Hidden Markov Model for speech processing

Nguyen Van Duong[1+] **and Le Thanh Tung**[2+]

[1]K62-CACLC3
[2]K62-CACLC2
[+]these authors contributed equally to this work

ABSTRACT

Modern general-purpose speech recognition systems are based on hidden Markov models. These are statistical models that output a sequence of symbols or quantities. HMMs are used in speech recognition because a speech signal can be viewed as a piecewise stationary signal or a short-time stationary signal. In a short time-scale (e.g., 10 milliseconds), speech can be approximated as a stationary process. Speech can be thought of as a Markov model for many stochastic purposes.

1 Introduction

Hidden Markov models are popular is because they can be trained automatically and are simple and computationally feasible to use. In speech recognition, the hidden Markov model would output a sequence of n-dimensional real-valued vectors (with n being a small integer, such as 10), outputting one of these every 10 milliseconds. The vectors would consist of cepstral coefficients, which are obtained by taking a Fourier transform of a short time window of speech and decorrelating the spectrum using a cosine transform, then taking the first (most significant) coefficients. The hidden Markov model will tend to have in each state a statistical distribution that is a mixture of diagonal covariance Gaussians, which will give a likelihood for each observed vector. Each word, or (for more general speech recognition systems), each phoneme, will have a different output distribution; a hidden Markov model for a sequence of words or phonemes is made by concatenating the individual trained hidden Markov models for the separate words and phonemes. In this work, we only focus on buiding a HMM model to recognize one single word in Vietnamese language.

2 Dataset

Dataset is collected from Vnexpress with a lot of work from all members of speech processing class. First, we crop the interested pieces of speech: "benh_nhan", "co_the", "khong", "nguoi", "duoc" from the large dataset above. Then, we split the dataset in train/test with proportion of 70/30.

3 Results

Top1 accuracy is properly used here for this task.

	Accuracy
duoc	0.89
nguoi	0.8
co_the	0.96
benh_nhan	0.9
khong	0.8
MeanAcc	0.87

4 Method

4.1 Train

As mention above, we crop 100 .wav files and split them in 70 samples for train and 30 samples for validation. Training phase follows step:

4.1.1 Read raw wav file

The raw waw is read by librosa library

4.1.2 Get MFCC features

We define 25ms per window length then slide it over 10ms per hop. Then we subtract the mean to normalize the data.
To get more features, we calculate the 1st and 2nd order of original mfcc feature. Finally we concatenate all the features to a vector

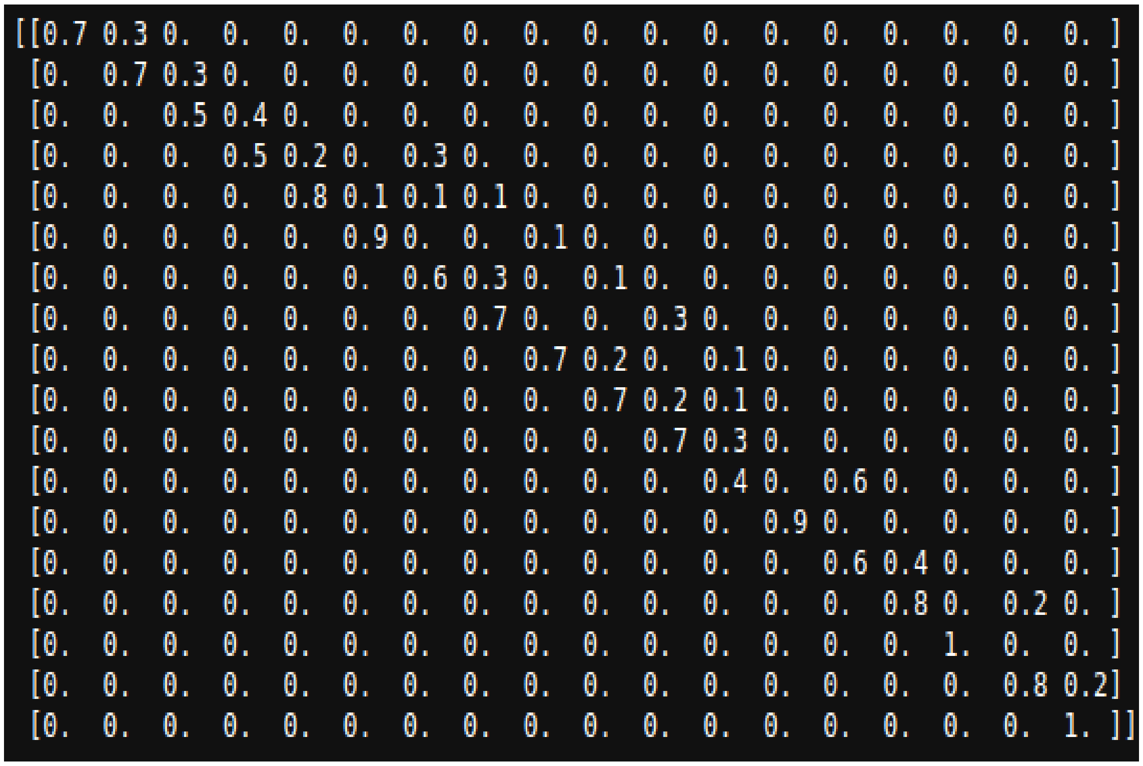
4.1.3 Clustering

We use Kmean clustering to separate the data in 10 part before put it into the HMM model

4.2 Validation

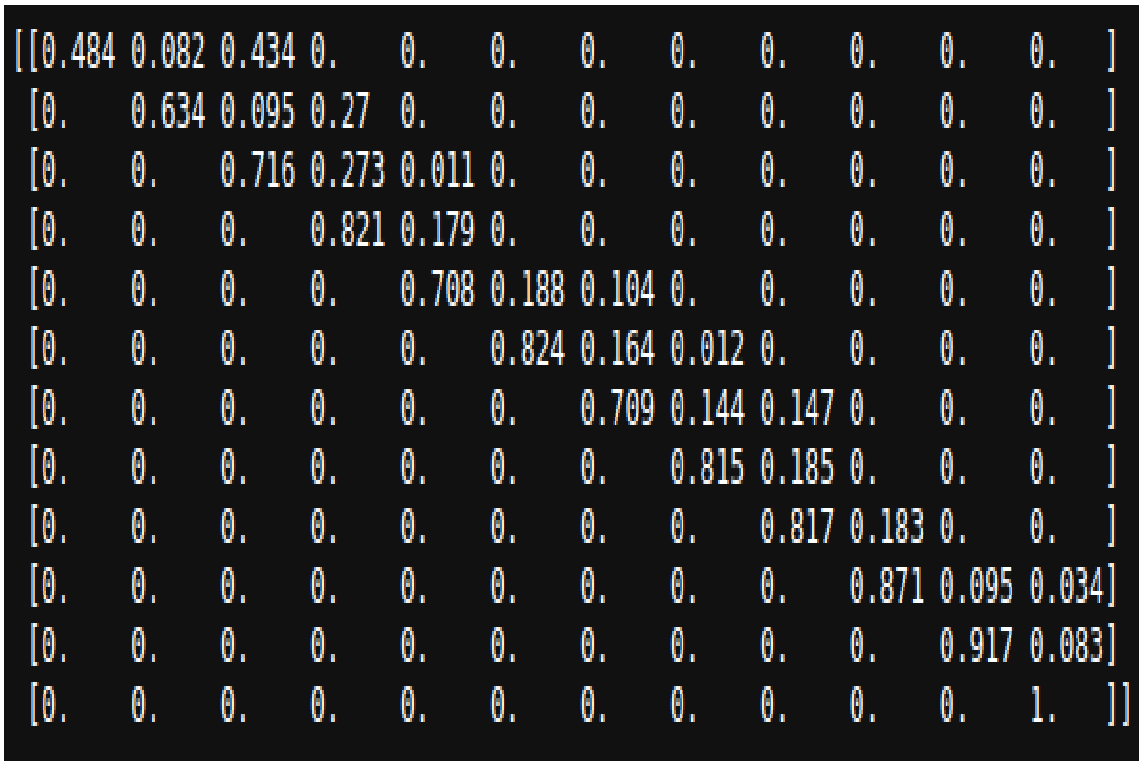
In validation phase, we do exactly the same as in training phase, but we try within or without Kmean clustering. We found out that Kmean actually improve the accuracy of validation data. Transmat matrix of:

4.2.1 Benh_nhan



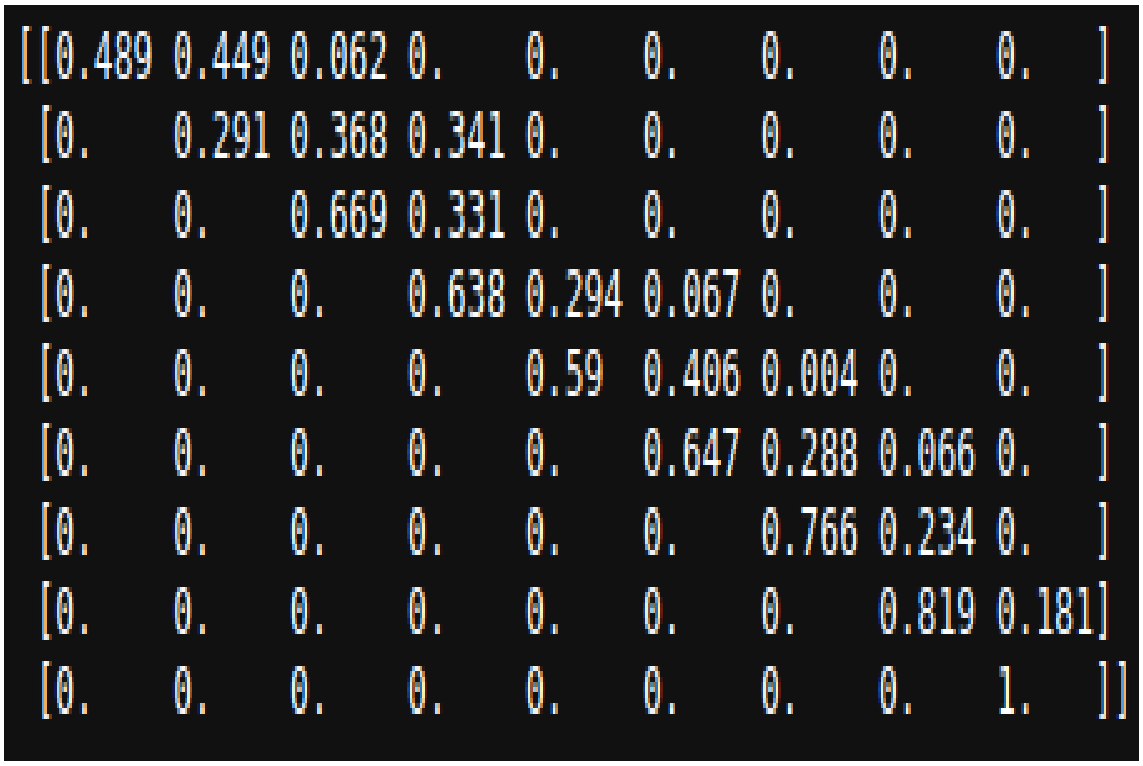
```
[[0.7 0.3 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.7 0.3 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.5 0.4 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.5 0.2 0.  0.3 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.8 0.1 0.1 0.1 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.9 0.  0.  0.1 0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.6 0.3 0.  0.1 0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.7 0.  0.  0.3 0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.7 0.2 0.  0.1 0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.7 0.2 0.1 0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.7 0.3 0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.4 0.  0.6 0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.9 0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.6 0.4 0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.8 0.  0.2 0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.8 0.2]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1. ]]
```

4.2.2 Co_the



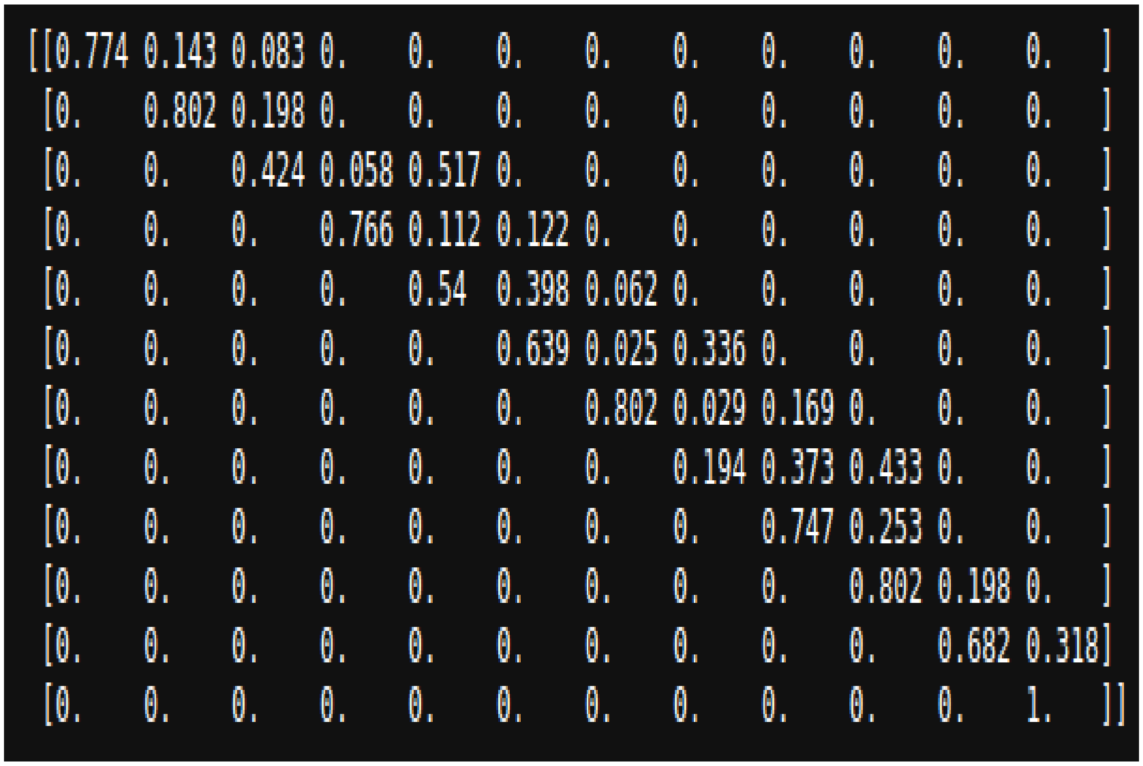
```
[[0.484 0.082 0.434 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.634 0.095 0.27  0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.716 0.273 0.011 0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.821 0.179 0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.708 0.188 0.104 0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.824 0.164 0.012 0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.709 0.144 0.147 0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.815 0.185 0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.817 0.183 0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.871 0.095 0.034]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.917 0.083]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    1.   ]]
```

