Probability in Speech Recognition

Ansh Khandelwal

September, 2020

Contents

1 Introduction

Speech recognition, or the speech-to-text, is the ability for a machine or program to identify words spoken aloud and convert then into readable text. Elementary speech recognition software has a limited vocabulary of words and phrases, and it may only identify these if they are spoken very clearly. More sophisticated software has the ability to accept natural speech, different accents and languages.

Speech recognition works using algorithms through acoustic and language modeling. Acoustic modeling represents the relationship between linguistic units of speech and audio signals; language modeling matches sounds with word sequences to help distinguish between words that sound similar.

Hidden Markov models are used to recognize temporal patterns in speech to improve accuracy within the system. This method will randomly change systems where it is assumed that future states do not depend on past states. Other methods used in speech recognition may include natural language processing (NLP) or N-Grams. NLP makes the speech recognition process easier and take less time. N-Grams, on the other hand, are a relatively simple approach to language models. They help create a probability distribution for a sequence.

More advanced speech recognition software will use AI and machine learning. These systems will use grammar, structure, syntax as well as composition of audio and voice signals in order to process speech.

2 Markov Model

Definition 2.1. *Markov Property*
A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it.

A Markov model is a stochastic method for randomly changing systems where it is assumed that future states do not depend on past states, i.e. it follows Markov Property. These models show all possible states as well as the transitions, rate of transitions and probabilities between them.
Markov models are often used to model the probabilities of different states and the rates of transitions among them. Markov models can also be used to recognize patterns, make predictions and to learn the statistics of sequential data.

There are four types of Markov models that are used :

- **Markov chain** - used by systems that are autonomous and have fully observable states
- **Hidden Markov model** - used by systems that are autonomous where the state is partially observable.
- **Markov decision processes** - used by controlled systems with a fully observable state.
- **Partially observable Markov decision processes** - used by controlled systems where the state is partially observable.

Applications of Markov modeling include modeling languages, natural language processing (NLP), image processing, bioinformatics, **speech recognition** and modeling computer hardware and software systems.

2.1 Markov Process

Definition 2.2. *It is considered as a Markov Process if :*

- *A continuous-valued Markov process $X(t)$ satisfies the conditional pdf expression given by*

$$f_X(x_n|x_{n-1}, x_{n-2}, ..., x_1; t_{n-1}, ..., t_1) = f_X(x_n|x_{n-1}; t_n, t_{n-1}) \tag{1}$$

  $\forall\ x_1, x_2, ..., x_n,\ \forall\ t_1 < t_2 < ..., < t_n$ *and for all integers* $n > 0$.

- *A discrete-valued Markov process $X(t)$ satisfies the conditional pmf expression given by*

$$P_X(x_n|x_{n-1}, x_{n-2}, ..., x_1; t_{n-1}, ..., t_1) = P_X(x_n|x_{n-1}; t_n, t_{n-1}) \tag{2}$$

  $\forall\ x_1, x_2, ..., x_n,\ \forall\ t_1 < t_2 < ..., < t_n$ *and for all integers* $n > 0$.

The value of the process $X(t)$ at a given time t thus determines the conditional probabilities for future values of the process. The values of the process are called *states* of the process, and the conditional probabilities are thought of as *transition probabilities* between the states. If only a finite or countable set of values of x_i is allowed, the discrete-valued Markov process is called a Markov chain.

Example 1. ***Poisson counting process**, **Weiner process** both are examples of continuous-valued Markov process, because they both follow independent-increments property*

2.2 Markov Chain

Definition 2.3. *A Markov chain is a sequence of random variables X1, X2, X3, . . . with the Markov property, namely that the probability of any given state X_n only depends on its immediate previous state X_{n-1}. Formally:*

$$P_X(x_n|x_{n-1}, x_{n-2}, ..., x_1; t_{n-1}, ..., t_1) = P_X(x_n|x_{n-1}; t_n, t_{n-1}) \tag{3}$$

where $P(A|B)$ is the probability of A given B.

