisane w [15] oraz [14] tego samego autora, które skupiają się na praktycznym wprowadzeniu do ukrytych modeli Markowa, użytych w tej pracy. Innym bardzo praktycznym zasobem na ten temat jest [21], opisująca rozwiązanie tego problemu zastosowane w pakiecie *Hidden Markov Model Toolkit* i publicznie dostępne zasoby kursu *CS 224S Spoken Language Processing* z Uniwersytetu Stanford.

O rozpoznawaniu mówcy traktuje [1] Jest to przydatny zasób, jednakże bardziej pomocne może okazać przejrzenie się mniejszych, lecz bardziej aktualnych publikacji na ten temat.

Ogólnie tematy związane z uczeniem maszynowym doskonale opisane są w [2]. Pozycja ta porusza takie tematy jak statystyka Bayesowska, *expectation maximization*, sieci neuronowe, RBM i *unsupervised learning*. Autor znalazł ten tekst jako przystępniejszy w odbiorze niż powszechnie uznany i polecany studentom *Pattern Recognition and Machine Learning* Christophera Bishopa. Doskonałym zasobem do głębokich sieci neuronowych jest dostępna w Internecie książka [4].

Godne polecenia są kursy internetowe dostępne za darmo na platformie Coursera. Wysoko oceniane, również przez autora, związane z tematem pracy kursy to *Machine Learning* od Andrew Ng, *Neural Networks for Machine Learning* od Geoffrey Hintona, *Digital Signal Processing* od Paolo Prandoniego i Martina Vetterliego. Są to zaadoptowane do formy kursu internetowego kursy akademickie. Pozwalają doświadczyć lekcji prowadzonych przez światowej sławy naukowców, niezależnie od miejsca zamieszkania, co jest bardzo cenne.

1.3 Układ pracy

Rozdział 1 zawiera wprowadzenie i określenie tematu oraz celu pracy. Rozdział 2 zawiera opis kilku zagadnień z zakresu rozpoznawania mówcy i rozpoznawania mowy, które zostały wybrane jako istotne dla tej pracy. Rozdział 3 opisuje zbiór danych, technologie i narzędzia wykorzystane w pracy. W rozdziale 4 przedstawiono opis stworzonych systemów i wyniki testów. Rozdział 5 zawiera porównanie wyników i podsumowanie czy założone cele zostały osiągnięte. W dodatku A znajduje się płyta CD z kodem źródłowym aplikacji oraz wersją elektroniczną tej pracy.

Rozdział 2

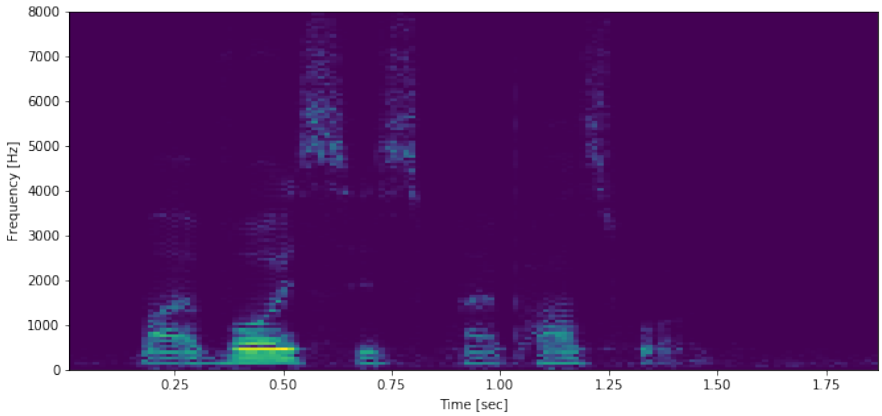
Teoria rozpoznawania mówcy

2.1 Przetwarzanie mowy

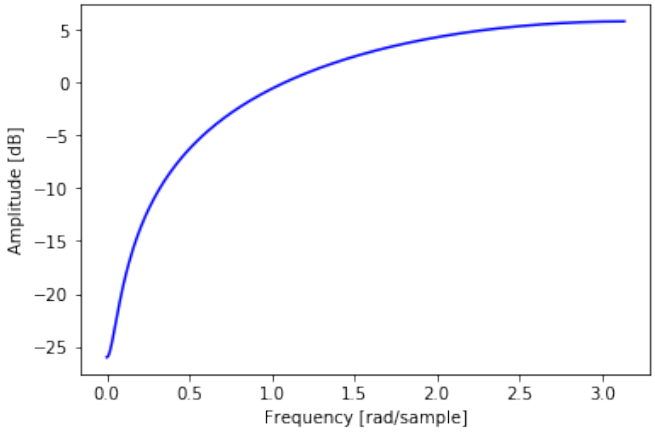
2.1.1 Współczynniki mel-cepstralne

Współczynniki mel-cepstralne (`MFCC`, *Mel Frequency Cepstral Coefficients*), obok blisko z nimi związanych *filter banks* stanowią popularny zestaw cech dla systemów rozpoznawania mowy i rozpoznawania mówcy. Systemy bazujące na głębokich sieciach neuronowych wykorzystują często nieprzetworzone dalej widmo sygnału (*spectrum*), gdyż obowiązuje w nich podejście, by zamiast ręcznie przetwarzać wejście lepiej zapewnić sieci neuronowej architekturę pozwalającą na nauczenie się odpowiednich cech. Na przykład zamiast ręcznie projektować filtr lepiej dodać do sieci warstwę konwolucyjną. Bardziej tradycyjne metody oparte o `GMM` jednak bardzo korzystają na zredukowaniu wymiarowości problemu oraz zapewnieniu, że poszczególne cechy są nieskorelowane, co zachodzi przy `MFCC`.

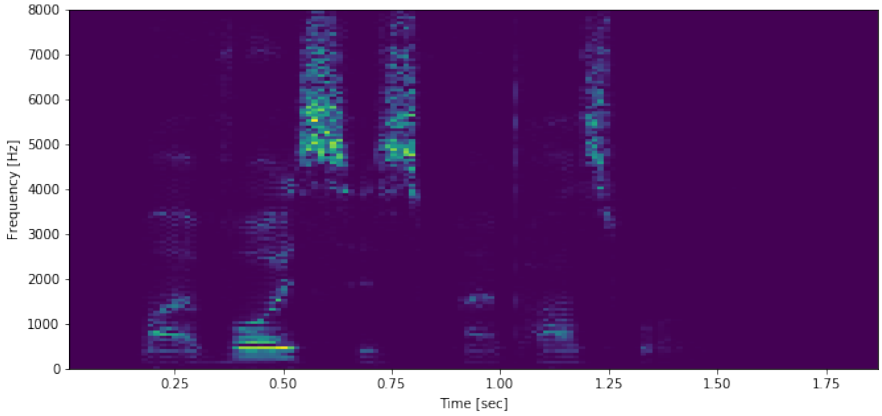
Krokiem wstępnym policzenia `MFCC` jest pre-emfaza (*preemphasis*), czyli przefiltrowanie sygnału prostym filtrem górnoprzepustowym, zwykle postaci wektora o współczynnikach $[-0.95\ 1.0]$ lub
$[-0.97\ 1.0]$. Jak pokazuje rysunek 2.3, energia w wyższych pasmach po tej operacji, choć początkowo znacznie niższa, została wyrównana z energią w niższych pasmach częstotliwości. Ogranicza to błędy numeryczne pojawiające się przy dalszym przetwarzaniu, co miało duże znaczenie w starszych komputerach.



Rysunek 2.1: Spektrogram sygnału `m0002/20150713085938321_m0002_31.pcm` ze zbioru `RedDots`



Rysunek 2.2: Charakterystyka częstotliwościowa filtru typu *preemphasis* o współczynnikach $[-0.95, 1.0]$



Rysunek 2.3: Spektrogram tego samego sygnału poddany *preemphasis*

W drugim kroku należy policzyć `STFT` (*Short Time Fourier Transform*), to znaczy podzielić sygnał na ramki i dla każdej ramki policzyć spektrum częstotliwościowe. Dalsze operacje będą wykonywane na reprezentacji sygnału w dziedzinie częstotliwości. Typowa szerokość okna to 25ms, przy czym każde kolejne okno jest przesunięte o ok. połowę szerokości, np. można przyjąć 10ms. W przypadku zbioru `RedDots` nagrania mają częstotliwość próbkowania 16kHz, więc by uzyskać takie przesunięcie okna muszą zawierać 400 próbek z przesunięciem 160 próbek. Następnie spektra zamieniane są w spektra mocy przez wzięcie kwadratu wartości absolutnej z każdej wartości w spektrum.

Wynik `STFT` można zwizualizować jako spektrogram, jak na rysunku 2.1, i zawiera on informacje o tym jak energia sygnału w różnych pasmach zmieniała się w czasie. Każde okno można wyciąć z użyciem okna Hamminga lub Hanna, które mają mniejszy wpływ na wynikowe spektrum sygnału niż okno prostokątne (wycięcie wybranego przedziału próbek z sygnału jest równoważne przemnożeniu przez okno prostokątne)

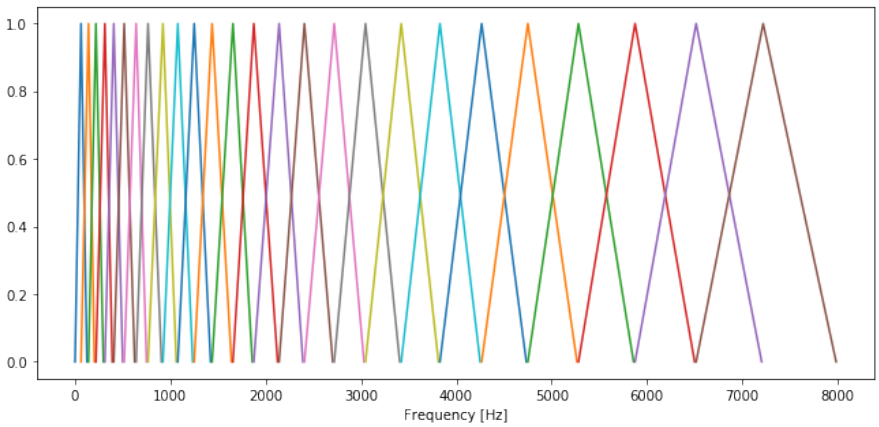
Następnie budowany jest bank filtrów w skali melowej. Przeliczenia częstotliwości w melach na częstotliwość w Hertzach dokonuje się według poniższych wzorów.

$$m = 2595 log_{10}(1 + \frac{f}{700})$$

$$f = 700(10^{m/2595} - 1)$$

Jest to skala w nieliniowej zależności z Hertzami. Jej postać i stałe zostały tak dobrane, że zmiana w tej skali odpowiada subiektywnemu odczuciu zmiany wysokości dźwięku przez ludzi.

Bank filtrów składa się z pewnej liczby, zwykle 20 lub 26, trójkątnych filtrów o równej szerokości w skali melowej, rozmieszczonych z przesunięciem 50% szerokości. Ich szerokość jest tak dobrana, by pokryły cały przedział częstotliwości sygnału.



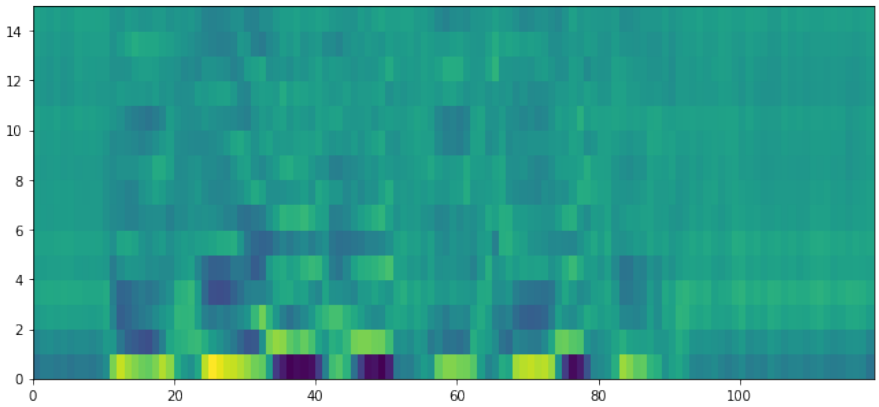
Rysunek 2.4: Charakterystyka w skali Hertza banku 20 trójkątnych filtrów o równej szerokości w skali melowej

Mając te filtry następnym etapem przetwarzania sygnału jest przemnożenie widma przez każdy z tych filtrów i zsumowanie dzięki czemu uzyskuje się jedną liczbę na filtr - całkowitą energię sygnału na pokrytych filtrami pasmach.

Uzyskane sumy są następnie logarytmowane. Pozwala to na dekonwolucję sygnału mowy i wpływu środowiska. Dekonwolucja przebiega w ten sposób, że najpierw dzięki policzeniu transformaty Fouriera, uzyskiwane jest spektrum, a konwolucja sygnału w czasie jest równoważna przemnożeniu widm częstotliwościowych. Następnie jak ten iloczyn widm zostanie zlogarytmowany, to można go przedstawić jako sumę logarytmów. W wynikowych cechach jeżeli sygnał mowy został z czymś spleciony, to skutkuje to tylko przesunięciem cech ze wszystkich ramek. Można wtedy, przy założeniu, że wpływ otoczenia jest stały, znormalizować średnią tych cech, by zmniejszyć jego wpływ.

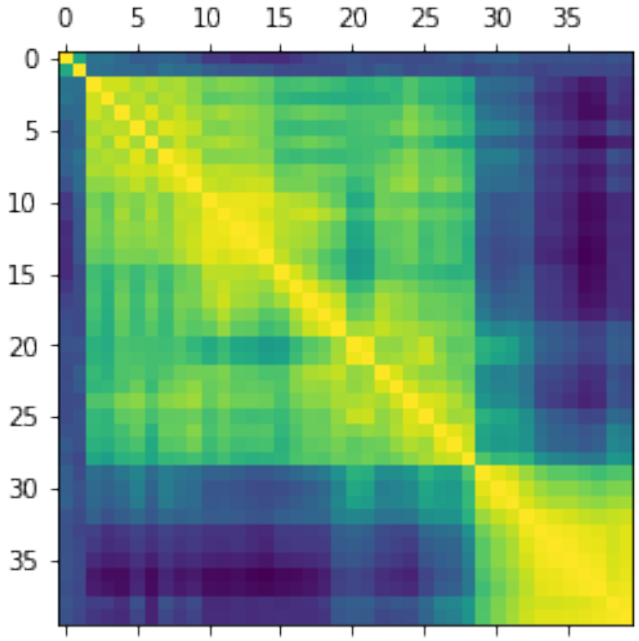
Znormalizowane współczynniki filtrów melowych bywają używane jako gotowy zestaw cech. Do uzyskania `MFCC`, widocznej na rysunku 2.5, potrzebne jest jeszcze policzenie dyskretnej transformaty cosinusowej (`DCT`, *Discrete Cosine Transform*). Dyskretna transformata cosinusowa to modyfikacja transformaty Fouriera, która w wyniku daje ciąg współczynników rzeczywistych, a nie zespolonych. Wykorzystana jest właściwość `DFT` (*Discrete Fourier Transform*), że jeżeli sygnał w dziedzinie czasu ma współczynniki rzeczywiste, to współczynniki transformaty będą cechować się symetrią sprzężoną. A jeżeli sygnał w dziedzinie czasu jest symetryczny, to transformata będzie miała tylko współczynniki rzeczywiste. `DCT` (*Discrete Cosine Transform*) liczy się zatem tak, iż tworzony jest nowy sygnał dwa razy większej długości niż sygnał wyjściowy, przez odbicie wyjściowego sygnału. Taki sygnał będzie symetryczny i rzeczywisty, więc jego transformata Fouriera będzie również symetryczna i rzeczywista. Połowę transformaty można odrzucić, a druga połowa, zawierająca współczynniki rzeczywiste i będąca takiego samego rozmiaru co wyjściowy sygnał, nazywamy dyskretną transformatą cosinusową.

Fakt, że `DCT` sygnału jest również rzeczywista i ma ten sam rozmiar jest wygodny, lecz liczy się ją przede wszystkim dlatego, że ma właściwości dekorelacyjne. Zademonstrowano to na rysunkach 2.6 i 2.7. Dzięki temu przy modelowaniu `MFCC` za pomocą wielowymiarowej dystrybucji normalnej można przyjąć, że współczynniki są nieskorelowane i zastosować uproszczenie, że macierz kowariancji parametryzująca tę dystrybucję jest macierzą diagonalną.

