

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐH NGOẠI NGỮ - TIN HỌC TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN



LUẬN VĂN TỐT NGHIỆP

TÌM HIỂU VỀ MÃ NGUỒN MỞ
CMUSPHINX VÀ XÂY DỰNG PHẦN MỀM
NHẬN DẠNG TIẾNG NÓI

GIẢNG VIÊN HƯỚNG DẪN: TS. VŨ THANH HIỀN
SINH VIÊN THỰC HIỆN: NGUYỄN ANH TUẤN (09DH11202)
HÒ THỊ HOÀNG YÊN (09DH11062)

TP. Hồ Chí Minh – 2013

LỜI CẢM ƠN

Trong suốt quá trình thực hiện báo cáo tốt nghiệp, chúng tôi đã nhận được rất nhiều sự giúp đỡ tận tình của quý thầy cô hướng dẫn cùng sự động viên từ gia đình, người thân và bạn bè. Nhân dịp hoàn thành báo cáo thực tập tốt nghiệp, chúng tôi xin gửi lời cảm ơn sâu sắc và chân thành tới:

TS Vũ Thanh Hiền – Giảng viên Khoa Công Nghệ Thông Tin Trường Đại Học Ngoại Ngữ - Tin Học TP.HCM, tuy luôn bận rộn với công việc nghiên cứu và giảng dạy riêng của mình, thầy vẫn luôn quan tâm và hướng dẫn nhóm hàng tuần để đảm bảo tiến độ luận văn. Đặc biệt, với chuyên môn và kiến thức trong lĩnh vực nghiên cứu của mình, thầy đã chỉ dẫn và hướng chúng tôi theo sát đề tài và phạm vi luận văn. Qua đó, chúng tôi đã học hỏi được rất nhiều, không chỉ kiến thức chuyên môn mà còn là thái độ nghiên cứu một cách khoa học, nghiêm túc, đúng chuẩn mực, giúp chúng tôi hoàn thành báo cáo một cách tốt nhất.

Các thầy cô giảng tại Khoa Công Nghệ Thông Tin Đại Học Ngoại Ngữ - Tin Học TP.HCM, chúng tôi đã học được từ các thầy cô những kiến thức chuyên môn quý báu trong khoảng thời gian 4 năm đại học. Có thể những kiến thức đó không trực tiếp được sử dụng trong luận văn này, nhưng phần nào đã giúp chúng tôi có một nền tảng chuyên môn vững chắc để tự tin hơn khi thực hiện luận văn.

Xin cảm ơn Ban Giám Hiệu Trường Đại Học Ngoại Ngữ - Tin Học TP.HCM đã cung cấp môi trường, cơ sở vật chất và kiến thức . Tạo cơ hội cho tất cả sinh viên trường thực hiện nghiên cứu. Xin trân trọng cảm ơn !

Chúng tôi cũng rất cảm ơn Khoa nghiên cứu khoa học của đại học Carnegie Mellon , đặc biệt là giáo sư Raj Reddy , người đứng đầu dự án mã

nguồn mở phát triển nhận dạng giọng nói CMUSphinx. Nhờ những đóng góp vô cùng to lớn của giáo sư, chúng tôi mới có thể thực hiện việc nghiên cứu của chúng tôi. Chúng em vô cùng cảm ơn !

Và đặc biệt, lời cảm ơn cuối cùng, chúng tôi xin được dành cho gia đình và những người thân của chúng tôi, những người luôn sát cánh và động viên chúng tôi trong những lúc khó khăn nhất, đó chính là nguồn động lực tinh thần quý giá giúp nhóm hoàn thành luận văn ngày hôm nay.

Một lần nữa chúng tôi cảm ơn và ghi nhận tất cả những sự giúp đỡ kể trên. Với tất cả sự nỗ lực và cố gắng của bản thân trong những tháng thực hiện, chúng tôi đã hoàn thành được đồ án, và tất nhiên sẽ không tránh khỏi những thiếu sót cần phải hoàn thiện, rất mong nhận được sự góp ý của quý thầy cô và các bạn.

TP. Hồ Chí Minh, tháng 3/2013

Nguyễn Anh Tuấn

Hồ Thị Hoàng Yến

MỤC LỤC

LỜI CẢM ƠN	2
MỤC LỤC.....	4
CHƯƠNG 1. GIỚI THIỆU TỔNG QUAN	8
1.1. Đặt vấn đề.....	8
1.2. Giới thiệu về CMU	9
1.3. Những khái niệm cơ bản trong CMU	10
1.4. Phạm vi	12
1.5. Đóng góp luận văn	13
1.6. Các khái niệm & thuật ngữ.....	13
CHƯƠNG 2. FRAMEWORK NHẬN DIỆN GIỌNG NÓI VÀ SPHINX...	15
2.1 Tổng quan về CMUSphinx	16
2.2 Áp dụng mô hình Markov ẩn (HMM) làm tiền đề để nhận dạng giọng nói trong Sphinx-4	17
2.3 Tổng quan về âm học và tiếng nói	18
2.3.1 Âm học	18
2.3.1.1 Biểu diễn tín hiệu âm thanh trong miền thời gian và tần số	18
2.3.1.2 Các loại âm thanh	19
2.3.1.3 Đơn vị đo âm thanh.....	20
2.3.2 Tiếng nói	20
2.3.3 Đặc điểm của tiếng Việt	22
2.3.3.1 Hệ thống mẫu tự và ngữ âm tiếng Việt	23
2.4 Hệ nhận dạng tiếng nói	25
2.4.1 Phân loại các hệ thống nhận dạng tiếng nói.....	25
2.4.2 Một số phương pháp nhận dạng tiếng nói.....	26
2.5 Rút trích đặc trưng tín hiệu tiếng nói	28
2.5.1 Giới thiệu.....	28
2.5.2 Làm rõ tín hiệu (pre-emphasis - tiền khuếch đại)	29

2.5.3	Tách từ	30
2.5.4	Lấy cửa sổ khung tín hiệu.....	32
2.5.5	Rút trích đặc trưng	33
2.6	Gaussian mixture model	45
2.7	Hidden Markov Model	47
2.7.1	Giới thiệu chuỗi Makov	47
2.7.2	Mô hình Markov ẩn.....	48
2.7.3	Ba bài toán cơ bản của HMM.....	51
2.8	Mixture of gaussians Hidden Markov Model	57
2.8.1	Đặc tả mô hình	57
2.8.2	Huấn luyện tham số	59
2.9	Các thành phần của Sphinx-4	61
2.10	Nhận xét.....	69
CHƯƠNG 3. CÁC PACKAGE VÀ CLASS		73
3.1	Package edu.cmu.sphinx.decoder	78
Class chính : Decoder.....		79
edu.cmu.sphinx.decoder.Decoder Class Decoder		79
3.2	Package edu.cmu.xphinx.frontend.util	79
Class chính: AudioFileDataSource, Microphone, WavWriter.		80
edu.cmu.sphinx.frontend.util.AudioFileDataSource Class AudioFileDataSource		81
edu.cmu.sphinx.frontend.util.Microphone Class Microphone		81
edu.cmu.sphinx.frontend.util.WavWriter Class WavWriter		83
3.3	Package edu.cmu.sphinx. recognizer.....	84
Class chính : Recognizer, Recognizer.State.....		85
3.4	Packages edu.cmu.sphinx.result.....	85
Class chính: Result, Lattice.....		85
Class Result		86
Class Lattice		89
3.5	Packages edu.cmu.sphinx.util.....	92
Các class quan trọng : ConfigurationManager		93
3.6	Packages edu.cmu.sphinx.linguist.language.grammar	93
Class chính: TextAlignerGrammar.		93

3.7 Packages edu.cmu.sphinx.jsgf	95
Class:JSGFGrammar,JSGFGrammarException, JSGFGrammarParseException	95
3.8 Package com.sun.speech.engine.recognition	99
Class: BaseRecognizer, BaseRuleGrammar.....	99
3.9 Packages edu.cmu.sphinx.jsapi.....	101
Class chính: SphinxRecognizer, Sphinx4Result.....	101
CHƯƠNG 4: CÀI ĐẶT	103
4.1 Cài đặt Sphinx.....	103
4.1.2 Chuẩn bị hệ điều hành.....	103
4.1.3 Chuẩn bị các gói cài đặt Sphinx	103
4.1.4 Cài đặt Sphinx	103
4.2 Chuẩn bị bộ huấn luyện cho Sphinx	117
4.3 Cách thức thu âm	121
4.4 Tiến hành huấn luyện mô hình bằng Sphinx	122
4.5 Các phần mềm yêu cầu.....	125
4.6 Tải Sphinx-4	125
4.7 Build Sphinx-4	126
1/ Cài JSAPI 1.0	126
2/ Run ant.....	126
4.8 Tạo javadocs	127
4.9 Cách để cấu hình IDE (Eclipse, NetBeans).....	127
4.10 Chạy chương trình Demo	128
4.11 Quản lý cấu hình Sphinx-4.....	129
4.11.1 Định nghĩa các thành phần.....	130
4.11.2 Định nghĩa cấu hình dữ liệu	131
4.11.3 Các loại dữ liệu cấu hình	132
4.11.4 Các thuộc tính toàn cục	136
4.11.5 Debug file cấu hình	139
CHƯƠNG 6 : ỨNG DỤNG DEMO	156
6.1 ZipCity - demo của Sphinx4	156
1. Màn hình khởi động : ấn Speak để bắt đầu	157
2. Khi không nhận được	158

3. Khi nhận dạng đúng :	158
4. Nhận dạng sai :	159
5. Nhận dạng đúng :	159
6. Nhận dạng số không tồn tại.....	159
6.2 Demo đọc số - của nhóm.....	160
6.3 Đánh giá kết luận	161
6.3.1 Khẳng định giá trị của đề tài	161
6.3.2 Kết quả đạt được	161
6.4 Tiềm năng và hướng phát triển	162
6.4.1 Đánh giá	163
❖ Ưu điểm	163
❖ Khuyết điểm :	164
TÀI LIỆU THAM KHẢO.....	166

CHƯƠNG 1. GIỚI THIỆU TỔNG QUAN

1.1. Đặt vấn đề

Ngày nay, cùng với sự phát triển của ngành điện tử và tin học, các hệ thống máy tự động đã dần thay thế con người trong nhiều giai đoạn công việc. Máy có khả năng làm việc hiệu quả và năng suất cao hơn con người rất nhiều. Song cho đến nay vấn đề giao tiếp người – máy tuy đã được cải thiện nhiều nhưng vẫn còn rất thủ công: thông qua bàn phím và các thiết bị nhập dữ liệu khác. Giao tiếp với máy bằng tiếng nói sẽ là phương thức giao tiếp văn minh và tự nhiên nhất, dấu ấn giao tiếp người – máy sẽ mất đi mà thay vào đó là cảm nhận của sự giao tiếp giữa người với người, nếu hoàn thiện thì đây sẽ là một phương thức giao tiếp tiện lợi và hiệu quả nhất.

Do có sự khác biệt về mặt ngữ âm giữa các ngôn ngữ nên ta không thể áp dụng các chương trình nhận dạng khác để nhận dạng tiếng Việt. Một hệ thống nhận dạng tiếng nói ở nước ta phải được xây dựng trên hệ thống nhận dạng tiếng nói là tiếng Việt.

Vấn đề nhận dạng tiếng nói tiếng Việt chỉ mới được quan tâm nghiên cứu trong những năm gần đây và chưa có một chương trình nhận dạng hoàn chỉnh nào được công bố.

Trên thế giới đã có rất nhiều hệ thống nhận dạng tiếng nói (tiếng Anh), đã và đang được ứng dụng rất hiệu quả như: Via Voice của IBM, Spoken Toolkit của CSLU (Central of spoken Language Under-standing)...Nhưng trong tiếng Việt thì còn rất nhiều hạn chế.

Đề tài nghiên cứu về khái niệm của kỹ thuật tiếng nói bắt đầu rát sớm từ 1936 tại các phòng nghiên cứu Bell. Nhưng vào lúc đó, các phòng nghiên cứu của Bell đã tạm dừng kế hoạch vì đã đi sai hướng: trí thông minh nhân

tạo là chìa khóa dẫn đến thành công. Sự nghiên cứu dành dừng lại mãi đến những năm 1970, khi Lenny Baum của đại học Princeton tìm ra mô hình Markov ẩn làm tiền đề cho việc chuyển đổi qua lại giữa văn bản và âm thanh. Từ đó đến nay, HMM qua nhiều lần làm lại với sự phát triển của tiến trình máy tính, nay đã được nhiều công ty, tổ chức sử dụng rộng rãi để phát triển hệ thống nhận dạng tiếng nói của họ.

Hiện nay, nước Mỹ đang dẫn đầu trong lĩnh vực nghiên cứu nhận dạng tiếng nói tự động (ASR – Automatic Speech Recognition). Hệ thống nhận dạng tiếng nói tự động của họ được biết là có khả năng nhận diện được tiếng nói tự nhiên trong điều kiện lý tưởng chính xác trên 90%. Cũng có nhiều hệ thống nhận dạng tiếng nói khác được phát triển cho các ngôn ngữ khác, như là tiếng Mexico, tiếng Úc, tiếng Tây Ban Nha, tiếng Pháp... tuy nhiên, vì mỗi ngôn ngữ có những đặc điểm khác nhau và các lĩnh vực khác nhau, nên mỗi quốc gia, mỗi công ty đều có công nghệ nhận dạng cho riêng mình, và giữ riêng cho quốc gia hay tổ chức chứ không phổ biến ra ngoài.

1.2. Giới thiệu về CMU

CMU là viết tắt của Carnegie Mellon University, cái nôi của CMUSphinx. Trên thực tế, Sphinx không phải tài sản riêng của CMU, mà là thành quả của tập thể cộng đồng đến từ nhiều quốc gia trên thế giới.

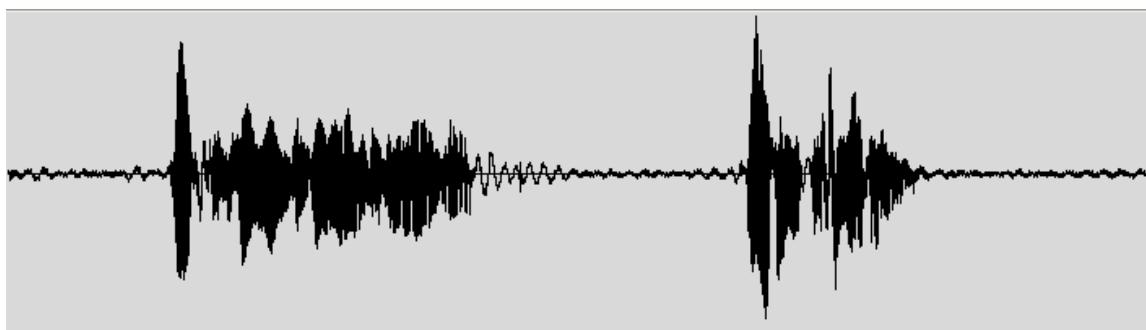
CMUSphinx được phát triển bởi nhóm các giáo sư của đại học Carnegie Mellon tại tiểu bang Pennsylvania của Mỹ, và một nhóm sinh viên của đại học này.

CMUSphinx được phát triển chủ yếu trên 2 ngôn ngữ: C# và Java. Và dựa trên tiền đề là mô hình Markov ẩn để lưu trữ và xử lý thông tin.

Ở đây chúng tôi sẽ đi sâu vào phiên bản Java của Sphinx là Sphinx4.

1.3. Những khái niệm cơ bản trong CMU

Lời nói là một khái niệm trừu tượng phức tạp. Ít ai hiểu nó được tạo ra như thế nào và nhận thức như thế nào. Nhận thức sơ khai cho rằng lời nói được tạo ra với các từ và mỗi từ tương ứng với các âm. Không may thay, thực tế lại rất khác. Lời nói là một tiến trình động mà không có các phần đặc trưng một cách rõ ràng. Nó luôn hữu dụng để dùng một chương trình chỉnh sửa âm thanh và xem lại các bản ghi âm, và nghe nó. Ở đây chúng ta ví dụ một bản ghi âm trong một chương trình chỉnh sửa.



Tất cả mô tả hiện đại của lời nói tương ứng với các mức xác suất nhất định. Điều này có nghĩa là không có một giới hạn cụ thể giữa các đơn vị, hay giữa các từ. Lời nói chuyển thành văn bản và một số ứng dụng của lời nói không bao giờ chính xác 100%. Ý tưởng này khá là hay đối với các nhà phát triển phần mềm, những người thường làm việc với các hệ thống có khả năng quyết định. Và nó tạo ra nhiều vấn đề đặc biệt đối với riêng công nghệ nói.

Cấu trúc của lời nói:

Lời nói là một dòng âm thanh liên tục với các trạng thái ổn định và bất ổn hòa lẫn vào nhau. Trong luồng của các trạng thái này, người ta có thể định nghĩa các lớp có âm thanh giống nhau nhiều hơn hoặc ít hơn, hoặc các tiếng.

Các từ được hiểu là được dùng để tạo ra các tiếng, nhưng chắc chắn rằng điều này không đúng. Các thuộc tính âm thanh của một dạng sóng âm trả về cho một tiếng có thể dựa trên nhiều nhân tố đa dạng - nội dung các tiếng, người nói, các nói, và nhiều thứ khác. Tổ chức tạo ra các phụ âm định nghĩa các tiếng rất khác với các tiêu chuẩn trình bày của họ. Ké đó, vì sự chuyển tiếp giữa các từ còn cung cấp nhiều thông tin hơn các âm vùng, các nhà phát triển thường nói về “**diphones**” (“luyến”) – các phần của âm giữa hai tiếng liên tục. Đôi khi, các nhà phát triển nói về các đơn vị phụ âm – khác với các trạng thái phụ của tiếng. Thường có ba hoặc hơn ba phần của một trạng thái tự nhiên có thể dễ dàng nhận thấy.

Con số 3 được giải thích dễ dàng. Phần đầu của tiếng phụ thuộc vào tiếng trước nó, phần giữa cố định và phần tiếp theo dựa vào các tiếng sau. Đó là lý do vì sao mà thường có 3 trạng thái trong một tiếng được chọn trong mô hình nhận dạng Markov ẩn.

Đôi khi, các tiếng được cho là có nội dung. Có những tiếng 3 âm, hoặc thậm chí là 5. Nhưng chú ý rằng không như đơn âm và hai âm, chúng phù hợp với cùng một vùng sóng âm và như các tiếng. Chúng chỉ khác tên. Đó là tại sao chúng ta thích gọi đối tượng này là **senone**. Sự phụ thuộc của một senone vào nội dung có thể phức tạp hơn chỉ nội dung bên trái và bên phải. Nó có thể được một hàm khá phức tạp định nghĩa bằng một cây quyết định, hoặc một vài cách khác.

Ké đó, các tiếng tạo nên các cụm từ, giống như các âm tiết. Thỉnh thoảng, các âm tiết được định nghĩa như “**các thực thể giảm dần đều**”. Để minh họa rõ hơn, khi lời nói trở nên nhanh, các tiếng thường bị biến dạng, nhưng các âm tiết thì vẫn giữ nguyên. Hơn nữa, các âm tiết thường bị chi phối bởi sự ngâm nga của người nói. Có các cách khác để tạo nên một cụm từ dựa trên mặt hình thái học trong một ngôn ngữ giàu hình thái hoặc dựa trên ngữ

âm. Các cụm từ thường được dùng trong các bộ từ điển mở trong quá trình nhận dạng giọng nói.

Cụm từ tạo nên các từ. Các từ thì quan trọng trong nhận dạng lời nói vì chúng có một giới hạn kết hợp giữa các tiếng theo một ý nghĩa đặc biệt. Nếu có 40 tiếng và trung bình một từ có bảy tiếng, có khoảng 40^7 từ. May mắn thay, ngay cả một người có học vấn cũng ít khi dùng hơn 20 ngàn từ trong cuộc sống, điều này làm việc nhận dạng trở nên càng khả thi hơn.

Các từ và các tiếng không thuộc ngôn ngữ, thứ chúng ta hay gọi là fillers (tiếng thở, um, uh, tiếng ho), từ nhiều lời nói. Chúng được chia từng khúc dựa vào các khoảng ngắt. Chúng không cần kết hợp các câu có nghĩa.

1.4. Phạm vi

Nhận dạng tiếng nói dần đang được sử dụng rất nhiều .Có nhiều hệ thống nhận dạng tiếng nói được ứng dụng khá hiệu quả (Tiếng Anh) .Tuy nhiên ,trong phạm vi luận văn thực tập tốt nghiệp này ,nhóm em tập trung vào việc tìm hiểu về mã nguồn mở CMUSPHINX và viết phần mềm nhận dạng giọng nói bằng tiếng Anh ,hướng đến việc viết được phần mềm nhận diện giọng nói bằng tiếng Việt bằng phương pháp MFCC (Mel-FrequencyCeptrums Coefficients) ,và nhận dạng bằng mô hình HMM (Hidden Markov Models) .Đồng thời một mô hình điều khiển bằng tiếng nói – tiếng Việt, hệ thống điều khiển bằng tiếng nói với một tập lệnh cố định .Tập lệnh này điều khiển chương trình ,cụ thể là demo mà nhóm em đang hướng phát triển.

Có rất nhiều phương pháp và hướng tiếp cận khác nhau đã được các tác giả nghiên cứu và đề xuất để huấn luyện giọng nói .Trong phạm vi luận văn tốt nghiệp ,việc tìm hiểu về mã nguồn mở CMUSPHINX và việc huấn luyện bằng tiếng Anh sẽ được thảo luận rõ hơn qua các chương sau.

1.5. Đóng góp luận văn

Trong luận văn tốt nghiệp này, chúng tôi sẽ trình bày giới thiệu về phần mềm mã nguồn mở CMUSPHINX, các thuật toán liên quan và cách thức huấn luyện giọng nói bằng tiếng Anh, phần mềm demo nhận diện giọng nói bằng tiếng Anh và hướng phát triển phần mềm nhận dạng giọng nói bằng tiếng Việt. Giúp chúng ta hiểu được cách hoạt động của CMUSPHINX, qua đó có thể phát triển các ứng dụng nhận dạng giọng nói phù hợp với nhu cầu của xã hội.

Trong quá trình tìm hiểu và phân tích mô hình mã nguồn mở CMUSPHINX, chúng tôi nhận thấy rằng tiềm năng vô cùng to lớn của CMUSPHINX, cùng với sự phát triển của công nghệ ,nhu cầu xã hội ngày càng tăng cao ,ban đầu chỉ là ở tốc độ ,dung lượng ,tiếp đó là đến sự linh động trong sử dụng khi xuất hiện các dòng sản phẩm cảm ứng ,chúng lập tức trở thành một cơn sốt trên thị trường. Trong tương lai không xa ,việc điều khiển chiếc máy tính thủ công trở nên lạc hậu, không chỉ riêng lĩnh vực thương mại mà còn nói đến các lĩnh vực khoa học công nghệ, do đó nhu cầu đơn giản hóa sự giao tiếp người – máy trở nên cần thiết.

Một đóng góp quan trọng nữa trong đề tài này là việc nghiên cứu về mô hình Markov ẩn, mô hình nền tảng để phân tích các âm thanh theo đặc trưng, giúp hệ thống xác định sự khác biệt giữa các âm thanh.

Bên cạnh đó, luận văn còn cung cấp thêm việc huấn luyện thêm tiếng Anh, để phục vụ cho việc huấn luyện và nhận diện giọng nói với quy mô từ điển nhỏ, nhóm đã xây dựng demo một phần mềm để diễn giải chi tiết hơn.

1.6. Các khái niệm & thuật ngữ

Acoustic model: mô hình âm học – là bản lưu các âm kèm với kí tự text.

VD: chữ A -> đọc là ah, được lưu theo định dạng từ điển lexicon.

Cepstra / Cepstral/ Cepstrum: Là một mô hình vec-tor đặc trưng, được dùng riêng cho kỹ thuật nhận dạng giọng nói. Hệ thống sẽ chuyển giọng của chúng ta thành mô hình này để tiện cho việc phân tích, có thể hiểu như hệ thống sẽ chuyển đổi lời nói của chúng ta thành một dãy các biến số. Khi một biến số thay đổi, nghĩa là chúng ta nói một từ khác.

Corpus (Corpora): tập lục – là một tập hợp các file giọng nói được thu sẵn để giúp máy học tốt hơn. Corpus này có thể là 1 tập tin dài có tiếng nói được thu âm sẵn (vd: các file của đài tiếng nói việt nam VOV, dài nhiều giờ đồng hồ). Máy sẽ cố gắng nhận dạng các tập tin này, và lưu lại các kết quả có thích phù hợp cao, và biến đó thành các âm chuẩn làm mẫu. Lấy ví dụ: trong cơ sở dữ liệu ta lưu từ “một” với tốc độ đọc vừa phải làm chuẩn, khi tìm thấy 1 từ “một” có tốc độ đọc nhanh hơn một tí, với cao độ và âm sắc khác đi một tí, máy sẽ cho điểm tương thích cho từ một này, đạt đến mức nhất định, hệ thống sẽ chấp nhận và lưu lại. Sau này khi ta đọc từ “một” nhanh hơn một tí, máy sẽ lập tức nhận ra ngay.

HMM: Hidden Markov Models - mô hình markov ẩn – là một mô hình dùng để phân tích các âm theo đặc trưng và giúp hệ thống xác định sự khác biệt giữa các âm thanh.

Language model: mô hình ngôn ngữ - đây như là 1 bộ từ điển sự kết hợp giữa các từ có thể có, để giúp hệ thống tiên đoán chính xác hơn. VD: từ “trái” sẽ có các từ đi sau như “cam”, “mận” ... Nhờ có mô hình ngôn ngữ mà hệ thống nhận dạng chính xác hơn.

Linguist: Cơ sở dữ liệu của Sphinx, lưu trữ các âm thanh chuẩn làm mẫu để so sánh , phân tích và nhận dạng. Gồm 2 phần là: acoustic model và language model.

Lattice: lưới – là một đồ thị có hướng diễn tả các biến trong quá trình nhận dạng. Thường là chọn ra các tổ hợp không có trong thực tế nhất ; Trong trường hợp đó đó, lưới (**lattice**) đóng vai trò trung gian để trình bày kết quả.

N-best list: danh sách các biến tốt nhất giống như **lattice** – lưới, nhưng các tổ hợp kết quả thưa hơn **lattice**.

Speech database: Cơ sở dữ liệu lời nói – là một tập hợp các đoạn ghi âm điển hình lấy từ các cơ sở dữ liệu nhiệm vụ. Nếu chúng ta phát triển hệ thống đối thoại, nó có thể đối thoại với các đoạn ghi âm của người dùng . Đối với hệ thống chính tả, nó có thể đọc các đoạn ghi âm. CSDL lời nói thường được dùng để huấn luyện, đồng điệu và kiểm tra hệ thống giải mã.

Text databases : các CSDL văn bản là các mẫu văn bản thu gom để dùng cho việc huấn luyện mô hình ngôn ngữ và nhiều ứng dụng khác. Thông thường, CSDL văn bản được thu gom trong một mẫu văn bản. Vấn đề là chuyển các tài liệu hiện tại (PDF , trang web , bản scan) thành các mẫu văn bản nói. Nghĩa là, bạn cần phải loại bỏ các thẻ và tiêu đề, mở rộng số hình thức nói của họ, và mở rộng các chữ viết tắt.

Word confusion networks : mạng các từ nghi vấn là các lưới lattice mà trật tự chặt chẽ của các nốt được lấy trong các cạnh của lưới lattice.

CHƯƠNG 2. FRAMEWORK NHẬN DIỆN GIỌNG NÓI VÀ SPHINX

Chương này sẽ giới thiệu ngắn gọn về một framework nhận dạng tiếng nói khá nổi tiếng – Sphinx4. Nó cũng cung cấp kiến thức cho người đọc hiểu được tiếng trình nhận dạng tiếng nói và cách để xây dựng một bộ máy nhận

dạng tiếng nói. Chúng tôi sẽ đặc tả các thành phần của Sphinx4 và nhiệm vụ của chúng trong toàn bộ tiến trình. Nhóm em cũng bàn đến một số kỹ thuật áp dụng vào Sphinx4 để chứng minh cho người đọc về thuận lợi và bất lợi của Sphinx4.

2.1 Tổng quan về CMUSphinx

Hệ thống Sphinx4 được viết bằng ngôn ngữ Java , được xây dựng bởi nhóm các nhà khoa học của đại học Carnegie Mellon , phòng nghiên cứu của Sun Microsystems, phòng nghiên cứu Mitsubishi , và Hewlett Packges , với sự đóng góp của đại học California Santa Cruz và MIT.

Framework Sphinx-4 là hệ thống nhận dạng tiếng nói được xây dựng trên kỹ thuật cơ sở HMM, Sphinx-4 cũng được dùng để nhận dạng nhiều ngôn ngữ trên thế giới bằng cách xây dựng lại một số tham số của Sphinx-4

Sphinx-4 hiện tại là phiên bản cao nhất của Sphinx. Nó giữ lại các hàm của các phiên bản trước và phát triển thêm một số tính năng mới.

Sphinx-4 được xây dựng trên mô hình Markov ẩn về nhận dạng tiếng nói mà được xem như là kĩ thuật phổ biến nhất hiện nay, bởi vì nó là framework chính xác nhất hiện nay, nó có khả năng cho kết quả chính xác rất cao với một kho từ vựng lớn.

Ngoài ra, Sphinx4 cũng có lợi thế về nhiều mặt mà ta ít thấy ở các framework khác như là khả năng viết theo mô-đun, plug-in. Sphinx4 cũng là một framework có tính thích nghi cao, làm việc độc lập với nhiều người đọc. Chúng ta có thể nói nhiều hơn về các thuận lợi này sau phần giới thiệu về các thành phần chính của Sphinx4.

Tổng quan về CMUSphinx Toolkit

CMUSphinx toolkit là một bộ nhận dạng giọng nói chính (chủ đạo) với các công cụ khác nhau được sử dụng để xây dựng các ứng dụng thoại. CMU Sphinx toolkit có một số gói cho các công việc và các ứng dụng khác nhau.

Đây là danh sách giúp tránh việc nhầm lẫn khi chọn lựa:

Pocketsphinx – Thư viện nhận dạng nhẹ viết bằng C.

Sphinxbase – Thư viện hỗ trợ theo yêu cầu từ Pocketsphinx.

Sphinx4 – Điều chỉnh, sửa đổi viết bằng Java.

CMUclmtk – Công cụ mô hình ngôn ngữ.

Sphinxtrain – Công cụ mô hình huấn luyện âm thanh.

Sphinx3 – Bộ giải mã cho nghiên cứu nhận dạng giọng nói được viết bằng C.

Chúng ta nên sử dụng những phiên bản mới nhất.

Sphinxbase-0.8

Pocketsphinx-0.8

Sphinx4-1.0beta6

Sphinxtrain-1.0.8

Tuy nhiên, mọi thứ dường như vẫn đang thiếu, chưa hoàn hảo, những điều như xây dựng một mô hình ngữ âm có khả năng xử lý một vốn từ vựng vô hạn, xử lý sau kết quả giải mã, ý thức khai thác và các công cụ ngữ nghĩa khác sẽ dần được thêm vào theo ngày tháng.

2.2 Áp dụng mô hình Markov ẩn (HMM) làm tiền đề để nhận dạng giọng nói trong Sphinx-4

Mô hình Markov ẩn là một mô hình trong đó hệ thống được mô hình hóa giả định là một tiến trình Markov với các ẩn số và vấn đề là làm sao xác định các tham số ẩn thông qua các tham số quan sát.

Trong tiến trình nhận dạng giọng nói, sau khi giọng nói của chúng ta được ghi lại, nó sẽ được chia thành nhiều khung mà chúng ta cần xử lý để tạo ra các câu trong text form. Với mỗi khung sẽ được biểu diễn như một trạng

thái, nhóm các trạng biếu diễn như âm vị, và nhóm các âm vị tạo thành từ mà chúng ta cần nhận diện.

Trong cở sở dữ liệu là bản mẫu ngôn ngữ (linguist), chúng ta lưu trữ các giá trị trạng thái liên quan, các âm vị, và các từ để so sánh với dữ liệu quan sát (giọng nói cần nhận diện).

Bằng cách áp dụng mô hình Markov ẩn, chúng ta dựng ra một mô hình thống kê trên mỗi âm mà trạng thái của nó ứng với một khả năng so sánh cụ thể cho giá trị tham khảo. Khả năng của mỗi trạng thái dựa vào chính nó và các trạng thái trước nó. Mục tiêu của hệ thống nhận dạng tiếng nói là tìm ra các trình tự của các trạng thái mà có xác suất cao nhất.

2.3 Tổng quan về âm học và tiếng nói

2.3.1 Âm học

Khi có nguồn phát ra âm thanh (như tiếng trống, tiếng nhạc cụ, tiếng nói), ta sẽ nghe và cảm nhận được âm thanh phát ra. Vật tạo ra được âm thanh còn được gọi là nguồn phát âm, âm thanh chính là sự dao động cơ của các thành phần vật chất trong một môi trường nào đó lan truyền và đến tai ta và khi đó ta cảm nhận được âm thanh. Trong môi trường không có vật chất tồn tại như chân không, không có dao động song cơ do đó cũng không có âm thanh tồn tại. Trong đời sống xã hội, âm thanh là phương tiện giao tiếp, truyền đạt thông tin phổ biến và xấu hiện từ lâu đời nhất của con người. Khi nghiên cứu về âm thanh, người ta thường quan tâm đến 2 đặc điểm: đặc trưng vật lý và đặc trưng sinh học.

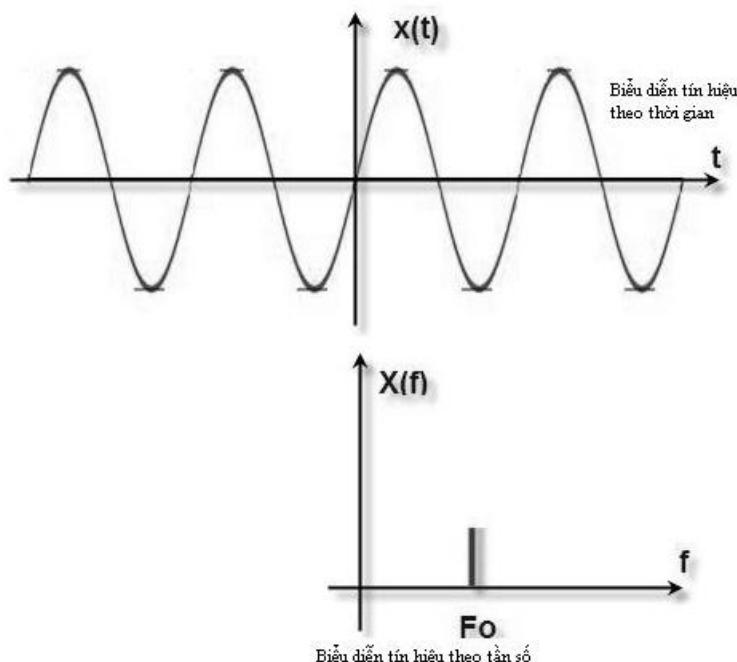
2.3.1.1 Biểu diễn tín hiệu âm thanh trong miền thời gian và tần số

Thông thường, người ta dùng hàm toán học $x(t)$ để biểu diễn âm thanh trong miền thời gian. Trong đó:

- t : thời gian.

- x : biên độ biến thiên, hay còn gọi là ly độ.

Như vậy, ta có thể biểu diễn $x(t)$ bằng đồ thị theo thời gian. Đặt $x(t) = A \sin \omega_0 t = A \sin 2\pi F_0 t$.



Hình 0.1 Biểu diễn tín hiệu âm thanh

Phổ tín hiệu: là cách biểu diễn các thành phần cấu tạo nên $x(t)$ theo tần số. với tín hiệu Sin nói trên, đồ thị phổ là một vạch có cao độ là A tại điểm có tần số F_0 . Ta nói đó là phổ vạch. Trong thực tế, với $x(t)$ bất kỳ, biến thiên, không tuần hoàn, người ta sẽ dùng phân tích Fourier để tính toán phổ tín hiệu. Khi đó, ta có phổ liên tục $X(\Omega)$.

2.3.1.2 Các loại âm thanh

Những dao động cơ mà con người nghe được gọi âm thanh (sound).

Âm thanh có thể biểu diễn theo thời gian, song cũng có thể biểu diễn theo tần số do có thể phân tích một tín hiệu âm thanh thành tổ hợp các thành phần tần số khác nhau (Chuỗi Fourier, tích phân Fourier). Hoặc nói một cách đơn giản thực tiễn hơn, một âm thanh có thể là tổ hợp từ nhiều đơn âm, từ nhiều nhạc cụ, mà mỗi cái có một tần số dao động nhất định.

Dải tần số nghe được là từ 20 Hz - 20000 Hz. Siêu âm là âm dao động ngoài 20000 Hz. Hạ âm là các âm dao động dưới 20 Hz. Tai người không nghe được siêu âm và hạ âm.

- **Tiếng nói** (voice, speech) là âm thanh phát ra từ miệng người, được truyền đi trong không khí đến tai người nghe. Dải tần số của tiếng nói đủ nghe rõ là từ 300 Hz đến 3500 Hz, là dải tần tiêu chuẩn áp dụng cho điện thoại. Còn dải tần tiếng nói có chất lượng cao có thể là từ 200 Hz-7000 Hz, áp dụng cho các ampli hội trường.
- **Âm nhạc** (music) là âm thanh phát ra từ các nhạc cụ. Dải tần số của âm nhạc là từ 20 Hz đến 15000 Hz.
- **Tiếng kêu** là âm thanh phát ra từ mồm động vật. Tiếng của Cá Heo (dolphins) là một loại âm thanh trong dải tần số 1-164 kHz, của Con Dơi (bats) 20 - 115 kHz, của Cá Voi (whale) 30-8000 Hz. (Cần xác minh lại số liệu).
- **Tiếng động** là âm thanh phát ra từ sự va chạm giữa các vật. Thí dụ tiếng va chạm của 2 cái cốc, tiếng va chạm của cánh cửa, tiếng sách rơi.
- **Tiếng ồn** (noise) là những âm không mong muốn.

Nhìn chung lại, xét về phương diện tín hiệu và sự cảm thụ của tai người, có hai loại âm:

- tuân hoà bao gồm tiếng nói, âm nhạc...
- không tuân hoà như tín hiệu tạp nhiễu, một số

2.3.1.3 Đơn vị đo âm thanh

Người ta thấy rằng con người cảm nhận độ to của âm thanh không tỉ lệ thuận với cường độ âm thanh mà theo hàm số mũ.

$Bel = 10 \lg P_2/P_1$. (Phát âm là Ben)

$decibel = 20 \lg I_2/I_1$ (Phát âm là Đề xi ben)

2.3.2 Tiếng nói

Tiếng nói là âm thanh phát ra từ miệng (người). Nghiên cứu tiếng nói gồm: Bộ máy phát âm của con người. Thụ cảm âm thanh của tai người. Phân loại tiếng nói.

Bộ máy phát âm của con người gồm:

- **Phổi** đóng vai trò là cái bơm không khí, tạo năng lượng hình thành âm.
- **Đôi dây thanh** (vocal fold, vocal cord) là hai cơ thịt ở trong cuống họng, có hai đầu dính nhau, còn hai đầu dao động với tần số cơ bản là Fo, tiếng Anh gọi là pitch, fundamental frequency. Fo của nam giới nằm trong khoảng 100-200 Hz, của nữ giới là 300-400 Hz, của trẻ em là 500-600 Hz.
- Thanh quản và vòm miệng: đóng vai như là hốc cộng hưởng, tạo ra sự phân biệt tần số khi tín hiệu dao động từ đôi dây thanh phát ra. Đáp ứng tần số của hốc cộng hưởng này có nhiều đỉnh cộng hưởng khác nhau được gọi là các *formant*.
- Miệng đóng vai trò phát tán âm thanh ra ngoài.
- Lưỡi thay đổi để tạo ra tần số formant khác nhau.
- Các âm khác nhau là do vị trí tương đối của formants.

Phân loại tiếng nói theo thanh:

- **Âm hữu thanh** (voiced, tiếng Pháp là voisé) là âm khi phát ra có sự dao động của đôi dây thanh, nên nó tuần hoàn với tần số Fo. Vì vậy phổi của nguyên âm là phổi vạch, khoảng cách giữa các vạch bằng chính Fo.
- **Âm vô thanh** (unvoiced, tiếng Pháp là non voisé) phát ra khi đôi dây thanh không dao động. Thí dụ phần cuối của phát âm English, chữ sh cho ra âm xát. Phổi tín hiệu có dạng là nhiễu trắng, phổi phân bố đều.

Phân loại tiếng nói:

- **Nguyên âm** (vowel) là âm phát ra có thể kéo dài. Tất cả nguyên âm đều là âm hữu thanh, nghĩa là tuần hoàn và khá ổn định trong một đoạn thời gian vài chục ms.
- **Phụ âm** (consonant) là âm chỉ phát ra một nhát, không kéo dài được. Có phụ âm hữu thanh và phụ âm vô thanh.

Thanh điệu của tiếng Việt tương ứng với các dấu: không dấu, huyền, hỏi, ngã, sắc, nặng khi viết. Phân tích máy móc cho thấy thanh điệu là sự thay đổi Fo, tần số cơ bản pitch, trong quá trình phát âm các nguyên âm và tai người cảm nhận được. Tiếng Việt có 6 thanh thể hiện sự phong phú và độc đáo, trong khi tiếng Trung Quốc có 4 thanh. Tuy nhiên cư dân một số vùng ở Việt Nam có thể không phân biệt dấu? và dấu ~ nên hay viết sai chính tả.

Giọng bỗng (high voiced pitch, hay high pitched) hay **giọng trầm** (low voiced pitch) là Fo cao hay thấp. Như vậy Fo đóng vai trò rất quan trọng trong cảm nhận, trong thụ cảm âm thanh của con người.

Tiếng bỗng hay **tiếng trầm** tương ứng với dải tần số cao hay thấp. Trong thực tế người ta dùng loa trầm là loa bass hay loa sub woofer, loa tép hay loa bỗng tương ứng với loa thích ứng phát các âm trong vùng tần số cao, treble.

2.3.3 Đặc điểm của tiếng Việt

Khác với một số ngôn ngữ khác như tiếng Anh, Pháp ..., tiếng Việt là ngôn ngữ đơn âm tiết, tức là các từ khi viết ra chỉ đọc lên thành một tiếng, không có từ nào (thuần Việt) phát âm từ 2 tiếng trở lên. Một từ có cấu tạo gồm 2 phần là: nguyên âm V (vowel) và phụ âm C (consonant) và được kết hợp theo 3 cách để tạo nên từ trong tiếng Việt:

- C+V (phụ âm + nguyên âm). Ví dụ: ba, mẹ, đi
- C+V+C (phụ âm + nguyên âm + phụ âm). Ví dụ: bàn, con, mong
- V+C (nguyên âm + phụ âm). Ví dụ: an, ông, én

Trong tiếng Việt, ngoài 2 thành phần chính là nguyên âm, phụ âm, còn có các thành phần khác giúp cho Việt phân loại trong âm tiết trở nên rõ ràng như nhị hợp âm, tam hợp âm, phụ âm đơn, phụ âm kép. Khi học tiếng Việt, ngay từ đọc phải học thuộc các nguyên âm, phụ âm, nhị hợp âm, tam hợp âm, phụ âm đơn, phụ âm kép, quy tắc ghép nối các thành phần đó để tạo thành âm tiết hoặc một từ, khi đó một từ tiếng Việt được Việt ra, sẽ kèm theo các đọc của từ đó bằng quy tắc kết hợp trên. Nếu một từ viết ra mà không theo quy tắc kế hợp được định sẵn trong tiếng Việt, tương đương với việc từ đó không thể đọc được và cũng không có nghĩa, một từ trong tiếng Việt chỉ có một cách đọc (trừ trường hợp tiếng vùng miền, địa phương), khác với tiếng Anh, không có quy tắc xác định trong việc tạo ra một từ, một từ chỉ tồn tại khi nó xuất hiện trong từ điển, khi đó phải kèm theo cách đọc của từ đó (pronunciation) thì mới có thể đọc được.

Tiếng Việt ngoài là một ngôn ngữ đơn âm (như nói trên) còn có yếu tố đa thanh. Đa thanh tức là có nhiều thanh điệu, nhiều dấu giọng. Cụ thể là có 6 thanh điệu, được ghi bằng 5 ký hiệu khác nhau : dấu sắc (Á), dấu huyền (À), dấu hỏi (Ã), dấu ngã (Ã), dấu nặng (Â). (Gọi tắt là 5 dấu 6 giọng). Không có dấu gọi là thanh-điệu “ngang”.

2.3.3.1 Hệ thống mẫu tự và ngữ âm tiếng Việt

Bảng chữ cái tiếng Việt có 29 chữ cái, theo thứ tự :

[a, ă, â, b, c, d, đ, e, ê, g, h, i, k, l, m, n, o, ô, ơ, p, q, r, s, t, u, ư, v, x, y]

chia làm hai phần: mẫu tự chính (khi phát âm thì gọi là nguyên âm) và mẫu tự phụ (khi phát âm thì gọi là phụ âm):

- Nguyên âm: Trong tiếng Việt, ngoài nguyên âm đơn còn có nguyên âm đôi, nguyên âm ba. Có mối liên hệ phức tạp giữa nguyên âm và cách phát âm của chúng. Một nguyên âm có thể biểu thị cho vài cách phát âm khác nhau, tùy theo nó nằm trong nguyên âm đơn, đôi hay ba; và nhiều khi các cách viết nguyên âm khác nhau tượng trưng cho cùng một cách phát âm.

Bảng 1 Cách phát âm có thể tương ứng với từng cách viết nguyên âm [8]:

Cách viết	Phát âm	Cách viết	Phát âm
a	/ə:/, /a/, /ɜ/	o	/ɔ/, /ə/, /w/
ă	/ə/	ô	/o/, /ɜw/, /ɔ/
â	/ɜ/	ơ	/ə:/, /ɜ/
e	/ɛ/	u	/u/, /w/
ê	/e/, /ɜ/	ư	/ɨ/
i	/i/, /j/	y	/ɨ/, /j/

Bảng 2 Cách viết có thể tương ứng với từng cách phát âm nguyên âm đôi và ba

Nguyên âm đôi & ba			
Phát âm	Cách viết	Phát âm	Cách viết
Nguyên âm đôi			
/uj/	ui	/iŋ/	iu
/oj/	ôi	/ɛŋ/	êu
/ɔŋ/	oi	/ɛŋ/	eo
/ə:j/	ɔi	/ə:w/	ou
/ɜŋ/	ây, ê	/ɜŋ/	âu, ô
/a:j/	ai	/a:w/	ao
/ɛŋ/	ay, a	/ɛ:w/	au, o
/iŋ/	uri	/ɪŋ/	uu
/iŋ/	ia, ya, iê, yê	/uŋ/	ua, uo
/ɪŋ/	ua, uo		
Nguyên âm ba			
/iŋŋ/	iêu, yêu	/uŋŋ/	uôi

/tʃj/	ươi	/tʃw/	ưu
-------	-----	-------	----

- Phụ âm: tiếng Việt có 17 phụ âm đơn trong tập trên gồm:

[b, c, d, đ, g, h, k, l, m, n, p, q, r, s, t, v, x]

và 11 phụ âm ghép:

[gi, gh, qu, ch, kh, ng, ngh, nh, ph, th, tr]

Trong đó chỉ có 8 phụ âm có thể nằm ở cuối từ:

[c, m, n, p, t, ng, nh, ch]

2.4 Hệ nhận dạng tiếng nói

2.4.1 Phân loại các hệ thống nhận dạng tiếng nói

2.4.1.1 Nhận dạng từ liên tục và nhận dạng từ tách biệt

Một hệ nhận dạng tiếng nói có thể là một trong hai dạng: nhận dạng liên tục và nhận dạng từng từ. Nhận dạng liên tục là nhận dạng tiếng nói được phát liên tục trong một chuỗi tín hiệu, chẳng hạn như một câu nói, một mệnh lệnh hoặc một đoạn văn được đọc bởi người dùng. Các hệ thống loại này rất phức tạp, nó phức tạp ở chỗ các từ được phát liên tục khó xử lý kịp (nếu cần thời gian thực), hoặc khó tách ra nếu như người nói liên tục không có khoảng nghỉ (thông thường rất hay xảy ra trong thực tế). Kết quả tách từ ảnh hưởng rất lớn đến các bước sau, cần xử lý thật tốt trong quá trình này. Trái lại, đối với mô hình nhận dạng từng từ, mỗi từ cần nhận dạng được phát âm một cách rời rạc, có các khoảng nghỉ trước và sau khi phát âm một từ. Mô hình loại này dĩ nhiên đơn giản hơn mô hình nhận dạng liên tục, đồng thời cũng có những ứng dụng thực tiễn như trong các hệ thống điều khiển bằng lời nói, quay số bằng giọng nói..., với độ chính xác khá cao, tuy nhiên khó áp dụng rộng rãi đối với mô hình trên.

2.4.1.2 Nhận dạng phụ thuộc người nói và độc lập người nói

Đối với nhận dạng phụ thuộc người nói thì mỗi một hệ nhận dạng chỉ phục vụ được cho một người, và nó sẽ không hiểu người khác nói gì nếu như chưa được huấn luyện lại từ đầu. Do đó, hệ thống nhận dạng người nói khó

được chấp nhận rộng rãi vì không phải ai cũng đủ khả năng kiến thức và nhất là kiên nhẫn để huấn luyện hệ thống. Đặc biệt là hệ thống loại này không thể ứng dụng ở nơi công cộng. Ngược lại, hệ thống nhận dạng độc lập người nói thì lý tưởng hơn, ứng dụng rộng rãi hơn, đáp ứng được hầu hết các yêu cầu đề ra. Nhưng không may là hệ thống lý tưởng như vậy gặp một số vấn đề, nhất là độ chính xác của hệ thống. Trong thực tế, mỗi người có một giọng nói khác nhau, thậm chí ngay cùng một người cũng có giọng nói khác nhau ở những thời điểm khác nhau. Điều này ảnh hưởng rất lớn đến việc nhận dạng, nó làm giảm độ chính xác của hệ thống nhận dạng xuống nhiều lần. Do đó để khắc phục khuyết điểm này, hệ thống nhận dạng độc lập người nói cần được thiết kế phức tạp hơn, đòi hỏi lượng dữ liệu huấn luyện lớn hơn nhiều lần (dữ liệu được thu từ nhiều giọng khác nhau của nhiều người). Nhưng điều này cũng không cải thiện được bao nhiêu chất lượng nhận dạng. Do đó, trong thực tế có một cách giải quyết là bán độc lập người nói. Phương pháp này thực hiện bằng cách thu mẫu một số lượng lớn các giọng nói khác biệt nhau. Khi sử dụng, hệ thống sẽ được điều chỉnh cho phù hợp với giọng của người dùng, bằng cách nó học thêm một vài câu có chứa các từ cần thiết (người dùng trước khi sử dụng hệ thống cần phải qua một quá trình ngắn huấn luyện hệ thống). Nhận dạng độc lập người nói khó hơn rất nhiều so với nhận dạng phụ thuộc người nói. Cùng một từ, một người, dù có cố gắng phát âm cho thật giống đi nữa thì cũng có sự khác biệt. Đối với bộ não con người, một hệ thống hoàn hảo, thì sự khác biệt đó có thể được bỏ qua do ngữ cảnh, và do có phần xử lý làm mờ đi của não. Nhưng đối với máy tính thì rất khó xây dựng được một mô hình giải quyết cho tất cả các trường hợp khác biệt đó.

2.4.2 Một số phương pháp nhận dạng tiếng nói

Có 3 phương pháp phổ biến được sử dụng trong nhận dạng tiếng nói hiện nay:

- phương pháp âm học- ngữ âm học.
- phương pháp nhận dạng mẫu.
- phương pháp ứng dụng trí tuệ nhân tạo.

2.4.2.1 Phương pháp âm học-ngữ âm học (acoustic-phonetic)

Phương pháp này dựa trên lý thuyết về Âm học-Ngữ âm học. Lý thuyết đó cho biết: tồn tại các đơn vị ngữ âm xác định, có tính phân biệt trong lời nói và các đơn vị ngữ âm đó được đặc trưng bởi một tập các tín hiệu tiếng nói. Các bước nhận dạng của phương pháp gồm:

Bước 1: Phân đoạn và gán nhãn. Bước này chia tín hiệu tiếng nói thành các đoạn có đặc tính âm học đặc trưng cho một (hoặc một vài) đơn vị ngữ âm, đồng thời gán cho mỗi đoạn âm thanh đó một hay nhiều nhãn ngữ âm phù hợp.

Bước 2: Nhận dạng. Bước này dựa trên một số điều kiện ràng buộc về từ vựng, ngữ pháp v.v... để xác định một hoặc một chuỗi từ đúng trong các chuỗi nhãn ngữ âm được tạo ra sau bước 1.

2.4.2.2 Phương pháp nhận dạng mẫu

Phương pháp nhận dạng mẫu không cần xác định đặc tính âm học hay phân đoạn tiếng nói mà sử dụng trực tiếp các mẫu tín hiệu tiếng nói trong quá trình nhận dạng. Các hệ thống nhận dạng tiếng nói theo phương pháp này được phát triển theo hai bước cụ thể là:

Bước 1: Sử dụng tập mẫu tiếng nói (cơ sở dữ liệu mẫu tiếng nói) để đào tạo các mẫu tiếng nói đặc trưng (mẫu tham chiếu) hoặc các tham số hệ thống.

Bước 2: Đôi sánh mẫu tiếng nói từ ngoài với các mẫu đặc trưng để ra quyết định. Trong phương pháp này, nếu cơ sở dữ liệu tiếng nói cho đào tạo có đủ các phiên bản mẫu cần nhận dạng thì quá trình đào tạo có thể xác định chính xác các đặc tính âm học của mẫu (các mẫu ở đây có thể là âm vị, từ, cụm từ...). Hiện nay, một số kỹ thuật nhận dạng mẫu được áp dụng thành công trong nhận dạng tiếng nói là lượng tử hóa vector, so sánh thời gian động (DTW), mô hình Markov ẩn (HMM), mạng nơron nhân tạo (ANN).

2.4.2.3 Phương pháp ứng dụng trí tuệ nhân tạo

Phương pháp ứng dụng trí tuệ nhân tạo kết hợp các phương pháp trên nhằm tận dụng tối đa các ưu điểm của chúng, đồng thời bắt chước các khả năng của con người trong phân tích và cảm nhận các sự kiện bên ngoài để áp dụng vào nhận dạng tiếng nói.

Đặc điểm của các hệ thống nhận dạng theo phương pháp này là: Sử dụng hệ chuyên gia để phân đoạn, gán nhãn ngữ âm. Điều này làm đơn giản hóa hệ thống so với phương pháp nhận dạng ngữ âm. Sử dụng mạng nơron nhân tạo để học mối quan hệ giữa các ngữ âm, sau đó dùng nó để nhận dạng tiếng nói.

Việc sử dụng hệ chuyên gia nhằm tận dụng kiến thức con người vào hệ nhận dạng:

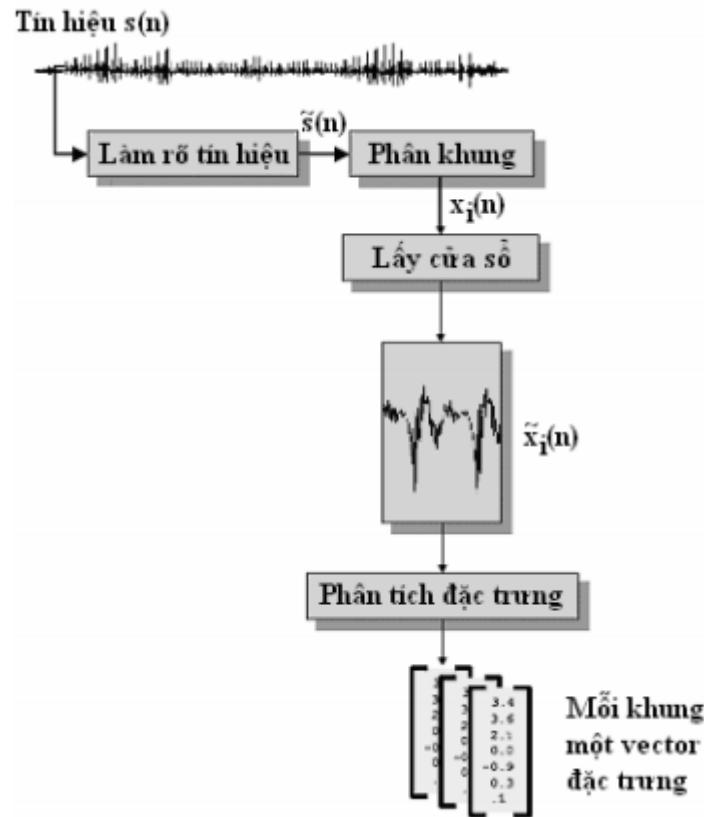
- Kiến thức về âm học: Để phân tích phổ và xác định đặc tính âm học của các mẫu tiếng nói.
- Kiến thức về từ vựng: sử dụng để kết hợp các khối ngữ âm thành các từ cần nhận dạng.
- Kiến thức về cú pháp: nhằm kết hợp các từ thành các câu cần nhận dạng.
- Kiến thức về ngữ nghĩa: nhằm xác định tính logic của các câu đã được nhận dạng.

2.5 Rút trích đặc trưng tín hiệu tiếng nói

2.5.1 Giới thiệu

Rút trích đặc trưng của tiếng nói là một trong những khâu quan trọng trong quá trình nhận dạng tiếng nói. Dữ liệu tiếng nói thông thường dưới dạng sóng âm đã lưu trữ trong máy tính là loại dữ liệu khó xử lý, học mẫu huấn luyện, và so sánh, vì thế việc rút trích đặc trưng tiếng nói là cần thiết. Kết quả của quá trình rút trích đặc trưng là 1 hoặc nhiều vector đặc trưng, các vector này chứa các tham số mang giá trị quan trọng của tín hiệu tiếng nói, làm giảm đi rất nhiều số lượng tính toán cần thực hiện, làm rõ ràng hơn sự khác biệt giữa 2 tín hiệu tiếng nói. Hình bên dưới minh họa cho quá trình rút trích đặc trưng.

Có nhiều phương pháp để thực hiện rút trích đặc trưng, 2 trong số đó là phương pháp MFCC và LPC.



Hình: Công đoạn rút trích đặc trưng

Hình trên mô tả quá trình của việc rút trích đặc trưng, tín hiệu âm thanh lưu trong máy tính là tín hiệu digital [9], mô hình hóa tín hiệu âm thanh trong máy tính dưới dạng toán học là một hàm $s(n)$, trong đó n chỉ thời gian (thông thường là ms) và $s(n)$ là biên độ âm.

2.5.2 Làm rõ tín hiệu (pre-emphasis - tiền khuếch đại)

Theo các nghiên cứu về âm học thì giọng nói có sự suy giảm 20dB/decade khi lên tần số cao do đặc điểm sinh lý của hệ thống phát âm con người. Để khắc phục sự suy giảm này, chúng ta sẽ phải tăng cường tín hiệu lên một giá trị gần 20dB/decade. Bên cạnh đó, hệ thống thính giác con người có xu hướng nhạy cảm hơn với vùng tần số cao. Dựa vào những đặc điểm trên, ta sẽ áp dụng bộ lọc thông cao để tiền xử lý các tín hiệu thu được nhằm làm rõ vùng tín hiệu mà tai người có thể nghe được. Bộ lọc áp dụng công thức sau:

$$H_{pre}(z) = 1 + a_{pre}z^{-1} \quad (0.1)$$

Trong đó a_{pre} là hệ số nhấn mạnh, thường có giá trị là 0.9700002861.

Bộ lọc này có chức năng tăng cường tín hiệu tại tần số cao (tần số trên 1KHz).

Tín hiệu tiếng nói đã số hoá, $s(n)$, được đưa qua một hệ số bậc thấp, để làm phẳng tín hiệu về phô và làm nó ít bị ảnh hưởng bởi các hiệu ứng có độ chính xác hữu hạn sau này trong quá trình xử lý tín hiệu. Hệ thống số được dùng trong khối tiền khuếch đại vừa cố định vừa thích nghi chậm (ví dụ, để lấy trung bình các điều kiện chuyển, các nền nhiễu, hoặc thậm chí lấy trung bình phô tín hiệu).

Trong trường hợp sử dụng bộ lọc áp theo công thức 0.1, đầu ra của dãy tiền khuếch đại $s'(n)$, liên quan đến đầu vào của dãy tín hiệu $s(n)$, theo đẳng thức vi phân sau:

$$s'(n) = s(n) - a_{pre}s(n-1) \quad (0.2)$$

2.5.3 Tách từ

Tín hiệu tiếng nói $s(n)$ sau khi được làm nỗi tín hiệu, sẽ được chuyển sang để tách từ, tách từ là công đoạn chia toàn bộ tín hiệu thu được thành những đoạn tín hiệu mà trong đó chỉ chứa nội dung của một từ

Có nhiều phương pháp để tách điểm đầu và điểm cuối của một ra khỏi toán bộ tín hiệu tiếng nói, trong đó phương pháp dùng hàm năng lượng thời gian ngắn là phương pháp được sử dụng phổ biến. Với một cửa sổ kết thúc tại mẫu thứ m , hàm năng lượng thời gian ngắn $E(m)$ được xác định:

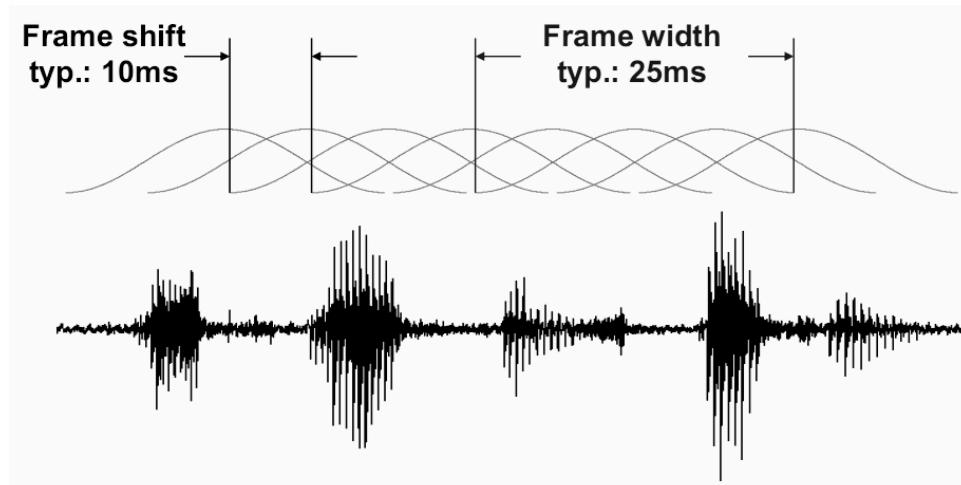
$$E_m = \sum_{n=m}^{m+N-1} [x(n) * w(n-m)]^2 \quad (0.3)$$

2.5.3.1 Phân đoạn thành các khung (Framing)

Tín hiệu tiếng nói là một đại lượng biến thiên theo thời gian và không ổn định nên không thể xử lý trực tiếp trên đó được. Do đó, tín hiệu được chia ra thành các khung với chiều dài tương đối nhỏ để lấy được các đoạn tín hiệu tương đối ổn định và xử lý tiếp trong các bộ lọc tiếp theo. Theo các nghiên cứu đã có thì trong khoang thời gian 10-20ms, tín hiệu tiếng nói tương đối ổn định. Nên ở bước này, người ta thường phân tín hiệu thành các khung với kích

thước 20-30ms. Nhưng để tránh mất mát và làm gián đoạn tín hiệu ban đầu, khi phân khung, người ta chồng lấp các khung lên nhau khoảng 10-15ms.

Trong bước này tín hiệu được tiền khuếch đại, $\tilde{s}^{(n)}$, được chia khói thành các khung N mẫu, với các khung kề nhau được ngăn cách bởi M mẫu.



Hình 0.2 Ví dụ phân khung đoạn tín hiệu

Khung đầu tiên chứa N mẫu tiếng nói đầu tiên. Khung thứ hai bắt đầu sau khung thứ nhất M mẫu, và chồng lên nó N - M mẫu. Thông thường, khung thứ 3 bắt đầu sau 2M so với khung đầu tiên (hoặc M mẫu sau khung thứ 2) và chồng lên khung đầu N - 2M mẫu. Quá trình này tiếp tục cho đến khi toàn bộ tiếng nói được tính hết cho một hay nhiều khung. Để thấy là nếu $M \leq N$ thì các khung chồng lên nhau và ước đoán phô LPC kết quả sẽ là tương quan từ khung này đến khung khác; nếu $M << N$ thì các ước đoán phô LPC từ khung này đến khung khác sẽ khá trôi chảy. Nói cách khác, nếu $M > N$, sẽ không có chồng lấp giữa các khung kề nhau; thực tế, một số tín hiệu tiếng nói sẽ hoàn toàn bị mất (tức là không bao giờ xuất hiện trong bất cứ khung phân tích nào), và mối tương quan giữa các ước đoán phô LPC của các khung kề nhau sẽ không chứa một thành phần nhiều mà cường độ của nó tăng như M (nghĩa là, khi có nhiều tiếng nói bị bỏ qua không phân tích). Tình trạng này là không thể chấp nhận trong phân tích LPC cho nhận dạng tiếng nói. Nếu ta biểu thị khung tiếng nói thứ ℓ là $x_{\ell}(n)$ và có L khung trong toàn bộ tín hiệu tiếng nói thì

$$x_\ell(n) = \tilde{s}(M\ell + n) \quad n = 0, 1, \dots, N-1 \quad \ell = 0, 1, \dots, L-1 \quad (0.4)$$

Tức là, khung tiếng nói đầu tiên, $x_0(n)$, chứa các mẫu tiếng nói $\tilde{s}(0), \tilde{s}(1), \dots, \tilde{s}(N-1)$, khung tiếng nói thứ hai, $x_1(n)$, chứa các mẫu $\tilde{s}(M), \tilde{s}(M+1), \dots, \tilde{s}(M+N-1)$, và khung thứ L, $x_{L-1}(n)$, chứa các mẫu $\tilde{s}(M(L-1)), \tilde{s}(M(L-1)+1), \dots, \tilde{s}(M(L-1)+N-1)$.

2.5.4 Lấy cửa sổ khung tín hiệu

Để làm rõ tín hiệu và đồng thời giảm tính gián đoạn tín hiệu ở đầu và cuối của mỗi khung trong quá trình xử lý rút trích đặc trưng, khi xử lý, các khung sẽ được nhân với hàm cửa sổ, thường là cửa sổ Hamming. Kết quả của việc này là làm cho khung tín hiệu mượt hơn, giúp cho các thành phần có tần số cao xuất hiện trong phô. Công thức hàm cửa sổ tổng quát:

$$w(n) = \begin{cases} \alpha + (1 - \alpha) \cdot \cos\left(\frac{2\pi n}{N}\right) & |n| \leq N/2 \\ 0 & |n| > N/2 \end{cases} \quad (0.5)$$

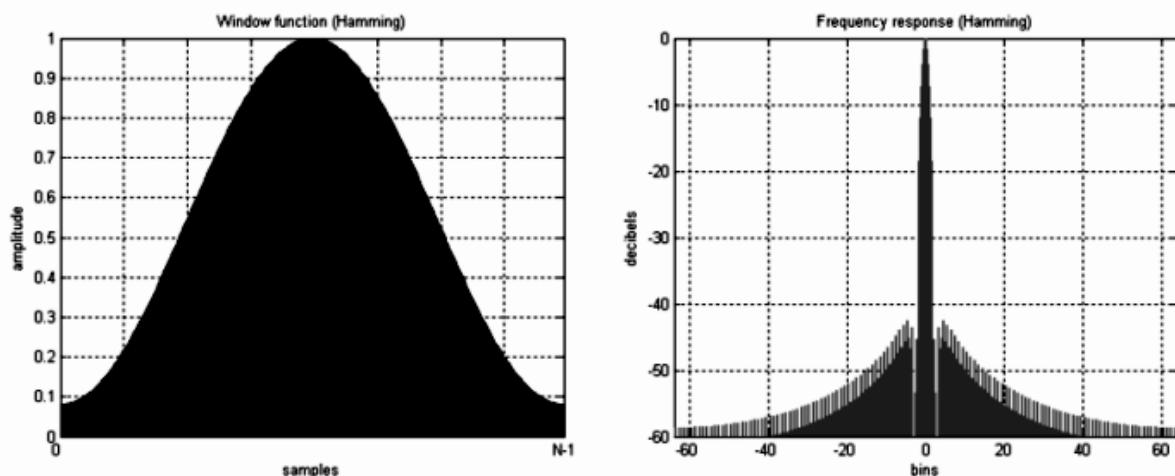
Trong đó $w(n)$ gọi là hàm cửa sổ. Tùy vào giá trị của α mà có các cửa sổ sau:

Với $\alpha=0.54$, ta có cửa sổ Hamming (Hamming Window):

$$w(n) = \begin{cases} 0.54 + 0.46 \cdot \cos\left(\frac{2\pi n}{N}\right) & |n| \leq N/2 \\ 0 & |n| > N/2 \end{cases} \quad (0.6)$$

Với định nghĩa cửa sổ $w(n)$ theo công thức trên, $0 \leq n \leq N-1$, thì kết quả chia cửa sổ cho khung $x(n)$:

$$\tilde{x}_\ell(n) = x_\ell(n)w(n) \quad 0 \leq n \leq N-1 \quad (0.7)$$



Hình 0.3 Cửa sổ Hamming và tín hiệu sau khi nhân với hàm cửa sổ Hamming

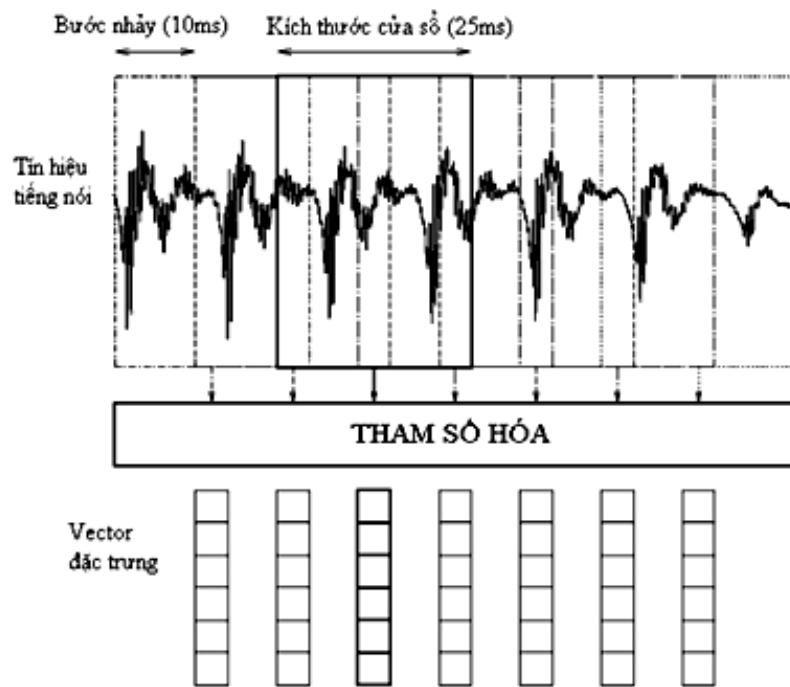
Ý nghĩa của việc áp cửa sổ : là nhằm mục đích có được dữ liệu theo miền tần số chuẩn để đưa vào phép biến đổi Fourier rời rạc.

2.5.5 Rút trích đặc trưng

2.5.5.1 Phương pháp rút trích đặc trưng MFCC

Trong lĩnh vực xử lý và nhận dạng tiếng nói, việc tiền xử lý các tín hiệu thu được và rút trích đặc trưng là một kỹ thuật thiết yếu mà bất cứ hệ thống nhận dạng nào cũng bắt buộc phải có. Trích rút đặc trưng có vai trò quan trọng quyết định hiệu suất của quá trình nhận dạng mẫu(cả trong quá trình nhận dạng và trong quá trình huấn luyện). Công việc của bước này là phân tích phổ spectral nhằm mục đích xác định các thông tin quan trọng, đặc trưng của tiếng nói, cắt giảm bớt các yếu tố không cần thiết trong quá trình nhận dạng để làm giảm khối lượng dữ liệu cần xử lý.

Mel Scale Frequency Cepstral Coefficients (MFCC) là một phương pháp rút trích đặc trưng sử dụng dãy bộ lọc được Davis và Mermelstein đưa ra vào năm 1980 khi họ kết hợp các bộ lọc cách khoảng không đều với phép biến đổi Cosine rời rạc (Discrete Cosine Transform) thành một thuật toán hoàn chỉnh ứng dụng trong lĩnh vực nhận dạng tiếng nói liên tục. Động thời định nghĩa khái niệm hệ số Cepstral và thang đo tần số Mel (Mel scale).



Hình 0.4 Tổng quát phương pháp rút trích đặc trưng MFCC

Tóm tắt quá trình rút trích đặc trưng theo MFCC sẽ như sau: Ban đầu tín hiệu sau khi qua tiền xử lý sẽ được chia thành các Frame có khoảng thời gian ngắn. Từ mỗi frame đó sau khi áp dụng các bước chuyển đổi và lọc sẽ ra được một vecto tương ứng. Và xong quá trình này, ta sẽ có đặc trưng của dãy tín hiệu input vào là một dãy vecto đặc trưng output ra.

a. Biến đổi FFT (Fast Fourier Transform)

Biến đổi FFT thực chất là một biến đổi DFT (Discrete Fourier Transform) nhưng được tối ưu bằng các thuật toán nhanh và gọn hơn để đáp ứng các yêu cầu xử lý theo thời gian thực trong các lĩnh vực như xử lý âm thanh, hình ảnh,...

Fast Fourier là một phép biến đổi thuận nghịch có đặc điểm bảo toàn tính tuyến tính bất biến, tuần hoàn và tính trễ. Dùng để biến đổi tín hiệu tương tự sang miền tần số, nó gồm các công thức như sau:

Công thức phép biến đổi thuận (dùng để phân tích tín hiệu):

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \quad k=0,1,2,\dots, N-1 \quad (0.8)$$

Công thức phép biến đổi nghịch (dùng để tổng hợp lại tín hiệu):

$$x(n) = \sum_{k=0}^{N-1} X(k) e^{j2\pi kn/N} \quad n=0,1,2,\dots, N-1 \quad (0.9)$$

Trong đó: $x(n)=a(n)+b(n)\sqrt{-1}$

Kết quả chúng ta có được khi thực hiện FFT là dãy tín hiệu $X_t(k)$ để đưa vào bộ lọc Mel-scale

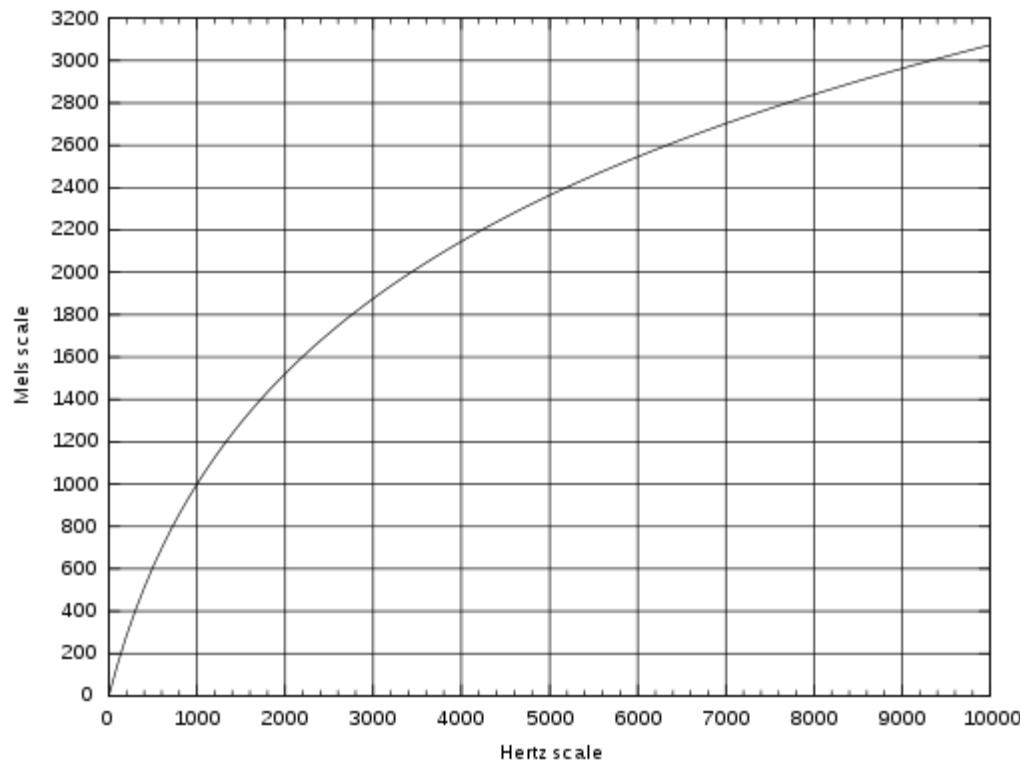
b. Lọc qua bộ lọc Mel-scale

Trong lĩnh vực nghiên cứu về nhận dạng tiếng nói, đòi hỏi chúng ta phải hiểu và mô phỏng chính xác khả năng cảm thụ tần số âm thanh của tai người. Chính vì thế các nhà nghiên cứu đã xây dựng một thang tần số - hay gọi là thang tần số Mel (Mel scale) dựa trên cơ sở thực nghiệm nhiều lần khả năng cảm nhận âm thanh của con người. Thang tần số Mel được định nghĩa trên tần số thực theo công thức:

$$m = 2595 \log_{10} \left(1 + \frac{f}{700} \right) \quad (0.10)$$

Trong đó: m là tần số trong thang Mel, đơn vị là Mel

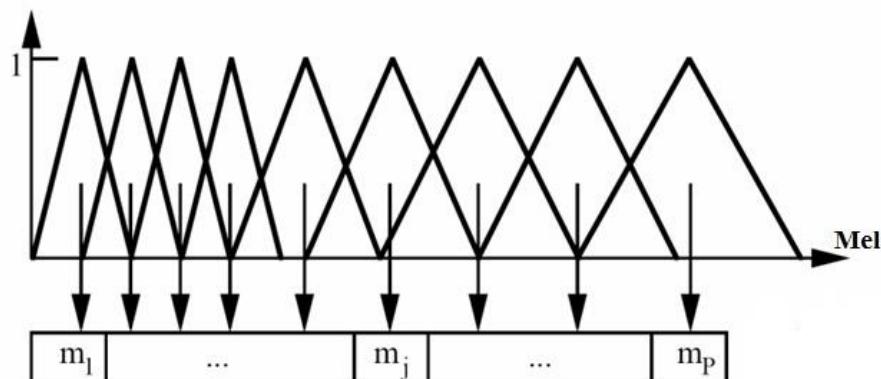
f là tần số thực, đơn vị là Hz.



Hình: Biểu đồ thang tần số Mel theo tần số thực.

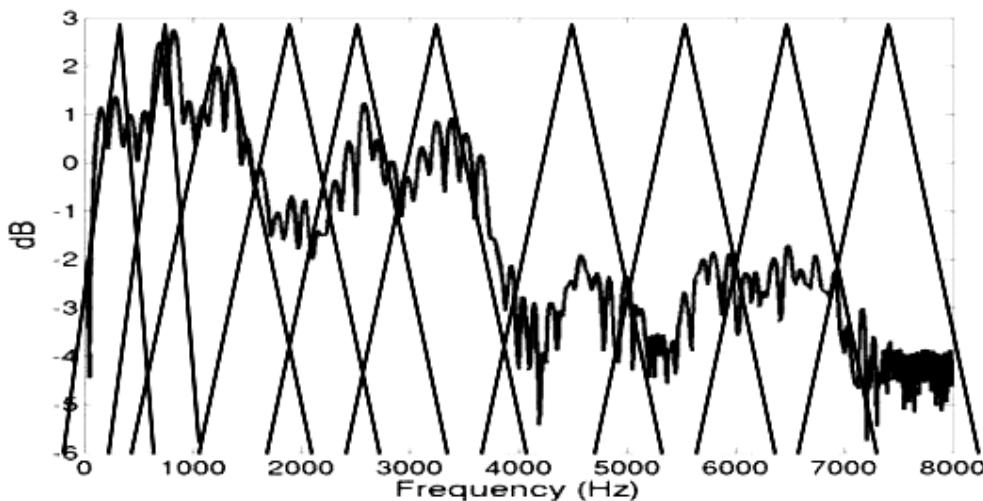
Theo biểu đồ trên thì trong khoảng tần số thấp hơn 1kHz thì đồ thị trên gần như là tuyến tính, nghĩa là trong khoảng tần số dưới 1kHz, tần số Mel và tần số thực. Trong khoảng tần số trên 1kHz thì mối quan hệ này là quan hệ Logarit.

Dựa vào các thực nghiệm trên tai người, người ta đã xác định được các tần số thực mà tai người có thể nghe được và chứa đựng nhiều thông tin. Sau đó chuyển các tần số này sang tần số Mel và xây dựng một thang đo như sau:



Hình: Băng lọc tần số Mel.

Ta dùng thang đo này để áp vào dãy sóng tín hiệu thu được sau khi thực hiện FFT



Hình: Dựa tín hiệu vào băng lọc tần số Mel.

Kết quả của bước này là chúng ta sẽ có được tập hợp các tần số $\mathbf{Y}_t(\mathbf{m})$ là giao điểm của sóng tần số với thang tần số Mel từ dãy tín hiệu $\mathbf{X}_t(\mathbf{k})$

c. Logarit giá trị năng lượng (Logarithm of filter energies)

Mục đích của bước này là nén các giá trị đã lọc được vào miền giá trị nhỏ hơn để xử lý nhanh hơn. Nên các giá trị thu được ở mỗi kênh lọc sẽ được lấy Logarit.

$$\log\{|Y_t(m)|^2\}$$

d. Biến đổi cosin rời rạc

Dựa vào phổ tín hiệu tiếng nói của con người trên miền tần số, ta có thể thấy rằng phổ tín hiệu khá trơn, nên khi lấy các giá trị năng lượng ra từ các bộ lọc, các giá trị này có sự tương quan khá gần nhau, dẫn đến các đặc trưng ta rút được sẽ không rõ ràng. Chính vì thế, ta thực hiện biến đổi DCT (Discrete Cosine Transform) để làm rời rạc các giá trị này ra cho nó ít tương quan với nhau, làm tăng tính đặc trưng của các tham số. Giá trị thu được sau bước này ta gọi là hệ số Cepstral.

$$c_i = \sum_{j=1}^N m_j \cos\left(\frac{\pi i}{N}(j - 0.5)\right) \quad (0.11)$$

Trong đó:

- N là số kênh lọc.
- M_j là giá trị logarit năng lượng của mạch lọc thứ j .
- i là bậc của hệ số cepstral.

Thông thường người ta lấy i trong đoạn [1,12] là số lượng đặc trưng trong mỗi vecto đặc trưng. Trong các hệ nhận dạng, số lượng đặc trưng nằm trong khoảng (10,15) là đủ để cho kết quả nhận dạng tương đối mà dữ liệu xử lý lại không quá lớn.

Sau khi thực hiện biến đổi DCT, theo công thức trên ta thấy các hệ số thu được sẽ tăng tuyến tính theo số bậc của nó. Hệ số Cepstral có số bậc cao sẽ có giá trị rất cao, ngược lại các hệ số với số bậc thấp sẽ có giá trị rất thấp. Sự chênh lệch này sẽ gây khó khăn cho chúng ta trong qua trình mô hình hóa dữ liệu và xử lý sau này. Vì khi có sự chênh lệch cao, ta phải dùng miền giá trị lớn để biểu diễn dữ liệu, và gặp khó khăn khi đưa vào các mô hình xử lý xác suất... Nên để có các hệ số tối ưu cho các qua trình sau, ta sẽ thực hiện việc điều chỉnh các hệ số này để giảm sự chênh lệch. Việc này thực hiện bằng công thức:

$$c'_n = \exp(n * k) \cdot c_n \quad (0.12)$$

Cuối cùng chúng ta sẽ thu được các giá hệ số Cepstral đã được tinh chế. Các hệ số này là đặc trưng MFCC mà chúng ta sẽ sử dụng để huấn luyện và nhận dạng.

2.5.5.2 Phương pháp trích đặc trưng LPC

e. Giới thiệu

LPC là chữ viết tắt của cụm từ: Linear Predictive Coding (mã hóa dự báo tuyến tính). Đây được xem là một trong những phương pháp được sử dụng rộng rãi trong việc rút trích đặc trưng của tín hiệu âm thanh (hay còn được gọi là tham số hóa tín hiệu âm thanh). Đóng vai trò quan trọng trong các kỹ

thuật phân tích tiếng nói. Đây còn được xem là một phương pháp hiệu quả cho việc nén (mã hóa với chất lượng tốt) dữ liệu tiếng nói ở mức bit rate thấp.

f. Cơ sở âm học của phương pháp LPC

Tiếng nói hay còn được xem là âm thanh do con người phát ra từ miệng bắt nguồn từ từ sự rung động của dây thanh âm (do sự thay đổi áp suất không khí từ phổi đưa lên), sự rung động này mang 2 đặc tính là cường độ (intensity) và tần số (frequency). Âm thanh này sau đó được truyền qua cuống họng đến khoang miệng và khoan mũi. Tại đây dựa vào cấu tạo vòng miệng khi nói, cách đặt lưỡi, chuyển động của lưỡi và cơ miệng... sẽ góp phần gây ra sự cộng hưởng của âm thanh (hay còn được gọi là các **Formant**), kết quả chính là tiếng nói mà ta nghe được. LPC phân tích những tín hiệu tiếng nói bằng cách ước tính các formant, loại bỏ đi những thành phần ảnh hưởng của nó (những thứ không mang giá trị tiếng nói trong âm phát ra), và ước lượng các đặc điểm về cường độ, tần số của phần âm thanh còn lại. Quá trình loại bỏ ở trên còn được gọi là quá trình lọc nghịch đảo (*inverse filtering*) và phần âm thanh còn lại gọi là “*căn*”(*residue*) mang những yếu tố và đặc trưng cốt lõi của âm thanh. Kết quả còn lại sau quá trình LPC là những con số, mà mô tả những đặc điểm quan trọng của các formant cũng như phần âm thanh còn lại. Các con số này có thể được dùng đại diện như tín hiệu tiếng nói ban đầu, hiệu quả hơn trong việc lưu trữ, phân tích nội dung, truyền tải tiếng nói... LPC còn được dùng trong quá trình tổng hợp lại tiếng nói từ các con số đặc trưng trên.

g. Nội dung phương pháp rút trích đặc trưng LPC

Ý tưởng của LPC là một mẫu tiếng nói cho trước ở thời điểm n , $s(n)$, có thể được xấp xỉ bằng một tổ hợp tuyến tính của p mẫu tiếng nói trước đó:

$$s(n) \approx a_1 s(n-1) + a_2 s(n-2) + \dots + a_p s(n-p) \quad (0.13)$$

Trong đó a_1, a_2, \dots, a_n coi là các hằng trên toàn khung phân tích tiếng nói. Ta chuyển đổi thức (1) trên tương đương bằng cách thêm giới hạn kích thích, $Gu(n)$, có:

$$s(n) = \sum_{i=1}^p a_i s(n-i) + Gu(n) \quad (0.14)$$

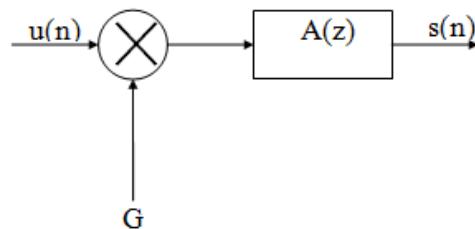
Trong đó $u(n)$ là kích thích đã chuẩn hoá, G là độ khuếch đại của kích thích. Biểu diễn đẳng thức (2) trong miền z ta có quan hệ:

$$S(z) = \sum_{i=1}^p a_i z^{-i} S(z) + GU(z) \quad (0.15)$$

dẫn đến hàm chuyen:

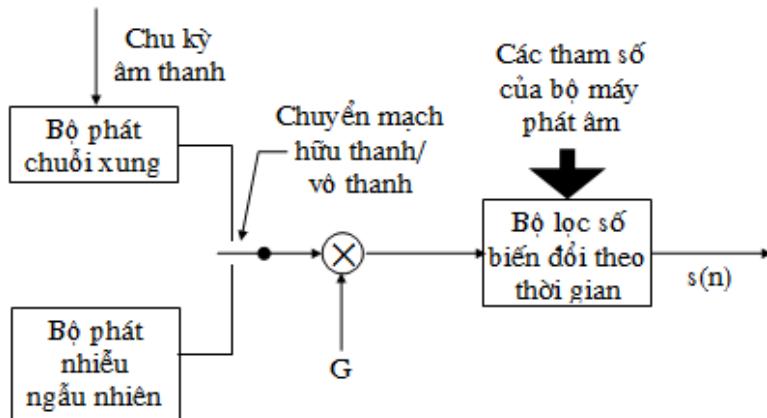
$$H(z) = \frac{S(z)}{GU(z)} = \frac{1}{1 - \sum_{i=1}^p a_i z^{-i}} = \frac{1}{A(z)} \quad (0.16)$$

Hình 0.5 diễn tả đẳng thức bên trên, mô tả nguồn kích thích đã chuẩn hóa $u(n)$, sau đó được nâng mức với độ khuếch đại G , và đóng vai trò đầu vào của hệ toàn cực, $H(z) = \frac{1}{A(z)}$, tạo ra tín hiệu tiếng nói $s(n)$.



Hình 0.5 Mẫu tiếng nói dự báo tuyến tính

Chức năng kích thích thực tế đối với tiếng nói về bản chất là chuỗi liên tục các xung gần như điều hoà (với các âm hữu thanh) và là một nguồn nhiễu ngẫu nhiên (với các âm vô thanh), mẫu hoà âm thích hợp đối với tiếng nói, ứng với phân tích LPC được mô tả trong phía dưới. Tại đây, nguồn kích thích đã chuẩn hoá được chọn nhờ một chuyển mạch mà vị trí của nó được điều khiển bởi đặc tính hữu thanh/vô thanh của tiếng nói, chọn cả chuỗi xung gần điều hoà làm kích thích đối với các âm hữu thanh, chọn cả chuỗi nhiễu ngẫu nhiên cho các âm vô thanh. Độ khuếch đại G của nguồn được ước tính từ tín hiệu tiếng nói, và nguồn được nâng mức được dùng làm đầu vào của bộ lọc số ($H(z)$), điều khiển bởi đặc tính của các tham số dài phát âm của tiếng nói đang được tạo. Như vậy các tham số của mẫu này là phân loại hữu thanh/vô thanh, chu kỳ cao độ của âm hữu thanh, tham số khuếch đại, và các hệ số của bộ lọc số, $\{a_k\}$. Các tham số này đều biến đổi rất chậm theo thời gian.



Hình 0.6. Mẫu phân tích tiếng nói theo phương pháp LPC

Dựa vào mẫu trên hình 1, quan hệ chính xác giữa $s(n)$ và $u(n)$ là

$$s(n) = \sum_{k=1}^p a_k s(n-k) + Gu(n) \quad (0.17)$$

Ta coi $\bar{s}(n)$ là ước lượng tuyến tính của các mẫu tiếng nói trước đó, được định nghĩa là:

$$\bar{s}(n) = \sum_{k=1}^p a_k (n-k) \quad (0.18)$$

Bây giờ ta thiết lập lỗi dự báo, $e(n)$, được định nghĩa là:

$$e(n) = s(n) - \bar{s}(n) = s(n) - \sum_{k=1}^p a_k (n-k) \quad (0.19)$$

với hàm chuyển đổi lỗi:

$$A(z) = \frac{E(z)}{S(z)} = 1 - \sum_{k=1}^p a_k z^{-k} \quad (0.20)$$

Rõ ràng là khi $s(n)$ được tạo thực sự bởi một hệ tuyến tính kiểu như hình 13 thì lỗi dự báo $e(n)$ sẽ bằng $Gu(n)$, kích thích được khuếch đại.

Vấn đề cơ bản của phân tích dự báo tuyến tính là xác định tập các hệ số dự báo, $\{a_k\}$, trực tiếp từ tín hiệu tiếng nói. Vì các đặc tính phổ của tiếng nói biến đổi theo thời gian nên các hệ số dự báo tại thời điểm cho trước n , phải được ước lượng từ một đoạn tín hiệu tiếng nói ngắn xuất hiện quanh thời điểm n .

Như vậy cách cơ bản là tìm một tập các hệ số dự báo giảm thiểu lỗi dự báo trung bình bậc hai trong một đoạn dạng sóng tiếng nói. (Thường thì kiểu phân tích phổ thời gian ngắn này được thực hiện trên các khung tiếng nói liên tiếp, có cách khoảng 10ms).

Nội dung tiếp theo trình bày các bước chính trong công đoạn phân tích đặc trưng theo phương pháp LPC.

h. Phân tích tự tương quan

Sau khi tiến hành công đoạn nhận với hàm cửa sổ (áp dụng công thức 0.7). Mỗi khung của tín hiệu được chia cửa sổ là tự tương quan với khung tiếp theo để cho:

$$r_{\ell}(m) = \sum_{n=0}^{N-1-m} \tilde{x}_{\ell}(n) \tilde{x}_{\ell}(n+m) \quad m = 0, 1, \dots, p \quad (0.21)$$

Trong đó giá trị tự tương quan cao nhất, p , là bậc của phép phân tích LPC. Các giá trị p thường dùng là từ 8 đến 16, với $p=8$ được sử dụng nhiều trong các hệ phân tích LPC.

i. Phân tích LPC

Bước xử lý tiếp theo là phân tích LPC, chuyển từng khung của $p+1$ mối tự tương quan thành một "tập tham số LPC", đó là các hệ số LPC. Phương pháp chính thức để chuyển từ các hệ số tự tương quan sang tập tham số LPC (để dùng cho phương pháp tự tương quan LPC) được gọi là phương pháp Durbin và có thể cho một cách hình thức như thuật toán (để thuận tiện, ta sẽ bỏ qua ℓ nhỏ ở dưới $r_{\ell}(m)$):

$$E^{(0)} = r(0) \quad (0.22)$$

$$k_i = \left\{ r(i) - \sum_{j=1}^{L-1} \alpha_j^{(i-1)} r(|i-j|) \right\} / E^{(i-1)}, \quad 1 \leq i \leq p \quad (0.23)$$

$$\alpha_i^{(i)} = k_i \quad (0.24)$$

$$\alpha_j^{(i)} = \alpha_j^{(i-1)} - k_j \alpha_{i-j}^{(i-1)} \quad (0.25)$$

$$E^{(i)} = (1 - k_i^2) E^{(i-1)} \quad (0.26)$$

Trong đó tổng trong đẳng thức (0.23) được bỏ qua đối với $i=1$. Tập các đẳng thức (0.22 - 0.26) được giải đê qui với $i=1,2,\dots,p$, và lời giải cuối cùng được cho là

$$a_m = \text{các hệ số LPC} = \alpha_m^{(p)}, \quad 1 \leq m \leq p \quad (0.27)$$

$$k_m = \text{các hệ số PARCOR} \quad (0.28)$$

$$g_m = \text{các hệ số truyền miền logarit} = \log\left(\frac{1-k_m}{1+k_m}\right) \quad (0.29)$$

Lúc này ta có thể dùng các hệ số LPC làm vector đặc trưng cho từng khung. Tuy nhiên có một phép biến đổi tạo ra dạng hệ số khác có độ tập trung cao hơn từ các hệ số LPC, đó là phép phân tích Cepstral.

j. Phân tích cepstral

Một tập tham số LPC rất quan trọng, có thể suy trực tiếp từ tập hệ số LPC, là các hệ số Cepstral LPC, $c(m)$. Dạng đê qui là:

$$c_m = a_m + \sum_{k=1}^{m-1} \left(\frac{k}{m} \right) c_k a_{m-k}, \quad 1 \leq m \leq p \quad (0.30)$$

$$c_m = \sum_{k=1}^{m-1} \left(\frac{k}{m} \right) c_k a_{m-k}, \quad Q > m > p \quad (0.31)$$

Trong đó σ^2 là độ khuếch đại trong mẫu LPC. Các hệ số cepstral, là các hệ số của biểu diễn chuyển đổi Fourier của logarit phổ cường độ, được mô tả là một tập đặc tính mạnh, đáng tin cậy hơn các hệ số LPC. trong đó $Q \approx \left(\frac{3}{2}\right)p$.

k. Đặt trọng số cho các hệ số cepstral

Do tính nhạy cảm của các hệ số cepstral bậc thấp đối với sườn phổ tổng thể và do tính nhạy cảm của hệ số cepstral bậc cao đối với nhiễu (và các dạng biến đổi giống nhiễu khác), nó trở thành kỹ thuật chuẩn để định trọng các hệ số cepstral nhờ một cửa sổ được làm hẹp sao cho giảm thiểu được những nhạy cảm này. Chuyển sang cepral có trọng số:

$$c'm = w_m.c_m \quad \text{với } 1 \leq m \leq Q \quad (0.32)$$

Hàm trọng số thích hợp là bộ lọc thông dài (trong miền cepstral):

$$w_m = \left[1 + \frac{Q}{2} \sin\left(\frac{\pi m}{Q}\right) \right] \text{ với } 1 \leq m \leq Q \quad (0.33)$$

I. Tính đạo hàm cepstral

$$\frac{dc_m(t)}{dt} = \Delta c_m(t) \approx \mu \sum_{k=-K}^K k c_m(t+k) \quad (0.34)$$

Với μ là hằng số chuẩn và $(2K+1)$ là số lượng Frame cần tính. $K=3$ là giá trị thích hợp để tính đạo hàm cấp. Vector đặc trưng của tín hiệu gồm Q hệ số cepstral và Q hệ số đạo hàm cepstral.

m. Nhận xét phương pháp LPC

Tóm lại, trong mô hình phân tích LPC trên, chúc ta cần phải đặc tả các tham số bao gồm:

- N: số mẫu trong mỗi Frame phân tích
- M: số mẫu cách nhau giữa 2 Frame kề nhau
- p: cấp phân tích LPC
- Q: số chiều của các vector cepstral dẫn xuất từ các hệ số LPC
- K: số Frame được dùng tính các đạo hàm cepstral.

Tuy mỗi tham số đều có thể thay đổi trên dải rộng, bảng sau cho các giá trị đặc trưng đối với các hệ phân tích ở 3 tần số lấy mẫu khác nhau (6.67kHz, 8kHz và 10kHz).

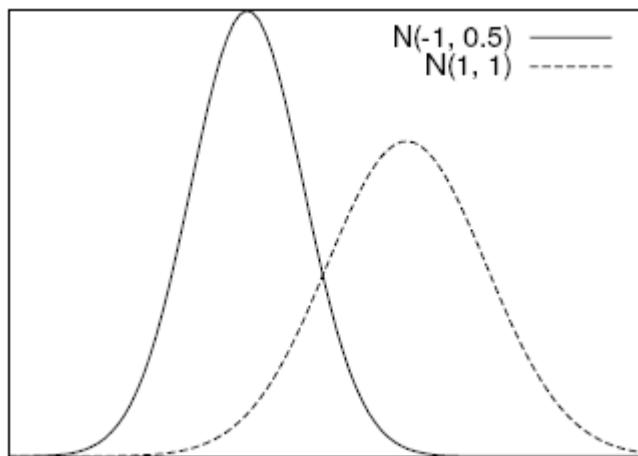
Bảng 3 Các giá trị đặc trưng cho các tham số phân tích LPC đối với hệ nhận dạng tiếng nói

Tham số	F1=6.67kHz	F1=8kHz	F1=10kHz
N	300 (45ms)	240 (30ms)	300 (30ms)
M	100 (15ms)	80 (10ms)	100 (10ms)
P	8	10	10
Q	12	12	12
K	3	3	3

Mô hình LPC là mô hình thích hợp cho việc xử lý tín hiệu tiếng nói. Với miền tiếng nói hữu thanh có trạng thái gần ổn định, mô hình tất cả các điểm cực đại của LPC chia một xấp xỉ tốt đối với đường bao phổ cơ quan phát âm. Với miền tiếng nói vô thanh, mô hình LPC tỏ ra ít hữu hiệu hơn so với miền hữu thanh, nhưng nó vẫn là mô hình hữu ích cho các mục đích nhận dạng tiếng nói. Mô hình LPC đơn giản và dễ cài đặt trên phần cứng và phần mềm.

2.6 Gaussian mixture model

Mô hình hợp Gauss (**Gaussian Mixture Model - GMM**) là một dạng mô hình thống kê được xây dựng từ việc huấn luyện các tham số thông qua dữ liệu học. Mô hình GMM còn có một số tên gọi khác như *Weighted Normal Distribution Sums* hay *Radial Basis Function Approximations*...



Hình 0.7: Hàm mật độ Gauss

Về cơ bản, mô hình GMM xấp xỉ một hàm mật độ xác suất bằng hợp các hàm mật độ Gauss. Hình 0.7 minh họa hai hàm mật độ Gauss với các tham số khác nhau. Một cách hình thức, hàm mật độ xác suất của phân phối Gauss $f_N(x, \mu, \sigma^2)$ được cho bởi công thức:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad (0.35)$$

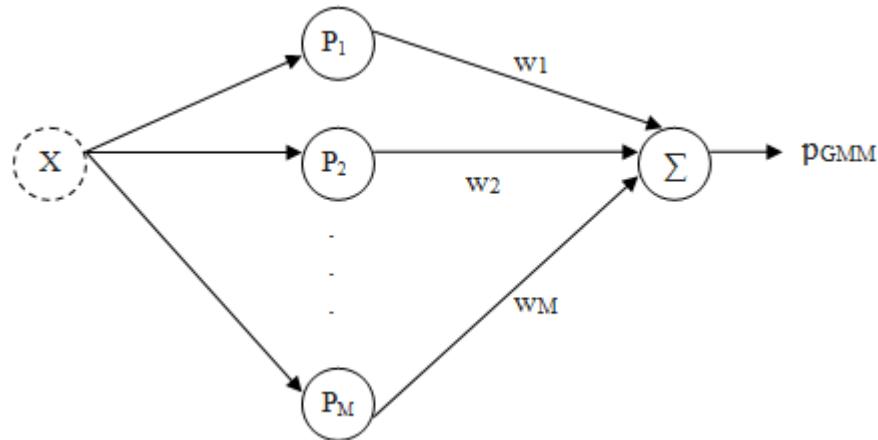
trong đó, μ là giá trị trung bình, σ là độ lệch chuẩn. Trong trường hợp x là vector gồm D thành phần, hàm mật độ xác suất của phân phối Gauss $f_N(x, \mu, \Sigma)$ được cho bởi công thức:

$$p(\vec{x}) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (\vec{x} - \vec{\mu})' \Sigma^{-1} (\vec{x} - \vec{\mu})\right) \quad (0.36)$$

khi đó, μ là vector trung bình, Σ là ma trận hiệp phương sai. Nếu chọn $\mu=0$ và $\sigma=1$, công thức (0.35) sẽ trở thành hàm mật độ chuẩn Gauss tiêu chuẩn:

$$p(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \quad (0.37)$$

Từ “Gauss” được đặt theo tên của nhà toán học người Đức Carl Friedrich Gauss. Ông đã định nghĩa hàm mật độ Gauss và áp dụng trong phân tích dữ liệu thiên văn.



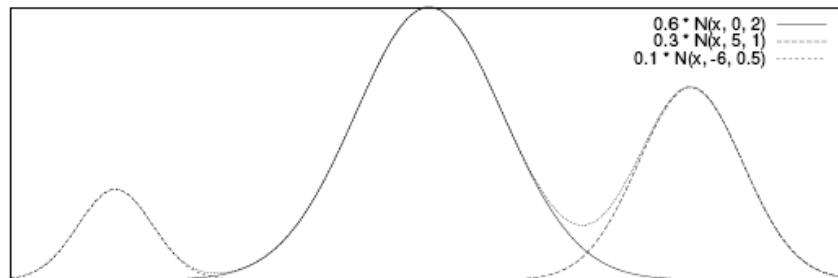
Hình 0.8: Mô hình GMM

Cho trước M phân phối Gauss có các hàm mật độ p_1, p_2, \dots, p_M , hàm mật độ xác suất của mô hình GMM được minh họa trong Hình 0.8 chính là tổng trọng của M phân phối Gauss theo công thức:

$$p_{GMM}(\vec{x}) = \sum_{i=1}^M w_i p_i(\vec{x}) \quad (0.38)$$

trong đó, w_i là trọng số của phân phối Gauss thứ i , thỏa ràng buộc $0 \leq w_i \leq 1$ và $\sum_{i=1}^M w_i = 1$. Các trọng số này thể hiện mức độ ảnh hưởng của mỗi phân phối Gauss đối với mô hình GMM. Như vậy, phân phối Gauss có phương sai và trọng số lớn bao nhiêu thì có mức độ ảnh hưởng lớn bấy nhiêu đối với kết

xuất của mô hình. Hình dưới cho thấy mức độ ảnh hưởng của từng phân phối Gauss lên GMM.



Hình 0.9:Hàm mật độ của GMM có 3 phân phối Gauss

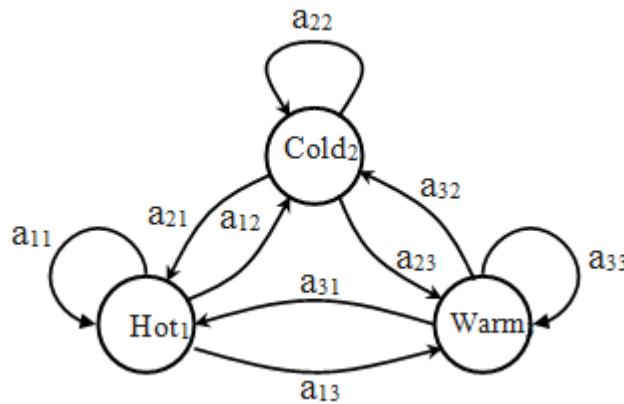
Như vậy, một mô hình GMM có M phân phối Gauss sẽ được đại diện bởi bộ tham số $\lambda = \{ w_i, \mu_i, \Sigma_i \}, i \in [1, M]$. Tùy thuộc vào cách tổ chức của ma trận hiệp phương sai (covariance matrix), GMM có thể có một số biến thể khác nhau:

- Nodal covariance matrices GMM: mỗi phân phối Gauss trong GMM có một ma trận hiệp phương sai riêng.
- Grand covariance matrix GMM: mọi phân phối Gauss trong một GMM dùng chung một ma trận hiệp phương sai.
- Global covariance matrix GMM: mọi phân phối Gauss trong tất cả các GMM dùng chung một ma trận hiệp phương sai.

Ngoài ra, xét về dạng thức, ma trận hiệp phương sai gồm hai loại: full (dạng đầy đủ) và diagonal (dạng ma trận đường chéo). Thông thường, dạng nodal-diagonal covariance matrices GMM được sử dụng phổ biến nhất.

2.7 Hidden Markov Model

2.7.1 Giới thiệu chuỗi Makov

*Hình 0.10 Mô hình Markov chain thời tiết*

Một mô hình Markov chain được cụ thể hóa bằng các thành phần:

$S = \{S_1, S_2, \dots, S_N\}$ Tập các trạng thái, gọi q_t là trạng thái đạt đến được ở thời điểm t .

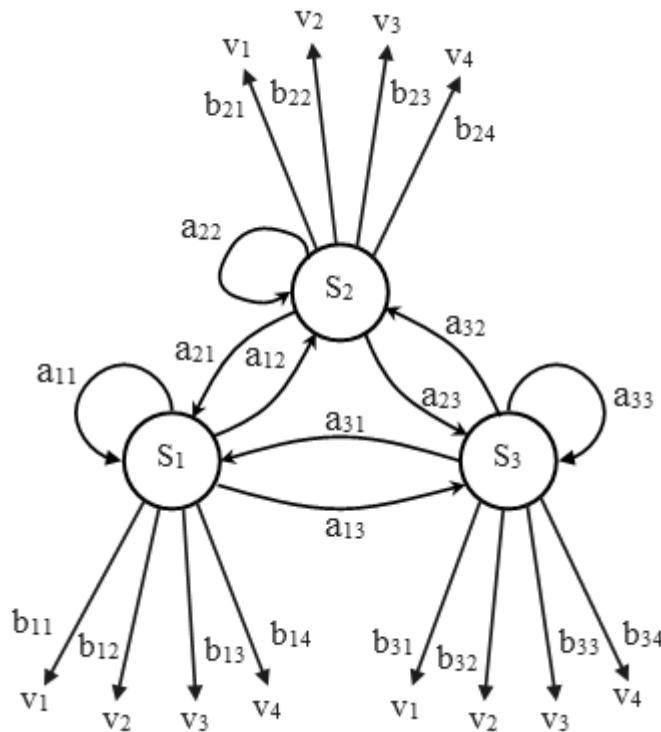
$A = \{a_{ij}\}$ Ma trận xác suất chuyển trạng thái, mỗi a_{ij} thể hiện xác suất di chuyển từ trạng thái S_i sang trạng thái S_j , $a_{ij} = p(q_{t+1} = S_j | q_t = S_i)$, $1 \leq i, j \leq N$, sao cho $a_{ij} \geq 0, \sum_{j=1}^N a_{ij} = 1$.

$\pi = \{\pi_i\}$ π_i là xác suất khởi đầu của trạng thái S_i – xác suất rơi vào trạng thái S_i ở thời điểm $t = 1$:

$$\pi_i = p(q_1 = S_i), 1 \leq i \leq N, \sum_{i=1}^N \pi_i = 1$$

2.7.2 Mô hình Markov ẩn

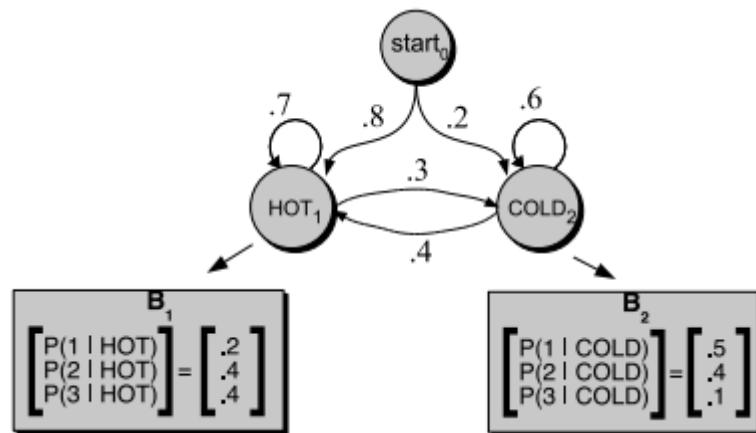
Trong mô hình Markov ẩn, các sự kiện quan sát được nằm trong mỗi trạng thái và phụ thuộc vào hàm mật độ xác suất trong các trạng thái đó.



Hình 0.11 Mô hình Markov ẩn 3 trạng thái

Hình 0.11 minh họa một mô hình Markov ẩn 3 trạng thái với các sự kiện có thể quan sát được trong mỗi trạng thái là $V = \{v_1, v_2, v_3, v_4\}$. Khả năng quan sát được sự kiện v_k trong trạng thái S_j phụ thuộc vào xác suất quan sát $b_j(k)$. Hàm b được gọi là hàm mật độ xác suất của các sự kiện được quan sát.

Hình 0.12 minh họa một ví dụ HMM đơn giản về mối liên hệ giữa số lượng que kem với thời tiết. Số lượng que kem được ăn mỗi ngày là các quan sát, ví dụ: $O = \{1, 2, 3\}$. Thời tiết nóng hay lạnh tương ứng với hai trạng thái. Giả sử chúng ta biết được số lượng que kem được ăn, không biết thời tiết, liệu chúng ta có dự đoán được thời tiết hôm ấy thế nào không. Thời tiết (trạng thái) được xem là “ẩn” so với số que kem được ăn (kết quả quan sát được). Chính vì vậy, mô hình này được gọi là mô hình Markov ẩn (hidden) – Hidden Markov Model (HMM).



Hình 0.12 Ví dụ HMM đơn giản

Một mô hình Markov ẩn được cụ thể hóa bằng các thành phần:

$S = \{S_1, S_2, \dots, S_N\}$ Tập các trạng thái, q_t là trạng thái đạt đến được ở thời điểm t .

$V = \{v_1, v_2, \dots, v_M\}$ Tập các tín hiệu quan sát, gọi O_t là tín hiệu quan sát được ở thời điểm t

$A = \{a_{ij}\}$ Ma trận xác suất chuyển trạng thái, mỗi a_{ij} thể hiện xác suất di chuyển từ trạng thái S_i sang trạng thái S_j , $a_{ij} = p(q_{t+1} = S_j | q_t = S_i)$, $1 \leq i, j \leq N$, sao cho $a_{ij} \geq 0$, $\sum_{j=1}^N a_{ij} = 1$.

$B = \{b_j(k)\}$ Ma trận các hàm mật độ xác suất trong mỗi trạng thái,

$$b_j(k) = p(v_k \text{ at } t | q_t = S_j), \quad 1 \leq j \leq N, \quad 1 \leq k \leq M$$

$$\text{Thỏa ràng buộc } \sum_{k=1}^M b_j(k) = 1$$

$\pi = \{\pi_i\}$ π_i là xác suất khởi đầu của trạng thái S_i – xác suất rơi vào trạng thái S_i ở thời điểm $t = 1$:

$$\pi_i = p(q_1 = S_i), \quad 1 \leq i \leq N, \quad \sum_{i=1}^N \pi_i = 1$$

Để thuận tiện cho việc trình bày, mỗi mô hình HMM sẽ được quy ước đại diện bởi bộ tham số $\lambda = (\pi, A, B)$.

2.7.3 Ba bài toán cơ bản của HMM

Để có thể áp dụng được mô hình HMM vào các ứng dụng phức tạp trong thực tế, trước hết cần có lời giải thỏa đáng cho 3 bài toán cơ bản của HMM:

- *Bài toán 1:* cho trước chuỗi tín hiệu quan sát $O = O_1 O_2 \dots O_T$ và mô hình HMM được đại diện bởi bộ tham số $\lambda = (\pi, A, B)$. Làm sao để tính toán một cách hiệu quả $p(O|\lambda)$ – xác suất phát sinh O từ mô hình λ ?
- *Bài toán 2:* cho trước chuỗi tín hiệu quan sát $O = O_1 O_2 \dots O_T$ và mô hình HMM đại diện bởi bộ tham số $\lambda = (\pi, A, B)$. Cần tìm ra chuỗi trạng thái tối ưu nhất $Q = q_1 q_2 \dots q_T$ đã phát sinh ra O ? Đây là bài toán rất quan trọng dùng cho nhận dạng.
- *Bài toán 3:* cho trước chuỗi tín hiệu quan sát $O = O_1 O_2 \dots O_T$. Làm thế nào để xác định các tham số mô hình $\lambda = (\pi, A, B)$ sao cho cực đại hóa xác suất $p(O|\lambda)$? Đây chính là bài toán học hay còn gọi là huấn luyện mô hình. Bài toán này đem lại một khả năng rất quan trọng của HMM: khả năng mô hình hóa một đối tượng cụ thể trong thực tế, mô hình hóa dữ liệu học.

Các tiêu mục tiếp theo sẽ lần lượt trình bày giải pháp cho ba bài toán này.

2.7.3.1 Bài toán 1 – Computing Likelihood

Mục tiêu của bài toán thứ nhất là tính $p(O|\lambda)$ – xác suất phát sinh O từ mô hình λ . Một giải pháp khả thi để tính $p(O|\lambda)$ là thông qua thuật toán forward-backward. Trước tiên, ta định nghĩa biến forward $\alpha_t(j)$ là xác suất ở trạng thái j tại thời điểm t và đã quan sát được đoạn O_1, O_2, \dots, O_t với mô hình λ cho trước:

$$\alpha_t(i) = p(O_1 | O_2 \dots O_t, q_t = S_i | \lambda) \quad (0.39)$$

Các biến $\alpha_t(i)$ có thể được tính theo qui nạp từng bước (hay thuật toán qui hoạch động) như sau:

1) Khởi tạo:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \quad (0.40)$$

2) Qui nạp:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1, \quad 1 \leq j \leq N \quad (0.41)$$

3) Đầu ra:

$$p(O | \lambda) = \sum_{i=1}^N \alpha_T(i) \quad (0.42)$$

Độ phức tạp tính toán của thuật toán forward là $O(N^2T)$.

Tương tự như trong thủ tục forward, thủ tục backward trước hết định nghĩa biến backward $\beta_t(i)$ là xác suất quan sát được đoạn $O_{t+1}, O_{t+2}, \dots, O_T$ cho trước trạng thái i ở thời điểm t và mô hình λ :

$$\beta_t(i) = p(O_{t+1} | O_{t+2} \dots O_T | q_t = i, \lambda) \quad (0.43)$$

Các biến $\beta_t(i)$ cũng được tính theo qui nạp từng bước như sau:

1) Khởi tạo:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \quad (0.44)$$

2) Qui nạp:

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \text{ với } t = T-1, T-2, \dots, 1 \text{ và } 1 \leq i \leq N \quad (0.45)$$

3) Đầu ra:

$$p(O | \lambda) = \sum_{j=1}^N \beta_1(j) \quad (0.46)$$

Cũng giống như thuật toán forward, độ phức tạp của thuật toán backward là $O(N^2T)$. Như vậy, thủ tục forward-backward là khả thi với chi phí tính toán hoàn toàn có thể chấp nhận được.

Đối với việc tìm lời giải cho bài toán 1, ta chỉ cần đến phần forward trong thủ tục forward-backward. Tuy nhiên, phần backward giúp tìm lời giải cho bài toán 3.

2.7.3.2 Bài toán 2 – decoding (Thuật toán Viterbi)

Mục tiêu của bài toán 2 là tìm ra chuỗi trạng thái “tối ưu” nhất $Q = q_1 q_2 \dots q_T$ đã phát sinh ra O . Một điều đáng lưu ý là có rất nhiều các tiêu chí “tối ưu” khác nhau cho việc xác định Q , nên lời giải cho bài toán này phụ thuộc vào tiêu chí ‘tối ưu’ được chọn.

Thuật toán viterbi định nghĩa biến $\delta_t(i)$ là xác suất cao nhất của đoạn chuỗi trạng thái dẫn đến S_i ở thời điểm t và đã quan sát được đoạn O_1, O_2, \dots, O_t cho trước mô hình λ :

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} p(q_1 q_2 \dots q_t = S_i, O_1 O_2 \dots O_t | \lambda) \quad (0.47)$$

$\delta_t(i)$ có thể được tính theo qui nạp:

$$\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] \cdot b_j(O_{t+1}) \quad (0.48)$$

Tuy nhiên thuật toán viterbi còn cần đến mảng $\psi_t(j)$ để lưu lại các tham số i làm cực đại hóa biểu thức (0.48), có thể hiểu tại thời điểm t trước j là i, nhằm mục đích truy vết (back trace). Chi tiết các bước của thuật toán viterbi như sau:

1) Khởi tạo:

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(O_1), & 1 \leq i \leq N \\ \psi_1(i) &= 0 \end{aligned} \quad (0.49)$$

2) Lặp qui nạp:

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i)a_{ij}]b_j(O_t), \quad 2 \leq t \leq T \\ 1 \leq j \leq N \quad (0.50)$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i)a_{ij}], \quad 2 \leq t \leq T \\ 1 \leq j \leq N \quad (0.51)$$

3) Kết thúc:

$$p^* = \max_{1 \leq i \leq N} [\delta_T(i)] \\ q_T^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)] \quad (0.52)$$

4) quay lui – backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \dots, 1 \quad (0.53)$$

Kết thúc thuật toán, chuỗi $Q = q_1^* q_2^* \dots q_T^*$ chính là lời giải thỏa đáng của bài toán 2.

2.7.3.3 Bài toán 3 – learning (Forward – Backward)

Mục tiêu của bài toán thứ 3, cũng là bài toán phức tạp nhất trong ba bài toán, là tìm cách cập nhật lại các tham số của mô hình $\lambda = (\pi, A, B)$ sao cho cực đại hóa xác suất $p(O|\lambda)$ – xác suất quan sát được chuỗi tín hiệu O từ mô hình.

Với một chuỗi tín hiệu quan sát hữu hạn bất kỳ O làm dữ liệu huấn luyện, chưa có một phương pháp tối ưu nào cho việc ước lượng các tham số $\lambda = (\pi, A, B)$ của mô hình theo hướng cực đại toàn cục. Tuy nhiên, bộ tham số λ có thể được chọn sao cho xác suất $p(O|\lambda)$ đạt cực đại cục bộ bằng thuật toán Forward-Backward hoặc thuật toán Baum-Welch, hoặc thuật toán Expectation-Maximization (EM).

Trước tiên, ta định nghĩa $\xi_t(i,j)$ là xác suất ở trạng thái S_i tại thời điểm t và rơi vào trạng thái S_j ở thời điểm $t+1$ cho trước mô hình λ và chuỗi tín hiệu quan sát O:

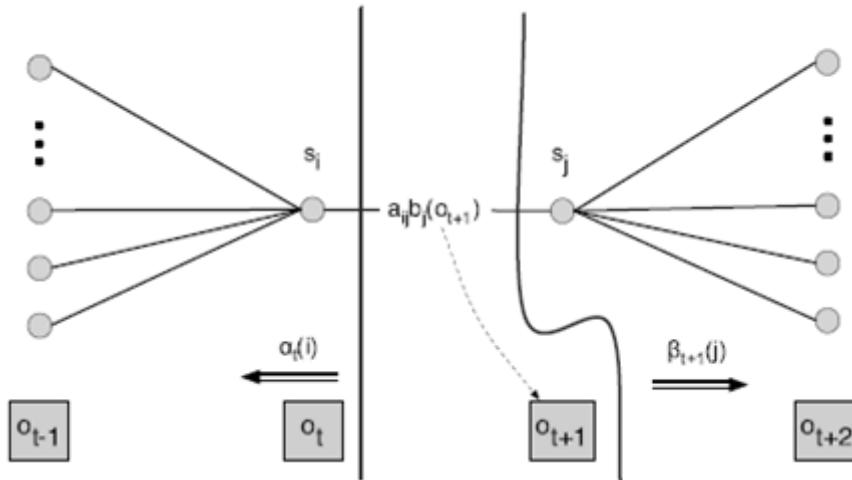
$$\xi_t(i, j) = p(q_t = S_i, q_{t+1} = S_j | O, \lambda) \quad (0.54)$$

Áp dụng xác suất Bayes:

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

Suy ra:

$$\xi_t(i, j) = \frac{p(q_t = S_i, q_{t+1} = S_j, O | \lambda)}{p(O | \lambda)} = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{p(O | \lambda)} \quad (0.55)$$



Hình 0.13 Cách tính $p(q_t = S_i, q_{t+1} = S_j, O | \lambda)$

Gọi $\gamma_t(i)$ là xác suất ở trạng thái S_i vào thời điểm t cho trước chuỗi tín hiệu quan sát O và mô hình λ :

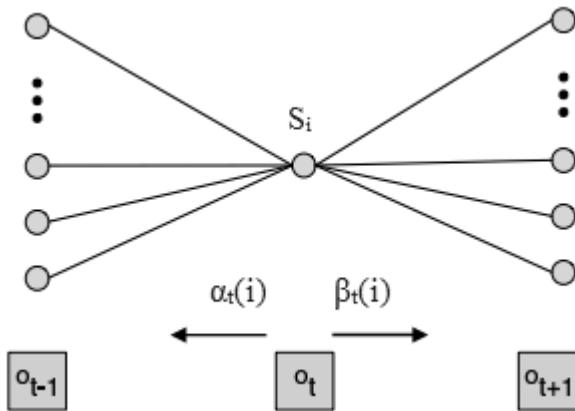
$$\gamma_t(i) = p(q_t = S_i | O, \lambda) \quad (0.56)$$

Áp dụng xác suất Bayes, suy ra:

$$\gamma_t(i) = \frac{p(q_t = S_i, O | \lambda)}{p(O | \lambda)} \quad (0.57)$$

Áp dụng xác suất forward và backward như trong Hình 0.14, suy ra:

$$\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{p(O | \lambda)} \quad (0.58)$$



Hình 0.14: $\gamma_t(i)$ được tính dựa trên xác suất forward và backward

Nếu ta lấy tổng $\gamma_t(i)$ theo $t \in [1, T-1]$, kết quả nhận được là số lần kỳ vọng chuyển trạng thái từ S_i . Tương tự, lấy tổng $\xi_t(i,j)$ theo $t \in [1, T-1]$, ta sẽ có số lần kỳ vọng chuyển từ trạng thái S_i sang S_j :

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{số lần kỳ vọng chuyển trạng thái từ } S_i. \quad (0.59)$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{số lần kỳ vọng chuyển từ trạng thái } S_i \text{ sang } S_j. \quad (0.60)$$

Với các đại lượng này, ta có các biểu thức cập nhật số của HMM như sau:

$$\bar{\pi}_i = \text{số lần kỳ vọng ở trạng thái } S_i \text{ vào thời điểm } (t=1) = \gamma_1(i) \quad (0.61)$$

$$\bar{a}_{ij} = \frac{\text{exnum}(S_i, S_j)}{\text{exnum}(S_i)} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (0.62)$$

$$\bar{b}_j(k) = \frac{\text{exnum_in}(S_j, v_k)}{\text{exnum_in}(S_j)} = \frac{\sum_{t=1}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)} \quad (0.63)$$

- $\text{exnum}(S_i, S_j)$: số lần kỳ vọng chuyển từ trạng thái S_i sang trạng thái S_j .

- $exnum(S_i)$: số lần kỳ vọng chuyển trạng thái từ S_i .
- $exnum_in(S_j, v_k)$: số lần kỳ vọng ở trạng thái S_j và quan sát được tín hiệu v_k .
- $exnum_in(S_j)$: số lần kỳ vọng ở trạng thái S_j .

Từ mô hình ban đầu $\lambda = (A, B, \pi)$ và chuỗi tín hiệu quan sát O , ta tính được vé phải của các biểu thức (3.4)(3.5)(3.6), kết quả nhận được chính là các tham số mới của mô hình $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$. Theo Baum đã chứng minh, ta luôn có $p(O | \bar{\lambda}) > p(O | \lambda)$ trừ phi mô hình λ đã đạt đến điểm tối ưu cục bộ (khi đó $\bar{\lambda} = \lambda$).

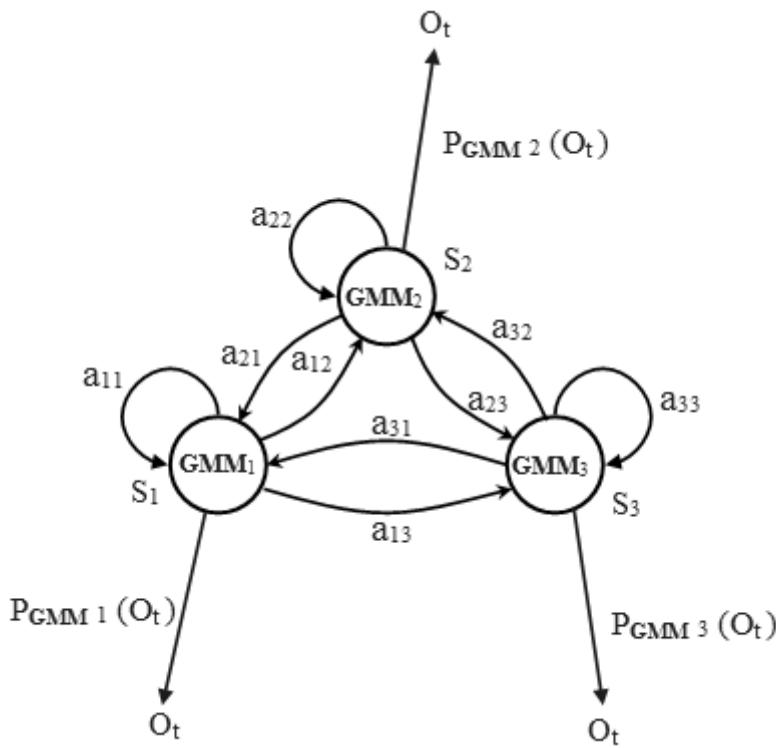
Như vậy, thuật toán Baum-Welch sẽ được áp dụng qua nhiều bước lặp để ước lượng và cập nhật các tham số mô hình cho đến khi hội tụ. Tuy nhiên, kết quả cuối cùng chỉ đạt được tối ưu cục bộ mà thôi. Thông thường, nếu các tham số được khởi tạo với các giá trị thích hợp, thì có một phần khả năng nào đó có thể giúp mô hình đạt được tối ưu toàn cục khi huấn luyện.

2.8 Mixture of gaussians Hidden Markov Model

2.8.1 Đặc tả mô hình

Phản **Error! Reference source not found.** đã trình bày về mô hình HMM và 3 bài toán cơ bản của nó, trong đó hàm mật độ xác suất của các tín hiệu quan sát là rời rạc. Mô hình HMM này được gọi là HMM rời rạc. Khi hàm mật độ xác suất trong mỗi trạng thái là hàm liên tục, ta có HMM liên tục. Thông thường, có hai dạng chính của HMM liên tục:

- **Gaussian Hidden Markov Model (GHMM)**: hàm mật độ xác suất trong mỗi trạng thái là hàm mật độ Gauss.
- **Mixture of Gaussians Hidden Markov Model (MGHMM)**: hàm mật độ xác suất trong mỗi trạng thái là hợp các hàm mật độ Gauss (mô hình GMM như đã trình bày trong mục **Error! Reference source not found.** ở trên).



Hình 0.15: Mô hình MGHMM 3 trạng thái

Phần này trình bày về mô hình MGHMM. Đây là một dạng của HMM liên tục, trong đó hàm mật độ xác suất của vector quan sát O_t được cho bởi mô hình GMM:

$$b_j(O_t) = p_{GMM_j}(O_t) \quad (0.64)$$

trong đó p_{GMM_j} chính là xác suất đầu ra của mô hình GMM trong trạng thái S_j . Như vậy, khả năng quan sát được các vector trong mỗi trạng thái sẽ bị chi phối bởi GMM của trạng thái đó. Hình 0.15 minh họa mô hình MGHMM có 3 trạng thái.

Cũng như trong định nghĩa của HMM, một mô hình MGHMM có N trạng thái và M phân phối Gauss trong mỗi trạng thái sẽ được đại diện bởi bộ tham số $\lambda = \{\pi, A, B\}$, trong đó:

- $A = \{ a_{ij} \}$, a_{ij} là xác suất chuyển từ trạng thái S_i sang trạng thái S_j .
- $\pi = \{ \pi_i \}$, π_i là xác suất khởi đầu của trạng thái S_i .
- $B = \{ b_j \}$, b_j là hàm mật độ xác suất trong trạng thái S_j .

a_{ij} và π_i thì không có gì thay đổi so với HMM, điểm khác biệt chính nằm ở b_j . Từ (0.38) ta có:

$$b_j(O_t) = \sum_{m=1}^M c_{jm} G(O_t, \mu_{jm}, U_{jm}) \quad (0.65)$$

trong đó trạng thái thứ $j \in [1, N]$, O_t là vector quan sát được ở thời điểm t , $G(\mu_{jm}, U_{jm})$ là hàm mật độ Gauss thứ m trong trạng thái S_j với vector trung bình μ và ma trận hiệp phương sai U tương ứng (dùng U thay cho Σ để tránh nhầm lẫn với ký hiệu Σ tổng), c_{jm} là trọng số (không âm) của phân phối Gauss

thứ m trong trạng thái S_j thỏa ràng buộc xác suất $\sum_{m=1}^M c_{jm} = 1$.

Về 3 bài toán cơ bản của HMM, giải pháp cho bài toán 1 và bài toán 2 là tương tự cho mô hình MGHMM. Riêng đối với bài toán 3, có một số điểm khác biệt trong công thức cập nhật tham số B của MGHMM so với HMM.

2.8.2 Huấn luyện tham số

Giải pháp cho bài toán huấn luyện mô hình MGHMM $\lambda = \{A, B, \pi\}$ vẫn là thuật toán Baum-Welch (tương đương EM [2]). Không có gì khác biệt nhiều so với huấn luyện HMM, các công thức cập nhật A và π vẫn là:

$$\bar{\pi}_i = \gamma_1(i) \quad (0.66)$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (0.67)$$

Tuy nhiên đối với $B = \{ b_j \}$, b_j gồm 3 tham số con $\{ c_{jm}, \mu_{jm}, U_{jm} \}$, thì phương pháp ước lượng phức tạp hơn. Trước tiên, ta định nghĩa $\gamma_t(j, k)$ là xác suất quan sát được vector O_t bởi phân phối Gauss thứ k trong trạng thái S_j tại thời điểm t cho trước chuỗi vector quan sát O và mô hình λ :

$$\gamma_t(j, k) = \left[\frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \right] \left[\begin{array}{l} \frac{c_{jk}G(O_t, \mu_{jk}, U_{jk})}{\sum_{m=1}^M c_{jm}G(O_t, \mu_{jm}, U_{jm})} \\ \end{array} \right] \quad (0.68)$$

trong đó, $\alpha_t(j), \beta_t(j)$ là các biến forward-backward, c_{jk} , μ_{jk} và U_{jk} lần lượt là trọng số (weight), vector trung bình (mean vector) và ma trận hiệp phương sai (covariance matrix) của phân phối Gauss thứ k trong trạng thái S_j . Với đại lượng $\gamma_t(j, k)$, ta có công thức cập nhật các thành phần của b_j như sau:

$$\bar{c}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k)}{\sum_{t=1}^T \sum_{k=1}^M \gamma_t(j, k)} \quad (0.69)$$

$$\bar{\mu}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) \cdot O_t}{\sum_{t=1}^T \gamma_t(j, k)} \quad (0.70)$$

$$\bar{U}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) \cdot (O_t - \bar{\mu}_{jk})(O_t - \bar{\mu}_{jk})'}{\sum_{t=1}^T \gamma_t(j, k)} \quad (0.71)$$

Giải thuật huấn luyện:

- Input: chuỗi các vector đặc trưng $O = \{O_1, O_2, \dots, O_T\}$,

mô hình $\lambda = \{A, B, \pi\}$

- Output: $\lambda^* = \{A^*, B^*, \pi^*\}$

while (!converged && iter < maxIter)

{

// Tính các biến forward α

for i = 1 → N

```

 $\alpha_t(i) = \pi_i b_i(O_t)$ 
for t = 1 → T-1
    for j = 1 → N
         $\alpha_{t+1}(j) = \left[ \sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(O_{t+1})$ 
    // Tính các biến backward β
    for i = 1 → N
         $\beta_T(i) = 1$ 
    for t = T-1 → 1
        for i = 1 → N
             $\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$ 
    // Cập nhật các tham số mô hình
    λ = updateParameters (λ, O, α, β)
}

```

Phương thức updateParameters cập nhật tham số của mô hình MGHMM theo các công thức (0.69), (0.70), (0.71).

Như đã đề cập, thuật toán Baum-Welch chỉ có thể huấn luyện mô hình đến tối ưu cục bộ mà thôi. Kết quả thu được không phải lúc nào cũng là tối ưu toàn cục. Chưa có giải pháp tuyệt đối cho vấn đề này, tuy nhiên việc khởi tạo tốt giá trị ban đầu cho các tham số mô hình sẽ đem lại khả năng cao hơn cho việc đạt tối ưu toàn cục, đồng thời đẩy nhanh tốc độ hội tụ trong huấn luyện.

2.9 Các thành phần của Sphinx-4

Có 3 thành phần chính trong Framework của Sphinx4. Chúng gồm:

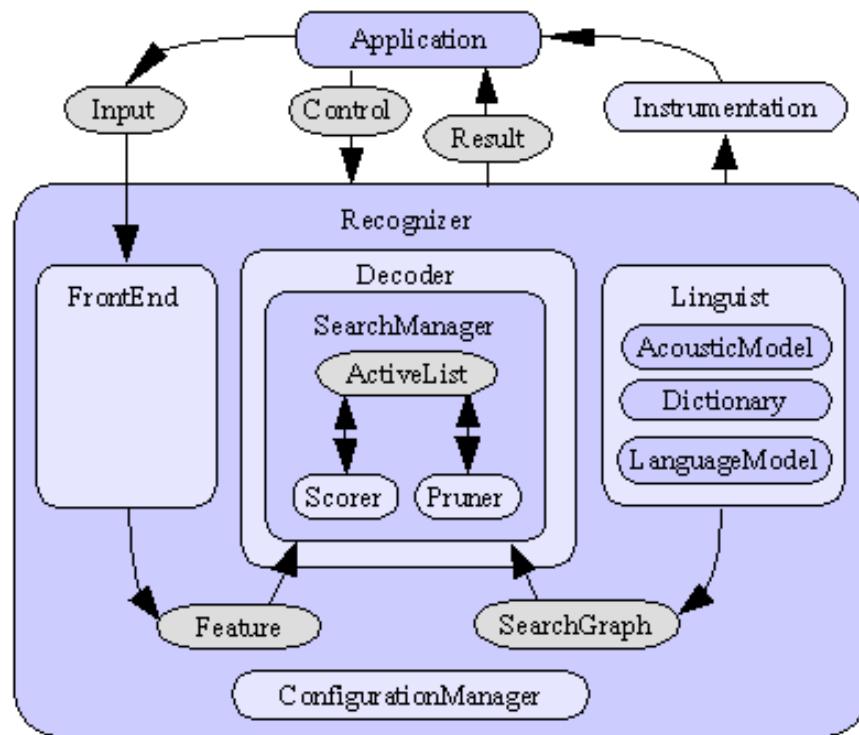
-  Frontend : thiết bị đầu cuối
-  Decoder : Bộ mã hóa và giải mã
-  Linguist : Bộ ngôn ngữ lưu trữ các dữ liệu ngôn ngữ mẫu

Trong nghiên cứu này, chúng ta sẽ tập trung vào module ngôn ngữ (Linguist) vì đó là module cao nhất để chúng ta thâm nhập trực tiếp và thay đổi các dữ liệu cần thiết cho việc xây dựng ứng dụng của chúng ta

Hai tiến trình nòng cốt của Sphinx là: Huấn luyện (Training) và Nhận dạng (Decoding).

Huấn luyện: là tiến trình quan trọng và cần thiết để xây dựng mô hình Linguist (gồm mô hình âm thanh – acoustic model và mô hình ngôn ngữ - language model). Trong bước này, chúng ta cần ghi âm lại âm thanh của nhiều người để tạo nên 1 tập tin lưu trữ âm thanh chuẩn (Corpus) mà chúng ta dùng để xây dựng ngôn ngữ và các mô hình âm thanh của bộ máy. Sau khi hệ thống được huấn luyện, nó bây giờ có thể được dùng để nhận dạng giọng nói được mã hóa từ các âm thanh trong text form.

- Nhận dạng : được đặc tả rất chi tiết trong biểu đồ sau:



Hình 1: Kiến trúc Sphinx4

2.10 Thiết bị đầu cuối – Frontend

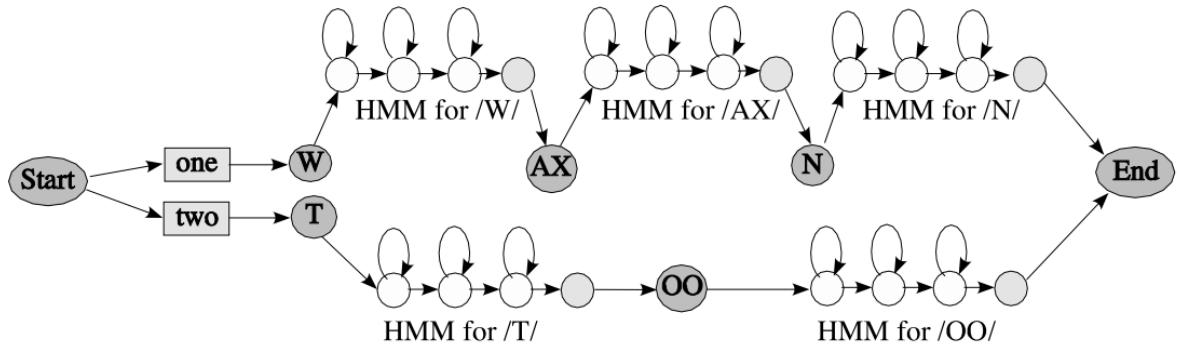
Frontend là một trong những thành phần chính của bộ máy mà tác động trực tiếp với người nói. Nó thu lấy tín hiệu đầu vào (âm thanh) sau đó chuyển thành các vector đặc trưng mà được dùng để đo lường các đặc tính của âm giọng.

2.11 Biểu đồ tìm kiếm – Search Graph

Search Graph dựa trên các đơn vị âm thanh của mô hình Markov ẩn. Nó được dựng dựa trên một cấu trúc đặc biệt.

Search Graph là biểu diễn của bộ ngôn ngữ đầu ra (Linguist) mà bộ giải mã nhận dạng dùng để nhận dạng và sinh kết quả.

Ví dụ:



Hình 2: ví dụ Search Graph

2.12 Bộ giải mã – decoder

Mục đích: dùng để so sánh âm thanh khảo sát (từ thiết bị vào) với SearchGraph (từ Linguist) để sinh ra kết quả

Các thành phần con:

- Trình tính điểm (Scorer): dùng để đo chính xác các đặc trưng của âm thanh được trả ra từ mô hình âm học (acoustic model).
- Bộ cắt tỉa (pruner) được dùng để giảm số khả năng có thể trong suốt quá trình tìm trong trường hợp nhiều nốt có sự tự chuyển tiếp.
- Danh sách hoạt động (active list) lưu vết các đường hoạt động thông qua SeachGraph bằng cách lưu trữ các trường hợp đã chọn trước.

2.13 Bộ ngôn ngữ - Linguist

Mục đích: Là một mô-đun rời, cho phép người dùng cấu hình hệ thống tự động và được phép kế thừa bộ ngôn ngữ linguist khác. Linguist sinh ra “SearchGraph” mà được bộ giải mã decoder truy xuất các cấu trúc ngôn ngữ, tham số âm học và từ điểm.

Các thành phần con :

- **Mô hình ngôn ngữ (Language model) :**

- Định nghĩa cấu trúc ngữ pháp cho câu dựa trên mô hình thống kê ngẫu nhiên N-Gram. Mô hình này cung cấp các từ có thể nối với n-1 từ đi trước. Nói đơn giản nghĩa là đây là một tập hợp các câu có thể có.
- Mô hình ngôn ngữ Sphinx-4 cung cấp 5 chuẩn định dạng:
 - Simplewordlistgrammar
 - JSFGrammar
 - LMGrammar
 - SimpleGramModel
 - LargeTriGramModel
 - **Mô hình âm học (Acoustic model):** mô hình này có thể kết nối giữa âm thanh và HMM.
 - **Từ điển :**
 - Từ điển cung cấp phát âm của các từ có trong mô hình ngôn ngữ. Cách phát âm sẽ chia công việc ra các phần riêng biệt có trong mô hình âm học (Acoustic model)
 - Dựa trên mô hình âm học, chúng ta có thể xây dựng một bộ từ điển mới bằng cách sử dụng các âm có sẵn trong bộ âm học.

2.14 Tiến trình huấn luyện

Tiến trình huấn luyện xây dựng cơ sở dữ liệu ngôn ngữ linguist , thành phần quan trọng nhất của hệ thống nhận dạng tiếng nói Sphinx4, cho hệ thống (gồm 2 phần : âm học và ngôn ngữ) Bước khởi tạo của quá trình huấn luyện chính là định nghĩa danh sách các âm thanh và từ điển cho mô hình. Sau đó chúng ta sẽ ghi âm lại giọng nói của chúng ta để xây dựng một tập hợp các âm thanh (corpora) làm nguyên liệu cho quá trình Training. Trong quá trình training, lời nói của chúng ta sẽ được chuyển thành dạng Cepstral (dưới dạng

véc tơ đặc trưng). Dựa trên các vector này, HMM được xây dựng cho mỗi âm trong danh sách âm. Sau khi training, chúng ta sẽ có một bộ ngôn ngữ linguist được xây dựng trên HMM mà có thể giúp ta nhận dạng lời nói của bất cứ người nào.

2.15 Tiến trình nhận dạng

Cách thường được dùng để nhận dạng tiếng nói là : chúng ta lấy sóng âm, tách các lời nhò vào các khoảng lặng, sau đó cố gắng nhận dạng xem cái điều gì đang được nói trong mỗi tiếng. Để làm được điều đó, chúng ta muốn lấy tất cả các tổ hợp có thể có, và cố gắng ghép chúng với các âm. Chúng ta chọn tổ hợp tốt nhất. Có vài thứ quan trọng trong cách ghép này.

Trên hết là khái niệm “**features**” – đặc trưng. Vì số tham số thì nhiều, mà chúng ta đang cố tối ưu nó. Con số được tính ra từ lời nói thường là bằng cách chia lời nói trên các khung. Sau đó mỗi khung có độ dài thông thường là 10 mili giây, chúng ta trích ra 39 con số mà dùng để thể hiện lời nói. Nó được gọi là **feature vector** – véc tơ đặc trưng. Cách sinh ra các số đó là chủ đề nghiên cứu sôi động., nhưng trong trường hợp đơn giản, nó bắt nguồn từ quang phổ (**spectrum**) .

Thứ hai là một khái niệm “**model**” – **mô hình** . mô hình được tả một vài đối tượng toán học mà gom các thuộc tính của từ được nói. Trong thực tế, đối với mô hình âm thanh của senone là tổ hợp Gauxor của 3 trạng thái của nó – để hiểu đơn giản, nó là véc tơ tính năng gần đúng nhất, Từ khái niệm của mô hình các vấn đề tăng lên – mô hình tốt cở nào để có thể áp dụng thực tế, mô hình có thể tốt hơn bằng cách giải quyết các vấn đề của mô hình, mô hình thích nghi như thế nào để thay đổi điều kiện.

Thứ ba, nó là tiến trình tự ghép. Vì nó có thể mất rất nhiều thời gian hơn cả thời vũ trụ hình thành đến nay để so sánh các véc tơ tính năng với tất cả mô hình, việc tìm kiếm thường được tối ưu hóa bằng nhiều thủ thuật . Dưới

bất cứ quan điểm nào chúng ta gồm các biến ghép tốt nhất và mở rộng chúng ngay khi chúng bắt đầu tạo ra các biến ghép cho khung kế tiếp.

Mô hình: Có 3 loại:

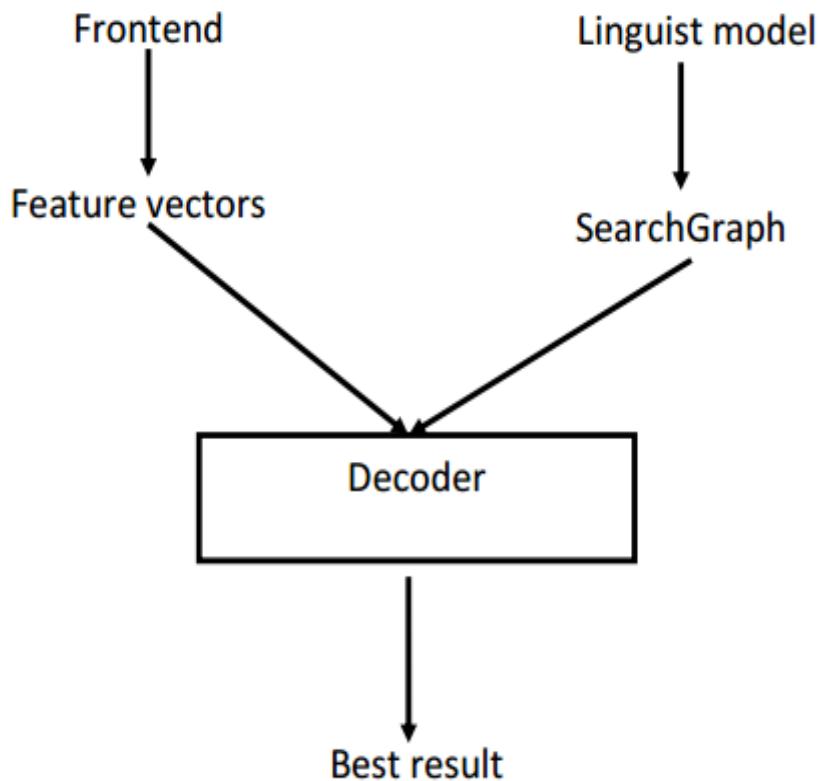
Mô hình âm thanh: gồm các thuộc tính âm thanh cho mỗi senone . Có nhiều mô hình độc lập nội dung mà có các thuộc tính (hầu hết là các véc tơ đặc trưng cho mỗi tiếng) và các mô hình phụ thuộc nội dung (xây dựng từ các senone có nội dung)

Phonetic dictionary: từ điển tiếng – gồm một bản có thứ tự các từ liên kết với các tiếng phiên âm. Bản này không hiệu quả lắm . Lấy ví dụ, chỉ 2 hoặc 3 phiên âm đa dạng được ghi lại, nhưng hầu như không đủ trong thực tế. Từ điển không chỉ là 1 danh sách các từ liên kết với phiên âm, mà nó còn phải chứa các hàm phức tạp về thuật toán máy học.

Language Model : Mô hình ngôn ngữ được dùng để thu hẹp số từ cần tìm. Nó định nghĩa các từ có thể theo sau các từ đã nhận dạng trước (nên nhớ rằng việc tổ hợp là một tiến trình liên tục) và hạn chế các tổ hợp vô nghĩa bằng cách loại bỏ các từ không có trong thực tế.

Một số khái niệm khác: Lattice, N-best, Word confusion networks, Speech database, Text databases.

Nhiệm vụ chính của tiến trình này là tìm ra từ thích hợp nhất với giọng nói đầu vào. Bước đầu của tiến trình nhận dạng cũng giống với huấn luyện, lời nói sẽ được ghi âm, chuyển thành dạng vecto đặc trưng bởi thiết bị đầu cuối. Mô hình ngôn ngữ Linguist sẽ sinh SearchGraph mà dùng để xác định từ bởi bộ giải mã decoder. Bộ giải mã này, sẽ dựa vào SearchGraph và vecto đặc trưng, nó sẽ áp dụng thuật toán tìm kiếm để tìm ra kết quả tốt nhất.



Hình 3: Tiến trình nhận dạng

TỐI ƯU HÓA

Khi nhận dạng giọng nói đang được phát triển, vấn đề phức tạp nhất là làm sao tìm kiếm chính xác (duyệt nhiều biến để tìm các tổ hợp có thể) và để làm cho nó đủ nhanh để không chạy hoài không dừng được. Ngoài ra còn có vấn đề là làm sao cho các mô hình phù hợp với lời nói vì mô hình này không hoàn hảo.

Thông thường, hệ thống được thử nghiệm trên một cơ sở dữ liệu thử nghiệm mà đại diện cho các mục tiêu nhiệm vụ một cách chính xác.

Các đặc điểm sau đây được sử dụng:

Mật độ lỗi: Word Error Rate.

$$\text{WER} = (I + D + S) / N$$

WER thường được đo bằng phần trăm.

Với:

I : inserted – là các từ được thêm vào.

D: deleted - các từ bị xóa bỏ.

S : substituted – các từ bị thay thế.

N: tổng số từ.

Độ chính xác cũng giống mật độ lỗi , nhưng nó không tính các từ thêm vào.

$$\text{Accuracy} = (N - D - S) / N$$

Độ chính xác là cách đo không tốt cho hầu hết các nhiệm vụ, vì việc thêm các từ cũng góp phần quan trọng để tìm ra kết quả. Nhưng đối với một số việc nhận dạng, độ chính xác trở thành một phép đo hợp lý cho việc thực thi giải mã.

Tốc độ: Với file âm thanh dài 2 giờ và giải mã mất 6 giờ. Tốc độ giải mã gấp 3 lần thời lượng ghi âm.

Cung ROC. Khi chúng ta nói về các nhiệm vụ dò tìm, có các cảnh báo sai và đúng/bỏ lỡ; Cung ROC được dùng một cung là hình vẽ đặc tả của số cảnh báo sai với số đúng, và có tìm điểm tối ưu mà các cảnh báo sai nhỏ và số cặp ghép đúng là 100%.

2.10 Nhận xét

Sphinx4 được cho là một framework tốt cho sinh viên, giáo sư nghiên cứu. Sphinx4 hỗ trợ chúng ta nhiều tình mà không framework nào có. Những tinh năng tiêu biểu sẽ được liệt kê bên dưới.

❖ Lợi thế

Sphinx4 nghệ thuật nhận dạng liên tục

Sphinx4 được phát triển nhờ sự kết hợp của các mô hình cấp cao trong hệ thống cũ cũng như các yêu cầu mới từ nhiều khu vực. Nó phát triển và mở rộng khả năng hơn các phiên bản trước (Sphinx3, Sphinx2)

Hầu hết các công nghệ nhận dạng giọng nói trong Sphinx là các công nghệ cao cấp được phát triển nâng cấp thường xuyên. Ví dụ như nhờ có kỹ thuật lọc nhiễu tốt nhất, chất lượng ghi âm giọng nói tăng cao.

Sphinx4 là một framework có thể sử dụng như các module , plugin

Kiến trúc module , plugin cho phép ta chia Sphinx4 thành nhiều phần nhỏ và thực hiện nghiên cứu một cách riêng lẻ. Các học giả có thể phát triển các tính năng mới mà cần cho nâng cao khả năng nhận dạng.

Kiến trúc đầu cuối bao gồm khả năng plug-in kế thừa từ nhiều module như : nhán mạnh tiền tố , cửa sổ Hamming , FFT (Fast Fourier Transform), tần số Mel, lọc dãy tần số, tách ánh xạ cô-sin...

Kiến trúc mô hình ngôn ngữ gồm các mẫu ngôn ngữ rời , hỗ trợ ASOO và phiên bản nhị phân của unigram, bigram, trigram, Java Speech API Grammar format (JSGF), and ARPA format FST grammars.

Kiến trúc mô hình âm học hỗ trợ trình cắm rời cho các mô hình âm học của Sphinx3

Sphinx-4 là 1 hệ thống linh hoạt mà có thể biểu diễn nhiều loại nhiệm vụ nhận diện : có thể làm việc với nhiều kiểu đầu vào để cho ra cùng một kết quả. Người nói có thể nói trực tiếp vào hệ thống , kết quả gần như được tạo ra ngay tức thì dưới dạng text hoặc các hình thức khác. Bên cạnh đó, chúng ta cũng có thể thực hiện nhận dạng bằng cách thực thi kiểm thử hỏi qui xem người dùng vừa nói gì sẽ được ghi âm thành một file wav , nội dung của file cũng có thể được nhận dạng như nói trực tiếp

Sphinx-4 cho phép ta dựng nhiều kiểu từ vựng lớn nhỏ, chúng ta cũng có thể đóng góp một từ vựng chỉ có các con số rời rạc, số liên tục , vài từ nhỏ,

hoặc trọng tất cả lại (từ vựng lớn). Trong Sphinx-4, các nhà phát triển xây dựng sẵn một bộ từ vựng gồm 64,000 từ tiếng anh. Chúng ta cũng nên làm một bộ từ vựng như vậy cho tiếng việt.

Sphinx-4 có thể làm việc độc lập với bất cứ người nói nào: Sau khi huấn luyện đủ dữ liệu, Sphinx-4 có thể nhận dạng một giọng chuẩn mà không cần train trước. Nó có thể giải thích đơn giản bởi mô hình HMM mà được dùng như cái nhân của Sphinx-4. Vì kết quả được sinh ra dựa trên các tham số khả năng của các mẫu quan sát, vì vậy, giọng nói không cần chính xác giống với giọng đã train sẵn.

Chúng ta xem xét tiếng việt, chúng ta nói 3 giọng của 3 miền Bắc , trung , nam, nếu chúng ta dùng Sphinxtrain cho cả 3 giọng , chúng ta có thể nhận ra tất cả người nào nói tiếng việt với độ chính xác rất cao.

Sphinx-4 là một hệ thống mã nguồn mở phát triển trên Java: chúng ta cũng có thể chạy trên các môi trường mà hỗ trợ nền tảng java như window, linux... nó cũng rất tiện cho những ai quen dùng các nền tảng khác mà có làm việc của họ trong Sphinx-4.

2.6 Bất lợi

Sphinx-4 không phải là một hệ thống thân thiện:

Sphinx-4 là mã nguồn mở , vậy nên ko phải là một hệ thống chuẩn , mà có thể chạy dễ dàng, nó gồm một nhánh các gói mà cần phải cấu hình mới chạy được

Bên cạnh đó , Sphinx-4 vẫn có các lỗi mà chúng ta phải sửa nếu muốn phát triển ứng dụng nhận diện giọng nói trên Sphinx-4. Vì không có nhiều chỉ dẫn chi tiết về kiến trúc Sphinx-4 và làm sao để hệ thống làm việc vì chúng ta

thường hiểu nhập nhằng về vấn đề này. Đây cũng là nguyên nhân chính dẫn đến chúng ta gặp các lỗi bất ngờ.

CHƯƠNG 3. CÁC PACKAGE VÀ CLASS

Danh sách tất cả các package :

Sphinx4 được tạo thành từ 10 package lớn, với hơn 40 package con.

Sau đây là danh sách tất cả các package của Sphinx4, chúng em xin phép được giữ nguyên nội dung chú giải của các package / class / method để đảm bảo tính toàn vẹn cho nội dung ý nghĩa.

Ghi chú : Những package được in đậm là những package quan trọng.

Packages	
com.sun.speech.engine	
com.sun.speech.engine.recognition	
com.sun.speech.engine.synthesis	Provides a JSML 0.6 parser as well as a partial and unsupported support of interfaces in the javax.speech.synthesis package for JSAPI 1.0.
edu.cmu.sphinx.decoder	Provides a set of high level classes that can be used to configure and initiate the speech recognition decoding process.
edu.cmu.sphinx.decoder.pruner	Provides an interface that represents the pruning

	facility
<u>edu.cmu.sphinx.decoder.scorer</u>	Provides an interface that represents entities that can be scored, and an interface and several implementations of a scorer that can score these entities.
<u>edu.cmu.sphinx.decoder.search</u>	Provides classes and interfaces that are used to manage the search through the search graph.
<u>edu.cmu.sphinx.decoder.search.stats</u>	
<u>edu.cmu.sphinx.frontend</u>	Provides a set of high level classes and interfaces that are used to perform digital signal processing for speech recognition.
<u>edu.cmu.sphinx.frontend.databranch</u>	This package contains some classes to create a subbranch of a FronEnd at an arbitrary position.
<u>edu.cmu.sphinx.frontend.endpoint</u>	Provides classes and interfaces used for speech endpointing.
<u>edu.cmu.sphinx.frontend.feature</u>	Provides classes that processes features.
<u>edu.cmu.sphinx.frontend.filter</u>	Provides classes that implement frequency filters
<u>edu.cmu.sphinx.frontend.frequencywarp</u>	Provides classes that perform frequency warping.

<u>edu.cmu.sphinx.frontend.transform</u>	Provides classes that transform data from one domain into another.
<u>edu.cmu.sphinx.frontend.util</u>	Provides classes that are generally useful to the various frontend classes.
<u>edu.cmu.sphinx.frontend.window</u>	Provides classes that implement windowing functions
<u>edu.cmu.sphinx.instrumentation</u>	Provides a set of classes that monitor and track operational aspects of the Sphinx system.
<u>edu.cmu.sphinx.jsapi</u>	Provides support for the Java Speech API for Sphinx-4
<u>edu.cmu.sphinx.jsgf</u>	
<u>edu.cmu.sphinx.jsgf.parser</u>	
<u>edu.cmu.sphinx.jsgf.rule</u>	
<u>edu.cmu.sphinx.linguist</u>	Provides a set of interfaces and classes that are used to define the search graph used by the decoder.
<u>edu.cmu.sphinx.linguist.acoustic</u>	Provides classes that represent the acoustic model.
<u>edu.cmu.sphinx.linguist.acoustic.tiedstate</u>	Provides classes that represent acoustic model in terms of a set of tied states.
<u>edu.cmu.sphinx.linguist.acoustic.tiedstate.HTK</u>	

<u>edu.cmu.sphinx.linguist.acoustic.trivial</u>	Provides classes that represent a trivial acoustic model.
<u>edu.cmu.sphinx.linguist.dflat</u>	
<u>edu.cmu.sphinx.linguist.dictionary</u>	Provides a generic interface to a dictionary as well as several implementations.
<u>edu.cmu.sphinx.linguist.flat</u>	Provides an implementation of the Linguist that statically represents the search space as a flat graph, where each word in the vocabulary has its own branch.
<u>edu.cmu.sphinx.linguist.language.classes</u>	
<u>edu.cmu.sphinx.linguist.language.grammar</u>	Provides classes and interfaces that can be used to represent a graph of words and word transitions.
<u>edu.cmu.sphinx.linguist.language.ngram</u>	Provides classes and interfaces that represent a stochastic language model
<u>edu.cmu.sphinx.linguist.language.ngram.large</u>	Provides an implementation of the LanguageModel interface.
<u>edu.cmu.sphinx.linguist.lextree</u>	Provides an implementation of the Linguist that represents the search space as a lex tree.
<u>edu.cmu.sphinx.linguist.util</u>	Provides a set of classes that are useful by implementations of the Linguist interface.

<u>edu.cmu.sphinx.recognizer</u>	Provides a set of high level classes and interfaces that are used to perform speech recognition with the Sphinx-4 speech recognition system.
<u>edu.cmu.sphinx.result</u>	Provides a set of classes that represent the result of a recognition.
<u>edu.cmu.sphinx.tools.audio</u>	Provides an tool that records and displays the waveform and spectrogram of an audio signal.
<u>edu.cmu.sphinx.tools.batch</u>	Provides an tool that performs batch-mode speech recognition
<u>edu.cmu.sphinx.tools.endpoint</u>	
<u>edu.cmu.sphinx.tools.feature</u>	Description Provides an tool that generates different types of features (MFCC, PLP, spectrum) from audio files.
<u>edu.cmu.sphinx.tools.gui</u>	
<u>edu.cmu.sphinx.tools.gui.reader</u>	
<u>edu.cmu.sphinx.tools.gui.util</u>	
<u>edu.cmu.sphinx.tools.gui.writer</u>	
<u>edu.cmu.sphinx.tools.live</u>	Provides an tool that performs pseudo-live-mode speech recognition
<u>edu.cmu.sphinx.tools.tags</u>	Provides tools to post-process JSGF RuleParse objects using ECMAScript

	Action Tags for JSGF.
<u>edu.cmu.sphinx.util</u>	Provides a set of general purpose utility classes for Sphinx.
<u>edu.cmu.sphinx.util.machlearn</u>	Provides a set of classes and interfaces which aim to unify the different probabilistic modeling approaches spread over the S4-library into a common probabilistic framework.
<u>edu.cmu.sphinx.util.props</u>	Provides a mechanism for managing persistent configuration data.
<u>edu.cmu.sphinx.util.props.tools</u>	

Trong số đó các package sau đây nắm vai trò quan trọng chủ chốt :

3.1 Package `edu.cmu.sphinx.decoder`

Đây là package chứa các class nắm vai trò chủ chốt trong việc *cấu hình và khởi tạo quá trình giải mã nhận dạng*.

Class Summary

<u>AbstractDecoder</u>	An abstract decoder which implements all functionality which is independent of the used decoding-paradigm (pull/push).
<u>Decoder</u>	The primary decoder class
<u>FrameDecoder</u>	A decoder which does not use the common pull-principle of

S4 but processes only one single frame on each call of decode().

Class chính : Decoder

edu.cmu.sphinx.decoder.Decoder

Class Decoder

```
java.lang.Object
└── edu.cmu.sphinx.decoder.AbstractDecoder
    └── edu.cmu.sphinx.decoder.Decoder
```

All Implemented Interfaces:

[ResultProducer](#), [Configurable](#)

Constructor Summary

[Decoder\(\)](#)

[Decoder](#)([SearchManager](#) searchManager, boolean fireNonFinalResults, boolean autoAllocate, java.util.List<[ResultListener](#)> resultListeners, int featureBlockSize)

Main Method Summary

Result	decode (java.lang.String referenceText)
------------------------	---

Decode frames until recognition is complete.

void	newProperties (PropertySheet ps)
------	---

This method is called when this configurable component needs to be reconfigured.

3.2 Package `edu.cmu.xphinx.frontend.util`

Đây là package quản lý các thiết bị đầu cuối

Class chính: [AudioFileDataSource](#), [Microphone](#), [WavWriter](#).

Class Summary	
AudioContinuityTester	FrontEnd element that asserts the audio-stream to be continuous.
AudioFileDataSource	An AudioFileDataSource generates a stream of audio data from a given audio file.
ConcatAudioFileDataSource	Concatenates a list of audio files as one continuous audio stream.
Microphone	A Microphone captures audio data from the system's underlying audio input systems.
StreamDataSource	A StreamDataSource converts data from an InputStream into Data objects.
VUMeterMonitor	A VU meter to be plugged into a front-end.
VUMeterPanel	
WavWriter	Stores audio data into numbered (MS-)wav files.

Một AudioFileDataSource tạo ra một dòng dữ liệu âm thanh từ một tập tin âm thanh nhất định. Toàn bộ thông tin cần thiết liên quan đến việc định dạng âm thanh được đọc trực tiếp từ tập tin này. Cần gọi setAudioFile (java.io.File, String) để lấy file, và dùng getData() để có được cấu trúc dữ liệu.

Sử dụng JavaSound như lớp phụ trợ, có thể xử lý tất cả các tập tin âm thanh được hỗ trợ bởi JavaSound, hỗ trợ cho: .wav, .au và .aiff. Sử dụng Plus-

in (cf. <http://www.jsresources.org/>), nó có thể mở rộng để hỗ trợ .ogg, .mp3, .speex và những file có đuôi khác.

`edu.cmu.sphinx.frontend.util.AudioFileDataSource`

Class AudioFileDataSource

```
java.lang.Object
└ edu.cmu.sphinx.util.props.ConfigurableAdapter
    └ edu.cmu.sphinx.frontend.BaseDataProcessor
        └ edu.cmu.sphinx.frontend.util.AudioFileDataSource
```

Constructor Summary

[`AudioFileDataSource\(\)`](#)

[`AudioFileDataSource\(int bytesPerRead,`](#)
`java.util.List<AudioFileProcessListener> listeners)`

Method Summary

<code>void</code> <code>addNewFileListener(AudioFileProcessListener l)</code> Adds a new listener for new file events.
<code>Data</code> <code>getData()</code> Reads and returns the next Data from the InputStream of StreamDataSource, return null if no data is read and end of file is reached.

`edu.cmu.sphinx.frontend.util.Microphone`

Class Microphone

```
java.lang.Object
└ edu.cmu.sphinx.util.props.ConfigurableAdapter
    └ edu.cmu.sphinx.frontend.BaseDataProcessor
        └ edu.cmu.sphinx.frontend.util.Microphone
```

Khi microphone thu được dữ liệu âm thanh từ hệ thống trên cơ sở âm thanh đầu vào. Chuyển đổi chúng thành đối tượng dữ liệu. Khi phương thức **startRecording()** được gọi, một luồng mới sẽ được tạo và bắt đầu ghi âm, và sẽ dừng khi phương thức **stopRecording()** được gọi. Sau đó gọi phương thức **getData()** để lấy ra dữ liệu âm thanh đã ghi được dưới định dạng đối tượng dữ liệu.

Class Microphone này sẽ cố gắng thu một thiết bị âm thanh với định dạng được đặc tả trong file cấu hình. Nếu một thiết bị mà không nhận được định dạng, nó sẽ thử nhận một thiết bị với một định dạng âm thanh mà có tỉ lệ mẫu cao hơn tỉ lệ mẫu trong cấu hình, nếu vẫn không được thì một biến cờ lỗi sẽ được bật, và hàm **startRecording()** sẽ trả về false

Constructor Summary

[Microphone\(\)](#)

[Microphone](#)(int sampleRate, int bitsPerSample, int channels, boolean bigEndian, boolean signed, boolean closeBetweenUtterances, int msecPerRead, boolean keepLastAudio, java.lang.String stereoToMono, int selectedChannel, java.lang.String selectedMixerIndex, int audioBufferSize)

Method Summary

Data [getData\(\)](#)

Reads and returns the next Data object from this Microphone, return null if there is no more audio data.

boolean	<u>isRecording()</u> Returns true if this Microphone is recording.
private boolean	<u>open()</u> Opens the audio capturing device so that it will be ready for capturing audio.
boolean	<u>startRecording()</u> Starts recording audio.
void	<u>stopRecording()</u> Stops recording audio.

`edu.cmu.sphinx.frontend.util.WavWriter`

Class WavWriter

```

java.lang.Object
  ↘ edu.cmu.sphinx.util.props.ConfigurableAdapter
    ↘ edu.cmu.sphinx.frontend.BaseDataProcessor
      ↘ edu.cmu.sphinx.frontend.util.WavWriter

```

Lưu trữ âm thanh dưới dạng file wav.

Constructor Summary

[WavWriter\(\)](#)[WavWriter](#)(java.lang.String dumpFilePath, boolean isCompletePath, int bitsPerSample, boolean isSigned, boolean captureUtts)

Main Method Summary

Data	<u>getData()</u> Returns the processed Data output.
void	<u>initialize()</u>

	Initializes this DataProcessor.
static void writeWavFile (double[] signal, int sampleRate, java.io.File targetFile)	Writes a given double array into a wav file (given the sample rate of the signal).

```
public WavWriter(java.lang.String dumpFilePath,
                boolean isCompletePath,
                int bitsPerSample,
                boolean isSigned,
                boolean captureUtts\)
```

CloseDataStream

```
private void CloseDataStream()  
    throws java.io.IOException
```

Throws: java.io.IOException

Cung cấp các lớp hữu ích cho các lớp khác nhau.

java.lang.Object

- └ [edu.cmu.sphinx.util.props.ConfigurableAdapter](#)
- └ [edu.cmu.sphinx.frontend.BaseDataProcessor](#)
- └ **edu.cmu.sphinx.frontend.util.AudioFileDataSource**

java.lang.Object

- └ [edu.cmu.sphinx.util.props.ConfigurableAdapter](#)
- └ [edu.cmu.sphinx.frontend.BaseDataProcessor](#)
- └ **edu.cmu.sphinx.frontend.util.Microphone**

3.3 Package **edu.cmu.sphinx. recognizer**

Cung cấp một bộ các lớp và giao diện level cao được sử dụng để nhận dạng giọng nói với hệ thống nhận dạng giọng nói Sphinx-4.

Class chính : Recognizer, Recognizer.State.

```
public void recognizeDigits() {
    URL digitsConfig = new URL("file:./digits.xml");
    ConfigurationManager cm = new ConfigurationManager(digitsConfig);
    Recognizer sphinxDigitsRecognizer
        = (Recognizer) cm.lookup("digitsRecognizer");
    boolean done = false;
    Result result;
    sphinxDigitsRecognizer.allocate();
    // echo spoken digits, quit when 'nine' is spoken
    while (!done) {
        result = sphinxDigitsRecognizer.recognize();
        System.out.println("Result: " + result);
        done = result.toString().equals("nine");
    }
    sphinxDigitsRecognizer.deallocate();
}
```

public static [Recognizer.State](#) valueOf(java.lang.String name)

Constructor Summary

[Recognizer\(\)](#)

[Recognizer\(Decoder decoder, java.util.List<Monitor> monitors\)](#)

java.lang.Object

└ edu.cmu.sphinx.recognizer.Recognizer

3.4 Packages edu.cmu.sphinx.result

Class chính: Result, Lattice.

Cung cấp một tập các lớp tương ứng với kết quả của sự nhận dạng. Kết quả có thể là một dạng lưới, nó là một đồ thị có hướng của toàn bộ các từ có thể nhận dạng.

Class Summary

<u>Lattice</u>	Provides recognition lattice results.
<u>Node</u>	A node is part of Lattices, representing the theory that a word was spoken over a given period of time.
<u>PivotSausageMaker</u>	This is an implementation of an alternative sausage making algorithm as described in the following paper.
<u>Result</u>	Provides recognition results.
<u>Sausage</u>	A Sausage is a sequence of confusion sets, one for each position in an utterance.
<u>SimpleWordResult</u>	Represents a single word result with associated scoring and timing information.
<u>TokenGraphDumper</u>	Dumps out the GDL graph of all the result token chains in a Result, as well as all the alternate hypotheses along those chains.
<u>WordResultPath</u>	An implementation of a result Path that computes scores and confidences on the fly.

Constructor detail

Class Result

```
java.lang.Object
└ edu.cmu.sphinx.result.Result
```

Constructor Summary

```
Result(ActiveList activeList, java.util.List<Token> resultList,  
int frameNumber, boolean isFinal, LogMath logMath)
```

Creates a result

```
Result(AlternateHypothesisManager alternateHypothesisManager, ActiveList  
activeList, java.util.List<Token> resultList, int frameNumber,  
boolean isFinal, LogMath logMath)
```

Creates a result

Method Summary

java.util.List< Token >	findPartialMatchingTokens (java.lang.String text)
--------------------------------	--

Searches through the n-best list to find the the branch that matches the beginning of the given string

ActiveList	getActiveTokens ()
-------------------	---------------------------

Returns a list of active tokens for this result.

java.lang.String	getBestFinalResultNoFiller ()
------------------	--------------------------------------

Returns the string of the best final result, removing any filler words.

java.util.List< Data >	getDataFrames ()
-------------------------------	-------------------------

Gets the feature frames associated with this result

int	getFrameNumber ()
-----	--------------------------

Returns the current frame number

LogMath	getLogMath ()
----------------	----------------------

Returns the log math used for this Result.

int	getStartFrame ()
-----	-------------------------

	Gets the starting frame number for the result.
java.lang.String	<u>getTimedBestResult</u> (boolean wantFiller, boolean wordTokenFirst) Returns the string of words (with timestamp) for this token.
boolean	<u>isFinal</u> () Determines if the result is a final result.
(package private) void	<u>setFinal</u> (boolean finalResult) Sets the results as a final result
void	<u>setReferenceText</u> (java.lang.String ref) Sets the reference text
java.lang.String	<u>toString</u> () Returns a string representation of this object

```
public Result(AlternateHypothesisManager alternateHypothesisManager,
              ActiveList activeList,
              java.util.List<Token> resultList,
              int frameNumber,
              boolean isFinal,
              LogMath logMath)
```

- ❖ activeList: danh sách các hoạt động liên quan đến kết quả này.
- ❖ resultList: danh sách các kết quả liên quan đến kết quả này.
- ❖ frameNumber: số frame dùng cho result này.
- ❖ isFinal: nếu đúng kết quả là kết quả cuối cùng.

```
public Result(ActiveList activeList,
              java.util.List<Token> resultList,
              int frameNumber,
              boolean isFinal,
              LogMath logMath)
```

- ⊕ activeList: danh sách các hoạt động liên quan đến kết quả này.
 - ⊕ resultList: danh sách các kết quả liên quan đến kết quả này.
 - ⊕ frameNumber: số frame dùng cho result này.
 - ⊕ isFinal: nếu trả về true thì kết quả được giải mã thành công.
- java.lang.Object
- └ edu.cmu.sphinx.result.Result

Class Lattice

java.lang.Object

└ **edu.cmu.sphinx.result.Lattice**

public class **Lattice**

extends java.lang.Object

Cung cấp kết quả nhận dạng lưới lattice. Lưới được tạo từ [Results](#) cục bộ hoặc toàn cục

Lưới lattice đặc tả tất cả các giả thuyết được cân nhắc bởi Recognizer mà chưa được cắt tỉa, thu gọn. Lưới lattice là một đồ thị trực tiếp chứa các [Nodes](#) và [Edges](#). Một node (nút) ứng với một giả thuyết mà một từ được nói qua một khoảng thời gian đặc biệt. Một Edge (Cạnh) ứng với số điểm của mỗi

từ theo sau một từ khác. Một bản kết quả thường là một dãy các nút liên tục , mặc dù lưới lattice với điểm số tốt nhất. Lưới lattice là một công cụ hữu ích để phân tích “kết quả thay thế”.

Một lưới lattice có thể được tạo từ một Result mà có một cây biểu diễn đầy đủ(với AlternativeHypothesisManager - trình quản lý giả thuyết thay đổi luân phiên của nó) . Hiện nay, chỉ có lớp [WordPruningBreadthFirstSearchManager](#) có một AlternativeHypothesisManager. Ngoài ra, Code xây dựng ra lưới lattice hiện nay chỉ làm việc cho bộ ngôn ngữ linguits mà [WordSearchState](#) trả về false cho phương thức isWordStart, i.e., nơi mà các trạng thái của từ xuất hiện ở cuối mỗi từ trong bộ ngôn ngữ linguist.Tuy nhiên, lưới lattice nên chỉ được tạo từ kết quả Result từ lớp [LexTreeLinguist](#) và [WordPruningBreadthFirstSearchManager](#).

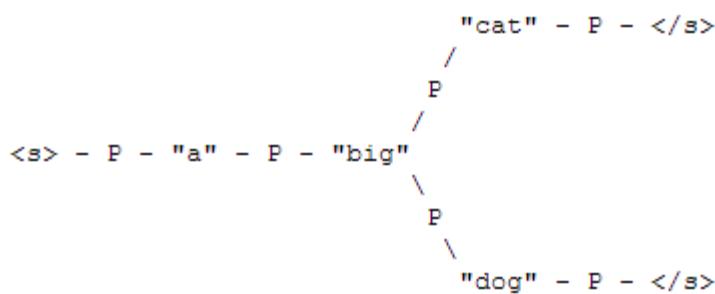
Lattice cũng có thể được tạo từ một cây [Token](#) dấu và trình quản lý thay đổi giả thuyết luân phiên – AlternativeHypothesisManager của nó. Đó chính là ý nghĩa của từ “ dấu – collapsed”. Thông thường, dấu hiệu (tokens) giữa hai từ là một chuỗi các dấu hiệu cho các kiểu trạng thái, như là một đơn vị unit , hoặc trạng thái HMM. Sử dụng W cho các dấu hiệu, U cho dấu hiệu đơn vị unit, H cho HMM, và một chuỗi các dấu hiệu có dạng :

W - U - H - H - H - U - H - H - H - H - W

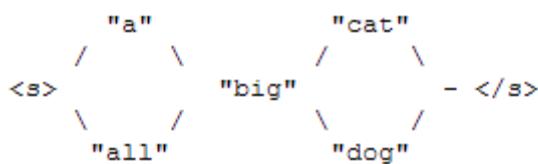
Thông thường, các dấu hiệu token HMM chứa các điểm âm thanh, và các dấu hiệu từ thì chứa các điểm ngôn ngữ. Nếu muốn biết tổng số điểm âm thanh và điểm ngôn ngữ giữa bất kỳ hai từ nào, không cần thiết phải giữ lại các phần quanh dấu hiệu token đơn vị và HMM. Tuy nhiên, tất cả điểm âm thanh và ngôn ngữ của chúng được dấu vào một token , vậy nên nó sẽ có dạng :

W - P – W

Chữ P là một token thể hiện đường đi giữa hai từ, và P cũng chứa điểm âm thanh và ngôn ngữ giữa hai từ. Đây là một loại cây token dấu mà lớp Lattice đòi hỏi. Bình thường thì nhiệm vụ dấu cây token được làm bởi [WordPruningBreadthFirstSearchManager](#). Một cây token dấu có dạng :



Khi một lưới Lattice được xây dựng từ một Result , cây token dấu bên trên ghép với giả thuyết thay đổi luân phiên của “tất cả” thay vì “một” , sẽ được chuyển thành một lưới Lattice như sau :



Ban đầu, một lưới lattice có thể có các nốt dư, các nốt nối với cùng một từ và bắt đầu từ cùng một nốt cha. Những nốt này có thể được dẹp đi bằng cách sử dụng lớp [LatticeOptimizer](#)

3.5 Packages edu.cmu.sphinx.util

Đây là package cung cấp cơ chế để quản lý cấu hình dữ liệu ổn định.

Interface Summary

<u>Configurable</u>	Defines the interface that must be implemented by any configurable component in Sphinx-4.
<u>ConfigurationChangeListener</u>	Describes all methods necessary to process change events of a ConfigurationManager.

Class Summary

<u>ConfigHandler</u>	A SAX XML Handler implementation that builds up the map of raw property data objects
<u>ConfigurableAdapter</u>	An default (abstract) implementation of a configurable that implements a meaningful <code>toString()</code> and keeps a references to the Configurable's logger.
<u>ConfigurationManager</u>	Manages a set of Configurables, their parameterization and the relationships between them.
<u>ConfigurationManagerUtils</u>	Some static utility methods which ease the handling of system configurations.

<u>PropertySheet</u>	A property sheet which defines a collection of properties for a single component in the system.
<u>RawPropertyData</u>	Holds the raw property data just as it has come in from the properties file.
<u>S4PropWrapper</u>	Wraps annotations
<u>SaxLoader</u>	Loads configuration from an XML file

Cung cấp một cơ chế để quản lý dữ liệu ổn định. Quản lý cấu hình và thiết lập liên kết các giao diện, các lớp cung cấp các services sau:

- ⊕ Tải cấu hình dữ liệu từ một file cấu hình XML cơ bản.
- ⊕ Điều chỉnh life-cycle cho cấu hình các đối tượng.
- ⊕ Cho phép phát hiện các thành phần thông qua tên hoặc kiểu.

Các class quan trọng : [ConfigurationManager](#)

java.lang.Object

└ [edu.cmu.sphinx.util.props.ConfigurationManager](#)

All Implemented Interfaces:

java.lang.Cloneable

Đây là class quản lý cấu hình , các tham số giá trị và quan hệ giữa chúng. Cấu hình có thể được đặc tả bằng file xml trong suốt quá trình chạy.

Chi tiết cách cấu hình , chúng em xin trình bày ở các chương sau.

3.6 Packages [edu.cmu.sphinx.linguist.language.grammar](#)

Class chính: TextAlignerGrammar.

Cung cấp các lớp và các giao diện để có thể được sử dụng cho một đồ thị của các từ và sự chuyển đổi từ.

Class Summary

<u>FSTGrammar</u>	Loads a grammar from a file representing a finite-state transducer (FST) in the 'ARPA' grammar format.
<u>Grammar</u>	Classes that implement this interface create grammars.
<u>LatticeGrammar</u>	A grammar build from a lattice.
<u>LMGrammar</u>	Defines a simple grammar based upon a language model.
<u>SimpleWordListGrammar</u>	Defines a grammar based upon a list of words in a file.
<u>TextAlignerGrammar</u>	Creates a grammar from a text.

Constructor Summary

[TextAlignerGrammar \(\)](#)

[TextAlignerGrammar](#)(java.lang.String text, [LogMath](#) logMath, boolean showGrammar, boolean optimizeGrammar, boolean addSilenceWords, boolean addFillerWords, [Dictionary](#) dictionary)

Main Method Summary

<code>protected GrammarNode</code>	<code><u>createGrammar()</u></code> Create a branch of the grammar that corresponds to a transcript.
<code>void</code>	<code><u>newProperties(PropertySheet ps)</u></code> This method is called when this configurable component needs to be reconfigured.
<code>void</code>	<code><u>newResult(Result result)</u></code> Method called when a new result is generated
<code>void</code>	<code><u>setText(java.lang.String text)</u></code>

java.lang.Object

└ [edu.cmu.sphinx.linguist.language.grammar.Grammar](#)

 └ edu.cmu.sphinx.linguist.language.grammar.TextAlignerGrammar

3.7 Packages `edu.cmu.sphinx.jsgf`

Class:JSGFGrammar, JSGFGrammarException,

JSGFGrammarParseException.

Định nghĩa một kiểu ngữ pháp BNF trên quy định của JSGF trong một file.

The Java Speech Grammar Format (JSGF) là một kiểu BNF, nền tảng độc lập và nhả cung cấp tự động là miêu tả gốc của những ngữ pháp được sử dụng trong nhận dạng giọng nói. Nó được sử dụng bằng Java Speech API(JSAPI), ở đây, chỉ đưa ra một vài ví dụ về ngữ pháp được viết bằng JSGF, có thể qua đây viết được ngữ pháp của riêng mình.

Class Summary

JSGFGrammar	Defines a BNF-style grammar based on JSGF grammar rules in a file.
JSGFRuleGrammar	
JSGFRuleGrammarFactory	
JSGFRuleGrammarManager	

Exception Summary

JSGFGrammarException	
JSGFGrammarParseException	

java.lang.Object

 └ [edu.cmu.sphinx.linguist.language.grammar.Grammar](#)

 └ [**edu.cmu.sphinx.jsgf.JSGFGrammar**](#)

java.lang.Object

 └ java.lang.Throwable

 └ java.lang.Exception

 └ [**edu.cmu.sphinx.jsgf.JSGFGrammarException**](#)

java.lang.Object

 └ java.lang.Throwable

 └ java.lang.Exception

 └ [**edu.cmu.sphinx.jsgf.JSGFGrammarParseException**](#)

Ví dụ: “HelloWorld” trong JSGF trong file có tên là helloworld.gram

```
#JSGF V1.0;
/*
 * JSGF Grammar for Hello World example
 */
grammar hello;
public <greet> = (Hi | Hello) ( Bhiksha | Evandro | Paul | Philip | Rita | Will );
```

Quy tắc trên được hiện thị theo kiểu public.

Ví dụ tiếp theo: ví dụ này phức tạp hơn, được gọi bởi “BasicCmd”.

```
#JSGF V1.0
public <basicCmd> = <startPolite> <command> <endPolite>;
<command> = <action> <object>;
<action> = /10/ open |/2/ close |/1/ delete |/1/ move;
<object> = [the | a] (window | file | menu);
<startPolite> = (please | kindly | could you | oh mighty computer) *;
<endPolite> = [ please | thanks | thank you ];
```

Hình 4: Lệnh ngữ pháp tạo các điều khiển đơn giản.

Những tính năng của JSGF được đưa ra trong ví dụ trên gồm:

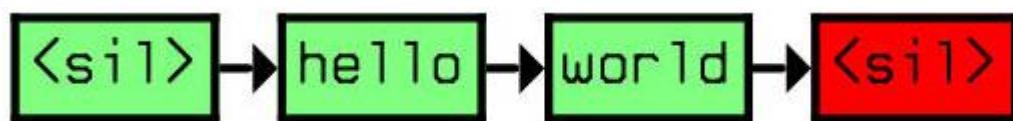
-  Sử dụng các luật ngữ pháp khác trong một quy tắc ngữ pháp.
-  Toán tử OR “|”
-  Toán tử nhóm “(...)”
-  Toán tử nhóm tùy chọn “[...]"
-  Toán tử không hoặc nhiều “*”



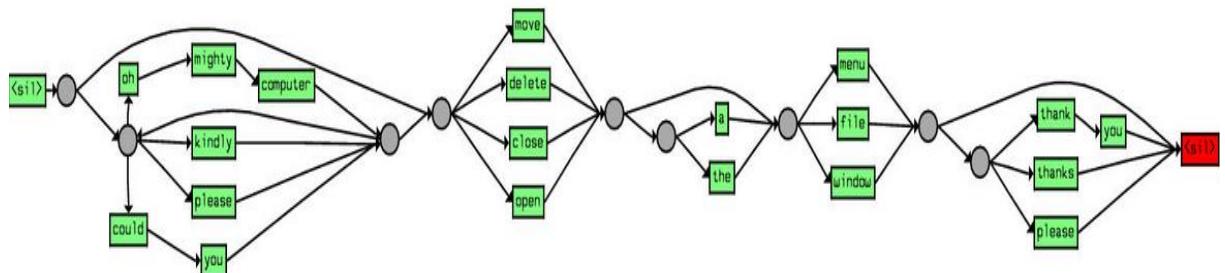
Xác xuất (ví dụ: “open” có khả năng hơn những cái khác(close, delete...)).

Từ JSGF cho ra đồ thị ngữ pháp.

Sau khi JSGF được đọc, nó được chuyển thành đồ thị các từ thay thế cho ngữ pháp. Gọi đây là đồ thị ngữ pháp (grammar graph). Từ biểu đồ này, tìm kiếm cấu trúc cuối cùng được sử dụng để xây dựng nhận dạng giọng nói.hình dưới này là grammar graph được tạo ra từ JSGF grammar. Nút “<sil>” nghĩa là “sự im lặng/sự ngắt quãng”.



Hình 5: Grammar graph created from the Hello World grammar.



Hình 6: Grammar graph created from the Command grammar.

Về sự hỗ trợ của JSGF hiện tại, có một hạn chế. Những vòng lặp không có âm thanh hiện thời là nguyên nhân làm cho bộ nhận dạng bị treo(đứng).

Ví dụ: trong grammar sau.

```
#JSGF V1.0
```

```
grammar jsgf.nastygram;
public <nasty> = I saw a ((cat* | dog* | mouse*)+)+;
```

((cat* | dog* | mouse*)+)+ kết quả có thể mở rộng trong một vòng lặp liên tục, từ lúc kết quả (cat* | dog* | mouse*) ko có (nghĩa là không có cat,dog,mouse), nó sẽ là ()+. Để tránh lỗi này, việc viết ngữ pháp cần đảm bảo là không có quy tắc nào là có thể không khớp với lời nói trong một toán tử thêm vào hoặc toán tử dấu “*” kleene star.

Tác động thay đổi ngữ pháp(Dynamic grammar behavior). Nó có thể thay đổi grammar của một ứng dụng đang chạy. Một số quy tắc và ghi chú:

- Không giống như một bộ nhận dạng JSAPI, JSGF chỉ duy trì một JSGF grammar. Sự hạn chế này có thể được cải thiện hơn tương lai.
- Grammar không nên bị thay đổi khi đang trong tiến trình nhận dạng.
- Gọi JSGFGrammar.loadJSGF, sẽ load một grammar mới hoàn toàn, đưa lên những ngữ pháp cũ hoặc thay đổi. không cần thiết gọi CommitChange, mặc dù gọi cũng không ảnh hưởng gì.
- RuleGrammars có thể được thay đổi thông qua việc gọi RuleGrammar.setEnabled và RuleGrammar.setRule. Để thay đổi được, JSGFGrammar.commicChanges phải được gọi sau những thay đổi đã làm.

3.8 Package com.sun.speech.engine.recognition

Class: BaseRecognizer, BaseRuleGrammar.

BaseRecognizer: sườn thực hiện của giao diện nhận dạng JSAPI.

BaseRuleGrammar: khung sườn của javax.speech.recognition.RuleGrammar.

Class Summary

<u>BaseGrammar</u>	Implementation of javax.speech.recognition.Grammar.
<u>BaseRecognizer</u>	Skeletal Implementation of the JSAPI Recognizer interface.
<u>BaseRecognizerAudioManager</u>	Skeletal Implementation of the JSAPI AudioManager interface for Recognizers.
<u>BaseResult</u>	
<u>BaseRuleGrammar</u>	Implementation of javax.speech.recognition.RuleGrammar.
<u>RecognizerUtilities</u>	Utilities methods.
<u>RuleParser</u>	Implementation of the parse method(s) on javax.speech.recognition.RuleGrammar.
<u>RuleState</u>	

java.lang.Object

 └ [com.sun.speech.engine.BaseEngine](#)

 └ [com.sun.speech.engine.recognition.BaseRecognizer](#)

```

public void setRule(java.lang.String ruleName,
                    Rule rule,
                    boolean isPublic)
throws java.lang.NullPointerException,
java.lang.IllegalArgumentException

```

Set a rule in the grammar either by creating a new rule or updating an existing rule.

Specified by:

[setRule](#) in interface [RuleGrammar](#)

Parameters:

ruleName - the name of the rule.
rule - the definition of the rule.
isPublic - whether this rule is public or not.

Throws:

java.lang.NullPointerException
java.lang.IllegalArgumentException

java.lang.Object

└ [com.sun.speech.engine.recognition.BaseGrammar](#)

 └ [com.sun.speech.engine.recognition.BaseRuleGrammar](#)

3.9 Packages edu.cmu.sphinx.jsapi

Class chính: SphinxRecognizer, Sphinx4Result.

Là những hỗ trợ cho Java Speech API trong sphinx-4.

Class Summary	
RecognitionThread	Recognition thread to run the recognizer in parallel.
Sphinx4Result	JSAPI compliant recognition result.
Sphinx4ResultListener	Result listener for results from the sphinx recognizer.
SphinxEngineCentral	Provides a list of SphinxRecognizer.ModeDesc objects that define the available operating modes of the sphinx recognition engine.
SphinxRecognizer	A SphinxRecognizer provides access to Sphinx speech recognition capabilities.
SphinxRecognizerModeDesc	Provides information about a specific operating mode of a sphinx recognition engine.

java.lang.Object

└ [com.sun.speech.engine.BaseEngine](#)

└ [com.sun.speech.engine.recognition.BaseRecognizer](#)
 └ [edu.cmu.sphinx.jsapi.SphinxRecognizer](#)

loadJSGF

```
public RuleGrammar loadJSGF(java.io.Reader reader)
    throws GrammarException,
           java.io.IOException,
           EngineStateError
```

Reader: đọc nội dung JSGF nhập vào.

Specified by:

[loadJSGF](#) in interface [Recognizer](#)

Overrides:

[loadJSGF](#) in class [BaseRecognizer](#)

Parameters:

`reader` - the Reader containing JSGF input.

Throws:

[GrammarException](#)
[java.io.IOException](#)
[EngineStateError](#)

CHƯƠNG 4: CÀI ĐẶT VÀ CẤU HÌNH

4.1 Cài đặt Sphinx

4.1.1 Chuẩn bị hệ điều hành

Linux là môi trường hệ điều hành thích hợp nhất để cài đặt Sphinx và thực hiện huấn luyện. Trong các hệ điều hành Linux thì Ubuntu được xem như là hệ điều hành phổ biến. Sphinx dễ dàng được cài đặt trong hệ điều hành Ubuntu, nhóm từng thử cài đặt trong Windows dùng CYGWIN [10] nhưng chưa thành công.

4.1.2 Chuẩn bị các gói cài đặt Sphinx

Các gói bao gồm:

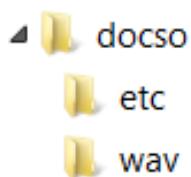
- *Pocketsphinx* —một thư viện nhận dạng viết bằng ngôn ngữ C.
- *Sphinxbase* — gói thư viện nền, hỗ trợ các thư viện cần thiết cho các gói khác
- *Sphinx4* — gói hỗ trợ nhận dạng viết bằng java
- *CMUclmtk* — bộ công cụ xây dựng mô hình ngôn ngữ
- *Sphinxtrain* — bộ công cụ huấn luyện mô hình ngôn ngữ âm

Các gói cài đặt có thể được tải trực tiếp từ trang chủ của CMU Sphinx [11]

4.1.3 Cài đặt Sphinx

➤ Trong window

1. Tạo thư mục, tên là "docso" chứa dữ liệu huấn luyện.



2. Tạo hai thư mục con trong "docso" là: "etc", "wav"

3. Bên trong “etc” , tạo file “docso.txt” – chứa các câu nói sẽ thu âm. Mỗi câu một dòng.

<s> không một hai ba bốn năm sáu bảy tám chín </s>

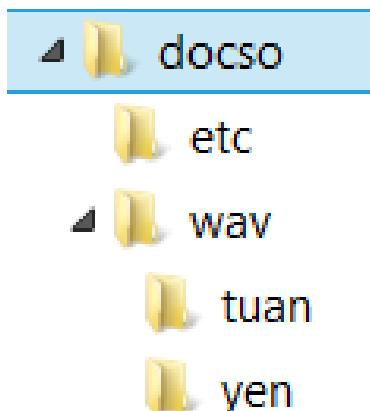
Lưu ý: đối với bộ từ điển càng lớn thì càng cần nhiều giờ thu , cụ thể như bản sau:

Vocabulary	Hours in db	Senones	Densities	Example
20	5	200	8	Tidigits Digits Recognition
100	20	2000	8	RM1 Command and Control
5000	30	4000	16	WSJ1 5k Small Dictation
20000	80	4000	32	WSJ1 20k Big Dictation
60000	200	6000	16	HUB4 Broadcast News
60000	2000	12000	64	Fisher Rich Telephone Transcription

Ở đây chúng tôi huấn luyện đọc số từ không đến chín (0 … 9), vậy nên cần tắt cả khoảng 3 giờ thu âm.

4. Ta có thể tiến hành thu âm trước.

Bên trong thư mục “wav”, ta tạo hai thư mục con để lưu trữ file thu âm của hai người.



Tải và cài đặt phần mềm thu âm , ở đây chúng tôi dùng “audacity” là một phần mềm thu âm miễn phí khá tốt.

Để thu âm tốt, chúng tôi khuyến khích cấu hình theo voxforge.

<http://www.voxforge.org/home/submitspeech/windows/step-2>

5. Trong Window, copy file docso.txt , để vào bên trong thư mục **cmuclmtk-0.7-win32**

Vào cmd gõ theo hình sau để tạo file từ vựng.

```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\YenHo>cd C:\Users\YenHo\Desktop\cmusphinx\cmuclmtk-0.7-win32

C:\Users\YenHo\Desktop\cmusphinx\cmuclmtk-0.7-win32>text2wfreq < docso.txt | wfreq2vocab > docso.vocab
text2wfreq : Reading text from standard input...
wfreq2vocab : Will generate a vocabulary containing the most
               frequent 20000 words. Reading wfreq stream from stdin...
text2wfreq : Done.
wfreq2vocab : Done.

C:\Users\YenHo\Desktop\cmusphinx\cmuclmtk-0.7-win32>
```

text2wfreq < docso.txt | wfreq2vocab > docso.vocab

sau khi xong, bên trong thư mục cmuclmtk-0.7-win32 , sẽ sinh ra 1 file “docso.vocab” (nên mở ra để kiểm tra lỗi chính tả)

6. Tạo mô hình ngôn ngữ :

text2idngram -vocab docso.vocab -idngram docso.idngram <
docso.txt

Sau khi chạy xong, máy sẽ sinh ra file “docso.idngram”

```

Command Prompt
C:\Users\YenHo\Desktop\cmusphinx\cmuclmtk-0.7-win32>text2idngram -vocab docso.vocab
cab -idngram docso.idngram < docso.txt
text2idngram
Vocab : docso.vocab
Output idngram : docso.idngram
N-gram buffer size : 100
Hash table size : 2000000
Temp directory : cmuclmtk-a27716
Max open files : 20
EOF size : 10
n : 3
Initialising hash table...
Reading vocabulary...
Allocating memory for the n-gram buffer...
Reading text into the n-gram buffer...
20,000 n-grams processed for each ".", 1,000,000 for each line.

Sorting n-grams...
Writing sorted n-grams to temporary file cmuclmtk-a27716/1
Merging 1 temporary files...

2-grams occurring:      N times      > N times      Sug. -spec_num value
  0                      96              106
  1                      80              90
  2                      14              66
  3                      5               61
  4                      4               57
  5                      1               56
  6                      1               55
  7                      0               55
  8                      0               55
  9                      1               54
  10                     9               45

3-grams occurring:      N times      > N times      Sug. -spec_num value
  0                      204             216
  1                      102             113
  2                      17              95
  3                      1               94
  4                      0               94
  5                      0               94
  6                      0               94
  7                      0               94
  8                      0               94
  9                      1               93
  10                     38              55

text2idngram : Done.

C:\Users\YenHo\Desktop\cmusphinx\cmuclmtk-0.7-win32>

```

idngram2lm -vocab_type 0 -idngram docso.idngram -vocab docso.vocab -
arpa docso.arpa

```
idngram2lm -vocab_type 0 -idngram docso.idngram -vocab docso.vocab -
arpa docso.arpa
```

```

64 Command Prompt
Discounting ranges :
 1-gram : 1      2-gram : 7      3-gram : 7
Memory allocation for tree structure :
  Allocate 100 MB of memory, shared equally between all n-gram tables.
Back-off weight storage :
  Back-off weights will be stored in four bytes.
Reading vocabulary.

read_wlist_into_siht: a list of 12 words was read from "docso.vocab".
read_wlist_into_array: a list of 12 words was read from "docso.vocab".
WARNING: <s> appears as a vocabulary item, but is not labelled as a
context cue.
Allocated space for 3571428 2-grams.
Allocated space for 8333333 3-grams.
table_size 13
Allocated 57142848 bytes to table for 2-grams.
Allocated (2+3333333) bytes to table for 3-grams.
Processing id n-gram file.
20,000 n-grams processed for each "...", 1,000,000 for each line.

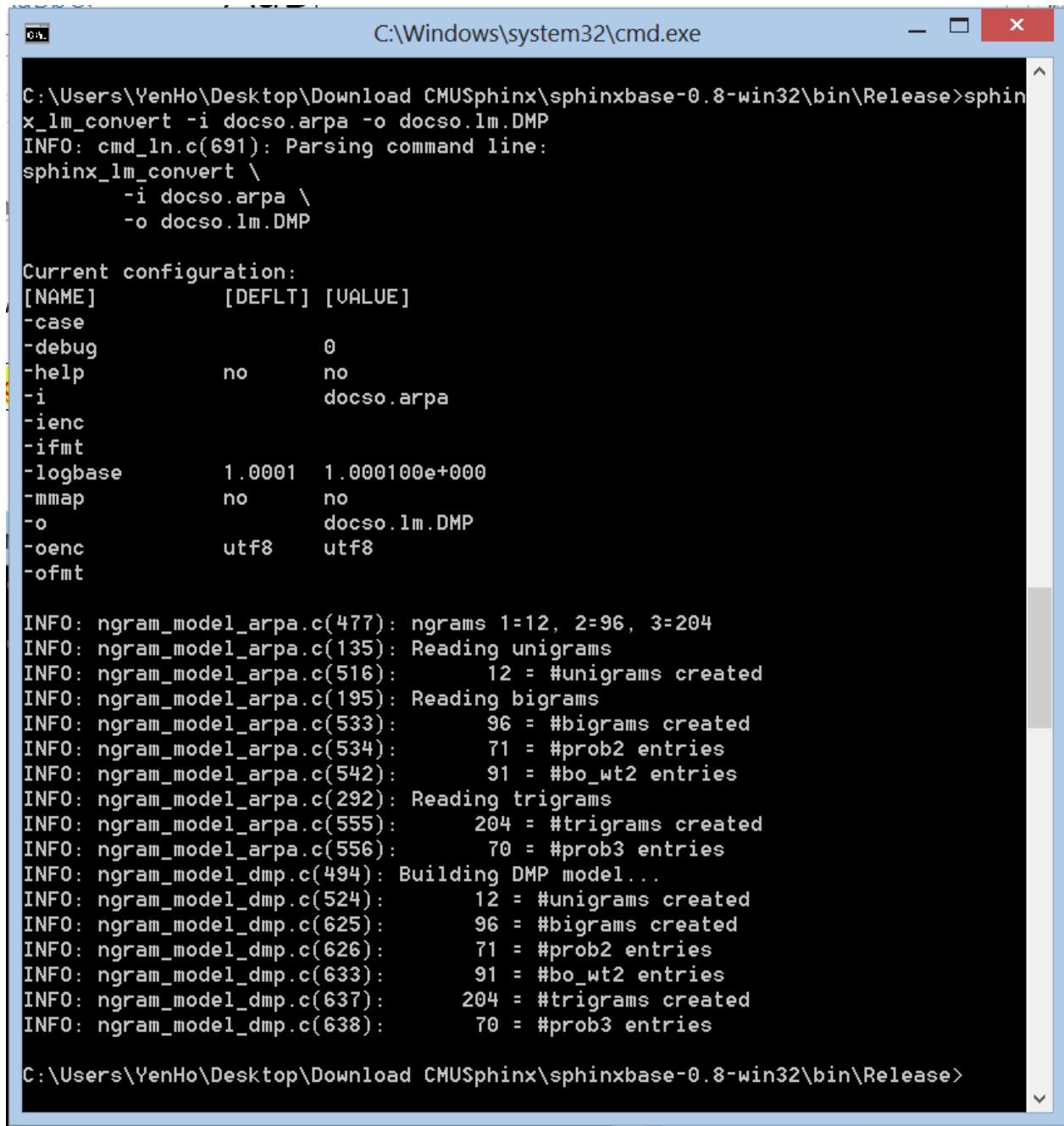
Calculating discounted counts.
Warning : 1-gram : f-of-f[1] = 0 --> 1-gram discounting is disabled.
Warning : 2-gram : GT statistics are out of range; lowering cutoff to 6.
Warning : 2-gram : GT statistics are out of range; lowering cutoff to 5.
Warning : 2-gram : Some discount values are out of range;
lowering discounting range to 4.
Warning : 2-gram : Some discount values are out of range;
lowering discounting range to 3.
Warning : 2-gram : GT statistics are out of range; lowering cutoff to 2.
Warning : 2-gram : Some discount values are out of range;
lowering discounting range to 1.
Warning : 2-gram : Discounting range of 1 is equivalent to excluding
singletons.
Warning : 2-gram : GT statistics are out of range; lowering cutoff to 0.
Warning : 2-gram : Discounting is disabled.
Warning : 3-gram : GT statistics are out of range; lowering cutoff to 6.
Warning : 3-gram : GT statistics are out of range; lowering cutoff to 5.
Warning : 3-gram : GT statistics are out of range; lowering cutoff to 4.
Warning : 3-gram : GT statistics are out of range; lowering cutoff to 3.
Warning : 3-gram : GT statistics are out of range; lowering cutoff to 2.
Unigrams's discount mass is 0 (n1/N = 0)
prob[UNK] = 1e-099
Incrementing contexts...
Calculating back-off weights...
Writing out language model...
ARPA-style 3-gram will be written to docso.arpa
idngram2lm : Done.

C:\Users\YenHo\Desktop\cmusphinx\cmuclmtk-0.7-win32>
```

Sau khi xong câu lệnh này, sẽ sinh ra file “docso.arpa”

7. Tạo mã nhị phân CMU : di chuyển đường dẫn qua **sphinxbase-0.8-win32\bin\Release**.

Copy file docso.arpa từ cmuclmtk-0.7-win32 qua **sphinxbase-0.8-win32\bin\Release**.



```
C:\Windows\system32\cmd.exe
C:\Users\YenHo\Desktop\Download CMUSphinx\sphinxbase-0.8-win32\bin\Release>sphinx_lm_convert -i docso.arpa -o docso.lm.DMP
INFO: cmd_ln.c(691): Parsing command line:
sphinx_lm_convert \
    -i docso.arpa \
    -o docso.lm.DMP

Current configuration:
[NAME]      [DEFLT] [VALUE]
-case
-debug          0
-help        no     no
-i             docso.arpa
-ienc
-ifmt
-logbase      1.0001  1.000100e+000
-mmap        no     no
-o             docso.lm.DMP
-oenc        utf8   utf8
-ofmt

INFO: ngram_model_arpa.c(477): ngrams 1=12, 2=96, 3=204
INFO: ngram_model_arpa.c(135): Reading unigrams
INFO: ngram_model_arpa.c(516):      12 = #unigrams created
INFO: ngram_model_arpa.c(195): Reading bigrams
INFO: ngram_model_arpa.c(533):      96 = #bigrams created
INFO: ngram_model_arpa.c(534):      71 = #prob2 entries
INFO: ngram_model_arpa.c(542):      91 = #bo_wt2 entries
INFO: ngram_model_arpa.c(292): Reading trigrams
INFO: ngram_model_arpa.c(555):      204 = #trigrams created
INFO: ngram_model_arpa.c(556):      70 = #prob3 entries
INFO: ngram_model_dmp.c(494): Building DMP model...
INFO: ngram_model_dmp.c(524):      12 = #unigrams created
INFO: ngram_model_dmp.c(625):      96 = #bigrams created
INFO: ngram_model_dmp.c(626):      71 = #prob2 entries
INFO: ngram_model_dmp.c(633):      91 = #bo_wt2 entries
INFO: ngram_model_dmp.c(637):      204 = #trigrams created
INFO: ngram_model_dmp.c(638):      70 = #prob3 entries

C:\Users\YenHo\Desktop\Download CMUSphinx\sphinxbase-0.8-win32\bin\Release>
```

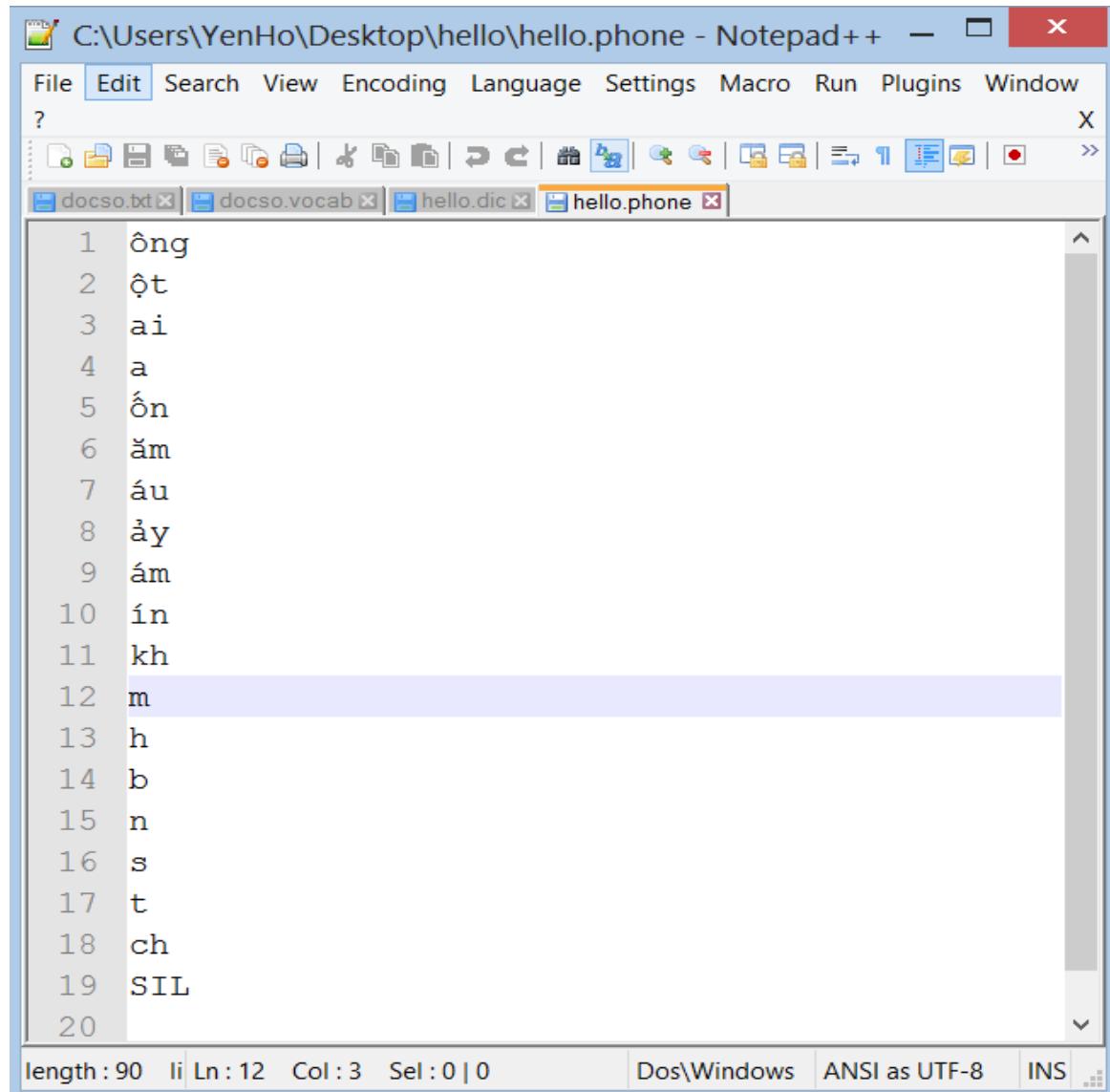
sphinx_lm_convert -i docso.arpa -o docso.lm.DMP

File “docso.lm.DMP” được sinh ra trong thư mục của sphinxbase.

Vậy là xong quá trình đầu tiên

Tiếp đến, tạo từ điển (bằng tay)

File docso.phone



The screenshot shows the Notepad++ application window with the title bar "C:\Users\YenHo\Desktop\hello\hello.phone - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, and Window. The toolbar below has various icons for file operations. The main editor area displays a list of phonetic transcriptions from line 1 to 20. The word 'm' is currently selected, highlighted with a light purple background. The status bar at the bottom shows "length : 90" and "Ln : 12".

Số	Từ
1	ông
2	ột
3	ai
4	a
5	ôn
6	ăm
7	áu
8	ảy
9	ám
10	ín
11	kh
12	m
13	h
14	b
15	n
16	s
17	t
18	ch
19	SIL
20	

Và file docso.dic

```

C:\Users\YenHo\Desktop\docso\etc\docso.dic - Notepad++ - X
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
X
docso.txt docso.vocab docso.dic
1 không kh ông
2 một m ôt
3 hai h ai
4 ba b a
5 bốn b ón
6 năm n ăm
7 sáu s áu
8 bảy b ảy
9 tám t ám
10 chín ch ín
length : 125 line Ln : 1 Col : 1 Sel : 0 | 0 Dos\Windows ANSI as UTF-8 INS

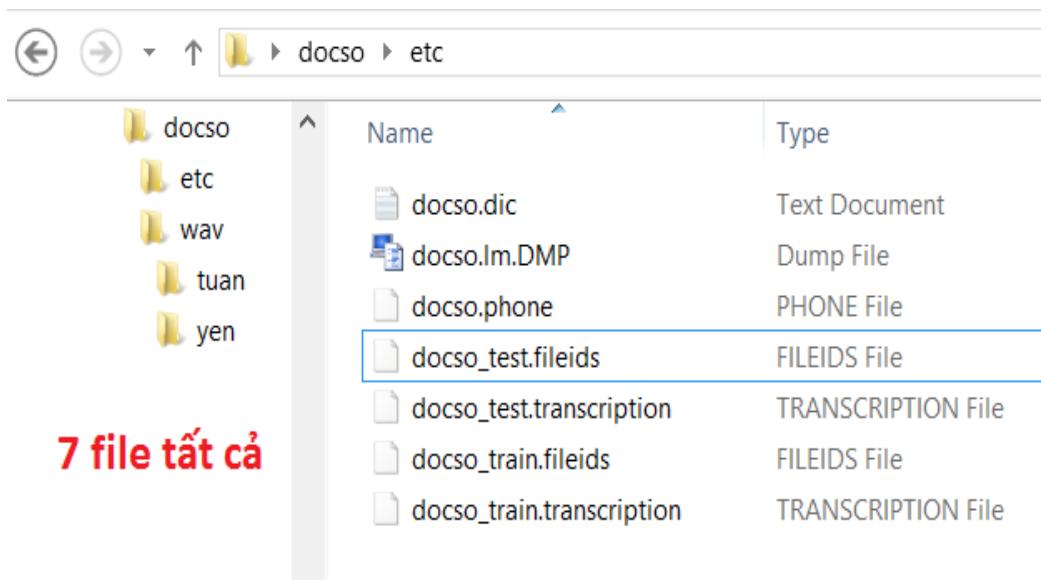
```

Tiếp đến ta xây dựng 4 file :

- docso_train.fileids
- docso _test.fileids
- docso _train.transcription
- docso _test.transcription

Như vậy ta đã xong quá trình chuẩn bị nguyên liệu.

Tổng kết lại, ta có cây thư mục như sau:



Tạo file filler gồm :

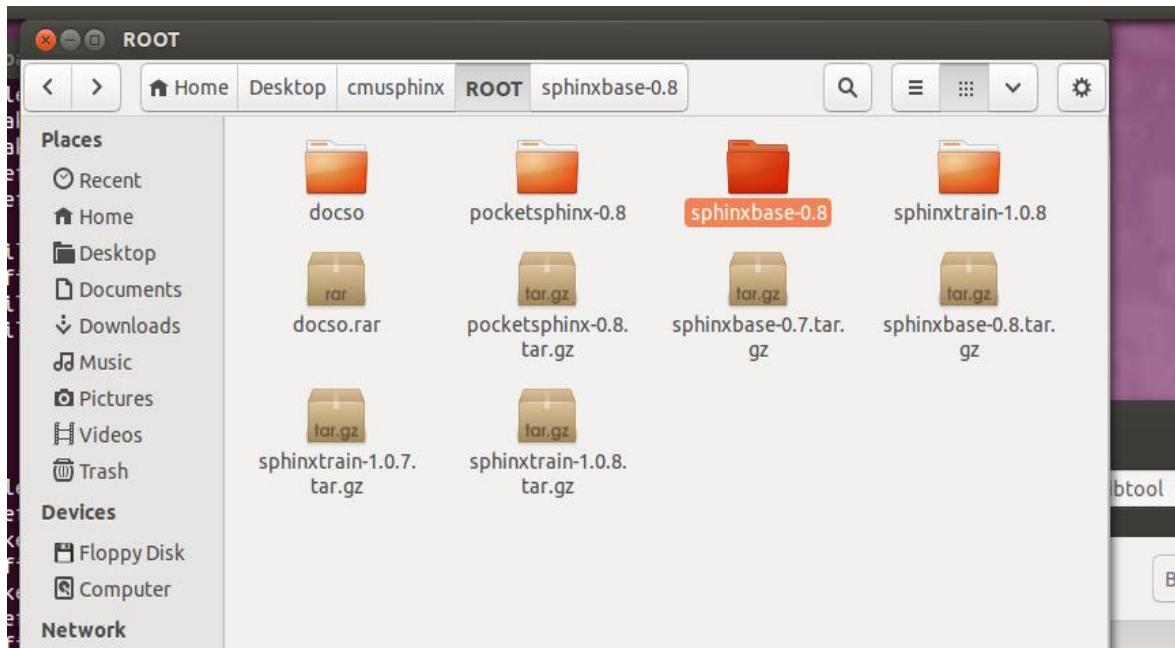
<S> SIL

</s> SIL

<Sil> SIL

➤ Trong ubuntu

Tạo một thư mục tên *sphinx* trong Home folder (trong máy ảo Ubuntu). Chép các file (*Sphinxbase*, *Sphinxtrain*, *Pocketsphinx*, *CMUclmtk*) vừa download trong mục trên vào đó, giải nén. (lưu ý xóa đi chỉ số version sau khi extract).



Mở Ubuntu Software center lên. Search và cài các thứ sau :

- libtool
- bison
- autoconf

ngoài ra nếu trong quá trình phía sau có bắt cứ lỗi gì dạng “ You need to install ...” thì cứ lên search cài vào.

Sử dụng cửa sổ Terminal trong Ubuntu: **Ctrl+Alt+t**.

Nhập vào **sudo apt-get update** sau đó nhập vào password của root user (password sẽ không hiện lên, nhập cẩn thận và nhấn Enter). Lệnh trên để update chẩn bị cho các gói cài đặt cần dùng bằng lệnh **apt-get**. Chờ update xong.

Ta tiến hành cài đặt theo thứ tự như sau:

Cài đặt python:

Mở Terminal lên và gõ vào dòng lệnh

```
sudo apt-get install python
```

Cài đặt Perl:

Mở Terminal lên và gõ vào dòng lệnh

```
sudo apt-get install perl
```

nhập vào: **cd sphinx** để di chuyển tới thư mục *sphinx* vừa tạo.

Cài đặt các gói cần thiết trước khi cài SphinxBase:

Gõ các lệnh:

- **sudo apt-get install bison**, đồng ý để tải và cài bison
- **sudo apt-get install autoconf**
- **sudo apt-get install automake**
- **sudo apt-get install libtool**

Lưu ý : nếu ko gõ sudo sẽ bị lỗi.

a. Cài đặt SphinxBase

Nhập: **cd sphinxbase** để đi vào thư mục *sphinxbase*.

Gõ các lệnh sau và chờ thi hành:

- **./autogen.sh**
- **./configure**
- **make**
- **sudo make install**

b. Cài đặt Sphinxtrain

Từ thư mục *sphinxbase* ở trên, gõ lệnh để chuyển sang thư mục sang thư mục *sphinxtrain* : **cd ..//sphinxtrain**. Gõ các lệnh sau và chờ thi hành:

- **./configure**
- **make**
- **sudo make install**

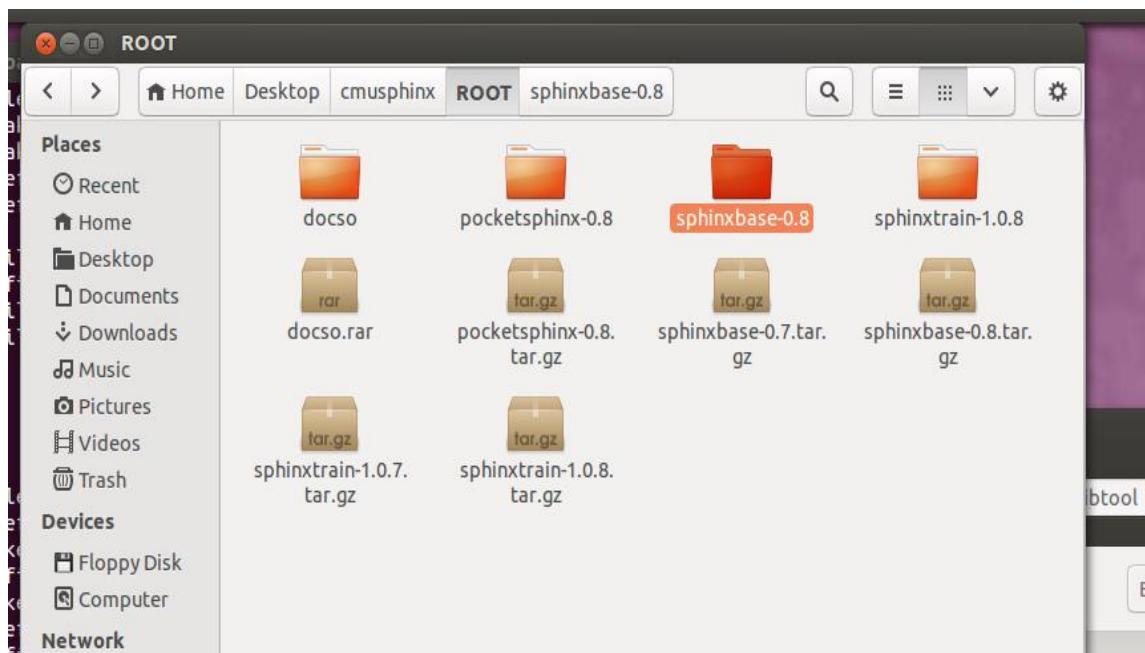
c. Cài đặt PocketSphinx

Chuyển sang thư mục pocketsphinx, gõ các lệnh sau và chờ thi hành:

- **./autogen.sh**
- **./configure**
- **make**
- **sudo make install**

gõ tiếp lệnh sau vào Terminal: **sudo ldconfig** để hệ điều hành thực hiện cập nhật các thực viễn động.

Chi tiết quá trình cài đặt có thể tham khảo theo nguồn



Hình 0.1 Thư mục "sphinx" chứa các file vừa tải và các thư mục sau khi đã giải nén, đổi tên
Vậy là xong phần cài đặt sphinxbase và sphinxtrain. Tiếp đến bạn chạy các dòng lệnh sau

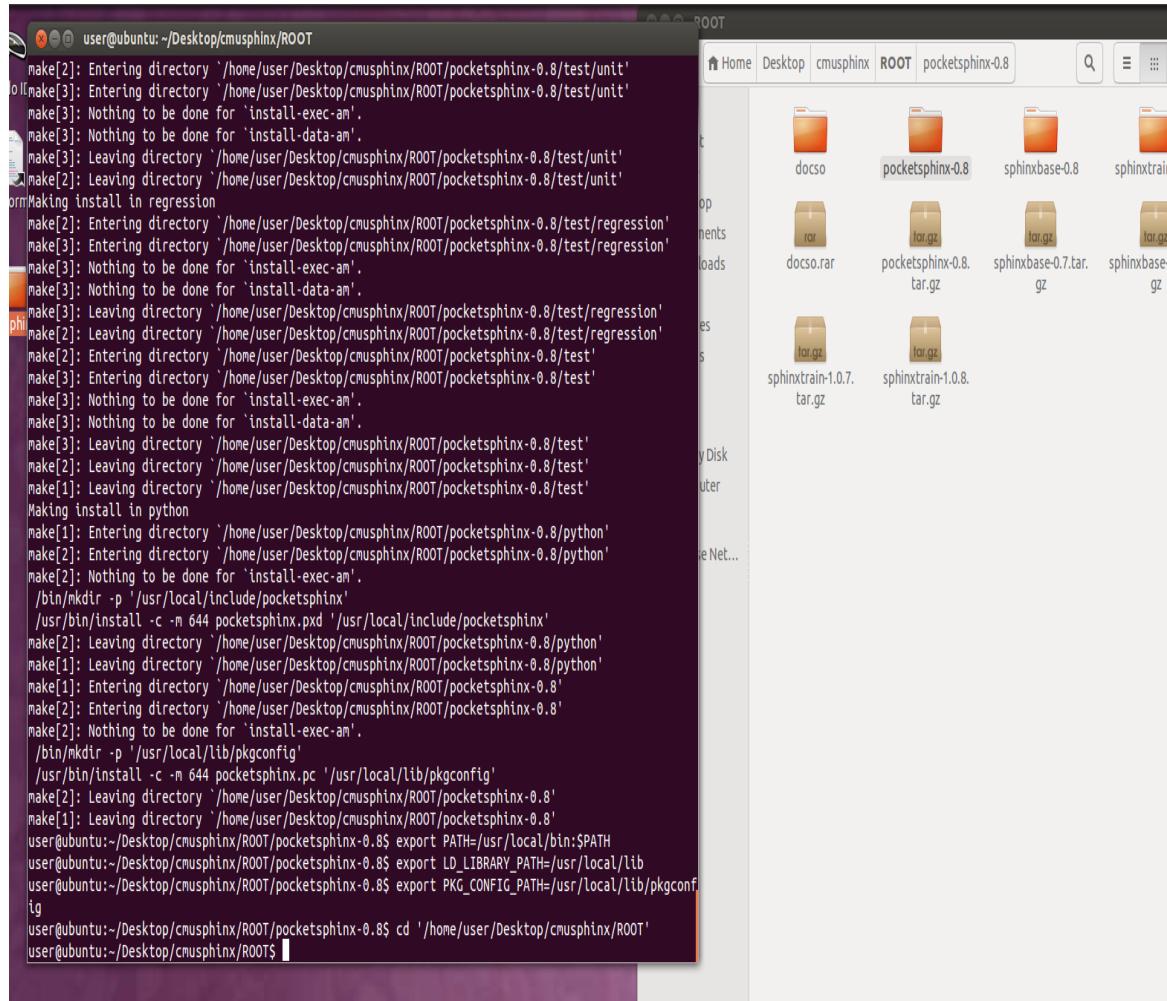
để thực hiện việc cập nhật các thư viện mới được cài đặt sẽ được sử dụng:

```
export PATH=/usr/local/bin:$PATH
```

```
export LD_LIBRARY_PATH=/usr/local/lib
```

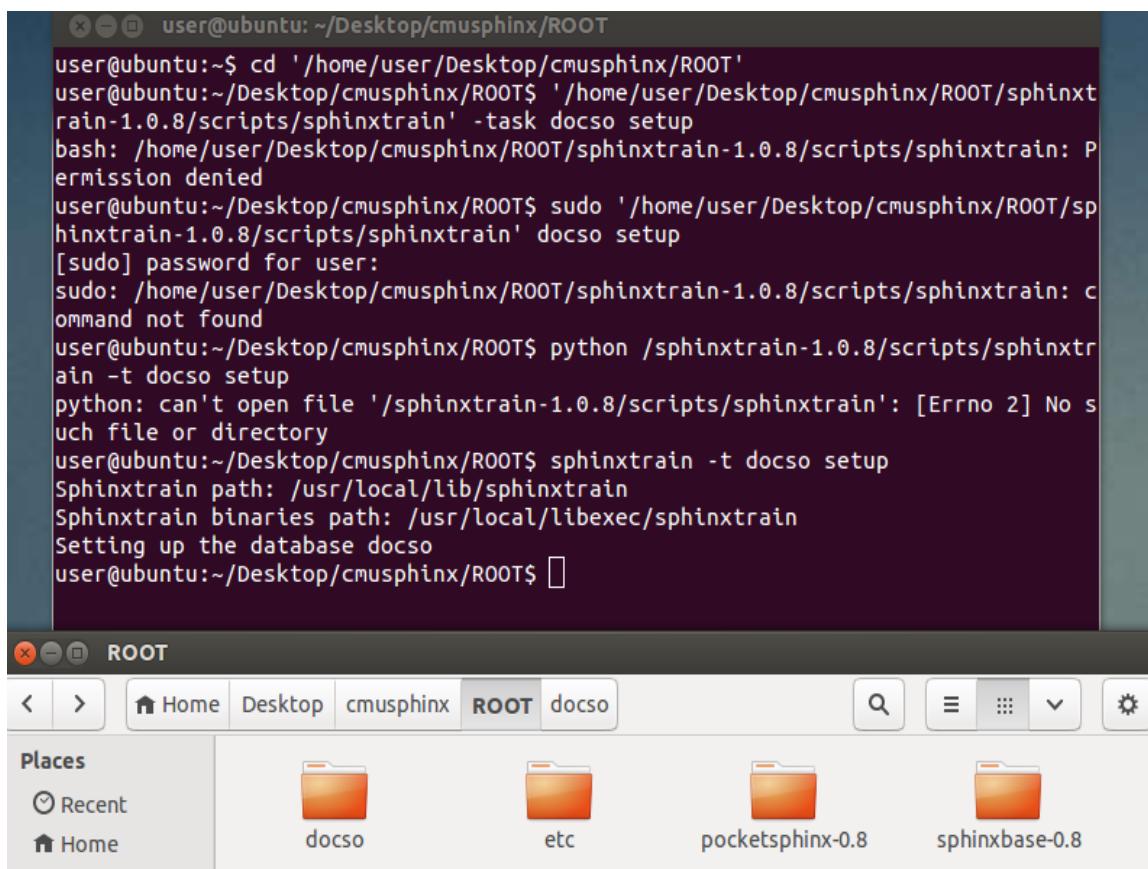
```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

Tiếp đến cd về ROOT

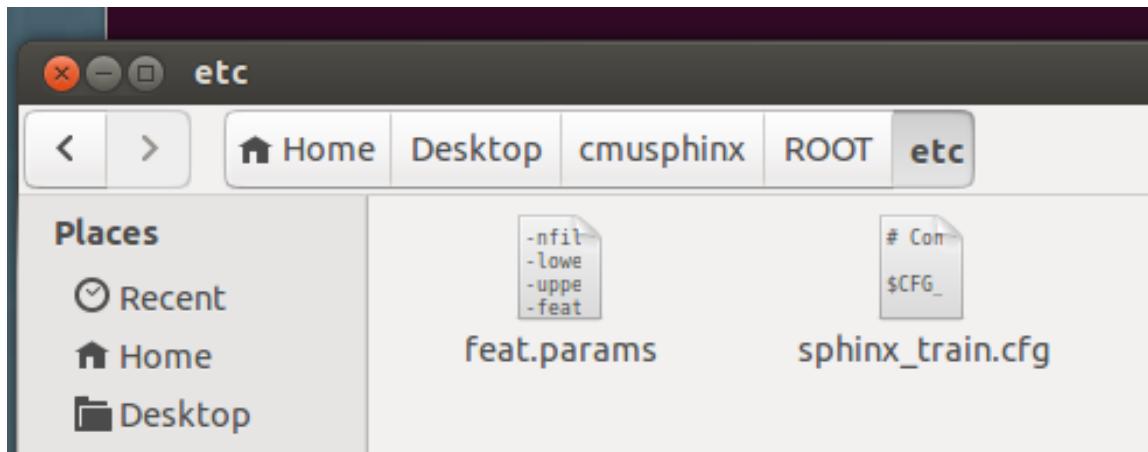


Gõ : sphinxtrain –t docso setup

Máy sẽ sinh ra 1 thư mục mới trong ROOT : là thư mục “etc”



Trong etc lúc này sẽ có hai file:



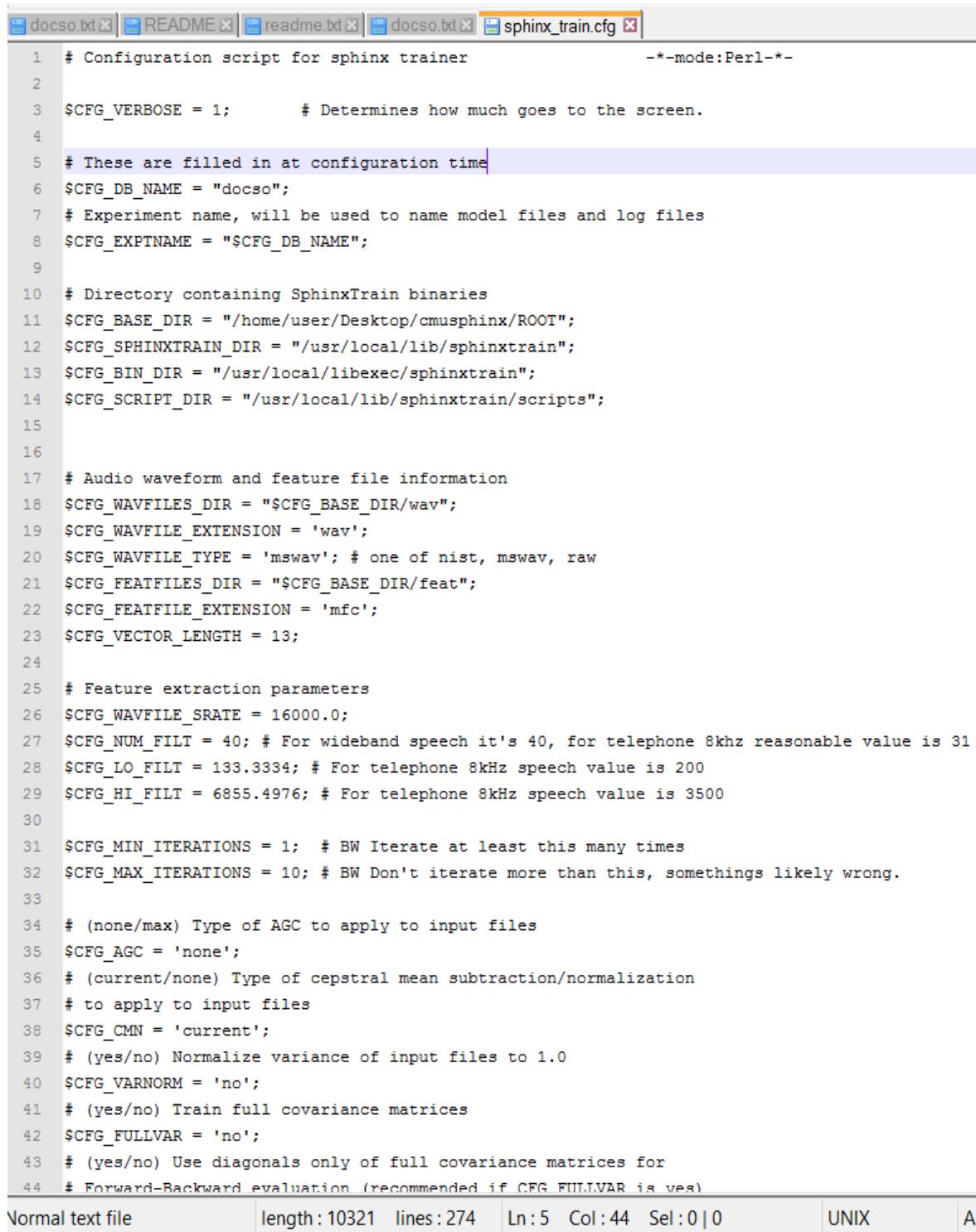
Edit file “sphinx_train.cfg” để chỉnh lại cấu hình cho đúng :

Cấu hình dựa vào sphinx :

http://cmusphinx.sourceforge.net/wiki/tutorial#setup_the_format_of_database_audio

Cụ thể là sẽ chú ý các dòng : 18 , 92, 102 ,

File này nếu sinh ra đúng sẽ như thế này :



```

1 # Configuration script for sphinx trainer           -*-mode:Perl-*-
2
3 $CFG_VERBOSE = 1;      # Determines how much goes to the screen.
4
5 # These are filled in at configuration time
6 $CFG_DB_NAME = "docso";
7 # Experiment name, will be used to name model files and log files
8 $CFG_EXPTNAME = "$CFG_DB_NAME";
9
10 # Directory containing SphinxTrain binaries
11 $CFG_BASE_DIR = "/home/user/Desktop/cmusphinx/ROOT";
12 $CFG_SPHINXTRAIN_DIR = "/usr/local/lib/sphinxtrain";
13 $CFG_BIN_DIR = "/usr/local/libexec/sphinxtrain";
14 $CFG_SCRIPT_DIR = "/usr/local/lib/sphinxtrain/scripts";
15
16
17 # Audio waveform and feature file information
18 $CFG_WAVFILES_DIR = "$CFG_BASE_DIR/wav";
19 $CFG_WAVFILE_EXTENSION = 'wav';
20 $CFG_WAVFILE_TYPE = 'mswav'; # one of nist, mswav, raw
21 $CFG_FEATFILES_DIR = "$CFG_BASE_DIR/feat";
22 $CFG_FEATFILE_EXTENSION = 'mfc';
23 $CFG_VECTOR_LENGTH = 13;
24
25 # Feature extraction parameters
26 $CFG_WAVFILE_SRATE = 16000.0;
27 $CFG_NUM_FILT = 40; # For wideband speech it's 40, for telephone 8khz reasonable value is 31
28 $CFG_LO_FILT = 133.3334; # For telephone 8kHz speech value is 200
29 $CFG_HI_FILT = 6855.4976; # For telephone 8kHz speech value is 3500
30
31 $CFG_MIN_ITERATIONS = 1; # BW Iterate at least this many times
32 $CFG_MAX_ITERATIONS = 10; # BW Don't iterate more than this, somethings likely wrong.
33
34 # (none/max) Type of AGC to apply to input files
35 $CFG_AGC = 'none';
36 # (current/none) Type of cepstral mean subtraction/normalization
37 # to apply to input files
38 $CFG_CMN = 'current';
39 # (yes/no) Normalize variance of input files to 1.0
40 $CFG_VARNORM = 'no';
41 # (yes/no) Train full covariance matrices
42 $CFG_FULLVAR = 'no';
43 # (yes/no) Use diagonals only of full covariance matrices for
44 # Forward-Backward evaluation (recommended if CFG_FULLVAR is yes)

```

Normal text file length:10321 lines:274 Ln:5 Col:44 Sel:0|0 UNIX A

4.1 Chuẩn bị bộ huấn luyện cho Sphinx

Tạo một thư mục huấn luyện, mang tên gì đó có ý nghĩa cho bộ dữ liệu huấn luyện, ví dụ: “**docso**”.

Trong đó tạo 2 thư mục con là **etc**, **wav**.

Sau đó tạo các file như cấu trúc sau:

etc

- |___ your_db.**dic** - *bộ tự điển âm vị, âm tiết*
- |___ your_db.**phone** - *file chứa danh sách các âm vị*
- |___ your_db.**lm.DMP** - *Mô hình ngôn ngữ*
- |___ your_db.**filler** - *Danh sách các khoảng lặng*
- |___ your_db_train.**fileids** - *Danh sách các file huấn luyện*
- |___ your_db_train.**transcription** – *Dữ liệu dạng text của file huấn luyện*
- |___ your_db_test.**fileids** - *Danh sách các file test*
- |___ your_db_test.**transcription** - *Bản text của file test*

wav

- |___ speaker_1
 - |___ file_1.wav - *file thu âm một câu nói của người huấn luyện*
 - |___ ...
- |___ speaker_2
 - |___ file_2.wav

d. Phonetic Dictionary (your_db.dic)

File này chứa nội dung về cách phát âm của một từ trong bộ huấn luyện. Vd: từ “HELLO” được phát âm bằng sự kết hợp của các âm vị sau:

HH AH L OW (theo như trang chủ Sphinx ví dụ). Khi đó, trong file này sẽ ghi là:

- HELLO H AH L OW

Mỗi một dòng trong file là định nghĩa cách đọc của một từ.

File này có phân biệt ký tự hoa – thường. thông thường để xây dựng được file này, cần tìm hiểu về cách phát âm của một từ trong một ngôn ngữ nhất định. Nếu là tiếng Anh thì họ có cách đọc cho từ tiếng Anh có trong tự điển. Đây cũng làm 1 bước quan trọng để xây dựng thành công bộ huấn luyện.

Trong tiếng Việt, cách đọc và các viết một từ là gần như gắp liền với nhau. Không cần có hướng dẫn cách đọc khi học tiếng Việt, trong tiếng Anh cách đọc và cách viết không phụ thuộc nhau, vd “lead” (dẫn đầu) & “head” (cái đầu). Ví dụ: muốn xây dựng file này cho tiếng Việt, ta có thể định nghĩa các từ bằng nhiều cách như sau:

- BAN B A N

Với cách trên, ta xem từ “BAN” là một âm tiết với sự kết hợp của 3 âm vị là B, A, N.

- BAN B AN

Với cách trên, ta xem từ “BAN” là một âm tiết với sự kết hợp của 2 âm vị là B, AN.

Sphinx không hỗ trợ định nghĩa ở dạng word-base, nghĩa là cách đọc của một từ không được chính là từ đó. Vd: BAN BAN là không được cho phép. Tuy nhiên có thể làm một phương pháp tương đương thay thế nếu muốn xây dựng theo kiểu word-base. Khi đó phải định nghĩa từ theo kiểu 1 từ có nhiều cách đọc, ví dụ:

BAN BAN BANG

Ý nghĩa của dòng định nghĩa trên là từ “ban” có thể đọc theo 2 cách là “ban” (cách đọc đúng chuẩn) hoặc đọc là “bang” (cách đọc người miền nam).

Chỉ được dùng các ký hiệu a-z, A-Z, 0-9 để đảm bảo không gây lỗi cho file này.

Vấn đề thanh điệu được giải quyết tại đây:

Ta sẽ xem các âm vị đi chung với thanh điệu sẽ là một âm vị đọc lập. khi đó thay vì xem thanh điệu như một âm vị khác theo cách định nghĩa sau (định nghĩa cho từ “bản”):

BA3N B A 3 N

Ta sẽ xem âm ả là một âm vị khác, độc lập với âm a khi đó ta định nghĩa như sau:

BA3N B A3 N

Theo một số bài khóa luận cao học thì phương pháp này cho ra kết quả nhận diện tốt hơn cho âm tiết có thanh điệu.

e. Phoneset file (your_db.phone)

File này chứa tất cả các âm vị sử dụng trong file trên, mỗi một dòng là một âm vị, nên sắp xếp các âm vị đó theo thứ tự để Sphinx dễ quản lý (yêu cầu này được đề cập rõ ràng trong việc huấn luyện HTK, trong Sphinx không thấy nói). Lưu ý thêm một âm vị đặc biệt vào file này đó là SIL, âm vị đại diện cho khoảng lặng.

f. Language model file (your_db.lm.DMP)

File này nên ở định dạng ARPA hoặc DMP. Vấn đề về tạo file DMP này nhóm sẽ nghiên cứu sau, nội dung file này định nghĩa ngữ pháp cho các câu nói nhận dạng, được dùng để huấn luyện, test, và chạy chương trình. File này được tạo tự động bằng nhiều công cụ khác nhau.

g. Filler dictionary (your_db.filler)

Chứa các âm tiết dùng để “làm đầy”, thông thường là các khoảng lặng, chúng ta có thể định nghĩa file này như sau:

<s> SIL

</s> SIL

<sil> SIL

h. Danh sách file training (your_db_train.fileids)

Là file text chứa nội dung về đường dẫn tới file thu âm (file .wav). nằm trong thư mục wav, có trong sơ đồ thư mục trình bày phía trên. Vd:

speaker_1/file_1

Không ghi đuôi file wav vào. Mỗi một dòng là một file.

**i. Nội dung file wav bằng text – transcript
(your_db_train.transcription)**

Đây là phần nội dung mà file wav mình thu âm được, để huấn luyện cho Sphinx hiểu những gì mình nói, thì mình cần cung cấp một file text để giúp cho Sphinx hiểu được mình nói gì và học từ đó. Cấu trúc một file transcript gồm nhiều dòng, mỗi một dòng là nội dung của một file wav kèm theo tên file wav đó. Vd:

<s> hello word </s> (file_1)

Lưu ý mỗi câu nói cần được bao trong tag <s> </s>.

Thực hiện tương tự với your_db_test.fileids và your_db_test.transcription.

4.2 Cách thức thu âm

Để có được các file wav, file huấn luyện, ta phải thu âm, thu càng nhiều càng tốt. số lượng từ cần huấn luyện và số tiếng (hour) cần thu như sau:

Bảng 4 Các thông số tương ứng với độ lớn của bộ huấn luyện

Vocabulary	Hours in db	Senones	Densities	Example
20	5	200	8	Tidigits Digits Recognition
100	20	2000	8	RM1 Command and Control
5000	30	4000	16	WSJ1 5k Small Dictation
20000	80	4000	32	WSJ1 20k Big Dictation
60000	200	6000	16	HUB4 Broadcast News

60000	2000	12000	64	Fisher Rich Telephone Transcription
-------	------	-------	----	-------------------------------------

Thu âm là một quá trình đòi hỏi sự kiên nhẫn và cẩn thận của người thực hiện. Đây cũng chính là điểm khó khăn nhất khi thực hiện khóa luận này. Công cụ dùng để thu âm là Audacity. Loại micro dùng trong thu âm nên chọn tốt nhất là loại headphone. Môi trường thu âm cần sự yên tĩnh, lưu ý: thiết lập tần số lấy mẫu là 16kHz, định dạng âm 16 bit mono (nếu dùng để nhận dạng trên máy tính) và là 8kHz, 16 bit mono (dành cho nhận dạng trên thiết bị di động), tắt loa máy tính khi thu, để micro hơi dưới miệng để tránh hơi thở từ mũi ra làm nhiễu tín hiệu. Các thông tin cần thiết để chuẩn bị cho việc thu âm có thể được tìm hiểu chi tiết tại VoxForge.

4.3 Tiến hành huấn luyện mô hình bằng Sphinx

Sau khi chuẩn bị một folder train (thư mục chứa toàn bộ các file chuẩn bị bên trên, file âm thanh,... người ta gọi tên folder huấn luyện là *task folder*) như phần trình bày trên. Tiếp theo ta sử dụng một số lệnh của Sphinx Train để tạo tự động các mã lệnh huấn luyện (Training Script). Mã lệnh huấn luyện có nhiệm vụ thực hiện toàn bộ các công đoạn huấn luyện bao gồm: Tiền xử lý tín hiệu âm thanh, rút trích đặc trưng âm học, xây dựng, ước lượng mô hình HMM nhờ thuật toán Baum-Welch,..

Để bắt đầu khởi tạo các thư mục chuẩn bị (các thực mục này Sphinx dùng cho quá trình huấn luyện, tạo tự động) và các file Script huấn luyện. Ta thực hiện dòng lệnh sau vào Command Line trong Linux:

- Dành cho Sphinxtrain từ bản 1.0.7 trở về trước:

```
./SphinxTrain/scripts_pl/setup_SphinxTrain.pl -task [task_folder_name]
```

```
./pocketsphinx/scripts/setup_sphinx.pl -task [task_folder_name]
```

- Dành cho Sphinxtrain bản *snapshot*:

```
sphinxtrain -t [task_folder_name]setup
```

Thực hiện dòng lệnh trên, Sphinx sẽ tự động tạo cho ta các thư mục do Sphinxtrain chuẩn bị để thực hiện huấn luyện:

- | |
|---|
| ▪ <i>bin</i> (có thể không xuất hiện trong bản Sphinxtrain mới) |
|---|

- *bwaccumdir*
- *etc*
- *feat*
- *logdir*
- *model_parameters*
- *model_architecture*
- *python* (có thể không xuất hiện trong bản Sphinxtrain mới)
- *scripts_pl* (có thể không xuất hiện trong bản Sphinxtrain mới)
- *wav*

Sau khi đã tạo thành công các thư mục như trên, ta thực hiện thao tác chỉnh sửa một số thông số để chuẩn bị bước vào tiến hành huấn luyện. Mở tập tin **etc/sphinx_train.cfg** tìm tới các dòng sau và thay đổi thông số.

j. Cài đặt định dạng âm thanh của file huấn luyện

```
$CFG_WAVFILES_DIR = "$CFG_BASE_DIR/wav";
$CFG_WAVFILE_EXTENSION = 'sph';
$CFG_WAVFILE_TYPE = 'nist'; # one of nist, mswav, raw
```

Thay đổi thông số *sph* thành *wav* và *nist* thành *mswav* như sau:

```
$CFG_WAVFILES_DIR = "$CFG_BASE_DIR/wav";
$CFG_WAVFILE_EXTENSION = 'wav';
$CFG_WAVFILE_TYPE = 'mswav'; # one of nist, mswav, raw
```

k. Cài đặt đường dẫn đến các file chuẩn bị

Kiểm tra xem các thông số sau đây có thay đổi hay không so với thư mục hiện tại, đây là đường dẫn do Sphinx tự tạo ra để truy cập đến các file mà ta đã chuẩn bị, trong đó \$CFG_DB_NAME là tên task folder của ta phía trên.

```
$CFG_DICTIONARY      = "$CFG_LIST_DIR/$CFG_DB_NAME.dic";
$CFG_RAWPHONEFILE   = "$CFG_LIST_DIR/$CFG_DB_NAME.phone";
```

```
$CFG_FILLERDICT      = "$CFG_LIST_DIR/$CFG_DB_NAME.filler";
$CFG_LISTOFFILES     = "$CFG_LIST_DIR/${CFG_DB_NAME}_train.fileids";
$CFG_TRANSCRIPTFILE =
"$CFG_LIST_DIR/${CFG_DB_NAME}_train.transcription"
```

I. Tinh chỉnh kiểu và các tham số của mô hình huấn luyện

```
$CFG_FINAL_NUM_DENSITIES = 8;
```

Số lượng senones phụ thuộc vào độ lớn của bộ từ vựng cũng như tập tin âm thanh huấn luyện, số lượng senone thích hợp có thể tra cứu trong (Bảng 4 Các thông số tương ứng với độ lớn của bộ huấn luyện). Các thông số trong bảng trên chỉ mang tính chất tham khảo, thực tế chúng ta có thể thực nghiệm bộ huấn luyện với các mức senones khác nhau để mang lại kết quả cao nhất.

```
# Number of tied states (senones) to create in decision-tree clustering
$CFG_N_TIED_STATES = 1000;
```

Công đoạn cuối cùng là thực hiện mã huấn luyện sau:

- Đối với Sphinxtrain từ bản 1.0.7 về trước

```
./scripts_pl/make_feats.pl -ctl etc/an4_train.fileids
./scripts_pl/make_feats.pl -ctl etc/an4_test.fileids
./scripts_pl/RunAll.pl
```

- Đối với Sphinxtrain bản *snapshot*

```
sphinxtrain run
```

Trong quá trình huấn luyện nếu xảy ra lỗi chủ yếu là do file chuẩn bị (phone, transcript, fileid,...) chưa đúng, các lỗi được ghi lại cụ thể trong 1 file log có tên [task_folder_name].html nằm trong thực mục task folder.

Phần cài đặt và hướng dẫn huấn luyện được trình bày chi tiết trong tài liệu.

4.4 Các phần mềm yêu cầu

Sphinx-4 được xây dựng và chạy thử trên môi trường Solaris Operating , Mac OS X, Linux và Window 32bit. Biên dịch, chạy và kiểm thử Sphinx-4 cần một số phần mềm hỗ trợ. Trước khi bắt đầu, cần các phần mềm sau trong máy tính :

- ❖ Java SE 6 Development Kit hoặc cao hơn. Truy cập vào java.sun.com, và chọn “J2SE” để tải về. Sphinx khuyến khích dùng bản JDK 6 Update 14 (Vì đây là phiên bản mà CMU đã dùng để xây dựng Sphinx-4)
- ❖ Ant 1.6.0 hoặc cao hơn , có thể tải tại địa chỉ : ant.apache.org
- ❖ Subversion (svn) , chỉ cần khi muốn tác động trực tiếp với cây svn. Nếu cài svn , phải cài cygwin, giả lập môi trường linux

4.5 Tải Sphinx-4

Sphinx-4 có 2 bản để tải [[download](#)]:

- ❖ **sphinx4-{version}-bin.zip**: cung cấp file jar, tài liệu và demo
- ❖ **sphinx4-{version}-src.zip**: cung cấp mã nguồn, tài liệu, demo, unit tests và regression tests.

Sau khi tải Sphinx về, giải nén ra sẽ được 1 thư mục “sphinx5-{version}” . Đồng thời cũng có mô hình âm thanh RM1 , và mô hình ngôn ngữ và âm thanh. Nếu muốn chạy regression test cho RM1 và HUB4 thì tải về tại trang SourceForge.

Link download source code tại svn:

<https://cmusphinx.svn.sourceforge.net/svnroot/cmusphinx/trunk/sphinx>

4/

4.6 Build Sphinx-4

1/ Cài JSAPI 1.0

Trước khi build Sphinx-4, có một điều rất quan trọng phải làm là cài môi trường hỗ trợ tiếng nói của Java là Java Speech API (JSAPI), vì 1 số file test và demo chạy dựa trên JSAPI

2/ Run ant

Để build Sphinx-4, trong cmd, duyệt đến thư mục chứa Sphinx-4 (thường là “cd sphinx4” là được). Tạo các biến môi trường: JAVA_HOME trỏ đến thư mục JDK, và ANT_HOME trỏ đến ant. Và PATH gồm cả 2 thư mục

```
export JAVA_HOME=/usr/local/jdk1.6.0_14
export ANT_HOME=/usr/local/apache-ant-1.8.0
export PATH=/usr/local/jdk1.6.0_10/bin:/usr/local/apache-ant-1.8.0/bin:$PATH
```

bin của JDK và ant:

Sau đó gõ trong cmd

```
ant
```

Các bước trên sẽ thực thi lệnh [Apache Ant](#) để build các class của Sphinx-4 dưới thư mục bld, file jar dưới thư mục lib và demo jar dưới thư mục bin

Để xóa các output của quá trình build để làm lại

4.7 Tạo javadocs

Javadocs đã được build sẵn nếu down sphinx4-{Version}-bin.zip . Để tự build javadocs, phải down source của sphinx4.

Đến thư mục gốc của sphinx4 ("sphinx4-{version}"), gõ:

```
ant javadoc
```

Điều này sẽ build javadocs từ các class public , chỉ hiển thị các class public và các trường.

Để biết thêm thông tin về các class private và protected

```
ant -Daccess=private javadoc
```

4.8 Cách để cấu hình IDE (Eclipse, NetBeans)

Cài đặt theo các bước sau:

Add tất cả **subfolders(!)** của đường dẫn **src**

Add **lib/js.jar**, **lib/tags.jar** và **lib/jsapi.jar** to your project classpath. Vì lý do bản quyền **jsapi.jar** sẽ không chạy trực tiếp với Sphinx4 nhưng cũng có thể dễ dàng tạo bằng cách chạy **lib/jsapi.sh** (hoặc **lib/jsapi.bat** trong windows) một lần nữa.

Để chạy các task chung chung (như phát triển sphinx4.jar , mô hình , hoặc các demo jar) trực tiếp trong IDW, có thể thêm bundled **build.xml** như file ant của project. Điều này có thể làm được trong hầu hết trường hợp bằng cách bấm chuột phải vào file **build.xml** trong navigator của IDE , và chọn

“Add as project ant file”. Để debug demo, có thể cần thêm thư mục `src/apps` và các file jar mô hình âm thanh(có thể được triển khai đến thư mục `lib` với 1 lệnh đơn giản `ant all`).

4.9 Chạy chương trình Demo

Sphinx-4 chứa một số chương trình demo. Nếu tải bản (sphinx4-{version}-bin.zip), file JAR chứa demo được build sẵn, vì vậy chỉ cần chạy trực tiếp. Tuy nhiên, nếu down bản source code (sphinx4-{version}-src.zip or via svn), sẽ phải build demo lại.

Có các demo như sau :

Các demo đơn giản cho người mới bắt đầu

- + [Hello World Demo](#): một chương trình console để nhận dạng các cụm từ đơn giản
- + [Hello N-Gram Demo](#): một chương trình console để nhận dạng bằng mô hình N-Gram

Demos trích file âm thanh

- + [Transcriber Demo](#): một chương trình chỉ cách ghi lại một file âm thanh liên tục mà có nhiều khoảng lặng
- + [Confidence Demo](#): một demo chỉ cách lấy độ chính xác
- + [Lattice Demo](#): demo đơn giản để trích lưới lattice từ kết quả nhận dạng.
- + [Class-Based Language model Demo](#): một demo đơn giản cho class mô hình ngôn ngữ
- + [Aligner Demo](#): Sắp xếp các file âm thanh đã trích ghi lại số lần của từ. Có thể dùng để làm phụ đề đóng

Demo dialog để viết các trình có giao diện cao cấp

- ✚ [ZipCity Demo](#): một chương trình dạng java web để nhận dạng mã vùng và định vị thành phố và tiểu bang của Mỹ.
- ✚ [JSGF Demo](#): một demo đơn giản chỉ cách chuyển đổi giữa nhiều cấu trúc JSGF
- ✚ [Dialog Demo](#): một demo đơn giản chỉ cách chuyển đổi giữa nhiều cấu trúc JSGF và cấu trúc chính tả
- ✚ [Action Tags Demo](#): demo chỉ cách sử dụng các thẻ action để tiến trình post cho phân tích ngữ pháp của đối tượng lấy từ cấu trúc JSGF\

Ngoài ra còn có một [live-mode test program](#) chỉ chạy nếu download source.

[AudioTool](#) là một công cụ để ghi âm và hiện lại sóng âm và âm phô curamoojt tín hiệu âm thanh. Có trong cả 2 bản download.

4.10 Quản lý cấu hình Sphinx-4

Nói đến các phần mềm mã nguồn mở, việc quản lý cấu hình là một điều vô cùng quan trọng, bởi vì khi toàn bộ framework đã được định nghĩa, thì việc còn lại là làm sao sử dụng chúng một cách hiệu quả, và để chúng chạy được. Ngoài ra, tuân theo cấu hình chuẩn của Sphinx-4 không chỉ vì lý do phần mềm có chạy được hay không, mà còn là sự tôn trọng đối với người viết ra framework. Vậy nên, sau đây chúng em xin phép được trình bày cách quản lý cấu hình dựa trên chỉ dẫn của CMU .

Configuration File : định nghĩa các phần sau :

- Tên và kiểu của các thành phần trong hệ thống
- Kết nối các thành phần này lại để chúng có thể giao tiếp với nhau

- Chi tiết cấu hình của mỗi thành phần

Ví dụ đơn giản:

```

<component name="decoder" type="edu.cmu.sphinx.decoder.Decoder">
    <property name="searchManager" value="searchManager"/>
</component>

<component name="searchManager"
    type="edu.cmu.sphinx.decoder.search.SimpleBreadthFirstSearchManager">
    <property name="logMath" value="logMath"/>
    <property name="linguist" value="flatLinguist"/>
    <property name="pruner" value="trivialPruner"/>
    <property name="scorer" value="threadedScorer"/>
    <property name="activeListFactory" value="activeList"/>
</component>

```

Ví dụ bên trên là 1 file XML đơn giản, định nghĩa 1 thành phần là “decoder”, “SearchManager” và kiểu “*edu.cmu.sphinx.decoder.Decoder*”, “*edu.cmu.sphinx.search.simpleBreadthFirstSearchManager*”

4.11.1 Định nghĩa các thành phần

Ví dụ 2:

```

<config>
    <component
        name="dct"
        type="edu.cmu.sphinx.frontend.transform.DiscreteCosineTransform"/>
    <component
        name="batchCMN"
        type="edu.cmu.sphinx.frontend.feature.BatchCMN"/>
    <component
        name="liveCMN"
        type="edu.cmu.sphinx.frontend.feature.LiveCMN"/>
    <component
        name="featureExtraction"
        type="edu.cmu.sphinx.frontend.feature.DeltasFeatureExtractor"/>
</config>

```

4.11.2 Định nghĩa cấu hình dữ liệu

```

<config>
    <component
        name="concatDataSource"
        type="edu.cmu.sphinx.frontend.util.ConcatFileDataSource">
        <property
            name="sampleRate"
            value="16000"/>
        <property
            name="transcriptFile"
            value="reference.txt"/>
        <property
            name="silenceFile"
            value="/lab/speech/sphinx4/data/tidigits/test/raw16k/silence1sec.raw"/>
        <property
            name="bytesPerRead"
            value="320"/>
        <property
            name="batchFile"
            value="tidigits.batch"/>
        <property
            name="addRandomSilence"
            value="true"/>
    </component>
</config> >

```

Trong Sphinx-4, dữ liệu của file cấu hình là các thuộc tính của nó. Ở đây có 6 thuộc tính cho thành phần concatDataSource . Các thuộc tính đơn giản là cặp name/value và được set các property như trên.

Các thuộc tính này có thể được định nghĩa cho một thành phần biến đổi dựa trên kiểu thành phần. Tài liệu API cho một thành phần gồm đặc tả các thuộc tính, kiểu dữ liệu của mỗi thành phần, và các giá trị mặc định của mỗi đặc tính. Ví dụ một đặc tả cho thuộc tính được dùng bên trên có thể tìm thấy ở trang [ConcatFileDataSource](#). Ở đây nhóm em không thể liệt kê hết tất cả các thành phần cũng như đặc tả của nó vì số lượng quá lớn, chúng em xin phép trình bày một số thành phần và thuộc tính quan trọng ở chương trước về các package và class của Sphinx4.

Nếu có một thuộc tính bị bỏ qua, thành phần thường sẽ lấy giá trị mặc định của thuộc tính đó.

4.11.3 Các loại dữ liệu cấu hình

Các thuộc tính đơn giản của Sphinx-4 là :

- **boolean** - "true" or "false"
- **float** – số thực
- **double** – số thực (lớn hơn float)
- **int** – số nguyên 32 bit có dấu
- **String** – chuỗi
- **Component** – tên của một thành phần Sphinx-4 (chi tiết xin nói ở phần sau)

Ví dụ :

<code>value=""Twas brillig and slithey toves"</code>	A string
<code>value="3.14"</code>	a float
<code>value="1E-140"</code>	a double
<code>value="16000"</code>	an integer
<code>value="false"</code>	a boolean

`value="beamPruner"`

a component

Thêm một vài kiểu đơn giản nữa là kiểu list :

- **String list** - a list of strings
- **Component list** - a list of components

Các list được định nghĩa một element propertylist . Mỗi item trong 1 list được định nghĩa như 1 item element. Ví dụ :

```
<component name="fileManager"
type="edu.cmu.sphinx.sample.FileManager">
  <propertylist name="fileNames">
    <item>file1.txt</item>
    <item>file2.txt</item>
    <item>file3.txt</item>
  </propertylist>
</component>
```

Các thuộc tính list được định nghĩa đơn giản :

```
<component name="mfcLiveFrontEnd"
type="edu.cmu.sphinx.frontend.FrontEnd">
  <propertylist name="pipeline">
    <item>concatDataSource </item>
    <item>speechClassifier </item>
    <item>speechMarker </item>
```

```

<item>nonSpeechDataFilter </item>
<item>preemphasizer </item>
<item>windower </item>
<item>fft </item>
<item>melFilterBank </item>
<item>dct </item>
<item>liveCMN </item>
<item>featureExtraction </item>
</propertylist>
</component>

```

Kiểm tra lỗi

Khi chạy một file cấu hình, trình quản lý cấu hình sẽ kiểm tra các lỗi chắc chắn và hủy tiến trình nếu một lỗi được phát hiện. Một vài lỗi được phát hiện như là :

- ❖ **Invalid XML** – file XML không đúng
- ❖ **Unknown XML elements** - Có các element không biết.
- ❖ **Missing, extra or Unknown XML attributes** - một element có sai số thuộc tính attributes
- ❖ **Multiply defined properties** - định nghĩa thuộc tính cho một thành phần nhiều lần
- ❖ **Bad data type for a property** – một giá trị cho trước không thể được chuyển thành một kiểu của thuộc tính
- ❖ **Multiply defined components** – mỗi thành phần chỉ được định nghĩa một lần

- ❖ **Out-of-range-data for a component** – Giá trị vượt ngưỡng cho phép

Các Elements

Chi tiết về các element và thuộc tính của nó trong file cấu hình

Element	Attributes	Sub-elements	Mô tả
<config>	Không có	<component> <property> <propertylist>	Nó là root. Nó có thể có bất cứ số thành phần, thuộc tính và danh sách thuộc tính của các element con nào
<component>	name - the component name type - the component type	<property> <propertylist>	Định nghĩa như một thực thể của thành phần. Element này phải có <i>name</i> và <i>type</i> .
<property>	name - the property name value - the type of the property	None	Dùng để định nghĩa một thuộc tính của một thành phần, hoặc một thuộc tính toàn cục. Phải có <i>name</i> and <i>value</i> .

<propertylist>	name - the name of the property list	<item>	Dùng để định nghĩa list strings hoặc components. Phải có <i>name</i> . Nó có thể có bất cứ số <i>item</i> con nào.
<item>	none	none	Nội dung của element này định nghĩa một chuỗi hoặc tên của một thành phần.

4.11.4 Các thuộc tính toàn cục

Các thuộc tính toàn cục là thuộc tính được định nghĩa bên ngoài các thành phần, ngay bên dưới thẻ <config>. Ví dụ :

```
<config>
    <property name="absoluteBeam" value="1000"/>
    <property name="relativeBeam" value="1E-10"/>
</config>
```

Các biến toàn cục này có thể có các thành phần bên trong. Một biến toàn cục được tham chiếu bằng cách dùng cú pháp ***\${tên biến}***. Để tham chiếu một biến toàn cục đã được định nghĩa trong ví dụ trên, hãy sử dụng ***\${absoluteBeam}*** và ***\${relativeBeam}***

Sau đây là ví dụ cách sử dụng biến toàn cục trong file cấu hình :

```
<config>
    <property name="sampleRate" value="16000"/>

    <component name="concatDataSource"
type="edu.cmu.sphinx.frontend.util.ConcatFileDataSource">
        <property name="sampleRate" value="${sampleRate}"/>
    </component>

    <component name="microphone"
type="edu.cmu.sphinx.frontend.util.Microphone">
        <property name="sampleRate" value="${sampleRate}"/>
    </component>

    <component name="streamDataSource"
type="edu.cmu.sphinx.frontend.util.StreamDataSource">
        <property name="sampleRate" value="${sampleRate}"/>
    </component>

</config> >
```

Biến toàn cục **sampleRate** được gán giá trị là 16000 , và sau đó, ba thành phần : **concatDataSource** , **microphone** và **streamDataSource** đều có thuộc tính sampleRate , và gọi giá trị được định nghĩa ở biến toàn cục là 16000. Khi muốn thay đổi sampleRate này, chỉ cần thay đổi giá trị của biến toàn cục, cả 3 thành phần sẽ thay đổi giá trị

Ví dụ cấu hình FrontEnd :

```
<config>
    <property name="cmn" value="liveCMN"/>

    <component name="mfcFrontEnd"
type="edu.cmu.sphinx.frontend.FrontEnd">
        <propertylist name="pipeline">
            <item>streamDataSource</item>
            <item>preemphasizer</item>
            <item>windower</item>
            <item>fft</item>
            <item>melFilterBank</item>
            <item>dct</item>
            <item>${cmn}/item>
            <item>featureExtraction</item>
        </propertylist>
    </component>
</config>
```

Lưu ý : không được dùng biến toàn cục để đặt tên và kiểu của các thành phần, mặc dù có đặt cũng không bị lỗi . (mặc dù điều này sẽ tiện lợi hơn khi có nhiều thành phần cùng kiểu, nhưng nó đi ngược lại với cấu trúc của file cấu hình do CMU qui định)

```
<config>
    <property name="cmn" value="liveCMN"/>
    <!-- hợp lệ nhưng KHÔNG được dùng như vậy -->
```

```

<component name="\$\{cmn\}"
type="edu.cmu.sphinx.frontend.CepstralMeanNormalizer">
</component>
</config>

```

4.11.5 Debug file cấu hình

Sau đây là một số điều giúp phát triển một file cấu hình và để nó hoạt động

- ❖ Khi cấu hình bị lỗi bộ quản lý cấu hình sẽ ném 1 PropertyException miêu tả chi tiết nguyên nhân của lỗi. Các exception này được báo bởi toàn bộ các chương trình và ứng dụng chính của Sphinx-4. Dùng những thông báo này để debug các bẩn đề trong file cấu hình.

- ❖ Có một thuộc tính toàn cục đặc biệt là ***showCreations***. Nếu thuộc tính này được gắn là “true”, trình quản lý cấu hình sẽ chỉ ra tên của tất cả các thành phần được tạo. Điều khi điều này giúp tìm ra các thành phần nào dư hoặc thiếu.

```

java -cp ../../bld/classes -DshowCreations=true
edu.cmu.sphinx.tools.batch.BatchModeRecognizer \
    tidigits.config.xml tidigits.batch
Creating: batch
Creating: connectedDigitsRecognizer
Creating: digitsDecoder
Creating: searchManager
Creating: logMath

```

```
Creating: flatLinguist
Creating: wordListGrammar
Creating: dictionary
Creating: acousticModel
Creating: sphinx3Loader
Creating: trivialPruner
Creating: threadedScorer
Creating: mfcFrontEnd
Creating: streamDataSource
Creating: preemphasizer
Creating: windower
Creating: fft
Creating: melFilterBank
Creating: dct
Creating: batchCMN
Creating: featureExtraction
Creating: activeList
Creating: accuracyTracker
Creating: speedTracker
Creating: memoryTracker
Creating: recognizerMonitor
Creating: linguistStats
```

- ❖ Có một thành phần là **ConfigMonitor** dùng để giúp debug file cấu hình bằng cách hiển thị các cấu hình mà hiện được hệ thống đang dùng.

Điển hình là, cấu hình màn hình được đặt để hiển thị các cấu hình của hệ thống sau khi nó khởi tạo hoàn tất, và trước khi bắt đầu nhận dạng. Ví dụ :

```
===== config =====
```

```
fft:
```

```
    numberFftPoints = [DEFAULT]
```

```
trivialPruner:
```

```
searchManager:
```

```
    scorer = threadedScorer
```

```
    activeListFactory = activeList
```

```
    pruner = trivialPruner
```

```
    logMath = logMath
```

```
    growSkipInterval = [DEFAULT]
```

```
    showTokenCount = [DEFAULT]
```

```
    wantEntryPruning = [DEFAULT]
```

```
    linguist = flatLinguist
```

```
    relativeWordBeamWidth = [DEFAULT]
```

```
wordRecognizer:
```

```
    decoder = decoder
```

```
melFilterBank:
```

```
    numberFilters = [DEFAULT]
```

```
    maximumFrequency = [DEFAULT]
```

```
    minimumFrequency = [DEFAULT]
```

```
threadedScorer:  
    numThreads = 0  
    scoreablesKeepFeature = true  
    frontend = mfcFrontEnd  
    isCpuRelative = true  
    minScoreablesPerThread = 10  
  
preemphasizer:  
    factor = [DEFAULT]  
  
memoryTracker:  
    showDetails = [DEFAULT]  
    showSummary = [DEFAULT]  
    recognizer = wordRecognizer
```

Lưu ý rằng các thuộc tính không định nghĩa rõ ràng trong file cấu hình hoặc các thuộc tính hệ thống từ dòng lệnh được gán [DEFAULT]. Điều này thể hiện rằng các biến này được gán các giá trị mặc định.

Ví dụ cụ thể cho chương trình nhận dạng kí số:

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<!--  
 Sphinx-4 Configuration
```

```
file
-->

<!-- **** -->
<!-- tidigits
    configuration file -->
<!-- **** -->

<config>

<!-- **** -->
-->
<!-- frequently tuned properties -->

<!-- **** -->
-->

<property
    name="absoluteBeamWidth" value="-1"/>
<property name="relativeBeamWidth" value="1E-200"/>
<property
    name="wordInsertionProbability" value="1E-36"/>
<property name="languageWeight" value="8"/>
<property
    name="silenceInsertionProbability" value="1"/>
<property name="skip" value="0"/>
```

```
<property
    name="linguist" value="flatLinguist"/>
<property name="frontend" value="mfcFrontEnd"/>

<!--
***** -->
<!-- batch tool configuration -->
<!--
***** -->

<component name="batch"
    type="edu.cmu.sphinx.tools.batch.BatchModeRecognizer">
<property name="recognizer"
    value="connectedDigitsRecognizer"/>
<property name="inputSource" value="streamDataSource"/>
<propertylist
    name="monitors">
    <item>accuracyTracker </item>
</propertylist>

</component>

<!-- **** -->
>
<!-- The
```

```
connectedDigitsRecognizer configuration -->
<!-- ****
-->

<component name="connectedDigitsRecognizer"
    type="edu.cmu.sphinx.recognizer.Recognizer">
<property name="decoder" value="digitsDecoder"/>
</component>

<!-- ****
-->
<!-- The Decoder configuration -->

<!-- ****
-->

<component name="digitsDecoder"
    type="edu.cmu.sphinx.decoder.Decoder">
<property name="searchManager" value="searchManager"/>
</component>

<component name="searchManager"
    type="edu.cmu.sphinx.decoder.search.SimpleBreadthFirstSearchManager">
```

```
<property name="logMath" value="logMath"/>
<property name="linguist" value="$\{linguist\}">
<property
    name="pruner" value="trivialPruner"/>
<property name="scorer" value="threadedScorer"/>
<property
    name="activeListFactory" value="activeList"/>
</component>

<component
    name="activeList"
    type="edu.cmu.sphinx.decoder.search.SortingActiveListFactory">
<property
    name="logMath" value="logMath"/>
<property name="absoluteBeamWidth" value="$\{absoluteBeamWidth\}">
    <property name="relativeBeamWidth" value="$\{relativeBeamWidth\}">
</component>

<component
    name="trivialPruner"
    type="edu.cmu.sphinx.decoder.pruner.SimplePruner"/>

<component
```

```
name="threadedScorer"
type="edu.cmu.sphinx.decoder.scorer.ThreadedAcousticScorer">
<property
    name="frontend" value="${frontend}"/>
<property name="isCpuRelative" value="true"/>
<property
    name="numThreads" value="0"/>
<property name="minScoreablesPerThread" value="10"/>
<property
    name="scoreablesKeepFeature" value="true"/>
</component>

<!-- **** -->
<!-- The linguist configuration -->

<!-- **** -->
<component
    name="flatLinguist"
type="edu.cmu.sphinx.linguist.flat.FlatLinguist">
<property name="logMath"
    value="logMath"/>
<property name="grammar" value="wordListGrammar"/>
<property
```

```
        name="acousticModel" value="acousticModel"/>
<property name="wordInsertionProbability"
          value="${wordInsertionProbability}"/>
<property name="silenceInsertionProbability"
          value="${silenceInsertionProbability}"/>
<property name="languageWeight"
          value="${languageWeight}"/>
</component>

<!-- **** -->
<!-- The Grammar configuration -->

<!-- **** -->
<component name="wordListGrammar"
          type="edu.cmu.sphinx.linguist.language.grammar.SimpleWordListGrammar">
<property name="path"
          value=".//tidigits.wordlist"/>
<property name="isLooping" value="true"/>
<property
```

```
        name="dictionary" value="dictionary"/>
<property name="optimizeGrammar" value="true"/>
<property
    name="logMath" value="logMath"/>
</component>

<!-- **** -->
<!-- The Dictionary configuration -->

<!-- **** -->

<component name="dictionary"
    type="edu.cmu.sphinx.linguist.dictionary.FullDictionary">
<property name="location"
    value="file:/lab/speech/sphinx4/data/tidigits_8gau_13dCep_16k_40mel_130
    Hz_6800Hz.bin.zip"/>
<property
    name="dictionaryPath" value= "dictionary"/>
<property name="fillerPath" value="fillerdict"/>
<property
```

```
        name="addSilEndingPronunciation" value="false"/>
    </component>

<!--
***** -->
<!-- The acoustic model configuration
-->
<!-- ***** --
>
<component
    name="acousticModel"
    type="edu.cmu.sphinx.linguist.acoustic.tiedstate.TiedStateAcousticModel">
    <property
        name="loader" value="sphinx3Loader"/>
    </component>

<component name="sphinx3Loader"
    type="edu.cmu.sphinx.linguist.acoustic.tiedstate.Sphinx3Loader">
    <property name="logMath" value="logMath"/>
    <property name="isBinary" value="true"/>
    <property name="location"
        value="file:/lab/speech/sphinx4/data/tidigits_8gau_13dCep_16k_40mel_130"/>
```

```
Hz_6800Hz.bin.zip"/>
<property
    name="definition_file"
    value="wd_dependent_phone.500.mdef"/>
<property name="data_location"
    value="wd_dependent_phone.cd_continuous_8gau"/>
<property name="properties_file" value="am.props"/>

    <property name="FeatureVectorLength" value="39"/>
</component>

<!--
***** -->
<!-- The frontend configuration -->

<!-- **** -->

<component name="mfcFrontEnd"
    type="edu.cmu.sphinx.frontend.FrontEnd">
<propertylist name="pipeline">
<item>streamDataSource</item>

    <item>preemphasizer</item>
<item>windower</item>
```

```
<item>fft</item>

<item>melFilterBank</item>
<item>dct</item>
<item>batchCMN</item>

<item>featureExtraction</item>
</propertylist>
</component>

<component
    name="preemphasizer"
    type="edu.cmu.sphinx.frontend.filter.Preemphasizer"/>

<component
    name="windower"
    type="edu.cmu.sphinx.frontend.window.RaisedCosineWindower">
</component>

<component name="fft"
    type="edu.cmu.sphinx.frontend.transform.DiscreteFourierTransform"/>

<component
    name="melFilterBank"
    type="edu.cmu.sphinx.frontend.frequencywarp.MelFrequencyFilterBank">
```

```
</component>

<component name="dct"
type="edu.cmu.sphinx.frontend.transform.DiscreteCosineTransform"/>

<component
  name="batchCMN"
type="edu.cmu.sphinx.frontend.feature.BatchCMN"/>

<component
  name="featureExtraction"
type="edu.cmu.sphinx.frontend.feature.DeltasFeatureExtractor"/>

<component
  name="streamDataSource"
type="edu.cmu.sphinx.frontend.util.StreamDataSource">
<property
  name="sampleRate" value="16000"/>
</component>

<component name="cepstrumSource"
type="edu.cmu.sphinx.frontend.util.StreamCepstrumSource">
<property name="sampleRate" value="16000"/>
```

```
</component>

<!-- **** -->
<!--

monitors -->
<!-- **** -->

<component
    name="accuracyTracker"
    type="edu.cmu.sphinx.instrumentation.BestPathAccuracyTracker">
<property
    name="recognizer" value="connectedDigitsRecognizer"/>
<property name="showAlignedResults" value="false"/>

<property name="showRawResults" value="false"/>
</component>

<!--
 **** -->
<!-- Miscellaneous components -->
<!--
 **** -->

<component name="logMath"
```

```
type="edu.cmu.sphinx.util.LogMath">
<property name="logBase" value="1.0001"/>
<property
  name="useAddTable" value="true"/>
</component>

</config>
```

CHƯƠNG 5 : ỦNG DỤNG DEMO VÀ ĐÁNH GIÁ KẾT LUẬN**6.1 ZipCity - demo của Sphinx4**

Cấu hình máy chạy demo: Win7 Ulltimate – 64 bit , Core 2 Duo ,
2.2Ghz , 4GB RAM, 320GB HDD

Môi trường ảnh hưởng :

Địa điểm : Quán cà phê

Tiếng ồn : Nhạc giao hưởng nhẹ , không có tiếng nói tạp âm

Kết quả

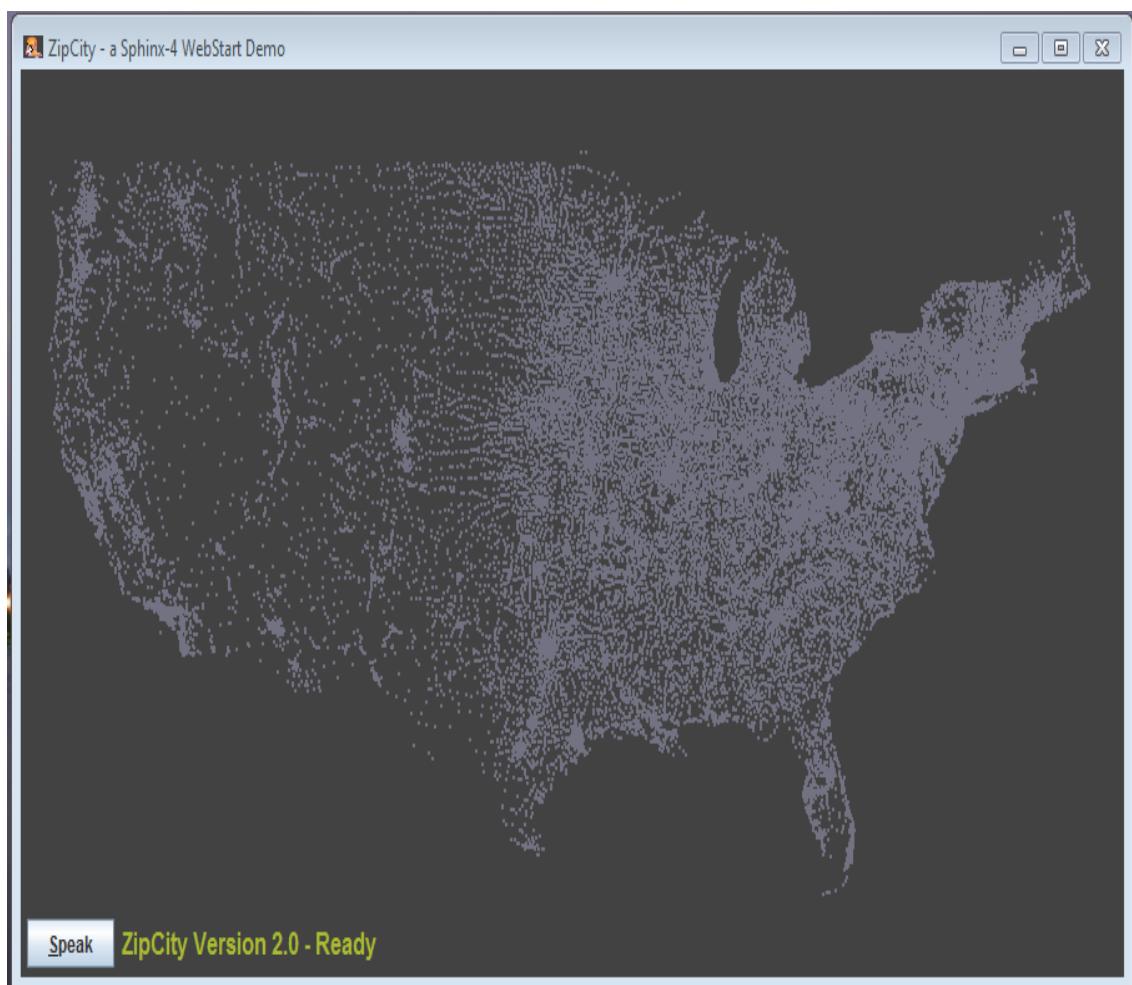
Câu Nói (Tiếng Anh)	Kết Quả	Đánh Giá
One- two – three – four - five	02345	Gần đúng
Nine – nine – nine – six - six	I didn't understand what you say	Không nghe được
One – nine – oh – five – five	19055	Chính xác
Oh – oh – one – eight – five	Can't not find 00185	Chính xác

Tỉ lệ lỗi :

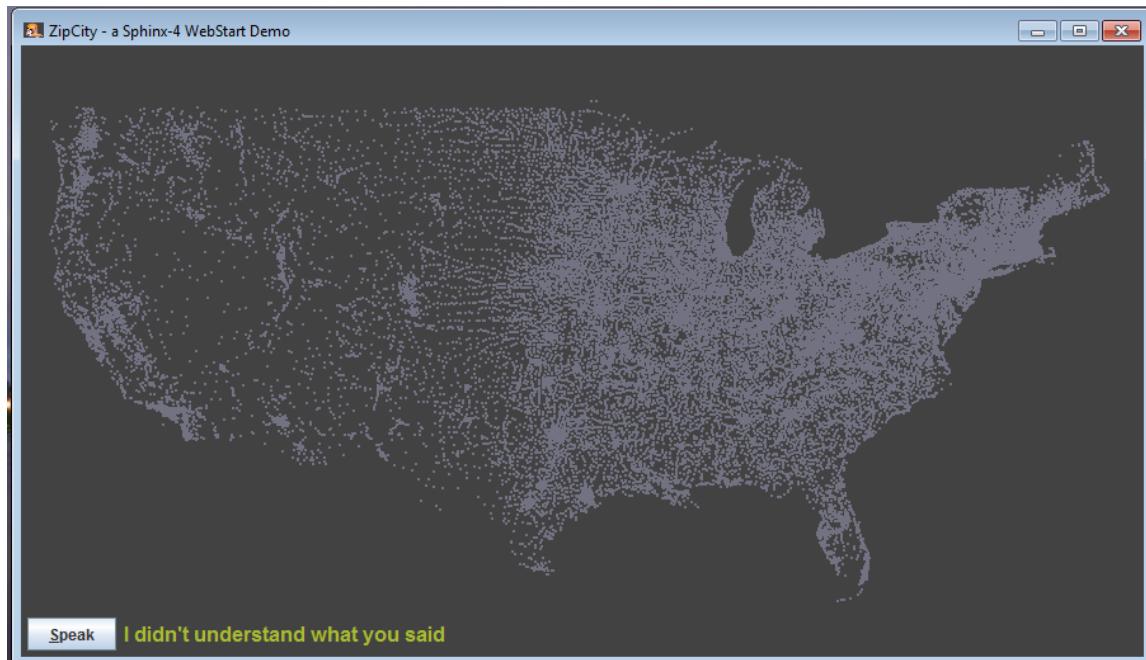
Người nói	Nội dung	Kết Quả	Độ chính xác
Hồ Thị Hoàng Yến Đúng 53/130 40%	12345	00050	10%
	01554	01855	30%
	67468	X	0%
	1578	X	0%
	99540	55005	10%
	20131	21301	60%
	21301	00010	20%
	00010	01015	60%
	21212	01212	90%
	05055	00585	20%
	57623	52035	30%
	12130	12130	100%
	12130	12130	100%

Nguyễn Anh Tuấn Đúng 63/130 48%	12345	10345	80%
	01554	01035	50%
	67468	00068	40%
	1578	10818	30%
	99540	98900	50%
	20131	00185	40%
	21301	01801	60%
	00010	00101	60%
	21212	01001	30%
	05055	80505	40%
	57623	50503	40%
	12130	12180	80%
	12130	10405	30%

1. Màn hình khởi động : ấn Speak để bắt đầu

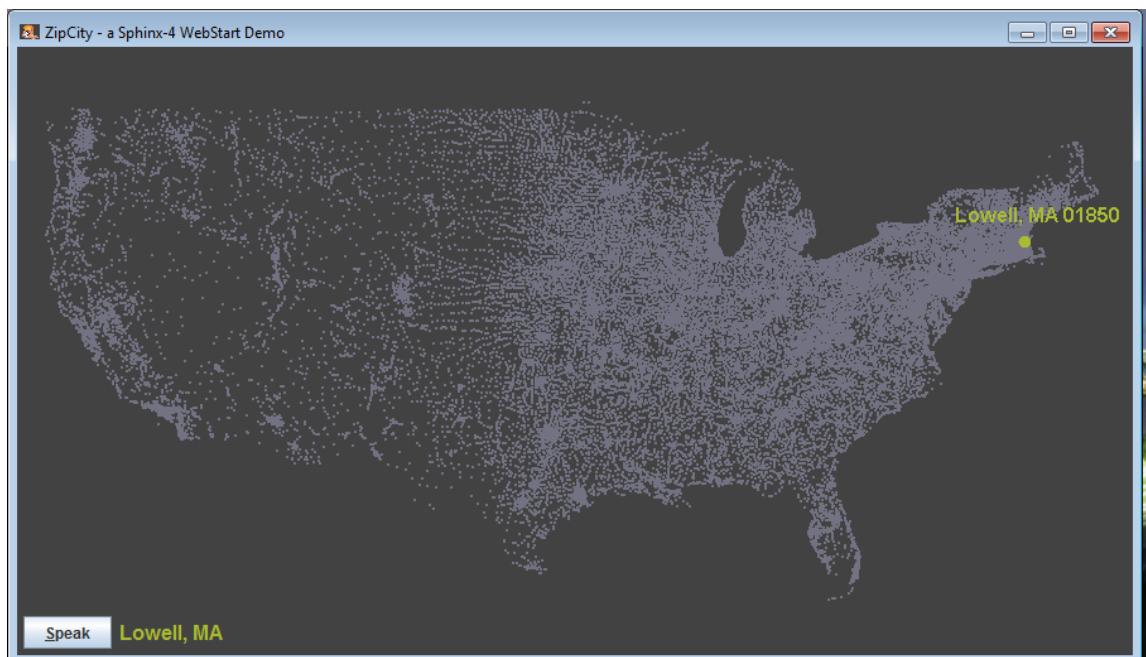


2. Khi không nhận được



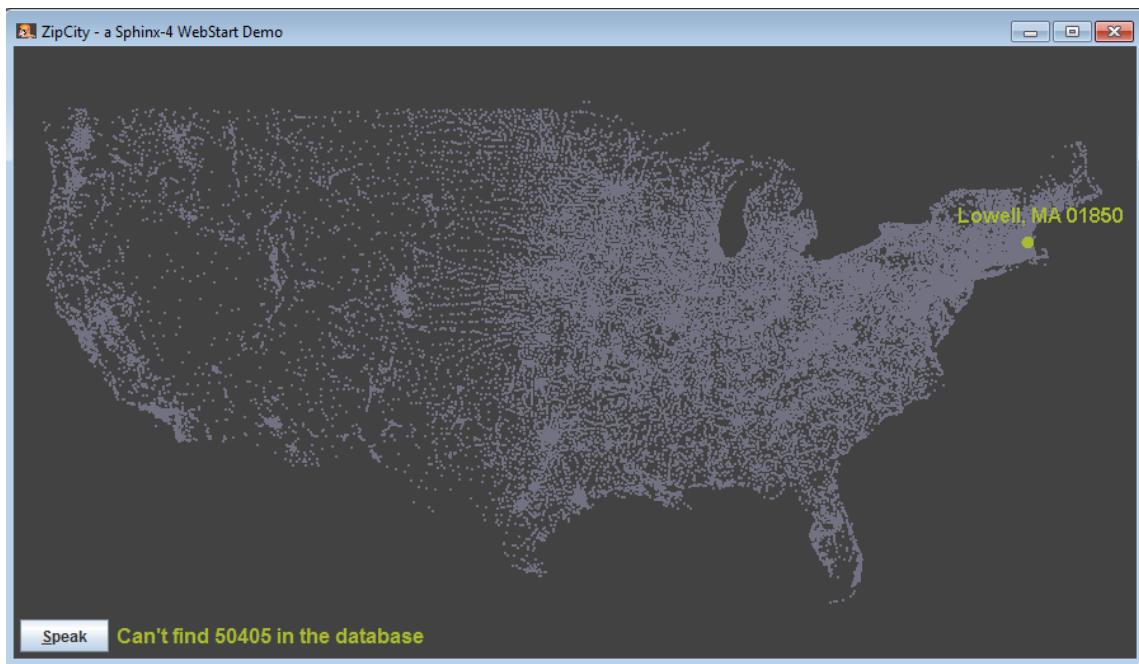
3. Khi nhận dạng đúng :

Nói : “ zero , one , eight, five , zero ”

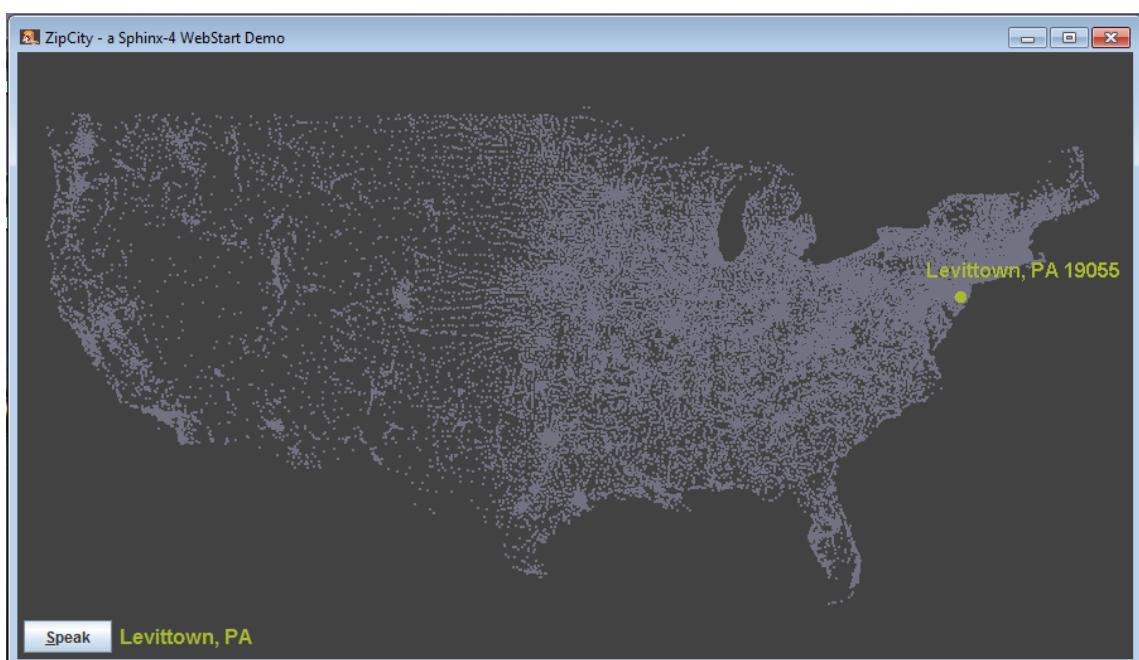


4. Nhận dạng sai :

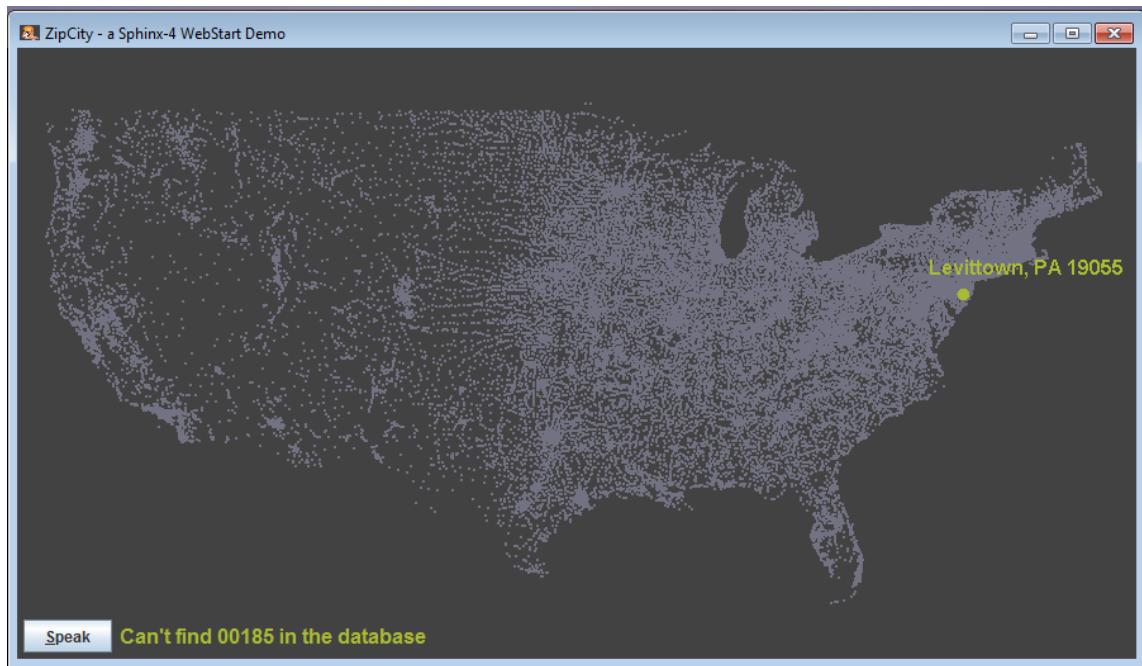
Nói : “one, two , three , four , five ”

**5. Nhận dạng đúng :**

Nói : one – nine – oh – five – five

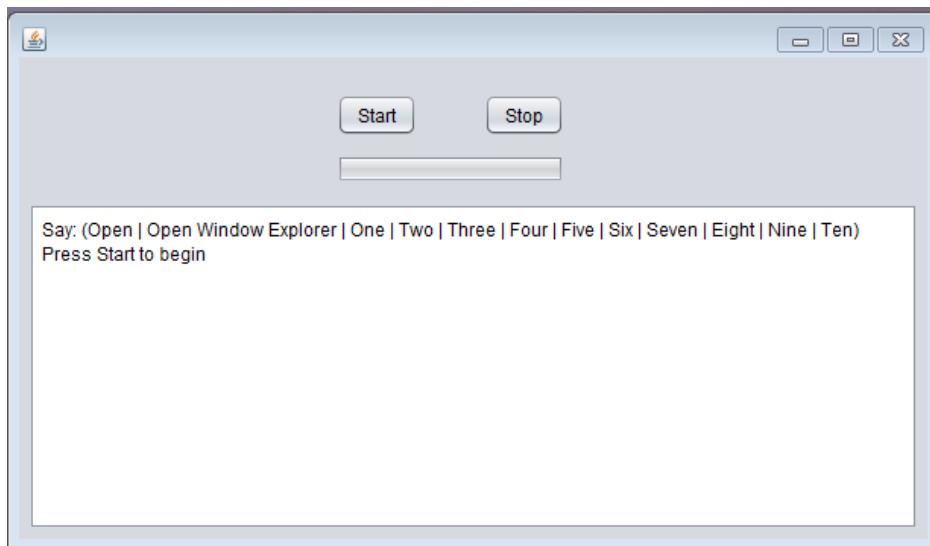
**6. Nhận dạng số không tồn tại**

Nói : oh – oh – one – eight – five

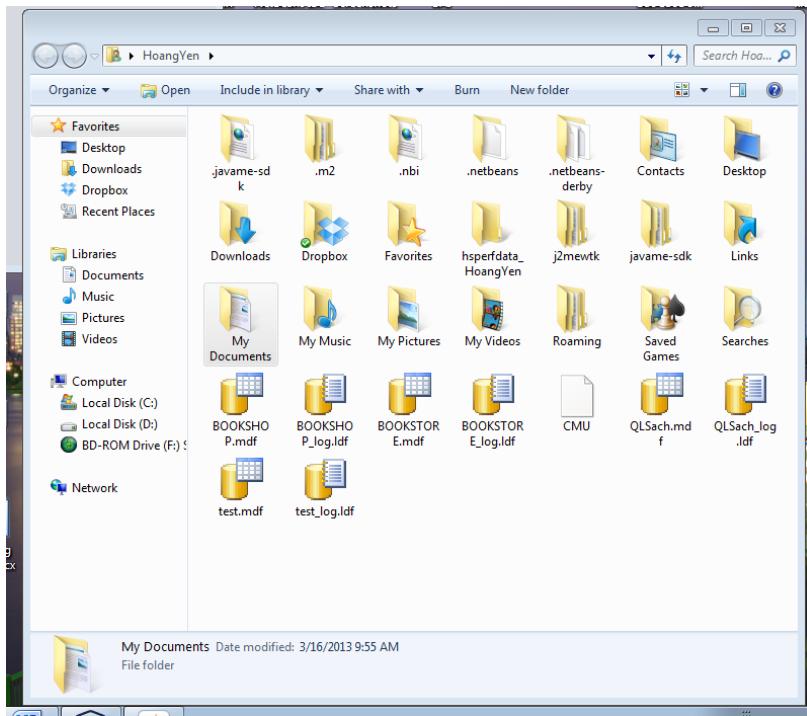


6.2 Demo đọc số - của nhóm

Độ chính xác rất thấp.



Nếu nói chính xác : open hoặc open window , hệ thống sẽ mở cửa sổ window explorer.



6.3 Đánh giá kết luận

6.3.1 Khẳng định giá trị của đề tài

CMUSphinx là một đề tài không phải mới lăm, nhưng lại vô cùng có giá trị thực tiễn. Để theo kịp sự phát triển của công nghệ phần cứng, công nghệ phần mềm cũng cần một bước nhảy vọt. Trong đó, việc phát triển trong khả năng giao tiếp với máy tính, mang máy tính đến với gần con người hơn.

6.3.2 Kết quả đạt được

Về kiến thức, chúng em đã được mở rộng thêm hiểu biết của mình về một mã nguồn mở giàu tiềm năng, có thêm kiến thức trong hành trang vào đời. CMUSphinx là một mã nguồn mở giàu tiềm năng, đưa sự giao tiếp của con người và máy tính lên một tầm cao mới, hướng tới sự phát triển của công nghệ phần mềm.

Về nội dung, chúng em hiểu được cơ chế làm việc của mô hình Markov ẩn, xương sống của Sphinx, là ý tưởng hay nhất trong thời điểm hiện tại, đồng thời hiểu được cách thức hoạt động của CMUSphinx và chi tiết cấu trúc, các package, class, hàm quan trọng chủ chốt. Qua đó, có khả năng tìm

hiểu và phát triển thêm các mô hình ngôn ngữ tiếng Việt để viết các ứng dụng thực tế cho người Việt.

Về ứng dụng, hiểu được các ứng dụng demo đơn giản đính kèm trong gói download của Sphinx4, đọc được code của các demo, và biết được cách để xây dựng một ứng dụng tương tự. Xây dựng lại được các ứng dụng demo đơn giản trên nền Java Application với Sphinx4. Hiểu được cách thức hoạt động của các package trong Sphinx4, cho chúng em khả năng phát triển các ứng dụng lớn hơn trong tương lai gần.

6.4 Tiềm năng và hướng phát triển

Việc phát triển CMUSphinx để nhận dạng tiếng Việt là một đề tài đáng quan tâm, bởi lẽ hiện nay các ứng dụng nhận dạng tiếng Việt còn rất ít.

Mặt khác, cuối năm 2013, Intel sẽ cho ra mắt một loại microphone có khả năng loại bỏ tiếng ồn và chip tích hợp ứng dụng nhận dạng giọng nói bằng tiếng anh. Như vậy, việc điều khiển các thiết bị vi tính bằng giọng nói sẽ trở nên càng dễ dàng hơn. Đó là một đề tài phát triển với tiềm năng vô cùng to lớn và được công nhận bởi nhiều chuyên gia của tập đoàn công nghệ hàng đầu trên toàn thế giới. Có thể thấy trong tương lai không xa, các ứng dụng điều khiển bằng tay sẽ sớm trở nên lạc hậu. Vậy nên, việc nghiên cứu nhận dạng giọng nói tiếng Việt đã trở nên bức thiết hơn bao giờ hết.

Sắp tới, nhóm em cũng sẽ cố gắng phát triển thêm về ứng dụng nhận dạng giọng nói với các tính năng điều khiển sử dụng tiếng Việt với các chức năng cơ bản như: mở trình duyệt internet, mở window explorer, và các chức năng khác. Ngoài ra, chúng em dự định thêm chức năng tạo menu điều khiển do người dùng định nghĩa, có thể chọn các ứng dụng ưa thích để đưa vào menu, ứng dụng trở thành một plug-in chạy kèm với window, khi người dùng muốn mở một ứng dụng, chỉ cần kích hoạt và điều khiển việc mở ứng dụng yêu thích bằng giọng nói.

6.4.1 Đánh giá

❖ **Ưu điểm**

Về CMUSphinx:

- Sự ra đời của Sphinx là một bước tiến quan trọng để đưa con người tiến gần hơn với sự hiện đại, khi mà việc điều khiển máy móc trở nên đơn giản hơn bao giờ hết.
- CMUSphinx là một framework mã nguồn mở , hoàn toàn miễn phí, việc nghiên cứu trở nên dễ dàng hơn nhờ sự miễn phí đó.
- Các phần mềm sản phẩm sử dụng Sphinx đa phần sẽ có giá trị cao trong thực tế.
- CMUSphinx thân thiện, có thể dễ dàng chạy được trên nhiều hệ điều hành nhờ vào việc được phát triển trên nền java. Và việc xây dựng ứng dụng trở nên đơn giản hơn bởi vì CMUSphinx được viết bằng ngôn ngữ Java với phiên bản Sphinx4, và bằng ngôn ngữ C với phiên bản pocketSphinx . Ngoài ra, Sphinx còn có thể viết các ứng dụng trên iOS và Android.
- Tiềm năng phát triển to lớn tạo nên một sức hút cho Sphinx để ngày càng có nhiều thành viên tham gia vào cộng đồng Sphinx, cùng xây dựng và phát triển để framework này trở nên hoàn thiện hơn. Trong tương lai, việc nghiên cứu CMUSphinx có thể sẽ trở nên dễ dàng hơn.

Về demo của nhóm :

- Demo đơn giản , dễ hiểu , thể hiện được các cơ chế vận hành cơ bản của Sphinx
- Có giá trị tham khảo để phát triển các ứng dụng cao cấp hơn

❖ **Khuyết điểm :**
Của Sphinx :

- Về mô hình , việc cấu hình cho Sphinx còn khá phức tạp và dễ phát sinh lỗi.
- Bản ghi âm cho việc huấn luyện chỉ thực hiện được với một số định dạng nhất định.
- Thời gian để huấn luyện quá nhiều để rút ra được một cơ sở dữ liệu nhỏ. Mong rằng CMUSphinx sẽ sớm phát triển được một thuật toán hay hơn để thu được nhiều dữ liệu có giá trị hơn
- Việc nhận dạng còn chưa chính xác vì chưa thật sự lọc được các tạp âm, chưa phân biệt được nhiều khi người dùng phát âm chưa chuẩn , hoặc các tiếng với kéo dài, và khi giọng của người nói có vấn đề như bị khản, quá trầm hay quá cao, sự thăng trầm trong câu nói...
- Lớp endpointer chưa thực sự tốt, việc xác định người dùng đã kết thúc câu nói thường không đạt kết quả mong đợi, do Sphinx còn chưa có khả năng xác định âm giọng của người đang nói để việc tách tiếng dựa trên tần số âm thanh trở nên chính xác hơn.
- Việc nghiên cứu còn gặp nhiều khó khăn vì nguồn tài liệu còn hạn chế, cộng đồng CMUSphinx còn chưa được sôi động. Các diễn đàn còn ít và ngay chính diễn đàn của CMUSphinx cũng ít được hỗ trợ, các câu hỏi của nhóm đặt ra cho CMU vẫn không nhận được giải đáp mặc dù đã được đưa ra hơn 4 tháng.

Của nhóm :

- Các tính năng còn quá đơn giản
- Nhận dạng chưa được chính xác (do tạp âm, tiếng ồn, người nói phát âm không chuẩn)
- Việc nghiên cứu và ứng dụng còn hạn chế (do trình độ còn giới hạn, thiếu kinh nghiệm, nhiều vấn đề còn chưa thực sự hiểu và áp dụng được)
- Chưa thực hiện được việc huấn luyện và nhận dạng bằng tiếng việt (do nhóm còn chưa làm được huấn luyện nên chưa thể tạo được cơ sở dữ liệu để nhận dạng tiếng việt)
- Chương trình khi nhận dạng lần đầu mất khá nhiều thời gian vì bước khởi tạo và cấu hình chưa được tốt.
- Việc áp dụng những thành quả nghiên cứu chưa được tốt. Nhiều vấn đề hiểu cơ chế nhưng vẫn chưa áp dụng thực tế được trong demo ứng dụng (do kiến thức và trình độ viết code còn yếu , chưa có kinh nghiệm)

TÀI LIỆU THAM KHẢO

- + Các packages và class chính được dùng.
 - <http://cmusphinx.sourceforge.net/sphinx4/javadoc/index.html?overview-summary.html>
- + Download các bản mới nhất từ CMUSphinx.
 - <http://cmusphinx.sourceforge.net/wiki/download/>
- + Training Acoustic Model For CMUSphinx
 - <http://cmusphinx.sourceforge.net/wiki/tutorialam>
- + A speech recognizer written entirely in the JavaTM programming language
 - <http://cmusphinx.sourceforge.net/sphinx4/>
- + Sphinx Knowledge Base Tool -- VERSION 3
 - <http://www.speech.cs.cmu.edu/tools/lmtool-new.html>
- + Download Voxforge
 - <http://www.voxforge.org/home/downloads>
- + Sphinx-4 Application Programmer's Guide
 - <http://cmusphinx.sourceforge.net/wiki/tutorialsphinx4>
- + Configuration Management for Sphinx-4
 - <http://cmusphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/util/props/doc-files/ConfigurationManagement.html>
- + How to use SphinxTrain
 - <http://ronaldramdhan.wordpress.com/2010/03/11/sphinxtrain/>
- + Creating a text corpus from Wikipedia
 - <http://trulymadlywordly.blogspot.ru/2011/03/creating-text-corpus-from-wikipedia.html>
- + The CMU Statistical Language Modeling (SLM) Toolkit
 - http://www.speech.cs.cmu.edu/SLM_info.html
- + SPEECH and LANGUAGE PROCESSING

- <http://www.cs.colorado.edu/~martin/slp.html>
- ✚ Sphinx Knowledge Base Tool
 - <http://www.speech.cs.cmu.edu/tools/lmtool-adv.html>
- ✚ The CMU Audio Databases
 - <http://www.speech.cs.cmu.edu/databases/>
- ✚ festvox
 - <http://www.festvox.org/index.html>
- ✚ Constant Field Values
 - http://cmusphinx.sourceforge.net/sphinx4/javadoc/constant-values.html#edu.cmu.sphinx.jsgf.JSGFGrammar.PROP_BASE_GRA_MMAR_URL
- ✚ LOGIOS Lexicon Tool
 - <http://www.speech.cs.cmu.edu/tools/lextool.html>
- ✚ Adapting the default acoustic model
 - <http://cmusphinx.sourceforge.net/wiki/tutorialadapt>
- ✚ The CMU Pronouncing Dictionary
 - <http://www.speech.cs.cmu.edu/cgi-bin/cmudict>