The possible values of X_i form a countable set S called the state space of the chain. If the state space is finite, and the Markov chain time-homogeneous (i.e. the transition probabilities are constant in time), the transition probability distribution can be represented by a matrix $P = (p_{ij})$ $i, j \in S$(State space), called the transition matrix, whose elements are defined as:

$$p_{ij} = P(X_n = i|X_{n-1} = j)$$

It means that in the matrix P at the position i, j is $p_{i,j}$, which is the probability of going to state i from the state j. Also, the sum of each column is unity. Transition probabilities can be then computed as power of the transition matrix:

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{P} \cdot \boldsymbol{x}^{(n)}$$

$$\boldsymbol{x}^{(n+2)} = \boldsymbol{P} \cdot \boldsymbol{x}^{(n+1)} = \boldsymbol{P}^2 \cdot \boldsymbol{x}^{(n)}$$

$$\boldsymbol{x}^{(n)} = \boldsymbol{P}^n \cdot \boldsymbol{x}^{(0)}$$

Definition 2.4. ***Process Diagram**, also called **State-transition Diagram***
The process diagram of a Markov chain is a directed graph describing the Markov process. Each node represent a state from the state space. The edges are labeled by the probabilities of going from one state to the other states. Edges with zero transition probability are usually discarded.

Example 2. *Let us consider a cat, named Mark. It is found only doing one of these three activities all the time: (a) Sleep (b) Play (c) Eat*
These three can be considered as three states the process can take.

$$S = \{sleep,\ play,\ eat\}$$

Now we define a transition matrix that denotes the probability of switching to another activity(state) given the current activity(state).

$$P = \begin{pmatrix} 0.9 & 0.8 & 0.7 \\ 0.05 & 0.2 & 0.3 \\ 0.05 & 0 & 0 \end{pmatrix}$$

which can be read as: probability of going to another state given the current state.

$$\begin{array}{c|ccc} & From: sleep & play & eat \\ \hline To: sleep & 0.9 & 0.8 & 0.7 \\ play & 0.05 & 0.2 & 0.3 \\ eat & 0.05 & 0 & 0 \end{array}$$

This whole thing can also be also represented graphically by *Process Diagrams* as follows:

`markov_process_diagram.jpg`

The transition matrix can be used to predict the probability distribution $x(n)$ at each time step n. For instance, let us assume that Mark is initially sleeping:

$$\boldsymbol{x}^0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

After one minute, we can predict the situation:

$$\boldsymbol{x}^{(1)} = \boldsymbol{P} \cdot \boldsymbol{x}^{(0)} = \begin{pmatrix} 0.9 \\ 0.05 \\ 0.05 \end{pmatrix}$$

Thus, after one minute, there is a 90% chance that the cat is still sleeping, 5% chance that he's eating and 5% that he's playing.
Similarly, we can predict that after two minutes:

$$\boldsymbol{x}^{(2)} = \boldsymbol{P} \cdot \boldsymbol{x}^{(1)} = \begin{pmatrix} 0.885 \\ 0.07 \\ 0.045 \end{pmatrix}$$

Example 3. *A simple weather prediction model*
Lets assume that weather condition tomorrow only depends on the weather today and not before.

Let us consider only two weather states.

$$S = \{sunny,\ rainy\}$$

Lets take our transition matrix as:

$$\boldsymbol{P} = \begin{pmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{pmatrix}$$

The matrix P represents the weather model in which a sunny day is 90% likely to be followed by another sunny day, and a rainy day is 50% likely to be followed by another rainy day.The columns can be labelled "sunny" and "rainy", and the rows can be labelled in the same order.
P_{ij} is the probability that, if a given day is of type j, it will be followed by a day of type i.

Markov_Chain_weather_model_matrix_as_a_graph.png

Predicting future weather:
Lets take the weather on day 0 (today) is to be sunny.

$$\boldsymbol{x}^{(0)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The weather on tomorrow(day 1) can be predicted by

$$\boldsymbol{x}^{(1)} = \boldsymbol{P} \cdot \boldsymbol{x}^{(0)} = \begin{pmatrix} 0.9 \\ 0.1 \end{pmatrix}$$

Which means there is 90% chance of tomorrow being sunny.
We can generalize the rules for day n as:

$$\boldsymbol{x}^{(n)} = \boldsymbol{P} \cdot \boldsymbol{x}^{(n-1)}$$

$$\boldsymbol{x}^{(n)} = \boldsymbol{P}^{n} \cdot \boldsymbol{x}^{(0)}$$

3 Hidden Markov Model

Definition 3.1. *Hidden Markov model is a statistical markov model in which the system being modeled is assumed to be a markov process(say X) with unobserved/hidden states but there exist another process (say Y) whose behaviour depends on X. Terminology:*

- *The term hidden refers to the first order of Markov process behind the observation*
- *The observation refers to the data that we can know and can observe*

And the goal is to learn about the hidden X from the observation Y.

