РЕФЕРАТ

Объем дипломного проекта составляет 100 страниц, в нем содержится 5 таблиц, 6 иллюстраций, 15 приложений и 20 источников литературы.

Ключевые слова: программное обеспечение, веб-ориентированная система, голосовое управление, распознавание речи, обучаемый помощник.

Целью работы является разработка и программная реализация модульной обучаемой веб-ориентированной системы голосового управления, предоставляющей интерфейс для написания пользователями собственных голосовых функций.

В работе рассматриваются современные технологии распознавания речи и методы классификации текстовых документов. Производится обзор существующих аналогов. Приводится обоснование выбора и характеристика комплекса технических средств, описание архитектуры программного продукта и оценка его информационных характеристик.

Результатом дипломного проектирования стала система голосового управления «Джарвис», разработанная в двух версиях: в виде desktop-приложения под все современные операционные системы и в виде web-приложения для браузеров, основанных на движке webkit (Google Chrome, Yandex Browser).

СОДЕРЖАНИЕ

1. Введение

2. Аналитический обзор

2.1 История развития систем распознавания речи

2.2 Обзор существующих систем с речевым интерфейсом

2.2 Классификация систем распознавания речи

2.3 Методы и алгоритмы систем распознавания речи

2.4 Методы и алгоритмы систем классификации текстовых документов

2.5 Сравнение, актуальность и выводы

3. Практическая часть

3.1 Обоснование выбора и характеристика языка программирования

3.2 Обоснование выбора и характеристика комплекса программных и аппаратных средств

3.3 Разработка алгоритма функционирования системы

3.4 Характеристика разработанной системы

3.5 Выводы

1. -описание математических методов, обоснование вводимых ограничений и допущений, используемых при решении задачи;
2. -описание алгоритмов и возможного взаимодействия программ с другими программами;
3. -описание и обоснование выбора метода организации входных и выходных данных;
4. -обоснование выбора языка программирования;
5. -описание программ и их применения;
6. -руководство для программиста, программиста и оператора;

ВВЕДЕНИЕ

Повсеместное внедрение средств вычислительной техники в различные сферы человеческой деятельности выводит актуальность разработки дружелюбного пользовательского интерфейса на новый уровень. Очевидно, что для человека самым естественным, быстрым и эффективным способом передачи информации является речь. Необходимость создания речевых интерфейсом не вызывает сомнения, поскольку они обладают рядом бесспорных и очевидных преимуществ:

* Преодоление физических недостатков. Для пользователей с нарушением зрения или нарушениями моторных функций зачастую трудно работать с компьютерами посредством клавиатуры и мыши. Приложения с качественным речевым интерфейсом предоставляют людям с данными заболеваниями более комфортный способ взаимодействия c машинами.
* Простота. Не требуется обучаться навыкам работы с компьютером. Речевой интерфейс предоставляет интуитивно-понятный способ взаимодействия.
* Обеспечение мобильности. Речевой интерфейс позволяет человеку взаимодействовать с компьютером посредством беспроводной гарнитуры, обеспечивая не зависимое от расположения человека взаимодействие.
* Удобство использования даже при неблагоприятных условиях. Акустические системы имеют большое преимущество когда затрудняется восприятие визуальной информации, например в условиях темноты, слишком большого количества света, бликах, вибрации.
* Свобода рук и глаз. Речевые интерфейсы позволяют выполнять сразу две задачи одновременно, например работать с измерительным устройством и вводить отображаемые значения в систему с помощью голоса. Это преимущество не только в терминах времени, но и в терминах удобства. Соответственно снижается и утомляемость.
* Эффективность. Современные системы распознавания речи позволяют распознать 180 слов в минуту с высокой точностью.

Современность темы дипломной работы обусловлена тем, что рынок систем распознавания голоса рос и продолжает стремительно развиваться. По данным исследований BBC, размер рынка в 2011 году составлял 47 миллиардов долларов, 53 миллиарда в 2012 году и ожидается рост до 113 миллиардов к 2017 году, что составляет в общей сложности 16.2% прироста. Это обусловлено тем, что такие технологические гиганты как Google, Facebook, Apple и Microsoft продолжают инвестировать в развитие данных технологий. При этом большую часть рынка занимает потребительский сектор, что отражает увеличивающийся интерес у конечных пользователей современных компьютеров и мобильных устройств. Следует отметить, что увеличивается спрос и в сфере здравоохранения.

Практической ценностью работы является разработка кросс-платформенной расширяемой системы с голосовым интерфейсом.

Объектом исследования являются средства распознавания речи и методы классификации текстовых документов.

Предметом исследования является анализ возможности применения средств распознавания речи в задаче разработки ПО с голосовым интерфейсом под десктопные и браузерные платформы.

Цель работы - разработка и программной реализации модульной обучаемой веб-ориентированной системы голосового управления, предоставляющей интерфейс для написания пользователями собственных голосовых функций.

Для достижения цели были поставлены следующие задачи:

* Произвести анализ предметной области.
* Определить комплекс программных средств
* Разработать схему функционирования системы с голосовым управлением.
* Разработать модульную систему с голосовым управлением.
* Реализовать систему плагинов для разработанной системы

Дипломная работа состоит из введения, аналитического обзора, основной части, экономического раздела, раздела охраны труда и безопасности жизнедеятельности, заключения.

В введении указывается цель и задачи для ее достижения, обосновывается современность и актуальность решаемой задачи.

В аналитическом обзоре производится анализ предметной области: рассматривается история развития, классификация и алгоритмы функционирования систем распознавания речи, а так же производится обзор существующих аналогов разрабатываемой системы.

В основной части работы осуществляется последовательное изложение этапов реализации поставленной перед автором задачи. Производится обоснование выбора и характеристика языка программирования, а так же комплекса программных и аппаратных средств для реализации описанной системы. Описываются алгоритмы, методы и вводимые ограничения. Подробно рассматривается схема функционирования разработанной системы, приводится руководство для программиста и оператора. Описываются результаты разработки, в частности, характеристики разработанной системы.

В экономическом разделе приводятся результаты обоснования принятых решений с точки зрения их экономической целесообразности. Приводятся расчеты себестоимости разработанного продукта и сроков окупаемости, капитальных и эксплуатационных затрат, его конкурентоспособности.

В разделе охраны труда и безопасности жизнедеятельности производится анализ условий труда на рабочем месте инженера-программиста и разрабатываются меры по их улучшению.

В заключении приводятся выводы о соответствии разработанного программного продукта исходной задаче, указываются основные и смежные области возможного его применения, а так же делаются выводы о направлениях и перспективах его дальнейшего развития.

1 АНАЛИТИЧЕСКИЙ ОБЗОР

1.1 Современное состояние проблемы

Еще со времен появления первых ЭВМ люди мечтали о том, чтобы можно было общаться с компьютером так же, как они общаются между собой, а не заставлять пользователей приспосабливаться к языку, который понятен машине. Речь - это самое прямое и естественное средство общения между людьми, и как следствие, это должно быть самым естественным способом взаимодействия с машинами. К сожалению, это было невозможно осуществить из-за технологических ограничений, в частности из-за того, что точность распознавания речи была неприемлемо низкой. Однако так было до недавнего времени. От узнавания звуков к интерпретации отдельных слов и к пониманию речи. Разработка качественных речевых интерфейсов стала возможной благодаря серии удивительных технологических открытий. Мощное сочетание технологий распознавания речи и обработки естественного языка, а так же значительное увеличение вычислительной мощности и доступные объемы памяти среднего персонального компьютера раскрывает огромные возможности для создания новых, более удобных интерфейсов продуктов. Теперь нет никаких проблем с техническим обеспечением, необходимым для работы систем распознавания голоса. А увеличение количества поставщиков этой технологии и конкуренция снизили цены разработок до вполне приемлемого уровня.

Актуальность выбранной темы обусловлена так же тем, что в настоящее время пользователями вычислительных машин и средств, оснащенных вычислительными машинами, становятся люди, не являющиеся специалистами в области программирования. А в некоторых областях применения речь стала единственно возможным средством общения с техникой (в условиях перегрузок, темноты или резкого изменения освещенности, при занятости рук, чрезвычайной сосредоточенности внимания на объекте, который не позволяет отвлечься ни на секунду, и т.д.).

1.2 Анализ предметной области

В общем случае голосовой интерфейс может быть представлен в виде системы распознавания речи, системы синтеза речи и системы классификации текста. Рассмотрим каждую из них подробней.

1.2.1 Система распознавания речи

Речь является основным средством человеческого общения. Проектирование машины, которая имитирует человеческое поведение, в частности, способность говорить на естественном языке и реагировать должным образом на человеческую речь интриговала инженеров и ученых на протяжении десятилетий. Чтобы понять возможности современных систем распознавания и как они развились до их текущей формы, исторический обзор может оказаться полезным.

Исследования концепций речевых технологий начались в 1936 году в американской компании Bell Labs. В 1939 году Bell Labs продемонстрировала первую машину синтеза речи, однако от усилий по развитию систем распознавания речи компания отказалась. Данное решение было основано на неверном заключении, что для успеха необходим искусственный интелект. Последующие разработки технологии распознавания речи продолжись в 1950-х годах, когда исследователи пытались использовать фундаментальные идеи акустической фонетики. В 1952 году исследователи Bell Labs разработали систему для изолированного распознавания чисел, сказанных одним голосом. Система полагалась на измерения спектральных резонансов в гласной области каждой цифры. В 1959 году в университете Англии был разработан фонетический распознаватель, позволявший распознавать 4 гласных звуков и 9 согласных. Путем добавление статистической информации о допустимой последовательности фонем в английском языке они увеличили точность распознавания фонем для слов, состоящих из двух и более фонем. Эта работа отмечена как первое использование статистической информации о синтаксисе на уровне фонем в задачах распознавания речи. Еще одним достижением этого года был распознаватель гласных, спроектированный в MIT Licoln laboratories и позволявший распознавать 10 гласных звуков в спикеронезависимой манере.

В 1960-х годах было опубликовано несколько фундаментальных идей распознавания речи. Одной из самых значительных проблем того периода являлась неоднородность шкал времени в речевых событиях. В 1960-х годах были начаты три ключевые исследовательские программы, которые оказали огромное влияние на исследование и разработку речевых технологий в течение последних 20 лет. Первым из этих проектов были усилия исследователей компании RCA Laboratories, направленные на решение описанной проблемы. Они разработали набор методов нормализации времени, основанные на способности достоверно определить время начала и окончания речи, что значительно уменьшило изменчивость оценок распознавания. В конечном счете один из этих исследователей основал первую коммерческую компанию распознавания речи Threshold Technology, которая разработала продукт VIP-100, способный распознавать до 200 слов. Примерно в тоже время исследователь СССР и исследователи NEC Laboratories независимо друг от друга предложили использовать метод динамического программирования (Dynamic Time Warping - DTW) для временного выравнивания пар речевых высказываний для решения проблемы неоднородности.

Последним достижением этого периода были новаторские исследования в области непрерывного распознавания речи с помощью динамического отслеживания фонем.

Так же в этом десятилетии на арену технологий распознавания речи вступили японские компании, разработав свои распознаватели гласных звуков, фонем и цифр.

**1970-1980**

Главным предметом исследований 1970-х годов было распознавание изолированных слов. Американский исследователь Itakura показал как идея кодирования линейного предсказания (Linear Predictive Coding - LPC), которая успешно использовалась для кодирования речи высокого качества при низкой скорости передачи данных, может быть применена в системах распознавания речи. Исследователи компании AT & T Bell Labs начали серию экспериментов, направленных на создание полностью спикеронезависимой ASR системы. Чтобы достигнуть этой цели был использован широкий спектр сложных алгоритмов кластеризации для определения числа различных шаблонов, необходимых для представления всех вариантов слов. Это исследование было усовершенствовано за более чем десятилетие, так что методы создания спикеронезависимой системы распознавания речи сейчас хорошо известны и широко применяются. В это время появилась амбизиозная программа, которая финансировалась агенством департамента обороны США Defense Advanced Research Projects Agency (DARPA) и привела ко многим важным результатам. Первыми успеха достигли исследователи Университета Карнеги Меллона, разработав систему Hearsay I в 1973 году. Система была способна использовать семантическую информацию чтобы значительно уменьшить количество альтернатив, рассматриваемых распознавателем. А их следующая система «Harpy» могла распознать 1011 слов с высокой точностью. Одним важным вкладом системы «Harpy» является концепция поиска на графах, где язык распознавания речи представляется в виде связанной сети, полученной из лексического представления слов, синтаксических правил и правил словообразования.

**1980-1990**

Так же как распознавание изолированных слов было главным объектом исследований 1970-х годов, проблема распознавания связанных слов являлась главным объектом исследований 1980-х годов. Здесь целью было разработать надежную систему, способную распознавать быстро сказанную фразу из связанных слов.

Было сформулировано и реализовано широкое разнообразие алгоритмов, основанных на соответствии шаблону соединения отдельных слов, в том числе двухуровневый подход динамического программирования. Каждый из методов имел свои преимущества и недостатки. Исследования речи в 1980-х годах можно охарактеризовать сдвигом методологий от более интуитивного подхода, основанного на шаблонах, к методам статического моделирования. Одной из ключевых технологий разработанных за этот период является статистический метод, известный как скрытая марковская модель (Hidden Markov Model – HMM). Используя шаблоны для слов и звуковые паттерны, она рассматривала вероятность того, что неизвестные звуки могли быть словами. В середине 1980-х эта техника стала широко применяться практически в каждой лаборатории распознавания речи в мире. Эта база использовалась другими системами еще на протяжении двадцати лет. Другой «новой» технологией, которая была вновь введена в конце 1980-х была идея применения нейронных сетей для распознавания речи. Нейронные сети впервые были введены в 1950-х, но они не доказали своей полезности сразу так как имели много практических проблем. Однако теперь было достигнуто более глубокое понимание сильных сторон и ограничений данной технологии. Таким образом было предложено несколько других способом реализации таких систем.

**1.3 Обзор существующих систем с голосовым интерфейсом**

**http://www.popsci.com/cortanas-coming-desktop-could-it-be-smarter**

Разрабатываемый голосовой помощник будет работать на десктопных операционных системах и в браузерах. Выбор таких платформ обусловлен тем, что на сегодняшний день мобильный рынок голосовых помощников переполнен. Даже не смотря на то, что в каждой из трех лидирующих мобильных ОС уже встроен свой интеллектуальный голосовой помощник от ведущих мировых производителей, существует большое количество их аналогов. В тоже время на рынке десктопных и браузерных приложений прослеживается дефицит ПО с речевым интерфейсом. Таким образом, в данном разделе будут рассматриваться только десктопные и браузерные голосовые ассистенты.

Среди веб-приложений можно выделить следующих представителей:

1. SpeechTexter и Speechnotes – оба этих продукта можно объединить в один пункт. так как все что они делают, это позволяют пользователю надиктовывать текст с возможностью последующего его копирования в буфер обмена.
2. Alexa Web – это новый продукт компании Alexa, дочерней компании Amazon.com. Он представляет веб-приложение, позволяющее общаться с Alexa Voice Service точно так же как это делает их технологии Alexa Echo и Fire TV, однако с существенными ограничениями. Веб-приложение дает доступ к более чем 100 способностям технологии Alexa Voice Service.

Среди десктопных приложений наиболее продвинутыми являются:

1. Dragon Assistant – продукт, входящий в пакет программного обеспечения Dragon NaturallySpeaking компании Dragon Systems, лидера разработки речевых технологий. Данный продукт предоставляет широкие возможности по голосовому управлению настольным компьютером: осуществлять поиск в браузере, публиковать обновления в социальные сети, проверять почту и т.д.
2. Cortana – персональный голосовой помощник компании Microsoft, интегрированный в Windows 10. Cortana так же предоставляет широкий спектр возможностей: помогает находить файлы, устанавливать напоминания, искать информацию, выполнять вычисления и конверсию, и даже распознавать музыку. При надобности, она может аккумулировать информацию, получаемую из различных приложений чтобы предугадывать потребности пользователя. Однако она не доступна на русском языке.
3. Awa – голосовой помощник компании Stahl Labs AB. Он предназначен для выполнения того же стандартного набора задач, но только под операционной системой Mac OS X.

Все рассмотренные системы, так же как и подавляющее число существующих, имеют ряд общих особенностей: ориентация на конкретную операционную систему (за исключением Dragon Assistant) и английский язык, а так же закрытость, т.е. отсутствие у пользователя возможности напрямую обучать систему, вводить новую функциональность.

Таким образом разрабатываемая система с голосовым интерфейсом «Джарвис» будет иметь ряд преимуществ:

* Кроссплатформенность. Прежде всего, она не будет привязана к конкретным аппаратным устройствам или операционным системам, а может функционировать как на мобильных устройствах в виде веб-приложения, так и на устройствах под управлением Windows, Linux или OS X в виде desktop-приложения.
* Самообучаемость. То есть ассистент буквально будет уметь обучаться новым командам от своего владельца. Если система не поняла чего-то, то она всегда спросит «Что с этим делать?», а пользователь сможет направить ее, сказав какое из действий, приведенных в подсказке, выполнить.
* Система плагинов. Приложение будет предоставлять простое и удобное API для создания собственных голосовых функций (плагинов). Вместе с тем, будет реализована система управления плагинами: Джарвис будет предоставлять графический интерфейс для доступа к списку всех доступных плагинов и автоматизирует процесс установки / обновления / удаления их по клику без каких-либо дополнительных действий.
* Русский язык. На сегодняшний день все рассмотренные системы в первую очередь ориентированы на англоязычное население, а некоторые и вовсе поддерживают только его. В связи с этим новая функциональность появляется с существенной задержкой для русскоговорящих пользователей. Таким образом лучшей альтернативой может стать стороннее решение, а именно то, что изначально создавалось для русского языка.
* Мультиязычность. Несмотря на изначальную направленность разрабатываемой системы на русскоговорящих пользователей, алгоритм функционирования системы будет реализован таким образом, что система будет так же хорошо выполнять голосовые функции для большинства других языков.

**1.4 Существующие методы решения задачи**

Для реализации поставленного рода задач, не существует типовых решений. Поэтому рассмотрим варианты решений для составных частей системы.

1.4.1 Подсистема распознавания речи

Дипломная работа автора не предполагает реализацию собственной системы распознавания речи, так как это является очень сложной, трудоемкой и ресурсозатратной задачей, которую сложно выполнить в рамках данной работы. В связи с этим необходимо провести обзор уже существующих систем преобразования голоса в текст, которые имеют открытое API для интеграции с разрабатываемой системой.

Прежде всего, исходя из требований к разрабатываемой системе, следует отметить основные характеристики, по которым производится выбор оптимальной аудио-системы распознавания речи:

* Поддержка русского языка
* Мультиязычность
* Доступность на всех ОС
* Скорость и точность обработки
* Высокие нагрузки
* Надежность и отказоустойчивость
* Ограничения на количество обращений
* Сложность интеграции

На сегодняшний день существует достаточно большой выбор среди систем распознавания речи, но лишь немногие соответствуют требованиям, представленным выше. Рассмотрим их:

1. Yandex SpeechKit

Прежде всего, взаимодействие с Yandex SpeechKit ведется через HTTP API. Из этого следует, что технология является доступной для всех платформ и операционных систем, а затраты на развертывание и поддержку собственной инфраструктуры – минимальны. Доля верно распознаваемых русских слов колеблется в диапазоне от 82 до 95%. Распознавание может происходить в реальном времени одновременно с передачей звуковых данных. Задержка от окончания отправки данных до получения результата распознавания не превышает 1 секунды. Инфраструктура SpeechKit спроектирована с учетом высоких нагрузок, что обеспечивает доступность и безотказную работу системы даже при большом количестве одновременных обращений. Кроме того, технология от Yandex обеспечивает простоту интерфейсов, высокую доступность и надежность. Так же, стоит отметить, что данная технология является одной из лучших для распознавания русского и турецкого языка, однако остальные языки она не поддерживает.

1. Google Cloud Speech

Google Cloud Speech, так же как и Yandex SpeechKit, предоставляет разработчикам удобный для использования web API для конвертации аудио в текст с помощью мощных моделей нейронных сетей. Однако, в отличии от технологии Яндекса, Google Cloud Speech распознает более 80 языков. Данная технология обладает так же рядом других особенностей: обработка шума и потоковая передача данных, т.е. возвращение промежуточных результатов распознавания как только они становятся доступны еще во время разговора. Минусом данной технологии является ограничение на количество запросов: до 10 000 в сутки.

1. HTML5 Speech Recognition

HTML5 Speech Recognition является одной из малоизвестных составляющих спецификации HTML5, которую сейчас поддерживают только браузеры, основанные на Webkit. Таким образом, будучи технологией, встроенной в браузеры, она не накладывает каких-либо ограничений на количество запросов в сутки и даже не требует подключения к сети Интернет. Данная технология способна распознавать как непрерывную речь, так и одиночные фразы, а так же работает с большим количеством языков, включая русский. Соответственно ее минусом является то, что не все современные браузеры ее поддерживают. Стоит отметить, что в HTML5 спецификации есть еще одна малоизвестная технология - Speech Synthesis, которая позволяет выполнить обратную операцию: синтезировать речь, используя большое разнообразие голосов.

Таким образом, рассмотрев наиболее подходящие, для поставленной задачи, технологии распознавания речи, можно сделать вывод, что по своей библиотеке данных наиболее точным следует считать продукт Google Cloud Speech. Он более всего подходит для задач распознавания на базе разрабатываемой системы. Однако необходимо заметить, что данный инструментарий имеет ограничение на максимальное количество запросов в сутки, что является критическим недостатком. В то же время у технологии от Яндекса таким недостатком является поддержка слишком малого количества языком. Следовательно, наиболее оптимальным выбором будет являтся HTML5 Speech Recognition, которая изначально интегрирована в браузер, имеет высокую точность и скорость распознавания, поддерживает множество языком, не накладывает каких-либо ограничений по количеству запросов, а так же может работать с непрерывным речевым потоком.

1.4.2 Подсистема классификации текста

После того как речь была распознана, необходимо произвести интеллектуальный анализ текста, а именно – классификацию. Задача классификации - отнесение объекта к одной из заранее определенных категорий на основании его формализованных признаков.

Перед тем, как приступать к любой задаче классификации, одна из наиболее фундаментальных процедур, которую необходимо выполнить, это выбрать представление документа и отобрать признаки. Хоть отбор признаков и применяется довольно часто в других задачах классификации, он особенно важен в задаче классификации текстов ввиду высокой размерности (большого количества признаков) и наличия нерелевантных (шумовых) признаков. В большинстве случаев, представление текста происходит одним из двух способов. Первый – документ как набор слов, в котором документу сопоставляются слова и частота их встречаемости в нем. Такое представление независимо от порядка слов, в котором они встречаются в тексте. Второй метод состоит в представлении текста собственно как набор строк, в котором документ является последовательностью слов. Большинство алгоритмов классификации текстов используют первое представление ввиду его простоты и удобства для задач классификации. Наиболее популярными способами отбора признаков являются удаление стоп-слов и стемминг. При удалении стоп-слов, мы определяем общие слова документов, которые не являются специфичными или разделяющими для разных классов. При стемминге, разные формы одного слова приводятся к одному виду. Например, так объединяются слова разного рода/формы/времени/падежа и пр. Подобная обработка — уже устоявшийся способ уменьшения размерности задачи и улучшения производительности.

Существует достаточно большое количество методов классификации текстовых документов. Хорошо себя зарекомендовали SVM, метод k-ближайших соседей, нейронные сети, деревья решений, классификаторы на основе регрессии, наивный байесовский классификатор и классификатор Роше [11]. Остановимся на некоторых из них поподробнее.

1. Наивный байесовский классификатор

Наивный Байесовский классификатор (Naшve Bayes Classifier, NBС) является простым вероятностным классификатором, основанным на применении теоремы Байеса об условной вероятности с наивным предположением о независимости отдельных слов в документе друг от друга. Такое предположение редко бывает верным. Но, как показывает практика, наивные байесовские классификаторы показывают на удивление хорошие результаты во многих задачах, в том числе в тех, где предположение о независимости, строго говоря, не выполняется. Данная модель классификации базируется на понятии условной вероятности принадлежности документа *d*классу *c*. Достоинствами наивного байесовского классификатора являются сравнительная простота имплементации и тестирования, нечувствительность к размерам обучающей выборки, высокая скорость обучения и устойчивость к так называемому переобучению (overfitting — явление, при котором алгоритм очень хорошо работает на тренировочной выборке и плохо на тестовой). Поэтому их применяют либо когда исходных данных очень мало, либо когда таких данных очень много, и на первое место выходит скорость обучения. Недостатки: не самая лучшая точность, особенно если нарушается предположение о независимости.

1. Машина опорных векторов

Машина опорных векторов (Support Vectors Machine, SVM) была предложена для числовых данных. Но ее можно применять и для классификации документов. Главным принципом SVM является определение разделителя в искомом пространстве, который разделяет классы наилучшим образом. Т.е. SVM умеет определять принадлежность документов только к одному из двух классов. Следовательно, чтобы классифицировать документ на принадлежность к одному из N классов, потребуется N классификаторов для каждого класса. Для распознавания текста машина опорных векторов предполагает создание словаря – списка всех слов в обучающей выборки и представление документов в виде вектора, где i-элемент вектора это мера вхождения i-го слова словаря в документ. По мере обучения классификатора, получаются вектора большой размерности, большинство элементов которых – нули. Таким образом данный метод достаточно медлителен, особенно для данных высокой размерности, таких, как текстовые документы.

1. Искусственные нейронные сети

Искусственные нейронные сети (ИНС) - это большой класс систем, архитектура которых имеет аналогию с построением нервной ткани из нейронов [12]. ИНС состоит из набора «нейронов», соединенных между собой. Каждый нейрон представляет собой элементарный преобразователь входных сигналов в выходные. Выходные сигналы вычисляются как функция от входных сигналов. Как правило, передаточные функции всех нейронов в сети фиксированы, а веса являются параметрами сети и могут изменяться. Некоторые входы нейронов помечены как внешние входы сети, а некоторые выходы - как внешние выходы сети. Подавая любые числа на входы сети, мы получаем какой-то набор чисел на выходах сети. Таким образом, работа нейросети состоит в преобразовании входного вектора в выходной вектор, причем это преобразование задается весами сети. Для того чтобы сеть решала заданную функцию, ее надо "натренировать" на данных, для которых известны и значения входных параметров, и правильные ответы на них. Тренировка состоит в подборе весов межнейронных связей, обеспечивающих наибольшую близость ответов сети к известным правильным ответам.

Для выбора метода классификации текста необходимо сравнить результаты работы имплементаций различных методов под конкретную платформу, полученные с помощью кросс-проверки на обучаемой выборке. Однако стоит заметить, что, как правило, наиболее эффективными системами классификации являются нейронные сети, потому что генерируют фактически большое число регрессионных моделей (которые используются в решении задач классификации статистическими методами).

1. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

2.1 Обоснование выбора и характеристика языка программирования

2.1.1 JavaScript

Для разработки программного обеспечения в качестве языка программирования был выбран Javascript - высокоуровневый, кросс-платформенный, объектно-ориентированный и интерпретируемый язык программирования. Наряду с HTML и CSS, это одна из трех основных веб-технологий, поддерживаемых всеми современными браузерами без расширений. Он широко используется для реализации как клиентской, так и серверной части веб-приложений. Однако Javascript не ограничивается только веб-ориентированными средами выполнения, а так же может использоваться на платформе node.js и базе данных Apache CouchDB. Характерными особенностями языка являются динамическая типизация, автоматическое управление памятью, прототипное наследование, функции как объекты первого класса, поддержка императивного и функционального стиля программирования. Он имеет API для работы с текстом, массивами, датами и регулярными выражениями, но не предоставляет средств для ввода-вывода, полагаясь в этом на среды выполнения.

Выбор данного языка обусловлен следующими факторами:

* Javascript является самым популярным языком программирования в мире, а следовательно имеет огромное сообщество и большое количество информационных и справочных источников
* Javascript можно использовать как на клиентской так и на серверной стороне, что увеличивает скорость разработки, так как можно сосредоточиться на изучении особенностей только одного языка и увеличивается количество переиспользуемого кода
* Javascript поддерживает объектно-ориентированное программирование, что упрощает модульную разработку.
* Он очень быстрый. Платформа node.js – это среда выполнения javascript, которая использует движок V8, разработанный компанией Google для использования в браузере Google Chrome. V8 компилирует JS в нативный машинный код, что делает его быстрее чем когда-либо
* npm - пакетный менеджер node.js. – позволяет быстро и эффективно управлять модулями и зависимостями и содержит огромное количество готовых модулей
* существует множество инструментов, облегчающих разработку и отладку javascript-программ

2.1.2 Node.js и Express.js

Для разработки веб-ориентированной версии системы была выбрана платформа Node.js - среда выполнения JavaScript, основанная на движке V8 и предназначенная для создания масштабируемых распределённых сетевых приложений, таких как веб-сервер. Так же был использован Express.js – минималистичный и гибкий Node.js фреймворк, который предоставляет широкий набор функций, избавляет от необходимо дублировать код и позволяет огранизовать веб-приложение в виде View-Model-Controller (MVC) архитектуры.

2.1.3 Electron

Для реализации desktop-версии приложения была использована технология Electron.

Разработка десктопных приложений является достаточно сложной задачей, так как требуется заботится о таких вещах как упаковка и установка приложения, управление обновлениями, а так же о дийзане приложения и портировании его под различные операционные системы. Поддержка различных платформ обычно предполагает использование целого набора специализированных инструментов.

Electron – это фреймворк от компании GitHub, который позволяет создавать нативные кроссплатформенные приложения с использованием исключительно веб-технологий, таких как: JavaScript, HTML и CSS. Он заботится о всех сложных нюансах разработки десктопных приложений, предоставляя унифицированный интерфейс для взаимодействия с различными операционными системами и позволяя сфокусироваться на сути самого приложения. В добавок к своему собственному набору API, Electron включает в себя весь набор Chromium API, все встроенные модули Node.js и полностью поддерживает модули третьих лиц. Electron использует веб-страницы для создания пользовательского интерфейса, что снимает присущие десктопным приложениям ограничения в дизайне. В данном случае разработчики не ограничены системным набором элементов интерфейса конкретной платформы и могут создавать UI с использованием огромных возможностей HTML5 и CSS3. А учитывая, что данный фреймворк использует один из самых продвинутых браузеров, мы можем использовать самые новые веб-технологии. Кроме того, выбор данной технологии обусловлен так же тем, что дальнейшее портирование десктопного приложения на веб-сервер Node.js займет минимум затрат по причине того, что Electon сам использует Node.js для работы с десктопной средой.

2.1.4 AngularJS

Для реализации клиентской части как десктопного так и веб-приложения был выбран фреймворк AngularJS. AngularJS - это производительный и гибкий клиентский open-source MVW фреймворк для создания кроссбраузерных веб-приложений, разработанный и поддерживаемый компанией Google. Фреймворк предназначен для построения single-page веб-приложений и является набором JavaScript функций для организации кода на стороне клиента. Он разработан с целью отделять клиентскую часть от серверной, а манипуляцию DOM-ом от логики работы приложения. AngularJS адаптирует и расширяет традиционный HTML, он вбирает концепции из разных языков программирования и делает HTML динамическим языком тоже. Это означает, что мы получаем полностью управляемый данными подход к разработке приложений без необходимости обновления модели, обновления DOM и других трудоемких задач. Мы фокусируемся на данных, а данные заботятся о нашем HTML, оставляя нас программировать наше приложение. MVW расшифровывается как Model-View-Whatever, что обеспечивает гибкость в выборе патерна проектирования при разработке приложения. Можно выбрать MVC (Model-View-Controller) или MVVM (Model-View-ViewModel) или MPV (Model-View-Presenter) подход. Отличие MVVM подхода от других заключается в том, что MVVM конкретно направлен на разработку пользовательского интерфейса. Представление (View) состоит из слоя презентации, модель представления (ViewModel) хранит логику представления, а модель (Model) хранит нашу бизнес-логику и данные. Паттерн MVVM был создан, чтобы сделать двустороннюю привязку данных между моделью и представлением проще. Разработчики использовали шаблон проектирования «Обозреватель» чтобы уведомлять представление об изменениях в модели. Автор выбирает путь MVVM так как в последние годы AngularJS склоняется к этому паттерну.

Необходимо так же рассмотреть и другие важные особенности фреймворка для понимания того, как функционирует разработанная система.

* Двустороняя привязка данных – автоматическая синхронизация данных между слоями модей и представлений. Изменения модели распространяются на представление, а изменения представления мгновенно отражаются на модели.
* Динамическая маршрутизация (Routing) – концепция, позволяющая создавать одностраничные веб-приложения. AngularJS позволяет конфигурировать состояние приложения используя маршрутизатор, который предоставляет возможность для каждого URL задать свой контроллер, шаблон, стили и параметры, и соответственно, реагирует на изменения URL-адреса.
* Внедрение зависимостей (Dependency Injection DI) – это шаблон разработки ПО, который определяет как компоненты получают свои зависимости. AngularJS предоставляет самый элегантный способ внедрения зависимостей – при определении любого компонента достаточно включить название компонента, от которого он зависит, в список параметров, а фреймворк позаботится об остальном.
* Модули. Каждое приложение в AngularJS создается с использованием модулей. Эти модули представляют собой контейнер для различных компонентов приложения(контроллеров, сервисов и т.д.), делая код более переиспользуемым и тестируемым. Модули могут иметь зависимости в виде других модулей.
* Контроллеры. AngularJS контроллеры позволяют нам взаимодействовать с моделью и представлением. Это место где описывается презентационная логика чтобы обеспечеть синхронизацию пользовательского интерфейса и модели, и управлять их изменениями.
* Директивы – переиспользвуемые компоненты пользовательского интерфейса. Директива может быть всем чем угодно, она может обеспечить мощную логику для существующего UI элемента, либо быть этим элементом сама по себе, обеспечивая внедряемый шаблон мощной логикой изнутри. Таким образом фреймворк позволяют создавать кастомные UI компоненты и предоставляет большое количество уже встроенных.
* Сервисы. Сервисы в AngularJS – это синглтон объекты, которые инстанциируются только один раз в приложении. Они не зависят от представлений и используются для организации кода, выполняющего общие для всего приложения задачи, а так же для организации коммуникации между компонентами фреймворка. Это место где описывается бизнес-логика приложения. Сервисы бывают нескольких типов: Constant, Value, Service, Factory и Provider.
* Фильтры – еще одна сущность фреймворка, которая используется для обработки данных для отображения их пользователю в надлежащем формате. Они могут использоваться в шаблонах, контроллерах или сервисах.
* Декораторы. Декораторы созданы для модификации, переопределения, расширения сервисов на этапе конфигурации.

2.1.5 Atom

В качестве среды разработки был выбран Atom – открытый, кроссплатформенный текстовый редактор от компании GitHub, расширяемый плагинами, написанными на Node.js. Редактор предоставляет средства кросс-платформенного редактирования кода, включает встроенный пакетный менеджер и интерфейс навигации по файловой системе, предоставляет средства для одновременной совместной работы с кодом, обладает интеллектуальной системой автодополнения ввода, предоставляет режимы совместимости с Vim и Emacs. Несколько файлов могут быть открыты в разных вкладках и одновременно отображены с использованием вертикального или горизонтального разбиения панелей. Интерфейс может настраиваться через темы оформления, поддерживаются вкладки, закладки, умный контекстный поиск кода, схлопывание блоков кода, одновременное использование нескольких курсоров и областей выделения, наглядная пометка изменений, автодополнение и проверка кода для разных языков (Ruby, Python, SQL, PHP, Perl, Objective-C, C/C++, JavaScript, Java, Go и т.п.). Для формирования статей и документации может быть использована разметка Markdown. Функциональность редактора формируется за счёт предоставления набора пакетов-дополнений, для установки которых предлагается встроенный пакетный менеджер «apm», похожий на «npm» от проекта Node.js. Формат пакетов аналогичен npm и отличается предоставлением некоторых дополнительных блоков для определения меню, стилей, клавиатурных комбинаций, задания логики активации. Разработка дополнения мало чем отличается от создания приложения для Node.js, в том числе доступны все модули Node.js, а также популярные JavaScript-библиотеки, такие как jQuery, Underscore и SpacePen. Через дополнения реализованы все функции, выходящие за рамки базового редактирования кода, в том числе панели, подсветка синтаксиса, оформление интерфейса, формы работы с файлами и т.п. Кроме базовых дополнений предоставляется каталог сторонних пакетов, в котором уже присутствует более двух тысяч дополнений и почти семьсот тем оформления. Стоит отметить, что Atom так же был разработан на технологии Electron, а Electron ранее носил название Atom-Shell.

2.1.6 Gulp

Для автоматизации сборки проекта был использован Gulp. Gulp – это быстрый потоковый Javascript task-менеджер, написанный на Node.js. Он используется для автоматизации скучных и рутинных, но, от того, не менее важных, задач, которые приходится постоянно выполнять в процессе разработки проекта. Такие задачи включают в себя, к примеру, отслеживание изменений в файлах проекта и повторный запуск связанных с этими файлами задач, таких как запуск модульных тестов, линтеров, конкатенацию файлов, минификацию, препроцессинг CSS, компиляцию JavaScript спецификации es2015 в предыдущие версии для поддержки кроссбраузерности кода. Просто создав task-файл, можно проинструктировать task-менеджер, каким образом следует выполнить ту или иную задачу. Довольно простая идея, которая позволяет сэкономить очень много времени, и помогает сохранять фокус на задачах, связанных непосредственно с разработкой проекта.

2.1.7 Git

Так же для организации рабочего процесса был использован Git. Git – это быстрая, масштабируемая, распределенная система контроля версий с широким набором команд. Она позволяет иметь «версии проекта», которые показывают изменения, которые были внесены в код с течением времени, и позволяет в случае необходимости возвратиться и отменить эти изменения. Сама по себе возможность сравнивать два варианта или отменять изменения делает системы контроля версий неоценимыми при работе над крупными проектами. Данная технология так же незаменима в случаях когда необходимо сделать backup, просмотреть историю кода, поэкспериментировать не затрагивая работающий код или узнать в какой момент все сломалось.

В суммарном итоге, для разработки системы с голосовым интерфейсом были использованы следующие инструментальные средства:

* JavaScript
* Node.js
* Express.js
* Electron
* Angular.js
* Atom
* Gulp
* Git

Все компоненты выбранного комплекса программных средств распространяются под свободными лицензиями, таким образом, разработанная система «Джарвис» была реализована без использования дорогостоящего проприетарного программного обеспечения.

2.2 Архитектура приложения

Архитектуру веб-приложения можно разделить на клиентскую часть и серверную часть (см. Рисунок 1). Клиентская часть реализует пользовательский интерфейс посредством фреймворка AngularJS. Серверная часть, реализованная на платформе Node.js, представляет собой REST API.

Архитектуру десктопного приложения можно разделить на два типа процессов (см. рисунок 1).

Первый тип — **основной процесс**, который отвечает за интеграцию и взаимодействие с GUI операционной системы. Под этим понятием скрывается интеграция в док на OS X или панель задач на Windows, а также сворачивание в трей, полноэкранный режим и прочие обыденные и нативные для ОС задачи. Такой процесс может быть запущен только один раз на всю жизнь приложения.

Второй тип — **процесс рендеринга**, отвечающий за отображение окна браузера, в котором с помощью одной магической строчки может быть открыта страница приложения или любая другая веб-страница. Таких процессов может быть произвольное число. За создание процесса рендеринга отвечает основной процесс.

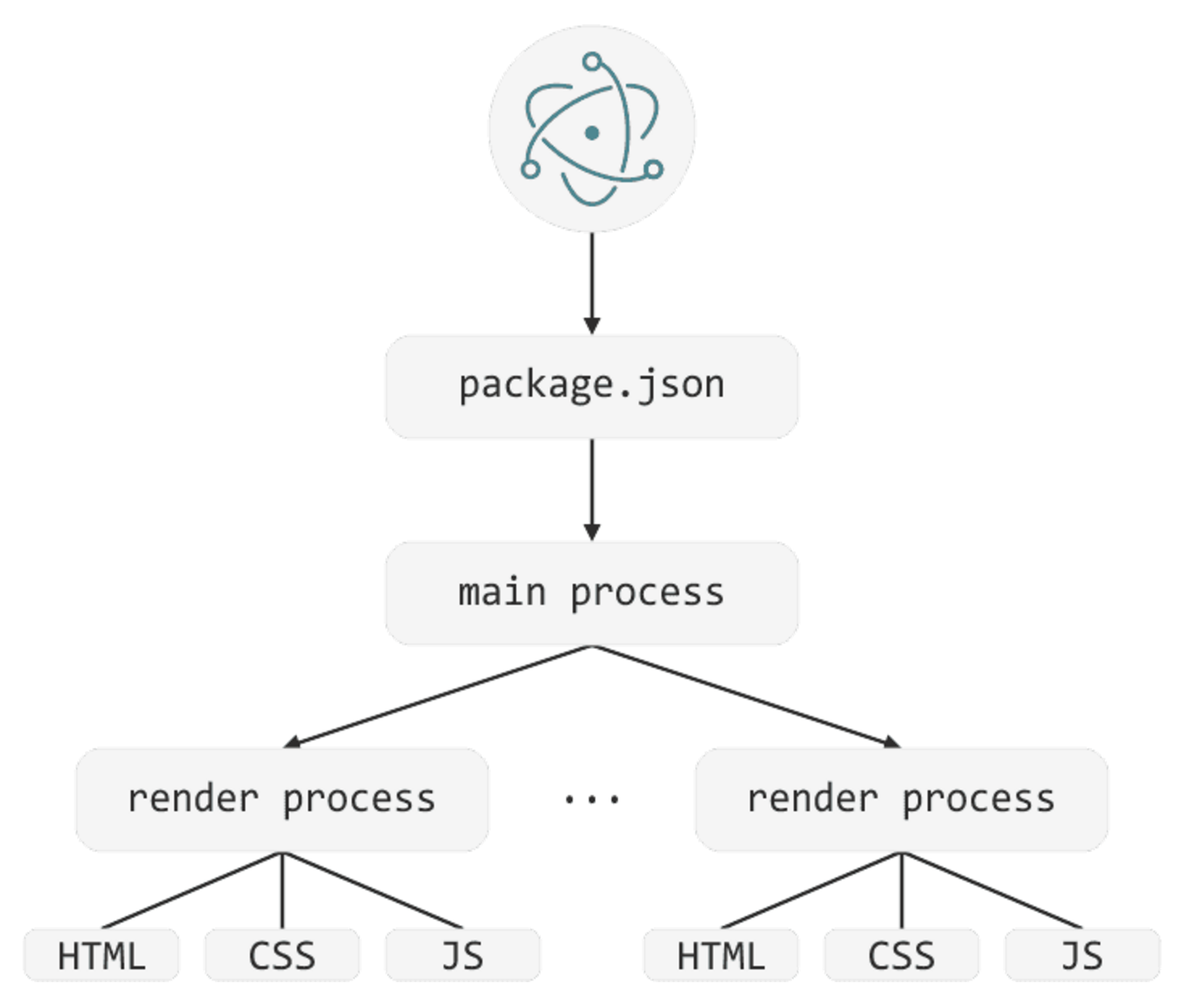


Рисунок 1- Архитектура Electron - приложения

2.3 Обобщенное описание алгоритма работы системы

После анализа текущего состояния предметной области была разработана обобщенная схема функционирования веб-ориентированной системы с голосовым интерфейсом (см. Рисунок 1).

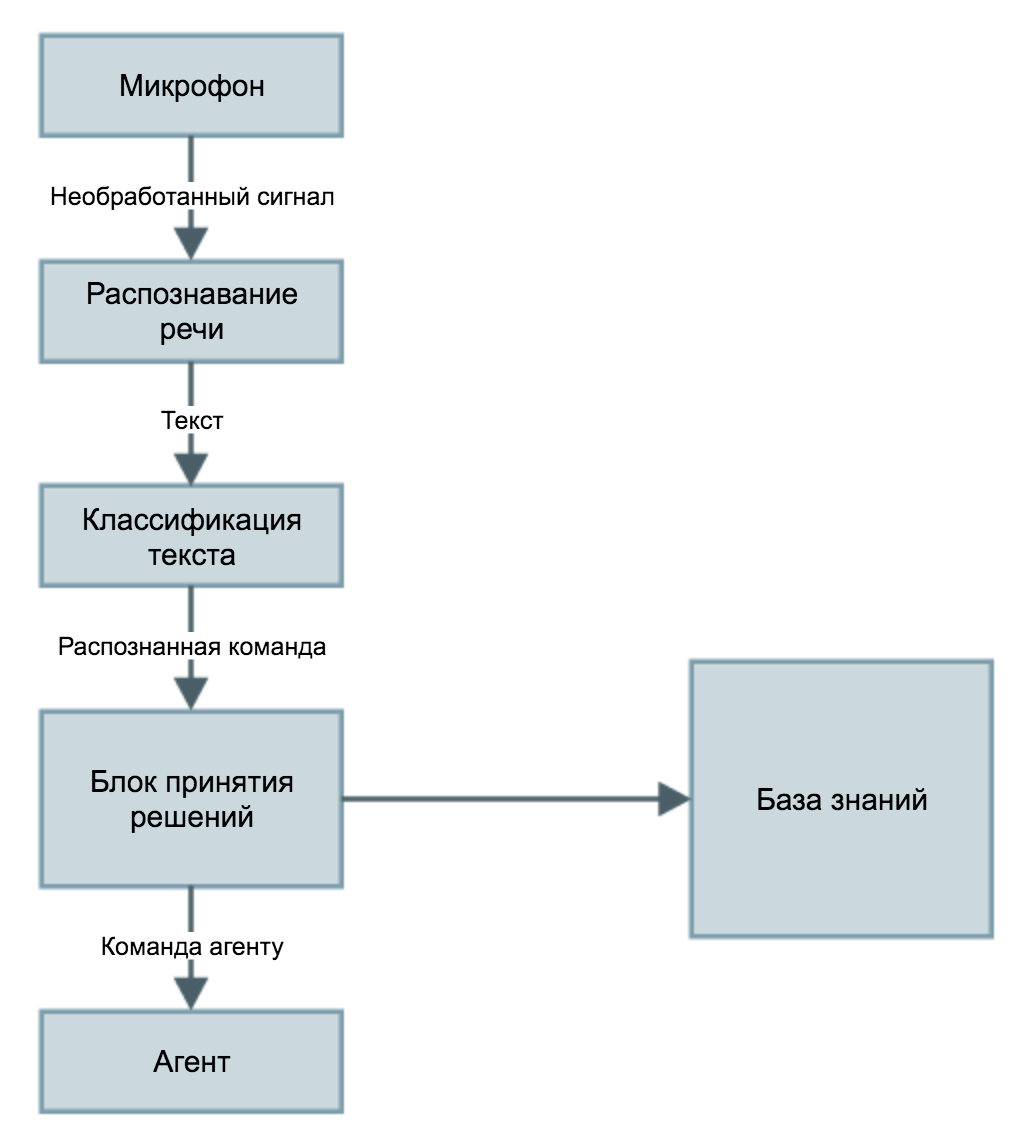


Рисунок 1 – Обобщенная схема функционирования системы с голосовым интерфейсом

Рисунок 1 наглядно иллюстрирует основные модули и последовательности их взаимодействия в процессе работы системы. Таким образом разработанная система с голосовым интерфейсом представлена в виде совокупности следующих подсистем:

* система синтеза речи
* система распознавания речи
* система классификации текста
* система извлечения ключевых слов из текста
* база знаний
* система принятия решений
* агент
* система плагинов

Теперь, когда рассмотрены компоненты фреймворка AngularJS, необходимо рассмотреть обобщенную архитектуру AngularJS – приложения.



Рисунок 2 – Обобщенная архитектура AngularJS – приложения

Таким образом алгоритм функционирования AngularJS-приложения будет следующим:

1. После загрузки фреймворка, AngularJS регистрирует все модули и их контроллеры, директивы, сервисы и фильтры.
2. После регистрации всех компонент и срабатывания события DOMContentLoaded, сигнализирующего о том, что все JavaScript и HTML файлы были загружены, наступает первая фаза жизненного цикла AngularJS-приложения – фаза загрузки. AngularJS инициализирует свои собственные необходимые компоненты, а потом и модуль, который указан в специальной директиве ng-app, вместе со всеми его зависимостями. Здесь же происходит конфигурация приложения. С помощью декораторов, происходит конфигурация начальных состояний сервисов, Настриваются параметры роутинга приложения: для каждого URL опционально указывается контроллер, шаблон, файлы стилей и принимаемые данные.
3. Вторая фаза – фаза HTML компиляции. Изначально загрузка веб -страницы предполагает загрузку в браузер статической формы DOM-a. Во время фазы компиляции, статический DOM заменяется на динамический DOM, который является представлением AngularJS. Фреймворк компилирует DOM только раз, а потом при необходимости ссылается на скомпилированный шаблон, что делает его намного эффективнее чем традиционные методы привязки данных.
4. После того как все обработчики конфигурационной стадии были выполнены, наступает третья фаза – фаза времени выполнения, которая продолжается до тех пор пока пользователь не перезагрузит или не уйдет с веб-страницы. В этот момент все изменения в модели отражаются в представлении, и любые изменения в представлении напрямую обновляют модель.

2.4 Описание программных модулей

2.4.1 Система распознавания речи

Как уже упоминалось, в качестве системы распознавания речи была выбрана технология HTML5 Speech Recognition. Эта технология является частью спецификации Web Speech API Specification и предоставляет удобный интерфейс для распознавания речи в браузерах на основе webkit.

Она позволяет распознавать как отдельные фразы так и непрерывную речь. В первом случае, распознавание заканчивается сразу как только пользователь заканчивает говорить, во втором случае – когда будет вызван метод stop().

Таким образом во втором случае мы можем позволить пользователям заканчивать распознавание по нажатию на кнопку, к которой будет прикреплен обработчик, вызывающий метод stop(). Результатом распознавания является объект, который содержит список гипотез вместе со всей прилежащей информацией (см рисунок 4).

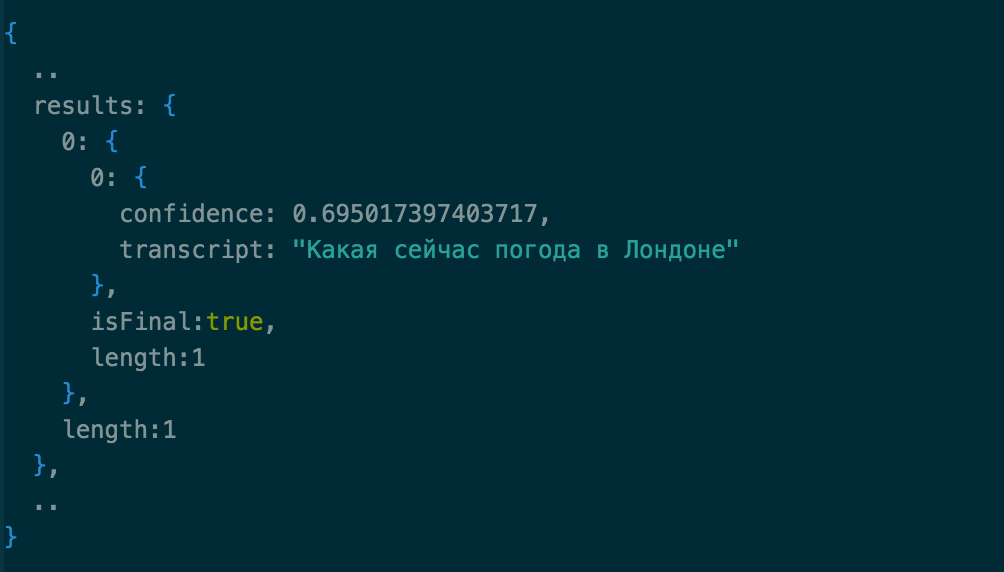


Рисунок 3 – Пример результата работы системы распознавания речи

Еще одной важной особенностью Web Speech API является то, что она позволяет указать объект грамматики – набор правил для определения языка. Использование объекта грамматики обычно приводит к улучшению результатов за счет ограничения возможностей языка.

### Рассмотрим методы и свойства данной технологии.

Инстанциирование распознавателя речи происходит следующим образом:

var recognizer = new SpeechRecognition();

Этот объект предоставляет следующие методы:

* onstart: Устанавливает функцию обратного вызова, которая вызывается когда распознаватель речи начинает слушать аудио с целью распознать.
* onresult: Устанавливает функцию обратного вызова, которая вызывается когда распознаватель речи возвращает результат
* onerror: Устанавливает функцию обратного вызова, которая вызывается когда возникает ошибка распознавания
* onend: Устанавливает функцию обратного вызова, которая вызывается когда распознаватель речи заканчивает работу.

В дополнение к этим методам, мы можем настроить объект распознавания речи, используя следующие свойства:

* continuous: Устанавливает тип распознавания (одиночных фраз или непрерывную речь). Если значение установлено в true то используется непрерывное распознавание, иначе процесс заканчивается как только пользователь заканчивает говорить. По умолчанию он установлен в false.
* lang: Определяет язык распознавания. По умолчанию он соответствует языку браузера.
* interimResults: Определяет нужны ли промежуточные результаты. Если значение установлено в true, будет предоставляться доступ к промежуточным результатам, что можно использовать для обеспечения ответа еще до того как пользователь закончил говорить. Если значение установлено в false, результаты будут получены только после того как закончится распознавание. По умолчанию он установлен в false.

Как параметр к обработчику события result, мы получаем объект типа SpeechRecognitionEvent, который содержит следующие данные:

* results[i]: Массив, содержие результаты распознавания. Каждый элемент массива соответствует распознанному слову.
* resultIndex: Текущий результирующий индекс распознавания.
* results[i].isFinal: Логическое значение, показывающее является ли текущий результат промежуточным или конечным.
* results[i][j]: двумерный массив, содержащий альтернативые распознанные слова. Первый элемент является наиболее вероятным распознанным словом.
* results[i][j].transcript: Текстовое представление распознанных слов.
* results[i][j].confidence: Вероятность того, что результат является верным. Число от 0 до 1.

Подсистема распознавания речи реализована в виде Angular-сервиса, что позволяет получать доступ к данной технологии из любой точки приложения посредством DI.

2.4.2 Система классификации текста

Джарвису необходимо знать вероятности того, что классифицируемый документ относится к некоторому классу и в случае если она ниже определенного значения предлагать пользователю самому соотнести документ с некоторым классом.

Так же необходимо было, чтобы текст внутри классификатора обрабатывался правильно, с учетом особенностей каждого языка, в частности русского. В конечном счете, после апробирования более 15 моделей, реализованных под платформу Node.js, в ходе кросс-проверки на небольшой выборке данных, лучший результат показала BrainJS - имплементация на языке JavasScript простой нейронной сети типа многослойный перцептрон с обратым распространением ошибки, которая продемонстрировала порядка 91% точности. В качестве метрик правильности классификации текстов были выбраны точность (precision) и полнота (recall). Точность в пределах класса – это доля текстов, действительно принадлежащих данному классу, относительно всех текстов, причисленных классификатором к этому классу. Полнота системы – отношение числа найденных классификатором текстов, принадлежащих классу, к числу всех текстов этого класса в тестовой коллекции. Для того, чтобы использовать нейронную сеть в качестве классификатора, необходимо было решить несколько проблем. Первая проблема заключалась в том, нейронные сети работает только с данными, представленными числами от 0 до 1, следовательно слова текстовых документов необходимо было преобразовать в цифры, а текстовые документы – в векторы из этих чисел. Это было реализовано за счет формирования словаря выборки, содержащего все слова из текстов обучающей выборки. Таким образом, текстовый документ можно было представить в виде бинарного вектора длинной, равной количеству слов в словаре обучаемой выборки, элементы которого показывали присутствует ли соответствующее слово из словаря в данном текстовом документе (1) или нет (0).

Проиллюстрируем примером: даны два коротких документа; они являются примером простейшей обучающей выборки.

1. ‘*Jane likes coffee and tea’*
2. *‘Jane also likes cookies’*

На данном этапе может быть сформирован словарь выборки; он будет содержать все слова из текстов обучающей выборки. Тогда бинарные вектора будут выглядеть следующим образом:

Таблица 2 – Представление текстовых документов в виде бинарных векторов

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | *Jane* | *likes* | *coffee* | *and* | *tea* | *also* | *cookies* |
| 1ый текст | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 2ой текст | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

Следующей проблемой была подача на вход нейронной сети цельного предложения. Дело в том, что обычные нейронные сети прямого распространения должны иметь входные данные фиксированной длины (см. рисунок 4).

А

Рисунок 4 – Нейронная сеть прямого распространения

Так как предложения содержат разное количество слов, необходимо было привести их к единой длине. Самый простой способ - сложить все вектора, получив таким образом результирующий вектор предложения. Но так как важной особенностью разрабатываемой системы является предоставление возможности пользователям динамически обучать ее, то необходимо расширить этот результирующий вектор на некоторую константу. Таким образом на стадии обучения нейронной сети, после представления предложений обучаемой выборки в виде векторов чисел, они складывались, и к результирующему вектору добавлялось 1000 пустых элементов, что, по мнению автора более чем достаточно для пользователя в рамках одной сессии работы с разработанной системой. Все описанные решения были реализованы с помощью стандартных средств языка JavaScript и библиотеки Natural, реализованной под платформу Node.js.

Рассмотрим пример предварительной обработки текста. Пусть в обучаемой выборке содержится два предложения: «Ну и какой был в этом смысл?» и «Какая же самая высокая гора в мире?». Тогда перед тем как попасть на вход нейронной сети они пройдут следующие этапы обработки:

1. Токенизация – разбиение текста на слова

tokenize('Ну и какой был в этом смысл?')

// -> [ 'Ну', 'и', 'какой', 'был', 'в', 'этом', 'смысл']

tokenize('Какая же самая высокая гора в мире?')

// -> [ 'Какая', 'же', 'самая', 'высокая', 'гора', 'в', 'мире']

1. Стемминг и удаление стоп слов

stem([ 'Ну', 'и', 'какой', 'был', 'в', 'этом', 'смысл' ])

// -> [ 'Как', 'эт', 'смысл' ]

stem([ 'Какая', 'же', 'самая', 'высокая', 'гора', 'в', 'мире' ])

// -> [ 'Как', 'самый', 'высок', 'гор', 'мир' ]

1. Объединение без пересечения (получаем словарь выборки)

join([ 'Как', 'эт', 'смысл' ], [ 'Как', 'самый', 'высок', 'гор', 'мир' ])

// -> [ 'Как', 'эт', 'смысл', 'самый', 'высок', 'гор', 'мир' ] + 1000 пустых элементов

1. Преобразование предложений в вектор цифр

// [ 'Как', 'эт', 'смысл', 'самый', 'высок', 'гор', 'мир' ]

features([ 'Как', 'эт', 'смысл' ])

// -> [ 1, 1, 1, 0, 0, 0, 0 ] + 1000 нулевых элементов

features([ 'самый', 'высок', 'гор', 'мир' ])

// -> [ 0, 0, 0, 1, 1, 1, 1 ] + 1000 нулевых элементов

1. Обучение

var net = new brain.NeuralNetwork();

// [ 'Как', 'эт', 'смысл', 'самый', 'высок', 'гор', 'мир' ]

net.train([

// [ 'Как', 'эт', 'смысл' ]

{ input: [ 1, 1, 1, 0, 0, 0, 0 ], output: { 'сорок два': 1 } },

// [ 'Как', 'самый', 'высок', 'гор', 'мир' ]

{ input: [ 1, 0, 0, 1, 1, 1, 1 ], output: { 'Эверест': 1 } }

]);

Пусть пользователь сказал «Можешь сказать какая самая высокая гора в мире?». Тогда классификация текста будет выглядеть следующим образом:

1. Токенизация и стемминг

tokenizeAndStem('Можешь сказать какая самая высокая гора в мире')

// -> [ 'можеш', 'сказ', 'как', 'самый', 'высок', 'гор', 'мир' ]

1. Преобразование предложений в вектор цифр

features([ 'можеш', 'сказ', 'как', 'самый', 'высок', 'гор', 'мир' ])

// -> [ 1, 1, 1, 0, 0, 0, 0 ] + 1000 нулевых элементов

1. Классификация

net.run([ 1, 1, 1, 0, 0, 0, 0 ]); + 1000 нулевых элементов

// -> { Эверест: 0.932615111520198, "сорок два": 0.0695108432441562 }

По-умолчанию, классификатор игнорирует русские слова, хотя поддержка русского языка в проекте есть. Чтобы заставить классификатор понимать русский язык, нужно проинициализировать классификатор, передав в него stemmer для русского языка, заменив таким образом используемый по-умолчанию английский stemmer.

Результатом классификации является массив вероятностей того, что классифицируемый текст относится к конкретному классу(см рисунок 5).



Рисунок 5 – Результат классификации предложения «Какая погода в Лондоне».

Обучение нейронной сети является ресурсозатратным процессом, однако данная библиотека позволяет сохранять результаты обучения и соответственно загружать, что позволяет обучать нейронную сеть в офлайн режиме.

Подсистема классификации текстовых документов реализована в виде Angular-сервиса по целому ряду причин. , который является сигнлтон-объектом, так как было бы не разумно (не целесообразно) каждый раз инстанциировать новую нейронную сеть для классификации нового текста.

2.1.3 Подсистема извлечения ключевых параметров текста

Ключевые параметры текста – это слова, которые необходимы агенту для выполнения своих функций. Например если из предложения «Скажи мне погоду в Харькове» не выделить слово «Харьков» и не передать его агенту, то агент, отвечающий за предоставление информации о погоде, не сможет ее предоставить. Таким образом, у элементов обучаемой выборки, помимо текста и соответствующего класса, есть еще один параметр – метка, которая определяет расположение ключевых(ого) слов(а) в тексте.

Сравнительно с системой классификации текста, механизм извлечения ключевых параметров довольно примитивен. Исходя из предположения, что классифицируемое предложение, с определенной долей вероятности, является схожим с уже изученным предложением, то механизм представляет собой сравнение слов рядом с меткой. Например, если классифицируемое предложение «Не знаешь, холодно ли в Лондоне» с меткой «Лондон» будет сопоставлено с предложением из обучаемой выборки «Скажи пожалуйста, холодно ли в Севастополе?», то механизм пометит слова после «холодно ли» как местоположение. Разумеется, если предложение будет сопоставлено с несколькими предложениями из обучаемой выборки, то алгоритм пометит то слово, возле которого будет найдено больше всего совпадений. Хоть это и не самый идеальный алгоритм, он работает достаточно хорошо благодаря точности классификации естественного языка.

2.1.4 База знаний

Как уже отмечалось ранее, основное отличие Джарвиса от остальных подобных систем заключается в том, что он предоставляет пользователям возможность самостоятельно обучать ее. Для этого Джарвису необходимо где-то хранить тренировочные данные. Так как формат данных для обучения нейронной сети максимально прост: это текст и соответствующие тексту класс и метка, и никаких дополнительных данных и связей между ними нет и не предвидится, то самым разумным решением, по мнению автора, будет использовать легковесную базу данных DiskDB, основанную на файловой системе и позволяющую хранить данные в json-формате. База знаний реализована в виде сервиса, который предоставляет стандартные методы базы данных: сохранение, изменение, удаление и загрузка данных.

Примерчик методов?

2.1.5 Блок принятия решений и агенты

В виду того, что для реализации блока принятия решений (БПР) и агентов используется одна концепция, целесообразно описать их в едином разделе.

Агент – это отдельный модуль, который выполняет конкретные действия по голосовой команде. В разработанном приложении каждый агент представляет собой отдельное состояние приложения. Для организации и управления состояниями была реализована динамическая маршрутизация с помощью технологии UI Router. UI Router – это библиотека для AngularJS. Она предлагает отличный от стандартного AngularJS маршрутизатора ngRoute подход. В то время как ngRoute организован вокруг URL-адресов, UI Router организован вокруг состояний, которые опционально могут включать в себя и URL-адреса, так же как и другое поведение. Таким образом, можно управлять работой приложения - настраивать какие контролеры, шаблоны и стили будут использоваться для каждого состояния -независимо от URL. Это преимущество имеет особенное значение при реализации десктопной версии, так как в Electron отсутствуют URL-адреса. Еще одним ключевым преимуществом данной библиотеки является возможность создавать вложенные состояния. Это позволяет определить одно состояние для реализации общей функциональности, которую смогут использовать вложенные состояния. Точно так же шаблоны вложенных состояний можно встраивать в шаблон внешнего состояния. Таким образом UI Router предоставляет огромные возможности по управлению интерфейсом приложения.

С учетом изложенного, структура состояний приложения имеет следующий вид (см. рисунок 6).

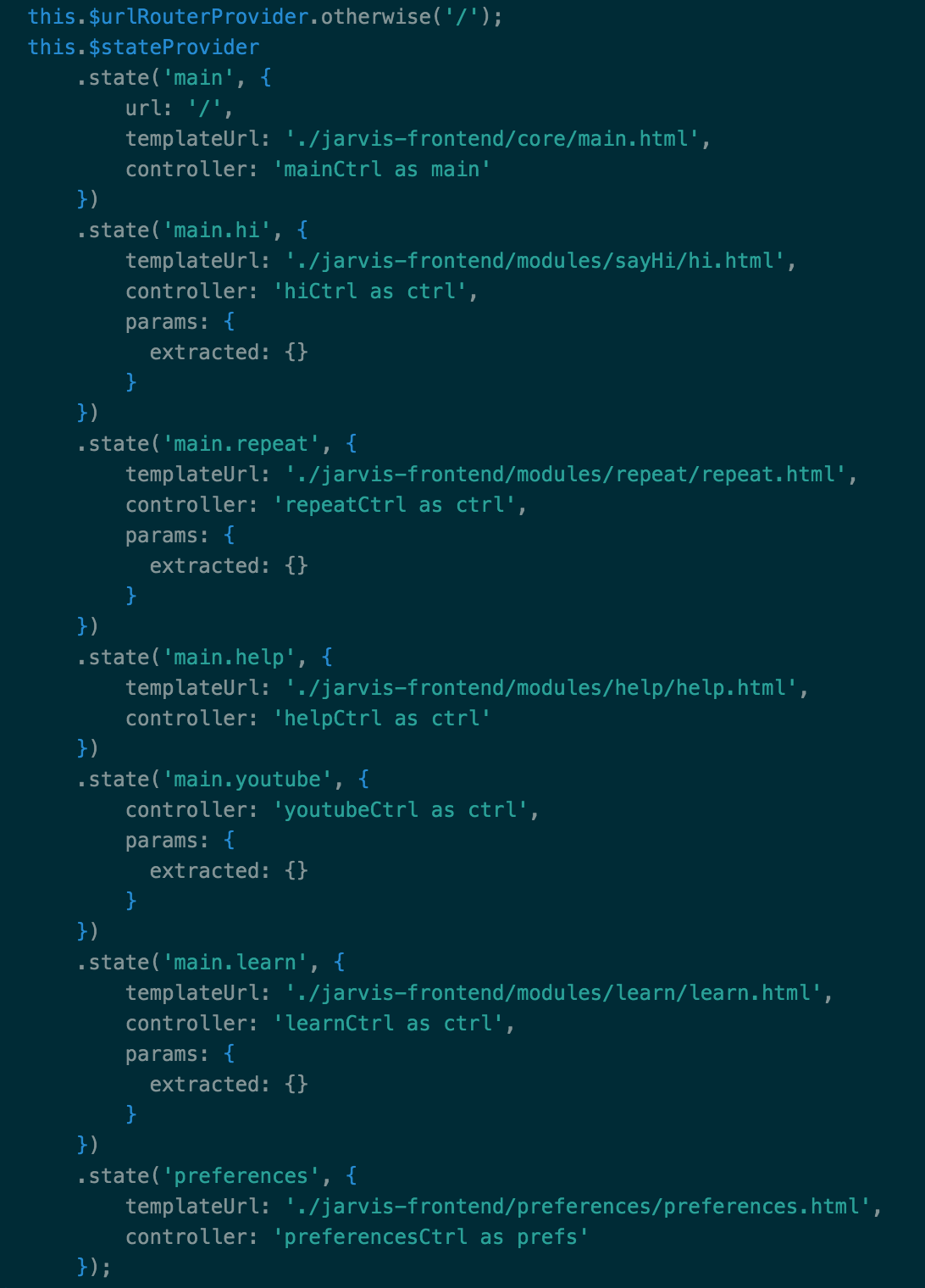


Рисунок 6 – Пример организации состояний приложения

Как видно из рисунка 6, основным состоянием является состояние «main». Контроллер основного состояния – это и есть БПР. Таким образом БПР функционирует всегда, независимо от того, в каком состоянии находится приложение (исключение – состояние preferences). Получая через DI доступ ко всем сервисам, он контролирует все области работы приложения, от запуска распознавателя речи до передачи управления агенту. В виду того, что некоторые операции выполняются асинхронно (например распознавание речи и классификация), был использован шаблон проектирования «Обозреватель», реализованный в виде статического класса. Он предоставляет два основных метода: регистрацию обработчика на конкретное событие и публикацию определенного события (см. Рисунок 7).



Рисунок 7 – Реализация шаблона проектирования «Обозреватель»

Таким образом, при запуске программы, БПР регистрирует обработчики для таких событий как «начало распознавания речи», «конец распознавания речи», «готов результат распознавания речи», «готов результат классификации текста» c помощью метода when(eventName). Эти события генерируются соответствующими сервисами с помощью метода publish(event).

В основное состояние «main» вложены такие состояния как «hi», «youtube», «help» и «learn». Вложенные состояния – это агенты, которые выполняют специализированные функции, например приветствие пользователя, отображение справки Джарвиса или выполнение поиска видео на YouTube по заданной пользователем теме.

После того как подсистема классификации текста распознала команду, она передает результаты классификации БПР, который проверяет их достоверность. Если достоверность превышает некоторое пороговое значение, то БПР осуществляет поиск соответствующего агента и передает ему управление, если такой существует.

try {

this.$state.go(`main.${actionType}`, {

extracted

});

} catch (e) {

this.$state.go(`main.learn`); //передача управления агенту обучения

}

В случае если такого агента не существует или если достоверность классификации меньше порогового значения, БПР предлагает пользователю самому соотнести текстовый документ с некоторым классом из предложенных в подсказке, которая строится так же на результатах классификации. Кроме того, пользователь может указать новый класс, которого не было в обучаемой выборке. Другими словами, БПР предлагает обучить систему. Как видно из рисунка 6,обучение Джарвиса так же реализовано в виде агента. Это позволяет модулю обучения получать доступ к результатам классификации для организации подсказки, так как контроллеры вложенных состояний имеют доступ к атрибутам и методам класса контроллера внешнего состояния. За пороговое значение была взята величина 0.3 т.е. 30% достоверности классификации, что является хорошим значением по мнению автора.

2.1.6 Система плагинов

Система плагинов – одна из ключевых особенностей разработанного приложения. Разработанное приложение предоставляет простое, удобное и вместе с тем, позволяющее использовать все возможности и преимущества фреймворка AngularJS и СУБД MongoDB, API для создания собственных голосовых функций (плагинов). Каждый плагин Джарвиса представляет собой npm-модуль. Данное решение избавляет от необходимости реализовывать собственное удаленное хранилище модулей и систему для их управления. Все эти возможности уже предоставляет пакетный менеджер npm.

Чтобы создать новый плагин, необходимо создать папку и поместить в него файл package.json, который должен иметь следующую структуру (см. Рисунок 8).



Рисунок 8 – Пример структуры файла package.json

Рассмотрим основные параметры:

* name – имя плагина
* version – версия плагина
* main – путь, относительно созданной папки, к JavaScript файлу, в котором реализовывается основная функциональность плагина
* jarvis – обязательный параметр, в котором описывается :
  + stateName – строка, которая будет использоваться в качестве имени состояния приложения, в котором выполняется плагин
  + server – относительный путь к JavaScript файлу, который экспортирует данные для обучения нейронной сети. Это должен быть массив объектов, содержащих следующие поля: текст, по которому будет выполняться плагин и соответствующие тексту класс и метка. В данном случае класс должен совпадать с параметром stateName.
  + template – путь к HTML – файлу, для реализации пользовательского интерфейса плагина. Этот параметр является опциональным.
  + styles – путь к CSS – файлу, который подключается к HTML – файлу. Этот параметр является опциональным.
  + directives – относительный путь к AngularJS директивам
  + filters – относительный путь к AngularJS фильтрам
* keywords – массив ключевых слов плагина, которые используются пакетным менеджером npm для поиска модулей. Важно чтобы он содержал ключевое-слово «jarvis-plugin», чтобы гарантировать, что он будет отображаться в каталоге плагинов Джарвиса.
* description – используется для описание модуля в пакетном менеджере npm, оно будет так же отображаться в каталоге плагинов Джарвиса.
* author – cоответственно автор плагина

Рассмотрим как происходит динамическое подключение плагинов к Джарвису. На первой фазе жизненного цикла AngularJS-приложения в блоке конфигурации, средствами платформы Node.js, Джарвис получает список абсолютных путей всех установленных npm-модулей. Затем для каждого модуля из списка выполняется функция загрузки loadModule(modulePath). Эта функция выполняет следующие действия:

* Загружает файл package.json и проверяет, есть ли у него поле «jarvis». Если нет, то выбрасывается исключение «Этот модуль не является Джарвис-плагином». Иначе получает из него пути ко всем компонентам (шаблонам, стилям, директивам, филтрам).

Так как основной файл плагина, реализующий всю функциональность, подключается к angular-модулю Джарвиса в виде контроллера, то разработчики могут использовать все cервисы, встроенные в ядро фреймворка AngularJS, а так же собственные сервисы Джарвиса, c помощью DI. Например, если плагину требуется делать HTTP запросы или сохранять данные в БД, достаточно просто указать названия соответствующих сервисов в качестве параметров конструктора, а AngularJS позаботится об остальном :

class Weather {

constructor($http, mysqlService) {

this.$http = $http;

this.mysqlService = mysqlService

}

…

}

2.2 Характеристики разработанного программного продукта

Таким образом разрабатываемая система с голосовым интерфейсом «Джарвис» будет иметь ряд преимуществ:

* Кроссплатформенность. Прежде всего, она не будет привязана к конкретным аппаратным устройствам или операционным системам, а может функционировать как на мобильных устройствах в виде веб-приложения, так и на устройствах под управлением Windows, Linux или OS X в виде desktop-приложения.
* Офлайн. Благодаря стеку используемых технологий система не будет требовать доступ к интернету для выполнения ваших команд.
* Самообучаемость. То есть ассистент буквально будет уметь обучаться новым командам от своего владельца. Если система не поняла чего-то, то она всегда спросит «Что с этим делать?», а пользователь сможет направить ее, сказав какое из действий, приведенных в подсказке, выполнить.
* Система плагинов. Система будет предоставлять простое, удобное и вместе с тем, позволяющее использовать все возможности и преимущества фреймворка AngularJS и базы данных SQL, API для создания собственных голосовых функций (плагинов). Причём их возможности становятся действительно безграничными. Реализована система управления плагинами: система предоставляет интерфейс для доступа к списку всех доступных плагинов и автоматизирует процесс установки / обновления / удаления их по клику без каких-либо дополнительных действий.
* Русский язык. На сегодняшний день все рассмотренные системы в первую очередь ориентированы на англоязычное население, а некоторые и вовсе поддерживают только его. В связи с этим новая функциональность появляется с существенной задержкой для русскоговорящих пользователей. Таким образом лучшей альтернативой может стать стороннее решение, а именно то, что изначально создавалось для русского языка.
* Мультиязычность. Несмотря на изначальную направленность разработанной системы на русскоговорящих пользователей, алгоритм функционирования системы реализован таким образом, что система будет так же хорошо выполнять собственные голосовые функции иноязычных пользователей.

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

ВЫВОДЫ

Когда голосовой помощник в полной мере станет понимать естественный язык?

Для этого, конечно, потребуется время. Развитие голосовых помощников, в этом плане, станет постоянным процессом, так как любой язык, являясь живой материей, приносит все новые слова и парадигмы, которым нужно время для интеграции. Необходимо также, чтобы технологии распознавания акцентов работали как надо.

Я задумался над тем, чтобы сделать для себя RSS-клиент с обучаемым фильтром статей на NodeJS. В принципе, под ноду есть готовые RSS ридеры, есть готовые нейронные сети с классификаторами, так что написать какой-то прототип мне показалось не особенно сложной задачей.

В конечном счете, после апробирования реализаций более 15 моделей под Node.js, включая многослойные нейронные сети на примере BrainJS и Node-fann, в результате кросс-проверки на небольшой выборке данных, лучший результат показала библиотека Natural, которая использует обычный наивный байесовский классификатор с корнями слов и POS-тегами, дав порядка 85%.

<http://speech-soft.ru/info/raspoznavanie-golosa> (продолжение)

Нерешенные задачи и взгляд в будущее.  
Три основных барьера стоят на пути развития систем распознавание речи:

1. большие объемы словарей  
2. шаблоны непрерывной речи  
3. различные акценты и произношения

Хотя в этой области и достигнуты существенные успехи, тем не менее, системы распознавания еще весьма далеки по своим возможностям от человеческих.

#### Бум приложений по распознаванию речи указывает на то, что время распознавания речи пришло, и мы можем ожидать огромного количества их и в будущем. Эти приложения не только позволят контролировать компьютер с помощью голоса или конвертировать голос в текст - они будут также способны различать разные языки, позволят выбирать голос помощника из различных вариантов. Вполне вероятно, что технология распознавания речи перейдет и на другие типы девайсов. Нетрудно представить, как в будущем мы будем управлять кофеварками, разговаривать с принтерами и говорить освещению, чтобы оно выключалось.

СПИСОК ЛИТЕРАТУРЫ

[1] K. H. Davis, R. Biddulph and S. Balashek, “Automatic recognition of spoken digits”, J. Acoust. Soc. Am., 24, pp. 637-642 (1952).

[2] J. Suzuki and K. Nakata, “Recognition of Japanese vowels – preliminary to the recognition of speech”, J. Radio Res. Lab, 37, pp. 193-212 (1961).

[11] Sebastiani, F.: Machine learning in automated text categorization, ACM Computing Surveys, vol. 34, pp. 1-47, 2002

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

UI – пользовательский интерфейс (user interface).

JS – JavaScript.

ASR – автоматическое распознавание речи (Automatic Speech Recognition).

HMM – Скрытая марковская модель (Hidden Markov Model).

Плагин – независимо компилируемый программный модуль, динамически подключаемый к основной программе.

API - интерфейс программирования приложений.

DOM – Document Object Model.

MVVM.  
MVC.

DI.

БПР – Блок принятия решений