САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ – ПРОЦЕССОВ УПРАВЛЕНИЯ

Попов Никита Алексеевич

Магистерская диссертация

Исследование задачи диаризации аудиопотока в режиме реального времени

Направление 03.04.01

Прикладная математика и физика

Магистерская программа «Прикладная информатика»

Руководитель магистерской программы,
д.ф.-м.н.,
профессор Егоров Н. В.

Научный руководитель,
доктор техн. наук,
доцент Печников А. А.

Рецензент,
кандидат техн. наук,
доцент Крижановский А. А.

Санкт-Петербург

2020

Содержание

Введение

В последние годы произошло многократное увеличение числа устройств, собирающих и хранящих данные (в частности, аудиозаписи окружающей эти устройства среды): смартфоны, камеры видеонаблюдения, элементы "умного дома". В связи с этим стала актуальной проблема точной и быстрой обработки большого объёма подобных данных.

Грамотная обработка этих данных позволяет упростить взаимодействие с устройствами и свести его только к управлению голосом. Поэтому актуальной стала задача выделения определённых частей аудиозаписей, например, человеческой речи, в особенности голосов отдельных участников диалога. Результаты подобного распознавания, или голосовой идентификации диктора, востребованы в таких областях, как биометрический поиск, голосовая верификация пассажиров и водителя, разграничение прав доступа к информации с помощью голосовой биометрии и т. д.

Корректная диаризация позволяет лучше приспособиться к особенностям произношения и акценту и качественно разделить высказывания разных людей. Технология находит применение, в частности, в создании субтитров к видеозаписям. Правильно распознанную речь легче перевести на другие языки, что, например, полезно для обучающих онлайн курсов. А возможность обрабатывать звук в реальном времени позволит делать это даже в прямом эфире.

Важным достоинством таких систем по отношению к другим биометрическим системам идентификации является их дешевизна. Важно также, что современные СГИД (системы голосовой идентификации диктора) по уровню надёжности идентификации не уступают, а иногда и превосходят, к примеру, системы идентификации человека по изображению. Эволюция систем распознавания речи привела к созданию

интеллектуальных систем, позволяющих не только распознавать, но и автоматически синтезировать человеческую речь. Несмотря на уникальность голоса человека, ни одна из СГИД, как и любая другая биометрическая система, не может гарантировать 100% надёжность идентификации. Основными источниками ошибок в системах голосовой идентификации являются:

- окружение (шум, реверберация и т. д.)
- особенности речи (длительность, тональность, уровень голосового усилия и т. д.)
- канал связи (искажения микрофона и канала передачи, погрешности кодирования аудио сигнала и т. д.)

Постановка задачи

В качестве научно-исследовательской работы предлагается исследовать необходимые характеристики и методы, с помощью которых возможно реализовать алгоритм, позволяющий эффективно определять участки аудиопотока, содержащие в себе человеческую речь, и отделять отрезки, принадлежащие различным дикторам.

Подразумевается, что в дальнейшем полученные результаты будут использоваться алгоритмами распознавания текста из аудиодорожек (speech-to-text), которые в последнее время показывают успешные результаты. В данной работе не предполагается распознавание текста в речи как такового, но отдельно стоит учитывать тот факт, что полученные аудиодорожки должны содержать как можно меньше шумов, т. к. незначительные отклонения аудиодорожки от "оригинала" могут повлиять на качество дальнейшей обработки.

Для подтверждения эффективности описанного алгоритма предполагается практическая его реализация в виде программного решения с использованием актуальных open-source библиотек и возможностью гибкой настройки под нужды пользователя. Дополнительным требованием является возможность производить распознавание в режиме реального времени, т.е. с минимально возможной задержкой между получением алгоритмом входного сигнала и выводом результатов обработки.

В силу того, что входные данные предполагаются как аудиопоток в режиме реального времени, необходимо предусмотреть модуль предварительной обработки данных. Помимо общей для всех подобных систем проблемы частой смены дикторов, дополнительные трудности в данных условиях добавляют отсутствие какой-либо априорной информации о количестве дикторов на фонограмме и отсутствие информации о личности дикторов, образцах их голоса и даже пола.

Также необходимо учесть возможность запуска модели на мобильных устройствах, в том числе элементах «умного дома», поэтому реализация должна учитывать соответствующие ограничения. Система должна быть достаточно точной в распознавании, при этом нетребовательной к ресурсам устройства, на котором она запущена. Зачастую даже самая тривиальная команда, например, «запусти мою любимую музыку», может потребовать от устройства не только одновременной диаризации и идентификации пользователя, но также и обращения к различным приложениям, в том числе онлайн сервисам.

Полученная реализация должна быть сравнена с другими имеющимися решениями данной проблемы в ходе исследования с использованием заранее составленных датасетов.

Цель данной работы – исследовать задачу диаризации аудиопотока, сформулировать и практически реализовать алгоритм выделения речи отдельных людей из аудиопотока в режиме реального времени.

Обзор литературы

Задача диаризации аудиопотока состоит в выделении сегментов речи, принадлежащих тому или иному говорящему на аудиозаписи. Формально эта проблема известна как разделение источников звука или разделение сигнала (audio source separation) [1]. Она заключается в восстановлении или реконструкции одного или нескольких исходных сигналов, которые в результате линейного или свёрточного процесса смешаны с другими сигналами. У этой области исследований много практических применений, в том числе улучшение качества звука (речи) и устранение шума, музыкальные ремиксы, пространственное распределение звука, ремастеринг и т. д. Звукоинженеры иногда называют эту технику расслоением (demixing).

Важно учитывать, что условие на распознавание в реальном времени добавляет некоторые сложности. Например, необходимо разбивать поступающий сигнал на небольшие отрезки и обрабатывать уже их, а не всю аудиозапись. При этом допускается некоторая задержка между вводом данных и получением результата, но она должна быть незначительной. Большинство работ не ориентируются на это условие, но основные методы распознавания речи и верификации дикторов могут успешно применяться независимо от этого.

Также большинство работ ориентируются на методы, предполагающие наличие «учителя» либо некоторых заранее определённых значений характеристик, с которыми можно сравнивать поступающий аудиопоток. В рамках данной работы предполагается, что данные условия невыполнимы.

Speaker diarization: a review of recent research

Работа [2] описывает основные подходы к решению данной задачи и рассматривает актуальные в 2010 году алгоритмы. Данная статья

предлагает весь необходимый объём информации для знакомства с изучаемой проблемой.

Авторы отмечают различные подходы к обработке данных в зависимости от основных рассматриваемых типов аудиозаписей: новостных фонограмм или записей различных конференций (совещаний, семинаров и т. д.). В проблеме диаризации авторами выделяются следующие подзадачи, необходимые для её успешного решения:

- Акустическое формирование луча (acoustic beamforming)
- Обнаружение речевой активности (Speech Activity Detection, SAD)
- Сегментация (segmentation)
- Кластеризация (clustering)
- Пошаговая сегментация и кластеризация для обработки в режиме реального времени (one-step segmentation and clustering)

Актуальными направлениями исследований авторы называют:

- Использование не-временных характеристик (таких как положение дикторов относительно микрофона)
- Использование просодических свойств речи (ритм, интонация, ударение)
- Распознавание наложения речи нескольких дикторов
- Аудиовизуальная диаризация (т. е. с использованием характеристик, извлекаемых одновременно из аудио- и видеопотока)

Отмечается, что наибольшего успеха на момент выхода работы достигли системы диаризации, основанные на применении скрытых марковских моделей (Hidden Markov Models, HMMs) для смешанных гауссовских моделей (Gaussian Mixture Models, GMMs), соответствующих различным дикторам.

Speaker change detection in broadcast TV using bidirectional Long Short-Term Memory networks

Обнаружение смены дикторов авторы [3] пытаются решить аналогично задаче маркировки двоичных последовательностей. Для этого ими применяются двунаправленные сети долговременной кратковременной памяти (Bi-LSTM). В качестве данных, подлежащих анализу, исследователи используют набор из аудиозаписей новостных вещаний французского ТВ.

Авторы отмечают, что предыдущие исследования предлагали использование TristouNet, в которой используется евклидово расстояние. Такие системы имеют тенденцию пропускать границы при быстром взаимодействии дикторов из-за относительно длинных соседних скользящих окон (2 секунды или более).

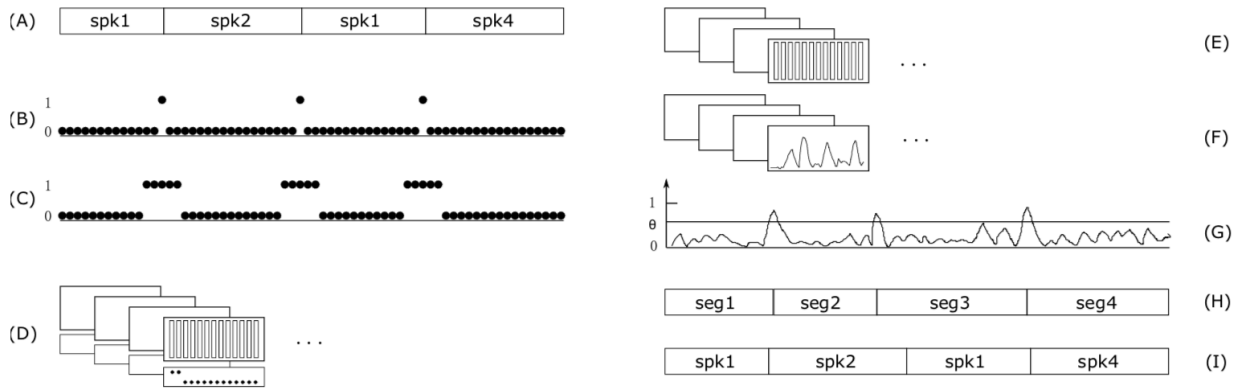


Рисунок 1. Демонстрация отдельных этапов диаризации

В качестве входных данных используются мел-кепстральные коэффициенты (MFCC) [1], которые получаются из перекрывающихся окон нарезки, на выходе алгоритм выдаёт двоичный класс, обозначающий принадлежность сегмента диктору (рис. 1). Система использует для обучения двоичную функцию потерь при перекрестной энтропии (binary cross-entropy loss).

- Bi-LSTM позволяют обрабатывать последовательности в прямом и обратном направлении, используя как прошлый, так и будущий контексты.
- Для решения классового дисбаланса количество положительных меток искусственно увеличивается путем маркировки как положительных каждого кадра в непосредственной близости от точки изменения с ручной аннотацией. Такой близостью считается положительное соседство в 100 мс (50 мс с обеих сторон) вокруг каждой точки изменения.
- Длинные аудиопоследовательности разбиваются на короткие перекрывающиеся последовательности. Они имеют фиксированную длину 3.2 секунды с шагом 800 мс (т. е. пересекаются на 25%)

Neural speech turn segmentation and affinity propagation for speaker diarization

В работе [4] задача диаризации делится на 4 подзадачи:

- Обнаружение речевой активности (Speech Activity Detection, SAD)
- Обнаружение смены дикторов (Speaker Change Detection, SCD)
- Кластеризация речи (Speech Turn Clustering)
- Пересегментация (Re-segmentation)

Как отмечается авторами работы, первые две подзадачи уже решаются с помощью рекуррентных нейронных сетей, поэтому авторы направляют свои усилия на поиск различных улучшений для остальных подзадач. Основные достижения авторов работы:

- Адаптация SAD и SCD на основе LSTM с неконтролируемой пересегментацией (без учителя). Ранее GMM (Gaussian mixture model) обучалась для каждого кластера (речевые сегменты,

которые включают в себя одного и того же диктора) применением алгоритма свёрточного декодирования Витерби для каждого сегмента. [2]

- Применение кластеризации методом распространения близости (affinity propagation) для нейронных встраиваний (вложений, embeddings) дикторов. (В контексте нейронных сетей встраивания являются низкоразмерными, обучаемыми непрерывными векторными представлениями дискретных переменных. Нейросетевые встраивания полезны, так как могут уменьшить размерность категориальных переменных и осмысленно представлять категории в преобразованном пространстве [5]).
- Для всех описанных подзадач, кроме кластеризации, были предложены решения, основанные на рекуррентных нейронных сетях (RNN).

Подтверждение гипотез и измерение метрик производилось на датасете, состоящем из записей эфира французского телевидения (более 60 часов и 1700 дикторов). Алгоритм показал лучшие результаты по сравнению со схожими алгоритмами диаризации (рис. 2).

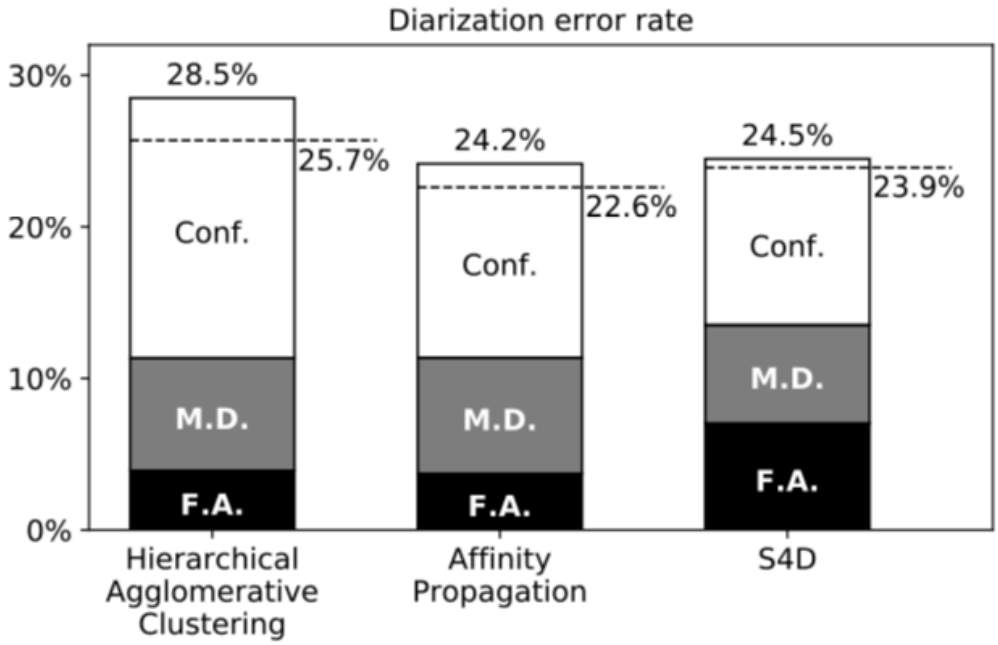


Рисунок 2. Сравнение ошибки DER для разных алгоритмов

Developing on-line speaker diarization system

Работа [6] команды исследователей из компании IBM посвящена разработке системы для проведения диаризации в режиме реального времени. Предложенная авторами общая архитектура модели представлена на рис. 3.

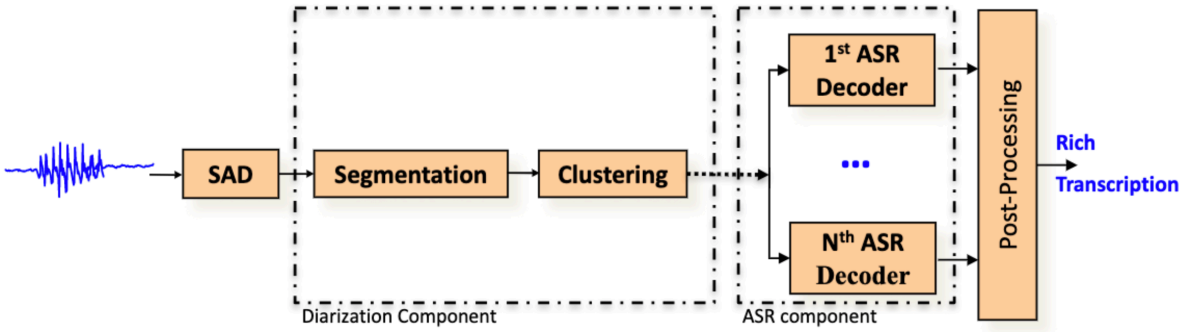


Рисунок 3. Архитектура модели

В ходе тестов авторы установили, что модели, использующие MFCC для извлечения i-векторов [7], показывают бóльшую эффективность в сравнении с решениями, использующими PLP (Perceptual Linear Prediction).

В качестве алгоритма кластеризации используется X-means – модификация алгоритма K-means [8].

В рамках работы был реализован и практически применён алгоритм диаризации аудиопотока в режиме реального времени. Система устойчива к неречевым участкам аудио и позволяет обрабатывать поток без заметных задержек.

Speaker diarization using deep neural network embeddings

В работе [9] авторы предлагают альтернативный подход к обучению представлений дикторов через глубокие нейронные сети (deep neural networks), чтобы полностью удалить извлечение i-вектора из процесса. Предложенный алгоритм представляет дискриминационно обучаемый DNN, который заменяет двухступенчатый генеративный процесс i-векторной системы диаризации.