Mathematically we can write it as:
Let X_n and Y_n be discrete time stochastic process and $n \geq 1$. The pair (X_n, Y_n) is a hidden markov model if:

- X_n is a markov process and is not directly observable.
- $\mathbf{P}(Y_n \in A | X_1 = x_1, \dots X_n = x_n) = \mathbf{P}(Y_n \in A | X_n = x_n)$

Example 4 (Drawing balls from the urns)**.** *This is one of the classical examples of the hidden markov model and every hidden markov model can be visualised as the generalisation of the urn problem with replacement.*

hidden_markov_model.png

In this example consider that you have a room which is closed and there is genie in that room and there is a conveyor belt which connect inside of room with outside. The room has urns $X_1, X_2, X_3, \dots$ and each urns contains a known mix of balls $Y_1, Y_2, Y_3, \dots$. The genie chooses a urn and draws a ball from it. And then puts this ball on the conveyor belt, and the observer can observe the ball and its sequence but it has no knowledge about the urn from which it is drawn.
The process of choosing of the urn is dependent only on the chosen urn at the $(n-1)^t h$ step, it has no direct dependence on any of the urns chosen before this single previous urn. Since the observer can only observe the balls that are coming out on the on the conveyor belt and not the actual markov process(selection of urns), it is called *"hidden markov model"*.

Example 5. *Suppose bob(observer) hears a sequence of T sounds $O_1, O_2, \dots, O_T$ and he wants to know something about this sequence of sounds that he heard, possible a sequence of T words which is denoted $S_1, S_2, \dots S_T$, which he never actually will be able to see(hidden states).*

In these types of cases HMM comes to rescue, it allows bob to assign appropriate probabilities to the sound sequences $O's$ and word sequences $S's$ and to make reasonable deductions about them

HMM_speech.png

As you can notice in this figure there is S_0 and O_0 state, these are called the starting states for the HMM model. There are couple of assumtions that the HMM model makes:

1. The observations $o_1, o_2 \ldots, o_T$ come from a known finite set V, called the observation space
2. The hidden states $s_1, s_2, \ldots, s_T$ come from a known, finite sets Q, called the hidden state space.
3. The HMM assigns a probability to any given sequence $s_1, s_2, \ldots, s_T$, The chain rule says that

   $$P(s_1, s_2, \ldots, s_T) = \prod_{t=1}^{T} P(s_t|s_0, s_1, \ldots, s_{t-1}) = \prod_{t=1}^{T} P(s_t|s_{<t})$$

   Because of the fact that the process S, is a markov process, so we can use the markov assumption and can write

   $$\begin{aligned} P(s_1|s_{<t}) &= P(s_t|s_{t-1}) \\ P(s_1, s_2, \ldots, s_T) &= P(s_t|s_{t-1}) \end{aligned} \tag{4}$$

   Note: This markov assumption does not holds for most of the times, like given in our case, sequence of words have dependencies that go back more than one immediate step.

4. The observation o_t only depends on the respective hidden state s_t. We can write:

   $$P(o_t|o_{<t}, s_{<t}) = P(o_t|s_t) \tag{5}$$

   This is known as **independent assumption**.

The probabilities $P(s_t|s_{t-1})$ and $P(o_t|s_t)$ are important to us and are called the *transition probabilities* and *emission probabilities* respectively.

$$\begin{aligned} Q &= \{q_1, q_2, \ldots, q_N\} = \text{distinct states of the Markov process(N states)} \\ V &= \{w_1, w_2, \ldots, w_V\} = \text{set of possible observations(V states)} \\ S &= \{s_1, s_2, \ldots, s_T\} = \text{sequence of states(T steps) } s_i \in Q \\ O &= \{o_1, o_2, \ldots, o_T\} = \text{sequence of observations(T steps) } o_i \in V \end{aligned}$$

The transition and emission probabilities are denoted by matrices A and B and each row of these matrices is a valid distribution(each element is non-zero and they sum up to one).

$$A \in \mathbf{R}^{N*N} = \text{transition probabilities}$$
$$B \in \mathbf{R}^{N*V} = \text{emission probabilities}$$

There are three major things which we care to calculate if we are given a hidden markov model with its transition and emission probabilities.

