Applications of Statistical Machine learning models in Indic Language Speech and Text processing tasks

Rashaad Baig - 210101085 Sreehari C - 210101101 Dhanesh V - 210101117 Ketan Singh - 210101118

Abstract

The project focuses on three main objectives: using Gaussian Mixture Models (GMM) for Language Identification in Gujarati, Tamil, and Telugu; employing Hidden Markov Models (HMM) for Isolated Speech Recognition to decipher spoken language patterns; and implementing Part-of-Speech (POS) tagging for sentence analysis across diverse linguistic contexts. It aims to understand statistical machine learning methods and their applications in addressing datasets and challenges specific to the Indian context.

1. Language Identification from speech

2. Dataset

We have used Gujrati, Tamil and Telugu Dataset from Microsoft Research Speech Corpus. The training set for each language consisted of 240 audio files of average length 4 seconds. These were preprocessed and feature vector was extracted according to the process described below.

2.1. Preprocessing and EDA

On raw audio, we applied Voice Activity Detection (VAD) to distinguish speech from non-speech in raw audio.This was converted into frequency domain using **FFT** and filter bank was applied and 13 **Mel-Frequency Cepstrum Coefficients** and its derivatives and double derivates were extracted to get the feature vector for further processing.

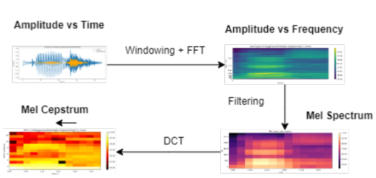


Figure 1. From Audio to Spectrum to Mel Cepstrum

From Figure 2, we can infer that as the feature index increases, the features more or less follow **normal distribution** with little variance. It also shows that most of the information is contained in the first few MFCC features.

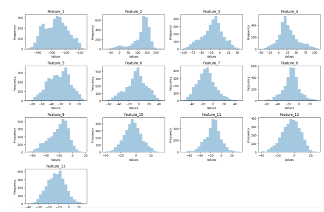


Figure 2. Each MFCC feature's plot

2.2. Why GMM

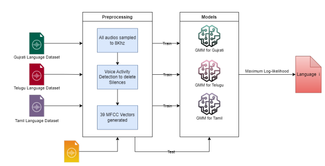


Figure 3. Basic pipeline for Language Identification using GMMs

Gaussian mixture models are **generative models** and can be used to model broad acoustic classes in a language as mixture of gaussians.

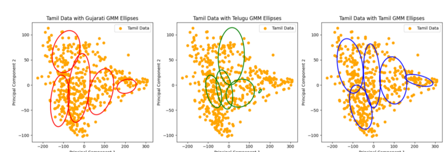


Figure 4. Each language GMM cluster

A given test voice data will be mapped to the GMM with maximum posterior probability of observed acoustic sequence, which will be of the original language as shown in Figure 4 (It can be visually seen that The Tamil GMM model only best fits the datapoints which belong to Tamil).

2.3. Algorithm Details

In GMM, we model the probability distribution of $p(x^{(i)}, z^{(i)}) = p(x^{(i)} \mid z^{(i)})p(z^{(i)})$. Here, $z^{(i)} \sim \text{Multinomial}(\alpha)$ The aim is to maximize the **log-likelihood**, which is given by $\log p(\mathbf{X}; \mu, \Sigma, \alpha) = \sum_{i=1}^{N} \log \sum_{k=1}^{K} \alpha_k \mathcal{N}(x^{(i)} | \mu, \Sigma_k)$ (1). Since there are three classes pertaining to three languages, we make 3 GMMs $\text{GMM}_1, \text{GMM}_2, \text{GMM}_3$ corresponding to them. We then preprocess each test speech occurrence X_t, calculate the log-likelihood of this data point with respect to the three models using Equation(1).Then the prediction is made as the Language i such that:

$$i = \underset{j \in \{\mathrm{GMM}_1, \mathrm{GMM}_2, \mathrm{GMM}_3\}}{\arg\max} \log p(\mathbf{X}; \mu_j, \Sigma_j, \alpha_j)$$

The algorithm to fit the dataset using **Expectation-Maximisation**(EM) algorithm is:

Algorithm 1 GMM EM Algorithm

Input: preprocessed dataset X, num_components K, num_iters $iters$
Initialize μ, α, Σ.
repeat
// Expectation Step
Calculate $w_j^{(i)} := p(z^{(i)} = j \mid x^{(i)}; \alpha, \mu, \Sigma)$ as
$w_k^{(i)} = \frac{\alpha_k \mathcal{N}(x^{(i)} | \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \alpha_j \mathcal{N}(x^{(i)} | \mu_j, \Sigma_j)}$
// Maximization Step
$\alpha_j = \frac{1}{n} \sum_{i=1}^{n} w_j^{(i)}$
$\mu_j = \frac{\sum_{i=1}^{n} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{n} w_j^{(i)}}$
$\Sigma_j = \frac{\sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^\top}{\sum_{i=1}^{n} w_j^{(i)}}$
until num_iters is done

2.4. Challenges Faced and Solutions

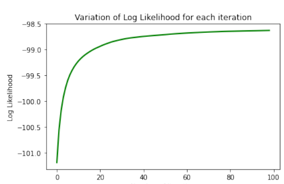


Figure 5. Change in Log-likelihood with number of iterations

We implemented GMM from scratch using numpy and python and compared it with the library implementation of `sklearn.mixtures.GaussianMixture`.We faced lot of challenges as described below:

- **Bad initialisation**: Initially random μ and Σ was used in first step of Algo 1, resulting in errors due to possibility of non-positive semi-definite Σ. To address this, the identity matrix was used. However, underflow errors occurred during Multivariate Normal PDF calculation for which data normalization and **Kmeans** was utilized for initializing cluster means, offering a more robust estimate.

- **Huge time taken**: We had used normal for loops in most of the places as outlined in Algo 1, but it was very slow for a dataset containing around 100k rows, the library version had taken advantage of numpy vectorization, which we tried to use in some places, which reduced the training time very much.

- **Inefficient computation of Multivariate PDF**: We have used normal matrix multiplication and inverses to calculate multivariate PDF, which takes $\mathcal{O}(d^3)$ and was a major bottleneck in the algorithm, the library had used Cholesky Decomposition for efficient calculation of inverses and to handle underflow errors.

2.5. Preliminary Results

The implementation was tested on 300 audio files consisting of the three different languages and the results are tabulated below. The **log-likelihood** increased in every iteration as shown in the Fig.5 which verifies the correctness of implementation.

Table 1. Accuracy per language on using all 39 MFCC components

Num. Components	Gujrati	Tamil	Telugu
32	1.00	0.43	1.00
64	1.00	0.43	1.00
120	1.00	0.51	1.00
256	1.00	0.53	1.00
512	1.00	0.57	0.74

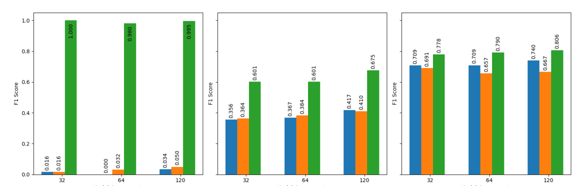


Figure 6. Number of Components v/s F1 Score Graph (Blue-13 Fea.,Orange-24 Fea.,Green-39 Fea.)

It is evident from the table that for *Tamil*, employing fewer GMM components results in underfitting, thus yielding optimal performance with 512 components. Conversely, for *Telugu*, utilizing 512 components leads to overfitting, while a more compact model with as few as 32 components achieves 100% accuracy. The perfect accuracy achieved for *Gujrati* is due to its distinct linguistic characteristics compared to *Telugu* and *Tamil*. Figure 6 indicates that Dimensionality reduction(done using PCA) could achieve the same accuracy for some languages in less time.

3. Part-of-Speech Tagging of Sentence

3.1. Why HMM

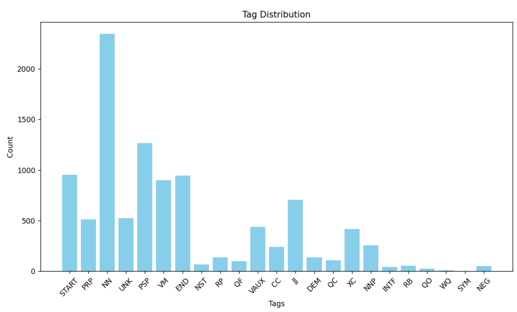
We use **HMM**, because obtained feature vector can be thought of as a time series data with each dependent on the previous ones. **HMM**s naturally capture these temporal dependencies through their probabilistic framework.

3.2. Dataset

We used pos tagged dataset scraped from web. The full dataset is given here.The training dataset around 10,000 words and more than 450 sentences.