Предлагаемая архитектура одновременно обучает встраивания фиксированной размерности для акустических сегментов переменной длины и функцию подсчета очков (рис. 4) для измерения вероятности того, что сегменты принадлежат одному и тому же или разным дикторам. Проводя тесты для разговорного телефонного речевого корпуса CALLHOME, они демонстрируют, что в дополнение к оптимизации архитектуры диаризации, предлагаемая система сравнима или эффективнее актуальных на тот момент решений. Исследователи также отмечают, что этот подход сильно зависит от начальных данных, т.к. является алгоритмом обучения с учителем.

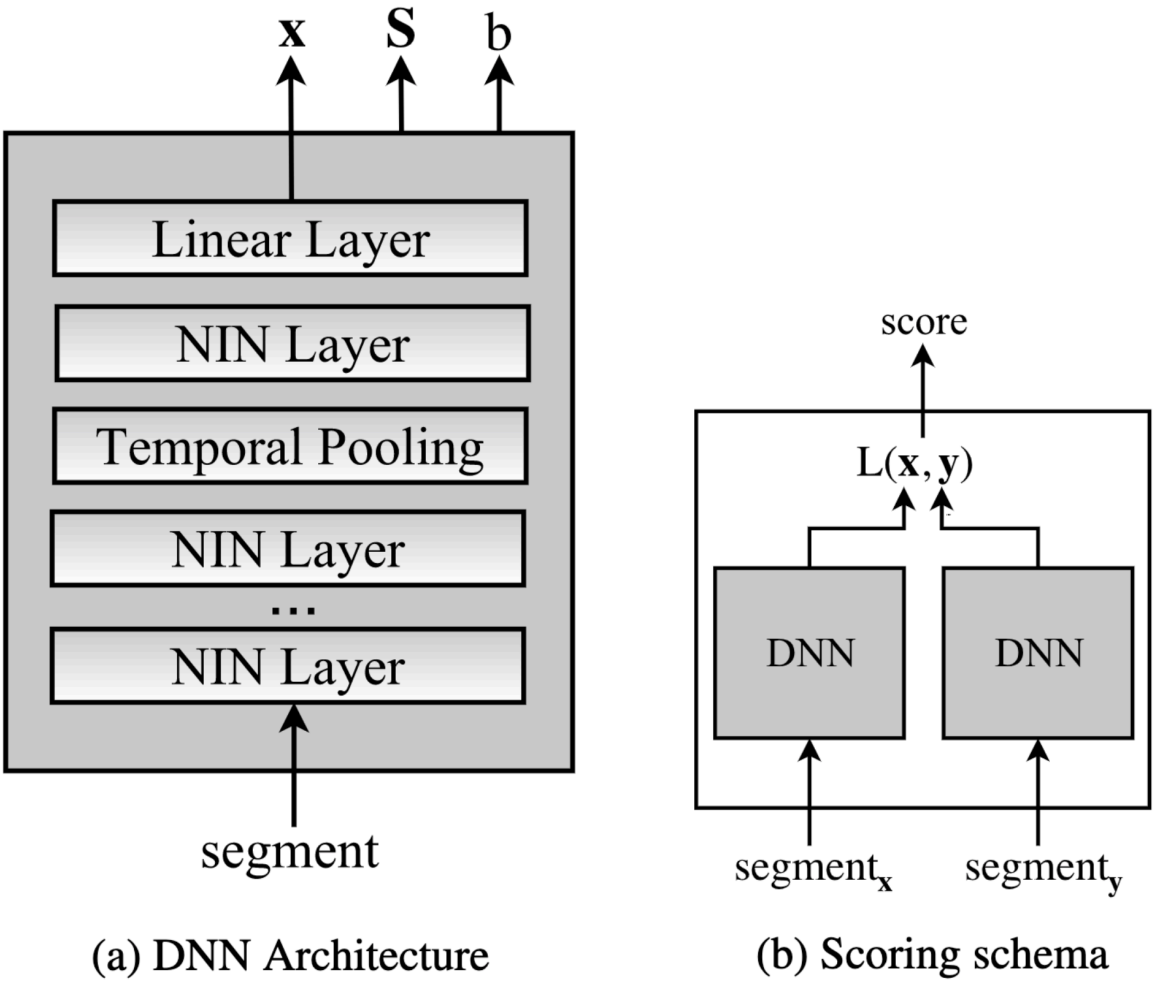


Рисунок 4. Архитектура DNN (a) и функция оценивания (b)

Deep learning approaches for online speaker diarization

В последнее десятилетие всё больше работ применяют глубокое обучение (deep learning) для решения задачи диаризации аудиопотока, например:

- Изучение встраиваний (embeddings) дикторов и использование их для классификации.
- Представление дикторов с помощью i-векторов (i-vectors)
- Bi-LSTM RNN

В исследовании [10] авторы применяют различные стратегии для решения этой проблемы.

- Встраивания дикторов строятся с использованием триплетной потери, что схоже с моделью Facenet для распознавания лиц. Предложенная модель используется для тренировки LSTM (рис. 5), которая может эффективно закодировать встраивания для речевых сегментов используя триплетную потерю (triplet loss)

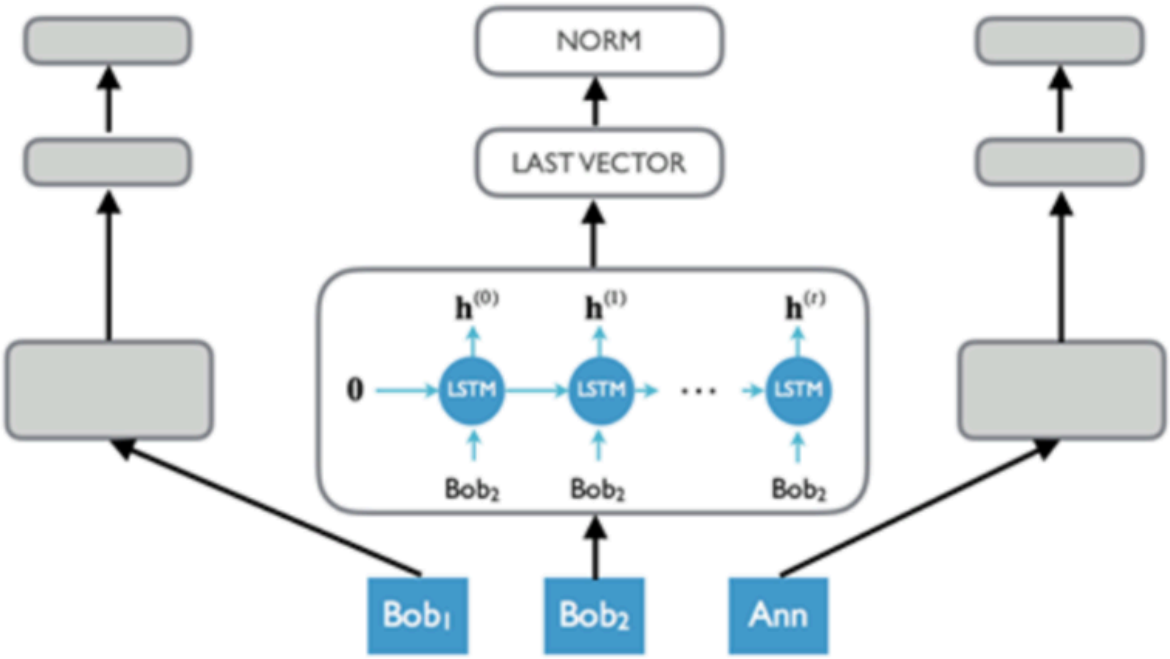


Рисунок 5. Предложенная модель

При этом данная система может пропустить некоторые изменения дикторов (если они происходят слишком часто и, следовательно, участки речи одного диктора имеют небольшую длину).

Speaker diarization with LSTM

Работа [11] основывается на успешном применении систем подтверждения диктора на основе d-векторов для разработки нового решения задачи диаризации. Предлагаемая система является комбинацией следующих методов:

- основанная на LSTM модель подтверждения диктора для извлечения встраиваний диктора (d-векторов)

- непараметрическая спектральная кластеризация. Авторы применяют алгоритм кластеризации к полученным встраиваниям для диаризации спикера.

Применение этих методов (рис. 6) даёт одно из наиболее успешных решений рассматриваемой проблемы.

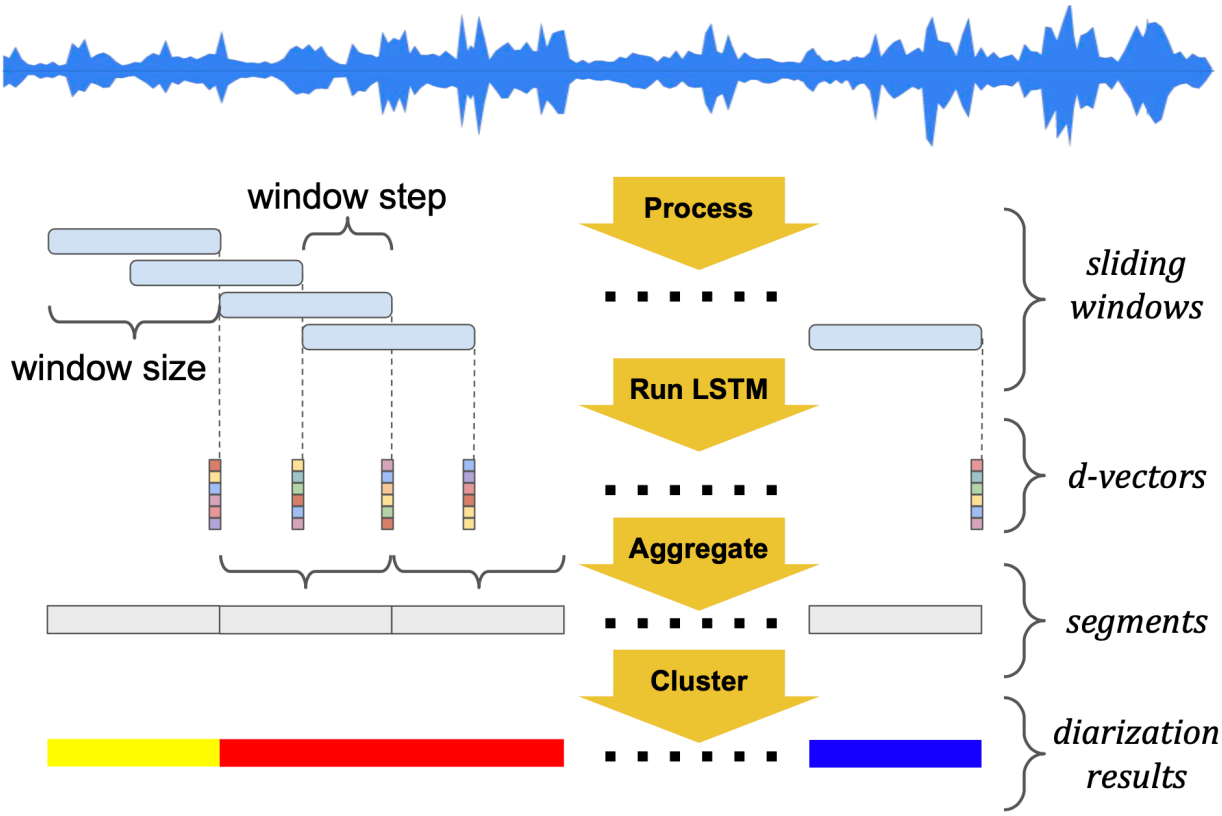


Рисунок 6. Архитектура алгоритма

В работе рассматриваются четыре различных алгоритма кластеризации. Два из них принадлежат к кластеризации в режиме реального времени (система размечает фреймы как только они становятся доступными, не учитывая остальные). Остальные два относятся к алгоритмам автономной кластеризации (система размечает фреймы когда доступна вся аудиозапись). Автономная кластеризация превосходит онлайновую по эффективности:

- Naive online clustering
- Links online clustering

- K-means offline clustering
- Spectral offline clustering (наилучшая по результатам тестов)

Авторы объясняют почему некорректно использовать обычные алгоритмы кластеризации, такие как K-means. Проблема исходит из свойств речевых данных:

- Негауссово распределение: речевые данные часто не являются гауссовыми.
- Кластерный дисбаланс: для большинства записей в основном говорит один диктор. И если мы используем K-means, то, к сожалению, он может разделить этот кластер на несколько мелких.
- Иерархическая структура: Разница между одним мужчиной и одной женщиной говорит больше, чем разница между двумя мужскими кластерами. Это свойство, обычно приводит к тому, что алгоритм K-means относит все вложения мужчин в один кластер и все вложения женщин в другой кластер.

Поэтому они предлагают новую непараметрическую спектральную автономную кластеризацию для решения этих проблем. Применение описанной системы предоставляет низкий процент ошибок, по сравнению с аналогичными решениями (рис. 7).

Embedding	**Clustering**	**CALLHOME American English Eval**				**NIST RT-03 English CTS Eval**			
		Confusion	FA	Miss	Total	Confusion	FA	Miss	Total
i-vector	Naive	26.41	2.40	3.55	32.36	35.35	4.66	2.62	42.63
	Links	25.40			31.36	33.56			40.48
	K-Means	22.86			28.81	24.38			31.66
	Spectral	14.59			20.54	13.84			21.12
d-vector	Naive	12.41	1.94	4.51	18.87	18.76	4.09	4.45	27.30
	Links	11.02			17.47	18.56			27.10
	K-Means	7.29			13.75	7.80			16.34
	Spectral	6.03			12.48	3.76			12.30

Рисунок 7. Сравнительные результаты тестов разных архитектур

Generalized End-To-End loss for speaker verification

Работа [12] является продолжением предыдущей [11] от тех же авторов. В ней рассматриваются некоторые улучшения, такие как двухступенчатая система распознавания диктора и функция обобщённой сквозной потери (generalized end-to-end loss).

Так, диктор предварительно регистрирует свой голос, а перед произнесением каждой фразы производит подтверждение своего голоса произнося заданную фразу (“OK Google” или “Hey Google”). После этого модель строит усреднённый вектор встраивания для этого диктора.

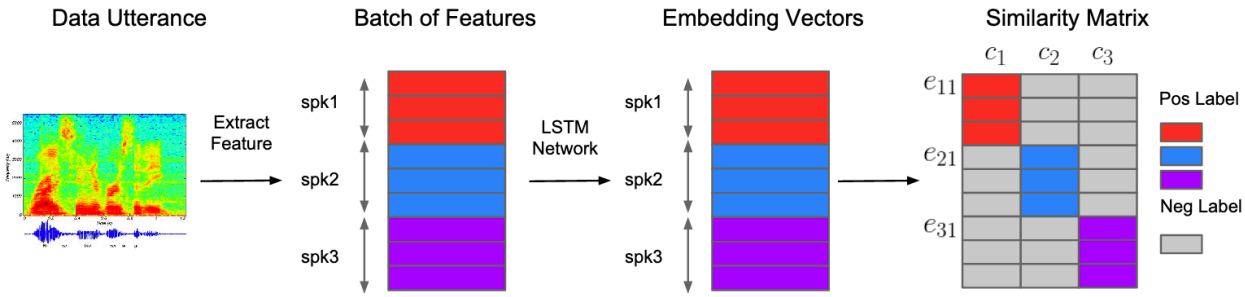
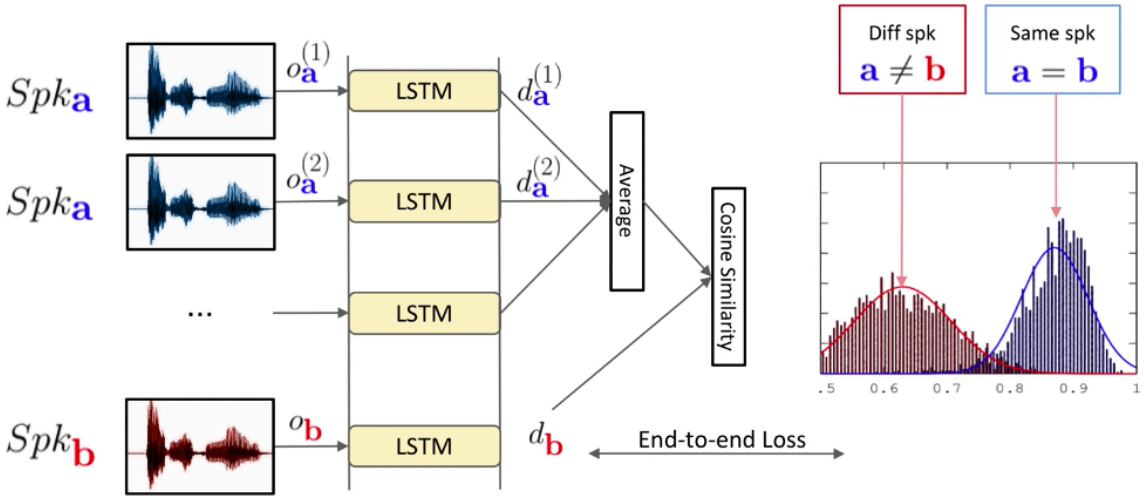


Рисунок 8. Извлечение характеристик и построение матрицы схожести

Для верификации диктора создаются встраивания для входных данных с помощью LSTM (рис. 8). Выходные данные последнего слоя этой сети были названы d-векторами. После этого для полученных встраиваний вычисляется косинусное расстояние. Если вычисленное значение больше заданного порога, то система верифицирует пользователя. Для извлечения встраивания необходимо определить функцию потери.

