Metody Probabilistyczne w Uczeniu Maszynowym
—— Projekt zaliczeniowy ——

Piotr Mikołajczyk

Kod jest dostępny w publicznym repozytorium:
`https://github.com/pmikolajczyk41/Automatic-Speech-Recognition-Simple-calculator`

Spis treści

1 Wprowadzenie

Omawiany projekt jest próbą adaptacji ukrytych modeli Markowa (HMM) do automatycznego rozpoznawania mowy. Głównym celem przedsięwzięcia było poznanie i zrozumienie nieprezentowanych na wykładzie tematów. 'Prezentowalnym' efektem pracy jest system *speaker dependent* pełniący rolę głosowego kalkulatora.

2 Materiały

Dogłębne zrozumienie wybranego tematu wymagało zdobycia nowej wiedzy z kilku dość obcych mi dziedzin, przede wszystkim fonetyki oraz przetwarzania sygnałów. W tym celu przestudiowałem większość kursu *Speech processing* wykładanego na Uniwersytecie Edynburskim:

- `speech.zone/courses/speech-processing/` – strona kursu
- `https://github.com/laic/uoe_speech_processing_course` – repozytorium z częścią materiałów kursu

Dostępne tam materiały pokrywały również interesujące mnie zagadnienia z uczenia maszynowego, algorytmiki oraz probabilistyki.

Projektem zaliczeniowym na tym przedmiocie jest właśnie system rozpoznawania mowy (sekwencji cyfr) oparty o ukryte modele Markowa. Różnica między naszymi realizacjami polega w dużej mierze na tym, że tam studenci korzystają z dostarczonego oprogramowania, które wyłącznie konfigurują oraz dostarczają danych treningowych, ja implementowałem system od podstaw.

Do przetwarzania i analizy danych korzystałem z dwóch dostępnych programów:

- `Praat`
- `Audacity`

Przy implementacji algorytmu Baum-Welcha wspomagałem się artykułem na Wikipedii:

- `https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm`

Poza wspomnianym kursem, do zapoznania sie z reprezentacją MFCC korzystałem z artykułu:

- `http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/`

3 Implementacja

Implementacja została popełniona w języku `Python` (wersja `3.6`). Dodatkowo wykorzystywanych jest kilka ogólnodostępnych bibliotek:

- `numpy`, `scipy` – głównie do obliczeń macierzowych
- `graphviz` – do wizualizacji modeli
- `sounddevice` – do konwersji plików audio do wektorów cech

4 Dane

4.1 Gromadzenie i oznaczanie

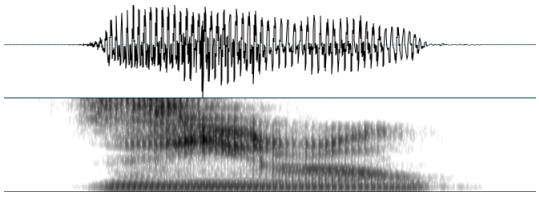
Do każdego z trenowanych modeli (10 cyfr i 4 operatory) zostało zebranych 7 obserwacji. Dane używane w procesie uczenia pochodziły od jednej osoby – w celu stworzenia systemu uniwersalnego ilość potrzebnych danych zdecydowanie przekraczała moje możliwości (warto wspomnieć, że głos jest dość powszechnie wykorzystywanym biometrykiem, co powoduje, że jego przetwarzanie i analiza przy budowie większych systemów mogą być postrzegane jako naruszanie prywatności). Dodatkowo na potrzeby testów każde ze słów bazowych zostało nagrane trzykrotnie przez dwie osoby (w tym oczywiście autora danych treningowych) oraz proste wyrażenia arytmetyczne (od jednego do trzech użytych operatorów).

Nagranie poddane zostały obróbce technicznej (przede wszystkim redukcja szumu środowiska), a następnie oznaczone.