3.3. Preprocessing and EDA

We had some words with unknown tags,with the probability of around 0.05. We tried to impute null values with **most frequent tag**(noun), basic **rule-based** imputation and **weighted random tag** imputation according to the frequency of their occurence in dataset.The training and test dataset split was 90%:10% randomly.



'

Figure 7. Tag Distributions

3.4. Algorithm

In this **supervised HMM**, we model each Part-of-the-Speech(Noun, Verb etc) as a state of the **markov** chain. Let $\boldsymbol{\theta} = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ be the parameters of an HMM, where $\mathbf{A}$ is the transition probability matrix, $\mathbf{B}$ is the observation probability matrix, $\boldsymbol{\pi}$ is the initial state distribution, which are set by **Maximum Likelihood Estimation** (MLE) of likelihood, $P(\mathbf{O}|\boldsymbol{\theta})$.

Given an observation sequence $\mathbf{O} = o_1, o_2, \ldots, o_T$, the likelihood of observing $\mathbf{O}$ given the HMM parameters $\boldsymbol{\theta}$ is given by: $P(\mathbf{O}|\boldsymbol{\theta}) = \sum_{\mathbf{q}} P(\mathbf{O}, \mathbf{q}|\boldsymbol{\theta})$ where the sum is taken over all possible state sequences $\mathbf{q} = q_1, q_2, \ldots, q_T$, and $P(\mathbf{O}, \mathbf{q}|\boldsymbol{\theta})$ is computed using the forward-backward algorithm as in Algorithm 2.

Let N be number of training examples and T_i be the length of the i-th training sequence, and n_{states} is the of possible states (POS tags).

Given the state and observation sequences for N training examples, denoted as $\{\mathbf{S}_1, \mathbf{O}_1\}, \{\mathbf{S}_2, \mathbf{O}_2\}, \ldots, \{\mathbf{S}_N, \mathbf{O}_N\}$, where $\mathbf{S}_i = s_{i1}, s_{i2}, \ldots, s_{iT_i}$ and $\mathbf{O}_i = o_{i1}, o_{i2}, \ldots, o_{iT_i}$, the MLE estimator for $\mathbf{A}$,$\mathbf{B}$, and $\boldsymbol{\pi}$ can be computed as follows:

Transition Probability Matrix $\mathbf{A}$:

$$A_{ij} = \frac{\sum_{i=1}^{N} \sum_{t=2}^{T_i} \mathbb{I}(s_{it-1} = j \text{ and } s_{it} = i)}{\sum_{i=1}^{N} \sum_{t=2}^{T_i} \mathbb{I}(s_{it-1} = j)}$$

Observation Probability Matrix $\mathbf{B}$:

$$B_{jk} = \frac{\sum_{i=1}^{N} \sum_{t=1}^{T_i} \mathbb{I}(s_{it} = j \text{ and } o_{it} = k)}{\sum_{i=1}^{N} \sum_{t=1}^{T_i} \mathbb{I}(s_{it} = j)}$$

Initial State Distribution $\boldsymbol{\pi}$:

$$\pi_i = \frac{\sum_{i=1}^{N} \mathbb{I}(s_{i1} = i)}{N}$$

where $\mathbb{I}(.)$ is the indicator function.

3.5. Smoothening techniques used

- **Laplace Smoothing:** Adding 1 to the numerator and the number of possible cases in the denominator while calculating each probability matrix to prevent zeros. This gave a mean accuracy of **0.43**.

- **Fixed Probability Smoothing:** Whenever the numerator is zero, replace that probability value with a small constant called **prob_small**, which can be used as a hyperparameter also and we can decide the best value based on **F1 score**, see 8. This was better than the previous as the mean accuracy was **0.64**.

Algorithm 2 Viterbi Algorithm

Initialize : for $i = 1$ *to* N **do**
  $\alpha_1(i) \leftarrow \pi_i \times B_i(o_1)$ Back$(1, i) \leftarrow 0$
end
for $t = 2$ *to* T **do**
  for $i = 1$ *to* N **do**
    $\alpha_t(i) \leftarrow \max_{1 \le j \le N} (\alpha_{t-1}(j) \times A_{ji} \times B_i(o_t))$
    Back$(t, i) \leftarrow \arg\max_{1 \le j \le N} (\alpha_{t-1}(j) \times A_{ji})$
  end
end
Terminate : $P^* \leftarrow \max_{1 \le i \le N} \alpha_T(i)$ $q_T^* \leftarrow \arg\max_{1 \le i \le N} \alpha_T(i)$
Backtrack : for $t = T - 1$ *to* 1 **do**
  $q_t^* \leftarrow$ Back$(t + 1, q_{t+1}^*)$
end
The most likely state sequence is $\mathbf{q}^* = q_1^*, q_2^*, \ldots, q_T^*$*, and* P^* *is its probability.*

