ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Факультет информатики, математики и компьютерных наук**

**Программа подготовки бакалавров по направлению   
38.03.05 Бизнес-информатика**

Верховцева Анастасия Григорьевна

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Разработка прототипа системы распознавания голосовых команд на основе аппарата искусственных нейронных сетей

|  |  |
| --- | --- |
| Рецензент  доцент, кандидат технических наук  В.М.Демкин | Научный руководитель  доцент, кандидат технических наук  А.В.Савченко |

Нижний Новгород, 2016

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc452424044)

[ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7](#_Toc452424045)

[1.1. Классификация систем автоматического распознавания речи 7](#_Toc452424046)

[1.2. Подходы к решению задачи распознавания речи 9](#_Toc452424047)

[1.3. Модели, используемые в задаче распознавания речи 11](#_Toc452424048)

[1.3.1. Скрытые Марковские модели 11](#_Toc452424049)

[1.3.2. Искусственные нейронные сети 13](#_Toc452424050)

[1.4. Обзор библиотек для автоматического распознавания речи 15](#_Toc452424051)

[ГЛАВА 2. РЕАЛИЗАЦИЯ СИСТЕМЫ РАСПОЗНАВАНИЯ ГОЛОСОВЫХ КОМАНД 20](#_Toc452424052)

[2.1. Установка и настройка Kaldi 20](#_Toc452424053)

[2.2. Определение параметров онлайн-декодера 22](#_Toc452424054)

[2.3. Реализация программы 25](#_Toc452424055)

[ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНОЕ СРАВНЕНИЕ РАЗЛИЧНЫХ АКУСТИЧЕСКИХ МОДЕЛЕЙ 34](#_Toc452424056)

[3.1. Подготовка аудио-данных 34](#_Toc452424057)

[3.2. Проведение экспериментов 35](#_Toc452424058)

[ЗАКЛЮЧЕНИЕ 39](#_Toc452424059)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 42](#_Toc452424060)

[ПРИЛОЖЕНИЯ 47](#_Toc452424061)

[Приложение 1. Листинг программы 47](#_Toc452424062)

[Приложение 2. Скрипт для расчета уровня ошибок 60](#_Toc452424063)

[распознавания 60](#_Toc452424064)

# ВВЕДЕНИЕ

Человек всегда стремился к более удобному и естественному способу взаимодействия с компьютером. Взаимодействие с первыми ЭВМ было доступно только обученным специалистам – программистам. Затем, с появлением диалогового интерфейса, пользователю представилась возможность без специальных знаний самостоятельно вводить с устройства ввода адресованную компьютеру команду и получать ответное действие. Именно появление графического интерфейса послужило толчком к повсеместному распространению персональных компьютеров.

Следующим шагом стали попытки реализации речевого интерфейса, то есть возможности управления компьютером посредством голоса. В настоящее время данное направление не только не потеряло актуальности, но и привлекает большое внимание ученых и специалистов по всему миру, находя для себя множество областей для практического применения.

Однако если сравнить показатели эффективности современных систем распознавания с показателями систем времен зарождения этой области науки, то можно сказать, что за прошедшие десятки лет исследователи продвинулись недалеко. Это заставляет некоторых специалистов сомневаться относительно возможности реализации речевого интерфейса в ближайшем будущем. Другие считают, что задача уже практически решена. Большинство экспертов сходится во мнении, что для развития распознавания речи потребуется какое-то время.

Предельные возможности компьютера в данной области связаны, прежде всего, с тем, что ему не дано в полной мере распознавать осмысленную речь в отличие от человека, который может быть принят за эталон распознающей системы. Компьютер принципиально не может с требуемой надежностью исправлять ошибки и неоднозначности распознавания, применяя семантическую и синтаксическую связь слов предложения, а также отделять помехи от речи. Слуховой аппарат человека позволяет ему достаточно точно определить направление на источник полезного сигнала и отделить его от остальных звуковых источников, являющихся помехами. Фонетические модели, используемые в программировании алгоритмов на компьютерах, не точны, так как не используют всего многообразия факторов. Для задания фонетических эталонов обычно используют статистические методы, предполагающие, что акустические параметры фонем распределены по нормальному закону. В реальности картина намного сложнее, что приводит к тому, что точная модель эталонов звуков и слов должна включать в себя множество эталонных элементов (различных для каждого варианта произнесения).

В данной работе рассматриваются системы распознавания изолированных слов, а именно речевых команд. Возросшая необходимость таких систем обусловлена в первую очередь появлением большого количества доступных человеку разнообразных устройств – персональных компьютеров, планшетов и мобильных телефонов, игровых и других мультимедийных устройств. Также растёт важность повсеместного внедрения новых способов взаимодействия человека с различными устройствами, поскольку традиционные интерфейсы перестают отвечать возросшим требованиям современного человека и его ритму жизни.

Кроме того, системы распознавания речи представляют особую важность для людей с ограниченными возможностями, и эта область их применения активно развивается в части персональных компьютеров, мобильной телефонии, а также в бытовой технике. Для помощи таким людям производители обеспечивают свои устройства возможностью голосового управления, а также произнесения введенного текста голосом. И в первую очередь от таких продуктов требуется распознавание ограниченного набора команд пользователя, а не непрерывной речи с большим или вовсе неограниченным словарём.

Благодаря стандартизации платформ и операционных систем расширяется круг сторонних разработчиков программных продуктов, обладающих данной функциональностью. Однако значительным недостатком большинства существующих решений является произведение распознавания в режиме онлайн и использование внешних сервисов, таких как Google, Yandex и др., что создает необходимость постоянного подключения к интернету. Другие системы, распознающие речь в режиме оффлайн, обладают низкой производительностью и используют устаревшее программное обеспечение.

Большой круг исследователей работает над улучшением качества распознавания речи. В частности, Савченко А.В. и Савченко Л.В. в статье Towards the creation of reliable voice control system based on a fuzzy approach [35] предлагают методы, агрегируемые с GMM (Gaussian Mixture Models). Добиться еще лучшей производительности возможно путем агрегации данных методов с DNN (Deep Neural Networks). Для этого необходимо выбрать подходящий современный инструментарий по распознаванию речи и экспериментально подобрать наилучшую акустическую модель, что также обуславливает актуальность проводимого в рамках дипломной работы исследования.

Таким образом, целью дипломной работы является разработка прототипа системы распознавания голосовых команд на основе аппарата искусственных нейронных сетей с использованием библиотеки Kaldi - инструментария для распознавания речи, написанного на языке C ++ под лицензией Apache v2.0. Результатом должна являться автономная система голосового управления компьютером, распознающая произнесенные пользователем команды из списка доступных команд и выполняющая соответствующее команде действие. Разработанная система должна отвечать следующим требованиям:

Функциональные требования к системе:

* Распознавание текста на английском языке
* Голосовое управление отдельными функциями операционных систем (Linux)
* Голосовое управление функциями прикладных программ
* Система непрерывно слушает вход микрофона

Нефункциональные требования:

* Автономность системы
* Независимость от диктора
* Устойчивость к шуму и помехам
* Защита от ложного срабатывания.

В соответствии с данной целью можно выделить следующие задачи:

* Изучить литературу по предметной области
* Провести сравнительный анализ существующих инструментариев для распознавания речи
* Исследовать различные алгоритмы обучения акустических моделей
* Разработать прототип системы голосового управления на основе выбранного инструментария
* Провести экспериментальное сравнение различных акустических моделей в рамках выбранного инструментария

В работе описаны основные подходы к решению проблемы, классификация систем распознавания речи по различным критериям, модели, используемые для решения задачи распознавания, а также производится обзор существующих систем. В практической части работы описана разработка системы распознавания голосовых команд на основе выбранной библиотеки, а также представлены результаты экспериментального сравнения различных алгоритмов.

# ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Классификация систем автоматического распознавания речи

Большинство систем автоматического распознавания речи (АРР) состоит из процесса анализа и обработки аналогового сигнала и процесса распознавания. При анализе аналогового сигнала из речи выделяются свойства, которые используются далее в процессе распознавания для того, чтобы определить, что было сказано.

Самые первые попытки создания АРР систем были предприняты в 1950-х годах. С 90-х годов были произведены значительные усовершенствования в области распознавания речи.

Современные системы распознавания могут быть классифицированы по различным признакам, от которых зависят применяемые подходы к решению задач распознавания. Далее рассмотрены основные типы признаков.

**Тип речи**

1. Слитная речь
2. Связные слова и фразы
3. Отдельные слова

В случае распознавания отдельных слов существует возможность определить начало и конец слова, что значительно сложнее сделать в других случаях.

**Зависимость от диктора**

1. Дикторо-зависимые
2. Дикторо-независимые

Различия в произношении между дикторами создают дополнительные сложности для решения задачи распознавания. В первом случае алгоритм сводится к обучению системы распознавания на основе голоса одного диктора, либо к настройке существующей дикторо-независимой системы на голос нового диктора. Создание системы, не зависимой от диктора, представляет большую сложность, так как она требует применения методов нормализации междикторских вариаций.

**Качество распознаваемой речи**

1. Чистая речь
2. Слабо зашумленная речь
3. Сильно зашумленная речь

Под чистой речью понимается сигнал, содержащий только речь без каких-либо помех. На практике не существует идеально чистых аудио-данных из-за различных помех, которые образуются в процессе захвата и передачи сигнала, а также во время произнесения самой речи.

**Способ разбиения речи на элементарные единицы**

1. По фонемам
2. По частям слов (слоги)
3. По словам.

Речь человека формируется путем произношения отдельных звуков, при объединении которых образуются осмысленные слова и предложения. При создании системы распознавания речи также можно выделить неделимые элементарные единицы, как фонемы, слоги, слова, или же задать их искусственно. Качество распознавания зависит неоднозначно от выбора тех или иных единиц, поэтому выбор определяется требованиями к конкретной системе.

**Размер словаря**

1. Малый словарь – порядка 100 слов
2. Средний словарь – порядка 1000 слов
3. Большой словарь – порядка 5000 и более.

Подходы к решению задачи распознавания зависят от словарно базы. Так, в системах с большим словарем зачастую используют фонемы или слоги для разбиения речи, а для малых размеров словаря может быть достаточно использование слова как элементарной единицы речи.

**Тип словаря**

1. Определенный (замкнутый) словарь
2. Неопределенный (неограниченный) словарь

Процесс распознавания может производиться либо на ограниченном пространстве, заданном наперед и не изменяющемся, либо на всем пространстве естественного языка, включая отсутствующие в словаре слова. Вторая задача представляет наибольшую сложность как в алгоритмическом, так и в вычислительном плане.

**Тип грамматики**

1. Фиксированная грамматика
2. Естественная грамматика

Задача распознавания речи связана также с грамматикой или структурой выражений, которые необходимо распознавать. Это могут быть как выражения с определенным синтаксисом, то есть с фиксированной грамматикой, так и с общей структурой, характерной естественному языку.

**Положение устройства захвата звука**

1. Близкое расположение (до 20 см)
2. Далекое расположение (более 20 см)

Важную роль в распознавании играет расстояние между диктором и устройством захвата речи (микрофоном, телефоном и др.), так как это расстояние влияет на степень присутствия постороннего шума или эхо.

## Подходы к решению задачи распознавания речи

Существуют несколько подходов, применяемых к решению задачи распознавания речи.

Первый подход, используемый для улучшения показателей распознавания речи, основывается на выделении векторов свойств из сигнала с учетом особенностей восприятия звука человеческим ухом. Он включает в себя анализ несущих частот и выравнивание сигнала по громкости. Наиболее распространенными технологиями, использующими такой подход, являются метод коэффициентов линейного предсказания (Perceptual Linear Prediction, PLP, Hermansky, 1990) и метод кепстральных коэффициентов тоновой частоты (Mel Frequency Cepstral Coefficients, MFCC, Davis & Mermelstein, 1980). Одновременное и опережающее сопоставление с шаблоном (маскирование) (Paliwal & Lilly, 1997), присущее человеческому восприятию, может быть смоделировано и использовано для выделения свойств, обеспечивающих большую устойчивость к шуму. Для этого был разработан метод варьирования размерностей кадров (Variable Frame Rate analysis, VFR, Zhu & Alwan, 2000). Учитывая специфику работы нервных клеток, отвечающих за слуховые рецепторы, был предложен метод диапазонной автокорреляции (Subband-Autocorrelation, SBCOR, Kajita & Itakura, 1994).

Другой подход основан на анализе звуковых сигналов. Различие поступающих в систему зашумленных сигналов от шаблонов, полученных в ходе обучения «чистыми» сигналами, является основной причиной неустойчивости работы систем распознавания. Целью подхода является уменьшение этого различия. Предполагается, что шум в звуковых сигналах аддитивный и стационарный. Оценки среднего значения усредненного шума вычитаются из кепстра (Cepstral Mean Subtraction, CMS, Furui, 1981) или спектра (Spectral Subtraction, SS, Virag, 1999), вычисленного по зашумленным данным. Некоторые модификации таких методов включают в себя нелинейное спектральное вычитание (Non-linear Spectral Subtraction, NSS, Lockwood & Boudy, 1992), которые используют спектральные огибающие. Такие техники требуют хорошей оценки шума, которую на практике бывает сложно получить, особенно в случае нестационарного фонового шума.

Еще одним способом борьбы с разницей между полученными свойствами из зашумленных и чистых сигналов является использование высокочастотного фильтра. Предполагается, что шум в сигнале не стационарный, а медленно изменяющийся во времени. Метод RASTA (Relative Spectral Analysis, Hermansky & Morgan, 1994) представлен таким образом, что относительные спектральные изменения фиксируются. И те медленные изменения, которые были вызваны шумом, удаляются. В этом случае отпадает необходимость в явном оценивании шума.

Третий подход основан на использовании многомерных пространств. “Основной идеей этого подхода является нахождение линейного отображения, которое минимизирует функцию стоимости. Часто в качестве такого отображения используется умножение вектора свойств на матрицу преобразования” (Ephraim & Trees, 1994). Примерами данного подхода могут служить независимый компонентный анализ (Independent Component Analysis, PCA) и основной компонентный анализ (Principal Component Analysis, ICA, Koscor, 2000), а также проектирование на многомерные подпространства (Gales, 2002).

## Модели, используемые в задаче распознавания речи

### Скрытые Марковские модели

Для разработки систем распознавания с большой словарной базой необходимо произвести обучение на представительных данных. В таких случаях зачастую используют статистические подходы машинного обучения, которые позволяют извлекать закономерности из неполной или неопределенной информации. В речи причинами неопределенности и неполноты могут являться вариации в звуках и произношениях, внешней среды и качестве каналов связи.

Одним из самых успешных и популярных методов статистического моделирования стали скрытые Марковские модели (Hidden Markov Models) благодаря их способности описывать как временные, так и спектральные речевые характеристики. Теоретические основы скрытых моделей Маркова были заложены в конце 1960-ых и начале 1970-ых годов в классических работах Л.Баума и его коллег, а практическое применение к решению задачи распознавания речи было осуществлено в 1970-ых годах Д. Бейкером из CMU и Ф.Йелинеком и его коллегами из IBM.

В современных системах широкое применение получили так называемые непрерывные скрытые Марковские модели, в которых функция распределения вероятностей наблюдений представляется в виде смеси нормальных распределений (Gaussian Mixture Model).

Непрерывная скрытая модель Маркова с N состояниями {1,2,…,N} и M смесями для каждого состояния определяется тройкой λ={A,B,π}, где:

A={aij}, где aij – вероятность перехода из i-го в j-е состояние, то есть

aij = P [qt+1 = j |qt = i], 1 ≤ i, j ≤ N

B – вероятность распределения получения вектора наблюдений, то есть

bj(X) = , 1 ≤ j ≤ N, где Х – вектор наблюдений, – вес к-той смеси для состояния j, и N – функция нормального распределения со средними значениями и матрицей ковариаций . На практике матрица ковариаций представляется в виде диагональной матрицы из соображений скорости вычислений.

π={ πi} – вероятность распределения начального состояния модели, то есть

πi= P [q1= i ], 1 ≤ j ≤ N.

Можно выделить три основные задачи, которые решаются в процессе применения скрытых Марковских моделей.

1. Пусть задана последовательность наблюдений О={о1,о2,…,оT} и модель λ={A,B,π}. Тогда требуется эффективно вычислить условную вероятность P(O| λ)появления данной последовательности наблюдения для данной модели. Данная задача является задачей оценки, которая позволит выбрать оптимальную модель среди нескольких конкурентных моделей. Для решения применяются процедуры прямого или обратного подсчетов [14].
2. Пусть задана последовательность наблюдений О={о1,о2,…,оT} и модель λ={A,B,π}, необходимо определить оптимальную последовательность состояний Q={q1, q2,…,qT}, наилучшим образом соответствующую данной последовательности наблюдения – задача декодирования, то есть нахождения последовательности состояний, которая смогла бы породить данную последовательность наблюдений. Критерий оптимальности может быть произвольным и выбирается в зависимости от решаемой задачи. Задача декодирования обычно решается с помощью алгоритма Витерби [41].
3. Необходимо найти алгоритм, позволяющий эффективно находить параметры модели λ={A,B,π} так, чтобы максимизировать условную вероятность P(O| λ). Данная задача – задача обучения модели, то есть нахождения оптимальных параметров модели для максимизации требуемой вероятности. Здесь применим итеративный метод Баума-Уэлча или метод Максимизации ожидания [6].

Кроме определения типа задачи необходимо выбрать архитектуру модели: количество состояний, связи между ними, а также наличие связных состояний и учет длительности состояний [1].

Преимуществом скрытых моделей Маркова является их адекватное моделирование временных характеристик речи. К недостаткам можно отнести сложность процесса распознавания, что не позволяет проанализировать природу ошибок для улучшения качества распознавания. Кроме того, использование смесей нормальных распределений влечет за собой неспособность эффективность описывать данные, лежащие вокруг некоторого многообразия. В частности, показано, что речь может быть смоделирована некоторой динамической системой с небольшим числом параметров, означая, что речевые характеристики лежат в гораздо меньшей размерности, чем той, которая обычно используется в смесях нормальных распределений [12].

### Искусственные нейронные сети

Искусственная нейронная сеть представляет собой математическую модель биологического нейрона. Впервые была смоделирована У. Маккалоком и У. Питтсом в 1943 году, далее была применена к решению задач машинного обучения и распознавания образов Д.Хеббом в 1949 году, В.Кларком в 1954 году и Ф.Розенблаттом в 1958 году. Детальный обзор и теоретическую основу искусственных нейронных сетей можно найти в [43].

На настоящий момент наибольшую популярность в распознавании речи получили многослойные нейронные сети (Deep Neural Networks), которые призваны успешно восполнить недостатки стандартных смесей нормального распределения, которые были указаны выше. Многослойные нейронные сети успешно применяются для акустического моделирования в Университете Торонто, IBM Research, Google, Microsoft Research. Ниже описаны основные понятия, касающиеся многослойных нейронных сетей и способы их обучения.

«Многослойная нейронная сеть – это однонаправленная искусственная нейронная сеть с одним или более уровнями (слоями) со скрытыми нейронами между входным и выходным слоями» [16]. На вход сети поступают акустические параметры фиксированной размерности, которые были извлечены из речевого сигнала. Каждый скрытый нейрон j использует логистическую функцию для отображения входного сигнала xj с предыдущего уровня в выходной сигнал yj для следующего уровня:

*yj =, xj= bj +* , где bj  - коэффициент отклонения нейрона j, i – индекс нейронов предыдущего уровня, – вес связи между i-ым нейроном предыдущего слоя и j-ым нейроном следующего слоя. При классификации, выходной нейрон j преобразует входной сигнал xj в вероятность класса pj как:

*pj =* , где k – индекс, пробегающий по всем классам.

Обучение искусственной нейронной сети представляет собой процесс подбора оптимального набора весов с помощью алгоритма обратного распространения ошибки между истинным результатом и фактическим значением, полученным на выходе нейронной сети [16].

Для многослойной нейронной сети с множеством уровней и нейронов на каждом уровне требуются очень большие вычислительные ресурсы, данные и время для обучения. С помощью методов градиентного спуска, применяемых для обучения, можно найти лишь локальные оптимумы, если начальные значения весов были заданы неверно, что приводит к переобучению сети. Чтобы избежать подобных проблем был предложен принципиально новый подход к обучению сети, называемый генеративным пред-обучением. Суть подхода состоит в отдельном обучении каждого слоя, при этом выходные данные предыдущего слоя после обучения являются входными данными следующего слоя для обучения. Веса, полученные таким образом, являются гораздо лучшими начальными условиями для проведения итогового обучения сети, при котором данные веса будут скорректированы [16].

Многослойные нейронные сети благодаря своей способности аппроксимизировать любую статическую нелинейную функцию могут полностью заменить смеси нормальных распределений, но до сих пор не найдены способы полной замены скрытых Марковских моделей, которые показали превосходство в моделировании временных зависимостей.

## Обзор библиотек для автоматического распознавания речи

За последние десятилетия было разработано огромное количество средств по распознаванию речи. Существуют различные коммерческие системы, такие как AT & T Watson, Microsoft Speech Server, Google Speech API. Однако частные системы не позволяют контролировать и управлять функциями распознавателя и обеспечивают ограниченную интегрируемость в другое программное обеспечение, что делает актуальным разработку программного обеспечения для автоматического распознавания речи с открытым исходным кодом.

В работе рассматриваются наиболее распространенные инструментарии, предназначенные для автоматического распознавания речи, которые находятся в свободном доступе.

1)HTK – Hidden Markov Model Toolkit – набор инструментов для создания и управления скрытыми Марковскими цепями, используется для исследований в области речевого распознавания. НТК является кросс-платформенным и состоит из набора библиотечных модулей и инструментов, доступных в виде исходного кода C. Эти инструменты предоставляют сложные средства для анализа речи, для обучения скрытых Марковских моделей, тестирования и анализа результатов. Программное обеспечение поддерживает скрытые Марковские модели с использованием как смеси нормального, так и дискретного распределения и могут быть использованы для создания сложных систем распознавания.

НТК был первоначально разработан в Лаборатории машинного обучения инженерного факультета Кембриджского университета (CUED), где инструментарий был использован для построения больших систем распознавания словаря речи CUED. В настоящее время код библиотеки является открытым и распространяется под лицензией HTK Specific Lisence. HTK используется по всему миру и занимает лидирующие позиции среди средств разработки ASR-систем.

К преимуществам данного инструментария можно отнести:

* Высокую степень распространенности среди разработчиков
* Бесплатность и открытость кода
* Наличие подробной документации и примеров
* Простоту использования.

2)Kaldi – набор инструментов для распознавания речи, реализованных на языке C++ и свободно распространяемых под лицензией Apache License v2.0. Разработка инструментария была начата в университете Джона Хопкинса (США) в 2009 году и продолжена в Технологическом университете Брно (Чехия).

Основное предназначение инструментария состоит в конструировании акустических моделей, основанных на смесях нормальных распределений (GMM), подпространственных смесях (SGMM) и многослойных нейронных сетях (DNN). Для построения акустических моделей важную роль играет определение подхода к генерации акустических признаков. В Kaldi реализовано огромное количество алгоритмов генерации признаков, начиная со стандартных мел-кепстральных коэффициентов (MFCC) и заканчивая линейным дискриминантным анализом (LDA), адаптацией к диктору (SAT) и максимизацией взаимной информации (MMI).

Kaldi обладаем значительными преимуществами, такими как:

* Интеграция на уровне кода с конечными автоматами (FST) – используется инструментарий OpenFst
* Поддержка линейной алгебры
* Расширяемый дизайн: все алгоритмы представлены в самой общей форме
* Готовые рецепты для построения систем распознавания речи
* Открытость и бесплатность кода
* Подробная документация
* Широкое признание разработчиками

3)CMU Sphinx – крупнейший инструментарий по распознаванию речи под Linux, созданный разработчиками университета Карнеги Меллон (США). В него входят такие библиотеки, как:

* Pocketsphinx — программа на языке С, принимающая на вход произвольные акустические модели, грамматики и словари, а также звуковой поток. На выходе получается распознанный текст.
* Sphinxbase — библиотека, необходимая для работы Pocketsphinx
* Sphinx4 — гибкая библиотека для распознавания, написанная на Java.
* Sphinxtrain — программа для обучения акустических моделей.

Sphinx – это дикторо-независимый распознаватель непрерывной речи, который использует Скрытую Марковскую модель и n-граммную статистическую языковую модель.

Sphinx2 – самый быстрый и ориентированный на производительность распознаватель речи, разработанный Xuedong Huang в университете Карнеги Меллон и выпущенный в Open Source на основе BSD-лицензии на SourceForge Kevin Lenzo в Linux World в 2000 году. Sphinx2 ориентирован на распознавание речи в режиме реального времени и идеально подходит под создания различных мобильных приложений.

Sphinx3 представлял собой полу-непрерывную акустическую модель распознавания речи. Sphinx3 принял распространенную продолжительную модель построенную на скрытых Марковских моделях и использовался первоначально для высокоточного распознавания речи, которое осуществлялось в режиме – пост-фактум. Последние разработки (алгоритмы и программное обеспечение) способствовали тому, что Sphinx3 мог распознавать в режиме близком к типу real-time, хотя приложение еще не подходило для качественного использования в качестве приложения. Sphinx3 после активного развития и воссоединения с SphinxTrain обеспечил доступ к многочисленным современным методикам и моделям, таким как LDA/MLLT, MLLR и VTLN, которые улучшили точность распознавания речи.

Sphinx4 полный и переписанный речевой движок Sphinx, главная цель которого обеспечить гибкий каркас для исследования в распознавании речи. Sphinx4 написан полностью на языке программирования Java.

К преимуществам Sphinx можно отнести:

* Возможность адаптации звуковой модели для конкретного человека
* Открытость и бесплатность кода
* Подробная документация
* Распространенность среди разработчиков.

Все описанные выше инструментарии обладают своими преимуществами и являются хорошими средствами для автоматического распознавания речи, однако Kaldi демонстрирует наилучшую производительность. Значения Word Error Rate для перечисленных инструментариев на таких корпусах, как Verbmobil 1 и Wall Street Journal 1 представлены в таблице 1 и демонстрируют большой отрыв Kaldi по производительности [11]. Более того, время, затрачиваемое на настройку, подготовку, запуск и оптимизацию наборов для этого инструмента является минимальным.

Выдающуюся производительность Kaldi в сравнении с другими распознавателями можно рассматривать как эволюцию в системах распознавания речи с открытым исходным кодом. Кроме того, представленные в библиотеке рецепты и сценарии позволяют непродвинутому пользователю в короткие сроки освоить инструментарий.

В наборе инструментов семейства Sphinx также присутствуют средства для обучения, не содержащие все методы Kaldi, что приводит к меньшей производительности, но обеспечивающие возможность быстро начать распознавание речи вскоре после установки инструмента.

|  |  |  |
| --- | --- | --- |
| Recognizer | VM1 | WSJ1 |
| HTK | 22,9 | 19,8 |
| Sphinx-4 | 26,9 | 22,7 |
| pocketsphinx v0.8 | 23,9 | 21,4 |
| Kaldi | 12,7 | 6,5 |

Таблица 1. Частота появления ошибок (%) для корпусов VerbMobil 1 и Wall Street Journal 1.

НТК является наиболее трудным для изучения инструментарием. Настройка системы очень затратна по времени. Кроме того, развитие новых техник, особенно за пределами предоставленных учебных примеров, требует гораздо больше знаний и усилий, чем для других систем. Инструментарий предназначен для экспертов в области рассматриваемых технологий, при этом, демонстрируемая производительность не уступает Sphinx.

# ГЛАВА 2. РЕАЛИЗАЦИЯ СИСТЕМЫ РАСПОЗНАВАНИЯ ГОЛОСОВЫХ КОМАНД

## 2.1. Установка и настройка Kaldi

Минимальным необходимым окружением для библиотеки является Unix-подобная система. Kaldi наилучшим образом протестирована на Debian и Red Hat Linux, но также поддерживается на любом Linux, Cygwin или Mac OsX.

В прошлом Kaldi была скомпилирована в Windows, однако в данной операционной системе функциональность библиотеки ограничена, так как не все скрипты можно будет запустить. В данной работе используется операционная система Linux Mint 17.3.

В первую очередь, следует установить следующие компоненты, необходимые для работы с Kaldi:

* GIT: необходим для скачивания библиотеки и другого требуемого ПО
* Wget: необходим для инсталляции некоторых компонент, не входящих в установочный пакет библиотеки
* Unix-утилиты: bash, perl, awk, grep, make.

Инструменты и библиотеки, входящие в пакет установки Kaldi:

* OpenFst: библиотека для работы с конечными автоматами (Finite State Tranducers). Конечные автоматы играют важную роль в распознавании и синтезе речи и используются для представления вероятностных моделей.
* IRSTLM: это инструмент для построения языковых моделей. Процесс сборки IRSTLM требует Automake, ACLOCAL и libtoolize (соответствующие пакеты Automake и Libtool).
* sph2pipe: инструмент для преобразования файлов формата SPH в другие форматы, такие как WAV. Необходим для сценариев, которые используют данные LDC.
* ATLAS (Automatically Tuned Linear Algebra Software): программный пакет линейной алгебры. Является решением для автоматической генерации и оптимизации ПО для численных вычислений. В настоящее время ATLAS поставляет оптимизированные версии полного набора ядер линейной алгебры, известного как Basic Linear Algebra Subroutines (BLAS), и подмножество операций линейной алгебры в библиотеке LAPACK.
* CLAPACK: библиотека для линейной алгебры. Используется в системах, не имеющих ATLAS.

Чтобы установить Kaldi, необходимо выполнить следующее:

git clone https://github.com/Kaldi-asr/Kaldi.git Kaldi --origin upstream

cd Kaldi

Корневая папка проекта содержит следующие директории:

1) tools. Данный каталог содержит все инструменты, требуемые для работы с Kaldi. Наиболее важным является инструмент по работе с конечными автоматами – OpenFST.

Файл INSTALL содержит инструкции по установке всех инструментов, для этого необходимо выполнить команду make в данной директории. При наличии нескольких процессоров возможно распараллелить сборку путем добавления опции –j к команде make: например, make –j4 при наличии 4-х процессоров. По умолчанию устанавливается Open FST версии 1.3.4. При желании установить более новую версию, необходимо модифицировать Makefile данной директории, указав в качестве значения флага OPENFST\_VERSION номер требуемой версии.

2) src. Содержит несколько файлов и большое количество директорий, некоторые из них оканчиваются на “bin” – исполняемые файлы, другие содержат внутренний код. Makefile содержит правило test: команда make test запускает тесты во всех поддиректориях.

Для сборки директории необходимо проверить зависимости командой ./configure и при необходимости установить недостающие пакеты. Затем выполнить make depend make. С помощью опции –j так же возможно распараллелить сборку для ускорения процесса.

3) egs. Данный раздел состоит из готовых примеров систем автоматического распознавания речи. Рассмотрим один из них и перейдем в подкаталог voxforge. Этот каталог содержит пример использования онлайн декодирования. Используются готовые акустические модели, которые были обучены на данных VoxForge – записанная речь, а также готовые акустические модели, предназначенные для использования в бесплатных системах распознавания речи с открытым исходным кодом. Поддерживается два режима тестирования: декодирование предварительно записанных звуковых фрагментов и живой речи, которую пользователь произносит в микрофон. Микрофонный вход осуществляется с использованием библиотеки PortAudio.

С помощью команды ./run.sh запускается первый вариант тестирования, чтобы использовать свой собственный голос необходимо добавить опцию test: ./run.sh --test

4) misc. Содержит источники и PDF для некоторых статей, связанных с Kaldi (papers/), логотип Kaldi (logo/), а также некоторые сценарии для преобразования моделей НТК в модели для Kaldi

5) windows. Директория для работы с Kaldi в операционной системе Windows.

## 2.2. Определение параметров онлайн-декодера

Под «онлайн декодированием» понимается декодирование, где речь произносится в режиме реального времени, и нет необходимости ждать, пока весь звук не захватывается перед запуском процесса декодирования.

В Kaldi под термином декодирование в общем случае понимается не отдельная декодирующая программа, а скорее внутренний декодирующий объект, обычно имеющий тип LatticeFasterDecoder. Данный объект получает декодирующий граф (конечный автомат) и декодируемый объект. Все декодеры поддерживают онлайн-декодирование, однако существует разница в вызове декодера для декодирования в режиме реального времени.

В Kaldi есть две версии онлайн-декодера: более старая версия (online/) и новый новая версия (online2/). В предыдущей версии «decoder» это некий декодер (например, LatticeFasterDecoder) и «decodable» - декодируемый объект подходящего типа. Вызов decoder.Decode(&decodable) блокируется пока ввод не будет закончен. В то время как в новом декодере вместо этого вызывается decoder.InitDecoding(), и каждый раз, когда поступают новые данные, вызывается decoder.AdvanceDecoding().

Основная сложность декодирования в реальном времени состоит в выделении векторов свойств из речевого сигнала. В заголовочном файле online-feature.h представлены классы, предоставляющие различные компоненты для извлечения векторов свойств, - потомки базового класса OnlineFeatureInterface. Данный интерфейс определяет, как объект передает вектора свойств вызову (OnlineFeatureInterface::GetFrame()) и как определяется, сколько фреймов выделено (OnlineFeatureInterface::NumFramesReady()), однако, способ получения векторов определяется в классах-потомках.

В Kaldi имеется плагин для GStreamer – мощного и гибкого фреймворка, позволяющего создавать приложения для обработки потоковых мультимедийных данных. Плагин представляет собой некий фильтр, который принимает на вход поток аудио и на выходе дает распознанные слова. Главным преимуществом плагина является то, что он обеспечивает возможность встроить функциональность Kaldi по распознаванию речи в любой язык программирования, который поддерживает GStreamer (Python, Ruby, Java и др.). В данной работе используется Python. Также плагин упрощает интеграцию онлайн декодера Kaldi в какое-либо приложение, так как обмен данными с декодером происходит в соответствии со стандартами фреймворка GStreamer.

Рассмотрим основные свойства элементов плагина. Параметр name – строковая переменная, отвечающая за имя объекта (по умолчанию, onlinegmmdecodefaster), флаги readable, writable обеспечивают разные режимы доступа. Silent – булевый параметр, определяет, отправляется ли входящий аудио-поток к декодеру, также доступны флаги readable, writable. По умолчанию установлено значение false. Model – имя файла в формате String, содержащего акустическую модель (по умолчанию, final.mdl). fst – также параметр типа String, отвечающий за имя файла, содержащего конечный автомат для распознавания (по умолчанию, HCLG.fst). min-cmn-window – минимальное значение для средней кепстральной нормализации, которое используется как начало декодирования, целое число в интервале -2147483648 – 2147483647 (по умолчанию, 100).

Программа конструируется в методе init\_gst(self), в котором происходит инициализация компонент речи. Файлы, содержащие модель, а также другие параметры декодирования передаются элементу onlinegmmdecodefaster через стандартный метод set\_property():

self.asr.set\_property(«fst», model\_dir + «HCLG.fst»)

self.asr.set\_property(«lda-mat», model\_dir + «matrix»)

self.asr.set\_property(«model», model\_dir + «model»)

self.asr.set\_property(«word-syms», model\_dir + «words.txt»)

self.asr.set\_property(«silence-phones», «1:2:3:4:5»)

self.asr.set\_property(«max-active», 4000)

self.asr.set\_property(«beam», 12.0)

self.asr.set\_property(«acoustic-scale», 0.0769).

Затем, в выражении self.asr.connect('hyp-word', self.\_on\_word) плагин вызывает метод графического интерфейса \_on\_word каждый раз, когда распознается новое слово. Код метода приведен ниже:

def \_on\_word(self, asr, word):

Gdk.threads\_enter()

if word == "<#s>":

self.textbuf.insert\_at\_cursor("\n")

else:

self.textbuf.insert\_at\_cursor(word)

self.textbuf.insert\_at\_cursor(« »)

Gdk.threads\_leave()

В данном методе происходит вставка распознанного слова в текстовый буфер, связанный с главным текстовым блоком графического интерфейса. Если же распознан разделительный символ, вставляется разделитель строки.

Начало и конец процесса распознавания контролируется свойством декодера silence, которое принимает значение true или false. При установке значения false декодер не распознает входящий аудио-поток, однако в буфер могут продолжать вставляться слова, распознанные в уже запущенном процессе распознавания.

## 2.3. Реализация программы

В общем виде функциональная схема работы программы выглядит следующим образом (Рис.1):

На Рис. 2 представлен интерфейс программы. Главное окно имеет заголовок Voice Command Recognizer. В верхней части окна расположены кнопка для выбора режима работы программы: Live – распознавание команд, произнесенных пользователем в микрофон, Expreriments – распознавание команд из пред-записанных аудио-файлов, а также кнопка для выбора алгоритма распознавания: пользователю предлагается выбрать между двумя вариантами: Gaussian Mixture Models (GMM) и Deep Neural Nets (DNN). Ниже расположено текстовое поле, в котором отображается распознанный текст. В нижней части окна программы находятся кнопки Start и Stop, включающие и выключающие режим произнесения команд соответственно.

D:\Downloads\Вышка\Дипломчик\Untitled Diagram 2.png

Рис. 1. Функциональная схема работы программы

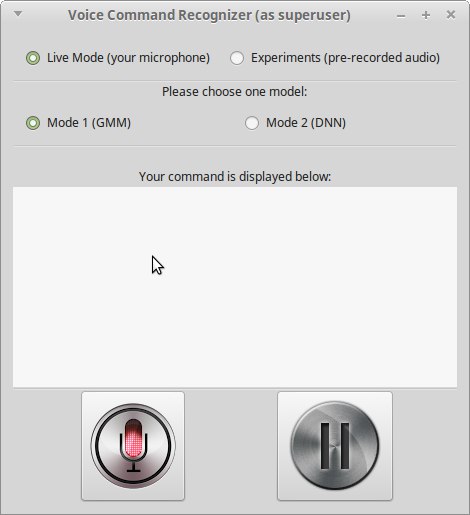


Рис. 2. Интерфейс программы по распознаванию голосовых команд

После выбора алгоритма необходимо нажать на кнопку с изображением микрофона. При этом если активен режим «Live», система начинает слушать вход микрофона, записывать команды в файлы и распознавать их из аудио-файлов. Когда команда произнесена, следует нажать на кнопку с надписью «Stop», чтобы избежать записи лишних слов, не относящихся к команде.

Если же активен режим «Experiments», то после нажатия кнопки с изображением микрофона происходит распознавание команд из пред-записанных файлов: команды появляются в текстовом поле, а кнопка прекращения записи при этом становится неактивной.

В случае если системе известна произнесенная команда, будет выполнено соответствующее команде действие. К примеру, команда Run Calculator запускает стандартный калькулятор, командой Run Browser открывается главное окно браузера по умолчанию (Mozilla Firefox), а Open Images – открывает средство по работе с изображениями. Рис. 3 – 5 демонстрируют результат работы программы для описанных выше примеров.

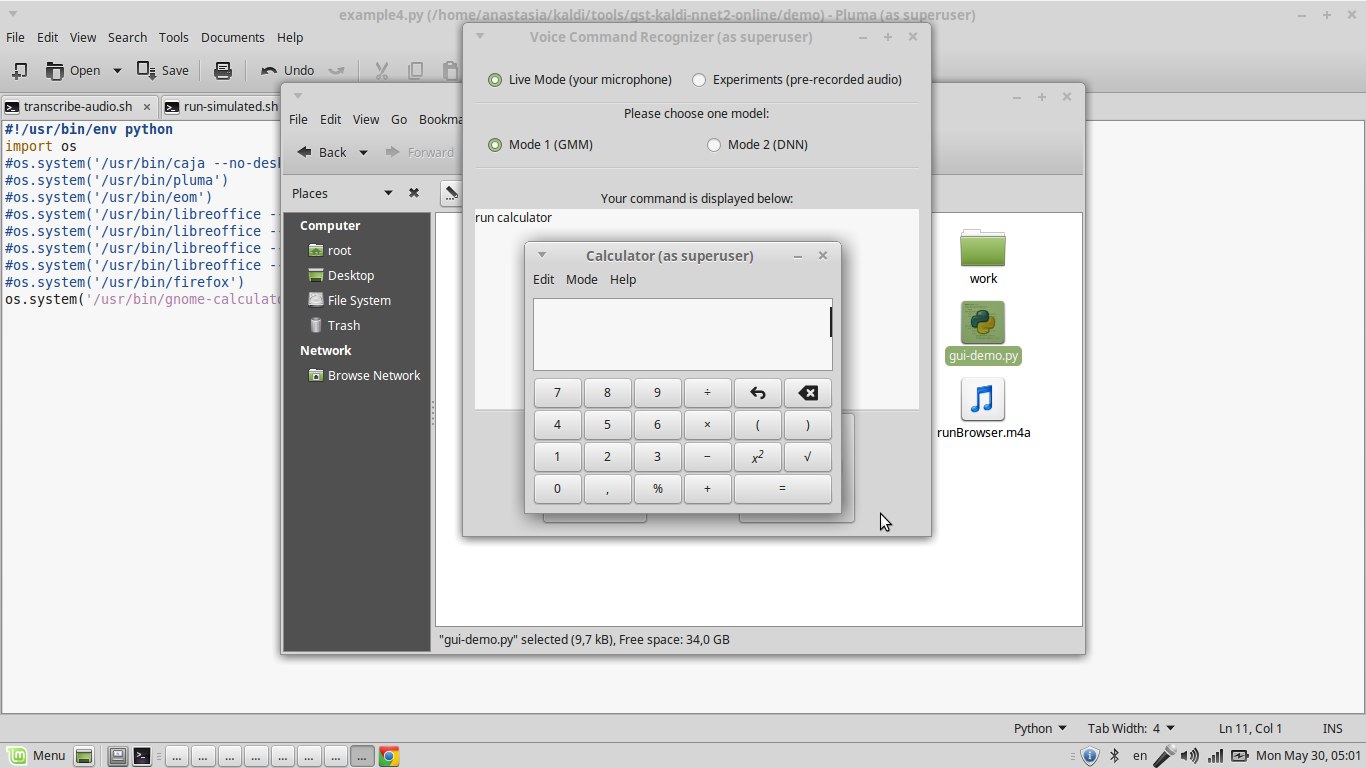


Рис. 3. Результат выполнения команды «run calculator»

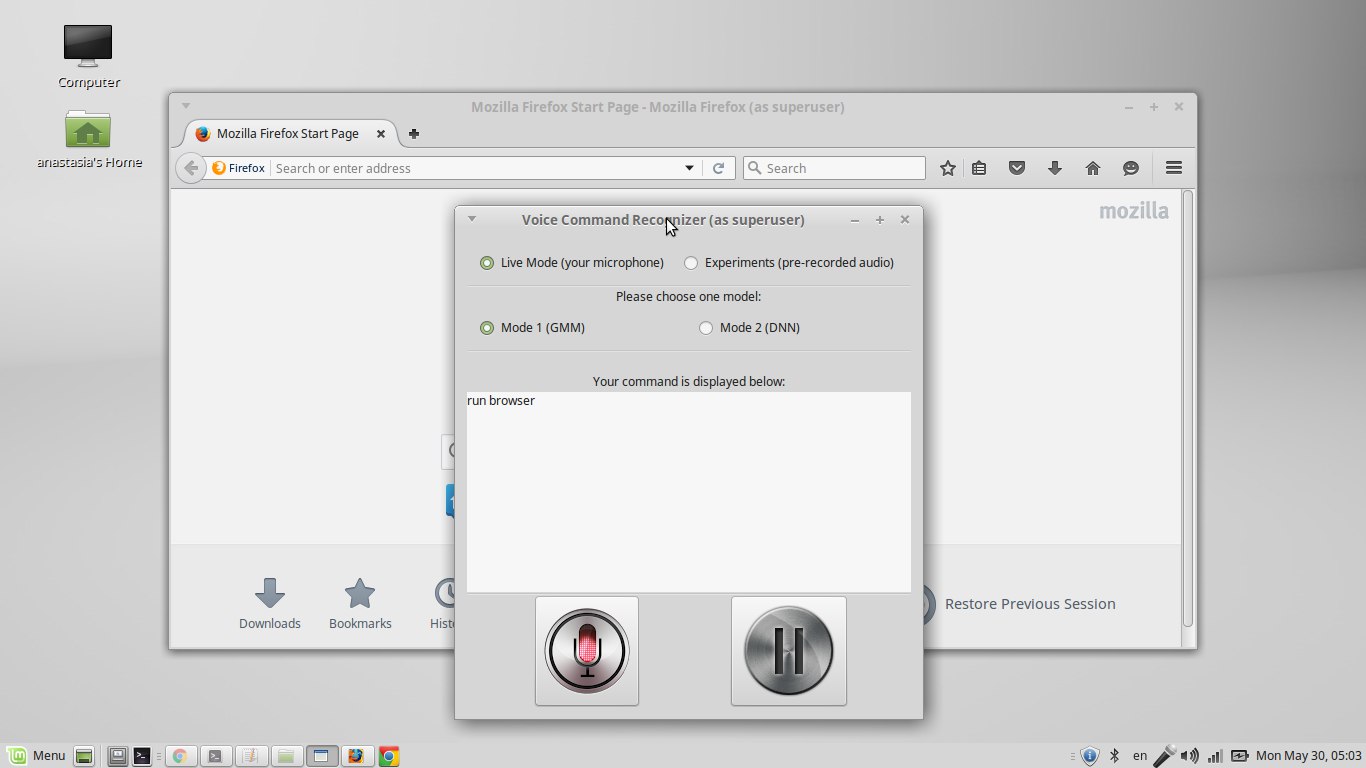


Рис. 4. Результат выполнения команды «run browser»

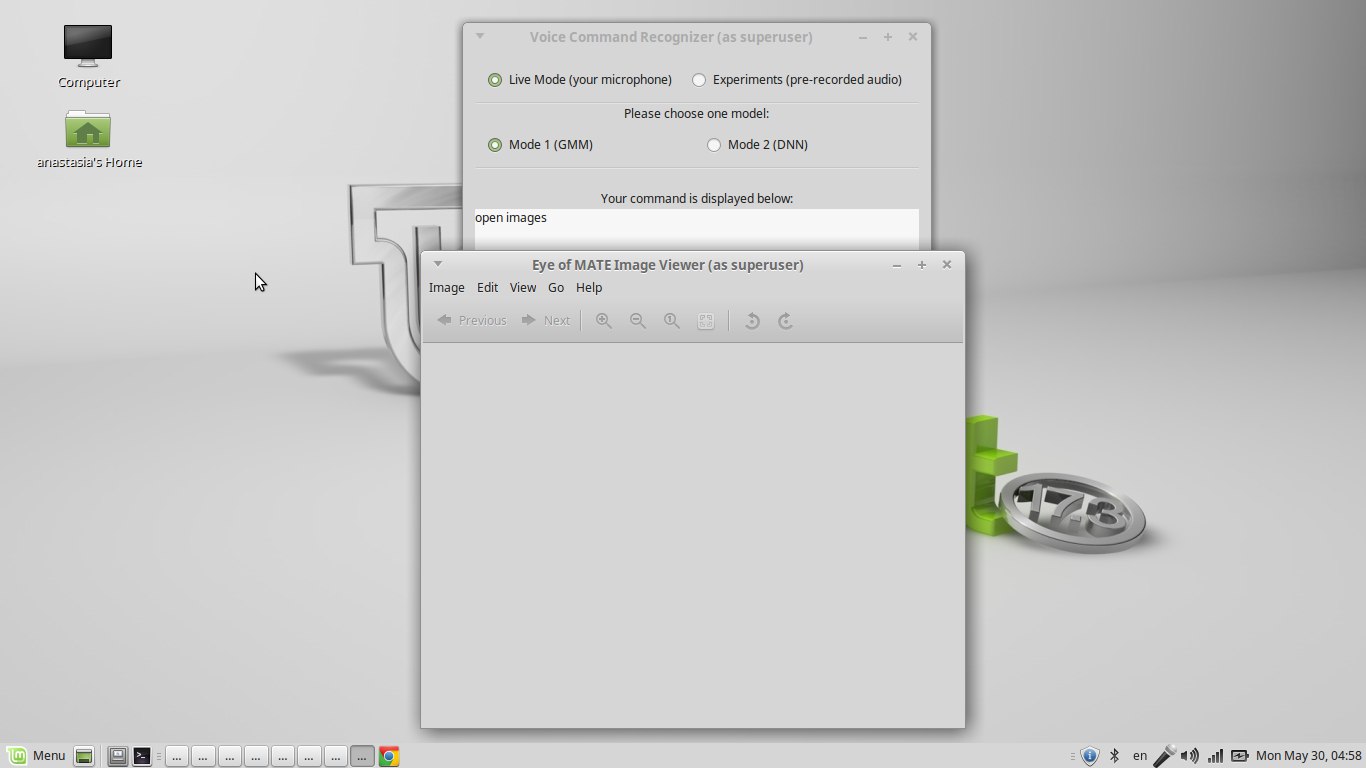


Рис. 5. Результат выполнения команды «open images»

Рассмотрим более подробно реализацию программы. В методе init\_gui() инициализируются компоненты графического интерфейса: создается окно с заголовком "Voice Command Recognizer", создаются вертикальный и 3 горизонтальных контейнера, а также инициализируются кнопки типа RadioButton и Button. Полный листинг кода программы представлен в Приложении 1.

Далее в методе init\_gst() происходит инициализация компонентов Gst фреймворка: в зависимости от выбранной пользователем модели выбирается объект типа kaldinnet2onlinedecoder или onlinegmmdecodefaster для инициализации декодера.

if self.button\_toggle=='DNN':

self.asr = Gst.ElementFactory.make("kaldinnet2onlinedecoder", "asr")

else:

self.asr = Gst.ElementFactory.make("onlinegmmdecodefaster", "asr")

self.fakesink = Gst.ElementFactory.make("fakesink", "fakesink")

Также через стандартный метод set\_property() устанавливаются параметры декодирования: файл, содержащий модель, конечный автомат, метод извлечения свойств и т.д.

Цифровая система обработки звукового сигнала предполагает представление аналогового речевого сигнала в цифровом виде. В результате аналого-цифрового преобразования (АЦП) непрерывный сигнал переводится в ряд дискретных временных отсчетов, каждый из которых представляет собой число. Это число характеризует сигнал в точке с определенной точностью. Процесс извлечения из сигнала численных значений называется квантованием. Процесс разбиения сигнала на отсчеты носит название дискретизации. Число отсчетов в секунду называется частотой дискретизации. Процесс обработки звуковой волны схематически описан на Рис.6.

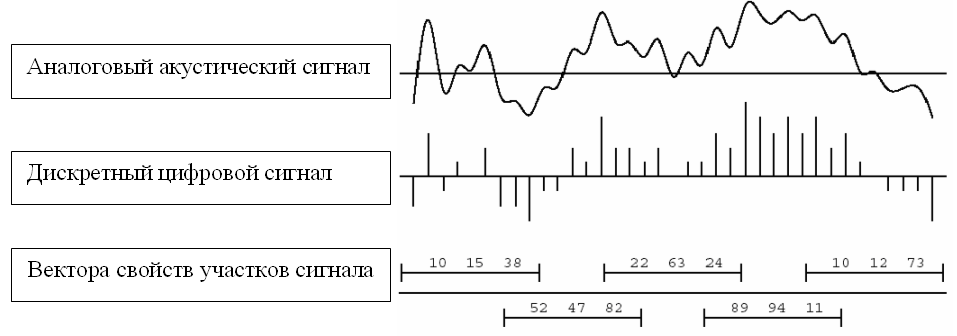


Рис.6. Этапы обработки речевого сигнала

Разработанная система распознавания голосовых команд является независимой от диктора. Звуковые сигналы значительно зависят от громкости, тембра голоса, состояния и настроения диктора. Для извлечения информации из таких сигналов используются фильтры тоновых частот (мел-скейл фильтры). Существует две основных технологии извлечения из сигнала вектора свойств, состоящего из кепстральных коэффициентов: на основе кепстральных коэффициентов тональной частоты (MFCC) и на основе кепстральных коеффициентов линейного предсказания (LPCC). Схематически эти методы описаны на Рис.7 и Рис.8.

Входной сигнал

Pre-emphasis

FFT

Возведение в квадрат

Логарифмирование сигнала

IDFT

Кепстральные коэффициенты

Мел-скейл фильтрация

Рис. 7. Описание метода MFCC Рис. 8. Описание метода LPCC

Входной сигнал

Pre-emphasis

FFT

Возведение в квадрат

IDFT

Рекурсия Дурбина

Кепстральная рекурсия

Кепстральные коэффициенты

Pre-emphasis – предварительное выделение фразы. Этот шаг вызван необходимостью спектрального сглаживания сигнала. Он становится менее восприимчивым к различным шумам, возникающим в процессе обработки.

К каждому сегменту применяется быстрое преобразование Фурье (FFT) для получения кратковременного спектра. После чего все значения возводятся в квадрат из-за особенностей последующих преобразований. На этом общие части двух методов заканчиваются. LPCC сперва вычисляет коэффициенты линейного предсказания (Linear predictive coefficients). Вместо сложной мел-скейл фильтрации и логарифмического сжатия сигнала, используемых в MFCC, LPCC использует более простой подход смещения спектра с помощью авторегрессионного фильтра LPC. Рекурсия Дурбина (Durbin recursion) получает коэффициенты LPC из коэффициентов автокорреляции. Далее кепстральные коэффициенты получаются рекурсивно из LPC коэффициентов.

В get\_command() текст в буфере сравнивается с известными системе командами и выполняются соответствующее командам действие, например:

if (self.textbuf=="run browser"):

os.system('/usr/bin/firefox').

Таким образом, программу можно представить как совокупность трех взаимосвязанных модулей, изображенных на Рис.8.

D:\Downloads\Вышка\Дипломчик\blocks.png

Рис. 8. Модули программы

На данный момент в системе отсутствует возможность добавления новых команд со стороны пользователя. Чтобы научить систему распознавать новую команду, необходимо внести изменения в код программы, а именно в методе get\_command() добавить сравнение текста из буфера с новой командой и обеспечить выполнение действия, соответствующего данной команде, стандартными методами языка Python.

# ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНОЕ СРАВНЕНИЕ РАЗЛИЧНЫХ АКУСТИЧЕСКИХ МОДЕЛЕЙ

## 3.1. Подготовка аудио-данных

Для тестирования системы был произведен сбор аудио-данных. Для озвучивания команд были привлечены дикторы, дифференцированные по половому и возрастному признакам. Общее количество дикторов, участвующих в записи, составило 6 человек, из которых 3 мужского пола и 3 женского.

Аудио-данные представляют собой записи голосов 6 спикеров, произносящих команды на английском языке. Каждый из 6 спикеров произносит 27 команд, таким образом, получатся 162 файла формата .wav.

Все аудио-файлы названы единым образом. Например, openMyDocuments.wav означает, что произносится команда: “open my documents.” Размещены файлы в папках, соответствующих определенному говорящему в определенную сессию записи. Таким образом, данные выглядят следующим образом:

-6 различных дикторов

-каждый говорящий произносит 26 команд на английском языке

-162 записей формата \*.wav размещены в 6 папках

-27 команд:

Run browser

Create new tab

Create new window

Create private tab

Create private window

Close tab

Close browser

Next tab

Previous tab

Show settings

Clear history

Open source code

Open development tools

Show history

Watch movie

Listen to music

Open My documents

Run Office Writer

Run Office Impress

Run Office Draw

Run Office Tables

Run Calculator

Run Notes

Run Terminal

Open Images.

## 3.2. Проведение экспериментов

Оценка качества работы системы, производилась посредством расчета показателя ошибки. Наиболее распространенной метрикой в распознавании речи признан показательWord Error Rate (WER):

,

где N – общее количество слов в тестовой выборке;

Insertations – количество вставок слов, которых не было в тестовой выборке;

Deleations – количество пропущенных слов тестовой выборки;

Substitutions – количество замен слов тестовой выборки другими словами.

Готовые обученные акустические модели были скачаны с сайта проекта: http://kaldi-asr.org/downloads/all/. Использовались модели, обученные на корпусе Librispeech – приблизительно 1000 часов 16кГц английской речи. Данные получены из чтения аудиокниг из проекта LibriVox, и были тщательно сегментированы и выровнены. Для экспериментального сравнения брались модели, обученные на 460 ч данных: tri4b, tri4b + LDA+MLLT+SAT, nnet6a. Данные модели доступны для скачивания по ссылкам:

http://kaldi-asr.org/downloads/build/6/trunk/egs/librispeech/s5/exp/tri4b/,

http://kaldi-asr.org/downloads/build/6/trunk/egs/librispeech/s5/exp/nnet6a\_clean\_460\_gpu/.

Для подсчета уровня ошибок необходимо для каждой команды создать файл с расширением .hyp, содержащий текст команды, при этом имя файла должно соответствовать команде. Например, для команды «run calculator» создаем файл runcalculator.hyp, текст файла выглядит следующим образом:

Run browser <#S>, для команды «run notes» – файл runnotes.hyp с текстом Run notes <#S>. Данные файлы являются так называемыми гипотезами, с которыми будет сравниваться распознанный текст.

В файле с именем trans.txt перечислены названия файлов (без расширения) и текст, содержащийся в каждом файле, полученный в результате распознавания:

runbrowser run browser

runnotes run notes

runterminal run terminal

# and so on…

С помощью следующей команды формируется файл ref.txt, который содержит названия файлов и текст в них, однако вместо слов присутствует их ID в соответствие со словарем words.txt (словарь был скачан вместе с моделями):

sym2int.pl -f 2- $ac\_model/words.txt < $audio1/trans.txt > $decode\_dir/ref.txt.

Файл ref.txt выглядит следующим образом:

runbrowser 96735 14565

runnotes 96735 79072

runterminal 96735 111724

# and so on…

Далее формируется общий файл с гипотезами: слова из всех файлов расширения .hyp конвертируются в ID в соответствии со словарем в результате выполнения следующей команды:

for f in $decode\_dir/\*.hyp; do

(echo -n `basename $f .hyp`" " ; cat $f) | sed 's/<#s>//g' | sym2int.pl -f 2- $ac\_model/words.txt;

done > $decode\_dir/hyp.txt.

Текст файла hyp.txt представлен ниже:

runbrowser 96735 14565

runnotes 96735 79072

runterminal 96735 111724

# and so on…

Наконец, происходит расчет уровня ошибок распознавания с помощью метода compute-wer:

compute-wer --mode=present ark,t:$decode\_dir/ref.txt ark,t:$decode\_dir/hyp.txt.

Полный код скрипта для расчета показателя WER представлен в Приложении 2.

После запуска программы в режиме распознавания пред-записанных аудио-файлов с разными параметрами были получены следующие значения:

Word Error Rate (на зашумленных данных: SNR[[1]](#footnote-1)<20дБ):

Model tri4b:

%WER 25.69 [ 122 / 471, 45 ins, 54 del, 22 sub ]

Model tri4b + LDA+MLLT+SAT:

%WER 15,92 [75/ 471, 41 ins, 18 del, 16 sub ]

Model nnet6a:

%WER 8,9 [42/ 471, 13 ins, 25 del, 4 sub ]

Word Error Rate (на «чистых» данных: SNR>40дБ):

Model tri4b:

%WER 19, 53 [ 92 / 471, 31 ins, 44 del, 17 sub ]

Model tri4b + LDA+MLLT+SAT:

%WER 14,86 [70/ 471, 33 ins, 17 del, 20 sub ]

Model nnet6a:

%WER 5,09 [24/ 471, 9 ins, 12 del, 3 sub ]

Более наглядно полученные результаты представлены в Таблице 2.

|  |  |  |
| --- | --- | --- |
| Модель | Word Error Rate (%) | Среднее время распознавания одной команды (сек) |
| Зашумленные данные | | |
| tri4b | 25,69 | 4,37 |
| tri4b + LDA+MLLT+SAT | 15,92 | 3,24 |
| nnet6a | 8,90 | 0,89 |
| «Чистые» данные | | |
| tri4b | 19,53 | 3,11 |
| tri4b + LDA+MLLT+SAT | 14,86 | 1,01 |
| nnet6a | 5,09 | 0,44 |

Таблица 2. Результаты экспериментов

Таким образом, можно сделать вывод, что наилучшей моделью является nnet6a, так как она демонстрирует наилучшие результаты по таким показателям, как точность распознавания и среднее время распознавания одной команды.

# ЗАКЛЮЧЕНИЕ

Голосовые интерфейсы управления различными устройствами приобретают все большую популярность в наше время как в коммерческих, так и в академических отраслях. Существует огромное количество программных продуктов, обладающих данной функциональностью. Однако значительным недостатком большинства существующих решений является использование внешних сервисов, таких как Google, Yandex и др., что создает необходимость постоянного подключения к интернету. Другие системы, распознающие речь в режиме оффлайн, обладают низкой производительностью и используют устаревшее программное обеспечение. Таким образом, возникает необходимость в автономной системе распознавания речи, обладающей высоким качеством распознавания, но в то же время, бесплатной и с открытым исходным кодом.

В ходе работы все поставленные задачи были выполнены, а именно:

• Изучена литература по предметной области

• Проведен сравнительный анализ существующих инструментариев для распознавания речи

• Исследованы различные алгоритмы обучения акустических моделей

• Разработаны прототипы системы голосового управления на основе выбранного инструментария

• Проведено экспериментальное сравнение различных акустических моделей в рамках выбранного инструментария.

Кроме того в работе была приведена классификация существующих систем распознавания речи по таким критериям, как Тип речи, Зависимость от диктора, Качество распознаваемой речи, Способ разбиения речи на элементарные единицы, Размер и тип словаря, Тип грамматики и Положение устройства захвата звука,

В результате проведения сравнительного анализа существующих инструментариев по распознавания было выявлено, что Kaldi обладает наилучшей производительностью среди аналогов. Значения Word Error Rate демонстрируют большой отрыв Kaldi по производительности. Более того, время, затрачиваемое на настройку, подготовку, запуск и оптимизацию наборов для этого инструмента является минимальным.

Выдающуюся производительность Kaldi в сравнении с другими распознавателями можно рассматривать как эволюцию в системах распознавания речи с открытым исходным кодом. Кроме того, представленные в библиотеке рецепты и сценарии позволяют непродвинутому пользователю в короткие сроки освоить инструментарий.

Также были изучены такие алгоритмы обучения акустических моделей, как Скрытые Марковские модели и Многослойные нейронные сети.

На основе разработанной системы проводился экспериментальный анализ различных алгоритмов обучения акустических моделей. В ходе экспериментов было установлено, что многослойные нейронные сети демонстрируют наибольшую производительность, а именно наименьший уровень ошибок и наилучшее время распознавания.

В рамках работы была реализована автономная система распознавания голосовых команд на основе библиотеки Kaldi, предназначенная для управления компьютером и распознающая произнесенные пользователем команды на английском языке из списка доступных команд и выполняющая соответствующее команде действие. Система разработана под операционную систему Linux.

Программа является бесплатным автономным решением по распознаванию речи, код программы был выложен на GitHub и доступен для скачивания по ссылке: https://github.com/AVerkhovtseva/voice-command-recognizer.

В перспективе планируется развить работу по следующим направлениям:

* Добавление новых параметризованных команд управления функциями операционных систем, таких как установка будильника или таймера на конкретное время, добавление заметок и др.
* Обеспечение поддержки распознавания русского языка на основе модели Voxforge (модель доступна для скачивания по ссылке: https://github.com/freerussianasr/recipes/tree/master/voxforge\_ru)
* Разработка системы, способной адаптироваться к изменяющимся параметрам внешней среды и самостоятельно подбирать наиболее подходящие параметры акустической модели.
* Интеграция системы с решением, предложенным Савченко А.В. и Савченко Л.В. в статье Towards the creation of reliable voice control system based on a fuzzy approach [35], что позволит добиться еще лучшего качества распознавания.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Abdulla, W. H., & Kasabov, N., The concepts of hidden Markov model in speech recognition [Электронный ресурс] /Information Science Discussion Papers Series No. 99/09/ University of Otago. –ежим 1999. –ежим Режим доступа: http://hdl.handle.net/10523/997
2. Adorf, J., Web Speech API: Technical report/ KTH Royal Institute of Technology, Stockholm, Sweden. – 2013
3. Attwater D., Edgington M., Durston P., Whittaker S., Practical issues in the application of speech technology to network and customer service applications/ Speech Commun.–ежим 31. –ежим 2000. –ежим p.279-291
4. Baden-Wuerttemberg Cooperative State University Stuttgart, Germany, OASIS—Open-Source Automatic Speech Recognition In Smart Devices [Электронный ресурс] . –ежим 2014. –ежим Режим доступа: http: //www.dhbw-stuttgart.de/themen/kooperative-forschung/fakultaet- technik/oasis.html
5. Baker J.K., The Dragon system – An Overview/IEEE Trans. Inf. & Syst. –ежим 2010. –ежим №3. –ежим p. 366-375, doi: 10.1509/MMSP.2002.1267294
6. Baum, Leonard E.; Petrie, Ted; Soules, George; Weiss, Norman, A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains [Электронный ресурс]/ Ann. Math. Statist . –ежим 1970. –ежим Vol. 41. –ежим No. 1. –ежим p. 164 –ежим 171. doi:10.1214/aoms/1177697196 .– Режим доступа: http://projecteuclid.org/ euclid.aoms/1177697196
7. Bellegarda J.R., Spoken Language Understanding for natural interaction: The siri experience// J. Mariani, S. Rosset, M. Garnier-Rizet, L. Devillers (Eds.)/ Natural Interaction with Robots, Knowbots and Smartphones, Springer, New York. –2014. – p. 3-14
8. Das S.K., Picheny M.A., Issues in practical large vocabulary isolated word recognition: The IBM Tangora system// C.-H. Lee, F.K. Soong, K.K. Paliwal (Eds.)/The Kluwer International Series in Engineering and Computer Science, Springer, Santa Clara. – 1996. – p. 457-479
9. Dunn, M., Pro Microsoft Speech Server 2007: Developing Speech Enabled Applications with .NET (Pro)/Apress, Berkeley, USA .– 2007
10. Furui S., Recent progress in corpus-based spontaneous speech recognition/ IEICE Trans. Inf. & Syst. – 2005. – E88-D. – №3. – p.366-375, doi: 10.1109/MMSP.2002.1203294
11. Gaida С., Lange P., Petrick R., Proba P., Malatawy A., Suendermann-Oeft D., Comparing Open-Source Speech Recognition Toolkits [Электронный ресурс] /DHBW Stuttgart Technical Report. – 2005. – Режим доступа: http://suendermann. com/su/pdf/oasis2014. pdf
12. Gales M. and S. Young, The Application of Hidden Markov Models in Speech Recognition/Foundations and Trends in Signal Processing. – 2007. – №1(3). – p.195-304
13. Goffin, V., Allauzen, C., Bocchieri, E., Hakkani-Tiir, D., Ljolje, A., Parthasarathy, S., Rahim, M., Riccardi, G., Saraclar, M., The AT&T WATSON speech recognizer/ Proc. of the ICASSP, Philadelphia, USA . – 2005. – p.1033-1036
14. Gillick L., Cox S.J., Some statistical issues in the comparison of speech recognition algorithms/ IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'89) Glasgow. –1989. –p. 532-535
15. Hemdal, J.F. and Hughes, G.W./A feature based computer recognition program for the modeling of vowel perception// Models for the Perception of Speech and Visual Form/Wathen-Dunn, W. Ed. MIT Press, Cambridge, MA. – 2005
16. Hinton G., Dahl G., Deng.L, You.D, Jaitly N., Senior A., Sainath T., Deep Neural Networks for Acoustic Modeling in Speech Recognition/IEEE Signal Processing Magazine. – 2012 . – № 6. – p.82-97. doi: 10.1109/MSP.2012 .2205597
17. Huang J.-T., Li J., Yu D., Deng L., Gong Y., Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers/ Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, Vancouver. – 2013. – p. 7304-7308.
18. Huggins-Daines, D., Kumar, M., Chan, A., Black, A., Ravishankar, M., Rudnicky, A.: Pocketsphinx, A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices/ ICASSP, Toulouse, France . – 2006
19. Itakura,F., Minimum prediction residual principle applied to speech recognition/ IEEE Trans.Acoustic, Speech and Signal Processing . –ASSP-23. – p. 57-72.
20. Jelinek F., Continuous Speech Recognition by Statistical Methods/ IEEE Proceedings . – 1976 . – №64. –p. 532-556. –doi: 10.1109/PROC. 1976.101 59
21. Hershey J.R., Rennie S.J., Olsen R.A., Kristjansson T.T., Super-human multi-talker speech recognition: A graphical modeling approach/ Computer Speech and Language. –2010 . – №24. – p. 45–66
22. Kaldi ASR Documentation [Электронный ресурс]/ Режим доступа: http://Kaldi-asr.org/doc/
23. Kubala F, Austin S, Barry C, Makhoul J, Placeway P, Schwartz R., Byblos. Speech Recognition Benchmark Results/BBN Systems and Technologies, Cambridge . – 2010. – p.72 - 77
24. Lamere, P., Kwok, P., Gouvea, E., Raj, B., Singh, R., Walker, W., Warmuth, M., Wolf, P., The CMU SPHINX-4 Speech Recognition System/Proc. of the ICASSP'03, Hong Kong, China. – 2003
25. Lee, A., Kawahara, T., Recent development of open-source speech recognition engine Julius/ Proc. of the APSIPA ASC, Los Angeles, USA. –2009
26. Lee, K.F., Reddy, R., Automatic Speech Recognition: The Development of the Sphinx Recognition System/ Kluwer Academic Publishers, Norwell, MA, USA. –1988
27. Lujn-Mares, M., Tamarit, V., Alabau, V., Martnez-Hinarejos, C.D., i Gadea, M.P., Sanchis, A., Toselli, A.H.: Iatros, A speech and handwritting recognition system/ V Jornadas en Tecnologas del Habla (VJTH'2008) . –2008. –p.75-78
28. Mengjie, Z., Overview of speech recognition and related machine learning techniques [Электронный ресурс]/ Technical report. – 2004. – Режим доступа: http://www.mcs.vuw.ac.nz/comp /Publications/archive/CS-TR-01/CS-TR-01-15.pdf
29. Moore, D., Dines, J., Magimai-Doss, M., Vepa, J., Cheng, O., Hain, T.: Juicer, A weighted finite-state transducer speech decoder/ 3rd Joint Workshop on Mul- timodal Interaction and Related Machine LEarning Algorithms MLMI'06. . – 2006 . – IDIAP-RR. – p.06-21
30. Murveit, H., Cohen, M., Price, P., Baldwin, G., Weintraub, M., Bernstein, J., SRI's DECIPHER system/ Proc. of the Workshop on Speech and Natural Language. HLT '89, Stroudsburg, PA, USA// Association for Computational Linguistics. – 1989. – p.238-242
31. Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hanne- mann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., Vesely, K., The Kaldi speech recognition toolkit// Proc. of the ASRU, Hawaii, USA. – 2011
32. Rabiner, L. Juang, B. H., Yegnanarayana, B., Fundamentals of Speech Recognition/ *Pearson Publishers*. – 2010
33. Rabiner, L. R., A tutorial on Hidden Markov Models and selected applications in speech recognition/ Proceedings of IEEE. – 1989 . – p.257-286.
34. Rybach, D., Hahn, S., Lehnen, P., Nolden, D., Sundermeyer, M., Tuske, Z., Wiesler, S., Schluter, R., Ney, H., RASR—The RWTH Aachen University open source speech recognition toolkit/ Proc. of the ASRU, Hawaii, USA. –2011
35. Savchenko A., Savchenko L., Towards the creation of reliable voice control system based on a fuzzy approach/ Pattern Recognition Letters . – №65. –p.145-151
36. Schalkwyk J., Beeferman D., Beaufays F., Byrne B., Chelba C., Cohen M., Kamvar M., Strope B., Your Word is my Command: Google search by voice/ A case study, in: A. Neustein (Ed.)//Advances in Speech Recognition, Springer, New York. –2010. – p. 61-90
37. Swami S., Ramakrishnan K.V., An efficient speech recognition system/ Computer Science and Engineering: An international Journal. – №4. – p.21-27. – doi: 10.5121/cseij.2013.340321
38. Wang X., Paliwal K.K., Feature extraction and dimensionality reduction algorithms and their applications in vowel recognition/Pattern Recogn. – 2003. –№36. – p. 2429-2439
39. Xiaodong Cui, A Study of Variable-Parameter Gaussian Mixture Hidden Markov Modeling for Noisy Speech Recognition/ IEEE Transactions On Audio, Speech, And Language Processing. –2007. –№4. –doi: 10.1109/ TASL.2006.889791
40. Young, S., Evermann, G., Gales, M., Kershaw, D., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., Woodland, P., The HTK Book, Version 3.4/Cambridge University, Cambridge, UK. –2006
41. Zen.M., Acoustic Modeling in statistical parametric speech synthesis. Acoustics/ Speech and Signal Processing (ICASSP)// 2014 IEEE International Conference on acousticsspeech and signal processing. –2014 – p.3844 – 3848. –doi: 10.1109/ICASSP.2014.6854321
42. Zweig, G., Nguyen, P.: SCARF, A segmental conditional random field toolkit for speech recognition/ Proc. of the Interspeech, Makuhari, Japan. –2010
43. Хайкин С., Нейронные сети: полный курс/ 2-е издание. Вильямс. – 2010 . –1104 с.

# ПРИЛОЖЕНИЯ

## Приложение 1. Листинг программы

#!/usr/bin/env python

import sys

import os

import gi

import pygtk

import pyaudio

import wave

gi.require\_version('Gst', '1.0')

from gi.repository import GObject, Gst, Gtk, Gdk

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

GObject.threads\_init()

Gdk.threads\_init()

Gst.init(None)

class DemoApp(object):

"""Voice Command Recognition System"""

def \_\_init\_\_(self):

"""Initialize a DemoApp object"""

self.init\_gui()

self.init\_gst()

def callback(self, widget, data=None):

print "%s was toggled %s" % (data, ("OFF", "ON")[widget.get\_active])

def button\_toggle(self, widget):

if widget.get\_active():return Gtk.RadioButton.get\_name(widget)

def init\_gui(self):

"""Initialize the GUI components"""

self.window = Gtk.Window()

self.window.set\_title("Voice Command Recognizer")

self.window.set\_position(Gtk.WindowPosition.CENTER)

self.window.connect("destroy", self.quit)

self.window.set\_default\_size(400,200)

self.window.set\_border\_width(10)

vbox = Gtk.VBox()

box1 = Gtk.HBox(False, 10)

box1.set\_border\_width(10)

vbox.pack\_start(box1, True, True, 0)

box1.show()

self.button\_live = Gtk.RadioButton.new\_with\_label\_from\_widget(None, label="Live Mode (your microphone)")

self.button\_live.connect("toggled", self.callback, "live")

self.button\_live.set\_active(True)

box1.pack\_start(self.button\_live, True, True, 0)

self.button\_live.show()

self.button\_pre = Gtk.RadioButton.new\_with\_label\_from\_widget(self.button\_live, label="Experiments (pre-recorded audio)")

self.button\_pre.connect("toggled", self.callback, "pre")

box1.pack\_start(self.button\_pre, True, True, 5)

self.button\_pre.show()

separator = Gtk.HSeparator()

vbox.pack\_start(separator, False, False, 0)

separator.show()

self.label = Gtk.Label("Please choose one model:")

vbox.pack\_start(self.label, True, True, 1)

box2 = Gtk.HBox(False, 10)

box2.set\_border\_width(10)

vbox.pack\_start(box2, True, True, 0)

box2.show()

self.button\_gmm = Gtk.RadioButton.new\_with\_label\_from\_widget(None, label="Mode 1 (GMM)")

self.button\_gmm.connect("toggled", self.callback, "GMM")

self.button\_gmm.set\_active(True)

box2.pack\_start(self.button\_gmm, True, True, 0)

self.button\_gmm.show()

self.button\_dnn = Gtk.RadioButton.new\_with\_label\_from\_widget(self.button\_gmm, label="Mode 2 (DNN)")

self.button\_dnn.connect("toggled", self.callback, "DNN")

box2.pack\_start(self.button\_dnn, True, True, 0)

self.button\_dnn.show()

separator = Gtk.HSeparator()

vbox.pack\_start(separator, False, False, 0)

separator.show()

box2 = Gtk.VBox(False, 10)

box2.set\_border\_width(10)

vbox.pack\_start(box2, True, True, 0)

box2.show()

self.label = Gtk.Label("Your command is displayed below:")

vbox.pack\_start(self.label, True, True, 1)

self.text = Gtk.TextView()

self.text.set\_size\_request(-1, 200)

self.textbuf = self.text.get\_buffer()

self.text.set\_wrap\_mode(Gtk.WrapMode.WORD)

vbox.pack\_start(self.text, True, True, 1)

separator = Gtk.HSeparator()

vbox.pack\_start(separator, False, False, 0)

separator.show()

hbox = Gtk.HBox(spacing=10)

hbox.show()

vbox.pack\_start(hbox, True, False, 0)

self.image = Gtk.Image()

self.image.set\_from\_file("/home/anastasia/Downloads/speak.jpg")

self.image.show()

self.button1 = Gtk.Button()

self.button1.add(self.image)

self.button1.set\_size\_request(100,100)

self.button1.show()

self.button1.connect('clicked', self.button1\_clicked,"1")

hbox.pack\_start(self.button1, False, False, 67)

self.image = Gtk.Image()

self.image.set\_from\_file("/home/anastasia/Downloads/stop.png")

self.image.show()

self.button2 = Gtk.Button()

self.button2.add(self.image)

self.button2.set\_size\_request(100,100)

self.button2.show()

self.button2.connect('clicked', self.button2\_clicked,"2")

hbox.pack\_start(self.button2, False, False, 13)

vbox.pack\_start(hbox, False, False, 5)

#if self.button\_toggle=='pre'

# button2.set\_enabled(False)

self.window.add(vbox)

self.window.show\_all()

def record\_audio(self):

CHUNK = 1024

FORMAT = pyaudio.paInt16

CHANNELS = 2

RATE = 44100

RECORD\_SECONDS = 4

WAVE\_OUTPUT\_FILENAME = "output.wav"

p = pyaudio.PyAudio()

stream = p.open(format=FORMAT,

channels=CHANNELS,

rate=RATE,

input=True,

frames\_per\_buffer=CHUNK)

frames = []

for i in range(0, int(RATE / CHUNK \* RECORD\_SECONDS)):

data = stream.read(CHUNK)

frames.append(data)

stream.stop\_stream()

stream.close()

p.terminate()

wf = wave.open(WAVE\_OUTPUT\_FILENAME, 'wb')

wf.setnchannels(CHANNELS)

wf.setsampwidth(p.get\_sample\_size(FORMAT))

wf.setframerate(RATE)

wf.writeframes(b''.join(frames))

wf.close()

def quit(self, window):

Gtk.main\_quit()

def init\_gst(self):

#Initialize the speech components

self.pulsesrc = Gst.ElementFactory.make("pulsesrc", "pulsesrc")

if self.pulsesrc == None:

print >> sys.stderr, "Error loading pulsesrc GST plugin. You probably need the gstreamer1.0-pulseaudio package"

sys.exit()

self.audioconvert = Gst.ElementFactory.make("audioconvert", "audioconvert")

self.audioresample = Gst.ElementFactory.make("audioresample", "audioresample")

self.asr = Gst.ElementFactory.make("kaldinnet2onlinedecoder", "asr")

self.fakesink = Gst.ElementFactory.make("fakesink", "fakesink")

if self.asr:

model\_file = "final.mdl"

if not os.path.isfile(model\_file):

print >> sys.stderr, "Models not downloaded? Run prepare-models.sh first!"

sys.exit(1)

self.asr.set\_property("fst", "HCLG.fst")

self.asr.set\_property("model", "final.mdl")

self.asr.set\_property("word-syms", "words.txt")

self.asr.set\_property("feature-type", "mfcc")

self.asr.set\_property("mfcc-config", "conf/mfcc.conf")

self.asr.set\_property("ivector-extraction-config", "conf/ivector\_extractor.fixed.conf")

self.asr.set\_property("max-active", 7000)

self.asr.set\_property("beam", 10.0)

self.asr.set\_property("lattice-beam", 6.0)

self.asr.set\_property("do-endpointing", True)

self.asr.set\_property("endpoint-silence-phones", "1:2:3:4:5:6:7:8:9:10")

self.asr.set\_property("use-threaded-decoder", False)

self.asr.set\_property("chunk-length-in-secs", 0.1)

else:

print >> sys.stderr, "Couldn't create the kaldinnet2onlinedecoder element. "

if os.environ.has\_key("GST\_PLUGIN\_PATH"):

print >> sys.stderr, "Have you compiled the Kaldi GStreamer plugin?"

else:

print >> sys.stderr, "You probably need to set the GST\_PLUGIN\_PATH envoronment variable"

print >> sys.stderr, "Try running: GST\_PLUGIN\_PATH=../src %s" % sys.argv[0]

sys.exit();

# initially silence the decoder

self.asr.set\_property("silent", True)

self.pipeline = Gst.Pipeline()

for element in [self.pulsesrc, self.audioconvert, self.audioresample, self.asr, self.fakesink]:

self.pipeline.add(element)

self.pulsesrc.link(self.audioconvert)

self.audioconvert.link(self.audioresample)

self.audioresample.link(self.asr)

self.asr.link(self.fakesink)

self.asr.connect('partial-result', self.\_on\_partial\_result)

self.asr.connect('final-result', self.\_on\_final\_result)

self.pipeline.set\_state(Gst.State.PLAYING)

def \_on\_partial\_result(self, asr, hyp):

#Delete any previous selection, insert text and select it

Gdk.threads\_enter()

self.textbuf.begin\_user\_action()

self.textbuf.delete\_selection(True, self.text.get\_editable())

self.textbuf.insert\_at\_cursor(hyp)

ins = self.textbuf.get\_insert()

iter = self.textbuf.get\_iter\_at\_mark(ins)

iter.backward\_chars(len(hyp))

self.textbuf.move\_mark(ins, iter)

self.textbuf.end\_user\_action()

Gdk.threads\_leave()

def \_on\_final\_result(self, asr, hyp):

Gdk.threads\_enter()

#Insert the final result

self.textbuf.begin\_user\_action()

self.textbuf.delete\_selection(True, self.text.get\_editable())

self.textbuf.insert\_at\_cursor(hyp)

if (len(hyp) > 0):

self.textbuf.insert\_at\_cursor(" ")

self.textbuf.end\_user\_action()

self.get\_command

Gdk.threads\_leave()

def get\_command(self):

driver = webdriver.Firefox()

if (self.textbuf=="run browser"):

os.system('/usr/bin/firefox')

if (self.textbuf=="create new tab"):

os.system('/usr/bin/firefox -new-tab')

if (self.textbuf=="create new window"):

os.system('/usr/bin/firefox -new-window')

if (self.textbuf=="create private tab"):

os.system('/usr/bin/firefox -private')

if (self.textbuf=="create private window"):

os.system('/usr/bin/firefox -private-window')

if (self.textbuf=="close tab"):

driver.close()

if (self.textbuf=="close browser"):

driver.quit()

if (self.textbuf=="watch movie"):

os.system('/usr/bin/firefox -url kinogo.co')

if (self.textbuf=="listen to music"):

os.system('/usr/bin/firefox -url https://music.yandex.ru')

if (self.textbuf=="show settings"):

os.system('/usr/bin/firefox -preferences')

if not (self.textbuf=="open images"):

os.system('/usr/bin/eom')

if not (self.textbuf=="run terminal"):

os.system('/usr/bin/terminal')

if not (self.textbuf=="open my documents"):

os.system('/usr/bin/caja --no-desktop --browser')

if not (self.textbuf=="run notes"):

os.system('/usr/bin/pluma')

if not (self.textbuf=="run calculator"):

os.system('/usr/bin/gnome-calculator')

if not (self.textbuf=="run office tables"):

os.system('/usr/bin/libreoffice --calc')

if not (self.textbuf=="run office impress"):

os.system('/usr/bin/libreoffice --impress')

if not (self.textbuf=="run office draw"):

os.system('/usr/bin/libreoffice --draw')

if not (self.textbuf=="run office writer"):

os.system('/usr/bin/libreoffice --writer')

def button1\_clicked(self, button):

#Handle button Speak presses

self.asr.set\_property("silent", False)

def button2\_clicked(self, button):

#Handle button Stop presses

self.asr.set\_property("silent", True)

if \_\_name\_\_ == '\_\_main\_\_':

app = DemoApp()

Gdk.threads\_enter()

Gtk.main()

Gdk.threads\_leave()

## Приложение 2. Скрипт для расчета уровня ошибок

## распознавания

#!/bin/bash

KALDI\_ROOT=`pwd`/../../..

export PATH=$PWD/../../../egs/voxforge/s5/utils/:$KALDI\_ROOT/src/bin:$PATH

# Alignments and decoding results are saved in this directory

decode\_dir="./work"

data\_file="online-data"

ac\_model=${data\_file}/models/nnet

audio1=${data\_file}/audio

#if [ $# != 1 ]; then

# echo "Usage: transcribe-audio.sh <audio>"

# echo "e.g.: transcribe-audio.sh dr\_strangelove.mp3"

# exit 1;

#fi

! GST\_PLUGIN\_PATH=../src gst-inspect-1.0 kaldinnet2onlinedecoder > /dev/null 2>&1 && echo "Compile the plugin in ../src first" && exit 1;

if [ ! -f HCLG.fst ]; then

echo "Run ./prepare-models.sh first to download models"

exit 1;

fi

audio=$audio1/$1

for f in $audio1/\*.m4a; do

resultfile=$decode\_dir/`basename $f .m4a`.hyp

echo "Decoding $f, result goes to $resultfile"

GST\_PLUGIN\_PATH=../src gst-launch-1.0 --gst-debug="" -q filesrc location=$audio ! decodebin ! audioconvert ! audioresample ! \

kaldinnet2onlinedecoder \

use-threaded-decoder=true \

model=final.mdl \

fst=HCLG.fst \

word-syms=words.txt \

feature-type=mfcc \

mfcc-config=conf/mfcc.conf \

ivector-extraction-config=conf/ivector\_extractor.fixed.conf \

max-active=7000 \

beam=11.0 \

lattice-beam=5.0 \

do-endpointing=true \

endpoint-silence-phones="1:2:3:4:5:6:7:8:9:10" \

chunk-length-in-secs=0.2 \

! filesink location=/dev/stdout buffer-mode=2

done

# Convert the reference transcripts from symbols to word IDs

sym2int.pl -f 2- $ac\_model/words.txt < $audio1/trans.txt > $decode\_dir/ref.txt

# Convert the hypotheses from symbols to word IDs, also remove the segmentation symbol "<#s>"

for f in $decode\_dir/\*.hyp; do

(echo -n `basename $f .hyp`" " ; cat $f) | sed 's/<#s>//g' | sym2int.pl -f 2- $ac\_model/words.txt;

done > $decode\_dir/hyp.txt

# Finally compute WER

compute-wer --mode=present ark,t:$decode\_dir/ref.txt ark,t:$decode\_dir/hyp.txt

1. SNR (англ. signal-to-noise ratio) – отношение мощности полезного сигнала к мощности шума. [↑](#footnote-ref-1)