- Большинство работ используют триплетную потерю (Triplet loss). Она очень проста для применения и может корректно моделировать пространство встраивания, однако не может моделировать поведение во время выполнения. Это означает что она не может моделировать усреднённый процесс. Таким образом она не является сквозной.

- В 2016 году авторами была предложена функция кортежной сквозной потери (tuple end-to-end loss). Она позволяет моделировать процесс усреднения. Однако большинство кортежей очень легко тренируются, что не очень эффективно в применении к данной задаче.



Google

* Heigold, Georg, et al. "End-to-end text-dependent speaker verification." *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016.

Рисунок 9. Процесс определения принадлежности аудиозаписи диктору

Поэтому авторами предлагается функция обобщенной сквозной потери (generalized end-to-end loss). Чтобы обучать модель с этими потерями, для каждого набора строится матрица схожести (similarity matrix) (рис. 9).

Другой проблемой является то, что длина высказывания (соответственно, аудиозаписи) может варьироваться. Очевидное решение – использовать всю фразу, однако это может сказаться на производительности и времени выполнения задачи. К тому же, для решения этой задачи неважен текст, произносимый автором, поэтому его можно не рассматривать, а использовать только частотные свойства речи. Авторы предлагают использовать скользящее окно для выбора отрезков с наложением (sliding window inference) и использовать их при обучении

(рис. 10). Оптимальным значением была принята длина окна 1.6 секунд с наложением 50%.

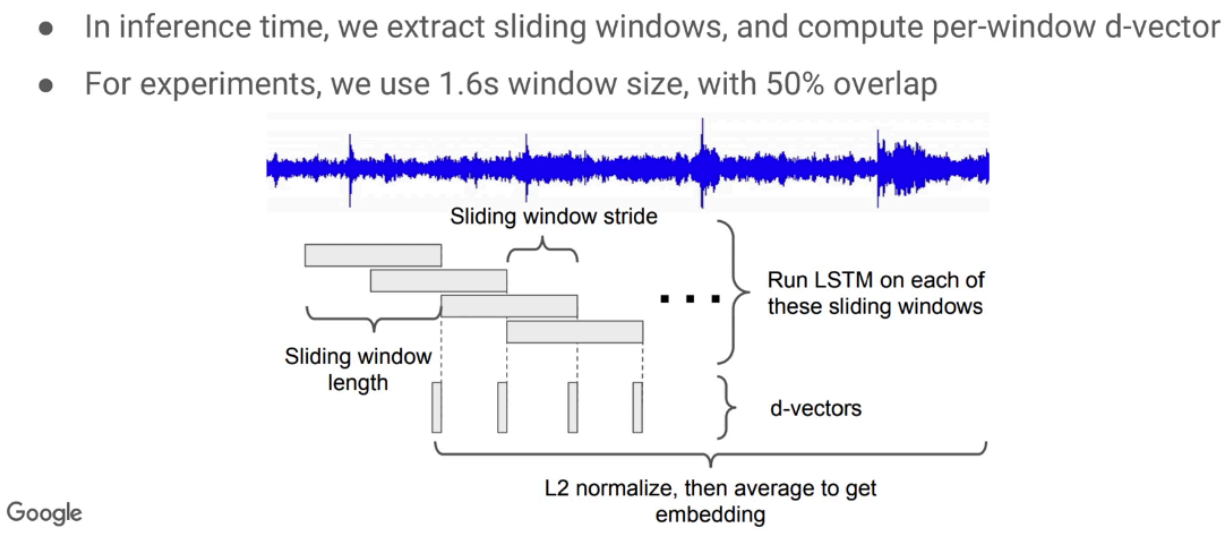


Рисунок 10. Использование скользящих окон для получения d-векторов

Fully supervised speaker diarization

Команде исследователей компании Google также принадлежит работа [13], но, в отличие от предыдущей, в ней рассматривается метод обучения с учителем. При этом алгоритм извлечения d-векторов идентичен предыдущей работе (рис. 6). После извлечения каждый отдельный диктор моделируется с помощью RNN (рекуррентных нейронных сетей) с совместным использованием параметров, в то время как RNN состоит из различных чередований во временной области для разных дикторов. С помощью этого метода системная расшифровка выполняется в режиме реального времени. Кроме того, их метод естественным образом интегрирован с ddCRP (distance dependent Chinese Restaurant Processes) [14]. Таким образом, система может определить, сколько дикторов присутствует на записи.

Данную систему создатели назвали UIS-RNN (unbounded interleaved-state recurrent neural networks). Пример процесса диаризации (определения принадлежности встраивания тому или иному диктору) с помощью представленного алгоритма показан на рис. 11.

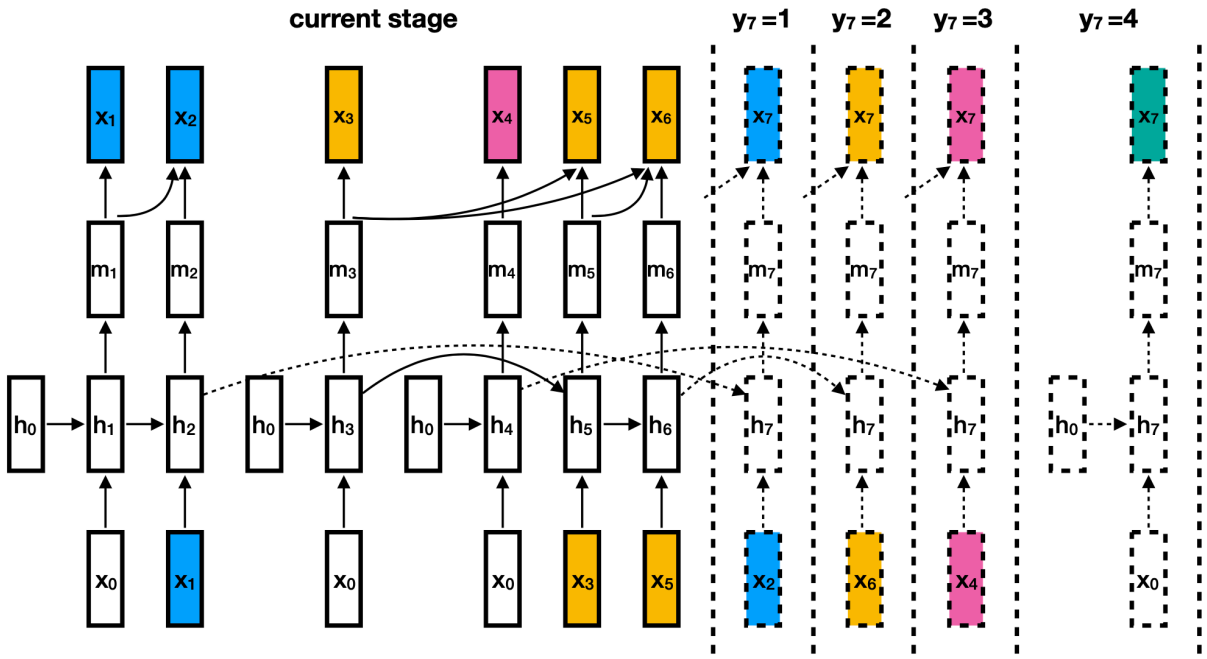


Рисунок 11. Пример диаризации с использованием UIS-RNN

Разработчики проверили эффективность нового алгоритма диаризации с помощью теста NIST SRE 2000 CALLHOME. Погрешность определения DER составила 7,6%. Использовавшиеся ранее методы кластеризации и выделения с помощью нейронной сети показывали погрешность 8,8% и 9,9% соответственно. Помимо меньшего количества ошибок алгоритм обладает производительностью, достаточной для обработки потока в реальном времени. В качестве параметров используется структура d-векторов, являющаяся внутренней разработкой компании Google. Её официальная реализация недоступна в открытом доступе.

Глава 1. Исследование задачи

Диаризация дикторов – это задача определения "кто когда говорил" в аудиопотоке, который обычно содержит неизвестное количество речи от неизвестного количества дикторов. Одними из наиболее сложных условий для решения данной задачи являются частая смена дикторов, отсутствие какой-либо априорной информации о количестве дикторов на фонограмме и отсутствие информации о личности дикторов, образцах их голоса и даже пола.

Следует отметить, что в рамках данной задачи выделяют [2, 15] 3 основных типа аудиозаписей, подлежащих анализу:

1. Аудиозаписи новостных радио- и телевещаний.
   Для них характерно неизвестное количество дикторов, а также их редкая смена (не чаще 3 секунд).
2. Фонограммы совещаний или телефонных переговоров в моноканале.
   Для них типичны частая смена дикторов и возможность наложения их речи. Данная работа рассматривает именно этот тип, т. к. он наиболее совместим со сценариями применения мобильных устройств.
3. Фонограммы совещаний или семинаров, записанные с помощью мультиканальных устройств.
   Подобные аудиозаписи отличает возможность использования данных о пространственном расположении дикторов и, соответственно, их диаризации на основе уровня шума на различных микрофонах.

Процесс диаризации состоит из трёх основных последовательных шагов: обнаружение речевой активности (Speech Activity Detection, SAD), обнаружение смены дикторов (Speaker Change Detection, SCD) и кластеризация выделенных сегментов.

1.1 Извлечение характеристик

Первым этапом является извлечение признаков – данных, из которых можно получить информацию о голосах разных людей на аудиозаписи. Звуковой сигнал – это упорядоченный массив значений амплитуды звука, к которому добавляется заголовок, содержащий количество каналов, частоту дискретизации и прочую информацию. Но эти данные не содержат отличительных признаков, благодаря которым модель распознавания может определить принадлежность аудиозаписи одному и тому же человеку. К тому же, вектор свойств должен быть некоторой фиксированной длины, поэтому напрямую отсчеты сигнала отправить нельзя. Кроме этого, у таких данных варьируется амплитуда, длительность звуков и т. д., поэтому требуется дополнительная обработка аудиосигнала для извлечения необходимой информации.

Т. к. основная задача состоит в разделении речи на отдельных дикторов, в первую очередь требуется выделить эту самую речь из аудиозаписи. Для этого применяется алгоритм обнаружения голосовой активности [1]. Алгоритм VAD определяет, является ли поданный на него сегмент аудиозаписи речью, или это, например, звук домашнего питомца или дверного звонка. Очевидно, что для того, чтобы такой алгоритм был качественным, необходимо обучение с учителем.

Одним из базовых алгоритмов детектирования речи является выявление и удаление из аудиозаписи фрагментов тишины и пауз во фразах дикторов. Более сложные системы основаны на оценке уровня энергии сигнала или его спектра [16].

1.1.1 MFCC

Основные свойства речи можно извлечь из понимания голосового тракта, производящего эту самую речь. Его форма описывается огибающей спектра, а MFCC были введены [17] для того, чтобы наиболее точно её

представить. До этого для распознавания речи в основном применялись линейные коэффициенты предсказания (LPC) и линейные кепстральные коэффициенты предсказания (LPCC). Шкала мел (рис. 12) описывает отношение высоты чистого тона (мел) с фактической измеренной частотой (Гц) следующей формулой:

$$M(f) = 1127,01048 \ln(1 + f/700).$$

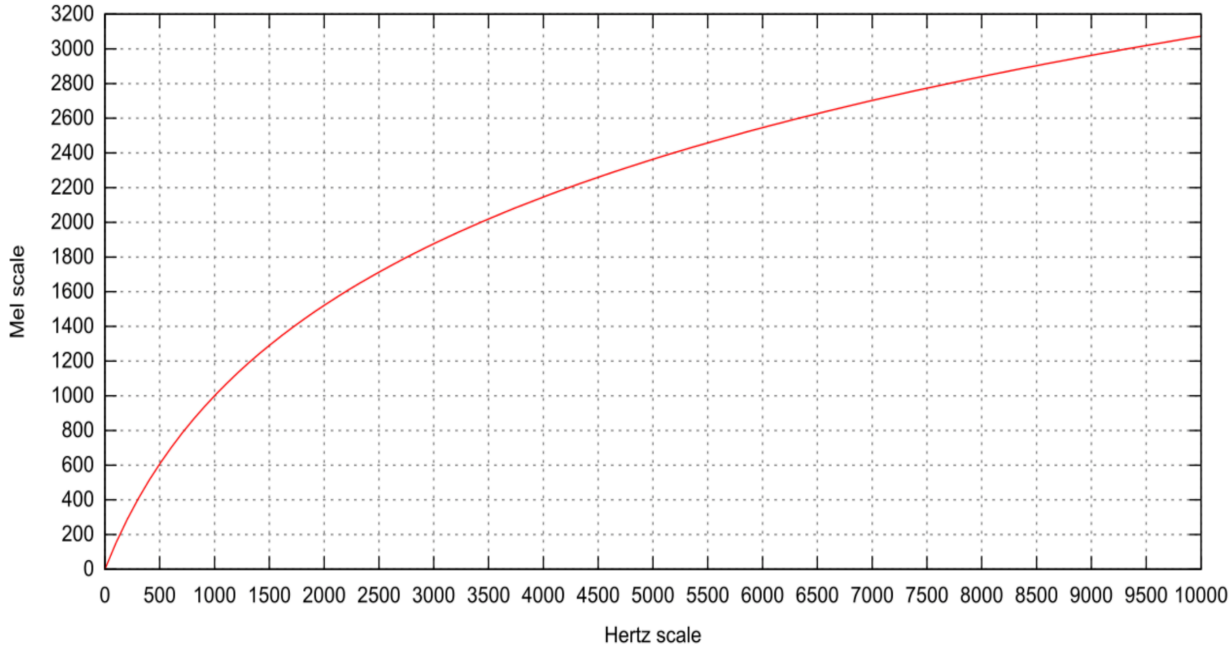


Рисунок 12. Шкала мел

Для вычисления MFCC используется следующий алгоритм. Исходный сигнал делят на фреймы фиксированной длины (от 16 до 40 мс). Затем к фрейму применяются окно Хемминга (Hamming window) и дискретное преобразование Фурье и вычисляется спектральная плотность мощности. После этого с помощью набор мел-фильтров строится мел-спектрограмма, к которой применяют дискретное косинусное преобразование (DCT).

Полученные таким образом коэффициенты представляют из себя некую сжатую характеристику фрейма, при этом, поскольку фильтры, которые применялись, расположены в мел-шкале, коэффициенты несут больше информации в диапазоне восприятия человеческого уха. Как правило, используется 12-13 MFCC на фрейм. Поскольку помимо самого

спектра индивидуальность голоса формируется скоростью и ускорениями, MFCC комбинируют с первой и второй производными [1].

1.1.2 GMM-UBM

Смесь гауссовых распределений (Gaussian Mixture Model) впервые была применена к определению принадлежности аудиозаписей дикторам в работе [18]. Плотность вероятности смеси для D-мерного вектора характеристик x можно представить формулой

$$p(x|\lambda) = \sum_{i=1}^{M} w_i p_i(x|\mu_i, \Sigma_i)$$

В ней плотность вероятности смеси $p(x|\lambda)$ представляет собой взвешенную сумму M D-мерных гауссовых плотностей вероятности $p_i(x|\mu_i, \sum_i)$ с весами w_i, которые характеризуются вектором математических ожиданий μ_i и ковариационной матрицей Σ_i

$$p_i(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu_i)'(\Sigma_i)^{-1}(x-\mu_i)\right\}$$

При этом веса компонент смеси w_i удовлетворяют ограничению $\sum_{i=1}^{M} w_i = 1$. Таким образом, все параметры модели гауссовой смеси можно представить как $\lambda = \{w_i, \mu_i, \Sigma_i\}$, где $i = 1, \ldots, M$. Кроме того, в большинстве систем используется не полная ковариационная матрица Σ_i, а диагональная. Это обосновано тем, что гауссова смесь с полной ковариационной матрицей может быть с высокой точностью представлена гауссовой смесью с диагональной ковариационной матрицей и большим количеством компонент GMM.

Безусловным преимуществом GMM является её низкая вычислительная сложность. Индивидуальные компоненты смеси могут

моделировать некоторое множество акустических классов, что позволяет описывать голосовой тракт диктора с высокой точностью.