4.2.3 Duoc



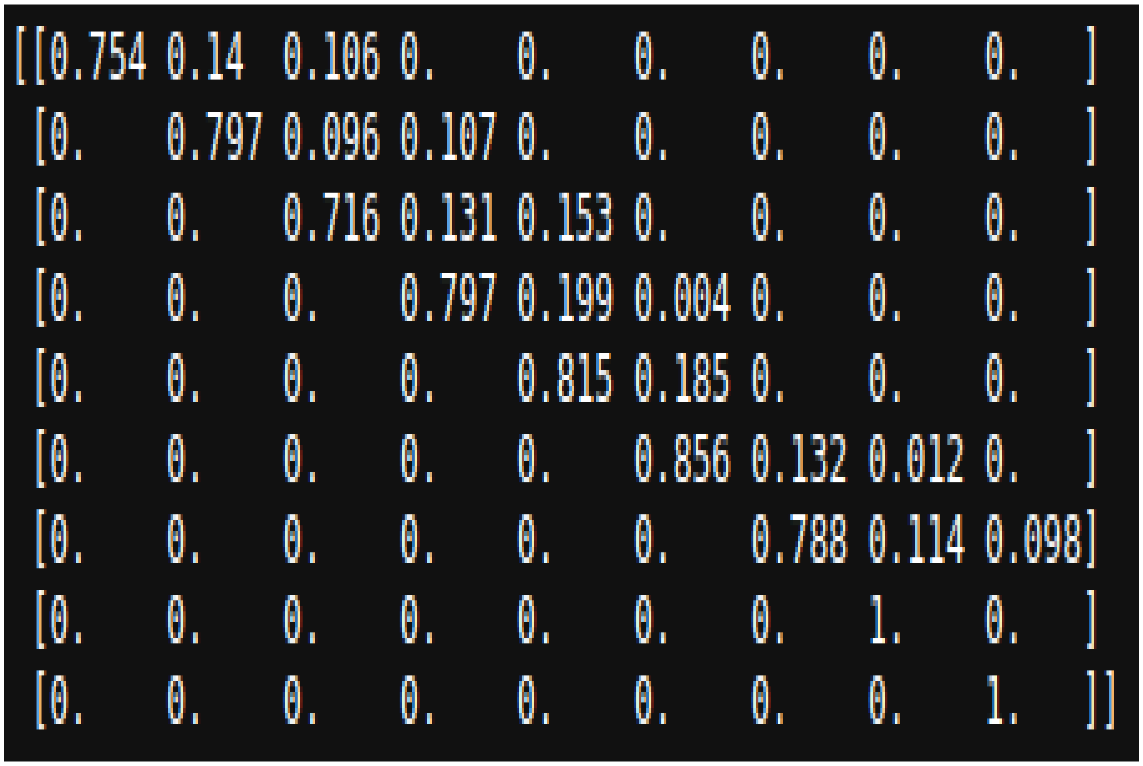
```
[[0.489 0.449 0.062 0.    0.    0.    0.    0.    0.   ]
 [0.    0.291 0.368 0.341 0.    0.    0.    0.    0.   ]
 [0.    0.    0.669 0.331 0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.638 0.294 0.067 0.    0.    0.   ]
 [0.    0.    0.    0.    0.59  0.406 0.004 0.    0.   ]
 [0.    0.    0.    0.    0.    0.647 0.288 0.066 0.   ]
 [0.    0.    0.    0.    0.    0.    0.766 0.234 0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.819 0.181]
 [0.    0.    0.    0.    0.    0.    0.    0.    1.   ]]
```

4.2.4 Nguoi



```
[[0.774 0.143 0.083 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.802 0.198 0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.424 0.058 0.517 0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.766 0.112 0.122 0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.54  0.398 0.062 0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.639 0.025 0.336 0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.802 0.029 0.169 0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.194 0.373 0.433 0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.747 0.253 0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.802 0.198 0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.682 0.318]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    1.   ]]
```

4.2.5 Khong



```
[[0.754 0.14  0.106 0.    0.    0.    0.    0.    0.   ]
 [0.    0.797 0.096 0.107 0.    0.    0.    0.    0.   ]
 [0.    0.    0.716 0.131 0.153 0.    0.    0.    0.   ]
 [0.    0.    0.    0.797 0.199 0.004 0.    0.    0.   ]
 [0.    0.    0.    0.    0.815 0.185 0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.856 0.132 0.012 0.   ]
 [0.    0.    0.    0.    0.    0.    0.788 0.114 0.098]
 [0.    0.    0.    0.    0.    0.    0.    1.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.    0.    1.   ]]
```

4.3 Test

Finally, we self-record our own data to test the model.
Although out model performs well on validation set, it is not as good as on test set. We realize that the 2 data distribution are so different that our model may overfit on training set.

5 Future work

To be more accurate on test set, we manage to collect more data and improve the model.