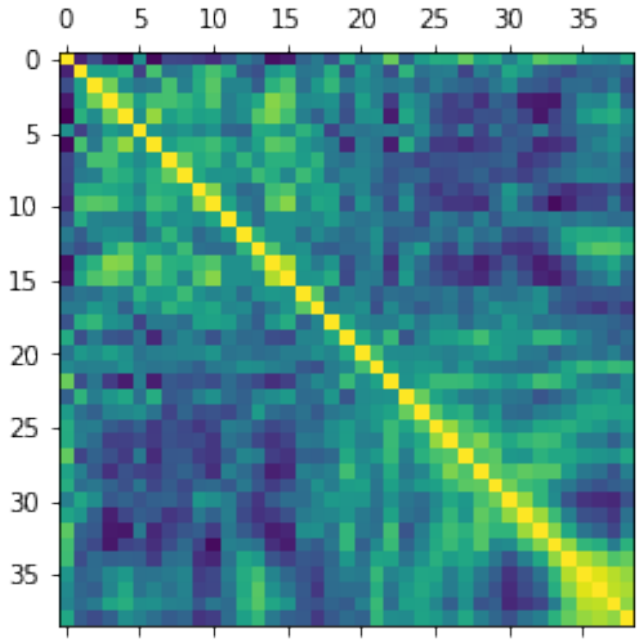


Rysunek 2.5: Wizualizacja 15 pierwszych współczynników `MFCC` dla każdej ramki przykładowego sygnału

Współczynniki wyjściowe `DCT` określamy jako współczynniki mel-cepstralne i ze względu na wspomniane właściwości użyjemy ich w naszych systemach. W praktyce z 26 współczynników wybiera się tylko pierwsze 13 lub 20, gdyż dalsze opisują wysokie częstotliwości kształtu widma, które są nieprzydatne przy rozumieniu sygnału mowy. Do tych współczynników dołącza się 26 lub 40 współczynników delta oraz delta-delta, obliczonych jako różnica między wartościami współczynników w obecnej i poprzedniej ramce. Współczynniki delta opisują jak konkretne współczynniki zmieniają się w czasie, choć zapewne tracą na przydatności w modelach jak sieci neuronowe, które mogą na wejściu przyjąć nie tylko jedną ramkę sygnału, ale też kilka sąsiadujących ramek.



Rysunek 2.6: Macierz korelacji dla współczynników uzyskanych z banku filtrów



Rysunek 2.7: Macierz korelacji dla współczynników `MFCC`

2.1.2 Mikstury gaussowskie

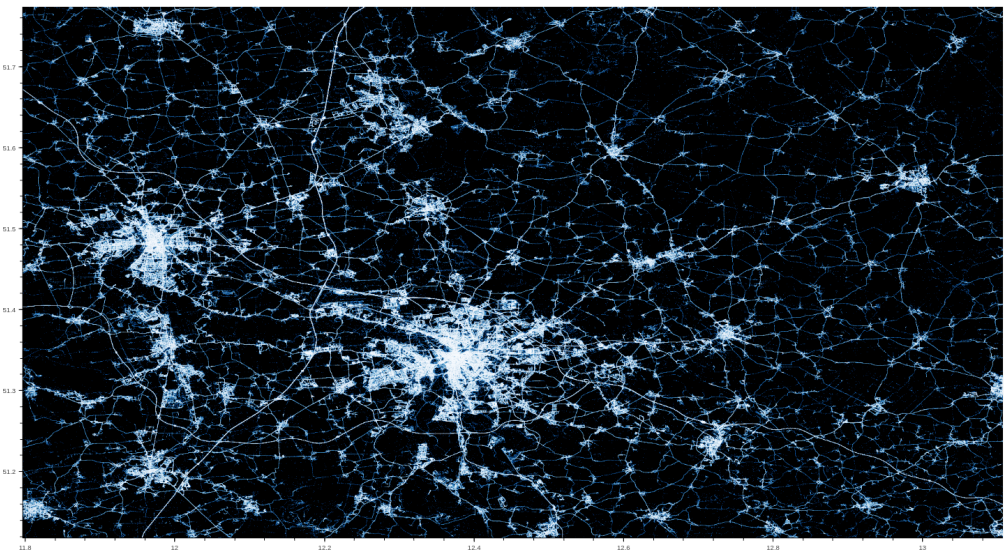
Mikstura rozkładów to sposób opisu rozkładu zmiennej losowej X przez wprowadzenie dodatkowej ukrytej zmiennej Θ. Następnie uznaje się, że dla każdej wartości zmiennej ukrytej rozkład zmiennej pochodzi z innego rozkładu. Zwykle zmienna ukryta jest dyskretna, wyjściowa zmienna może być dyskretna lub ciągła i przyjmuje się, że wszystkie warunkowe rozkłady $P(X|\Theta)$ pochodzą z tej samej rodziny i różnią się parametrami. Rozkład zmiennej X można uzyskać przez marginalizację zmiennej ukrytej.

$$f(x) = \sum_{\theta_i} f(X|\theta_i)P(\theta_i)$$

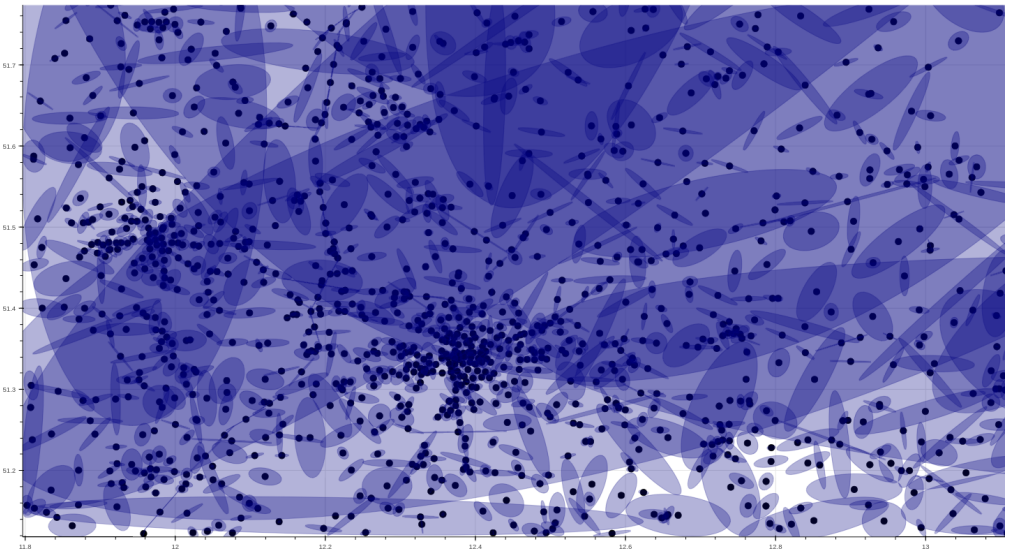
Przy miksturach gaussowskich (`GMM`, *Gaussian Mixture Model*) przyjmuje się, że warunkowe prawdopodobieństwa zmiennej wyjściowej od zmiennej ukrytej należą do rodziny wielowymiarowych rozkładów normalnych, tj. mają gęstość

$$f(x|\theta_i) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_i|}} \exp(-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i))$$

gdzie μ_i to d-wymiarowy wektor średnich, Σ_i to macierz kowariancji rozmiaru $d \times d$



Rysunek 2.8: Wizualizacja *datashader* danych o przejazdach w okolicach Lipska pobranych z nawigacji samochodowych



Rysunek 2.9: Wizualizacja `GMM` dopasowanych za pomocą `EM` do danych o przejazdach z rysunku 2.8

Na przykład, na rysunku 2.9 zaprezentowano model `GMM` dwuwymiarowej dystrybucji, z której pochodzą punkty z rysunku 2.8. W pracy `GMM` zostaną wykorzystane jednak do zamodelowania dystrybucji cech opisujących ramki z nagrania mowy.

2.1.3 Maksymalizowanie wartości oczekiwanej

Parametry wielowymiarowego rozkładu normalnego można wyestymować na podstawie próby, na przykład stosując estymator maksymalizujący prawdopodobieństwo danych.

Estymatory parametrów maksymalizujące prawdopodobieństwo M elementów x^j to:

$$\hat{\mu} = \frac{1}{m}\sum_{j=0}^{M} x^j$$

$$\hat{\Sigma} = \frac{1}{m}\sum_{j=0}^{M}(x^j - \hat{\mu})^T(x^j - \hat{\mu})$$

Jednak w przypadku mikstury, jeżeli nie przypiszemy z góry próbek wybranym wartościom ukrytej zmiennej, problem estymacji parametrów wymaga innego podejścia. Jedną z metod jest *Expectation Maximization*, iteracyjny algorytm wyznaczania wartości parametrów dla mikstur dystrybucji lub ogólniej dla modeli z ukrytymi zmiennymi. Składa się z dwóch kroków wykonywanych na zmianę do uzyskania zbieżności. Algorytm nie gwarantuje znalezienia globalnie optymalnego rozwiązania.

- *Expectation* - oszacuj wartości ukrytych parametrów, w przypadku `GMM` przypisz próbom prawdopodobieństwo tego, z jakiej mikstury zostały wygenerowane. Parametry jawne są tutaj niezmieniane, ich początkowe wartości można losowo zainicjalizować. Trzeba pamiętać jednak, by cechowały się dużą wariancją lub inaczej gwarantowały, że każda z próbek jest zgodnie z ich wartościami możliwa do wygenerowania. Inaczej pełne dopasowanie do danych będzie niemożliwe. Bardzo małe prawdopodobieństwo może przez błędy numeryczne zostać zaokrąglone do 0 i również spowodować błąd.

- *Maximization* - zakładając wyznaczone wartości ukrytych parametrów, znaleźć nowe `MAP` (*Maximum a Posteriori*) estymaty jawnych parametrów. Trzeba zwrócić uwagę, że pewne mikstury mogą nie mieć przypisanych żadnych próbek i wtedy uaktualnienie parametrów jest niemożliwe. Do tego uaktualniając wagi `GMM` należałoby przypisać takiej miksturze wagę 0, przez co nigdy nie byłaby już użyta. Trzeba zwrócić uwagę na ten problem i postarać się o lepsze zaincjalizowanie parametrów, dodatkowo modyfikować parametry, by nie dopuścić do takiej sytuacji lub przynajmniej pamiętać, by usunąć mikstury z wagą 0, gdyż będą niepotrzebnie zajmować zasoby komputera.

2.1.4 Dynamic Time Warping (DTW)[16]

Jako że praca dotyczy rozpoznawania zależnego od tekstu, konieczne by uzyskać wartościowe wyniki będzie użycie metod pozwalających na uwzględnienie informacji o sekwencji ramek. Samo dopasowanie mikstur do ramek tej informacji nie uwzględni. Jedną z takich metod jest *dynamic time warping*.

`DTW` to algorytm do porównywania dwóch ciągów. Ciągi mogą być różnej długości i algorytm zwróci dla nich wysokie podobieństwo (małą odległość) nawet jeśli pewne podciągi będą bardziej rozciągnięte lub krótsze w jednym ciągu niż w drugim.

Algorytm wymaga wybrania metryki (oznaczmy ją d) określającej odległość pojedynczych elementów ciągu. Elementy te mogą być liczbami, ale również wektorami.

`DTW` wykorzystuje technikę programowania dynamicznego i w podstawowej wersji znanej autorowi wymaga liczby operacji rzędu $\Theta(n \times m)$, gdzie n i m to długości porównywanych ciągów oraz ma złożoność pamięciową liniową względem długości krótszego ciągu. Istnieją jego bardziej wydaje wersje, w tym wersje z oknem, tzn. bazujące na założeniu, że nie ma sensu porównywać elementów z dwóch ciągów ze zbyt odległymi indeksami.

Algorytm wygląda tak: Na początku inicjalizowana jest macierz odległości D rozmiaru $n+1 \times m+1$. Pierwszy rząd i pierwsza kolumna jest wypełniana cyfrową reprezentacją ∞, co oznacza tyle, że te wartości nie są brane pod uwagę w następnym kroku. Wyjątkiem jest komórka w pierwszym rzędzie i pierwszej kolumnie z wartością $D_{0,0} = 0$.

Pozostałe wartości macierzy są liczone ze wzoru

$$D_{i+1,j+1} = min(D_{i+1,j}, D_{i,j+1}, D_{i,j}) + d(N_i, M_j)$$

gdzie N_i iM_j oznaczają elementy porównywanych ciągów, a $d(N_i, M_j)$ to jak już wspomniano wybrana niezależnie od algorytmu odległość między dwoma elementami. Odległość ciągów od siebie znaleziona przez `DTW` to $D_{n,m}$. Licząc te wartości rzędami możliwe jest osiągnięcie liniowej złożoności pamięciowej względem długości krótszego ciągu.

Powyższy wzór na $D_{i+1,j+1}$ należy zinterpretować tak, że liczymy odległość pary elementów N_i, M_j i dodajemy do najbardziej korzystnego wyniku spośród trzech opcji:

- Wybór $D_{i+1,j}$ oznacza, że obecny rezultat składa się z $d(N_i, M_{j-1})$ i $d(N_i, M_j)$, a więc ten sam element ciągu N jest porównany z dwoma elementami ciągu M. Na przykład porównując $N = [1, 20, 20, 20, 20]$, $M = [1, 1, 1, 10]$ i stosując jako miarę odległości różnicę absolutną ten przypadek będzie kilka razy najkorzystniejszy.

- Wybór $D_{i,j+1}$ oznacza, że w sumie będzie zarazem $d(N_{i-1}, M_j)$ i $d(N_i, M_j)$, a więc dwa elementy ciągu N są porównane z tym samym elementem ciągu M. Na przykład w wyniku dla $N = [1, 1, 1, 1, 1, 20]$ i $M = [1, 10]$ ten przypadek będzie się składał na wynik minimalizujący odległość ciągów.

- $D_{i,j}$ oznacza, że obliczono odległość następnego elementu zarówno z ciągu N i z ciągu M. Przy porównywaniu $N = [1, 2, 3, 4]$ oraz $M = [1, 2, 3, 4]$ zajdzie wyłącznie ten przypadek.

Poniżej przedstawiono kilka przykładów tabelek z wynikami *DTW*. Elementami są liczby, a w roli metryki jest moduł różnicy.

Tabela 2.1: Macierz D dla *DTW* na dwóch przykładowych sekwencjach N i M. Przykład gdy elementy jednej sekwencji są wielokrotnie dopasowane do pierwszego elementu z drugiej sekwencji, dopóki jest to możliwe

			N				
		-	1	20	20	20	20
	-	0	∞	∞	∞	∞	∞
	1	∞	0	19	38	57	76
M	1	∞	0	19	38	57	76
	1	∞	0	19	38	57	76
	10	∞	9	10	20	30	**40**

Tabela 2.2: Macierz D dla *DTW* na dwóch przykładowych sekwencjach N i M. Przykład z dokładnie dopasowanymi sekwencjami, z wydłużonymi lub skróconymi fragmentami

			N							
		-	1	1	2	3	3	3	4	4
	-	0	∞	∞	∞	∞	∞	∞	∞	∞
	1	∞	0	0	1	3	5	7	10	13
	2	∞	1	1	0	1	2	3	5	7
M	2	∞	2	2	0	1	2	3	5	7
	2	∞	3	3	0	1	2	3	5	7
	3	∞	5	5	1	0	0	0	1	2
	4	∞	8	8	3	1	1	1	0	**0**

2.1.5 Ukryte modele Markowa

Ukryty model Markowa (`HMM`, *Hidden Markov Model*) to proces stochastyczny, dyskretny w czasie, pozwalający na modelowanie dystrybucji sekwencji zdarzeń. Będzie zatem użyteczny ze względu na możliwość uwzględnienia w nim zależności między ramkami w czasie.

Model składa się ze zbioru N ukrytych stanów $X = \{x_0, x_1, \dots x_{N-1}\}$, ciągu zmiennych losowych X_t, $t = 0, 1, \dots, T-1$ oznaczających ukryty stan w chwili t, rozkładu początkowego stanu ukrytego $\pi_i = P(X_0 = x_i)$, zbioru obserwacji (emisji, stan jawny) Y, przy czym obserwacje mogą być dyskretne lub ciągłe i ciągu zmiennych losowych Y_t, $t = 0, 1, \dots, T-1$ określających obserwację w chwili t.

Do tego zdefiniowane są prawdopodobieństwa przejść między stanami oraz rozkłady obserwacji w każdym stanie. Kluczową kwestią w modelu Markowa, zwaną regułą Markowa, jest iż rozkład prawdopodobieństwa następnego stanu X_t zależy

wyłącznie od stanu poprzedzającego, nie jest ważna cała historia stanów w przeszłości. Podobnie rozkład obserwacji Y_t zależy wyłącznie od obecnego stanu ukrytego.