Новым этапом в развитии данной модели стало применение универсальной фоновой модели (universal background model, UBM) для сокращения количества данных, необходимых для обучения. Для точного обучения модели одного диктора «с нуля» с помощью GMM необходимо несколько часов аудиозаписей его речи, с UBM же этого удаётся избежать [19]. GMM-UBM же обучается на всех доступных тестовых данных и затем, для построения модели каждого диктора, адаптируется с помощью размеченных для него аудиосегментов. Таким образом, для обучения модели требуется меньше данных и возрастает скорость оценки принадлежности сегмента диктору.

В качестве обучающих данных выступают векторы MFCC, выходом системы является супервектор μ, характеризующий сдвиг модели UBM для адаптации к определённому диктору.

1.1.3 i-векторы

Векторные характеристики (i- [20, 21], x- [22] и d-векторы [11]) представляют собой уникальный отпечаток голоса диктора, извлечённый с помощью GMM или глубоких нейронных сетей (DNN deep neural network) набором фильтров.

Для описания речевых особенностей диктора (составления его «акустической модели») была разработана характеристика, названная i-вектором [20]. Значение этой характеристики соответствует значению w из следующего выражения:

$$M = m + Tw,$$

где m – дикторо- и каналонезависимый супервектор, T – прямоугольная матрица низкого ранга, процесс получения которой аналогичен процессу вычисления матрицы собственных голосов V в работе

[23],

w – произвольный вектор, подчиняющийся стандартному нормальному распределению $N(0, I)$.

M считается нормально распределенным со средним вектором m и ковариационной матрицей TT^t.

Вычисление косинусного расстояния для двух i-векторов, вычисленных для различных сегментов речи позволяет принимать решение об их принадлежности одному диктору (чем значение ближе к 1, тем эта вероятность больше, если же значения ближе к -1, то это говорит в пользу принятия обратной гипотезы).

1.1.4 d-векторы

Данная характеристика извлекается с помощью обучения LSTM на фреймах, полученных из исходной аудиозаписи [11]. К полученным значениям последнего слоя нейросети применяется L2 нормализация. Центроида полученного кортежа $(e_{k1}, \cdots, e_{kM})$, где e – встраивание соответствующего фрейма, представляет собой уникальный голосовой отпечаток для M встраиваний и описывается следующим выражением:

$$\mathbf{c}_k = \mathbb{E}_m[\mathbf{e}_{km}] = \frac{1}{M}\sum_{m=1}^{M}\mathbf{e}_{km}$$

Вычисленное косинусное расстояние s между такими центроидами в дальнейшем используется в функции потерь для кластеризации отпечатков между различными дикторами.

$$s = w \cdot \cos(\mathbf{e}_{j\sim}, \mathbf{c}_k) + b$$

На данный момент работы, использующие d-векторы для описания дикторов, показывают наилучшие результаты в решении данной задачи [11, 13]. При этом, даже при небольшом количестве доступных примеров речи

конкретного диктора, эту характеристику можно успешно применять для его описания [24].

1.2 Обнаружение смены дикторов

На следующем этапе необходимо разделить выделенную речь на сегменты, принадлежащие разным дикторам. Обнаружение смены дикторов является важной частью системы диаризации. Она направлена на поиск границ между речевыми оборотами двух человек. Для этого на каждом сегменте аудиозаписи определяются так называемые точки смены дикторов (speaker change points).

Современные методы, основанные на применении нейронных сетей, предполагают следующий алгоритм нахождения таких точек [25]:

- аудиозапись разбивается на фреймы с наложением минимальной длины
- каждому фрейму в соответствие ставится обученное встраивание диктора
- для соседних встраиваний (либо их кортежей) вычисляется косинусное расстояние и сравнивается с некоторым заданным пороговым значением
- принимается решение о том, принадлежат ли встраивания одному диктору

1.3 Кластеризация

Последним этапом диаризации является кластеризация распознанных участков речи по отдельным дикторам. Для этого можно использовать следующие алгоритмы [2, 26]:

- Hidden Markov Model (HHM)
- Hierarchical Agglomerative Clustering (HAC)

- Probabilistic Linear Discriminant Analysis
- Spectral clustering

Либо специализированные решения на основе нейронных сетей:

- Discriminative Neural Clustering (DNC)
- Unbounded Interleaved-State Recurrent Neural Network (UIS-RNN)

Однако для задачи распознавания в режиме реального времени добавляются значительные ограничения на проведение кластеризации: анализируется не конечное множество элементов, а каждый новый относится либо к уже определённым кластерам, либо к новому, обозначающему диктора, не представленного на аудиозаписи ранее. Использование таких алгоритмов, как указано в [11], негативно сказывается на результатах классификации.

1.3.1 Спектральная автономная кластеризация

Этот непараметрический алгоритм, представленный в работе [11], состоит из следующих этапов:

- построение матрицы схожести (affinity). Элементы этой матрицы представляют косинусное сходство между встраиванием отрезков.
- применение некоторых операций уточнения к матрице схожести (на рис. 13 приведена наглядная демонстрация):
    - гауссово размытие для сглаживания данных (для уменьшения эффекта выбросов)
    - построковые пороговые значения (элемент приравнивается нулю если его значение меньше заданного для этой строки порога)
    - симметризация для восстановления симметрии матрицы, важной для данного алгоритма

- диффузия для повышения чёткости границ между строками
- максимальная построковая нормализация для избавления от нежелательных эффектов масштаба

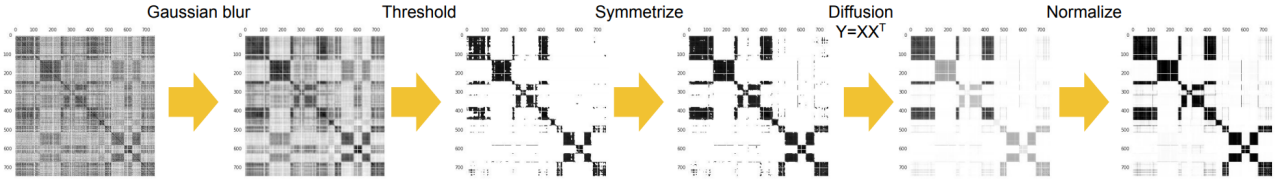


Рисунок 13. Применение некоторых операций уточнения к матрице схожести

- применение собственного разложения

Матрица схожести описывается полный граф с вершинами, соответствующим встраиваниям, и рёбрами между ними с весом, соответствующим степени их похожести [26]. В качестве встраиваний можно использовать как супервекторы из GMM-UBM, так и векторные характеристики (i-, x-, d-векторы).

1.3.2 UIS-RNN

Данный алгоритм кластеризации основывается на следующих трёх фактах:

- Каждого выступающего можно моделировать как экземпляр рекуррентной нейронной сети.
- Нет ограничений на указание номера выступающего, т.е. система может учиться и угадывать количество дикторов на аудиозаписи.
- Состояния различных экземпляров RNN соответствует отдельным дикторам, чередующимся во временном пространстве.

Таким образом, процесс разбивается на следующие этапы:

- генерирование последовательностей (sequence generation)
- определение диктора (speaker assignment)
- смена дикторов (speaker change)

Основное предположение заключается в том, что последовательность наблюдения встраиваний дикторов X генерируется распределениями, которые параметризуются выходом RNN. Эта RNN имеет несколько экземпляров, соответствующих разным дикторам, и они имеют один и тот же набор параметров. В качестве архитектуры RNN для долговременной памяти используются управляемые рекуррентные блоки (Gated Recurrent Units, GRU). [13]

Одной из главных проблем задачи диаризации является определение общего числа дикторов. Для этой задачи было предложено использовать алгоритм ddCRP, который представляет собой байесовскую непараметрическую модель. В момент смены дикторов существует 2 возможных варианта. Следующий диктор может быть либо уже известным системе, либо абсолютно новым. Вероятность возврата к ранее выступившему оратору пропорциональна количеству произнесенных им непрерывных речей. Существует также возможность переключиться на нового диктора, вероятность которого пропорциональна постоянной a.

$$p(\mathbf{Y}|\mathbf{Z},\alpha) = \frac{\alpha^{K_T-1}\prod_{k=1}^{K_T}\Gamma(N_{k,T})}{\prod_{t=2}^{T}(\sum_{k\in[K_{t-1}]\setminus\{y_{t-1}\}}N_{k,t-1}+\alpha)^{\mathbb{1}(z_t=1)}}.$$

В предыдущей формуле z_t представляет собой смену дикторов. Очевидно, значение z_t находится между 0 и 1. Это можно параметризовать с помощью любой функции, однако авторы для простоты используют постоянное значение. Так что значение становится двоичной переменной.

$$p(z_t = 1|z_{[t-1]},\boldsymbol{\lambda}) = g_{\boldsymbol{\lambda}}(z_{[t-1]})$$

UIS-RNN является генеративным процессом:

- Для встраивания последовательности используется X, представляющий d-вектор сегмента аудиопотока.
- Для обозначения истинных значений используется Y. Например, Y = (1, 2, 2, 3, 3). Эти числа представляют собой идентификатор диктора на соответствующем сегменте аудиопотока.

$$p(\mathbf{X}, \mathbf{Y}) = p(\mathbf{x}_1, y_1) \cdot \prod_{t=2}^{T} p(\mathbf{x}_t, y_t | \mathbf{x}_{[t-1]}, y_{[t-1]}).$$

Но в предыдущей модели не содержится информации об изменении диктора, для этого вводится величина Z. Например, для Y = (1, 1, 2, 3, 2, 2) получим Z = (0, 1, 1, 1, 1, 0):

$$p(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) = p(\mathbf{x}_1, y_1) \cdot \prod_{t=2}^{T} p(\mathbf{x}_t, y_t, z_t | \mathbf{x}_{[t-1]}, y_{[t-1]}, z_{[t-1]}),$$

$$p(\mathbf{x}_t, y_t, z_t | \mathbf{x}_{[t-1]}, y_{[t-1]}) = \underbrace{p(\mathbf{x}_t | \mathbf{x}_{[t-1]}, y_{[t]})}_{\text{sequence generation}} \cdot \underbrace{p(y_t | z_t, y_{[t-1]})}_{\text{speaker assignment}} \cdot \underbrace{p(z_t | z_{[t-1]})}_{\text{speaker change}} \cdot$$

1.4 Анализ результатов диаризации

Для оценки эффективности работы систем разделения дикторов на фонограмме существует несколько методик. Одна из существующих методик разработана национальным институтом стандартов и технологий США (National Institute of Standards and Technology, NIST). В соответствии с этой методикой, мерой оценки эффективности систем разделения дикторов на аудиозаписи выступает величина Diarization Error Rate (DER) [27], которая рассчитывается по формуле:

$$DER = \frac{\sum_{s=1}^{S} \mathrm{dur}(s) \cdot (\max(N_{ref}(s), N_{hyp}(s)) - N_{correct}(s))}{\sum_{s=1}^{S} \mathrm{dur}(s) \cdot N_{ref}}$$

где dur(s) – длительность речевого сегмента *s*; $N_{ref}(s)$ – количество дикторов, голос которых присутствует на речевом сегменте *s* в соответствии с эталонной разметкой; $N_{hyp}(s)$ – количество дикторов, голос которых присутствует на речевом сегменте *s* в соответствии с результатом работы тестируемой системы; $N_{correct}(s)$ – количество верно отнесенных к речевому сегменту *s* дикторов.

Величина DER является суммой трёх ошибок: ошибки ложного детектирования речи E_{FA},

$$E_{FA} = \frac{\sum_{s=1}^{S} \mathrm{dur}(s) \cdot (N_{hyp}(s) - N_{ref}(s))}{T_{score}} \quad \forall\ (N_{hyp}(s) - N_{ref}(s)) > 0$$

ошибки ложного пропуска речи E_{miss}

$$E_{MISS} = \frac{\sum_{s=1}^{S} \mathrm{dur}(s) \cdot (N_{ref}(s) - N_{hyp}(s))}{T_{score}} \quad \forall\ (N_{ref}(s) - N_{hyp}(s)) > 0$$

и ошибки разделения дикторов E_{spkr}

$$E_{Spkr} = \frac{\sum_{s=1}^{S} \mathrm{dur}(s) \cdot (\min(N_{ref}(s), N_{hyp}(s)) - N_{correct}(s))}{T_{score}}$$

Глава 2. Обзор существующих решений

2.1 Google Cloud Speech API

Основной областью применения данного решения является преобразование речи из аудио в текст, но в нём также имеются инструменты для проведения диаризации. Распознавание происходит удаленно на серверах компании Google, поэтому для работы приложения потребуется подключение к сети Интернет.

Кроме того, данное решение использует закрытые разработки компании, что накладывает дополнительные ограничения на возможности его использования. При этом, команда разработчиков регулярно делится своими основными достижениями на различных конференциях, а также реализацией отдельных методов и алгоритмов.

Доступ к API предоставляется через web интерфейс посредством HTTP запросов, либо через библиотеки для языков Java, Python, JavaScript (Node.JS). Пример HTTP запроса:

```
curl -s -H "Content-Type: application/json" \
    -H "Authorization: Bearer $(gcloud auth application-default print-access-
token)" \
    https://speech.googleapis.com/v1p1beta1/speech:recognize \
    --data '{
    "config": {
        "encoding": "LINEAR16",
        "languageCode": "en-US",
        "enableSpeakerDiarization": true,
        "diarizationSpeakerCount": 2,
        "model": "phone_call"
    },
    "audio": {
        "uri": "gs://cloud-samples-tests/speech/commercial_mono.wav"
    }
}' > speaker-diarization.txt
```

Ответ на запрос, записанный в speaker-diarization.txt

```
{
  "results": [
    {
      "alternatives": [
        {
          "transcript": "hi I'd like to buy a Chromecast and I was wondering
whether you could help me with that certainly which color would you like we
have blue black and red uh let's go with the black one would you like the new
Chromecast Ultra model or the regular Chrome Cast regular Chromecast is fine
```

```
thank you okay sure we like to ship it regular or Express Express please
terrific it's on the way thank you thank you very much bye",
          "confidence": 0.92142606,
          "words": [
            {
              "startTime": "0s",
              "endTime": "1.100s",
              "word": "hi",
              "speakerTag": 2
            },
            {
              "startTime": "1.100s",
              "endTime": "2s",
              "word": "I'd",
              "speakerTag": 2
            },
            {
              "startTime": "2s",
              "endTime": "2s",
              "word": "like",
              "speakerTag": 2
            },
            {
              "startTime": "2s",
              "endTime": "2.100s",
              "word": "to",
              "speakerTag": 2
            },
            ...
          ]
        }
      ],
      "languageCode": "en-us"
    }
  ]
}
```

2.2 IBM Watson Speech to Text

Компания IBM предлагает своё end-to-end решение для распознавания речи в текст, в том числе с использованием диаризации дикторов на аудиозаписи. Доступ к сервису, использующему мощности суперкомпьютера IBM Watson, осуществляется с помощью web-интерфейса. Пример HTTP запроса:

```
curl -X POST -u "apikey:{apikey}"
--header "Content-Type: audio/flac"
--data-binary @{path}audio-multi.flac
"{url}/v1/recognize?model=en-US_NarrowbandModel&speaker_labels=true"
```

Полученный ответ на запрос:

```
{
  "results": [
    {
      "alternatives": [
```

```
        {
          "timestamps": [
            [
              "hello",
              0.68,
              1.19
            ],
            . . .
          ]
          "confidence": 0.82,
          "transcript": "hello yeah yeah how's Billy "
        }
      ],
      "final": true
    }
  ],
  "result_index": 0,
  "speaker_labels": [
    {
      "from": 0.68,
      "to": 1.19,
      "speaker": 2,
      "confidence": 0.42,
      "final": false
    },
    {
      "from": 1.47,
      "to": 1.93,
      "speaker": 1,
      "confidence": 0.52,
      "final": false
    },
    {
      "from": 1.96,
      "to": 2.12,
      "speaker": 2,
      "confidence": 0.41,
      "final": false
    },
    {
      "from": 2.12,
      "to": 2.59,
      "speaker": 2,
      "confidence": 0.41,
      "final": false
    },
    {
      "from": 2.59,
      "to": 3.17,
      "speaker": 2,
      "confidence": 0.41,
      "final": false
    },
    . . .
  ]
}
```


Нетрудно заметить, что данное решение во многом похоже на аналогичное API от компании Google и проявляет те же недостатки:

невозможность автономной работы алгоритма и закрытый исходный код. Несмотря на высокую репутацию обеих компаний, к подобным реализациям всегда имеются претензии относительно защищённости пользовательских данных [28].

2.3 DeepSpeech

Эта библиотека поддерживается компанией Mozilla и основывается на разработках компании Baidu. В первую очередь она предназначена для решения задачи распознавания речи в текст (speech-to-text), однако может быть применена и к рассматриваемой задаче после некоторых модификаций. Код данной библиотеки находится в открытом доступе на платформе github: https://github.com/mozilla/DeepSpeech. Основными недостатками технологии являются высокие требования к производительности устройств, как для обучения модели на собственных данных, так и непосредственного применения её для анализа.

