POLITECNICO DI TORINO

ICT For Smart Societies

ICT for Health (Lab 2)

Prof. Monica Visintin

HIDDEN MARKOV MACHINES

GENNARO RENDE

S218951

1 Introduction

1.1 Parkinson's Disease

Parkinson's disease (PD) is a progressive disease that affects nerve cells in the brain responsible for body movement. Because of the death of dopamine-producing neurons, patients affected by PD show symptoms such as slowness, tremor, balance problems and stiffness. Treatments focus on reducing symptoms to allow a more active lifestyle. Patients often have resting tremor, rigidity, akinesia and bradykinesia. The loss of muscles control affects vocal cords, thus causing speech impairment and voice disorders in 90% of the patients.

1.2 Objective of the laboratory & Data-Set

The aim of this laboratory is to develop an algorithm that can identify patients as healthy or ill by analyzing their vocal samples, for one of the most common symptoms of the disease is the inability of keeping the vocal tract opened, due to muscular contraction. The data-set is made of 20 .wav recordings of patients pronouncing the $''a''$ vowel for a certain amount of time (according to the patients' muscular possibility). The recordings have been made with a mobile phone at a sampling frequency of 8 kHz.

2 The Algorithm

2.1 How it works

The code we developed has helped us to identify healthy or ill patients by means of Hidden Markov Machines (HMMs). An HMM builds the system as a finite state machine with hidden (i.e. unobserved) states. In the first place, all 20 voice samples are processed and quantified by the function pre_process_data, then quantified samples are used in the main code (`main.m`) to train and test the two HMMs, one for the healthy and one for the ill patients.

2.2 Function `pre_process_data.m`

The Fourier transformation of the voice signal is characterized by several peaks. The one at the lowest frequency gives the fundamental frequency f. The signal is then analyzed in the time domain. Peaks are found approximately at time intervals of $\frac{1}{f_0}$. Each interval is interpolated taking $N_s - 1$ samples, while the newly obtained signal $x[n]$ is quantified using N_q values. The function outputs two cell vectors: `hq` with the 10 quantified voice signals of the healthy patients and `pq` with the 10 quantified voice signals of the patients affected by PD.

2.3 The algorithm in `main.m`

In the main code, the initial set values are N_s, N_q. Tolerance and the number of iterations, which are the HMM's parameters, are also set at the beginning of the code; the Baum–Welch algorithm is used to find the unknown parameters of the HMM by using a forward-backward algorithm. The initial transition and emission matrices are created during the training phase.

- *The emission matrix* **B** is randomly generated with dimensions of $N_s \times N_q$. Data along the rows are then normalized;
- *The transition matrix* **A** is a $N_s \times N_s$ and is generated in two different ways: randomly (normalized rows) and circularly.

The circular matrix **A** is set with a first row in the following sequence: $[q\ p\ q\ ...\ q]$ where:

- p is determined equal to 0.9 and represents the probability of changing from state 1 to state 2;
- q is defined as $q = \frac{1-p}{N_s-1}$.

The following rows have the same structure but the position of the value q is shifted by one position to the right. The Matlab function `hmmtrain` is used to train the HMMs using both the training sets of healthy and ill patients and the two different transition matrices. The function's outputs are the final emission and transition matrices to be used in the next step. The testing phase uses the Matlab function `hmmdecode` to obtain the logarithm of the probability that a given sample belongs to that machine at the end of the process. The function is applied to training and testing sets, one time by using the matrices obtained from the randomly generated **A** and another time by using the matrices obtained from the circulant **A**. To evaluate the specificity, the algorithm compares the probability of the true negative with the probability of the false positive, while the evaluation of the sensitivity is carried out by comparing the probability of the true positive with the probability of the false negative.

3 Results

The algorithm has been implemented using three different sets for training and testing, as a sort of 3-fold cross validation:

- samples 1-7 as training and 8-10 as testing
- samples 4-10 as training and 1-3 as testing
- samples 1-3, 7-10 as training and 4-6 as testing

Several trials have been made by changing the parameters and all the following results are shown as an average over the three different training and testing sets. The Mean

values of specificity and sensitivity for the two transition matrices and for different values of $N_s = N_q$ are displayed in table [**1**]. The results are very similar and as it is to be expected, in both cases they are better for the training sets than for the testing ones. A difference can be seen looking at the specificities and sensitivities for the different values of $N_s = N_q$. A small number of states gives poor results in comparison to the ones obtained with bigger values. At the same time it can be seen that values larger than $N_s = 8$ don't improve significantly the specificity or the sensitivity, actually they start to decrease.

Matrix A random					
N_s	**5**	**7**	**8**	**9**	**11**
Spec. train (%)	90,47	95,24	100	100	100
Spec. test (%)	66,67	66,67	66,67	77,78	66,67
Sens. train (%)	80,95	90,47	100	95,24	100
Sens. test (%)	55,56	88,89	84,13	88,89	77,78

Matrix A circular					
N_s	**5**	**7**	**8**	**9**	**11**
Spec. train (%)	95,24	95,24	100	100	100
Spec. test (%)	55,56	77,78	100	88,89	77,79
Sens. train (%)	66,67	100	0	95,24	100
Sens. test (%)	66,67	88,89	88,89	88,89	77,78

Table 1: Mean values of specificity and sensitivity over the three different training and testing sets. $N_s = N_q$, with $tolerance = 10^{-3}$ and number of iterations = 200

Using different values of tolerance for the `hmmtrain` function, gives the results shown in table [**3**]. As the tolerance increases, the specificities and sensitivities obtained tend to decrease slightly.

	Spec. train (%)	**Spec. test (%)**	**Sens. train (%)**	**Sens. test (%)**
random **A**	95,24	88,89	80,95	66,67
circ. **A**	95,24	66,67	85,71	66,67

(a) $N_s = 5,\ N_q = 10$

	Spec. train (%)	**Spec. test (%)**	**Sens. train (%)**	**Sens. test (%)**
random **A**	100	88,89	90,47	77,78
circ. **A**	100	88,89	90,47	77,78

(b) $N_s = 10,\ N_q = 5$

Table 2: Mean values of specificity and sensitivity evaluated for $N_s \neq N_q$, with $tolerance = 10^{-3}$ and number of iterations = 200

Changes in the results are more clearly visible when using $N_s \neq N_q$, as shown in table [**2**]. The biggest difference that we can see respect to table [**1**], is that with $N_s < N_q$ we get slightly worse results because there aren't enough states to descrive the system.

	Spec. train (%)	**Spec. test (%)**	**Sens. train (%)**	**Sens. test (%)**
random **A**	80,95	77,78	100	88,89
circ. **A**	100	100	100	88,89

(a) $tolerance = 10^{-2}$

	Spec. train (%)	**Spec. test (%)**	**Sens. train (%)**	**Sens. test (%)**
random **A**	90,47	55,56	80,95	66,67
circ. **A**	100	77,78	95,24	88,89

(b) $tolerance = 10^{-1}$

Table 3: Mean values of specificity and sensitivity for two different values of tolerance, with $N_s = N_q = 8$, and number of iterations = 200

4 Comments

Among all the changes of parameters made, we can say that the variations are not significant between the different environments created and that for this dataset, $N_s = N_q = 8$ is the best choice since it gives the highest percentage results. The small

variations are due to the poor data availability, thus training is not efficient. Not all the patients affected by this disease show speech impairment, so a classification algorithm based on voice signals should just be a starting point for the diagnosis.