Inną właściwością, którą się przyjmuje, jest jednorodność. Oznacza to, że przejścia między stanami i emisje są niezależne od czasu t. Przy dyskretnych i jednorodnych stanach ukrytych prawdopodobieństwa przejść między stanami można zapisać w postaci macierzy H (*transition matrix*) $P(X_t = x_i|X_{t-1} = x_j) = H_{j,i}$. Podobnie jeżeli emisje są dyskretne, to można je zapisać w postaci macierzy V $P(Y_t = y_i|X_t = x_j) = V_{j,i}$. Lecz emisje często mają jakieś określone parametryczne dystrybucje i wtedy definiuje się w postaci wektorów parametrów. H (oraz V jeżeli mowa o dyskretnych emisjach) jest macierzą stochastyczną, więc $\forall_{i,j=0\ldots N-1} 0 \leqslant H_{i,j} \leqslant 1$ i $\forall_{i=0\ldots N-1} \sum_{j=0}^{N-1} H_{j,i} = 1$ (wiersze sumują się do 1).

Czasami, w innych modelach regułę Markowa rozszerza się tak, iż prawdopodobieństwo może zależeć od k ostatnich stanów $P(X_t|X_{t-1} \ldots X_{t-k})$ i nazywa się to regułą Markowa k rzędu. Są też inne wersje, w których zbiór ukrytych stanów jest ciągły, w których rząd k jest zmienny lub na przykład pola Markowa, gdzie nie mówimy o ciągu w czasie, lecz o zmiennych rozmieszczonych w kilku wymiarach, których wartości zależą od pewnego sąsiedztwa. Wszystkie modele Markowa mają jednak cechę wspólną, iż zmienne losowe zależą w nich od pewnego skończonej liczby innych zmiennych.

Jeżeli macierz przejść jest tak zdefiniowana, że zawsze możliwe jest przejście z każdego stanu do każdego, to mówimy, że model jest ergodyczny. Jednak na przykład w rozpoznawaniu mowy często definiuje się macierz przejść, by pozwalała tylko na przejście do stanu o takim samym lub wyższym indeksie, tzw. model Bakisa lub model lewo-prawo (*left-to-right*) albo by miała niezerowe prawdopodobieństwo przejścia wyłącznie z danego stanu do jego samego lub stanu o indeksie o jeden większym (model *beads-on-string*).

Ważne w praktyce jest, że istnieją wydajne algorytmy rozwiązujące następujące problemy `HMM`.

1. Algorytm w przód - problem znalezienia prawdopodobieństwa stanu ukrytego w danej chwili wraz z pewnym ciągiem obserwacji do tego chwili. Jeżeli tym punktem w czasie będzie koniec sekwencji obserwacji, to można zmarginalizować prawdopodobieństwo końcowego stanu i uzyskać prawdopodobieństwo wygenerowania danej sekwencji obserwacji przez model.

2. Algorytm w tył - problem znalezienia prawdopodobieństwa ciągu zdarzeń w przyszłości, pod warunkiem danego stanu ukrytego w danej chwili. Jest on istotny, gdyż jest częścią następnego algorytmu.

3. Algorytm w przód i w tył - problem znalezienia prawdopodobieństwa stanu ukrytego w danej chwili pod warunkiem wystąpienia danego ciągu obserwacji. Brany pod uwagę jest cały ciąg obserwacji, tych przed i tych po danym stanie ukrytym. Działa przez połączenie wyników algorytmu w przód i w tył, stąd nazwa.

4. Algorytm Viterbiego - problem znalezienia najbardziej prawdopodobnego ciągu stanów ukrytych dla danego ciągu obserwacji parametrów modelu. Jest podobny do algorytmu w przód, z tym, że na każdym kroku wybiera najbardziej prawdopodobną sekwencję zamiast uwzględniać prawdopodobieństwa wszystkich stanów.

5. Algorytm Bauma-Welcha - problem znalezienia parametrów modelu Markowa (prawdopodobieństwo przejść między stanami ukrytymi, rozkład obserwacji dla każdego stanu ukrytego, rozkład stanu początkowego) maksymalizujących prawdopodobieństwo wystąpienia danego ciągu obserwacji. Wykorzystuje algorytm w przód i w tył oraz opisany wcześniej algorytm *expectation maximization*, algorytm w przód i w tył służy do wyznaczenia ukrytych zmiennych opisujących jaki był stan w każdej chwili czasu na potrzeby kroku *expectation*, a następnie wyznaczane są najbardziej prawdopodobne parametry emisji w kroku *maximization*.

Poniżej wyprowadzono te algorytmy, zakładając przy tym dyskretne emisje.

Algorytm w przód

Algorytm w przód (*forward algorithm*) liczy łączone prawdopodobieństwo znalezienia się w jakimś stanie w chwili t i wystąpienia pewnego ciągu obserwacji $y_{t-1} \dots y_0$ w przeszłości. To znaczy oblicza on $P(x_t, y_t \dots y_0)$, korzystając z rekurencyjnej zależności:

$$P(x_t, y_t \dots y_0) = \sum_{x_{t-1}} P(x_t, x_{t-1}, y_t \dots y_0) =$$

$$\sum_{x_{t-1}} P(y_t|x_t, x_{t-1}, y_{t-1} \dots y_0) P(x_t|x_{t-1}, y_{t-1} \dots y_0) P(x_{t-1}, y_{t-1} \dots y_0)$$

Dzięki własności Markowa można to uprościć do:

$$P(y_t|x_t, x_{t-1}, y_{t-1} \dots y_0) = P(y_t|x_t) = V_{x_t, y_t}$$

$$P(x_t|x_{t-1}, y_{t-1} \dots y_0) = P(x_t|x_{t-1}) = H_{x_{t-1}, x_t}$$

Ostateczną zależnością jest

$$P(x_t, y_t \dots y_0) = V_{x_t, y_t} \sum_{x_{t-1}} H_{x_{t-1}, x_t} P(x_{t-1}, y_{t-1} \dots y_0)$$

gdzie $P(x_{t-1}, y_{t-1} \dots y_0)$ może być rekurencyjnie obliczone w ten san sposób aż do przypadku bazowego $P(x_0, \emptyset) = \pi_1$, gdzie π to rozkład początkowy.

Użycie własności Markowa pozwala zredukować złożoność obliczeniową problemu z $\theta(MN^M)$ do $\theta(MN^2)$, gdzie M to możliwa liczba obserwacji, a N to liczba stanów ukrytych.

Algorytm w tył

Algorytm w tył (*backward algorithm*), jest podobny do algorytmu w przód. Oblicza $P(y_{t+1} \dots y_{T-1}|x_t)$, to znaczy prawdopodobieństwo wystąpienia wskazanej sekwencji obserwacji $y_{t+1} \dots y_{T-1}$, gdzie T to liczba obserwacji, zakładając początkowy stan ukryty x_t w chwili t. Obliczenia zaczyna się od końcowego stanu x_{T-1}.

$$P(y_{t+1} \dots y_{T-1}|x_t) = \sum_{x_{t+1}} P(y_{t+1} \dots y_{T-1}, x_{t+1}|x_t) =$$

$$\sum_{x_{t+1}} P(y_{t+1}|y_{t+2} \dots y_{T-1}, x_{t+1}, x_t) P(y_{t+2} \dots y_{T-1}|x_{t+1}, x_t) P(x_{t+1}|x_t)$$

Korzystając z tego można uprościć:

$$P(y_{t+1}|y_{t+2} \dots y_{T-1}, x_{t+1}, x_t) = P(y_{t+1}|x_{t+1}) = V_{x_{t+1}, y_{t+1}}$$

$$P(y_{t+2} \dots y_{T-1}|x_{t+1}, x_t) = P(y_{t+2} \dots y_{T-1}|x_{t+1})$$

$$P(x_{t+1}|x_t) = H_{x_t, x_{t+1}}$$

Ostateczna zależność to:

$$P(y_{t+1} \dots y_{T-1}|x_t) = \sum_{x_{t+1}} V_{x_{t+1}, y_{t+1}} H_{x_t, x_{t+1}} P(y_{t+2} \dots y_{T-1}|x_{t+1})$$

Prawdopodobieństwo jest liczone rekurencyjnie, z przypadkiem bazowym

$$P(\emptyset|x_{T-1}) = 1$$

Algorytm w przód i w tył

Algorytm w przód i w tył (*forward-backward algorithm*) działa przez połączenie wyników algorytmu w przód i w tył.

Przypomnijmy, że:

1. Algorytm w przód oblicza $P(x_t, y_t \dots y_0)$, to znaczy prawdopodobieństwo dla ukrytego stanu w chwili t oraz ciągu stanów z przeszłości.

2. Algorytm w tył oblicza $P(y_{T-1} \dots y_{t+1}|x_t)$, czyli warunkowe prawdopodobieństwo ciągu obserwacji z przyszłości wobec ukrytego stanu w chwili t.

3. Algorytm w przód i w tył oblicza $P(x_t|y_{T-1} \dots y_0)$. Oblicza zatem prawdopodobieństwo ukrytego stanu x_t w dowolnej chwili t biorąc pod uwagę cały ciąg obserwacji. Możliwe jest dzięki niemu wybranie najbardziej prawdopodobnego stanu w każdej chwili. Trzeba zauważyć, że biorąc najbardziej prawdopodobny stan z każdej chwili niekoniecznie uzyska się najbardziej prawdopodobny ciąg stanów ukrytych odpowiadający ciągowi obserwacji, bo przejście z jednego bardzo prawdopodobnego stanu do drugiego może być bardzo nieprawdopodobne. Dlatego do tego zadania służy algorytm Viterbiego.

Algorytm *forward-backward* wykorzystuje poniższe zależności:

$$P(x_t, y_t \dots y_0)P(y_{T-1} \dots y_{t+1}|x_t) = P(y_t \dots y_0|x_t)P(y_{T-1} \dots y_{t+1}|x_t)P(x_t) =$$

$$P(y_{T-1} \dots y_0|x_t)P(x_t) = P(y_{T-1} \dots y_0, x_t) = P(x_t|y_{T-1} \dots y_0)P(y_{T-1} \dots y_0)$$

Skąd wynika natychmiast:

$$P(x_t|y_{T-1} \dots y_0) = \frac{P(x_t, y_t \dots y_0)P(y_{T-1} \dots y_{t+1}|x_t)}{P(y_{T-1} \dots y_0)}$$

Prawdopodobieństwa w liczniku zostały obliczone odpowiednio przez algorytm w przód i algorytm w tył. Prawdopodobieństwo w mianowniku można wyznaczyć przez zsumowanie po możliwych wartościach X_{T-1} w wynikach algorytmu w przód $\sum_{x_{T-1}} P(x_{T-1}, y_{T-1} \dots y_0)$

Algorytm Bauma-Welcha

Algorytm Bauma-Welcha (*Baum-Welch algorithm*) pozwala na znalezienie najbardziej prawdopodobnych parametrów H, π, V modelu Markowa zależnie od ciągu obserwacji. Algorytm bazuje na metodzie *expectation-maximization*. Wykorzystuje przy tym algorytm w przód i w tył (*forward-backward*) do znalezienia w kroku *expectation* prawdopodobieństw ukrytych stanów dla danej sekwencji obserwacji. W kroku *maximization* estymuje parametry szukając takich, które maksymalizują prawdopodobieństwo wystąpienia obserwacji, przy założeniu takich ukrytych stanów jak to wyznaczono w poprzednim kroku. Operacje te są powtarzane.

Jak pokazano powyżej, algorytm *forward-backward* pozwala obliczyć $P(x_t|y_{T-1} \dots y_0)$, łącząc $P(x_t, y_t \dots y_0)$ $P(y_{T-1} \dots y_{t+1}|x_t)$ liczone przez algorytmy *forward* i *backward*.

Do przeprowadzenia treningu *Bauma-Welcha* potrzebne jest dodatkowo wyznaczenie $P(x_t, x_{t+1}|y_{T-1} \dots y_0)$, czyli prawdopodobieństwa przejścia z jednego stanu w chwili t do drugiego w chwili $t+1$, uwzględniając sekwencję T obserwacji i, niejawnie, parametrów modelu.

$$P(x_t, x_{t+1}|y_{T-1} \dots y_0) = \frac{P(x_t, x_{t+1}, y_{T-1} \dots y_0)}{P(y_{T-1} \dots y_0)}$$

$$P(x_t, x_{t+1}, y_{T-1} \dots y_0) = P(y_{T-1} \dots y_0|x_t, x_{t+1})P(x_{t+1}|x_t)P(x_t) =$$

$$P(y_{T-1} \dots y_{t+2}|x_t, x_{t+1})P(y_{t+1} \dots y_0|x_t, x_{t+1})P(x_{t+1}|x_t)P(x_t)$$

Iloczyn drugiego i czwartego czynnika możemy wyrazić następująco:

$$P(y_{t+1} \dots y_0|x_t, x_{t+1})P(x_t) = P(y_{t+1} \dots y_0, x_t|x_{t+1}) =$$

$$P(y_{t+1}|x_{t+1})P(y_t \dots y_0, x_t|x_{t+1})$$

Licznik wyrażenia w pierwszej linijce, korzystając z wyprowadzonej właśnie zależności oraz z reguły Markowa o tym, iż stan ukryty zależy tylko od poprzedniego stanu, a emisja zależy od obecnego stanu, można wyrazić tak:

$$P(x_t, x_{t+1}, y_{T-1} \dots y_0) =$$

$$P(y_{T-1} \dots y_{t+2}|x_t, x_{t+1})P(y_{t+1}|x_{t+1})P(y_t \dots y_0, x_t|x_{t+1})P(x_{t+1}|x_t) =$$

$$P(y_{T-1} \dots y_{t+2}|x_{t+1})P(y_{t+1}|x_{t+1})P(y_t \dots y_0, x_t)P(x_{t+1}|x_t)$$

$$P(x_t, x_{t+1}|y_{T-1} \dots y_0) = \frac{P(y_{T-1} \dots y_{t+2}|x_{t+1})P(y_t \dots y_0, x_t)V_{x_{t+1},y_{t+1}}H_{x_t,x_{t+1}}}{P(y_{T-1} \dots y_0)}$$

Analogicznie jak w przypadku algorytmu w przód i w tył, pierwszy czynnik w liczniku można obliczyć za pomocą algorytmu w tył, a drugi za pomocą algorytmu w przód. Wtedy można uaktualniać parametry w poniższy sposób:

$$\pi_i = P(X_0 = x_i|y_{T-1} \dots y_0)$$

$$H_{x_t,x_{t+1}} = \frac{\sum_{t=0}^{T-1} P(x_t, x_{t+1}|y_{T-1} \dots y_0)}{\sum_{t=0}^{T-1} P(x_t|y_{T-1} \dots y_0)}$$

$$V_{x_t,y_t} = \frac{\sum_{t=0}^{T-1} 1_{Y_t=y_t} P(x_t|y_{T-1} \dots y_0)}{\sum_{t=1}^{T} P(x_t|y_{T-1} \dots y_0)}$$

Założone jest tutaj, iż obserwacje są dyskretne i opisane macierzą stochastyczną. W przeciwnym razie trzeba użyć innego estymatora.

Algorytm Viterbiego

Używając algorytmu *forward-backward* można obliczyć prawdopodobieństwa dla każdego stanu w każdej chwili, biorąc pod uwagę cały ciąg obserwacji $P(x_t|y_{T-1} \dots y_0)$. Można by wziąć najbardziej prawdopodobny stan z każdej chwili, lecz nie dałoby to koniecznie najbardziej prawdopodobnej sekwencji stanów ukrytych, gdyż możliwa jest sytuacja, że prawdopodobieństwo przejścia między dwoma wybranymi stanami byłoby bardzo małe lub niemożliwe i w efekcie wystąpienie i jednego i drugiego z nich w jednym ciągu byłoby mało prawdopodobne.

Algorytm Viterbiego (*Viterbi algorithm*) rozwiązuje problem znalezienia najbardziej prawdopodobnej sekwencji ukrytych stanów pod warunkiem danego ciągu obserwacji, $y_{T-1} \dots y_0$ maksymalizującego $P(x_{T-1} \dots x_0|y_{T-1} \dots y_0)$.