2.4 Vosk API

Одна из наиболее производительных библиотек для анализа речи, анонсированная в конце 2019 года и опубликованная в свободном доступе (https://github.com/alphacep/vosk-api). На данный момент практически отсутствует документация, что значительно усложняет работу с данным продуктом. Является разработкой группы учёных Новосибирского Государственного Университета (https://alphacephei.com/). Для платформы Android подготовлен APK-пакет, а для Linux можно использовать Python-библиотеку, производительности которой достаточно для работы на платах Raspberry Pi. Библиотека работает на усовершенствованном движке Kaldi, ранее считавшимся одним из наилучших решений для речевого анализа. Языковая модель занимает всего 50Мб и работает точнее DeepSpeech (с моделями размером более 1Гб). Поддерживаются языки: русский,

английский, немецкий, французский, китайский. Ожидается поддержка испанского, хинди, арабского и португальского. Данная реализация наиболее соответствует поставленным условиям, однако её релиз состоялся уже в момент написания этой работы.

Глава 3. Программная реализация

В качестве основного языка разработки был выбран Python 3 за его простоту в использовании и широкий выбор доступных библиотек для обработки аудиофайлов и аудиопотоков, реализации решений с использованием нейронных сетей, многопоточным вычислениям и обработке полученных статистических данных.

3.1 Техническое обеспечение

Т.к. одним из важных критериев является возможность запуска программной реализации на мобильных устройствах, основной платформой для тестирования была выбрана однопроцессорная плата Raspberry Pi 3 Model B (рис. 14). Благодаря 64-битному процессору на архитектуре ARMv53 эта платформа поддерживает множество операционных систем различной направленности, в том числе IoT (Internet of Things, или «интернет вещей»). Это предоставляет широкие возможности по моделированию элементов «умного дома» на основе данной платформы [30, 31, 32].

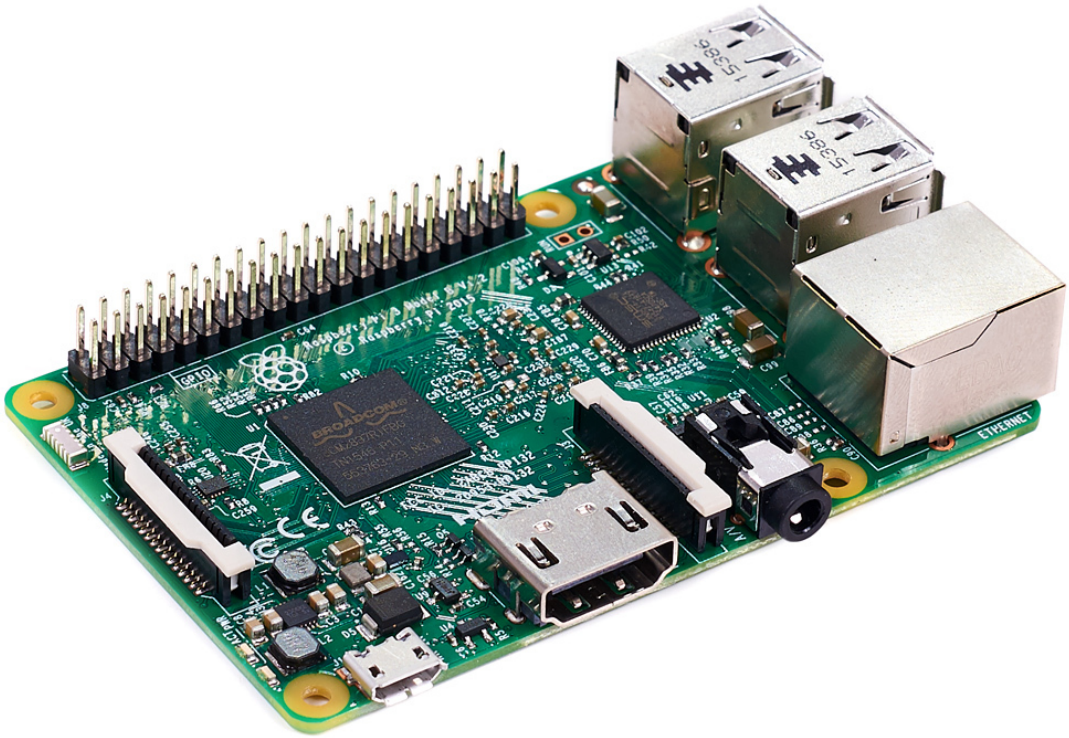


Рисунок 14. Внешний вид Raspberry Pi 3 Model B

Основным минусом такой системы будет являться отсутствие дискретной графической подсистемы, которая бы обеспечивала высокую производительность при вычислениях с применением нейронных сетей. Тем не менее, установленный процессор обеспечивает необходимую вычислительную мощность для обработки аудипотока и его анализа с помощью выбранного программного обеспечения.

Помимо наличия стандартных разъёмов 3.5 мм mini-jack и HDMI, устройство обладает дополнительными интерфейсами, позволяющими расширить основной функционал устройства подключением штатных модулей через 15-пиновые слоты:

- CSI-2 — для подключения камеры по интерфейсу MIPI;
- DSI — для подключения штатного дисплея.

В качестве низкоуровневых интерфейсов доступны:

- 40 портов ввода-вывода общего назначения;
- UART (Serial);

- I²C/TWI;
- SPI с селектором между двумя устройствами;
- пины питания: 3,3 В, 5 В.

На платформу была установлена операционная система Raspbian с ядром linux версии 4.19. Для запуска исполняемых файлов использовалось виртуальное окружение языка Python версии 3.7 с установленными библиотеками, описанными в следующем разделе и всеми необходимыми для них зависимостями.

3.2 Используемые программные инструменты

Разработку и тестирование моделей было решено производить в Jupyter Notebook. Это крайне удобный инструмент для создания красивых аналитических отчетов, так как он позволяет хранить вместе код, изображения, комментарии, формулы и графики (рис. 15).

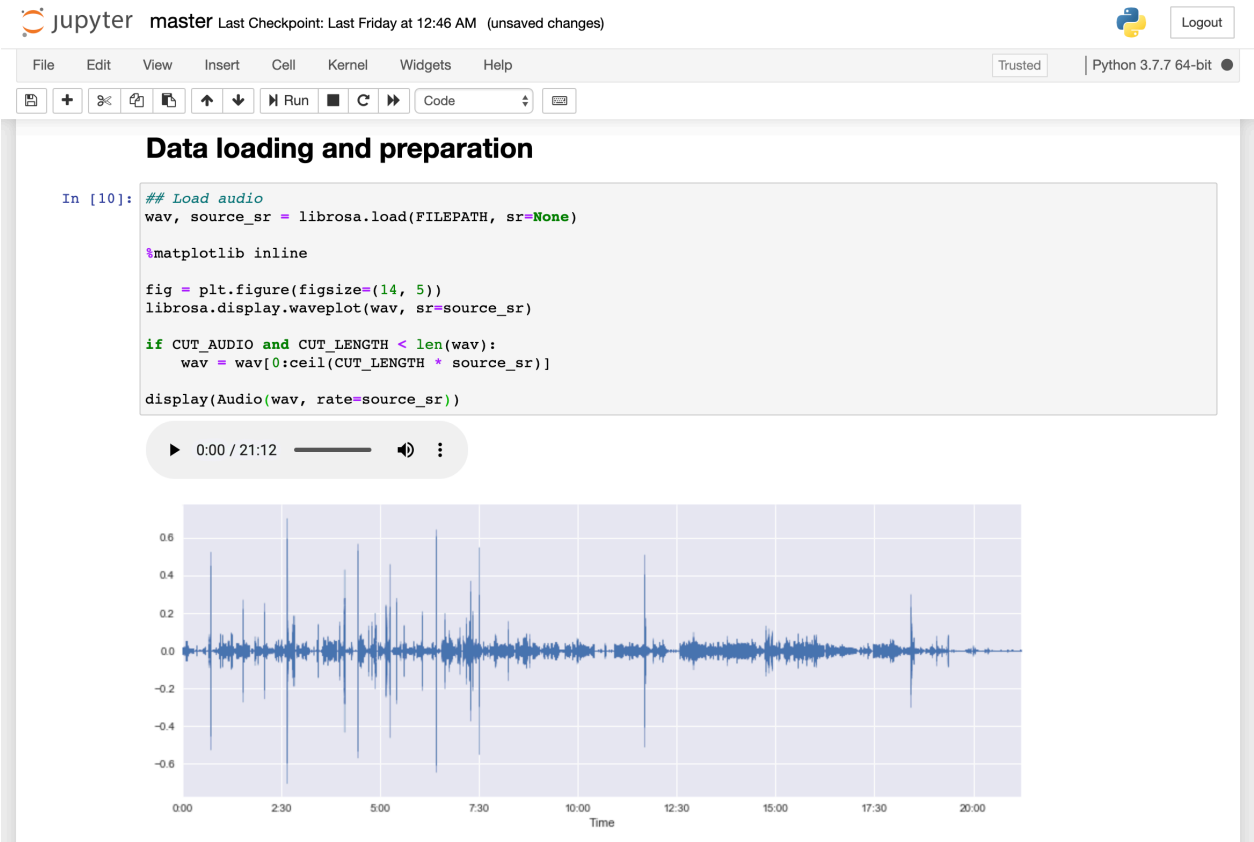


Рисунок 15. Пример Jupyter Notebook с выводом графика и аудиозаписи

История ввода и вывода сохраняется на протяжении всего времени исполнения файла, что позволяет вернуться к определённым этапам и продолжить исследование.

Модуль PyAudio позволяет работать с кросплатформенной библиотекой PortAudio, проигрывать аудиофайлы и записывать аудиопоток (из файла или со внешних устройств вычислительного устройства) для дальнейшей обработки.

LibROSA – это пакет Python для анализа музыки и аудио, предоставляющий набор модулей, необходимых для создания музыкальных информационно-поисковых систем. В их число входят модули для чтения и записи аудиофайлов, преобразования аудиоданных и вывода графических данных об их различных характеристиках.

В качестве инструмента для детектирования участков речи был выбран пакет WebRTCVAD Wrapper, представляющий собой расширение функционала библиотеки py-webrtcvad, которая в свою очередь является интерфейсом для реализации алгоритма VAD в проекте WebRTC. Кроме того, выбранный пакет предоставляет альтернативный алгоритм, предназначенный специально для подготовки данных для обработки нейронными сетями, однако в данной реализации он показал результаты хуже, по сравнению с основным. Для высоко- и низкоуровневой обработки аудиопотока в этой библиотеке используются возможности пакетов librosa и pydub.

Возможности high pass фильтра для минимизации шумов и алгоритм вычисления косинусного расстояния между векторами реализованы в пакете для научных расчётов SciPy.

Обучение встраиваний дикторов реализовано средствами библиотеки Resemblyzer, основанной на современной библиотеке глубокого обучения PyTorch. Низкое время обработки аудиосегментов нейронными сетями на CPU позволяет использовать её и в режиме реального времени. Исходный

код Resemblyzer находится в открытом доступе на платформе github, поэтому не составило значительных трудностей по её модификации для улучшения некоторых параметров и адаптации для обработки данных в режиме реального времени.

Для анализа статистических данных используется библиотека scikit-learn, отличающаяся эффективными методами работы с большими объёмами данных. Все алгоритмы кластеризации, представленные в ней, обладают множеством параметров, что позволяет точно настроить их под конкретную задачу.

От использования реализации uis-rnn компанией Google было решено отказаться в силу того, что версия алгоритма, находящаяся в открытом доступе, не предоставляет весь заложенный потенциал, т. к. основана на некоторых внутренних зависимостях компании, недоступных извне. Кроме того, для работы алгоритма требуется наличие предварительно обученной модели, размер которой может достигать нескольких сот мегабайт, что критично для рассматриваемых малопроизводительных устройств.

Применение различных метрик при анализе результатов производится с помощью библиотеки pyannote.metrics [33], поддерживающей работу в том числе с аудиопотоками, речь дикторов на которых накладывается. Возможности продукта включают вычисление значений таких метрик, как Diarization Error Rate (а также её модификации Greedy DER) и Jacard Error Rate, прикладные к задаче диаризации аудиопотока.

В качестве инструмента для реализации графических элементов была использована библиотека Matplotlib, де-факто являющаяся стандартом для подобных решений в среде Jupyter Notebook. Пример вывода временных сегментов, соответствующих различным дикторам на аудиозаписи изображён на рис. 16.

```
%matplotlib inline

fig = plt.figure(figsize=(13, 3))
axes = plt.axes()
plt.title('Ground truth')
librosa.display.waveplot(wav * 2, sr=source_sr, color='lightgray')

for item in ami_corpus_transcripts:
    id = item['speaker_id']
    plt.plot(
        [item['start'], item['end']],
        ['Speaker ' + str(id), 'Speaker ' + str(id)],
        color=color[id % 15],
        linewidth=4
    )

axes.xaxis.set_major_formatter(formatter)
ticks_freq = ceil(len(wav) * .1 / source_sr)
axes.xaxis.set_major_locator(plt.MultipleLocator(ticks_freq))

plt.margins(x=0.01)
plt.tight_layout()
```

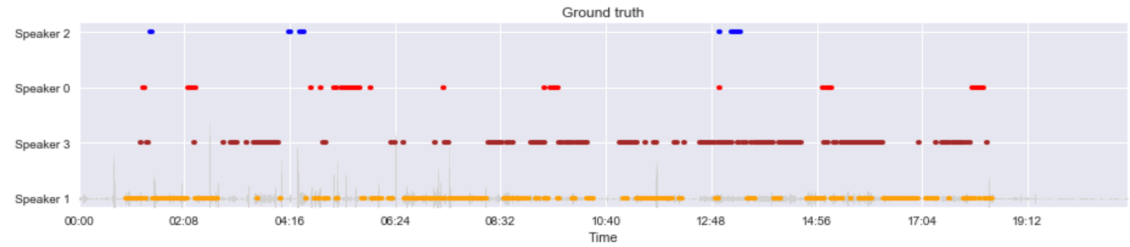


Рисунок 16. Пример вывода графика с помощью Matplotlib

3.3 Описание практической реализации алгоритма

Исходный код предлагаемого решения находится в открытом доступе по адресу https://github.com/nikitalpopov/master. Аудиопоток записывается с внешнего микрофона либо из локального аудиофайла в отдельные блоки (также их называют фреймами (frames) или чанками (chunks)). Каждый блок имеет одинаковую длину, предварительно объявленную, и имеет пересечения с соседними. Алгоритм разбиения локального аудиофайла (считанного в переменную *wav*) на фреймы с наложением:

```
import librosa
from math import ceil


wav, source_sr = librosa.load(FILEPATH, sr=SR)
chunk_size = ceil(CHUNK_TIME_LENGTH * SR)

for i in range(0, len(wav), ceil(chunk_size * (1 - CHUNKS_OVERLAP))):
    start = i
    end = i + chunk_size if i + chunk_size < len(wav) - 1 else len(wav) - 1

    chunk = wav[start:end]
```

Алгоритм записи потока с микрофона и его разделения на фреймы:

```
import argparse
import sys
import sounddevice as sd
```

```
def int_or_str(text):
    try:
        return int(text)
    except ValueError:
        return text


parser = argparse.ArgumentParser(add_help=False)
parser.add_argument(
    '-l', '--list-devices', action='store_true',
    help='show list of audio devices and exit')
args, remaining = parser.parse_known_args()
if args.list_devices:
    print(sd.query_devices())
    parser.exit(0)
parser = argparse.ArgumentParser(
    description=__doc__,
    formatter_class=argparse.RawDescriptionHelpFormatter,
    parents=[parser])
parser.add_argument(
    'filename', nargs='?', metavar='FILENAME',
    help='audio file to store recording to')
parser.add_argument(
    '-d', '--device', type=int_or_str,
    help='input device (numeric ID or substring)')
parser.add_argument(
    '-r', '--samplerate', type=int, default=16000, help='sampling rate')
parser.add_argument(
    '-c', '--channels', type=int, default=1, help='number of input channels')
parser.add_argument(
    '-t', '--subtype', type=str, help='sound file subtype (e.g. "PCM_24")')
args = parser.parse_args(remaining)

do_ record = True
audio = []


def callback(indata, frames, time, status):
    global audio
    audio.extend(indata.copy())

    if status:
        print(status, file=sys.stderr)


try:
    with sd.InputStream(
        samplerate=args.samplerate,
        device=args.device,
        channels=args.channels,
        callback=callback
    ):
        print('Press Ctrl+C to stop the recording')

        while do_record:
            process_chunk()

except KeyboardInterrupt:
    print('Recording has finished')
    do_record = False
```

```
    pass

except Exception as e:
    parser.exit(type(e).__name__ + ': ' + str(e))
```