3.6. Preliminary results

In this way, we have implemented the model from scratch and we got the following results(on most frequency word imputed dataset):

Table 2. Performance Metrics

Method	Mean F1 Score	Accuracy
Weighted Randomized	0.59	0.67
Most Freq	0.63	0.63
Rule Based	0.63	0.63

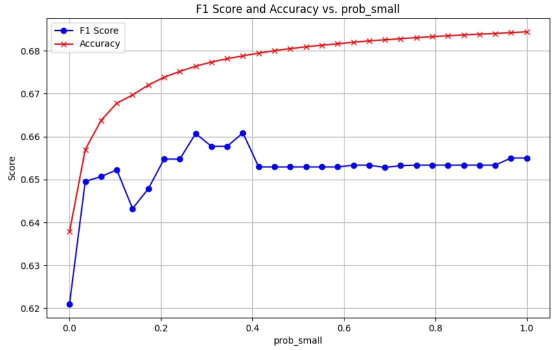


Figure 8. Scores

4. Hindi digit recogniton using GMM-HMM

4.1. Dataset

We have used Hindi Digits Dataset for this task, which contains 20 utterances of each Hindi digits from 1-9 (36% for testing).

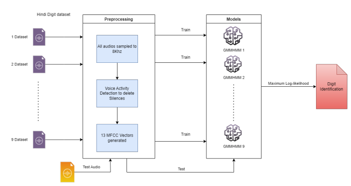


Figure 9. Basic Pipeline of Digit Identification using GMMHMM

4.2. Algorithm Details

We use a variation of HMM called **HMM-GMM**, where we model "phoneme" as a hidden state which is a mixture of Gaussians after preprocessing similar to Section 2.1.

Here, we have created a separate model for each digit and trained it on a dataset specific to that digit. To make a prediction, we calculate the likelihood of a given test dataset for all models and predict the digit that gives the maximum likelihood. The algorithm is almost similar to the one discussed in section 3.4,but with variations due to GMM. Here the probability density function of observation O_t when the model is in state i is given by: $P(O_t) = \sum_{m=1}^{M} c_m \cdot g(O_t|\mu_{im}, \Sigma_{im}), \forall i \in \{1, 2, \ldots, N\}$ and $t \in \{0, 1, \ldots, T-1\}$ such that $\sum_{m=1}^{M} c_{im} = 1$ for $i \in \{1, 2, \ldots, N\}$, where $g(O_t|\mu_{im}, \Sigma_{im})$ is multivariate normal distribution. Thus a HMM-GMM can be defined by a **5-tuple** $\boldsymbol{\lambda} = (\mathbf{A}, \boldsymbol{\pi}, \boldsymbol{c}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$.

$\xi_t(i,j)$ is defined as $P(q_t = S_i, q_{t+1} = S_j|\mathbf{O}, \boldsymbol{\lambda})$, which can be derived as:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}$$

Where α_t and β_t have the same meaning as the backwards and forward pass referenced above. Let $\gamma_t(i) = P(q_t = S_i|\mathbf{O}, \boldsymbol{\lambda}) = \sum_{j=1}^{N} \xi_t(i,j)$.

We define $\gamma_t(j,k)$ as probability of being state q_j at time t with respect to the k^{th} Gaussian mixture, $P(x_t = q_j|k, \mathbf{O}, \boldsymbol{\lambda})$, which can be written as:

$$\gamma_t(j,k) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \cdot \frac{c_{jk} N(O_t|\mu_{jk}, \Sigma_{jk})}{\sum_{m=1}^{M} c_{jm} N(O_t|\mu_{jm}, \Sigma_{jm})}$$

Thus, we can re-estimate the elements of the $\mathbf{A}$ matrix in a discrete HMM as $a_{ij} = \frac{\sum_{t=0}^{T-2} \gamma_t(i,j)}{\sum_{t=0}^{T-2} \gamma_t(i)}$. The re-estimates for the weights $c_j k$ of the Gaussian mixtures are given by, $\hat{c}_{jk} = \frac{\sum_{t=0}^{T-1} \gamma_t(j,k)}{\sum_{t=0}^{T-1} \sum_{k=1}^{M} \gamma_t(j,k)}$ and μ_{jk} and Σ_{jk} are of the form:

$$\hat{\mu}_{jk} = \frac{\sum_{t=0}^{T-1} \gamma_t(j,k) O_t}{\sum_{t=0}^{T-1} \gamma_t(j,k)} \quad \hat{\Sigma}_{jk} = \frac{\sum_{t=0}^{T-1} \gamma_t(j,k)(O_t - \hat{\mu}_{jk})(O_t - \hat{\mu}_{jk})}{\sum_{t=0}^{T-1} \gamma_t(j,k)}$$

4.3. Preliminary Results

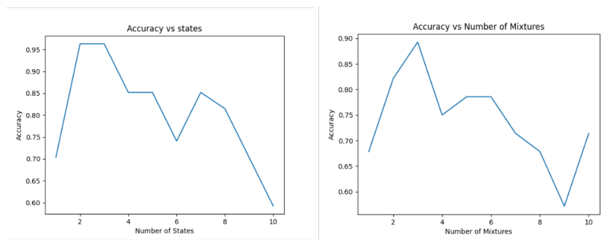


Figure 10. (a) Number of states vs Accuracy (b) Number of Mixtures vs Accuracy. Best results are for modelling with 3 Gaussian Mixtures.

Fig.10 agrees logically also as most *Hindi numbers* from 1-9 consist of 2-4 phonemes thus, we come across the best accuracy in those number of states.

We have obtained the best results by modelling each emission as a mixture of 3 GMMs as seen in the figure above.

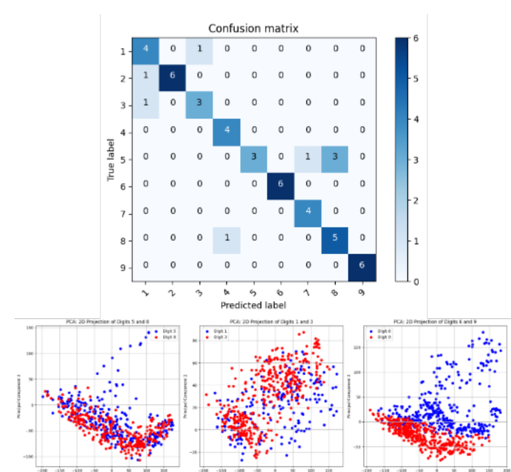


Figure 11. Confusion Matrix for testing around 50 audio samples Agrees with results from PCA analysis

The confusion matrix in Fig.11 agrees with the basic visualization by **PCA** to 2 components where the datapoints for digit 5,8 and for digits 1,3 are almost the same (5 is misclassified as 8 and vice versa in confusion matrix) while digits 6,9 are different (6,9 aren't misclassified).

5. Conclusion

After the mid-term in the GMM part, we plan to experiment with different covariance matrices and component selection criteria like **AIC**, **BIC** and Silhoutte Score and also try other models like KNN. For POS tagging part, we plan to explore some more probabilistic models like **CRF**(Conditional Random Fields) and some ensemble methods, if time permits. For Isolated word detection, we plan on improving the speed of algorithm using **vectorisation** and exploring parameter estimation using Viterbi algorithm instead of MLE.

References

Sta 4273h: Statistical machine learning. https://www.utstat.toronto.edu/~rsalakhu/sta4273/notes/Lecture11.pdf.

Athiyaa, N. and Jacob, D. G. Spoken language identification system using mfcc features and gaussian mixture model for tamil and telugu languages. *International Research Journal of Engineering and Technology (IRJET)*, 06, 2019.

Gales, M., Young, S., et al. The application of hidden markov models in speech recognition. *Foundations and Trends® in Signal Processing*, 1(3), 2008.

Joshi, N., Darbari, H., and Mathur, I. Hmm based pos tagger for hindi. volume 3, 09 2013. ISBN 9781921987007. doi: 10.5121/csit.2013.3639.

Reynolds, D. *Gaussian Mixture Models*. Springer US, Boston, MA, 2009. ISBN 978-0-387-73003-5. doi: 10.1007/978-0-387-73003-5_196.

Soehardinata, J. A Brief Introduction to Gaussian Mixture Model (GMM) Clustering in Machine Learning — joey.soehardinata. https://shorturl.at/huU78.