Algorytm wykorzystuje technikę programowania dynamicznego. Tworzy dwie matryce, oznaczmy je A i B.

$$A_{t,x_i} = max_{x_1 \dots x_{t-1}} P(x_1 \dots x_{t-1}, X_t = x_i, y_0 \dots y_t)$$

czyli zawiera największe prawdopodobieństwo jakie może mieć ciąg stanów $x_0 \dots x_{t-1}$ wraz z zajściem ciągu obserwacji $y_0 \dots y_t$ oraz z prawdopodobieństwem, że w następnym kroku stanem będzie x_i .

$$B_{t,x_i} = argmax_{x_1 \dots x_{t-1}} P(x_1 \dots x_{t-1}, X_t = x_i, y_0 \dots y_t)$$

i służy do odtworzenia na koniec, które ukryte stany wchodzą w skład wybranej ścieżki. Warto zauważyć, że w tym przypadku $t > 0$.

Mając A_{t-1,x_i}, algorytm oblicza prawdopodobieństwo dla następnej chwili czasu A_{t,x_i} w taki sposób:

$$P(x_1 \dots x_t, y_0 \dots y_t) = P(y_t | x_1 \dots x_t, y_0 \dots y_{t-1}) P(x_1 \dots x_t, y_0 \dots y_{t-1}) =$$

$$P(y_t | x_1 \dots x_t, y_0 \dots y_{t-1}) P(x_t | x_1 \dots x_{t-1}, y_0 \dots y_{t-1}) P(x_1 \dots x_{t-1}, y_0 \dots y_{t-1}) =$$

$$P(y_t | x_t) P(x_t | x_{t-1}) P(x_1 \dots x_{t-1}, y_0 \dots y_{t-1}) =$$

$$V_{x_t,y_t} H_{x_{t-1},x_t} P(x_1 \dots x_{t-1}, y_0 \dots y_{t-1})$$

$$A_{t,x_i} = max_{x_1 \dots x_{t-1}} P(x_1 \dots x_{t-1}, X_t = x_i, y_0 \dots y_t) =$$

$$max_{x_1 \dots x_{t-1}} V_{x_i,y_t} H_{x_{t-1},x_i} P(x_1 \dots x_{t-1}, y_0 \dots y_{t-1}) =$$

$$V_{x_i,y_t} max_{x_{t-1}} H_{x_{t-1},x_i} A_{t-1,x_{t-1}}$$

Przypadek bazowy to $A_{1,x_i} = P(X_1 = x_i, y_0) = \pi_i V_{x_i,y_0}$

Jeśli chodzi o B, ta macierz zachowuje argumenty, dla których w A jest maksimum:

$$B_{t,x_i} = argmax_{x_1 \dots x_{t-1}} P(x_1 \dots x_{t-1}, X_t = x_i, y_0 \dots y_t) =$$

$$argmax_{x_{t-1}} H_{x_{t-1},x_i} A_{t-1,x_{t-1}}$$

Po zakończeniu obliczeń możemy znaleźć najbardziej prawdopodobną ścieżkę stanów przez wybranie stanu x_i z największym prawdopodobieństwem w chwili $T-1$ i odtworzyć całą ścieżkę korzystając z B[14].

2.2 Rozpoznawanie mówcy

W mojej pracy nad systemami rozpoznawania mówcy wyróżniłem trzy etapy działania: treningu (*training*), zapisów (*enrollment*) i prób (*evaluation*). Jest to podział inspirowany podziałem w zbiorze `RedDots`.

W czasie treningu model jest uczony na możliwie dużym zbiorze danych i przez być może długi czas. W czasie zapisów jest on dostosowywany do wybranego mówcy na podstawie możliwie małego zbioru nagrań wyłącznie tego mówcy. Jest to istotne, gdyż proces nagrywania może być uciążliwy i ze względów praktycznych byłoby lepiej, gdyby wymagał jak najmniej pracy ze strony rejestrowanej osoby. W czasie prób następuje weryfikacja mówcy na podstawie niewidzianych wcześniej nagrań.

2.2.1 Adaptacja MAP

Adaptacja MAP (*maximum a posteriori*) pozwala na uzyskanie modelu dla mówcy na podstawie niewielu nagrań. Wykorzystywany jest model niezależny od mówcy, jak UBM (*Universal Background Model*), nauczony na dużym zbiorze danych. Metoda polega na znalezieniu parametrów λ maksymalizujących $P(\lambda|X)$, czyli prawdopodobieństwo wartości parametrów przy uwzględnieniu zestawu danych.

Zgodnie z regułą Bayesa:

$$P(\lambda|X) = \frac{P(X|\lambda)P(\lambda)}{P(X)}$$

Nazwa MAP (*Maximum a Posteriori*) wzięła się z terminów używanych wobec zmiennych w tym równaniu:

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

W statystyce bayesowskiej parametry opisuje się jako zmienne losowe. Przy wyborze ich dystrybucji zwraca się uwagę na to czy są rozkładem a priori sprzężonym (*conjugate prior*) wobec rozkładu a posteriori $P(X|\lambda)$, czyli dystrybucji, która zależy od tych parametrów. Oznacza to taką cechę, że *posterior* będzie po uwzględnieniu *likelihood* miał rozkład z tej samej rodziny co *prior*, lecz z uaktualnionymi parametrami. Przykładowo rozkład normalny ma taką własność wobec rozkładu normalnego. MAP różni się nieco od tego podejścia, gdyż zamiast dystrybucji zwraca jedną wartość parametru, dla której prawdopodobieństwo jest największe.

Mając GMM niezależny od mówcy (czyli UBM) i zakładając że parametry mają rozkład normalny, można wyznaczyć $P(\lambda)$ i obliczyć $P(X|\lambda)$ dla zestawu nagrań danego mówcy. Pozwala to zastosować MAP i znaleźć nowe estymaty parametrów modelu. Przy tym im mniej danych, tym większy wpływ na wynik będzie miał *prior*.

Jeżeli dystrybucja z której generowane są dane ma rozkład normalny, to wzór na uaktualnioną średnią, to:

$$\bar{\mu_{MAP}} = \frac{\sigma^2}{\sigma^2 + n\sigma_0^2}\mu_0 + \frac{n\sigma_0^2}{\sigma^2 + n\sigma_0^2}\bar{\mu}$$

gdzie μ_0, σ_0^2 to średnia i wariancja rozkładu *a priori*, n to rozmiar próbki, $\bar{\mu}$ to średnia obliczona na podstawie nowej próbki, σ^2 to wariancja próbki. Wariancji się dla uproszczenia nie reestymuje i przyjmuje, że wartość *prior* jest prawdziwa. Jak widać dla $n = 0$ MAP estymatą będzie po prostu poprzednia średnia μ_0, a im większa liczba danych, tym większy będzie współczynnik przy $\bar{\mu}$. Do tego im większa będzie wariancja *priora* σ_0^2 względem σ^2, tym również poprzednia wartość średniej będzie miała mniejszy wpływ na wynik, przy bardzo dużej wariancji mówi się o *uninformative prior*[15].

Adaptacja GMM wygląda podobnie, lecz dodatkowo uwzględnia przy każdej miksturze prawdopodobieństwo, że próbka pochodzi z i elementu mikstury $P(\theta_i|x_j)$. Wartości te służą jako wagi dla próbek. Do tego w praktyce, jako że *prior* nie

jest opisana jako zmienna, lecz jako punktowa wartość, to zamiast operować na σ_0 i σ, wprowadzony zostaje parametr r, *relevance factor*, odpowiadający stosunkowi wariancji $\frac{\sigma^2}{\sigma_0^2}$. Im większy r, tym większy wpływ poprzedniej wartości parametru na wynik adaptacji przy tej samej liczbie danych. Wzór na adaptację k mikstury z `GMM`[11]:

$$\bar{\mu_{MAP}}_k = (1 - \alpha_k)\mu_{0,k} + \alpha_k \bar{\mu_k}$$

gdzie

$$\alpha_k = \frac{N_k}{N_k + r}$$
$$\bar{\mu_k} = \frac{F_k}{N_k}$$
$$N_k = \sum_{x_i \in X} P(\theta_k | x_i)$$
$$F_k = \sum_{x_i \in X} P(\theta_k | x_i) x_i$$

2.2.2 Ocena klasyfikatorów binarnych

Problem klasyfikacji binarnej to problem przyporządkowania elementów z pewnej dziedziny do jednej z dwóch klas. Nazywa się te klasy *positive* (klasa obiektów posiadających pewną cechę) i *negative*. W przypadku weryfikacji mówcy problemem będzie czy osobą na nagraniu jest ten mówca, którego oczekujemy. Rezultaty klasyfikacji można podzielić na cztery grupy:

True positive Rezultat poprawnie (to znaczy zgodnie ze znanym, prawdziwym podziałem) zaklasyfikowany jako posiadający daną cechę. W problemie weryfikacji mówcy oznacza to, że mówca został poprawnie zweryfikowany.

False positive Rezultat niepoprawnie zaklasyfikowany jako posiadający daną cechę. Mówca został zweryfikowany jako uprawniony użytkownik, choć tak naprawdę nim nie jest. Ten rodzaj błędnej klasyfikacji nazywany jest błędem typu pierwszego (*type I error*).

True negative Rezultat poprawnie zaklasyfikowany jako nieposiadający danej cechy. Mówca nie przeszedł pomyślnie weryfikacji jako ktoś inny.

False negative Rezultat niepoprawnie zaklasyfikowany jako nieposiadający danej cechy. To znaczy mówca nie przeszedł weryfikacji, choć w rzeczywistości jest tym, za kogo się podaje. Ten rodzaj błędnej klasyfikacji nazywany jest błędem typu drugiego (*type II error*).

Przez tp, fp, tn, fn oznaczmy liczby elementów należące kolejno do wymienionych grup. Można dokonać oceny klasyfikatora na podstawie tego, na ile podział na klasy, którego dokonuje, jest zgodny ze znanym, prawdziwym podziałem. W tym celu zdefiniowano pewne współczynniki zależne od liczb tp, fp, tn i fn, to znaczy:

Precyzja (*precision*) Określa jaka część z elementów zaklasyfikowanych jako posiadająca cechę rzeczywiście ją posiada. $Precision = \frac{tp}{tp+fp}$

Kompletność (*recall*) lub czułość (*sensitivity, true positive rate*) Określa ile elementów, które posiadają cechę zostały tak zaklasyfikowanych. $TPR = \frac{tp}{tp+fn}$

Swoistość (*specifity, true negative rate*) Określa ile elementów spośród nieposiadających danej cechy zostało tak zaklasyfikowanych. $TNR = \frac{tn}{tn+fp}$

fall-out, false positive rate Określa ile elementów spośród nieposiadających danej cechy zostało nieprawidłowo zaklasyfikowanych. $FPR = \frac{fp}{tn+fp} = 1 - TNR$

Dokładność (*accuracy*) Określa ile elementów zostało poprawnie zaklasyfikowanych, niezależnie od tego do której klasy. $ACC = \frac{tp+tn}{tp+fp+tn+fn}$

Można też wyznaczyć inne stosunki między tymi liczbami elementów, jednak te pięć zdaje się być często wykorzystywane do oceny modeli. Każdy z tych współczynników daje ograniczony ogląd na skuteczność klasyfikatora. Na przykład dokładność równa 1.0 może oznaczać, iż klasyfikator wszystkie elementy klasyfikuje jako nieposiadające cechy i przez to unika błędu typu pierwszego, będąc jednak bezużytecznym. Z tego powodu do oceny modelu najczęściej wykorzystuje się te współczynniki parami: precyzja i kompletność lub czułość i swoistość.

Można również obliczyć współczynnik F dla pary współczynników.

$$F-measure = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$$

Jest to średnia harmoniczna współczynników i pozwala uwzględnić oba jednocześnie.

Krzywa ROC

Klasyfikatory binarne mogą działać na takiej zasadzie, iż ich rezultatem jest prawdopodobieństwo tego, że dany element należy do danej klasy. Są wtedy parametryzowane progiem, jakie prawdopodobieństwo jest wymagane, by obiekt został do zaklasyfikowany jako prawdziwy. Można wtedy opisać klasyfikator przez wykres krzywej `ROC` (*receiver operating characteristic*). Na wykresie tym nanosi się punkty z wartościami (TPR, FPR), czyli kompletność oraz $(1.0 -$ swoistość$)$, dla progu klasyfikacji zmieniającego się od 0 do 1.0.

Cechy `ROC` są takie, iż dla progu równego 0 (wszystko uznajemy za prawdziwe, $fn = tn = 0$), `TPR` i `FPR` wyniosą 1.0, bo wszystko jest wtedy akceptowane, niezależne od rzeczywistości. Dla progu równego 1 (wszystko klasyfikujemy jako fałszywe, $tp = fp = 0$), `TPR` i `FPR` wyniosą 0.0, bo nie będzie w ogóle żadnych pozytywów. Krzywa zatem będzie biec od punktu $(0, 0)$ do $(1, 1)$. Krzywa dla idealnego klasyfikatora będzie przechodzić przez punkt $(0, 1)$, gdzie kompletność i swoistość

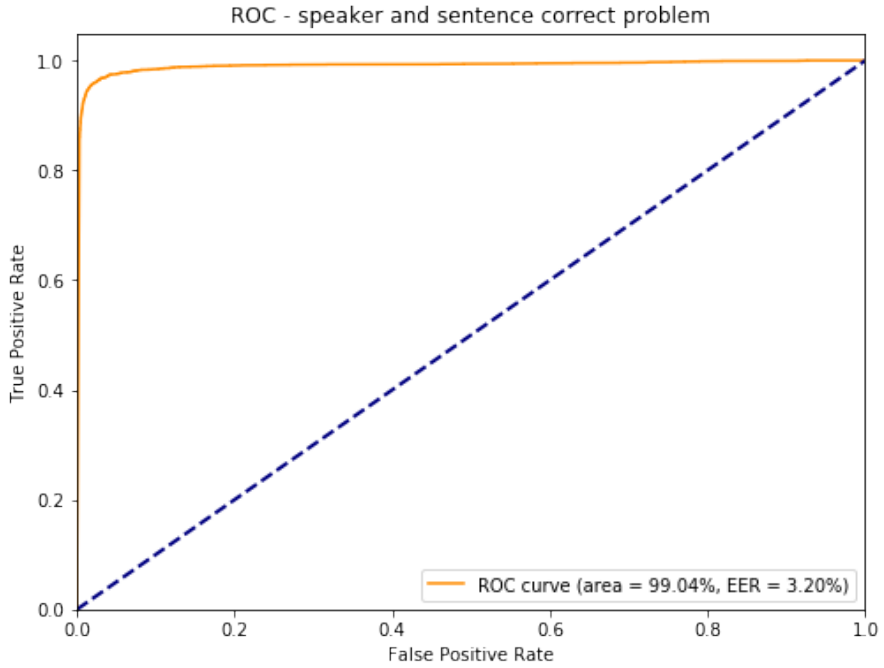
wynoszą 1. Klasyfikator działający zupełnie losowo będzie miał krzywą biegnącą prosto od $(0, 0)$ do $(1, 1)$. Z krzywej ROC można odczytać optymalną wartość progu dla klasyfikatora - jest to próg dla którego punkt na wykresie jest najbliżej punktu $(0, 1)$.

Przydatna jest możliwość zawarcia oceny klasyfikatora za pomocą jednej liczby zamiast całego wykresu. Dlatego wylicza się pole pod krzywą ROC, tzw. AUC (*area under the curve*).

$$AUC = \int_{-\infty}^{\infty} TPR(T)FPR'(T)dT$$

AUC przyjmuje wartości od 0.5 dla klasyfikatora losowego do 1.0 dla klasyfikatora idealnego.

Alternatywą dla AUC jest EER (*equal error rate*). Wyznacza się je przez znalezienie punktu, dla którego wartości FPR i 1 - TPR są równe. Wartość błędu w tym punkcie stanowi wartość EER. Graficznie będzie to punkt przecięcia krzywej ROC z linią od $(0, 1)$ do $(1, 0)$.



Rysunek 2.10: Przykładowy wykres krzywej ROC wygenerowany na podstawie wyników modelu GMM-UBM opisanego w pracy

2.2.3 Long Short-Term Memory

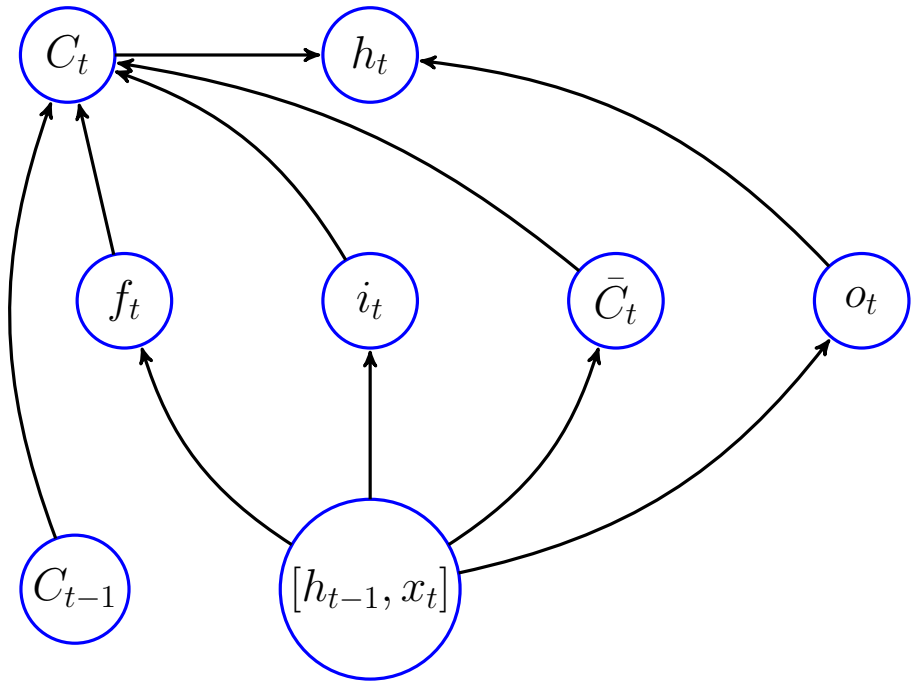
Rekurencyjne sieci neuronowe (RNN, *Recursive Neural Network*) to rodzaj sieci neuronowych, które, jak HMM, służą do modelowania ciągów obserwacji w czasie. Posiadają one połączenia między ukrytymi warstwami, które prowadzą do tej samej lub poprzednich warstw (w przeciwieństwie do sieci *feed-forward*) i przekazują wartości z opóźnieniem w czasie[13].

Choć prosta architektura, w której wyjścia z warstwy są ponownie przekazywane na jej wejście, pozwala wyrazić wiele zależności, to wyznaczenie parametrów w tej sieci sprawia trudności. Jeżeli zamiast ręcznie dobierać wagi zastosuje się do ich wyznaczenia metodę gradientową, pojawia się problem z zanikającym lub eksplodującym gradientem. Polega on na tym, iż przy liczeniu gradientu błędu następuje wielokrotne mnożenie przez macierz wag. Jeżeli ta operacja powoduje zmianę długości wektora, to przy wielokrotnym mnożeniu ta długość szybko będzie rosnąć lub maleć. Sprawia to, że trudno jest odpowiednio dopasować wagi, gdy wynik zależy od ramki w odległej przeszłości.

W `LSTM` ten problem został ominięty. W ich przypadku poza tym, że wartości na wyjściu są przekazywane na wejście w następnej chwili czasu, na stan składa się także dodatkowa komórka z ukrytym stanem. Architektura sieci jest tak wymyślona, że stan ukryty jest mnożony tylko przez bramkę o ograniczonych wartościach oraz w sumowany z nowym kandydatem na stan ukryty, lecz nie jest wielokrotnie mnożony przez żadną macierz o trudnej do ograniczenia postaci. Może zatem zostać w pewnej części wymazany lub zastąpiony, lecz jego długość nie ma tendencji, by z każdym krokiem wykładniczo rosnąć lub maleć. Tak samo przy liczeniu gradientu zachowywana jest jego długość. Utrzymywanie takiej samej wartości stanu ukrytego przez wiele kroków lub jego całkowite zastąpienie w jednym kroku jest proste do osiągnięcia.

W poniższych wzorach $\cdot$ oznacza iloczyn macierz-wektor, a $\circ$ operację wymnożenia parami elementów wektorów równej długości. Pojedynczy neuron sieci `LSTM` opisuje się tak:

x_t	wejście w chwili t
$X_t = [h_{t-1}, x_t]$	konkatenacja x_t i h_t
$f_t = \sigma(W_f \cdot X_t + b_f)$	bramka zapomnienia
$i_t = \sigma(W_i \cdot X_t + b_i)$	bramka wejścia
$o_t = \sigma(W_o \cdot X_t + b_o)$	bramka wyjścia
$\bar{C}_t = tanh(W_c \cdot X_t + b_c)$	nowy kandydat na stan
$C_t = f_t \circ C_{t-1} + i_t \circ \bar{C}_t$	nowy ukryty stan komórki
$h_t = o_t \circ tanh(C_t)$	wyjście w chwili t



Rysunek 2.11: Graf przedstawiający zależności w opisie LSTM

Parametrami są wagi przy bramce zapomnienia W_f i b_f, wagi przy bramce wejścia W_i, b_i, wagi przy bramce wyjścia W_o, b_0, wagi, którymi generowany jeset nowy kandydat na stan W_c, b_c. Jeżeli wektor wejściowy x_t ma rozmiar N, a stan ukryty C_t oraz wyjście sieci h_t to wektory rozmiaru M, to X_t ma rozmiar $M + N$, wyjścia na czterech bramkach mają rozmiar M. Macierze parametrów na bramkach mają rozmiar $M + N \times M$ oraz M. Sieć można wytrenować z użyciem wstecznej propagacji w czasie[7].

2.3 Istniejące systemy weryfikacji mówcy

2.3.1 Metody niezależne od tekstu

Jednym ze sposobów weryfikacji mówcy stosowanym od dawna jest `GMM-UBM`. Polega on na utworzeniu modelu tła (*universal background model*), modelu mikstur gaussowskich i dopasowaniu ich do ramek `MFCC` z nagrań treningowych. Dla każdego mówcy tworzony jest osobny model przez `MAP`-adaptację modelu tła do nagrań danego mówcy. W [8] następnie liczona jest różnica między logarytmem prawdopodobieństwa wygenerowania nagrania przez model dla mówcy i przez model tła. Różnica ta porównywana jest z progiem i na tej podstawie podejmowana jest decyzja o weryfikacji. Podobnie można weryfikować treść nagrania, przez adaptację modelu tła do nagrań konkretnej treści. Jeżeli treść wszystkich wypowiedzi jest znana i jest to zbiór zamknięty, to zamiast porównywać model dla jednej treści z modelem tła, można wykorzystać *mean norm* lub *max norm*, to znaczy porównać

wynik weryfikowanego modelu ze średnim wynikiem pozostałych lub z najlepszym wynikiem pozostałych. Niezależnie weryfikowany jest mówca i treść wypowiedzi.

Autorzy testują wiele modeli, w tym zarówno do weryfikacji mówcy, jak i treści wykorzystują GMM-UBM z 512 miksturami. Jako cech używają 19 MFCC z delta i delta-delta cechami. MAP-adaptację przeprowadzono z parametrem *relevance factor* 3. Do weryfikacji mówcy przy adaptacji korzystają z nagrań tego mówcy, a do weryfikacji treści nagrań tej samej treści.

Podejście z miksturami można rozwinąć. Bardzo popularnym rozwinięciem są i-wektory, w których zamiast liczyć log-prawdopodobieństwo liczone są wektory charakteryzujące mówcę, a następnie do odpowiedzenia na problem weryfikacji wykorzystywane jest LDA (*Linear Discriminant Analysis*), które jest metodą nadzorowanego klastrowania.

Na przykład w [8] zastosowano i-wektory do weryfikacji mówcy. Trenują GMM-UBM z 512 miksturami, osobno dla każdej płci. I-wektor ma rozmiar 400, i-wektor dla danego mówcy jest średnią z wszystkich i-wektorów dla jego nagrań. Długość i-wektorów jest następnie normalizowana. Do weryfikacji użyta jest gPLDA. (*gaussian probabilistic linear discriminant analysis*)

2.3.2 Metody zależne od tekstu

Powyższe systemy nie uwzględniają kolejności w jakiej ramki powinny być w wypowiedzi. W problemie weryfikacji zależnej od tekstu jest to jednak możliwe i powinno pozwolić na poprawienie skuteczności systemu. W GMM-UBM te same miksture uczestniczą niezależnie od czasu, a i-wektory są uśredniane dla całej wypowiedzi. Na szczęście dzięki użyciu HMM lub DTW możliwe jest uwzględnienie tych informacji[12].

W pracy [12] pracy utworzono kilka modeli HMM z emisjami GMM. Autorzy osobno zajmują się problemem weryfikacji mówcy i weryfikacji treści. Używany jest zbiór RedDots.

Najpierw tworzony jest tam HMM dla każdej frazy w ten sposób, że emisje modelu są inicjalizowane przez skopiowanie parametrów UBM, a następnie dotrenowane z użyciem wszystkich nagrań treningowych z pewnym zdaniem. Następnie dla każdego mówcy model danego zdania jest kopiowany i MAP-adaptowany do jego nagrań tego zdania. Przy weryfikacji mówcy za pomocą algorytmu Viterbiego obliczany jest logarytm prawdopodobieństwa, że nagranie pochodzi od danego mówcy oraz że nagranie pochodzi od HMM tła. Ocena nagrania to różnica między tymi wynikami i system decyduje o weryfikacji porównując tę różnicę z pewnym progiem.

Następnie tworzą drugi podsystem do weryfikacji treści. Dzielą nagrania tej samej frazy na pewną liczbę segmentów i tworzą dla każdej frazy HMM typu *left-to-right*. Jako, że nagrań jest mało, to trudno wytrenować model, by był w stanie wykryć, które ramki zawierają jakie fony, dlatego segmenty dzielą na sztywno i właściwie ręcznie dopasowują mikstury do ramek. Weryfikacja frazy przebiega w ten sposób, że liczą średnią różnicę w log-prawdopodobieństwie między pochodzeniem frazy z odpowiedniej dla mówcy i segmentu mikstury, a pochodzeniem

ramki z GMM-UBM, zamiast HMM.

Użyli 19 MFCC z delta i delta-delta cechami. GMM-UBM ma 512 mikstur. MAP adaptacja dotyczy tylko wektora średnich. Liczba segmentów w drugiej części to 4 i 8, lecz 8 nie poprawiło im wyników. Liczba stanów HMM w pierwszej części to 4 lub 8 i 8 lekko poprawiło im wyniki.

Podobny pomysł jest wykorzystany w jednym z systemów do weryfikacji treści w [8], gdzie autorzy wytrenowali 512 miksturowy GMM UBM. Ich HMM jest typu *left-to-right* z 5 stanami. Naukę rozpoczynają dzieląc jedno nagranie na 5 równych segmentów. Każdy z 5 GMM emitujących MAP-adaptują z modelu tła korzystając z próbek odpowiedniego segmentu. Pozostałe nagrania są dzielone na segmenty korzystając z tego modelu. Następnie trenują ostateczne modele dla każdej wypowiedzi MAP-adaptując model tła korzystając z wszystkich nagrań, zakładając podział na ramki jak w poprzednim kroku.

W [11] model HMM został bardziej rozwinięty. Wykorzystano wiedzę o treści wypowiedzi w postaci transkrypcji. Mając transkrypcje można stworzyć i dopasować wiele GMM do rozpoznawania różnych głosek, zamiast używać jednej mikstury z bardzo dużą liczbą elementów. Można wykorzystać model stworzony na potrzeby rozpoznawania mowy. Taki model ma stany ukryte, które mają odpowiadać fonemom języka. Na przykład angielski ma 39 fonemów, czemu odpowiadałoby 39 stanów, lecz powszechne jest rozszerzenie modelu do 3 stanów na jeden fonem, co ma pozwolić osobno określić cechy dla początku, środka i końca fonemu. Bardziej złożone systemy, jak HTK[21] , dodatkowo pozwalają na zbudowanie tzw. modelu trójfonowego (*triphone*), co oznacza, że nie używają trzech stanów na fonem, lecz przypisują inne stany w zależności od fonemu bezpośrednio przed i bezpośrednio po danym fonemie. Powoduje to duży wzrost wymaganych stanów, to znaczy z 3×39 do 3×39^3, dlatego te systemy dodatkowo łączą bardzo podobne trójki, by dzieliły ze sobą mikstury. Określenie podobieństwo bazuje na wiedzy o podziałach głosek i tym jak układ głosowy przechodzi między nimi. W [11] użyto modelu jednofonowego (*monophone*), choć użyli też trójfonowego w nieco innym przypadku.

Do modelowania emisji używa się GMM. Jako, że stanów jest dużo więcej niż w GMM-UBM lub w HMM, w którym wypowiedź jest podzielona na 4-5 części, to w opisanym przykładzie tego modelu zastosowane jest 8 mikstur zamiast 512 lub 1024. Wyjątkiem są ramki, gdzie spodziewana jest cisza (początek, koniec i przerwa między wyrazami), tam użyto 16 mikstur. Warto zauważyć, że we front-endzie użyto 20 MFCC wziętych z banku filtrów rozmiaru 40 i dołączono cechy delta i delta-delta.

Mając taki model można użyć algorytmu Viterbiego lub *forward-backward*, by przypisać ramkom z nagrania fonemy, które na nich występują. Dla każdego mówcy przez MAP-adaptację można stworzyć ich własny zestaw mikstur do każdego stanu i potem weryfikować mówcę przez porównanie logarytmu prawdopodobieństwa pochodzenia nagrania od mówcy, a pochodzenia od modelu do rozpoznawania mowy, tak jak w opisanej wcześniej GMM-UBM.

Zamiast tego na przykład w tej pracy [22] użyto HMM, lecz dokonując rejestracji i weryfikacji z wykorzystaniem metody i-wektorów. W przeciwieństwie do poprzedniej pracy, tu HMM jest jednofonowy. Autorzy przetestowali i-wektory rozmiaru 400

oraz 600 i dokonują weryfikacji przez obliczenie podobieństwa cosinusowego.

2.3.3 Metody wykorzystujące sieci neuronowe

W [10] połączono ideę i-wektorów z DNN. Autorzy zauważyli, że do obliczenia i-wektorów można użyć dowolnego modelu, który definiuje K klas i pozwala obliczyć prawdopodobieństwo przynależności do klasy dla ramki $P(\theta_k|x)$. Tutaj użyto DNN wytrenowanej na nagraniach z transkrypcją do rozpoznawania mowy. Sieć ma 7 warstw, 1200 wymiarów w warstwach ukrytych, w ostatniej warstwie ma 3450 wymiarów i funkcję aktywacji softmax. Wyjścia interpretowane są jako przynależności do trójek fonów. Trójki te pełnią rolę klas, a wartość na wyjściu traktowane są jako wymagane prawdopodobieństwo, dzięki czemu DNN może zastąpić GMM przy obliczaniu statystyk w metodzie i-wektorów. Weryfikacja następuje z wykorzystaniem gPLDA.

W [17] również użyta została DNN do rozpoznawania mowy i zaadaptowana do weryfikacji mówcy. Wykorzystana sieć ma architekturę TDNN (*Time-delay neural network*). Ma 6 warstw *feed-forward*, na wejściu przyjmuje kilka kopii sygnału wejściowego z różnym przesunięciem w czasie, wagi w warstwach przetwarzających kopie są współdzielone. Takie przetwarzanie kilku kopii jest wykonywane na 4 warstwach, z różnym przesunięciem i w efekcie sieć ma uwzględniać kontekst aż 13 ramek z przeszłości i 9 ramek z przyszłości. Ostatnia warstwa kończy się softmaxem z wyjściami odpowiadającymi 5297 trójkom fonemów. Ciekawe są następne kroki autorów, gdyż trenują GMM rozmiaru również 5297 i dopasowują mikstury w ten sposób, że dla k-tej mikstury estymują jej wektor średnich i macierz kowariancji (pełną) do danych ważonych przynależnością do k-tej trójki fonemów wyznaczoną przez TDNN. Następnie używają GMM standardowo do i-wektorów (600 wymiarowych) i PLDA. Autorzy zauważają również, że sieć i GMM mogą używać różnych wektorów cech, co może być korzystne, gdyż rozwiązują różne problemy - klasyfikacji fonemów oraz weryfikacji mówcy.

W [20] używany jest DNN wytrenowany do rozpoznawania fonemów. Najpierw autorzy wytrenowali GMM-HMM, by wykorzystać go do przypisywania do ramek sygnału jaki jest w nich fonem. Następnie na tych danych, tym razem już oznakowanych przez GMM-HMM, trenowany jest DNN do dyskryminacji fonemów. Autorzy używają jednocześnie nagrań z angielskiego i chińskiego, by rozpoznawać więcej fonemów i uzyskać model działający dla wypowiedzi w różnych językach. Sieć jest wykorzystywana w roli ekstraktora cech. Wektor wartości na wyjściu przedostatniej warstwy tej sieci jest wykorzystywany jako tzw. *bottleneck features*. Warstwa ta ma nieco mniej neuronów, by uzyskać tam niskowymiarową reprezentację ramek na wejściu. Jako, że sieć została wytrenowana do rozpoznawania fonemów, to cechy te powinny wydajnie uwzględniać informację niezbędne właśnie do tego celu, choć nie jest jasne, czemu miałyby być tam informacje charakteryzujące mówcę. GMM są trenowane na tych cechach, tak jak w innych systemach są trenowane na MFCC i używane do ekstrakcji i-wektorów. Weryfikacja następuje przez gPLDA.

Zaletą DNN jest możliwość operowania na większych wektorach, dlatego na wejściu sieć otrzymuje oprócz jednej ramki też po 5 ramek sąsiadujących przed i po,

łącznie 11, do tego każda ramka zawiera 40 współczynników z filtrów w częstotliwości melowej oraz 80 cech delta i delta-delta. Sieć ma pięć warstw ukrytych, wszystkie mają wyjścia rozmiaru 1200, z wyjątkiem ostatniej, która ma rozmiar 39 i generuje *bottleneck features*. Na wyjściu autorzy rozważają różny rozmiar wektora, jednak wszystkie rozważone parametry są ponad 2000 i są zależne od budowy `HMM-GMM` i tego ile trójek fonemów rozróżnia. Tworzone później `GMM` ma 2048 mikstur, i-wektory są rozmiaru 400, `gPLDA` operuje w 200 wymiarach. Warto zauważyć, że ostateczny wynik osiągnięty przez autorów ma wyższą `EER` niż *baseline* w postaci `GMM-UBM`.

W pracy [6] opisany jest model stworzony przez Google do weryfikacji mówcy. Wszystkie wykorzystane w nim nagrania zawierały fazę "Ok Google", a zatem zbiór ten jest uboższy wobec poprzednio wymienionych rozwiązań. Zaproponowane modele, bazujące na sieci neuronowej, charakteryzują się tym, że w całości rozwiązują problem weryfikacji i są trenowane w całości, podczas gdy poprzednio wymienione modele były rozbite na osobno działające podsystemy. Na przykład `DNN` z poprzednich prac były trenowane do rozpoznawania mowy i zaadaptowane do rozpoznawania mówcy albo w przypadku `GMM` celem jest modelowanie dystrybucji cech, nie weryfikacja. Takie rozbicie skutkuje nieoptymalnym działaniem i zwiększa złożoność obliczeń.

Pierwsza sieć, którą zaproponowali autorzy, to `DNN` z lokalnie połączoną pierwszą warstwą i w pełni połączonymi kolejnymi warstwami. Warstwy używają funkcji aktywacji `ReLU`, z wyjątkiem ostatniej z *softmaxem*. Wektor na wyjściu ma wymiar na każdego mówcę. Po wytrenowaniu sieci, dla każdej ramki z nagrania obliczane są wartości na wyjściach ostatniej ukrytej warstwy, są one uśredniane dla nagrania i wynik nazywany jest d-wektorem. Pierwszy system działa przez obliczenie d-wektora na nagraniach rejestracyjnych i uśrednieniu, by uzyskać d-wektor opisujący mówcę. Dla nagrań testowych również jest liczony d-wektor i weryfikacja następuje przez obliczenie cosinusowego podobieństwa między tym d-wektorem i d-wektorem mówcy. Warto zauważyć, że w przeciwieństwie do *bottleneck features*, tutaj sieć do ekstrakcji cech jest wytrenowana do rozpoznawania mówców, a nie fonemów, więc można się spodziewać, że zachowuje informacje charakteryzujące mówcę. Przy *bottleneck features* można podejrzewać, że cechy mówców są wręcz odrzucane, gdyż w problemie rozpoznania fonemów są przeszkodą.

Druga sieć przyjmuje nagranie testowe oraz kilka nagrań rejestracyjnych i zwraca na wyjściu jedną liczbę reprezentującą wynik weryfikacji. Sieć ma architekturę `LSTM`. Brane pod uwagę jest tylko wyjście sieci dla ostatniej ramki nagrania, przy czym jako, że to sieć rekurencyjna, to wynik może zależeć od wszystkich ramek poprzedzających. Reprezentacja mówcy jest tworzona przez uśrednienie wyjścia `LSTM` dla nagrań rejestracyjnych. Nagrania testowe również przechodzą przez sieć i są porównywane z reprezentacjami mówców przez obliczenie podobieństwa cosinusowego. Autorzy raportują, że dopiero dla bardzo dużej liczby danych to rozwiązanie osiągnęło wysoką skuteczność.

Rozdział 3

Użyte narzędzia i technologie

3.1 Zbiór danych

3.1.1 RedDots

W pracy wykorzystano zbiór `RedDots`[9]. Jest to zbiór nagrań głosowych przeznaczony do problemu weryfikacji mówcy. Jest dostosowany do problemu weryfikacji zależnej od tekstu, gdyż transkrypcja każdej wypowiedzi jest znana, a mówcy wypowiadają w nim wielokrotnie zdania o tej samej treści. Zbiór został zebrany od ochotników z całego świata przez Internet, co sprawia, że choć treści wypowiedzi są po angielsku, to mówcy wypowiadają słowa z różnym akcentem. Nagrania były tworzone w niekontrolowanych warunkach i często są zakłócone. Wśród ochotników znalazło się 49 mężczyzn i 13 kobiet z 21 krajów.

Gromadzenie danych przebiegało w cotygodniowych sesjach. Autorzy mieli plan zebrać po 52 sesji od każdego uczestnika, czyli zbierać je przez cały rok, ale w praktyce liczba sesji na mówcę jest bardzo różna. Łącznie przeprowadzono 473 sesji mężczyzn i 99 sesji kobiet, gromadząc 15306 nagrań. W każdej sesji uczestnik miał za zadanie nagrać 24 wypowiedzi.

- 10 wypowiedzi miało stałą treść dla wszystkich uczestników i we wszystkich sesjach. Stanowią one podstawę pierwszego zadania testowego zdefiniowanego przez twórców. Ich treść to zdania o identyfikatorach $31 - 40$ z innego zbioru `TIMIT`[3].

- 10 wypowiedzi miało stałą treść we wszystkich sesjach, lecz unikalną dla każdego uczestnika. To znaczy każdemu uczestnikowi przydzielono na stałe oddzielny zestaw 10 zdań. Nagrania te są podstawą drugiego zadania.

- 2 wypowiedzi o treści nie zmieniającej się między sesjami, lecz wybranej przez uczestnika. Powoduje to w porównaniu z poprzednim punktem ryzyko, iż użytkownik wybierze krótkie hasło. Do tego niektórzy użytkownicy wybierali zdania w innym języki niż angielski. Pozwala to zbadać efekty pozostawienia wyboru hasła użytkownikom i jest przedmiotem trzeciego zadani.

- 2 wypowiedzi o treści unikalnej w całym zbiorze. Na ich treść składają się wycinki Wikipedii. Są wykorzystywane w czwartym zadaniu, które sprawdza skuteczność systemu na nagraniach o niezarejestrowanej wcześniej treści.

Wszystkie nagrania są w nieskompresowanym formacie `pcm`, który oznacza, że wartości kolejnych próbek w czasie są zapisane jedna po drugiej. Częstotliwość próbkowania nagrań to 16kHz. Do tego zbiór zawiera plik z transkrypcjami wszystkich wypowiedzi.

Poza tymi nagraniami w zbiorze są pliki z definicjami czterech problemów. W skład każdej definicji wchodzi zestaw nagrań rejestracyjnych (*enrollment*) i zestaw nagrań weryfikacyjnych. (*trial*) Problemy te to:

1. Wszyscy mówcy wypowiadają te same 10 zdań. Trzy nagrania na zdanie na osobę przeznaczone są do rejestracji. Podana jest też pewna liczba nagrań weryfikacyjnych. Każde nagranie weryfikacyjne jest wielokrotnie podawane do systemu i za każdym razem oczekiwany jest inny mówca i zdanie. Dla każdego nagrania sprawdzane są wszystkie kombinacje znanych mówców i zdań z tej części. Do tego niektórzy mówcy nie są obecni w zbiorze rejestracyjnym i występują wyłącznie w zbiorze testowym. Stanowią oni test, jak system sprawdza się przy otwartym zbiorze osób.

2. Wszyscy mówcy wypowiadają po 10 zdań, każdy mówca ma swój unikalny zestaw. Podobnie jak powyżej, w zbiorze rejestracyjnym są po trzy nagrania na osobę na zdanie. Reszta nagrań służy do testów i niektórzy mówcy w testach nie występują w zbiorze rejestracyjnym. Jako że treść jest wyłączna dla użytkowników, to w przypadku, gdy mówca się nie zgadza, treść też się nie zgadza.

3. Wszyscy mówcy wypowiadają po 2 zdania, każdy mówca ma unikalny zestaw. Mówcy sami decydują o treści. Przypadek podobny do poprzedniego, lecz wypowiedzi jest mniej i zdarzają się wypowiedzi w innych językach.

4. (a) Wariant niezależny od tekstu. Zestaw rejestracyjny zawiera po sześć nagrań na osobę o unikalnej treści. Zestaw testowy zawiera nagrania, również o unikalnej treści.

   (b) Wariant zależny od tekstu. Zestaw rejestracyjny zawiera nagrania z części 1-3. Zestaw testowy również zawiera nagrania z poprzednich części oraz dodatkowo nagrania z treścią unikalną w całym zbiorze.

W przejrzanych pracach zwykle zadania 2 i 3 były ignorowane. Możliwe, że budzą mniejsze zainteresowanie przez fakt, że różni mówcy zawsze różnią się treścią nagrania. Czyni to te części prostszymi, niż pozostałe dwie. Dodatkowo nagrań w 3 części jest mniej. Sprawdzenie jak system działa w sytuacji, gdy ktoś podszywa się pod mówcę i zna jego hasło jest bardzo ważne w praktycznych zastosowaniach. Do tego w niektórych pracach ignorowane były nagrania kobiet, gdyż jest ich kilkakrotnie mniej niż nagrań mężczyzn. Wszystkie testy uwzględniały część 1 i często uwzględniana była część 4.

Zbiór został sporządzony na potrzeby konkursu w 2016 roku. Teraz jest dostępny za darmo pod warunkiem ograniczenia wykorzystania do celów naukowych. Jest warty uwagi, gdyż trudno znaleźć inny zbiór dostosowany do problemu weryfikacji zależnej od tekstu, który byłby publicznie dostępny i darmowy. Istnieje na przykład zbiór `YOHO` oraz zbiór `RSR2015`, które zawierają więcej danych zebranych w lepszych warunkach, lecz wymagają opłaty. Tym niemniej warto o nich pamiętać.

3.1.2 CMUDict

`CMUDict`[19] to zbiór zawierający zapisy fonetyczne ponad 134000 angielskich słów. Rozróżnione jest w nim 39 fonemów typowych dla angielskiego. Do tego w samogłoskach wyróżniony jest akcent. Słowa mogą mieć więcej niż jeden zapis fonetyczny.

Zbiór został wykorzystany do zamiany transkrypcji z `RedDots` na ciągi fonemów. Stała za tym taka idea, że fonemy będą stanowić lepsze klasy w systemach rozpoznawania mówcy, jak `HMM-GMM`, niż litery, którym mogą w angielskim odpowiadać przeróżne fonemy. Niestety niektóre wypowiedzi z części trzeciej, wybrane przez użytkowników, były w innych językach i dla nich operacja zawiodła. Dlatego w systemach, które bazują na fonetycznych transkrypcjach, są one odrzucane. Nie jest to problemem jeżeli zupełnie zignoruje się trzeci problem.

3.2 Sprzęt

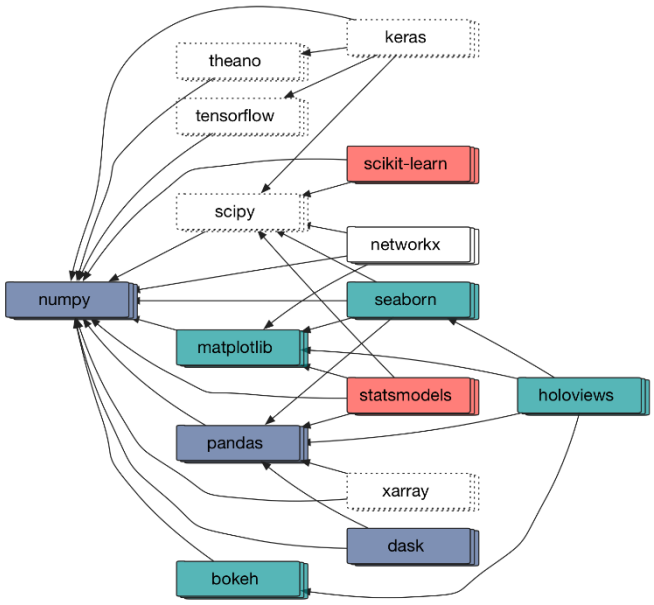
Utworzone w ramach pracy programy tworzono i wykonywano na maszynie udostępnionej w Centrum Technologii Informatycznych Politechniki Łódzkiej. 64GB pamięci i 40 rdzeniowy procesor mocno przyśpieszył pracę oraz umożliwił załadowanie całego zbioru do pamięci. Praca nie wymaga jednak żadnych niestandardowych urządzeń lub warunków, odtworzenie eksperymentów jest możliwe na zwykłym komputerze.

3.3 Technologie

3.3.1 Język programowania

Programy utworzone na potrzeby pracy stworzono w języku `Python`. Wybrano go ze względu na to, iż zdominował popularnością inne języki w dziedzinie przetwarzania danych. Wiąże się to z tym, iż istnieje dużo dojrzałych bibliotek, takich jak `scikit-learn` oraz `TensorFlow`, zawierających implementacje metod, które wykorzystano w pracy. Poza tym bezproblemowo integruje się środowiskiem ze `Jupyter`, które jest bardzo wygodne przy zdalnej pracy na klastrze. Umożliwia ono pracę interaktywną, co jest cenne, gdy zachodzi potrzeba przetestowania wielu rozwiązań i reagowania na wyniki.

3.3.2 Biblioteki



Rysunek 3.1: Ekosystem narzędzi Pythonowych przedstawiony na prezentacji z PyData Warsaw 2017[18]

Program wykorzystuje popularny *stack* związany z językiem `Python` składający się z:

- `numpy` dostarczający wydajną i uniwersalną implementację macierzy wielowymiarowej i operacji na niej. Inne pakiety zależą od tej biblioteki i wykorzystują macierze w swojej implementacji.
- `pandas` przede wszystkim udostępnia *DataFrame*, to znaczy obiekt do przechowywania i analizowania danych tabelarycznych, przypominający w funkcji arkusz kalkulacyjny. W porównaniu z macierzami z `numpy`, *DataFrame* mogą składać się z kolumn różnych typów i o różnej semantyce. Przy operowaniu na nich często zachodzi potrzeba selekcji, grupowania bądź agregowania, liczenia statystyk opisowych wybranych kolumn.
- `scipy` operuje na macierzach `numpy` i dostarcza wiele algorytmów numerycznych, optymalizacyjnych, z algebry liniowej, statystycznych, itd. W tej pracy wykorzystano przede wszystkich moduł z metodami przetwarzania sygnałów.
- `scikit-learn` zawiera algorytmy uczenia maszynowego. W pracy przydatna była przede wszystkim implementacja mikstur gaussowskich oraz obliczenia krzywej `ROC`.
- `matplotlib` pozwala na generowanie statycznych wizualizacji np. spektrogramów, krzywych `ROC`. Z jej pomocą wygenerowane zostały także ilustracje

konceptów z teorii przetwarzania mowy zaprezentowane w rozdziale 2. Warto wspomnieć, że niektóre z tych ilustracji zostały wygenerowane z użyciem `bokeh` i `holoviews`.

Jeśli chodzi o przetwarzanie dźwięku, to wykorzystany został pakiet `python-speech-features` do obliczenia `MFCC` oraz cech delta i delta delta dla nagrań ze zbioru `RedDots`. Przed przetworzeniem cech używany jest również pakiet `webrtcvad` do przeprowadzenia *voice activity detection*, tzn. wykrycia ramek o niskiej energii. Znalezione ramki z początku i końca są usuwane.

Przy pracach nad ukrytymi modelami Markowa wypróbowane zostały pakiety `hmmlearn` i `pomegranate`. Niestety sprawiały wiele problemów numerycznych podczas dopasowywania modelu do nagrań. Ostatecznie zrezygnowano z nich i zastąpiono własną implementacją.

W pracy użyty został również `TensorFlow`. Biblioteka ta dostarcza implementacji tensora - pozornie macierzy wielowymiarowej jak z `numpy`. Jest jednak dostosowana do potrzeb implementacji algorytmów uczenia maszynowego. Stosuje model opóźnionej ewaluacji, podobnie jak `Spark` lub `dask`. To znaczy, że osobno definiuje się w niej operacje do wykonania przez zbudowanie grafu i osobno przekazuje go do interpretacji. Ten sposób wykonania umożliwia dokonanie optymalizacji po zdefiniowaniu obliczeń i przed wykonaniem oraz obchodzi się tym samym wadę Pythona - relatywnie powolne wykonywanie kodu. `Tensorflow` zawiera komponenty potrzebne do zdefiniowania grafu, w tym wysokopoziomowe z pakietu `tf.estimator`. Umożliwia automatyczne obliczanie pochodnych na potrzeby wstecznej propagacji. Pozwala na wykonywanie operacji na `GPU` oraz rozproszenie obliczeń.

W pracy `TensorFlowa` jest wykorzystany do wczytania modelu stworzonego przez organizację `Mozilla`. Model ten powstał w oparciu o pracę `DeepSpeech`[5], opisującą komercyjny model do rozpoznawania mowy rozwinięty przez `Baidu`. Ten model jest wczytywany w sesji `TensorFlowa` i używany do dopasowania ramek do klas.

3.3.3 Narzędzia

W pracy wykorzystany został `Docker`. Uruchamiany jest w nim obraz `tensorflow/tensorflow:1.4.0-gpu-py3`, z przygotowanym środowiskiem z `TensorFlowem`. Umożliwia to szybką instalację tego pakietu, która ogólnie nie jest trywialna. W kontenerze uruchamiany jest serwer `Jupyter` i z jego wykorzystaniem przebiegła większość prac.

`Jupyter` stanowi rozwinięcie starszych narzędzi programowania interaktywnego jak `IPython` czy zwykły `Pythonowy` interpreter. W sesjach interaktywnych występuje taki problem, że instrukcje trzeba wprowadzać linijka po linijce, a wpisany kod jest zapisywany w ograniczonej historii. Przeciwieństwem sesji są skrypty, które pozwalają wygodnie manipulować kodem oraz są trwałe i powtarzalne, lecz zupełnie nieinteraktywne i po każdej zmianie wymagają ponownego wykonania. `Jupyter`

łączy dobre strony obu podejść i pozwala tworzyć skrypty, które można blokami wykonywać w interaktywnej sesji, tzw. notesy. (*notebook*, format `ipynb`)

`Jupyter` działa jako serwer `HTTP` i dostarcza przeglądarkowe środowisko do edycji kodu. Jest ono ubogie w funkcje, przynajmniej w porównaniu z rozwiniętymi środowiskami programistycznymi. Fakt, że program działa jako serwer ma taką zaletę, iż można go uruchomić zdalnie na komputerze dysponującym dużymi zasobami obliczeniowymi oraz bezpośrednim dostępem do danych, a następnie pracować na nim zdalnie.

Dużym atutem jest wsparcie dla bibliotek jak `pandas`, `matplotlib` czy `bokeh`. *DataFrame* są automatycznie wyświetlane jako tabele, a generowane wizualizacje są bezpośrednio wyświetlane między blokami notesu. Jest też możliwość tworzenia interaktywnych elementów jak przyciski czy pola tekstowe i w reakcji na zmiany ich stanu można wykonywać kod w interaktywnej sesji.

Poza tym warto wspomnieć, że wykorzystano `gita` do wersjonowania pracy, zarówno *notebooków*, pakietów jak i tekstu tego dokumentu. Do zarządzania pakietami wykorzystano `pip`, ale być może lepszym wyborem są nowsze narzędzia jak `conda` lub `pipenv`.

Rozdział 4

Stworzony system weryfikacji mówcy

4.1 Przetworzenie danych

Nagrania `RedDots` zostały przetworzone w ten sposób:

- Przeprowadzana jest procedura `VAD` (*voice activity detection*), czyli wyodrębnianie fragmentów nagrania zawierających sygnał mowy, za pomocą pakietu `webrtcvad`. Użyty parametr agresji 2. Z każdego nagrania zostały usunięte ramki bezpośrednio z początku i z końca, które zostały uznane za ciche.

- Dla każdego obciętego nagrania obliczone zostały `MFCC` korzystając z metody z pakietu `python-speech-features`. Analizę przeprowadzono w oknach 25ms z 10ms skokiem. Użyto okna Hamminga. Utworzono 26 filtrów pokrywających częstotliwości nagrania i zachowano z nich 13. Użyto filtru *preemphasis* z parametrem 0.97. Wynikiem jest 13 współczynników.

- Dla każdej ramki obliczono cechy delta i delta-delta z użyciem pakietu `python-speech-features`. Cechy te obliczono biorąc pod uwagę sąsiedztwo 4 ramek przed i 4 ramek po danej ramce.

- Bardzo krótkie nagrania, składające się z mniej niż 85 ramek, zostały odrzucone. Kilka nagrań miało nawet długość 1 ramki. Próg został wyznaczony przez zbadanie jakiej długości są nagrania. Między 85 i krótszymi jest skok, więc uznano je za anomalie.

Wykorzystano słownik `CMUDict` do zamiany wszystkich transkrypcji z `RedDots` na listy 39 fonemów używanych w języku angielskim. Zignorowano przy tym akcenty. Części nagrań nie udało się zamienić, gdyż zawierały treść w innym języku, i one nie będą używane w modelach wykorzystujących fonemy.

4.2 Wykorzystane modele

W ramach pracy utworzone zostały trzy modele do weryfikacji mówcy. Zostały one tutaj opisane.

4.2.1 Model GMM-UBM

Mikstury gaussowskie z uniwersalnym modelem tła

System `GMM-UBM` jest wzorowany na tym opisanym w pracy [8]. Posłużono się również parametrami z tej pracy.

Wpierw zebrano wszystkie dane ze zbioru `RedDots`, które nie są użyte w żadnym teście. Dopasowano do nich model tła, który jest miksturą 512 dystrybucji normalnych. Założono, że cechy są nieskorelowane i macierz kowariancji w miksturach jest diagonalna.

W ramach zapisów model tła jest kopiowany i `MAP`-adaptowany z *relevance factor* równym 3.0 dla nagrań rejestracyjnych danego mówcy. Dodatkowo dla każdej grupy nagrań o tej samej treści również tworzona jest kopia modelu tła i `MAP` adaptowana w ten sam sposób do nagrań z tej grupy.

W czasie testów dla każdego nagrania liczone jest prawdopodobieństwo, że nagranie pochodzi z modelu tła, z `GMM` mówcy, który jest oczekiwany w weryfikacji i z `GMM` zdania, które jest oczekiwane. Na podstawie tych wartości liczone są prawdopodobieństwa, że mówca i treść są poprawne jako stosunek prawdopodobieństw.

$$S_{sp}(x) = \frac{P(\theta_{sp}|x)}{max(P(\theta_{sp}|x), P(\theta_{UBM}|x))}$$

$$S_{st}(x) = \frac{P(\theta_{st}|x)}{max(P(\theta_{st}|x), P(\theta_{UBM}|x))}$$

$$S_{both}(x) = S_{sp}(x)S_{st}(x)$$

4.2.2 Model HMM-GMM

Ukryty model Markowa z emisjami w postaci mikstur gaussowskich

Ten model jest wzorowany na modelu opisanym w pracy [11]. Wykorzystuje system rozpoznawania mowy do określenia, które ramki zawierają które fonemy. System ten to ukryty łańcuch Markowa, w który każdy fonem jest reprezentowany przez trzy stany. Został użyty model monofonemowy. Dla wypowiedzi w zbiorze znany jest ciąg fonemów, gdyż `RedDots` zawiera transkrypcje, które następnie zostały zamienione na fonemy przez odszukanie odpowiednich słów w słowniku `CMUDict`. Dla każdej wypowiedzi budowany jest `HMM` przez złączenie odpowiednich trójek stanów zgodnie z kolejnością fonemów w wypowiedzi. Tworzony jest łańcuch z przejściami typu *beads-on-string*.

Dla każdego mówcy i tła budowany jest zestaw mieszanin GMM, po jednej na stan HMM, a zatem po trzy na fonem. Każda mikstura składa się z 8 dystrybucji z diagonalną macierzą kowariancji.

Parametry podsystemu rozpoznawania mowy są wyznaczane tak, że najpierw do wszystkich danych dopasowywana jest mała 8 składnikowa mikstura tła. Następnie budowane są wszystkie trójki GMM odpowiadające fonemom. Potem dla każdej wypowiedzi budowany jest model Markowa przez połączenie odpowiednich trójek zgodnie ze znanym ciągiem fonemów danej wypowiedzi. Korzystając z HMM i algorytmu Viterbiego ramki nagrania są przyporządkowywane do części fonemów. Algorytm jest zmodyfikowany, by dawał wyższe prawdopodobieństwo podziałowi, w którym każdemu ukrytemu stanowi odpowiada ta sama liczba ramek. To samo jest robione dla innych nagrań. Następnie parametry trójek GMM są dopasowywane przez EM do ramek, które zostały im przypisane. Po ustaleniu parametrów mikstur, ramki są przypisywane ponownie z użyciem algorytmu Viterbiego i proces jest ponawiany.

Rejestracje wyglądają tak, że dla każdego mówcy jest budowany jego zestaw trójek GMM. Wypowiedzi rejestracyjne danego mówcy są przyporządkowywane częściom fonemów z użyciem algorytmu Viterbiego przez opisane HMM. Używając tego przyporządkowania mikstury tła są MAP adaptowane do nagrań danego mówcy.

Testy wyglądają podobnie. Najpierw z użyciem systemu rozpoznawania mowy i algorytmu Viterbiego znajdowane jest, które ramki zawierają które części fonemów. Następnie dla każdej ramki obliczany jest logarytm prawdopodobieństwa, iż pochodzi ona z odpowiedniej mikstury mówcy i odpowiedniej mikstury tła. Sumując je uzyskiwany jest wynik dla mikstur mówcy i mikstur tła. Wynik weryfikacji mówcy obliczany jest jako stosunek tych prawdopodobieństw. Treść nagrania nie jest osobno sprawdzana.

$$S_{sp}(x) = exp(\sum_{x_i \in x} (logP(\theta_{sp}|x_i) - max(logP(\theta_{sp}|x_i), logP(\theta_{UBM}|x_i))))$$

4.2.3 Model DNN-GMM

Głęboka sieć neuronowa do dopasowania ramek, ocena mikstu-rami gaussowskimi

W tym modelu wykorzystano głęboką sieć neuronową do rozpoznawania tekstu taką jak opisano w pracy [6]. Sieć składa się z pięciu warstw, w tym jednej rekurencyjnej warstwy LSTM. Wytrenowanie tej sieci wymaga niestety ogromnej ilości danych i czasu. Z raportów innych osób wynikało, że na zwykłym komputerze pojedyncza epoka nauki może zająć nawet dwa tygodnie. Z tego powodu wykorzystany został gotowy model stworzony przez organizację Mozilla.

Model ten został wytrenowany na ok. 2500 godzinach nagrań ze zbiorów WSJ, Switchboard i Fisher. Warto zauważyć, że autorzy cytowanej pracy z Baidu poza tymi zbiorami użyli dodatkowo własnego, zamkniętego zbioru 5000 godzin nagrań. Sieć ta przyjmuje kolejne ramki, przy czym muszą one być wstępnie przetworzone

do współczynników MFCC. Dla każdej chwili zwracany jest wektor 28 liczb, które odpowiadają 25 literom alfabetu, przerwie, apostrofowi i szumowi.

Podobnie jak w HMM-GMM dla każdego nagrania znamy treść i z wykorzystaniem systemu rozpoznawania mowy przyporządkowujemy ramki literom. Istotna różnica jest taka, że tutaj system rozpoznawania mowy zwraca tekst, nie sekwencje fonemów, co dla naszych potrzeb jest niepożądane. Gdyby trenować model od podstaw na potrzeby rozpoznawania mówcy warto byłoby zmienić ten aspekt. Do przypisania ramek wykorzystano odpowiednio dostosowany algorytm Viterbiego. Wyjścia sieci są w nim znormalizowane do przedziału $[0.0, 1.0]$ i traktowane jako prawdopodobieństwa emisji danej ramki w kolejnych stanach ukrytych. Stany ukryte są tożsame z literami i algorytm wymusza, by zmieniały się zgodnie ze znaną treścią oraz by wszystkie litery się pojawiły. W przypadku, gdy największe prawdopodobieństwo przypada na szum algorytm pomija ramkę. Wynikiem jest przypisanie do niektórych ramek liter oraz zwrócenie prawdopodobieństwa danej sekwencji - jak jest bardzo małe to prawdopodobnie spodziewana treść i treść na nagraniu się różnią.

Trening wygląda analogicznie jak opisano w HMM-GMM. To znaczy korzystając z przypisania ramek zwróconego przez DNN, tworzy się szereg GMM z 8 składnikami mikstury i macierzami diagonalnymi, po jednym na literę. Następnie wyznaczane są ich parametry, by maksymalizowały prawdopodobieństwo wystąpienia ramek z ich klasy. Rejestracja wygląda tak samo, tylko MAP adaptuje modele tła do mniejszej liczby nagrań rejestracyjnych mówcy.

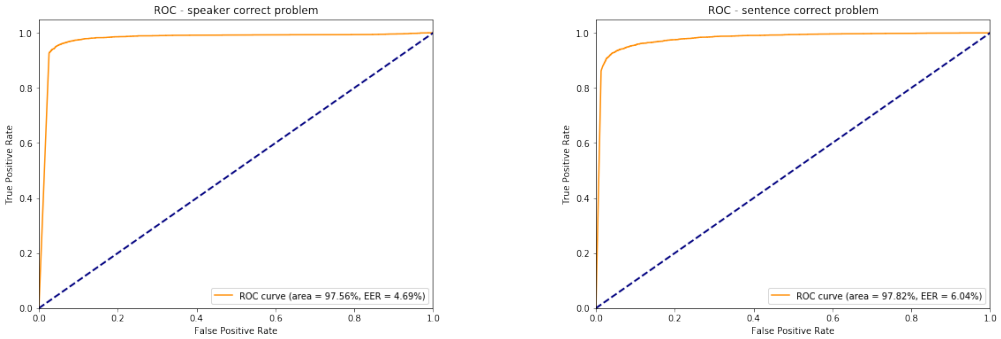
Metoda testowania ponownie polega na policzeniu stosunku prawdopodobieństw między GMM mówcy, a tła. Wynik jest sumowany dla wszystkich ramek, które mają przypisane litery. Weryfikacja treści nagrania bazuje na prawdopodobieństwie zwróconym przez algorytm Viterbiego, czyli prawdopodobieństwie najbardziej prawdopodobnej sekwencji przyporządkowań liter do ramek, w której litery zgadzają się z transkrypcją.

4.3 Wyniki testów

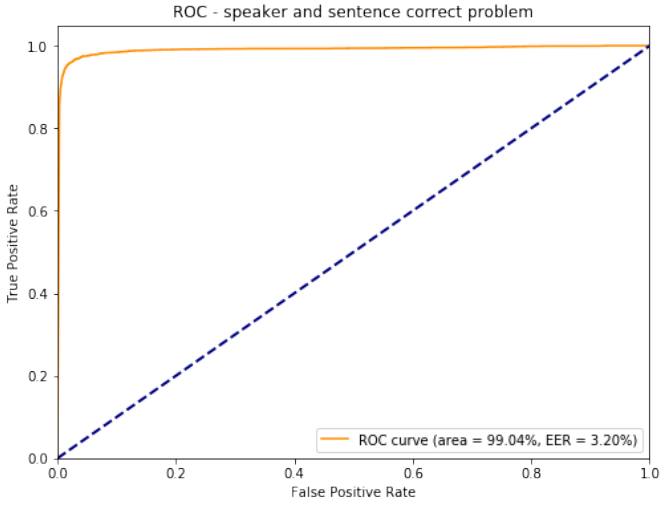
W celu oceny systemów wykonano pierwsze zadanie zdefiniowane w zbiorze RedDots. Każdy z modeli został wykorzystany do zapisów mówców, a następnie do testów. Algorytm wykorzystania modeli wraz ze wszystkimi parametrami zapisany jest w plikach Jupyter, co powinno uczynić wyniki łatwymi do odtworzenia, choć powtórzenie obliczeń może zająć dużo czasu. Wyniki w postaci krzywej ROC, EER i AUC zapisano w poniższej tabeli i zwizualizowano pod tabelą.

Tabela 4.1: Wyniki EER i AUC uzyskane przez modele GMM-UBM, HMM-GMM, DNN-GMM na pierwszym zadaniu zdefiniowanym w zbiorze RedDots

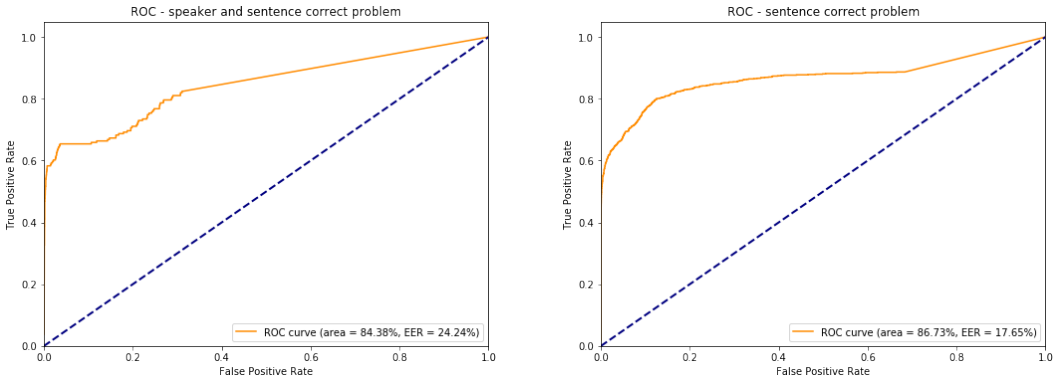
Model	Tylko mówca		Tylko treść		Mówca i treść	
	EER	AUC	EER	AUC	EER	AUC
GMM-UBM	4.69%	97.56%	6.04%	97.82%	3.20%	99.04%
HMM-GMM	15.86%	92.73%	-	-	5.16%	98.32%
DNN-GMM	39.51%	65.12%	17.65%	86.73%	24.24%	84.38%



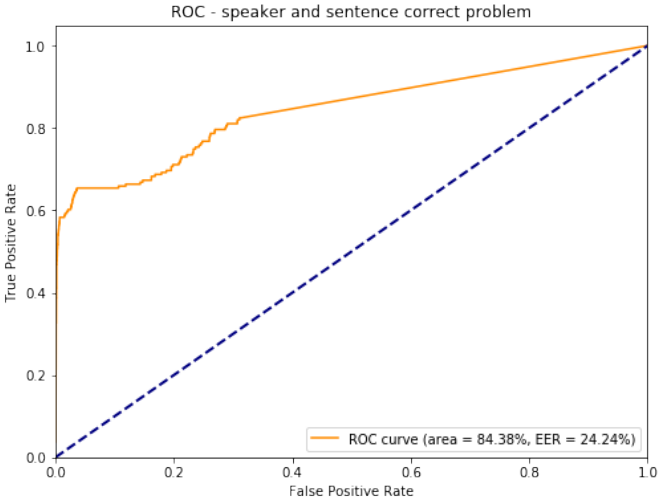
Rysunek 4.1: Wykresy krzywej ROC dla GMM-UBM, pierwszego zadania RedDots i przypadków weryfikacji mówcy oraz weryfikacji treści



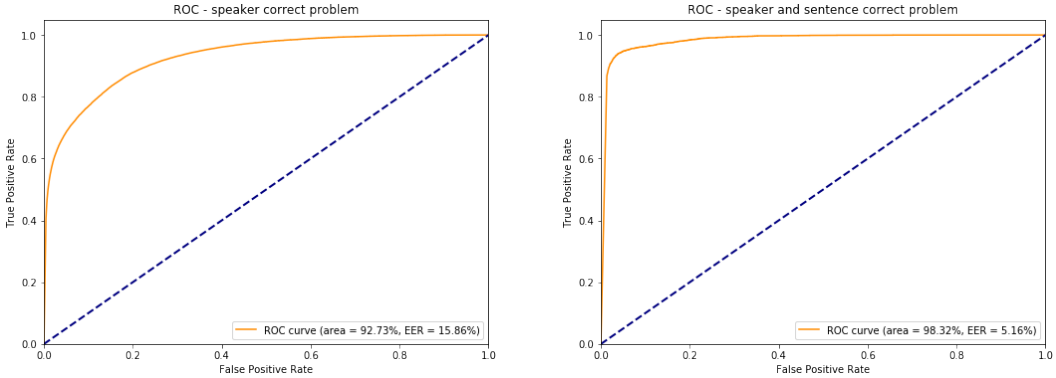
Rysunek 4.2: Wykres krzywej ROC dla GMM-UBM, pierwszego zadania RedDots i jednoczesnej weryfikacji mówcy i treści



Rysunek 4.3: Wykresy krzywej `ROC` dla `DNN-GMM`, pierwszego zadania `RedDots` i przypadków weryfikacji mówcy oraz weryfikacji treści



Rysunek 4.4: Wykres krzywej `ROC` dla `DNN-GMM`, pierwszego zadania `RedDots` i jednoczesnej weryfikacji mówcy i treści



Rysunek 4.5: Wykresy krzywej `ROC` dla `HMM-GMM`, pierwszego zadania `RedDots` i przypadków weryfikacji mówcy oraz jednoczesnej weryfikacji mówcy i treści

Rozdział 5

Podsumowanie i wnioski

5.1 Dyskusja wyników

W kilku przypadkach wyniki zbliżyły się do tych z przejrzanych prac naukowych. W większości przypadków, w szczególności w przypadku najbardziej złożonego modelu z `DNN`, wyniki były gorsze.

Model `GMM-UBM`, dzięki jego prostocie, udało się odtworzyć i może pełnić rolę *baseline* w pracy. Sposób weryfikacji tekstu przez odpowiednie zaadaptowanie `GMM` okazał się skuteczny, mimo że model nie uwzględnia tego jak fonemy następują po sobie w czasie.

Niestety drugi model `HMM-GMM` osiągnął wyniki gorsze od oczekiwanych. Możliwe, że należało, podobnie jak w przypadku `DNN`, użyć gotowego systemu do rozpoznawania mowy, zamiast implementować własne rozwiązanie. Stany `HMM`, które miały odpowiadać fonemom, nie dopasowują się do nagrania zgodnie z tym założeniem. Wyniki udało się poprawić modyfikując algorytm Viterbiego, by przypisywał większe prawdopodobieństwo ciągom, w którym ukryte stany pokrywają równą liczbę okien. Założenie to przypomina jednak to z jednego z modeli, w którym nagranie arbitralnie podzielono na kilka równych części. Mimo tego jest on cenny, gdyż demonstruje on kilka ciekawych technik wykorzystujących wiedzę o fonetyce. Dostarcza on informacji o tym jaką skuteczność powinien wykazywać system z `DNN`.

Stworzony model `DNN-GMM` jest istotny, gdyż sposób wykorzystania sieci neuronowej jest w nim w pewnym stopniu nowy. Jego skuteczność niestety cierpi z powodu tego, że `DeepSpeech` wykrywa litery, choć odpowiednie do porównywania mówców byłyby fonemy. Wytrenowanie go od postaw przy obecnej dostępności zasobów i zbiorów danych byłoby jednak trudne i mało prawdopodobne, by mimo architektury odpowiadającej problemowi, uzyskana skuteczność była wyższa. Również ocena zgodności mówców mogłaby zyskać na skuteczności, gdyby wykorzystać popularne i-wektory. W problemie weryfikacji treści model ten ma taką zaletę, iż można go użyć do weryfikacji dowolnej treści. Nie jest ograniczony do zamkniętego zbioru zdań jak `GMM-UBM`. Ostatecznie wyniki są mało obiecujące i mimo złożoności modelu nie są lepsze niż te z pozostałych dwóch systemów.

5.2 Perspektywy dalszych badań

Przede wszystkim warto byłoby wypróbować sieć nauczoną do rozpoznawania fonemów, a nie liter. Jest to inne zadanie i szczególnie w angielskim mapowanie między literami i fonemami jest bardzo niejednoznaczne. Z przejrzanych prac wiadomo, że wykorzystanie sieci do dopasowywania ramek może dać dobre wyniki, lecz nie użyto w nich sieci rekurencyjnych.

Wyniki tworzonych modeli z pewnością dałoby się poprawić wykonując dokładną optymalizację hiperparametrów. Podobnie można spróbować zdobyć inny zbiór danych, niż `RedDots` na potrzeby treningu i walidacji. Możliwe wtedy byłoby lepsze porównanie wyników z wynikami innych prac.

Metoda wyznaczania prawdopodobieństwa przynależności przez obliczanie stosunku przynależności do mikstur gaussowskich użytkownika oraz modelu tła nie jest tak powszechnie stosowana jak metoda i-wektorów. Połączenie tej metody z metodą przypisywania ramek z tej pracy mogłoby poprawić wyniki. Zwłaszcza, że i-wektory można wyliczyć w oparciu o statystyki z głębokich sieci neuronowych do rozpoznawania fonemów. W przejrzanych pracach te sieci bazowały na dopasowaniu ramek dokonanym przez system `HMM-GMM`.

Przegląd prac wykazał, że można wymyślić wiele różnych sposobów na wykorzystanie sieci neuronowych przy problemie weryfikacji mówcy. Cechy *bottleneck*, wykorzystanie *DNN* w metodzie i-wektorów, wykorzystanie *TDNN* do dopasowania ramek czy system *end-to-end* od Google. Nie wszystkie cechują się wyższą skutecznością, na przykład system *bottleneck* również nie przewyższył tradycyjnych metodami, w których dopracowanie włożono więcej czasu. Możliwości jest wiele i jak pokazują te przykłady, niektóre dają obiecujące wyniki. Ze wzrostem wiedzy łatwiej będzie wyselekcjonować dobre pomysły, dlatego warto zmienić podejście i próbować dalej.

Bibliografia

[1] Homayoon Beigi. *Fundamentals of Speaker Recognition*. Springer Publishing Company, Incorporated, 2011.

[2] Richard O. Duda, Peter E. Hart, David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[3] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren. DARPA TIMIT acoustic phonetic continuous speech corpus CDROM, 1993.

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, Andrew Y. Ng. Deep Speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[6] Georg Heigold, Ignacio Moreno, Samy Bengio, Noam Shazeer. End-to-end text-dependent speaker verification. *CoRR*, abs/1509.08062, 2015.

[7] Sepp Hochreiter, Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[8] Tomi Kinnunen, Md Sahidullah, Ivan Kukanov, Héctor Delgado, Massimiliano Todisco, Achintya Sarkar, Nicolai Bæk Thomsen, V Hautamaki, Nicholas Evans, Zheng-Hua Tan. Utterance verification for text-dependent speaker recognition: A comparative assessment using the reddots corpus, 2016.

[9] Kong-Aik Lee, Anthony Larcher, Guangsen Wang, Patrick Kenny, Niko Brümmer, David A. van Leeuwen, Hagai Aronowitz, Marcel Kockmann, Carlos Vaquero, Bin Ma, Haizhou Li, Themos Stafylakis, Md. Jahangir Alam, Albert Swart, Javier Perez. The reddots data collection for speaker recognition. *INTERSPEECH*, strony 2996–3000. ISCA, 2015.

[10] Y. Lei, N. Scheffer, L. Ferrer, M. McLaren. A novel scheme for speaker recognition using a phonetically-aware deep neural network. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, strony 1695–1699, 2014.

[11] Yi Liu, Liang He, Yao Tian, Zhuzi Chen, Jia Liu, Michael T. Johnson. Comparison of multiple features and modeling methods for text-dependent speaker verification, 2017.

[12] Jianbo Ma, Saad Irtza, Kaavya Sriskandaraja, Vidhyasaharan Sethu, Eliathamby Ambikairajah. Parallel speaker and content modelling for text-dependent speaker verification. *Interspeech 2016*, strony 435–439, 2016.

[13] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio. On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, strony III–1310–III–1318. JMLR.org, 2013.

[14] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[15] Lawrence Rabiner, Biing-Hwang Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[16] Hiroaki Sakoe, Seibi Chiba. A dynamic programming approach to continuous speech recognition. *Proceedings of the Seventh International Congress on Acoustics, Budapest*, wolumen 3, strony 65–69, Budapest, 1971. Akadémiai Kiadó.

[17] D. Snyder, D. Garcia-Romero, D. Povey. Time delay deep neural network-based universal background models for speaker recognition. *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, strony 92–97, 2015.

[18] Christian Staudt. The Python ecosystem for data science: A guided tour, 2017. PyData Warsaw 2017.

[19] The CMU Pronouncing Dictionary. `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`. (Stan na dzień: 2018-02-17).

[20] Yao Tian, Meng Cai, Liang He, Jia Liu. Investigation of bottleneck features and multilingual deep neural networks for speaker verification, 2015.

[21] Steve Young, G Evermann, M.J.F. Gales, Thomas Hain, D Kershaw, Xunying Liu, G Moore, J Odell, D Ollason, Daniel Povey, V Valtchev, Philip Woodland. *The HTK book*. Cambridge University Engineering Department, 2002.

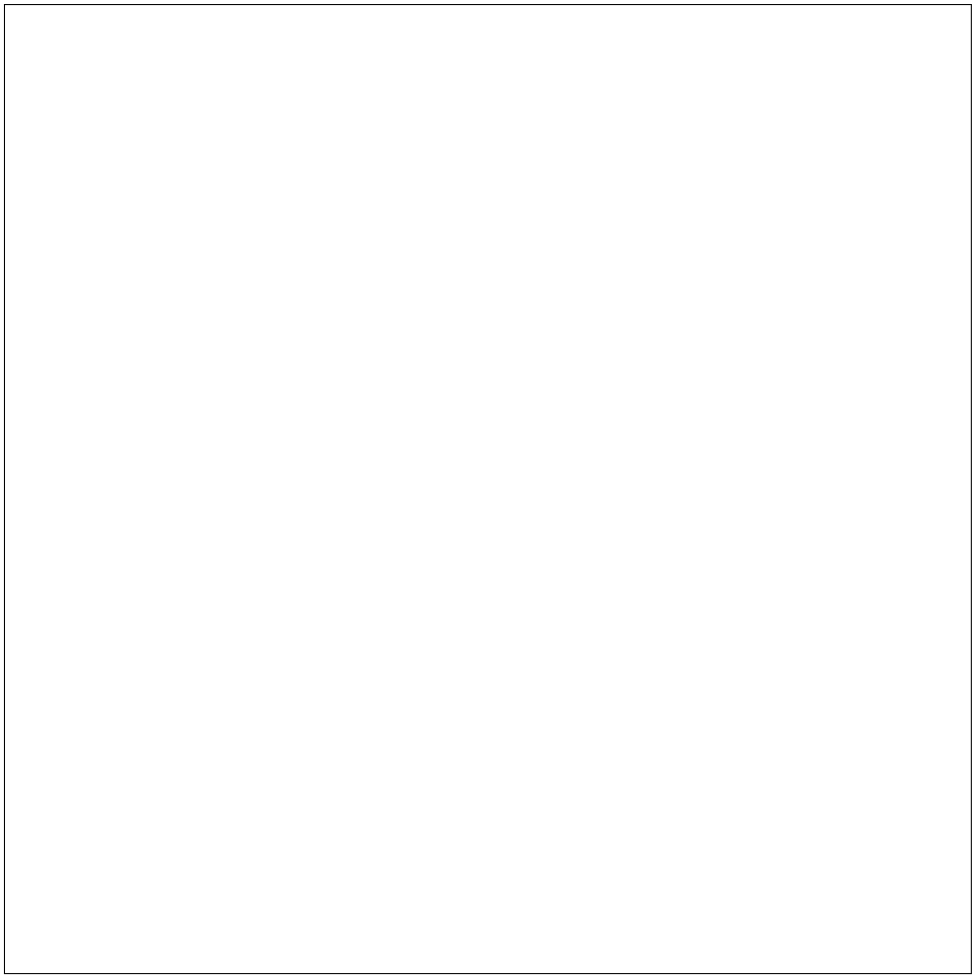
[22] Hossein Zeinali, Hossein Sameti, Lukas Burget, Jan Cernocky, Nooshin Maghsoodi, Pavel Matejka. i-Vector/HMM based text-dependent speaker verification system for RedDots challenge, 2016.

Spis rysunków

Spis tabel

Dodatek A

Płyta CD



Zawartość katalogów na płycie:

doc : elektroniczna wersja pracy dyplomowej oraz dwie prezentacje wygłoszone podczas seminarium dyplomowego

src : repozytorium z kodem źródłowym aplikacji

Miejsce budowania modeli i wizualizacje zawarte są w katalogu **src/notebooks** w postaci notatników *Jupyter*. Są to pliki z rozszerzeniem **.ipynb**. Wykonanie ich wymaga uruchomienia serwera *Jupyter*, pomocne w tym mogą być skrypty *runInDocker.sh* i *runJupyter.sh*. Przeglądać i edytować je można przez interfejs webowy. Skrypty zostały sporządzone dla systemu operacyjnego Linuks, lecz *Jupyter* i *Python* dostępne są na wielu platformach.