К фреймам применяется технология удаления частотного шума. Затем к фреймам применяется алгоритм детектирования участков активной речи (VAD) и удаляются лишние сегменты с сохранением информации об исходном положении участков в оригинальной аудиозаписи (которую будем называть маской).

```
from webrtcvad_wrapper import VAD
from pydub import AudioSegment
import soundfile as sf
import pydub.scipy_effects


vad = VAD(sensitivity_mode=SENSITIVITY_MODE)
vad_segments = []


def get_vad_segments(audio_segment):
    filtered_segments = vad.filter(audio_segment)

    return [
        (filtered_segment[0], filtered_segment[1])
        for filtered_segment in filtered_segments if filtered_segment[-1]
    ]


chunk = normalize_volume(chunk)
chunk = trim_long_silences(chunk)

if (len(chunk) > 0):
    filename = f'chunks/chunk_{i}.wav'
    sf.write(filename, chunk, samplerate=SR)
    audio = AudioSegment.from_file(filename)

    if audio.sample_width != 2:
        audio = audio.set_sample_width(2)
    if audio.channels != 1:
        audio = audio.set_channels(1)

    audio = audio.high_pass_filter(FILTER_VALUE, order=3)

    segments = get_vad_segments(audio)
    vad_segments += [(start + ceil(segment[0] * SR), start + ceil(segment[1]
* SR)) for segment in segments]

    segments = [s for s in remove_overlap(vad_segments) if s[1] - s[0] > .2 *
SR]
    segments = [(s[0] - start, s[1] - start) for s in segments if s[0] >=
start]
```

Сравнение исходного (синий) и обработанного шумоподавлением (зелёный) аудиопотоков на примере аудиозаписи диалога длиной более 2 минут представлено на рис. 17

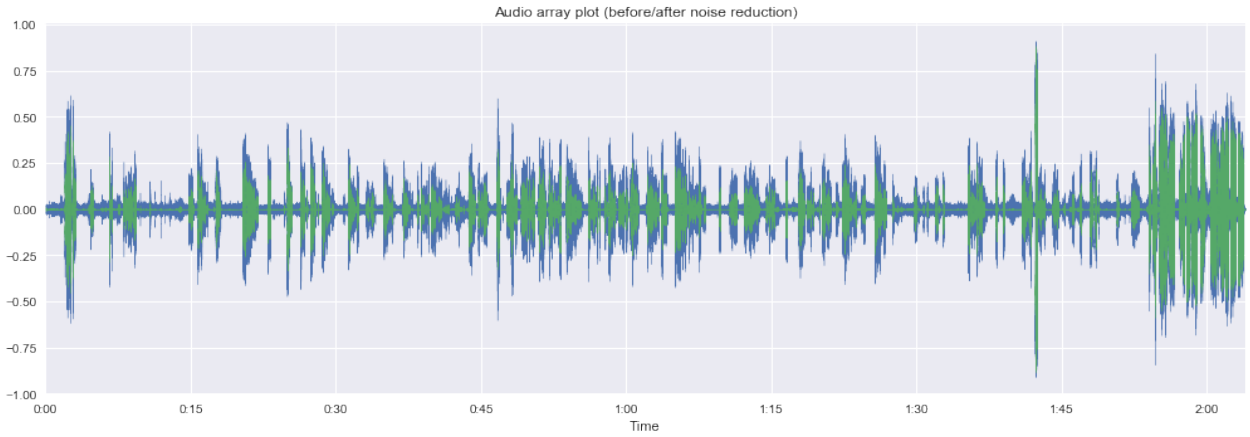


Рисунок 17. Сравнение исходного и обработанного алгоритмом шумоподавления аудиопотоков

На следующем этапе производится обучение встраиваний дикторов с помощью d-векторов для каждого распознанного сегмента с активной речью. Для полученных значений векторов вычисляется косинусное расстояние и производится их кластеризация с динамическим определением числа кластеров:

```
from resemblyzer import VoiceEncoder
import numpy as np
from scipy.spatial.distance import cosine


## load encoder
encoder = VoiceEncoder()

for segment in segments:
    seg_start, seg_end = segment
    frame = chunk[seg_start:seg_end]
    embed = encoder.embed_utterance(frame)

    if len(embeds) > 0:
        distances = []
        for speaker_id, speaker_embeddings in groupby(sorted(embeds,
key=lambda x: x[1]), lambda x: x[1]):
            es = [e for e, s_id in list(speaker_embeddings)]

            mean_dist = np.mean([get_similarity(embed, embedding) for
embedding in es])
            min_dist = np.min([get_similarity(embed, embedding) for embedding
in es])
            max_dist = np.max([get_similarity(embed, embedding) for embedding
in es])

            distances.append((min_dist, speaker_id))
```

```
        distances = sorted(distances, key=lambda x: x[0])

        best_id = distances[0][1]

        if distances[0][0] < SPEAKER_CHANGE_THRESHOLD:
            id = best_id
        else:
            counter += 1
            id = counter

        speakers_ids.append(id)

        speaker_segments.append({
            'start': start + seg_start,
            'end': start + seg_end,
            'speaker_id': id
        })

    embeds.append((embed, speakers_ids[best]))
```

Сравнение результатов диаризации с заранее известным транскриптом для аудиозаписи, обработанной в режиме реального времени:

```
from pyannote.core import Segment, Timeline, Annotation, notebook
from pyannote.metrics.diarization import DiarizationErrorRate,
JaccardErrorRate
import librosa


der = DiarizationErrorRate()
jer = JaccardErrorRate()

x = librosa.times_like(wav.shape[0], sr=source_sr, hop_length=1)


def measure_metrics(reference, hypothesis):
    print('DER:', der(reference, hypothesis))
    print('JER:', jer(reference, hypothesis))


%matplotlib inline

fig = plt.figure(figsize=(13, 5))

# plot hypothesis
ax = plt.subplot(211)
plt.title('hypothesis')

hypothesis = Annotation()
for t in speaker_segments:
    try:
        hypothesis[Segment(x[t['start']], x[t['end']])] =
str(t['speaker_id']) + '_hyp'
    except:
        pass

notebook.plot_annotation(hypothesis, legend=True, time=True)
librosa.display.waveplot(wav, sr=source_sr, color='lightgray')
```

```
# plot reference
ax = plt.subplot(212)
plt.title('reference')

reference = Annotation()
for t in ami_corpus_transcripts:
    try:
        reference[Segment(t['start'], t['end'])] = str(t['speaker_id']) +
'_ref'
    except:
        pass

notebook.plot_annotation(reference, legend=True, time=True)
librosa.display.waveplot(wav, sr=source_sr, color='lightgray')

plt.margins(x=0.01)
plt.tight_layout()
plt.show()

measure_metrics(reference, hypothesis)
```

На каждом графике одинаковыми цветами отображаются сегменты, относящиеся к одному диктору. Первый график на рис. 18 отражает полученные результаты в ходе работы алгоритма, второй – известные заранее, «реальные», значения.

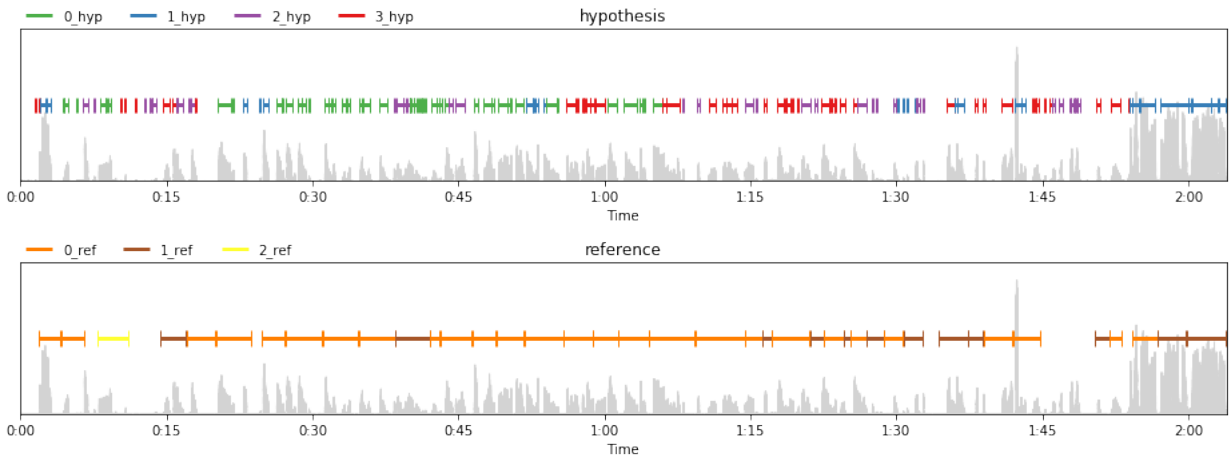


Рисунок 18. Сравнение полученных результатов и данных из датасета

Выводы

В качестве тестового датасета для подсчёта статистических метрик был выбран AMI Meeting Corpus [34]. Он состоит из 100 часов записи заседаний и доступен для использования исследователями на безвозмездной основе. В записях используется ряд сигналов, синхронизированных с общей временной шкалой. К ним относятся микрофоны ближнего и дальнего действия, индивидуальные и комнатные видеокамеры, а также выход со слайд-проектора и электронной доски. Во время заседаний участники также имеют в своем распоряжении несинхронизированные ручки, которые записывают написанное. Встречи записывались на английском языке в трёх разных залах с разными акустическими свойствами, и в них в основном принимали участие лица, не являющиеся носителями языка.

При анализе данных можно заметить встречающиеся на аудиозаписях сильные шумы, которые влияют на алгоритмы детектирования речи (например, резкие «всплески» на первой половине аудиозаписи, график которой представлен на рис. 19).

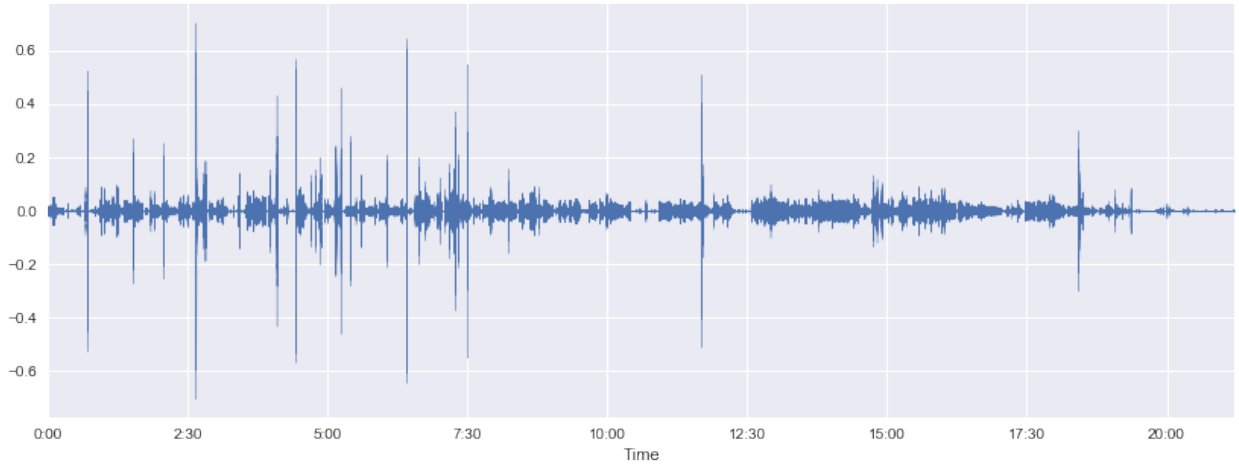


Рисунок 19. График колебаний для аудиозаписи AMI.ES2002.a

Аудиозаписи сохранены в wav формате в моноканале. Для получения данных об отрезках аудиофрагментов, принадлежащих различным дикторам, датасет содержит различные xml файлы с информацией о фразах,

их таймкодах и транскриптах. Для их преобразования в более удобный для данной задачи формат, был реализован скрипт, доступный в приложении 1. Пример полученного транскрипта:

```
[..., {
    "start": 14.333,
    "end": 16.983,
    "text": "This place? A coffee shop?",
    "speaker_id": 1
}, {
    "start": 17.073,
    "end": 20.023,
    "text": "What's wrong with that?",
    "speaker_id": 0
}, ...]
```

Поля *start* и *end* обозначают время начала и конца сегмента в аудиозаписи (в секундах), *text* – транскрипт произнесённой фразы, *speaker_id* – уникальный идентификатор диктора, которому принадлежит фраза. Наличие транскриптов позволяет тестировать более сложные системы, включающие в себя алгоритмы преобразования речи в текст (speech-to-text, STT).

В ходе тестирования программной реализации было отмечено, что алгоритм допускает некоторые ошибки по идентификации дикторов на отдельных сегментах аудиопотока. Причём количество этих ошибок возрастает с количеством уникальных дикторов на аудиозаписи. Наибольшие проблемы вызвали записи совещаний из 4 человек с частым наложением их речи.

Наглядное сравнение полученных результатов для аудиозаписи с 4 дикторами на примере заседания ES2002.a представлено на рис. 20. На нём представлены два графика, отражающие результаты диаризации алгоритмом и реальные данные о репликах, извлечённые из транскриптов. Уникальные идентификаторы отрезков обозначают отдельных дикторов (таким образом, мы видим, что вместо 4 алгоритм распознал 10 уникальных голосов). Можно заметить, что алгоритм распознал все реплики основного диктора (*1_hyp*, *1_ref*), однако речь остальных дикторов во многих местах

содержит ошибки идентификации. Кроме того, сегменты, на которых несколько дикторов говорили одновременно, распознаны со значительными отклонениями от реальных данных (*3_hyp* соответствует репликам *3_ref*), а некоторые посторонние шумы были распознаны как речь и отнесены либо к реальным дикторам (*1_hyp* одновременно включает и шумы, и реплики *1_ref*), либо к новым (*4_hyp, 5_hyp, ...*).

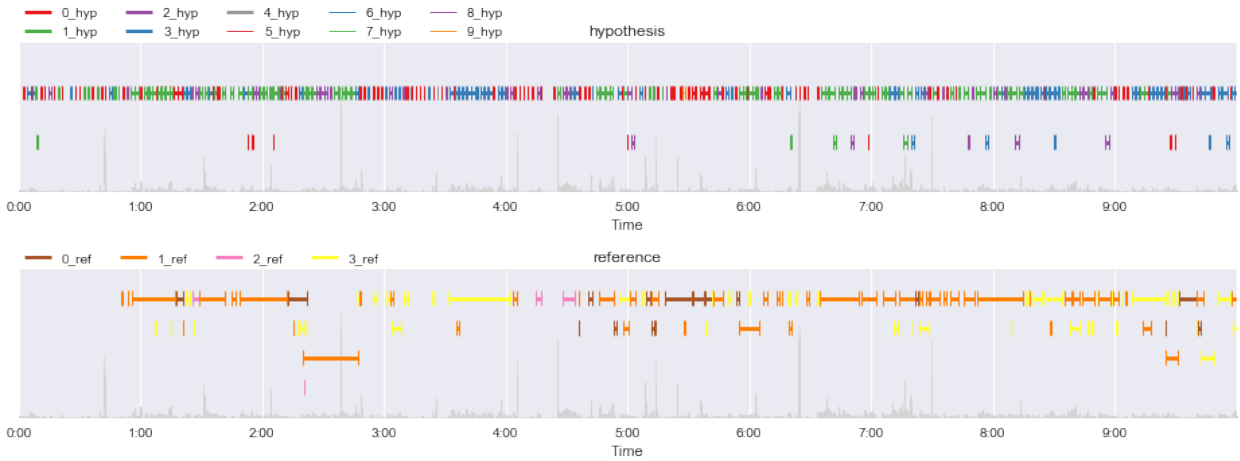


Рисунок 20. Сравнение результатов работы алгоритма с данными датасета для аудиозаписи AMI.ES2002.a

Однако на аудиозаписях с небольшим количеством дикторов (2-3) и заметным уровнем шума (лай собаки, звон посуды) алгоритм показывает уверенные результаты. В дополнение к исходному датасету, были проведены тесты с использованием небольшого количества фрагментов кинофильмов, на которых запечатлены диалоги героев в различных условиях (рис. 21). Данные о реальных значениях временных промежутков и дикторах извлекались из размещённых в свободном доступе субтитров, обработанных в используемый формат.

