Speech to Text Recognition using Hidden Markov Model

Dinesh (202211020), Kamal Meena (202211036), Pankaj Kumar (202211064)
Group Name: M416

***Abstract*—Speech recognition is a pivotal domain in artificial intelligence and human-computer interaction, facilitating seamless communication between humans and machines. This research employs Hidden Markov Models (HMMs) to recognize speech from audio data. Key features such as Mel-Frequency Cepstral Coefficients (MFCCs) are extracted from .wav files to capture essential speech characteristics. The HMM framework is then applied to model the temporal patterns of speech through Forward, Backward, Baum-Welch, and Viterbi algorithms. The study demonstrates the efficacy of HMM-based speech recognition and its advantages over alternative methods.**

Introduction

Speech recognition helps computers understand human speech and turn it into text. It is used in many applications, like virtual assistants and automatic subtitles. In this project, we use Hidden Markov Models (HMMs) to recognize speech.

We start by taking audio files in .wav format and extracting important features that represent the speech. One of the main features we use is Mel-Frequency Cepstral Coefficients (MFCCs), which help identify patterns in speech sounds.

Using HMM, we model the speech to find the text it represents. We use algorithms like Forward, Backward, Baum-Welch, and Viterbi to make this process accurate. This project shows how HMM can be used to create a simple and effective speech recognition system.

Data Engineering

Data engineering involves the preprocessing and feature extraction steps necessary for effective speech recognition.

Preprocessing the Data

- **Removing Null Values:** Any samples with missing or incomplete data are removed to ensure the dataset's quality.
- **Removing Duplicates:** Duplicate audio samples are identified and removed to avoid bias in the data.

Feature Engineering

Once the data is cleaned, we extract meaningful features from the audio files to represent the speech signals. These features help the Hidden Markov Model (HMM) understand and recognize patterns in speech.

- **Mel-Frequency Cepstral Coefficients (MFCCs):** These features capture the important characteristics of speech signals, like pitch and tone.
- **Additional Features:** Other features such as energy, zero-crossing rate, and spectral entropy may also be used to enhance the model's performance.

Hidden Markov Model (HMM)

A Hidden Markov Model (HMM) is a statistical model that is used to represent systems which are assumed to follow a Markov process with hidden states. In an HMM, the system is modeled by a set of states, and transitions between these states have certain probabilities. These states are hidden, meaning we cannot directly observe them, but we can observe outputs that are probabilistically related to the states.

How HMM Works

States: In speech recognition, the states of the HMM represent different phonemes or components of speech.

Observations: The observations in speech recognition are the extracted features (such as MFCCs) from the audio files. These features are related to the hidden states, but the exact state cannot be directly observed.

Transitions: The model also defines the transition probabilities between states, which determine how likely one state is to transition to another. These probabilities are learned during training.

Emission Probabilities: The emission probabilities define the likelihood of observing a particular feature given a certain state.

Using HMM for Speech Recognition

In the context of speech recognition, HMM is ideal because speech is inherently sequential, and the model can capture the temporal dependencies between different speech sounds. Here's how it helps:

Training: We train the HMM using a dataset of speech samples, where the system learns the transitions between speech sounds (states) and the probability of certain features being observed.

Recognition: After training, given a new speech input, the HMM uses the Forward and Backward algorithms to find the most likely sequence of states (phonemes) corresponding to the speech features. The Viterbi algorithm is used to decode the most likely sequence of states that produced the observed speech features.

Baum-Welch Algorithm: This is used to train the model by iterating between the Expectation (E-step) and Maximization (M-step) steps to estimate the model parameters.

Why HMM?

Hidden Markov Models (HMMs) are well-suited for speech recognition due to their ability to handle sequential data and model temporal dependencies in speech. Some key reasons why HMM is a good choice are:

- **Sequential Nature**: HMMs effectively model the sequential nature of speech, where each sound depends on the previous one.
- **Probabilistic Model**: HMMs can handle uncertainty and variability in speech, such as different accents and noisy data.
- **Capturing Temporal Dependencies**: HMMs capture the time-dependent patterns in speech, making them ideal for continuous speech recognition.
- **Efficiency**: Algorithms like Forward-Backward and Viterbi ensure efficient computation and real-time recognition.

GETTING THE MOST LIKELY SEQUENCE OF OBSERVATIONS IN HMM

In Hidden Markov Models (HMMs), the primary goal is to find the most likely sequence of hidden states (in our case, the phonemes or speech units) that generated a given sequence of observations (extracted features like MFCCs from the speech). This is done by maximizing the joint probability of the state sequence and the observations.

The formula to find the most likely sequence of observations in an HMM is based on Bayes' theorem and the Markov property, and is typically computed using algorithms like the Viterbi algorithm or the Forward-Backward algorithm.

Given an observed sequence of features $O = \{o_1, o_2, \ldots, o_T\}$, the goal is to compute the most likely sequence of states $S = \{s_1, s_2, \ldots, s_T\}$ that generated those observations. This can be formulated as:

$$P(S|O) = \frac{P(S, O)}{P(O)}$$

Where:

- $P(S|O)$ is the probability of the state sequence S given the observations O,
- $P(S, O)$ is the joint probability of the state sequence and the observation sequence,
- $P(O)$ is the probability of observing the sequence O, which is a normalization factor.

The brute-force approach to finding the most likely sequence of states involves evaluating all possible permutations of the states. The idea is to compute the joint probability for each possible state sequence and then choose the sequence with the highest probability. The formula for this brute-force approach can be expressed as:

$$P(S|O) = \arg\max_S P(S, O)$$

$$= \arg\max_S \left(P(s_1) \prod_{t=2}^{T} P(s_t|s_{t-1}) P(o_t|s_t) \right)$$

Where:

- $P(s_1)$ is the initial state probability,
- $P(s_t|s_{t-1})$ is the transition probability from state s_{t-1} to state s_t,
- $P(o_t|s_t)$ is the emission probability of observing o_t given state s_t,
- T is the total number of time steps (or observations),
- S is the state sequence we want to find, and O is the observed sequence.

For each possible permutation of state sequences S, we calculate the product of the probabilities $P(s_1)$, $P(s_t|s_{t-1})$, and $P(o_t|s_t)$, and select the state sequence S that maximizes the above expression.

The total number of possible state sequences grows exponentially with the number of time steps T and the number of states N. Specifically, if there are N possible states, the number of state sequences is N^T, leading to an exponential time complexity:

$$O(N^T)$$

This shows the inefficiency of the brute-force approach for large values of T and N, making it impractical for real-world speech recognition tasks.

OVERCOMING THE INEFFICIENCY OF BRUTE FORCE USING FORWARD AND BACKWARD ALGORITHMS

In speech recognition using Hidden Markov Models (HMM), finding the most likely sequence of hidden states involves evaluating all possible state sequences, which leads to an exponential time complexity. This brute-force approach is computationally expensive and inefficient, particularly for large datasets and numerous states.

The Forward and Backward algorithms offer a more efficient solution to this problem. These algorithms significantly reduce the computational complexity by considering the entire sequence of observations in a dynamic programming framework. Instead of evaluating every possible state sequence, they calculate the probability of being in a particular state at a given time, given the observed data, in a recursive manner.

Forward Algorithm

The Forward algorithm computes the probability of observing the entire sequence up to time t and being in state i at time t, given the observation sequence. It reduces the number of calculations by storing intermediate results in a table, which can be reused during computation.

The Forward probability $\alpha_t(i)$ is defined as:

$$\alpha_1(i) = \pi_i b_i(o_1)$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(o_{t+1})$$

Backward Algorithm

The Backward algorithm computes the probability of observing the remaining part of the sequence from time $t+1$ to T, given that the system is in state i at time t. Like the Forward algorithm, it stores intermediate results to avoid redundant calculations.

The Backward probability $\beta_t(i)$ is defined as:

$$\beta_T(i) = 1$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

Improved Efficiency

By using the Forward and Backward algorithms, we can avoid the exponential time complexity of the brute-force approach. The total complexity of these algorithms is linear with respect to the number of time steps and states, specifically $O(T \times N^2)$, where T is the length of the observation sequence, and N is the number of states in the HMM. This makes the Forward and Backward algorithms much more efficient and feasible for real-time speech recognition applications.

TRAINING THE MODEL

A. Basics of the Baum-Welch Algorithm

The Baum-Welch algorithm is an Expectation-Maximization (EM) algorithm used to estimate the parameters of a Hidden Markov Model (HMM). It iteratively adjusts the model parameters (A, B, π) to maximize the likelihood of the observed sequence V^T. The algorithm operates in two main steps:

- **E-Step (Expectation Step):** Using the current estimates of the model parameters, compute the probabilities of the latent variables, such as state occupancy $(\gamma_t(i))$ and state transition $(\xi_t(i, j))$. These probabilities are derived using the Forward (α) and Backward (β) probabilities.
- **M-Step (Maximization Step):** Update the model parameters (A, B, π) based on the expected values of the latent variables obtained from the E-Step. This improves the likelihood of the observation sequence given the model.

The Baum-Welch algorithm repeats these steps until the improvement in log-likelihood is below a predefined threshold, indicating convergence.

B. Detailed Explanation

If we know the probability of a given transition from i to j at time step t, then we can sum over all the T times to estimate for the numerator in our equation for $\hat{a}$. By the way, $\hat{a}$ is just the matrix representation of a_{ij}, so don't be confused.

We can define this as the probability of being in state i at time t and in state j at time $t+1$, given the observation sequence and the model. Mathematically:

$$P(s(t) = i, s(t+1) = j \mid V^T, \theta)$$

We already know from basic probability theory that:

$$P(X, Y \mid Z) = P(X \mid Y, Z)P(Y \mid Z)$$

$$P(X \mid Y, Z) = \frac{P(X, Y \mid Z)}{P(Y \mid Z)}$$

We can now say:

$$P(s(t) = i, s(t+1) = j \mid V^T, \theta) = \frac{P(s(t) = i, s(t+1) = j, V^T \mid \theta)}{P(V^T \mid \theta)}$$

The numerator of the equation can be expressed using Forward and Backward Probabilities:

$$P(s(t) = i, s(t+1) = j, V^T \mid \theta) = \alpha_i(t) a_{ij} b_{jk} v(t+1) \beta_j(t+1)$$

The denominator $P(V^T \mid \theta)$ is the probability of the observation sequence V^T by any path given the model θ. It can be expressed as the marginal probability:

$$P(V^T \mid \theta) = \sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i(t) a_{ij} b_{jk} v(t+1) \beta_j(t+1)$$

We will define $\xi_{ij}(t)$ as the latent variable representing $P(s(t) = i, s(t+1) = j \mid V^T, \theta)$. We can now define $\xi_{ij}(t)$ as:

$$\xi_{ij}(t) = \frac{\alpha_i(t) a_{ij} b_{jk} v(t+1) \beta_j(t+1)}{\sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i(t) a_{ij} b_{jk} v(t+1) \beta_j(t+1)}$$

The $\xi_{ij}(t)$ defined above is only for one time step. We need to sum over all t to get the total joint probability for all the transitions from hidden state i to hidden state j. This will be our numerator of the equation of a_{ij}.

For the denominator, we need to get the marginal probability which can be expressed as following:

$$\sum_{t=1}^{T-1} \sum_{j=1}^{M} \xi_{ij}(t)$$

Now we can define $\hat{a}_{ij}$ as:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \sum_{j=1}^{M} \xi_{ij}(t)} 1$$

1) Probabilistic View of the Denominator: Before we move on estimating B, let's understand more about the denominator of $\hat{a}_{ij}$. The denominator is the probability of a state i at time t, which can be expressed as:

$$p(s(t) = i | V^T, \theta) = \frac{p(s(t) = i, V^T | \theta)}{p(V^T | \theta)}$$

$$= \frac{p(v(1) \cdots v(t), s(t) = i | \theta) p(v(t+1) \cdots v(T) | s(t) = i, \theta)}{p(V^T | \theta)}$$

$$= \frac{\alpha_i(t) \beta_i(t)}{p(V^T | \theta)}$$

$$= \frac{\alpha_i(t) \beta_i(t)}{\sum_{i=1}^{M} \alpha_i(t) \beta_i(t)} = \gamma_i(t) 9$$

This expression shows that the denominator in the expression for $\hat{a}_{ij}$ is simply the quantity $\gamma_i(t)$, which represents the probability of being in state i at time t given the observations up to time T.

2) Estimation for A: If we use the above equation to define our estimate for A, it will be:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)} 2$$

This is the same equation as (1) we derived earlier.

However, since $\gamma_i(t) = \sum_{j=1}^{M} \xi_{ij}(t)$, we can just use $\xi_{ij}(t)$ to define the $\hat{a}_{ij}$. This will save some computation.

In summary, in case you see the estimate of a_{ij} with this equation, don't be confused, since both (1) and (2) are identical, even though the representations are different.

3) Derivation of b_jk: b_j is the probability of a given symbol v from the observations v, given a hidden state j. We already know the probability of being in state j at time t, which is:

$$\gamma_j(t) = \frac{\alpha_j(t)\beta_j(t)}{\sum_{i=1}^{M} \alpha_i(t)\beta_i(t)} 4$$

We can compute b_jk by using $\gamma_j(t)$:

$$b_j(k) = \frac{\sum_{t=1}^{T} \gamma_j(t) \cdot 1(v(t) = k)}{\sum_{t=1}^{T} \gamma_j(t)}$$

where $1(v(t) = k)$ is the indicator function that is 1 when $v(t) = k$, and 0 otherwise.

EMISSION PROBABILITY IN OUR CASE

In our model, we assume that the observations corresponding to each hidden state follow a **Gaussian distribution**. The **emission probability** is the likelihood of observing a particular feature k_t given that the system is in state i.

Since we assume that the observations in each state are Gaussian-distributed, the emission probability $b_i(k_t)$ for state i and observation k_t is given by:

$$b_i(k_t) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(k_t - \mu_i)^2}{2\sigma_i^2}\right)$$

Where:

- $b_i(k_t)$ is the emission probability of observing k_t from state i.
- μ_i is the **mean** of the Gaussian distribution for state i.
- σ_i^2 is the **variance** of the Gaussian distribution for state i.
- k_t is the observed feature at time t.

C. Parameters: Mean (μ_i) and Variance (σ_i^2)

- **Mean (μ_i)**: The mean μ_i represents the central tendency of the observations in state i. During training, we estimate μ_i by taking the **weighted average** of all the observations k_t associated with state i, weighted by the state occupation probabilities $\gamma_t(i)$. This gives:

$$\mu_i = \frac{\sum_{t=1}^{T} \gamma_t(i) \cdot k_t}{\sum_{t=1}^{T} \gamma_t(i)}$$

- **Variance (σ_i^2)**: The variance σ_i^2 represents the spread or dispersion of the observations around the mean in state i. We estimate σ_i^2 by calculating the **weighted average** of the squared differences between the observations k_t and the mean μ_i, again weighted by the state occupation probabilities $\gamma_t(i)$. This gives:

$$\sigma_i^2 = \frac{\sum_{t=1}^{T} \gamma_t(i) \cdot (k_t - \mu_i)^2}{\sum_{t=1}^{T} \gamma_t(i)}$$

These two parameters, μ_i (mean) and σ_i^2 (variance), are updated iteratively during the training phase to better fit the observed data.

MOST LIKELY SEQUENCE

The **Viterbi Algorithm** is a dynamic programming method used to find the most likely sequence of hidden states in a *Hidden Markov Model (HMM)*, given a sequence of observations. It is widely applied in fields such as speech recognition, natural language processing, bioinformatics, and computational biology.

An HMM consists of:

- A set of **states** $S = \{s_1, s_2, \ldots, s_N\}$, representing the different possible states the system can be in.
- A set of **observations** $O = \{o_1, o_2, \ldots, o_T\}$, representing the observed outputs corresponding to the hidden states.
- **Transition probabilities** $A = \{a_{ij}\}$, where $a_{ij} = P(s_j|s_i)$ represents the probability of transitioning from state s_i to state s_j.
- **Emission probabilities** $B = \{b_j(o_t)\}$, where $b_j(o_t) = P(o_t|s_j)$ represents the probability of observing o_t given that the system is in state s_j.
- An **initial state distribution** $\pi = \{\pi_i\}$, where $\pi_i = P(s_i)$ is the probability that the system starts in state s_i.

The goal of the Viterbi algorithm is to determine the most likely sequence of hidden states, $Q = \{q_1, q_2, \ldots, q_T\}$, that produced the observed sequence $O = \{o_1, o_2, \ldots, o_T\}$.

D. Algorithm Steps

1. Initialization:

At the initial time step $t = 1$, compute the probability of being in each possible state s_i given the initial state distribution π_i and the first observation o_1:

$$V_1(i) = \pi_i \cdot b_i(o_1)$$

where $V_1(i)$ represents the probability of starting in state s_i and observing o_1. The array $\psi_1(i)$ is used to store the backpointer, which is set to 0 for all i at the initialization step.

2. Recursion:

For each subsequent time step $t = 2, \ldots, T$, update the probability of being in state s_i at time t based on the previous state s_j and the transition and emission probabilities:

$$V_t(i) = \max_j \left(V_{t-1}(j) \cdot a_{ji} \cdot b_i(o_t)\right)$$

This equation represents the highest probability of transitioning from state s_j to s_i and observing o_t. The backpointer $\psi_t(i)$ records the state s_j that maximized this probability:

$$\psi_t(i) = \arg\max_j \left(V_{t-1}(j) \cdot a_{ji}\right)$$

Thus, for each state s_i, we keep track of the previous state s_j that led to the highest probability.

3. Termination:

Once the last observation has been processed at $t = T$, find the most likely last state q_T by selecting the state s_i that maximizes the probability $V_T(i)$:

$$q_T = \arg\max_i V_T(i)$$

This state q_T corresponds to the most likely final state in the sequence.

4. Backtracking:
To find the full most likely sequence of states, backtrack from the final state q_T. Using the backpointer array $\psi_t(i)$, reconstruct the sequence of states by iteratively selecting the previous state:

$$q_{t-1} = \psi_t(q_t)$$

Repeat this process for each time step $t = T-1, T-2, \ldots, 1$ until the entire sequence $Q = \{q_1, q_2, \ldots, q_T\}$ is obtained.

E. Complexity

The time complexity of the Viterbi algorithm is $O(N^2T)$, where:

- N is the number of states in the model,
- T is the length of the observation sequence.

This makes the Viterbi algorithm efficient for moderate-sized problems, but it can become computationally expensive as the number of states or observations increases.

CONCLUSION

This paper demonstrates the feasibility of using Hidden Markov Models (HMMs) for speech recognition. The process starts with cleaning and preprocessing the data, extracting features such as MFCCs, and then applying the HMM framework with algorithms like Forward, Backward, Baum-Welch, and Viterbi. The results show that HMM-based models perform well in speech recognition, capturing the temporal dependencies in speech. Further work could explore more advanced HMM variants or other machine learning models for improving accuracy.