4.2 Konwersja do MFCC

Nagrany dźwięk najczęściej jest przechowywany w wymiarach czasu i amplitudy, tzn. dla każdego momentu w czasie trzymamy wartość, która mówi o chwilowym poziomie ciśnienia powietrza przy mikrofonie. Ponieważ wszystkie nagrania zostały wykonane przy częstotliwości 16kHz, to każda sekunda nagrania dostarcza 16000 liczb.

Bardzo łatwo jest się przekonać po krótkim wprowadzeniu do fonetyki i przetwarzaniu sygnałów, że znacznie wygodniej i pożyteczniej jest pracować w nieco innej przestrzeni – takiej, gdzie wymiarami są (czas,) częstotliwość i magnituda. Ta forma reprezentacji powinna być w miarę znajoma dla tych, którzy mieli styczność z transformatą Fouriera. Radykalnie upraszczając temat, jeśli falę dźwiękową potraktujemy jako funkcję $\mathbb{R} \mapsto \mathbb{R}$, to poprzez aplikację odpowiednich przekształceń wyrazimy ją jako ważoną sumę funkcji postaci $\sin(\alpha \cdot x + \omega)$, gdzie $\alpha \in \mathbb{N}$ odpowiada za częstotliwość funkcji bazowej, a ω z jej fazę (przesunięcie). Jak wynika z podstawowej teorii sygnałów i zdrowego rozsądku, przy naszym próbkowaniu 16kHz najwyższa wyrażalna częstotliwość nagrań to 8kH, zatem wspomniana suma powinna mieć skończoną ilość wyrazów. Otrzymujemy zatem reprezentację dźwięku za pomocą kombinacji liniowej fal sinusoidalnych, które tworzą tzw. czyste tony. Co jest bardzo ciekawe (i użyteczne), to fakt, że ucho ludzkie jest niewrażliwe na zmiany w fazie (ω). Dzięki temu możemy zredukować ilość parametrów o połowę, pamiętając jedynie o magnitudzie.



Rysunek 1: Słowo *zero* przedstawione w obu domenach: (czas, amplituda) oraz (częstotliwość, magnituda).

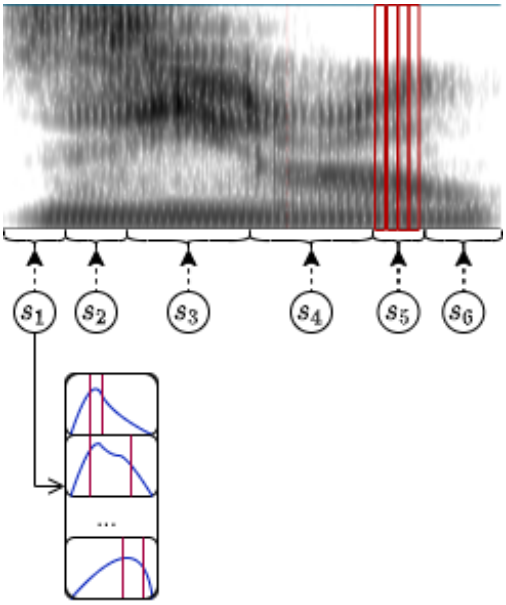
Dźwięki, a w szczególności fonemy wydawane przez człowieka, możemy najprościej podzielić na okresowe (deterministyczne) oraz nieokresowe (stochastyczne). Pierwszy typ dotyczy powtarzalnego wzoru fali na przestrzeni kilku momentów, drugi brak takiego wzorca. Podział ten stanowi dobry punkt wyjścia do analizy dźwięku w krótkich oknach czasowych (zazwyczaj 25ms z przesunięciem 10ms). Tak zawężona fala zwykle daje się dość wyraźnie przedstawić w domenie częstotliwości. Jak się okazuje, taka reprezentacja (jej wykres nazywamy spektrogramem) daje bezpośrednio bardzo dużo przydatnych informacji (przede wszystkim podstawową częstotliwość, formanty, obwiednię sygnału). Teoretycznie już na tym etapie moglibyśmy myśleć o ekstrakcji tych wiadomości do wektorów cech. Niestety jest jeszcze z tym kilka problemów. Między innymi, przez nachodzenie ramek mielibyśmy bardzo dużą zależność pomiędzy kolejnymi wektorami cech. Co więcej, musielibyśmy uwzględnić sporo informacji, które na potrzeby rozpoznawania mowy są zupełnie zbędne, jak na przykład składowe harmoniczne. Problem rozwiązuje przejście do reprezentacji tak zwanych MFCC (mel-frequency cepstral coefficients). Z grubsza rzecz biorąc aplikujemy serią przekształceń (stratnych) reprezentację fourierowską. Obejmuje to stosowanie odpowiednich serii filtrów, transformaty cosinusowe oraz zmiany skali. Jako rezultat otrzymujemy zdekorelowane wektory współczynników bez zbędnych informacji. Najczęściej korzysta się z pierwszych 13 wartości. Opcjonalnie można rozszerzyć wektor o kolejne 13/26 współczynników opisujących dynamikę spektrogramu (system używa wszystkich 39 parametrów).

5 Ukryte modele Markowa

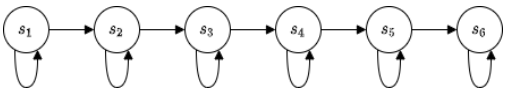
5.1 Wprowadzenie

Bardzo niełatwo jest określić, jak dokładnie niewykładana teoria powinna być opisana w takim raporcie. Wydaje się, że najlepszym wyjściem będzie delikatny zarys intuicji na podstawie naszego konkretnego przypadku.

Mamy dany spektrogram reprezentowany przez m wektorów mfcc. Poprzez analizę organoleptyczną mamy wrażenie, że sąsiadujące ramki czasowe (a również i wektory cech) wyglądają bardzo podobnie do siebie. Jesteśmy zatem w stanie uwierzyć, że takie spektrum można by wygenerować z kilku, powiedzmy n rozkładów.



Dokładniej, załóżmy że mamy n (niezależnych) rozkładów prawdopodobieństwa, z których każdy jest rozkładem produktowym 39 (niezależnych) rozkładów normalnych. Zatem każdy z nich potrafi wygenerować wektor mfcc. Na razie załóżmy, że znamy $39n$ bazowych rozkładów (ich oczekiwane i wariancje). Będziemy już wkrótce potrzebować poznać prawdopodobieństwo wygenerowania konkretnej sekwencji obserwacji przez dany zbiór rozkładów $\{s_1, \ldots, s_n\}$. Dla uproszczenia, możemy założyć, że początkowy blok obserwacji został wygenerowany przez s_1, kolejny blok przez s_2 itd. Myśleć o tym możemy poniekąd jak o skrzyżowaniu łańcucha Markowa ze skończonym transduktorem (*finite state transducer*), tzn. automacie skończonym gdzie pomiędzy stanami przechodzimy z pewnym (stałym) prawdopodobieństwem, a wchodząc do stanu emitujemy obserwację ze skojarzonego rozkładu. Nasza uproszczona topologia wygląda następująco:



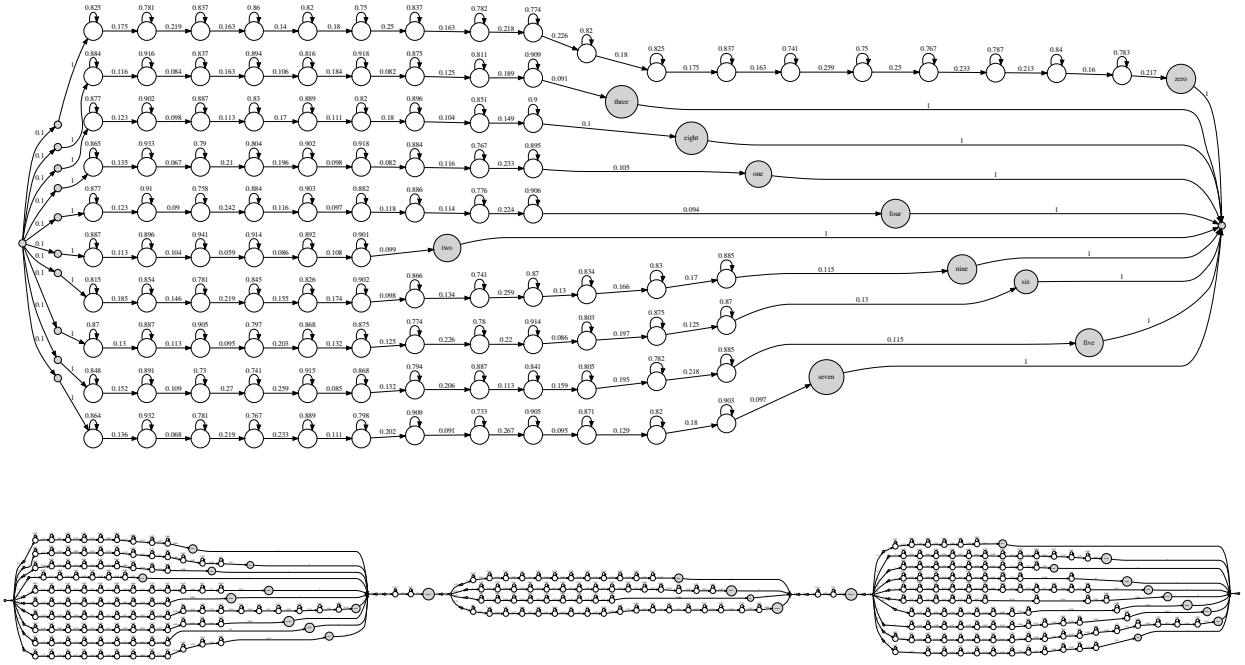
Obliczenie prawdopodobieństwa wygenerowania sekwencji sprowadza się zatem to wskazania odpowiedniej funkcji mapującej $[m] \mapsto [n]$. Formalnie, z ciągiem obserwacji $Y_1, \ldots, Y_m$ skojarzony jest ciąg zmiennych losowych $X_1, \ldots, X_m$ mówiących z którego rozkładu pochodzi dana obserwacja. Konkretne algorytmy obliczające tę wartość poznamy nieco później.

5.2 Generatywny klasyfikator

Załóżmy, że mamy 10 modeli dla każdej z cyfr, tzn. każdy model (uzbrojony w prawdopodobieństwa przejść między stanami wraz z parametrami rozkładów w każdym stanie) potrafi obliczyć z jakim prawdopodobieństwem dany ciąg obserwacji mógł być przezeń wygenerowany. Porównawszy te wartości przyjmujemy, że dane reprezentują to słowo, dla którego skojarzony model zwrócił najbardziej obiecującą odpowiedź.

5.3 Łączenie modeli

Jak już zostało wspomniane, ukryte modele Markowa można w pewien sposób traktować jak automaty. Co więcej, cały proces odbywa się 'bez pamięci'. Dzięki temu możemy łączyć modele dla bazowych jednostek (jak fonemy czy słowa) otrzymując modele dla skomplikowanych wyrażeń (jak zdania czy cały język). O tym jak łączyć decyduje narzucona gramatyka. Poniżej znajduje się model rozpoznający pojedynczą cyfrę oraz model dla prostych wyrażeń arytmetycznych – aplikacji binarnego operatora do dwóch cyfr.



5.4 Obliczanie prawdopodobieństwa generacji

Mając dany model z n stanami oraz sekwencję m obserwacji chcemy dowiedzieć się, ile wynosi prawdopodobieństwo generacji. Jedyne założenie (które na potrzeby łączenia modeli musimy przyjąć) to takie, że pierwsza obserwacja musi pochodzić z pierwszego stanu, natomiast ostatnia z ostatniego. Możemy to zrobić na dwa sposoby:

1. porządnie: należy rozważyć wszystkie możliwe przejścia modelu w m krokach startując w pierwszym stanie i kończąc w ostatnim; tu stosujemy algorytm Bauma-Welcha
2. heurystycznie: szukamy ścieżki długości m z pierwszego stanu do ostatniego, której prawdopodobieństwo jest najwyższe; tu stosujemy algorytm Viterbiego

Formalnie prawdopodobieństwem ścieżki jest iloczyn prawdopodobieństw na napotkanych krawędziach oraz gęstości prawdopodobieństwa wygenerowania konkretnej obserwacji ze skojarzonego stanu. Tu uwaga techniczna, z powodów numerycznych rozważa się raczej sumę logarytmów czynników.

Jak się okaże, oba algorytmy to tak naprawdę adaptacja programowania dynamicznego do automatów skończonych. Pozwolę sobie podać wyłącznie bardzo ogólne pomysły, które za nimi stoją, bez wzorów ani dowodów.

Uczciwie mówiąc, omawiane poniżej algorytmy służą (zwłaszcza pierwszy) do trenowania modeli wyliczając parametry metodą największej wiarygodności. Jednak osobiście wydaje mi się, że przedstawienie ich najpierw w uproszczonej wersji do celów obliczania prawdopodobieństwa sprzyja zrozumieniu materiału (ja się właśnie z tej strony uczyłem).

5.4.1 Algorytm Bauma-Welcha

Wykrywanie każdej możliwej ścieżki w automacie wydaje się dość drogim obliczeniowo zadaniem. Możemy jednak zdefiniować naszą odpowiedź jako połączenie dwóch programów dynamicznych. Otóż zauważmy, że prawdopodobieństwo wygenerowania j-tej obserwacji z i-tego stanu możemy wyliczyć jako iloczyn:

- sumy po wszystkich stanach i' (z których można przejść do i-tego) prawdopodobieństwa wygenerowania obserwacji $Y_1, \ldots, Y_{j-1}$ spacerem kończącym się w i'
- prawdopodobieństwa przejścia $i' \to i$
- gęstości prawdopodobieństwa (tu mamy formalną szarlatanerię) wygenerowania Y_i ze stanu i
- prawdopodobieństwa wygenerowania obserwacji $Y_{j+1}, \ldots, Y_m$ wychodząc ze stanu i

Do obliczania pierwszego i czwartego punktu konstruujemy dwa niezależne programy liniowe (w literaturze *forward/backward procedure*), które działają w czasie $\mathcal{O}(n^m)$, przy czym w modelach o ograniczonych sąsiedztwach złożoność wynosi $\mathcal{O}(nm)$.

5.4.2 Algorytm Viterbiego

Poprzedni algorytm wylicza poprawny wynik kosztem jednak długiego czasu działania. Empirycznie zostało wykazane, że w przypadku ukrytych modeli Markowa suma prawdopodobieństw ścieżek jest zdominowana przez najbardziej prawdopodobną. Jest to niby heurystyka, ale działa dobrze. Do jej realizacji można podjeść na (co najmniej) dwa sposoby:

- programowanie dynamiczne: w kracie $n \times m$ szukamy najdroższej ścieżki między jej dwoma rogami; w zasadzie *dynamic time warping*
- symulacja pionkami: zaczynamy z jednym pionkiem w stanie startowym; w każdej jednostce czasu pionek replikuje się na każdą krawędź wychodzącą z jego obecnego stanu, dolicza do swojego lokalnego prawdopodobieństwa koszt przejścia oraz generacji kolejnej obserwacji; jeśli kilka pionków spotyka się w jednym stanie, przeżywa tylko ten z najwyższym dotychczasowym prawdopodobieństwem; jako wynik zwracamy zwycięzce z ostatniej rundy ze stanu końcowego

Złożoność czasowa jest jak poprzednio. Pamięciowo drugie podejście wymaga jedynie $\Theta(n)$ miejsca, natomiast algorytmy korzystające wprost z kraty potrzebują oczywiście $\Theta(nm)$.

Wydajność algorytmu 'pionkowego' można znacznie podnieść stosując różne rodzaje odcinania, m.in.:

- *'beam search'*: utrzymujemy stałą liczbę najlepiej rokujących pionków, tj. po każdej rundzie zostawiamy tylko k pionków z najwyższym dotychczasowym prawdopodobieństwem
- pewien wariant poprzedniego podejścia: utrzymujemy tylko te pionki, których dotychczasowe prawdopodobieństwo jest nie mniejsze niż prawdopodobieństwo najlepszego pionka pomniejszone o ustaloną stałą

W praktyce algorytm Bauma-Welcha jest istotnie wolniejszy od Viterbiego, wobec czego w systemach interaktywnych używa się drugiego.

5.5 Trening

5.5.1 Inicjalizacja parametrów modelu

Najczęściej potrzebujemy w jakiś sposób nadać modelowi początkowe parametry. Najprostszym sposobem, który dodatkowo lekko może przyspieszyć zbieżność jest tzw. jednostajna segmentacja. Mając m obserwacji i n stanów, robimy prosty podział danych na n grup równego rozmiaru i na podstawie takiego przyporządkowania obliczamy wstępne wartości dla rozkładów oraz przejść. Oczywiście wystarczy jedna iteracja takiego procesu.

5.5.2 Trening algorytmem Viterbiego

Mając sekwencję obserwacji uruchamiamy algorytm Viterbiego, który zwróci nam ścieżkę zwycięskiego pionka (czyli najbardziej prawdopodobne dopasowanie). Na jej podstawie uaktualniamy parametry modelu, dla każdego stanu niezależnie, korzystając z przypisanych mu obserwacji.

5.5.3 Trening algorytmem Bauma-Welcha

W swojej pełnej wersji algorytm Bauma-Welcha jest tak naprawdę adaptacją metody maksymalizacji wartości oczekiwanej. Na podstawie obliczonych prawdopodobieństw możemy wyznaczyć estymatory parametrów modelu (maksymalizując wiarygodność).

Oczywiście wszystkie z powyższych algorytmów dają się łatwo zgeneralizować na cały zbiór sekwencji obserwacji.

6 Wyniki

6.1 Rozpoznawanie pojedynczych wyrazów

Wytrenowanych zostało 15 modeli bazowych:

- </> po jednym na każdą cyfrę (w języku angielskim)
- </> po jednym na każdy operator $(+, -, /, *)$
- </> cisza

Otrzymane zostały 3 generacje:

1. po treningu algorytmem Viterbiego (do 10 iteracji, zwykle wystarczało około 6 przebiegów)
2. pierwsza generacja wzbogacona o 25 iteracji algorytmu Bauma-Welcha
3. druga generacja wzbogacona o kolejne 25 iteracji algorytmu Bauma-Welcha

Jak się okazało, już pierwsza generacja osiąga 100% skuteczności na danych testowych pochodzących od tego samego mówcy, co dane treningowe. Na danych od nowego mówcy wszystkie generacje osiągają skuteczność w granicach $31\% - 33\%$.

6.2 Rozpoznawanie złożonych wyrażeń

Ostatecznie przetestowane zostały wyłącznie wyrażenia postaci `<cyfra> <operator> <cyfra>`. Tutaj wszystkie generacje utrzymują skuteczność 50%. Niestety nie udało się dojść przyczyny problemu.

7 Interaktywna aplikacja

Do kodu dołączona jest prosta aplikacja z interfejsem graficznym, która umożliwia wykorzystanie modelu online.