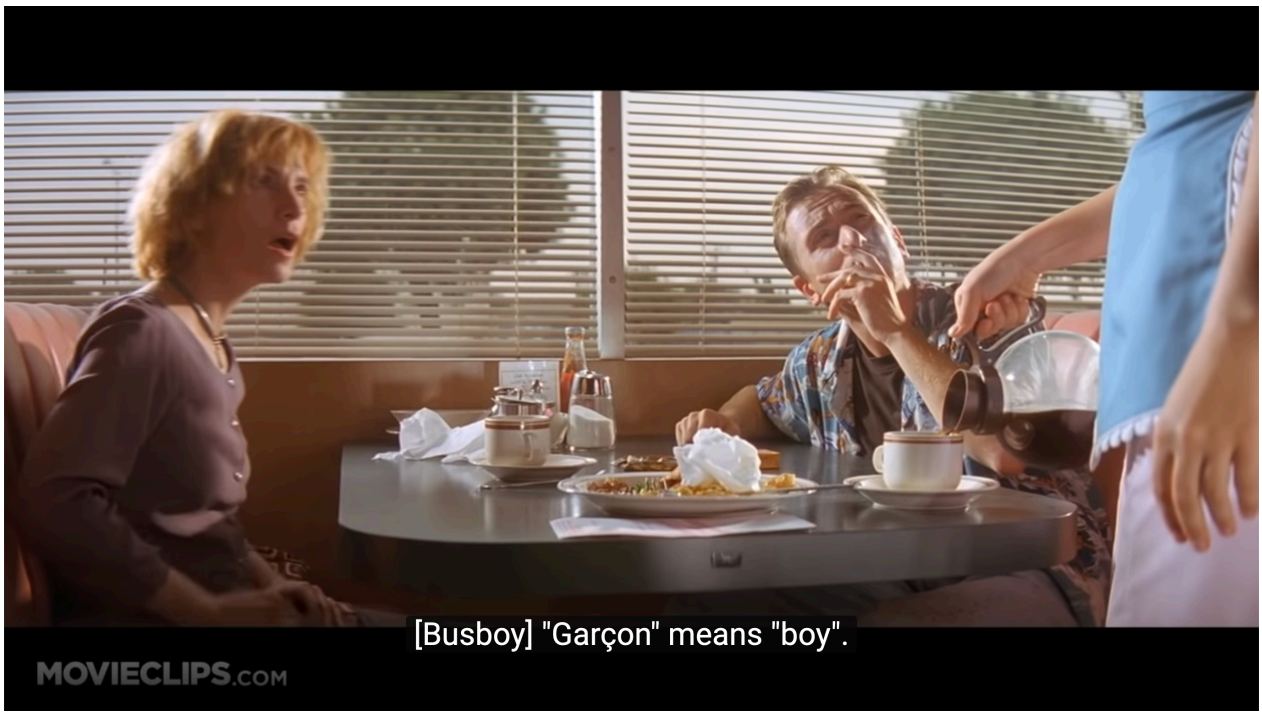


Рисунок 21. Кадр из художественного фильма Pulp Fiction

Лучшая картина была получена (рис. 22) для сцены диалога двух персонажей в кафе из художественного фильма Pulp Fiction. Данную запись характеризует высокий уровень фонового шума (на фоне диалога часто слышно посторонние голоса и шум посуды), а в конце герои переходят на крик. Однако голос молодого человека (*0_hyp*, *0_ref*) был верно распознан практически во всех фразах, и лишь некоторые из них были отнесены девушке (*1_ref* включает 0_*hyp* и *1_hyp*).

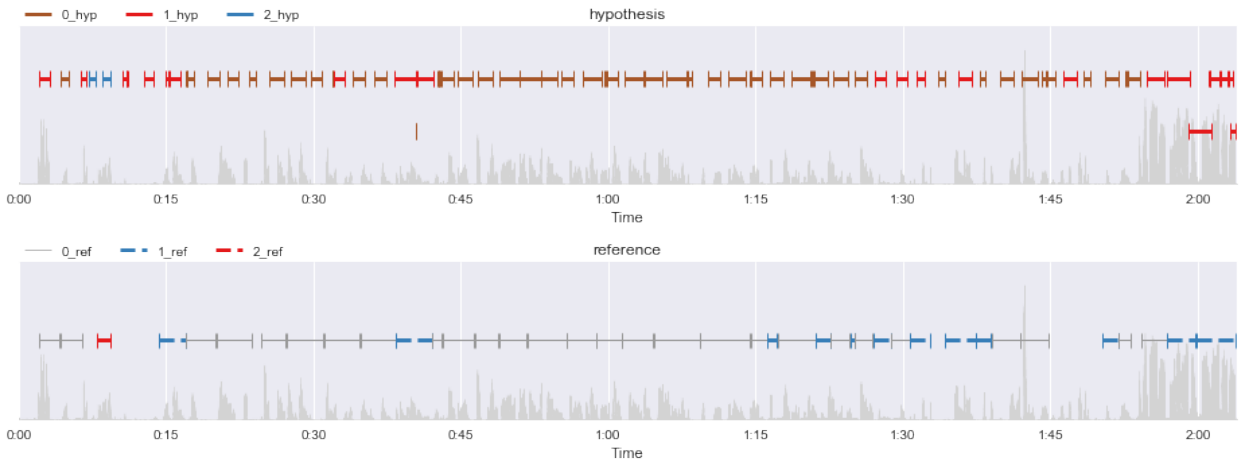


Рисунок 22. Сравнение результатов диаризации диалога персонажей с данными из файла субтитров

Также можно заметить, что алгоритм распознал множество пауз, отсутствующих в данных, извлечённых из субтитров. Это связано с тем, что

зачастую время реплик в субтитрах указывается приблизительно, без учёта пауз в речи, поэтому вычисление точного значения ошибки DER в данном случае было бы некорректным.

Несмотря на то, что алгоритм показывает неточные результаты при определении отдельных дикторов, речь которых наименее представлена на аудиозаписях, полученные показатели в целом удовлетворяют поставленным условиям. В дальнейшем развитии работы необходимо обратить внимание на факторы, в первую очередь связанные с ошибками алгоритма распознавания речи из-за значимого фонового шума и отсутствием какой-либо предварительно информации о голосах дикторов, с которой можно было бы сравнивать характеристики, извлекаемые из выделенных сегментов.

В ходе испытаний была установлена оптимальная длина скользящего окна: 2.5 секунд с пересечением 15%. Время обработки одного сегмента длиной 2.5 секунды составило в среднем 0.12 секунд.

Среднее значение ошибки DER, полученное для датасета AMI: 31.6%. В таблице 1 приведены результаты для некоторых аудиозаписей из датасета с данными о количестве дикторов и полученной ошибке диаризации.

Идентификатор аудиозаписи	**Количество дикторов**	**Ошибка DER**
ES2002.a	4	35.4%
ES2002.b	4	31.3%
ES2002.c	4	37.1%
IS1001.a	3	27.5%
IS1001.b	3	25.2%
IS1001.c	3	24.8%

Таблица 1. Вычисленные значения ошибки DER для некоторых аудиозаписей датасета AMI

Заключение

В рамках данной работы была исследована задача диаризации аудиопотока в режиме реального времени. Рассмотрены основные подходы к её решению и описаны значимые характеристики и методы, применяемые для этого. Произведён анализ доступных программных решений.

Также было предложено программное решение, для которого были установлены такие ограничения как автономность алгоритма и низкая требовательность к ресурсам устройства, что позволяет использовать его в устройствах «умного дома».

Для реализованного алгоритма были произведены сравнительные тесты и анализ полученных результатов с применением соответствующей метрики, предназначенной для данной задачи. Несмотря на то, что реализация уступает неавтономным аналогам, использующим алгоритмы анализа «с учителем» и возможности платформ с дискретными графическими подсистемами, полученные результаты говорят об эффективном решении данной задачи даже на устройствах с низкой производительностью.

Список литературы

1. Huang X., Acero A., Hon. H. Spoken Language Processing: A Guide to Theory, Algorithm, and System Development. // Prentice Hall, 2001, ISBN: 0130226165
2. Anguera X., Bozonnet S., Evans N., Fredouille C., Friedland G., Vinyals O. Speaker Diarization: A Review of Recent Research. // IEEE Transactions on Audio, Speech, and Language Processing, 2012, Vol. 20, No. 2, PP. 356–370.
3. Yin R., Bredin H., Barras C. Speaker Change Detection in Broadcast TV using Bidirectional Long Short-Term Memory Networks. // [Электронный ресурс] https://pdfs.semanticscholar.org/edff/b62b32ffcc2b5cc846e26375cb300fac9ecc.pdf
4. Yin R., Bredin H., Barras C. Neural speech turn segmentation and affinity propagation for speaker diarization. // Proc. Interspeech 2017, PP. 3827-3831. [Электронный ресурс] https://www.isca-speech.org/archive/Interspeech_2018/pdfs/1750.pdf
5. Will Koehrsen. Neural Network Embeddings Explained. // Towards Data Science. [Электронный ресурс] https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526
6. Dimitriadis D., Fousek P. Developing On-Line Speaker Diarization System. // Proc. Interspeech, 2017, PP. 2739-2743. [Электронный ресурс] https://www.researchgate.net/publication/319185396_Developing_On-Line_Speaker_Diarization_System
7. Sell G., Garcia-Romero D. Speaker diarization with PLDA i-vector scoring and unsupervised calibration. // IEEE Spoken Language Technology Workshop (SLT), 2014, PP. 413-417
8. Pelleg D., Moore A. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. // 17th Intern. Conf. on Machine Learning, 2000, PP. 727–734.

9. Garcia-Romero D., Snyder D., Sell S., Povey D., McCree A. Speaker diarization using deep neural network embeddings. // Proc. International Conference on Acoustics, Speech and Signal Processing, 2017, PP. 4930–4934. [Электронный ресурс] https://www.danielpovey.com/files/2017_icassp_diarization_embeddings.pdf
10. Asawa C., Bhattasali N., Jiang A. Deep Learning Approaches for Online Speaker Diarization. // [Электронный ресурс] https://pdfs.semanticscholar.org/e1d7/9befa38a3eb1a29195f838ca0fe010a60c96.pdf
11. Wang Q., Downey C., Wan L., Mansfield P. A., Lopez Moreno I. Speaker diarization with LSTM. // International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2018, PP. 5239–5243. [Электронный ресурс] https://arxiv.org/pdf/1710.10468.pdf
12. Wan L., Wang Q., Papir A., Lopez Moreno I. Generalized end-to-end loss for speaker verification. // [Электронный ресурс] https://arxiv.org/pdf/1710.10467.pdf
13. Zhang A., Wang Q., Zhu Z., Paisley J., Wang C. Fully Supervised Speaker Diarization. // International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, PP. 6301–6305. [Электронный ресурс] https://arxiv.org/abs/1810.04719
14. Blei D. M., Frazier P. I. Distance Dependent Chinese Restaurant Processes. // Journal of Machine Learning Research, Vol. 12, 2011, PP. 2383–2410. [Электронный ресурс] http://www.cs.columbia.edu/~blei/papers/BleiFrazier2011.pdf
15. Dimitriadis D. Enhancements for Audio-only Diarization Systems. // ArXiv, abs/1909.00082. [Электронный ресурс] https://arxiv.org/abs/1909.00082

16. Волченков В. А., Витязев В. В. Методы и алгоритмы детектирования активности речи. // Цифровая обработка сигналов. – 2013. – No 1. – С. 54–60.
17. S. Davis and P. Mermelstein. Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. // IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 28 No. 4, 1980.
18. D. A. Reynolds, Richard C. Rose. Robust text-independent speaker identification using Gaussian Mixture speaker models. // IEEE Transactions on Speech and Audio Processing, Vol. 3, No. 1, PP. 72–83.
19. Soldi G., Beaugeant C., Evans N. Adaptive and online speaker diarization for meeting data. // 23rd European Signal Processing Conference (EUSIPCO), 2015, PP. 2112–2116. [Электронный ресурс] http://www.eurecom.fr/en/publication/4617/download/mm-publi-4617.pdf
20. Dehak N., Kenny P., Dehak R., Dumouchel P., Ouellet P. Front-End Factor Analysis for Speaker Verification. // IEEE Transactions on Audio, Speech, and Language Processing, 2010.
21. Snyder D., Garcia-Romero D., Povey D., Khudanpur S. Deep Neural Network Embeddings for Text-Independent Speaker Verification. // Proc. Interspeech 2017, PP. 999–1003. [Электронный ресурс] http://danielpovey.com/files/2017_interspeech_embeddings.pdf
22. Snyder D., Garcia-Romero D., Sell G., Povey D., Khudanpur S. X-vectors: robust DNN embeddings for speaker recognition. // IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018, PP. 5329–5333. [Электронный ресурс] https://www.danielpovey.com/files/2018_icassp_xvectors.pdf
23. Kenny P., Boulianne G., Dumouchel P. Eigenvoice modeling with sparse training data. // IEEE Transactions on Speech Audio Processing, Vol. 13, No. 3, 2005.

24. Purohit K. Building Speaker Recognition Systems and Diarization Using d-vectors. // Medium. [Электронный ресурс] https://medium.com/saarthi-ai/using-d-vector-for-speaker-recognition-and-diarization-4a3450dd8a01
25. Zajıc Z., Hruz M., Muller L. Speaker Diarization Using Convolutional Neural Network for Statistics Accumulation Refinement. // Proc. Interspeech 2017, PP. 3562-3566. [Электронный ресурс] https://pdfs.semanticscholar.org/35c4/0fde977932d8a3cd24f5a1724c9dbca8b38d.pdf
26. Ning H., Liu M., Tang H., Huang T. A Spectral Clustering Approach to Speaker Diarization. // Proc. ICSLP, 2006. [Электронный ресурс] http://www.ifp.illinois.edu/~hning2/papers/Ning_spectral.pdf
27. Anguera X. Diarization Error Rate. // Xavier Anguera personal page. [Электронный ресурс] http://www.xavieranguera.com/phdthesis/node108.html
28. Newman L. H. The Privacy Battle to Save Google From Itself. // Wired. ISSN 1059-1028. [Электронный ресурс] https://www.wired.com/story/google-privacy-data/
29. Shachtman N. 'Don't Be Evil,' Meet 'Spy on Everyone': How the NSA Deal Could Kill Google. // Wired. ISSN 1059-1028. [Электронный ресурс] https://www.wired.com/2010/02/from-dont-be-evil-to-spy-on-everyone/
30. How to Make Your Own Open-Source Voice Assistant With Raspberry Pi. // PCMag. [Электронный ресурс] https://www.pcmag.com/how-to/how-to-make-your-own-open-source-voice-assistant-with-raspberry-pi
31. How to build your own private smart home with a Raspberry Pi and Mozilla's Things Gateway. // Mozilla. [Электронный ресурс] https://hacks.mozilla.org/2018/02/how-to-build-your-own-private-smart-home-with-a-raspberry-pi-and-mozillas-things-gateway/

32. Raspberry Pi Smart Speaker and Roll-Your-Own Services. // Medium. [Электронный ресурс] https://medium.com/@syed.r.ali/raspberry-pi-smart-speaker-and-roll-your-own-services-dfdaddb6cc85
33. Bredin H. pyannote.metrics: a toolkit for reproducible evaluation, diagnostic, and error analysis of speaker diarization systems. // Hypothesis, Vol. 100, No. 60, PP. 90, 2017 [Электронный ресурс] http://herve.niderb.fr/download/pdfs/Bredin2017a.pdf
34. Carletta J. Unleashing the killer corpus: experiences in creating the multi-everything AMI Meeting Corpus. // Language Resources and Evaluation Journal, Vol. 41, No. 2, PP. 181–190. [Электронный ресурс] http://homepages.inf.ed.ac.uk/jeanc/carletta.LREC-keynote06.pdf

Приложение 1. Исходный код скрипта для обработки датасета AMI Corpus

```
from operator import itemgetter
from lxml import etree
import xml.etree.ElementTree as ET
import json


def parse_ami_transcript_xml(file, speaker_id):
    """
    Parsing AMI transcript XML file
    """

    xmlp = ET.XMLParser(encoding="ISO-8859-1")
    f = ET.parse(file, parser=xmlp)

    # with open(file) as f:
    #     xml = f.read()

    root = f.getroot()

    transcript = []

    for element in list(root):
        # print(element)
        if element.tag == 'w':
            if element.text:
                text = element.text
            else:
                text = ''

            if element.get('punc'):
                punc = True
            else:
                punc = False

            transcript.append({
                'start': float(element.get('starttime')),
                'end': float(element.get('endtime')),
                'text': text,
                'punc': punc,
                'speaker_id': speaker_id
            })

    previous = transcript[0]
    for index, elem in enumerate(transcript):
        if elem['start'] == previous['end']:
            previous['end'] = elem['end']
            previous['text'] += ('' if elem['punc'] else ' ') + elem['text']
            transcript[index] = None
        else:
            del elem['punc']
            previous = elem

    transcript = [t for t in transcript if t and len(t['text'])]

    return transcript
```

```
if __name__ == "__main__":
    speakers_phrases = []

    for index, speaker in enumerate(['A', 'B', 'C', 'D']):
        t = parse_ami_transcript_xml(
            "ami-corpus/ES2002a.{}.words.xml".format(speaker), index)
        speakers_phrases.extend(t)

    final = sorted(speakers_phrases, key=itemgetter('start'))

    with open('ami-corpus/transcripts.json', 'w') as f:
        json.dump(final, f)
```