1. *Computing the likelihood*: Given A, B and a sequence of observations O($o_1, o_2, \ldots, o_T$). Compute $P(O|A, B)$.
   Forward Pass Algorithm is a dynamic programming algorithm which can be efficiently used for this.

2. *Finding the hidden sequence:* Given A, B and a sequence of observations O($o_1, o_2, \ldots, o_T$). Finding the hidden sequence of markov process S that is most likely to generate O, that is:

$$S^* = argmax\ P(O|S, A, B)$$

   Viterbi Algorithm is also a dynamic programming based algorithm which is used to calculate this.

3. *Estimating the parameters:* Given a sequence of observation O. Estimate the transition and emission probabilities that are most likely to give O, that is to find:

$$A*, B* = argmax\ P(O|A, B)$$

   Baum-Welch Algorithm is used to calculate this.

4 Architecture of an HMM-based Recogniser

Hidden Markov Models (HMMs) provide a simple and effective frame- work for modelling time-varying spectral vector sequences. As a con- sequence, almost all present day large vocabulary continuous speech recognition (LVCSR) systems are based on HMMs.
The principal components of a large vocabulary continuous speech recogniser are illustrated in figure below.

ArchitectureHMM.png

Initially, when the audio waveform is inputted via a microphone it is converted into a sequence of fixed size acoustic vectors $Y_1 : T = y_1, ..., y_T$ in a process called feature extraction
The decoder then finds the sequence of words $w_1 : L = w_1, ..., w_L$ which is most likely to have generated $\mathbf{Y}$, i.e. the decoder tries to find

$$\hat{w} = arg_w max\{P(w|Y)\} \tag{6}$$

However, since $P(w|Y)$ is difficult to model directly, Bayes' Rule is used to transform (4) into the equivalent problem of finding:

$$\hat{w} = arg_w max\{P(Y|w)P(w)\} \tag{7}$$

The likelihood $p(Y|w)$ is determined by an **acoustic model** and the prior $P(w)$ is determined by a **language model**. The basic unit of sound represented by the acoustic model is the phone. For example, the word "bat" is composed of three phones /b/ /ae/ /t/. About 40 such phones are required for English.
For any given $\mathbf{w}$, the corresponding acoustic model is synthesised by concatenating phone models to make words as defined by a pronunciation dictionary. The parameters of these phone models are estimated from training data consisting of speech wave forms and their orthographic transcriptions.

The language model is typically an N-gram model in which the probability of each word is conditioned only on its $N-1$ predecessors. The N-gram parameters are estimated by counting N-tuples in appropriate text corpora. The decoder operates by searching through all possible word sequences using pruning to remove unlikely hypotheses thereby keeping the search tractable. When the end of the utterance is reached, the most likely word sequence is output. Alternatively, modern decoders can generate lattices containing a compact representation of the most likely hypotheses.

4.1 Feature Extraction

Feature extraction is a process of dimensional reduction by which an initial set of raw data is reduced to more manageable groups for processing. A characteristic of these large data sets is a large number of variables that require a lot of computing resources to process.
The feature extraction stage seeks to provide a compact representation of the speech waveform. This form should minimise the loss of information that discriminates between words, and provide a good match with the distributional assumptions made by the acoustic models.

Feature vectors are typically computed every 10 ms using an overlapping analysis window of around 25 ms. One of the simplest and most widely used encoding schemes is based on mel-frequency cepstral coefficients. These are generated by applying a truncated discrete cosine transformation to a log spectral estimate computed by smoothing an FFT with around 20 frequency bins distributed nonlinearly across the speech spectrum. The nonlinear frequency scale used is called a mel scale and it approximates the response of the human ear. The DCT is applied in order to smooth the spectral estimate and approximately decorrelate the feature elements. After the cosine transform the first element represents the average of the log-energy of the frequency bins. This is sometimes replaced by the log-energy of the frame, or removed completely.
In addition to the spectral coefficients, first order (delta) and second-order (delta–delta) regression coefficients are often appended in a heuristic attempt to compensate for the conditional independence assumption made by the HMM-based acoustic models. If the original (static) feature vector is y_t^s t , then the delta parameter, Δy_t^s t, is given by

$$\Delta y_t^s = \frac{\sum_{i=1}^{n} w_i (y_{t+i}^s - y_{t-i}^s)}{2\sum_{i=1}^{n} w_i^2} \tag{8}$$

where n is the window width and w_i are the regression coefficients. The delta–delta parameters, are derived in the same fashion, but using differences of the delta parameters. When concatenated together these form the feature vector y_t,

$$y_t = [y_t^{sT} \Delta y_t^{sT} \Delta^2 y_t^{sT}]^T \tag{9}$$

This feature vector whose dimensionality is typically around 40 and which has been partially but not fully decorrelated.
Shown below is an example on how exactly feature vectors are formed

examplevector.png

4.2 HMM Acoustic Models

Each word (w) can be decomposed into a sequence of K_w basic sounds. $q_{1:K_w}^{(w)} = q_1, q_2, q_3 ... q_{K_w}$. Where q_i are basic sounds known as base phones. This sequence of q_is is called the pronunciation of the word w. Each word can have multiple pronunciation, so while calculating $p(Y|w)$ we need sum over all valid pronunciations.

$$p(Y|w) = \sum_{Q} p(Y|Q)P(Q|w) \tag{10}$$

Where Q is is a particular sequence of pronunciation,

$$P(Q|w) = \prod_{i=1}^{L} P(q^{(w_i)}|w_i) \tag{11}$$

Where each $q^{(w_i)}$ is a valid pronunciation of the word w_i. As in practice each word has very few other valid pronunciation. Thus the summation in equation (10) is easily tractable.

HMM-based_phone.png

Individual base phones are represented by a continuous density HMM as shown in the figure above. At every time step, an HMM makes a transition from one state to any of the connected neighbouring states. The state it will be transitioning to is determined by the transition probability $\{a_{ij}\}$ associated with the pair of states $\{s_i \rightarrow s_j\}$. After attaining the final state s_j, A feature vector of s_j is generated using the distribution $\{b_j()\}$ associated with it.
This form of process requires some assumptions to be made of the HMM:-

- Given the previous state, current state is conditionally independent of all other states .
- observations are conditionally independent of all other observations given the state that generated it.

We define mean of a state s_j as $\mu^{(j)}$ and $\sum^{(j)}$ as its co variance.Then we will assume a single multivariate Gaussian for output distribution given by:-

$$b_j(y) = \mathcal{N}\left(y; \mu^{(j)}, \sum\nolimits^{(j)}\right)$$

here $\mu^{(j)}$ is mean of s_j and $\sum^{(j)}$ is co variance. Because of the high dimensionality of acoustic vector y, the covariances can be constrained to be diagonal.
By concatenating all constituent base phones if we could form composite HMM Q ,then it implies we can calculate acoustic likelihood according to:

$$p(Y|Q) = \sum_{\theta} p(\theta, Y|Q) \tag{12}$$

$$p(\theta, Y|Q) = a_{\theta_0}\theta_1 \prod_{i=1}^{T} b_{\theta_i}(y_i) a_{\theta_i}\theta_{i+1}$$

Where $\theta = \theta_0, \ldots, \theta_{T+1}$ and θ_0 and θ_{T+1} correspond to non-emitting entry and exist states and serve the purpose of simplifying the process of concatenating phone models (making words). We can ignore them for analysing the sequence.

To estimate the acoustic model parameters, $\lambda = [\{a_{ij}\}, \{b_j()\}]$, we use corpus of training utterances using the forward-backward algorithm (Expectation Maximisation (EM)). Consider the utterance $\mathbf{Y}_{(r)}, r = 1, \ldots, R$,of sequence of base forms with length $T^{(r)}$, we find the HMMs that correspond to the word sequence in the utterance and construct corresponding composite HMM. We divide the process into two steps,E and the M steps.

For the first ,E step, we calculate the forward(α) and the backward probability(β) using the recursion formulas

$$\alpha_t^{(rj)} = p\left(Y_{1:t}^r, \theta_t = s_j; \lambda\right) = \left[\sum_i \alpha_{t-1}^{(ri)} a_{ij}\right] b_j\left(y_t^{(r)}\right) \tag{13}$$

$$\beta_t^{(ri)} = p\left(Y_{t+1:T^{(r)}}^r | \theta_t = s_i; \lambda\right) = \left[\sum_j a_{ij} b_j\left(y_{t+1}^{(r)}\right) \beta_{t+1}^{(rj)}\right] \tag{14}$$

here i and j are summed over all states. In practice the log-probabilities are stored and log arithmetic is used to avoid the problem of underflow accruing while evaluating the recursions for long speech segments.
The probability of the model occupying any arbitrary state in terms of forward and backward probabilities is given by:

$$\gamma_t^{(rj)} = P\left(\theta_t = s_j | Y^{(r)}; \lambda\right) = \frac{1}{P^{(r)}} \alpha_t^{(rj)} \beta_t^{(rj)} \tag{15}$$

Here, $P^{(r)} = p\left(Y^{(r)}; \lambda\right)$. $\gamma_t^{(rj)}$ represent a soft alignment of the model states to the data. They are also known as occupation probabilities or occupation events.it is straightforward to show that the new set of Gaussian parameters are calculated as

$$\hat{\mu}^{(j)} = \frac{\sum_{r=1}^R \sum_{t=1}^{T^{(r)}} \gamma_t^{(rj)} y_t^{(r)}}{\sum_{r=1}^R \sum_{t=1}^{T^{(r)}} \gamma_t^{(rj)}} \tag{16}$$

$$\hat{\sum}^{(j)} = \frac{\sum_{r=1}^R \sum_{t=1}^{T^{(r)}} \gamma_t^{(rj)} \left(y_t^{(r)} - \hat{\mu}^{(j)}\right)\left(y_t^{(r)} - \hat{\mu}^{(j)}\right)^T}{\sum_{r=1}^R \sum_{t=1}^{T^{(r)}} \gamma_t^{(rj)}} \tag{17}$$

For the second or M-step of this algorithm, we maximise the likely hood of the data given these alignments. We can derive a similar re-estimation equation for transition probabilities.

$$\hat{a}_{ij} = \frac{\sum_{r=1}^R \frac{1}{P^{(r)}} \sum_{t=1}^{T^{(r)}} \alpha_t^{(ri)} a_{ij} b_j\left(y_{t+1}^{(r)}\right) \beta_{t+1}^{(rj)} y_t^{(r)}}{\sum_{r=1}^R \sum_{t=1}^{T^{(r)}} \gamma_t^{(ri)}} \tag{18}$$

If initially we have some estimate of the parameters, $\lambda^{(0)}$, then using the successive iterations of this EM algorithm will yield parameter sets $\lambda^{(1)}, \lambda^{(2)}, \ldots$ which improve the likelihood up to some local maximum. Most common choice for the initial parameter $\lambda^{(0)}$ is to assign the global mean and co-variance of the data to the Gaussian output distributions and to set all transition probabilities to be equal. This is known as *flat start model.*

As in this approach of acoustic modelling , all speech utterances are represented by concatenating a sequence of phone models together, it is analogous to *beads on a string.*The biggest issue with this is that decomposing each vocabulary word into a sequence of context-dependent base phones fails to capture the very large degree of context-dependent variations that exists in real speech. For example, according to context and influence of preceding and following consonant, the pronunciations for the "oo" in "mood" and "cool" would be drastically different. Context independent phone models are referred to as *monophones.*
To mitigate this problem, we could use a unique phone model for every possible pair of left and right neighbours. These models are then called *triphones.* If there are N base phones, then there are N^3 potential triphones.

`ContextDep.PNG`

Here the notation $x - p + y$ denotes the triphone corresponding to phone p spoken in context of a preceding phone x and a following phone y. Now to avoid data sparsity problems that will result from this method, the complete set of logical triphones L can be mapped a reduced set of physical models P by clustering and tying together the parameters in each cluster.

`Tiedstate.PNG`

Each base phone pronunciation q is derived by simple look-up from the pronunciation dictionary. These are then mapped to logical phones according to the context, finally the logical phones are mapped to physical models. Important thing to observe here is that the context-dependence spreads across word boundaries, as seen in the /p/ in "stop that" has its burst suppressed by the following consonant.

Clustering of logical to physical models usually operates at the state-level rather than the model-level since it allows a larger set of physical models to be robustly estimated and is simpler too. The decision of choosing which states to tie is usually made using decision trees.Each state position of each phone q has a binary tree associated with it. In the tree, each node contains a question regarding the context.

To cluster state i of phone q, all states i of all the logical models derived from q are collected into a single pool at the root node of the tree. According to the answer of the question at each node, the set of states is successively split until all states have been divided to leaf nodes of the tree. To form a physical model, all leaf nodes are then tied together. The question at each node are selected from a predetermined set in such a way that it maximises the likelihood of training data given the final set of state-tying. In case when the state output distributions are single component Gaussians and the state occupation counts are known, then we can calculate the increase in likelihood ,achieved from splitting the Gaussians at any node ,from the counts and model parameters without reference ot the training data. Similarly, the decision trees can be formed efficiently using a greedy-iterative node splitting algorithm.

decisiontre.PNG

In the above figure, the logical phones $s-aw+n$ and $t-aw+n$ are assigned to leaf node 3 and hence they will share the same central state of representative physical model.

The major advantage for splitting the states using phonetically driven decision trees is that required logical models that were not seen at all in training data can easily be synthesised. The disadvantage of using this partitioning is that it can be quite coarse. We can reduce this coarse partitioning using the *soft-tying* scheme. In soft-tying scheme, each state is grouped with its one or two nearest neighbours and their Gausians are pooled by a post-processing stage. By doing this, we turn the signal Gaussian model to mixture of Gaussian models while also keeping the total number of Gaussians in the system constant.

Summary of acoustic model:-

- We create a flat-start monophone set in which each base phone is a monophone single-Gaussian HMM with means and co-variances equal to the mean and co-variance of training data.
- Three to Four iterations of EM algorithm are used to re-estimate the Gaussian monophone parameters.
- Each Gaussian monophone (q) is cloned for each distinct triphone $(x-q+y)$ in training data.

- Using the EM algorithm the resulting set of training data triphones is again re-estimated and state occupation counts of last iteration are stored.
- For each state in each phone, a decision tree is created. The training-data triphones are mapped into a smaller set of tied-state triphones and iteratively re-estimated using EM algorithm.
- We finally get the required tied-state context dependent acoustic model set.

5 Language Models

In essence, language models are the sort of models that give probability to word sequences. We'll understand the easiest model, the n-gram, which assigns probabilities to phrases and word sequences.

Let's begin with the $P(w \mid h)$ equation, the probability of word w, given some history, h. One way to estimate the probability function is through the relative frequency count approach, where we count the times we saw the history h and then count how many times did the word w follow it. It is not feasible to perform this over an entire collection.

This shortcoming and ways to decompose the probability function using the chain rule serves as the base intuition of the N-gram model. Here we can approximate it using just a few historical terms instead of computing probability using the entire collection.

5.1 N-Gram

Let's start with the bigram model which approximates the probability of a word given all the previous words by using only the conditional probability of one preceding word. When we use a bigram model to predict the conditional probability of the next word, we are making use of the following approximation:

$$P(w_i \mid w_1, w_2, ..., w_{i-1}) \approx P(w_i \mid w_{i-1}) \tag{19}$$

This assumption that the probability of a word depends only on the previous word is also know as Markov assumption.

The bigram model can be further generalized to the trigram model which looks two words into the past.

$$P(w_i \mid w_1, w_2, ..., w_{i-1}) \approx P(w_i \mid w_{i-2}, w_{i-1}) \tag{20}$$

It can be further generalized to the N-gram model.

$$P(w_i \mid w_1, w_2, ..., w_{i-1}) \approx P(w_i \mid w_{i-n+1}, ..., w_{i-1}) \tag{21}$$

To estimate the probability function we use Maximum Likelihood Estimation (MLE). For example, to compute a particular bigram probability of a word y given a previous word x, we can determine the count of the bigram C(xy) and normalize it by the sum of all the bigrams that share the same first-word x.

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})} \tag{22}$$

Challenges

Sensitivity

The N-gram model is significantly dependent on the training corpus. As a result, the probabilities often encode particular facts about a given training corpus. Besides, the performance of the N-gram model varies with the change in the value of N. In addition, we will have a language task in which we know all the words that may exist, and so we know the scale of vocabulary beforehand. The closed vocabulary assumption assumes there are no unknown words, which is unlikely in practical scenarios.

Smoothing

Sparse data is a notable concern with the MLE solution. Meaning, any N-gram that has occurred a large number of times might have a reasonable estimate for its probability. But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it.