Приложение 2. Jupyter Notebook с примером проведённого теста

Imports

In [1]:

```
from IPython.display import Audio, display

from itertools import groupby
import itertools
flatten = itertools.chain.from_iterable
from math import ceil
from pathlib import Path
import datetime
import json
import collections
from timeit import default_timer as timer
import os

from resemblyzer import preprocess_wav, VoiceEncoder
import matplotlib.pyplot as plt
import librosa
import librosa.display
import matplotlib
from matplotlib import style
import numpy as np
from numpy import linalg as LA
import sounddevice as sd
import webrtcvad
from pyannote.core import Segment, Timeline, Annotation, notebook
from pyannote.metrics.diarization import DiarizationErrorRate, GreedyDiarization
ErrorRate, JaccardErrorRate
import scipy
from scipy.sparse import csgraph
from scipy.spatial.distance import pdist, squareform, euclidean, cosine
import scipy.signal as sps
import pydub
from webrtcvad_wrapper import VAD
import pydub.scipy_effects
from pydub import AudioSegment
import soundfile as sf

AudioSegment.converter = '/usr/local/Cellar/ffmpeg/4.2.2_2/bin/ffmpeg'
```

Constants and functions definitions

In [2]:

```
## Plots preparation

style.use('seaborn')

color = {
    -1: "black",
    0: "red",
    1: "orange",
    2: "blue",
    3: "brown",
    4: "purple",
    5: "plum",
    6: "gray",
    7: "khaki",
    8: "magenta",
    9: "cyan",
    10: "lime",
    11: "indigo",
    12: "royalblue",
    13: "dodgerblue",
    14: "lightcoral"
}

def timeTicks(x, pos):
    d = datetime.timedelta(seconds=x)
    return str(d)[-5:] if x < 3600 else str(d)
formatter = matplotlib.ticker.FuncFormatter(timeTicks)
```

In [3]:

```
def get_vad_segments(audio_segment):
    filtered_segments = vad.filter(audio_segment)

    return [
        (filtered_segment[0], filtered_segment[1])
        for filtered_segment in filtered_segments if filtered_segment[-1]
    ]

def remove_overlap(ranges):
    result = []
    current_start = -1
    current_stop = -1

    for start, stop in sorted(ranges):
        if start > current_stop:
            # this segment starts after the last segment stops
            # just add a new segment
            result.append( (start, stop) )
            current_start, current_stop = start, stop
        else:
            # segments overlap, replace
            result[-1] = (current_start, stop)
            # current_start already guaranteed to be lower
            current_stop = max(current_stop, stop)

    return result
```

In [4]:

```
def plot_original_audio(speaker_segments):
    # display(Audio(wav, rate=source_sr))

    fig = plt.figure(figsize=(13, 3))
    axes = plt.axes()
    plt.title('Original audio')
    librosa.display.waveplot(original_audio * 2, sr=source_sr, color='lightgray'
)

    for line in speaker_segments:
        plt.plot(
            [x[line['start']], x[line['end']]],
            ['Speaker ' + str(line['speaker_id']), 'Speaker ' + str(line['speake
r_id'])],
            color=color[line['speaker_id'] % 15],
            linewidth=3
        )

    if SHOW_SPEAKER_SWITCHES:
        for xc in speaker_switches:
            plt.axvline(x=xc, linewidth=.5)

    axes.xaxis.set_major_formatter(formatter)
    ticks_freq = ceil(len(wav) * .1 / source_sr)
    axes.xaxis.set_major_locator(plt.MultipleLocator(ticks_freq))

    plt.margins(x=0.01)
    plt.tight_layout()
```

In [5]:

```
def plot_ground_truth():
    fig = plt.figure(figsize=(13, 3))
    axes = plt.axes()
    plt.title('Ground truth')
    librosa.display.waveplot(original_audio * 2, sr=source_sr, color='lightgray'
)

    for item in ami_corpus_transcripts:
        id = item['speaker_id']
        plt.plot(
            [item['start'], item['end']],
            ['Speaker ' + str(id), 'Speaker ' + str(id)],
            color=color[id % 15],
            linewidth=4
        )

    if SHOW_SPEAKER_SWITCHES:
        for xc in speaker_switches:
            plt.axvline(x=xc, linewidth=.5)

    axes.xaxis.set_major_formatter(formatter)
    ticks_freq = ceil(len(original_audio) * .1 / source_sr)
    axes.xaxis.set_major_locator(plt.MultipleLocator(ticks_freq))

    plt.margins(x=0.01)
    plt.tight_layout()
```

In [6]:

```
def get_speaker_segments(labels_list):
    N = np.array(labels_list)
    counter = np.arange(1, np.alen(N))
    groupings = np.split(N, counter[N[1:] != N[:-1]])

    segments = []
    start = 0
    for group in groupings:
        if len(group) > 0 and group[0] is not None:
            segments.append({
                'start': start,
                'end': start + len(group) - 1,
                'speaker_id': group[0]
            })
        start += len(group)

    return segments
```

In [7]:

```
def get_similarity(embed_i, embed_j):
    return cosine(embed_i, embed_j)
```

In [8]:

```
def get_hypothesis(speaker_segments):
    hypothesis = Annotation()
    for t in speaker_segments:
        try:
            hypothesis[Segment(x[t['start']], x[t['end']])] = str(t['speaker_id'
]) + '_hyp'
        except:
            pass

    return hypothesis

der = DiarizationErrorRate()
gder = GreedyDiarizationErrorRate()
jer = JaccardErrorRate()

def measure_metrics(reference, hypothesis):
    der_value = der(reference, hypothesis)
    print('DER:', der_value)
    # print(der.report())

    ## Gives no difference with DER
    # print('GDER:', gder(reference, hypothesis))
    # print(gder.report())

    print('JER:', jer(reference, hypothesis))
    # print(jer.report())

    return der_value
```

In [9]:

```
def plot_der(hypothesis):
    fig = plt.figure(figsize=(13, 5))

    # plot hypothesis
    ax = plt.subplot(211)
    plt.title('hypothesis')
    ax.xaxis.set_major_formatter(formatter)
    ticks_freq = ceil(len(original_audio) * .1 / source_sr)

    notebook.plot_annotation(hypothesis, legend=True, time=True)
    librosa.display.waveplot(original_audio, sr=source_sr, color='lightgray')

    # plot reference
    ax = plt.subplot(212)
    plt.title('reference')
    ax.xaxis.set_major_formatter(formatter)

    notebook.plot_annotation(reference, legend=True, time=True)
    librosa.display.waveplot(original_audio, sr=source_sr, color='lightgray')

    plt.margins(x=0.01)
    plt.tight_layout()
    plt.show()
```

In [10]:

```
# SOURCE_FOLDER = 'ami_corpus'
# SOURCE_FILE = 'ES2002a.Mix-Headset.wav'
# TRANSCRIPTS = f'{SOURCE_FOLDER}/transcripts.json'
# N_SPEAKERS = 4

SOURCE_FOLDER = 'records'
SOURCE_FILE = 'Pumpkin_and_Honey_Bunny.wav'
TRANSCRIPTS = f'{SOURCE_FOLDER}/transcripts.json'
N_SPEAKERS = 3

FILEPATH = str(Path(SOURCE_FOLDER, SOURCE_FILE))
SR = 32000

CHUNKS_FOLDER = 'chunks'

CUT_AUDIO = True
CUT_LENGTH = 10. * 60. # seconds
```

Audio file loading

In [11]:

```
## Load audio

if CUT_AUDIO:
    wav, source_sr = librosa.load(FILEPATH, sr=SR, offset=0.0, duration=CUT_LENG
TH)
else:
    wav, source_sr = librosa.load(FILEPATH, sr=SR)

original_audio = wav
print(len(wav))
```

```
3965361
```

Loading transcripts

In [12]:

```
## Load transcripts from prepared file and mark reference annotations

with open(TRANSCRIPTS, 'r') as f:
    ami_corpus_transcripts = json.load(f)

SHOW_SPEAKER_SWITCHES = False

if SHOW_SPEAKER_SWITCHES:
    if CUT_AUDIO:
        speaker_switches = [t['start'] for i, t in enumerate(ami_corpus_transcri
pts) \
            if i > 0 \
                and t['speaker_id'] != ami_corpus_transcripts[i - 1]['speaker_i
d'] \
                and t['end'] < CUT_LENGTH
        ]
    else:
        speaker_switches = [t['start'] for i, t in enumerate(ami_corpus_transcri
pts) \
            if i > 0 \
                and t['speaker_id'] != ami_corpus_transcripts[i - 1]['speaker_i
d']
        ]

    speaker_switches.insert(0, ami_corpus_transcripts[0]['start'])

if CUT_AUDIO and CUT_LENGTH < len(wav):
    ami_corpus_transcripts = [t for t in ami_corpus_transcripts if t['start'] <
CUT_LENGTH]

reference = Annotation()
for t in ami_corpus_transcripts:
    try:
        reference[Segment(t['start'], t['end'])] = str(t['speaker_id']) + '_ref'
    except:
        pass
```

In [13]:

```
# Prepare everything for clustering
```

In [14]:

```
timers = []

x = librosa.times_like(len(wav), sr=SR, hop_length=1)
```

In [15]:

```
## Load voice encoder

encoder = VoiceEncoder()
```

```
Loaded the voice encoder model on cpu in 0.01 seconds.
```

In [16]:

```
CHUNK_TIME_LENGTH = 2.5 # seconds
CHUNKS_OVERLAP = .15 # share (percents)
SPEAKER_CHANGE_THRESHOLD = .3  # percents
SENSITIVITY_MODE = 2 # 4 is supposed to be better for neural networks, but it's
 not
FILTER_VALUE = 300
```

In [17]:

```
## Define Voice Activity Detector

vad = VAD(sensitivity_mode=SENSITIVITY_MODE)
```

In [18]:

```
## Clean chunks folder

filelist = [f for f in os.listdir(CHUNKS_FOLDER)]
for f in filelist:
    os.remove(os.path.join(CHUNKS_FOLDER, f))

chunk_size = ceil(CHUNK_TIME_LENGTH * SR)

embeds = []
speakers_ids = [0]
speaker_segments = []
vad_segments = []

counter = 0
```

Chunks processing

In [19]:

```
## Process overlapping chunks
for i in range(0, len(wav), ceil(chunk_size * (1 - CHUNKS_OVERLAP))):
    t_start = timer()

    start = i
    end = i + chunk_size if i + chunk_size < len(wav) - 1 else len(wav) - 1

    chunk = wav[start:end]
    chunk = preprocess_wav(chunk)[0]

    if (len(chunk) > 0):
        filename = f'chunks/chunk_{i}.wav'
        sf.write(filename, chunk, samplerate=SR)
        audio = AudioSegment.from_file(filename)

        if audio.sample_width != 2:
            audio = audio.set_sample_width(2)
        if audio.channels != 1:
            audio = audio.set_channels(1)

        ## Apply 3rd order high pass filter with given value
        audio = audio.high_pass_filter(FILTER_VALUE, order=3)

        segments = get_vad_segments(audio)

        vad_segments += [(start + ceil(segment[0] * SR), start + ceil(segment[1]
* SR)) for segment in segments]

        new_segments = [s for s in remove_overlap(vad_segments) if s[1] - s[0] >
.2 * SR]
        segments = [(s[0] - start, s[1] - start) for s in new_segments if s[0] >
= start]

        for segment in segments:
            seg_start, seg_end = segment
            frame = chunk[seg_start:seg_end]
            embed = encoder.embed_utterance(frame)

            best = 0

            if len(embeds) > 0:
                distances = []
                for speaker_id, speaker_embeddings in groupby(sorted(embeds, key
=lambda x: x[1]), lambda x: x[1]):
                    es = [e for e, s_id in list(speaker_embeddings)]

                    mean_dist = np.mean([get_similarity(embed, embedding) for em
bedding in es])
                    min_dist = np.min([get_similarity(embed, embedding) for embe
dding in es])
                    max_dist = np.max([get_similarity(embed, embedding) for embe
dding in es])

                    distances.append((min_dist, speaker_id))

                distances = sorted(distances, key=lambda x: x[0])

                best_id = distances[0][1]
```

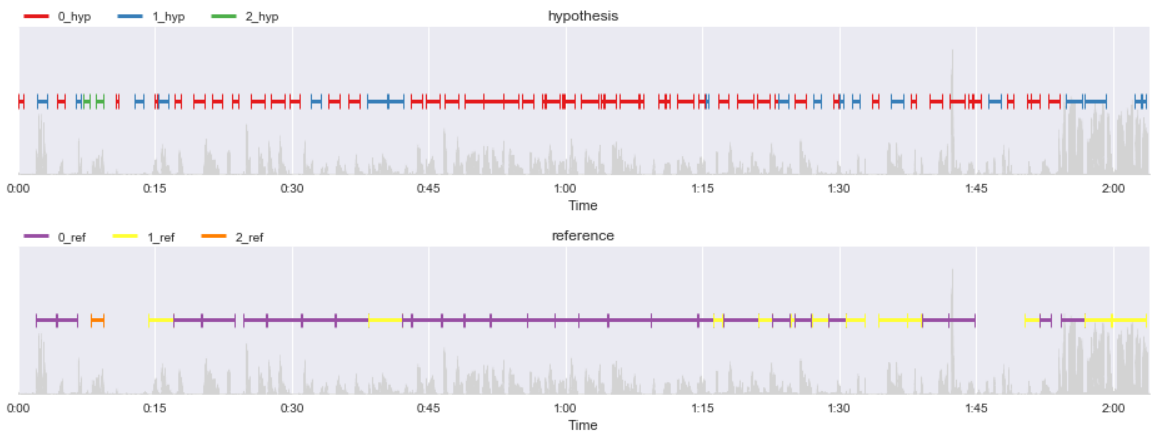
```
                if distances[0][0] < SPEAKER_CHANGE_THRESHOLD:
                    id = best_id
                else:
                    counter += 1
                    id = counter

                speakers_ids.append(id)

                speaker_segments.append({
                    'start': start + seg_start,
                    'end': start + seg_end,
                    'speaker_id': id
                })

            embeds.append((embed, speakers_ids[best]))

    t_end = timer()
    timers.append(t_end - t_start)
print(np.mean(timers))
```

```
0.10021170491666688
```

Metrics measuring

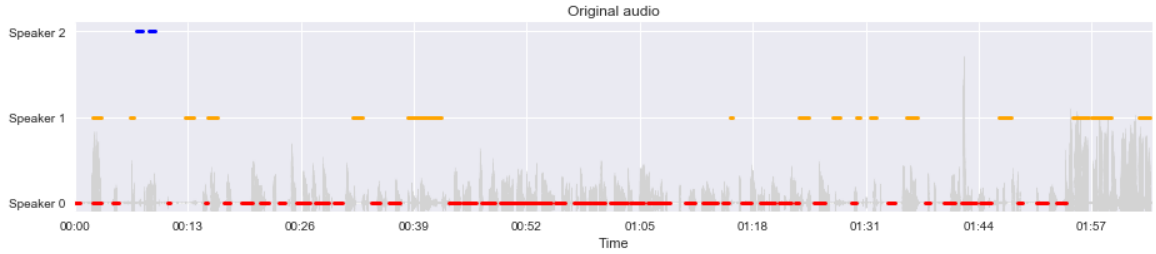
In [20]:

```
hypothesis = get_hypothesis(speaker_segments)
# der_value = measure_metrics(reference, hypothesis)

## PyAnnote DER and other metrics

%matplotlib inline

plot_der(hypothesis)
```



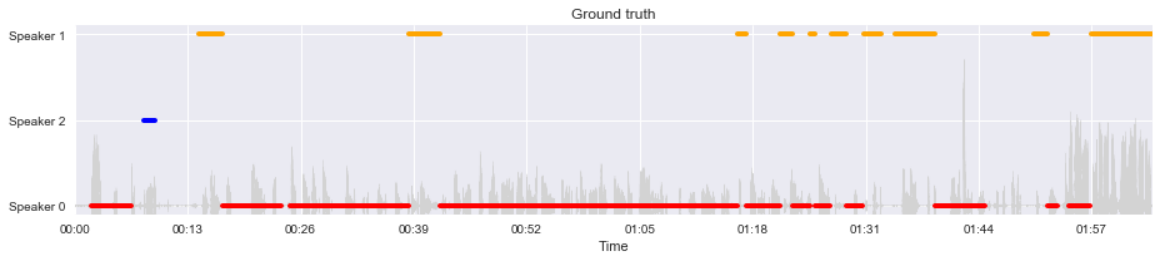
In [21]:

```
%matplotlib inline
plot_original_audio(speaker_segments)
```



In [22]:

```
%matplotlib inline
plot_ground_truth()
```

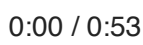


Diarization results: audios of unique speakers

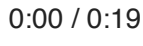
In [23]:

```
for speaker_id, segs in groupby(sorted(speaker_segments, key=lambda x: x['speaker_id']), lambda x: x['speaker_id']):
    print(speaker_id)
    audio = np.concatenate([wav[s['start']:s['end']] for s in segs], axis=0)
    print(audio.shape)
    display(Audio(audio, rate=SR))
```

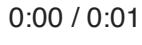
```
0
(1696640,)
```




```
1
(633600,)
```




```
2
(46720,)
```



In [ ]: