Сажин В.Г. Идентификация человека по голосу, Выпускная квалификационная работа бакалавра по специальности: «Вычислительные машины, комплексы, системы и сети», Нижегородский государственный технический университет им. Р. Е. Алексеева, кафедра: «Вычислительные системы и технологии», Нижний Новгород, 2016. Руководитель: доцент кафедры «Вычислительный системы и технологии» Гай В. Е.

Работа посвящена разработке и прототипа системы идентификации человека по голосу. Описывается структура системы, приводится выбор методов для реализации модулей, входящих в систему.

В результате тестирования установлено, что разработанная система корректно решает поставленную задачу идентификации человека по голосу.

Объём работы 42 страницы. Использовано источников – 9, рисунков – 7.

Содержание

Введение

Сейчас во всех сферах нашей жизни распространены компьютеры. И в связи с этим, в наше время возникает вопрос защиты информации. Стандартные защитные системы уже не могут справиться с идентификацией личности, поэтому вводятся дополнительные меры. В качестве этих мер могут выступать проверки по биометрическим данным человека.

Биометрия определяется как комплекс приемов для распознавания, которые основываются на исключительных физиологических характеристиках человека, а также поведенческих. Сейчас в качестве биометрических характеристик можно выбрать радужную оболочку глаз, отпечатки пальцев, почерк, рисунок сетчатки, узор вен, геометрию лица или печать на клавиатуре. Любая из характеристик позволит выделить человека из множества. Использование нескольких из них положительно скажется на точности идентификации.

Распознавание базируется на том, что с человека берутся данные (фото сетчатки или радужки глаза, образец голоса и т.д.) и дальше они сравниваются с тем, что хранится в базах с данными пользователей.

Польза биометрических систем весьма ощутима: указанные выше человеческие характеристики уникальны и их проблематично подделать, потому, что трудно выдать свои отпечатки пальцев за чужие или изменить рисунок сетчатки глаза на иной. В отличие от привычных идентификаторов, биометрические характеристики не могут каким-либо образом быть утеряны. Некоторые люди могут имитировать голоса, но, этот навык редко встретишь в обыденной жизни. Чтобы обезопаситься от подобных случаев, достаточно лишь ввести проверку нескольких биометрических характеристик.

Целью данной работы является создание программной реализации идентификации человека по голосу.

1 Техническое задание

1.1 Назначение разработки и область применения

Разрабатываемая система предназначена для идентификации человека по голосу. Для работы данной системы необходима база голосов дикторов для определения принадлежности к одному из них.

Области применения разрабатываемой системы:

– фоноскопическая экспертиза в криминалистике;

– системы учета времени на работе, в образовательных учреждениях и т.п.;

– контроль доступа на закрытые объекты и контроль доступа к информации;

– идентификация/аутентификация граждан для исключения фальсификаций данных на голосовании;

1.2 Технические требования

Рассмотрим требования, предъявляемые разрабатываемой системой к ЭВМ:

– операционная система Microsoft Windows не ниже версии XP;

– требования к аппаратному обеспечению определяются операционной системой;

– клавиатура и мышь;

– микрофон и динамики.

2 Анализ технического задания.

2.1 Выбор операционной системы

Перед началом разработки необходимо в первую очередь выбрать операционную систему. Сейчас, в связи с огромным распространением мобильных устройств, лидирующие позиции занимают мобильные операционные системы Android и iOS (38,5% от общего количества операционных систем). Среди десктопных, на данный момент наибольшее распространение получили операционные системы Windows, Linux и Mac OS. У каждой из них присутствуют свои преимущества и недостатки.

Рассмотрим эти операционные системы более подробно:

– Windows – это одна из самых распространенных операционных систем, которая была разработана компанией Microsoft. Эта система получила большое распространение благодаря интерфейсу, дружественному пользователю. Она имеет разные версии, в зависимости от задач, которые на нее возлагаются: для предприятий (с расширенными возможностями по обеспечению безопасности, серверные решения) и для персональных компьютеров. Сейчас Windows основывается на Windows NT, а ранее они основывались на MS-DOS. Самая последняя версия Windows, которую Microsoft активно продвигают, это Windows 10. Её доля растет, но самой распространенной до сих пор остается Windows 7.

– Mac OS – это операционная система, разработанная компанией Apple. Она основана на системе Unix, и, соответственно, является частью семейства Unix систем. Сейчас Mac OS является второй операционной системой после Windows.

– Linux – общее название Unix-подобных операционных систем, основанных на ядре Unix. Linux имеет открытый исходный код, и каждый может настроить его под себя. Также Linux создаётся и распространяется как свободное и открытое программное обеспечение. Распространяется посредством готовых дистрибутивов, которые уже настроенные под определенные нужды. Примером могут быть дистрибутивы для тестирования систем на проникновение, которые содержат множество утилит для этого тестирования. Операционная система Linux не так популярна, по причине того, что подавляющее большинство развлекательных приложений пишутся под Windows, и обычные пользователи выбирают Windows в качестве операционной системы для своего компьютера. Но в последнее время популярность Linux систем возрастает.

Основные различия, преимущества и недостатки операционных систем Windows, Mac OS, Linux:

1) Защита

Вследствие широкой распространенности операционной системы Windows, большинство вредоносных программ пишется именно для этой системы. Для операционных систем семейства Unix вредоносные программы тоже пишутся, однако им сложнее нанести какой-либо ущерб, так как они могут сделать это путем эксплуатирования уязвимостей и получения root-привилегий. Но уязвимости оперативно устраняются, и вследствие того, что технический уровень пользователей Linux, обладающих правами администратора, высок, вредоносным программам не так просто получить root-привилегии.

2) Стабильность работы

По стабильности, на протяжении большого периода времени, операционная система Linux намного превосходила и Windows, и Mac OS. Однако после выхода Windows 7, разрыв сократился, но еще остается существенным, если главным параметром выбираемой операционной системы является стабильность. Стабильность операционной системы Mac OS находится на уровне Windows версии 7 и выше.

2) Необходимое оборудование

Для операционной системы Mac OS необходимо только оборудование от компании Apple. Совсем иная ситуация видна для операционных систем Windows и Linux. Для них подходит оборудование большого числа производителей, и при таком многообразии можно собрать ЭВМ под свои нужды.

4) Программное обеспечение

Самой большой библиотекой программного обеспечения обладает операционная система Windows. Однако это не значит, что под Linux и Mac OS отсутствует программное обеспечение. Почти для всего можно найти аналоги, которые могут быть даже мультиплатформенными. Например, офисный пакет Open Office.

5) Применение и разработка программного обеспечения

Основной платформой для разработки и применения программного обеспечения является операционная система Windows. Большинство программ для этой операционной системы пишутся на таких языках как Java, C#, C++ и Visual Basic. Среди них есть и мультиплатформенные языки, например, C++ и Java, что сделает доступным разработку сразу под все платформы. Также большинство пользователей используют операционную систему Windows, что позволяет охватить широкий круг аудитории.

Для разработки прототипа системы идентификации человека по голосу была выбрана операционная система Windows. Она соответствует необходимым требованиям для разработки данного проекта.

2.2 Выбор языка программирования

Одним из самых важных этапов разработки приложения является выбор языка программирования. Самыми распространенными для операционной системы Windows являются C++, C# и Java.

Рассмотрим их более подробно:

– C++ - это компилируемый, статически типизированный язык программирования. Он является и процедурным, как его предок C, и объектно-ориентированным. Также он позволяет применять обработку исключений, обобщенное программирование, абстрагирование данных. C++ имеет очень широкий спектр применения: от драйверов до операционных систем. Является наследником языка C и имеет с ним полную совместимость. C++ оказал влияние на становление таких языков программирования как C# и Java.

– C# - это объектно-ориентированный язык, разработанный компанией Microsoft для своей платформы Microsoft .NET Framework. C# унаследовал многие хорошие решения из других языков программирования. К примеру, множественное наследование интерфейсов. Также есть исключения слабых мест, которым является множественной наследование классов из языка C++. Синтаксис языка похож на своих предшественников. Он близок к C++ и Java. C# имеет перегрузку операторов, статическую типизацию и много других полезных средств (таких как анонимные функции полиморфизм, обобщенные методы и типы, исключения).

– Java - этот объектно-ориентированные язык программирования создавался для бытовой электроники. Причиной создания стало нежелание производителей электроники писать код для каждой архитектуры оборудования с нуля. Был создан язык программирования Java, программы которого транслируются в байт код, выполняемый на виртуальной машине (JVM). Виртуальная машина обрабатывала байт-код и передавала его в виде инструкций на оборудование и работала как интерпретатор, но гораздо быстрее. Теперь необходимо было лишь написать виртуальную машину под нужную архитектуру и выполнять на ней Java код.

Если выбирать любой из этих языков, то пришлось бы реализовывать алгоритмы для обработки звуковых файлов, работы с аудио данными вручную. Поэтому, решающим фактором выбора языка программирования стало наличие готовых решений по реализации многих алгоритмов. Таким языком был выбран R.

– R – это универсальный язык программирования, который может быть полезен при обработке большого количества данных, т.к. он является статистическим языком программирования. Также с помощью него можно работать с графикой. R продолжает развиваться, в значительной степени посредством добавления пакетов. На платформе R можно найти пакет для решения практически любой задачи. R поставляется в виде исходных кодов на Linux, Windows и Mac OS.

Решающим факторам выбора языка стало наличие библиотек (пакетов) для работы со звуковыми файлами, и для классификации.

Для работы со звуковыми файлами был выбран пакет tuneR. Основные функции, которые включает пакет tuneR, и которые необходимы для реализации проекта это:

– записывать звуковой сигнал

– считывать информацию из звуковых файлов

– из полученной аудио информации получать признаки

Для классификации звуковых файлов по полученным признакам был выбран пакет e1071.

2.3 Выбор среды разработки

Изначально работа с R ведется из командной строки. Консольный интерфейс предоставляет пользователю историю команд, выдает список возможных продолжений команд. Также по желанию пользователя может сохранять информацию и объекты между сессиями. Графические же оболочки никоим образом не ухудшают работу с языком R, а лишь расширяют базовые возможности путем добавления интегрированного редактора кода с подсветкой, отладчика, редактора массива данных. Исходя из этого, можно было обойтись стандартным консольным интерфейсом, но была выбрана среда с графической оболочкой RStudio, которая предоставляет необходимый функционал среды разработки с графическим интерфейсом.

RStudio – относительно новая оболочка для R, открытая, с доступным исходным кодом. Она может быть доступна в двух версиях: RStudio Desktop и RStudio Server. Первая позволяет работать локально как с обычным приложением, а вторая посредством доступа через командную строку или браузера на сервер, где и запущена RStudio. Эта среда доступна для операционных систем Wndows, Mac OS и Linux. RStudio имеет редактор кода с автоматической подсветкой, автодополнением, простейшими преобразования кода. Также имеется поддержка систем контроля версий, таких как Git или Subversion.

2.4 Обзор существующих решений

Задачу идентификации человека по голосу, независимо от выбранного пути решения ее, можно разбить на следующие основные этапы:

1) Извлечение признаков из аудио сигнала
2) Построение модели диктора, на основе полученных на предыдущем шаге признаков

Процесс определения человека по голосу, из представленных в системе образцов голосов, во всех методах состоит в поиске более подходящего по модели, на основании выбранных критериев.

Речевые сигналы - это звуковые колебания, которые распространяются в воздушной среде. Они характеризуются частотой (числом колебаний в секунду), интенсивностью (амплитудой колебаний) и длительностью. На протяжении всего речевого сигнала эти характеристики подвергаются изменениям, и фиксируются с помощью электронно-акустических приборов, таких как осциллограф, спектрограф. Затем, при помощи аналого-цифрового преобразователя, аналоговый речевой сигнал переводится в цифровой, который на ЭВМ представлен в виде WAV-файла, с которого уже происходит сбор речевых признаков.

2.4.1 Выбор системы признаков

Одной из самых основных частей данной работы является выбор системы признаков. От этого будет зависеть точность распознавания. Сейчас самыми распространенными методами извлечения признаков являются спектрально-временные и кепстральные признаки. Рассмотрим каждый из них более подробно.

2.4.1.1 Спектрально-временные признаки

Спектрально-временные признаки характеризуют сигнал речевого аппарата человека исходя из его физико-математической сущности. Эти признаки отражают как формы временного ряда, так и спектр голосовых импульсов вместе с особенностями фильтрующих функций речевых трактов у разных лиц. Речевой сигнал представляется в виде последовательности значений энергетических спектров, измеренных в разные моменты времени. Дальше речевой сигнал через быстрое преобразование Фурье подвергается спектральному анализу. Результатом будет набор амплитудного спектра i-той частоты, который и будет составлять набор признаков для данного звукового сигнала.

Полученные, путем преобразования Фурье, из спектральной информации параметры, которые и должны создавать ощущения звуков, теряют важную динамическую составляющую, и будут ассоциироваться с другими параметрами, которые уже к речи могут не относиться, и лишь добавляют сигналу лишние шумы, соответственно уменьшая точность распознавания. Но этот

подход может обеспечить точность распознавания свободной речи только в фиксированных условиях, но еще будет неустойчив к влиянию внешней среды и к самому голосу говорящего.

2.4.1.2 Кепстральные признаки

Кепстр показывает частоты повторяющихся колебаний спектра. Таким образом, он является результатом косинусного преобразования от логарифма амплитудного спектра сигнала, и с помощью него можно увидеть частоты колебаний спектра, которые с некоторой периодичностью повторяются. Этот термин имеет несколько источников образования. По мнению одних, этот термин связан с именем советского математика Колмогорова, а по мнению других, это слово образовано путем перестановки букв в слове «спектр».

Мел-шкала – это модель частотной чувствительности человеческого слуха. Эти два понятия основывают мел-частотные кепстральные коэффициенты (mel-frequency cepstral coefficients или MFCC). Путем обработки большого числа данных была произведена количественная оценка звука по высоте, впоследствии эту единицу высоты звука назвали мелом. Исследования установили, что изменение частоты в 2 раза в диапазоне низких и высоких частот воспринимается человеком по-разному. Реальное увеличение частоты в 2 раза совпадает с восприятием удвоения частоты до 1000 Гц, из-за чего мел-шкала близка к линейной до 1000 Гц. И мел-шкала становится логарифмической для частот выше 1000 Гц (рис. 1).

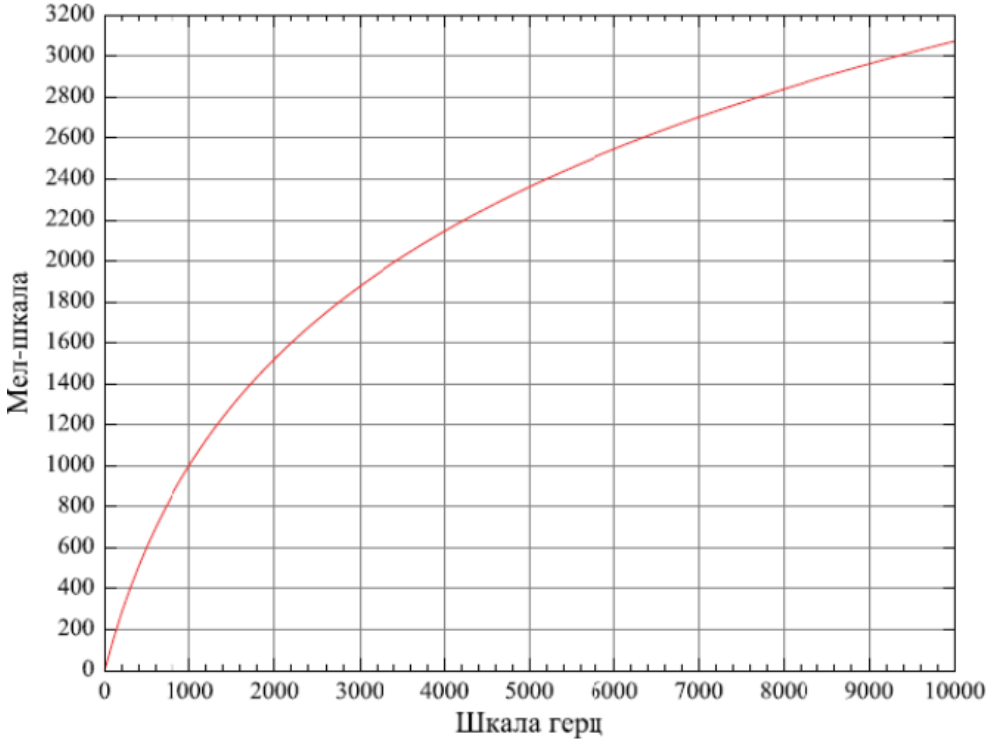

Рис. 1 Мел шкала

Перевод из шкалы мелов в шкалу герц и обратно происходит по следующим формулам:

$$F_{mel}(f_{hz}) = 1127{,}01048\ln\left(1 + \frac{f_{hz}}{700}\right) \quad (1)$$

$$F_{hz}(f_{mel}) = 700(e^{f_{mel}/1127{,}01048} - 1) \quad (2)$$

MFCC – это значения кепстра, которые распределены по мел-шкале с использованием банка фильтров.

Существует алгоритм нахождения MFCC:

1. Сигнал *s[t]* проходит предварительную обработку, а затем разбивается на *K* кадров по *N* отсчетов, которые пересекаются на половину длины:

$$s[t] \rightarrow S_n[t], n = 1, \dots, K$$

2. В каждом кадре получаются комплексные представления сигнала по частотам.
3. Получается спектральная плотность мощности получившегося сигнала:

$$P_n[k] = A_n[k]^2 \quad (3)$$

$$A_n[k] = \sqrt{ReX_n[k]^2 + ImX_n[k]^2} \quad (4)$$

4. Применение банка фильтров (рис. 2)

   а) Задается количество фильтров, а также начальная f_1 и конечная f_h частоты (f_h не должна превосходит половины частоты дискретизации);

   б) Далее эти частоты переводятся в мелы:

$$f_l^m = F_{mel}(f_l),$$

$$f_h^m = F_{mel}(f_h);$$

   в) На мел шкале отрезок $[f_l^m, f_h^m]$ разбирается на $P+1$ равных непересекающихся подотрезков $[f_l^m, f_{j+1}^m]$, $1 \le l \le P+1$ длины

$$len = \frac{f_h^m - f_l^m}{P+1}; \quad (5)$$

   г) Находятся их центры:

$$C^m[i] = f_l^m + i \cdot len, 1 \le i \le P \quad (6)$$

   и, переводя в шкалу Гц,

$$C[i] = F_{hz}(C^m[i]), 1 \le i \le P \quad (7)$$

Таким образом, находятся центральные частоты треугольных фильтров.

д) Центры треугольных фильтров переводятся в номера отсчетов массива $P_n[k]$ из герц:

$$f_{smp}[i] = \frac{M}{F_S} C[i], 1 \le i \le P \quad (8)$$

Где F_S – частота дискретизации исходного сигнала;

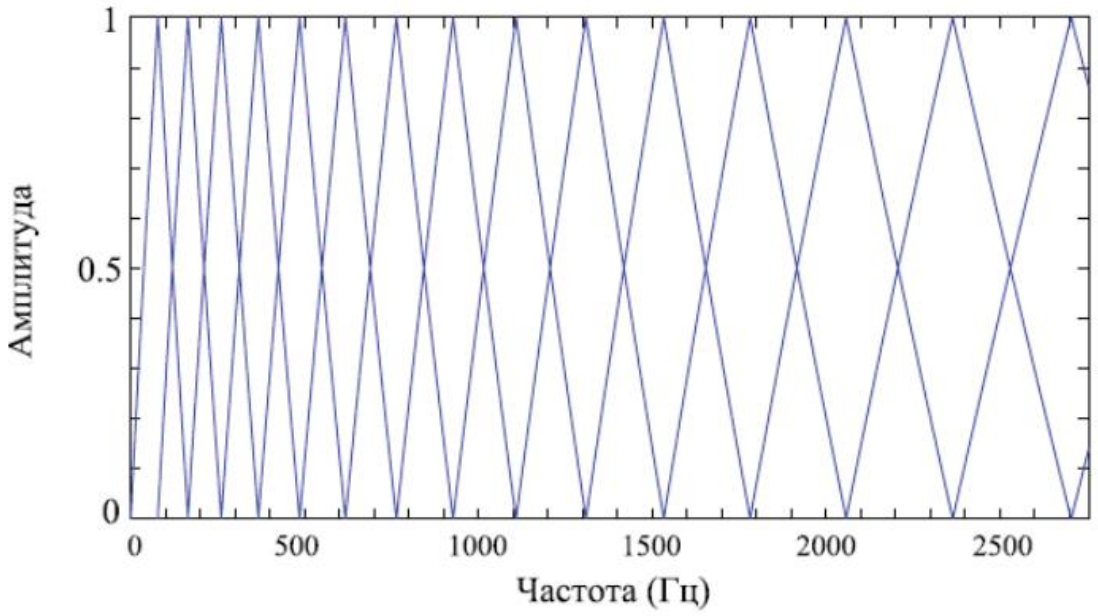

Рис. 2 Банк фильтров

е) Умножаем отсчеты спектральной плотности на соответствующий фильтр, и так для каждого фильтра:

$$X_n[i] = \sum_{k=1}^{M} P_n[k] H_i[k], 1 \le i \le P \quad (9)$$

$$H_i[k] = \begin{cases} 0, k < f_{smp}[i-1], \\ \frac{k - f_{smp}[i-1]}{f_{smp}[i] - f_{smp}[i-1]}, f_{smp}[i-1] \le k \le f_{smp}[i], \\ \frac{f_{smp}[i+1] - k}{f_{smp}[i+1] - f_{smp}[i]}, f_{smp}[i] \le k \le f_{smp}[i+1], \\ 0, k > f_{smp}[i+1]. \end{cases} \quad (10)$$

Берем логарифм:

$$X_n[i] = \ln(X_n[i]), 1 \le i \le P \quad (11)$$

Дискретное косинусное преобразование:

$$C_n[j] = \sum_{k=1}^{P} X_n[k] \cos(j(k - \frac{1}{2})\frac{\pi}{P}), 1 \le j \le J, \quad (12)$$

где $C_n[j]$ – массив кепстральных коэффициентов; J – желаемое число коэффициентов $(J < P)$.

2.4.2 Выбор классификатора признаков

Создание моделей дикторов прошло путь, начиная от простых (усреднение признаков) и до сложных генеративных (за основу взято моделирование данных, которые получаются специально для обучения) и дискриминативных (основанных на построении границ между классами) моделей. На данный момент методы создания моделей делятся на текстозависимые и, соответственно, текстонезависимые. К текстозависимым относятся такие методы как динамическое искажение времени, или Dynamic Time Warping (DTW), и скрытые марковские модели, или Hidden Markov Model (HMM). К текстонезависимым системам относят векторное квантование, или Vector Quantization (VQ), модели гауссовых смесей, или Guassian Mixture Model (GMM), и метод опорных векторов, или Support Vector Machine (SVM). Принятие решения может осуществляться как с использованием одного классификатора, так и с помощью объединения нескольких классификаторов для увеличения точности классификации. Рассмотрим подробнее каждый из них.

2.4.2.1 Dynamic Time Warping

Динамическое искажение времени (DTW) – это метод, который позволяет найти за некоторый промежуток времени близость между двумя последовательностями измерений. Измерения близости могут происходить с разной скоростью, а также для последовательностей разной длины.

Последовательность векторов признаков входного речевого сигнала из обучающей выборки $Q = \{q_1, \dots, q_n\}$ в данном методе и будет выступать в роли модели диктора. Возьмем в качестве тестовой выборки последовательность векторов признаков входного речевого сигнала $C = \{c_1, \dots, c_m\}$. Необходимо также ввести понятие набора индексов матрицы $W = \{w_1, \dots, w_T\}$, которая является матрицей выравнивания последовательностей $M_{m \times n}$, в позиции (i, j) которой содержатся значение выравнивания между элементами c_i и q_j последовательностей C и Q. Набор индексов матрицы W определяет соответствие между сопоставимыми последовательностями. А элементы набора W должны удовлетворять условиям:

1. $w_1 = (1,1), w_T = (m, n)$
2. Если $w_{t-1} = (a', b')$, то $w_t = (a, b)$, где $a - a' \le 1, b - b' \le 1$

Нахождение набора W, который удовлетворяет условиям 1 и 2, с суммарным искажением последовательностей C и Q равным минимальному, и является целью алгоритма DTW:

$$DTW(Q,C) = min\left\{\frac{1}{T}\sqrt{\sum_{t=1}^{T} M(w_t)}\right\} \quad (13)$$

Мера близости последовательностей C и Q будет определяться этим выражением. Для нахождения значения $DTW(Q,C)$ применяется метод динамического программирования, где на каждом шаге вычисляется значение $M(i,j)$ по формуле:

$$M(i,j) = d(i,j) + min\{M(i-1,j-1), M(i-1,j), M(i,j-1)\} \quad (14)$$

При этом $M(0,0) = 0$, а все остальные элементы столбца и строки с индексом 0 инициализируются значением ∞. $d(i,j)$ определяет евклидово расстояние между элементами c_i и q_j. Результатом работы алгоритма будет значение $DTW(Q,C) = M(m,n)$.

Для определения диктора вычисляется значение $DTW(Q_i,C)$, для каждого сохраненного шаблона. Значение i, при котором достигается минимум, определяет номер диктора, образце которого наиболее близок к образцу входного речевого сигнала.

Основным преимуществом алгоритма DTW является простота реализации. Тем не менее, данный алгоритм неприменим для решения задачи текстонезависимой идентификации диктора.

2.4.2.2 Hidden Markov Model

Скрытые Марковские модели (HMM) – это статистическая модель, которая может использоваться для решения задачи классификации скрытых параметров на основе наблюдаемых. HMM представляет собой конечный автомат, в котором переходы между состояниями осуществляются с некоторой вероятностью, и задано стартовое состояние, с которого начинается процесс. Через дискретные моменты времени может осуществляться переход в новые состояния. При этом каждому скрытому состоянию с заданной вероятностью соответствует наблюдаемое состояние. Кроме того, текущее состояние автомата зависит только от конечного числа предыдущих, а закон смены состояний не меняется во времени. Рассмотрим случай, когда текущее состояние зависит только от предыдущего (т.е. модель первого порядка).

HMM определяется следующими параметрами:

а) Множество скрытых состояний $Q = \{q_0, \dots, q_N\}$, где q_0 – начальное состояние, q_N – конечное состояние;
б) Множество наблюдений $O = \{o_1, \dots, o_M\}$;
в) Исходное распределение состояний $\pi = \{\pi_i\}, 1 \le i \le N$, которое определяет вероятность начать работу в состоянии i;
г) Матрица вероятностей переходов между скрытыми состояниями $A_{N\times N}: A(i,j) = a_{ij} = P(q_i, q_j), 1 \le i,j \le N$;
д) Матрица вероятностей наблюдений

$$B_{N\times M}: B(i,j) = b_{ij} = P(o_j|q_i), 1 \le i \le N, 1 \le j \le M;$$

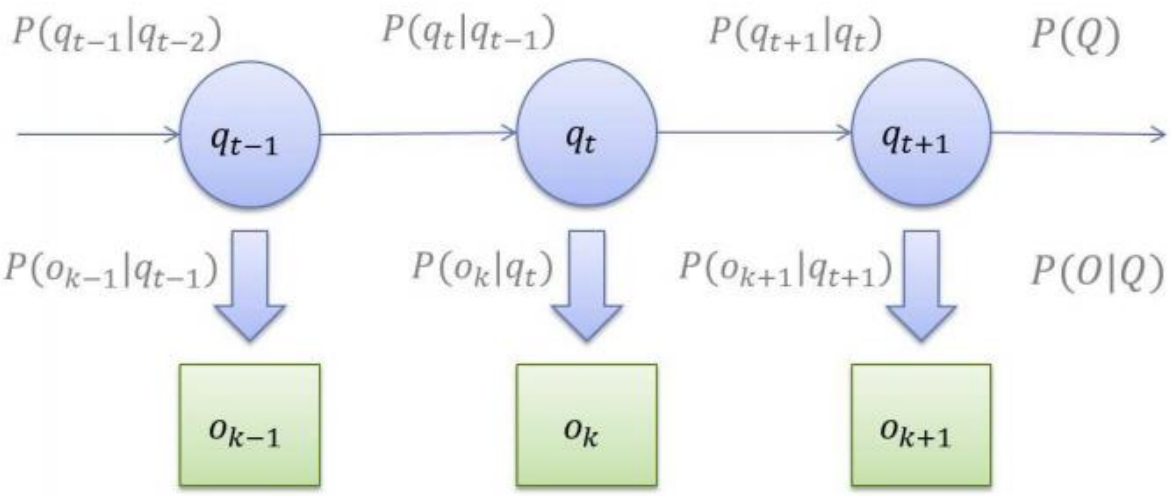

Рис. 3 HMM

Выделяются три основные задачи, решаемые с помощью HMM:

Задача 1. Вычисление вероятности последовательности наблюдений.

Требуется определить вероятность появления заданной последовательности наблюдений $O = \{o_1, \dots, o_R\}$. Для этого используется следующий алгоритм:

$$a_0(i) = a_{0i}, 1 \le i \le N \quad (15)$$

$$a_j(r) = \sum_{i=1}^{N} a_i(r-1)a_{ij}b_{ij}, 1 \le j \le N, 1 \le r \le R \quad (16)$$

$a_j(r)$ – вероятность того, что на r-ом шаге модель окажется в состоянии j.

Тогда искомая вероятность определяется по формуле:

$$P(O|A,B) = \sum_{i=1}^{N} a_i(R) \quad (17)$$

Задача 2. Нахождение наиболее правдоподобной последовательности скрытых состояний для наблюдаемой последовательности.

Требуется найти наиболее правдоподобную последовательность скрытых состояний

$Q = \{q_1, \dots, q_R\}$ для заданной последовательности наблюдений $O = \{o_1, \dots, o_R\}$, при которой достигается max $P(O|Q)$. Эта задача решается с помощью алгоритма, подобному алгоритму из предыдущего пункта с той лишь разницей, что на каждом шаге запоминается состояние i, в котором $a_i(r)$ принимает наибольшее значение. В итоге выбирается последовательность состояний, для которой $a_i(R)$ принимает наибольшее значение. Этот алгоритм называется алгоритмом Витерби.

Задача 3. Обучение параметров модели по заданной последовательности наблюдений и множеству скрытых состояний.

Требуется вычислить для заданной модели матрицы A и B. Для решения данной задачи применяется алгоритм Баума-Велша.

Для задачи распознавания диктора скрытыми состояниями являются векторы признаков речевого сигнала из обучающей выборки, в качестве наблюдений – векторы признаков речевого сигнала из тестовой выборки. В качестве сохраняемой модели здесь выступают матрицы A и B.

HMM достаточно просты в понимании, имеют достаточно высокую точность распознавания, но, как и DTW, применяются в основном для задач текстозависимой идентификации диктора.

2.4.2.3 Vector Quantization

Задача векторного квантования для последовательности входных векторов $C = \{c_1, c_2, \dots, c_m\}$ с кодовыми векторами $W = \{w_1, w_2, \dots, w_n\}$ сводится к задаче с минимизацией искажения при замещении каждого вектора из Q соответствующим кодовым вектором.

Моделью диктора в данном методе является множество кодовых векторов, получаемое из входной последовательности векторов признаков речевого сигнала. Для построения этого множества исходная последовательность векторов признаков разбивается на L кластеров, и в качестве кодовых векторов берутся их центры.

Процесс идентификации диктора по входному речевому сигналу сводится к следующему. Для каждого тестового вектора c_i определяются k ближайших кодовых векторов. Пусть k_{ij} – количество векторов, принадлежащих диктору S_j, среди найденных ближайших. Тогда вероятность того, что вектор c_i принадлежит диктору S_j, определяется формулой:

$$P(S_j|c_i) = \frac{k_{ij}}{k} \qquad (18)$$

Тестовые вектора классифицируются следующим образом:

$$S = \underset{1 \le j \le N}{\operatorname{argmax}} \prod_{i=1}^{L} P(S_j|c_i) \qquad (19)$$

Для сглаживания погрешности измерения, связанной с близкими к нулю вероятностями, с учетом постоянности числа k, чаще используют правило:

$$S = \underset{1 \le j \le N}{\operatorname{argmax}} \prod_{i=1}^{L} P(S_j|c_i) = \underset{1 \le j \le N}{\operatorname{argmax}} \prod_{i=1}^{L} k_{ij} \qquad (20)$$

Метод векторного квантования прост в реализации, применим к задаче текстонезависимой идентификации диктора, однако не всегда дает высокую точность распознавания.

2.4.2.4 Support Vector Machine

Support Vector Machine (метод опорных векторов) – это бинарный классификатор, строящий в пространстве признаков функцию, которая разделяет это пространство и задает гиперплоскость, вида:

$$f(x) = w \cdot x + b \qquad (21)$$

Пусть задана последовательность точек пространства признаков $X = \{x_1, x_2, \dots, x_n\}$ с метками $Y = \{y_1, y_2, \dots, y_n\}, y_i \in \{-1,1\}, 1 \le i \le n,$ соответствующими двум классам. В случае линейной разделимости данных условия для нахождения функции $f(x)$ записываются в виде:

$$\begin{cases} w \cdot x_i + b \ge 1, y_i = 1 \\ w \cdot x_i + b \le -1, y_i = -1 \end{cases} \Leftrightarrow y_i(w \cdot x_i + b) - 1 \ge 0, 1 \le i \le n \qquad (22)$$

Для надежного разделения классов необходимо чтобы расстояние между плоскостями, разделяющими признаки, было как можно большим. Расстояние вычисляется как $\frac{2}{\|w\|}$, следовательно, задача поиска разделяющей гиперплоскости сводится к минимизации $\|w\|^2$ при указанных условиях (21). Эту задача также можно решить с помощью методов множителей Лагранжа.

Если полученные множества невозможно разделить, то вводится функция ядра. Основной идеей является то, что исходное пространство отображается в другое пространство, которое является пространством с более глубокой размерностью. В этом новом пространстве множества уже можно разделить линейно. При этом в силу, того что всюду в алгоритме признаки используются не отдельно, а в виде скалярных произведений, то необходимость строить данное преобразование явно отпадает. Задается функция ядра, которая уже определяет скалярное произведение в новом пространстве:

$$K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j) \qquad (23)$$

В качестве сохраняемой модели диктора в методе опорных векторов выступают параметры разделяющей функции $f(x)$, а также параметры функции ядра. Параметры ядра обычно определяются перебором некоторого множества значений и оценкой методом кросс-валидации.

После того, как решающая функция $f(x)$ вычислена, принадлежность вектора x' соответствующему классу определяется знаком выражения $f(x')$.

Метод опорных векторов также можно применять для задачи классификации более двух классов. Для этого используется стратегия «один против остальных». Для этого строятся q классификаторов, и обучение каждого предполагает способность отличать конкретный класс от всех остальных. При выполнении распознавания разделяющая функция $f(x)$ классификатора с максимальным значением для распознаваемого объекта и определяет к какому классу этот объект относится.

Метод опорных векторов дает высокую точность классификации, имеет теоретическое обоснование, позволяет применять различные подходы к классификации в соответствии с выбором функции ядра. Среди недостатков следует отметить проблему выбора ядра, а также медленное обучение в случае задачи многоклассового распознавания.

2.4.2.5 Gaussian Mixture Model

Модель гауссовых смесей представляет собой взвешенную сумму M компонент и может быть описана выражением:

$$P(\bar{x}|\lambda) = \sum_{i=1}^{M} p_i b_i(\bar{x}) \quad (24)$$

Где $\bar{x}$ – D-мерный вектор случайных величин, $p_i, 1 \le i \le M$ – веса компонентов модели, $b_i(\bar{x}), 1 \le i \le M$ – функции плотности распределения составляющих модели:

$$b_i(\bar{x}) = \frac{1}{(2\pi)^{D/2}|\sum_i|^{1/2}} exp\left\{-\frac{1}{2}(\bar{x}-\bar{\mu_i})^T \sum_i^{-1}(\bar{x}-\bar{\mu_i})\right\} \quad (25)$$

Где $\bar{\mu_i}$ - вектор математического ожидания и $\sum_i -$ ковариантная матрица. При этом веса смеси удовлетворяют условию:

$$\sum_{i=1}^{M} p_i = 1 \quad (26)$$

Полностью модель гауссовой смеси определяется векторами математического ожидания, ковариационными матрицами и весами смесей для каждого компонента модели:

$$\lambda = \{p_i, \bar{\mu_i}, \Sigma_i\}, i = 1, \dots M \quad (27)$$

При использовании данного метода каждый диктор представляется моделью гауссовых смесей λ.

Для построения модели диктора необходимо оценить её параметры, которые наилучшим образом соответствуют распределению векторов признаков обучающего высказывания. Наиболее популярным и широко используемым методом решения этой задачи является метод оценки максимального правдоподобия. Целью оценки максимального правдоподобия является нахождение

параметров модели, которые максимизируют правдоподобие этой модели, при заданных обучающих данных.

Для последовательности обучающих векторов $X = \{\overline{x_1}, \dots, \overline{x_T}\}$ правдоподобие модели гауссовых смесей может быть записано в виде:

$$P(X|\lambda) = \prod_{t=1}^{T} P(\overline{x_t}|\lambda) \quad (28)$$

Это выражение представляет нелинейную функцию от параметров λ, и ее непосредственное вычисление невозможно, поэтому обычно для оценки параметров применяется EM-алгоритм.

Пусть $S = \{S_1, S_2, \dots, S_N\}$ – группа дикторов, которые представлены набором гауссовых смесей $\lambda_1, \lambda_2, \dots, \lambda_N$.

При идентификации диктора требуется найти модель, которая имеет наибольшее значение апостериорной вероятности для заданного высказывания:

$$S = \underset{1 \le k \le N}{\operatorname{argmax}} P(\lambda_k|X) = \underset{1 \le k \le N}{\operatorname{argmax}} \frac{P(X|\lambda_k)P(\lambda_k)}{P(X)} = \underset{1 \le k \le N}{\operatorname{argmax}} P(X|\lambda_k) \quad (29)$$

Используя логарифм и независимость между наблюдениями, система идентификации диктора в итоге вычисляет:

$$S = \underset{1 \le k \le N}{\operatorname{argmax}} \sum_{t=1}^{T} \log P(\overline{x_t}|\lambda_t) \quad (30)$$

Модели гауссовых смесей представляют собой эффективный алгоритм с высокой точностью распознавания. Вместе с тем возникает ряд проблем, связанных с выбором числа компонентов модели и инициализацией её начальных параметров.

2.4.3 Выводы

Среди рассмотренных методов извлечения признаков, наиболее емкими по своим характеристикам являются кепстральные коэффициенты. Они довольно точно передают информацию, которая поступает в слуховой анализатор мозга человека.

На данный момент существует небольшое количество методов, позволяющих решать задачу текстонезависимой идентификации диктора по голосу, причем каждый из приведенных методов имеет свои преимущества и недостатки. За основу системы идентификации человека по голосу был взят SVM.

Метод опорных векторов позволяет классифицировать со сравнительно большой точностью. Также он относится к текстонезависимым методам классификации, что позволяет конструировать систему идентификации диктора без использования определенного словаря для диктора. Метод опорных векторов первоначально рассчитан на бинарную классификацию, но функции ядра позволяют сделать классификацию для большего количества классов. При, относительно, простой реализации получается достаточно надежная система классификации.

Именно поэтому данный подход можно успешно применять для решения задачи идентификации человека по голосу.

3 Разработка алгоритма программы

Рассмотрим блок-схемы работы системы идентификации диктора в режимах обучения и идентификации:

Рис. 4 Общая схема работы системы в режиме обучения.

Рис. 5 Общая схема работы системы в режиме идентификации.

На рисунке 6 более подробно отображен процесс извлечения признаков входного речевого сигнала.

Рис. 6. Процесс извлечения признаков из входного речевого сигнала

4 Разработка программных средств

В первую очередь, для реализации системы идентификации человека по голосу, необходимо представить звуковой файл в виде вектора данных. Поскольку после получения признаков из звукового файла получается тоже вектор, но уже с мел-кепстральными коэффициентами, то два действия были объединены в один метод:

1) Чтение звукового файла переменную

2) Получение признаков из прочитанного файла

В итоге вышла следующая функция:

```
fileToVector=function(fileName){
 soundFile <- readWave(fileName)
 mel <- melfcc(WaveMC(data = soundFile@left));
 melArray <- as.matrix(mel)
 melAverage <- rowMeans(melArray)
 return (melAverage)
}
```

В качестве аргумента требуется имя файла. Далее метод *readWave()* считывает звуковой файл и записывает в объект soundFile, по умолчанию частота была выбрана 44,1 кГц, количество бит равное 16. Метод *melfcc()* считает мел-кепстральные коэффициенты, в качестве аргумента передается объект класса WaveMC, к нему явно приводится объект soundFile. Т.к. *melfcc()* не поддерживает стерео, то приходится брать один канал. Метод *melfcc()* вызывается с минимальным количеством аргументов, все остальные поля, такие как количество коэффициентов и другие, принимают стандартное значение. По умолчанию, количество коэффициентов равно 12, поэтому на выходе метода *melfcc()* получается матрица размером 12х(количество секунд х 100). Для дальнейшей обработки необходимо получить одномерный массив коэффициентов для каждого файла, и, в целях упрощения, это было сделано путем нахождения среднего среди коэффициентов для каждого временного интервала. Результатом работы этой функции будет одномерный массив коэффициентов.

Следующим этапом будет формирование обучающей выборки данных. Формируем матрицу из одномерных массивов с признаками. Затем формируем тестовые данные.

С помощью метода *svm()* получаем модель классификации. В качестве параметров указываем обучающую выборку данных, классы, соответствующие признакам из обучающей выборки, и указываем тип «C-classification» для решения задачи классификации.

Для осуществления предсказаний на новых данных с помощью обученной SVM-модели в пакете e1071 служит функция *predict()*. В качестве аргументов передаем SVM-модель и тестовую выборку данных. Результатом будет структура данных, показывающая к какому классу относится каждый элемент из тестовой выборки.

5 Тестирование системы

В качестве базы голосов дикторов были выбраны восемь образцов голосов дикторов. Эти образцы были нарезаны на более маленькие файлы по 5 секунд. Полученные файлы преобразовывались в вектора с признаками. Большая часть из них определена на обучающую выборку, а часть определена в тестовую выборку.

Во время тестирования было выявлено, что необходимо большое количество файлов с длиной записи голоса не менее 5 секунд. Уменьшение длины записи и увеличение количества файлов результата не дало. Как и не дало результата увеличение длины записи и уменьшение количества файлов. В этом случае точность распознавания снизилась. Из этого можно сделать вывод, что чем больше файлов в обучающей выборке, тем выше точность распознавания. Для тестирования прототипа системы распознавания диктора по голосу было использовано по 30 образцов голоса для каждого диктора.

Модель была построена для следующих классов: "alehin", "bashkov", "brilev", "brileva", "chercas", "panova", "shadrina", "yarkeev". В качестве тестовой выборки были взяты дикторы alehin, bashkov, brilev, yarkeev, brileva, chercas в следующем порядке: alehin, bashkov, brilev, yarkeev, alehin, bashkov, chercas, yarkeev, alehin, brilev, brileva, chercas. Результатом работы функции *predict()* стало:

```
> model <- svm(trainMatrix, classes, type = "C-classification")
> new <- predict(model, testMatrix)
> show(new)
      1       2       3       4       5       6       7       8       9      10      11      12
 alehin bashkov  brilev yarkeev  alehin bashkov chercas yarkeev  alehin  brilev brileva chercas
Levels: alehin bashkov brilev brileva chercas panova shadrina yarkeev
```

Рис. 7 Результат работы программы

Заключение

В результате выполнения выпускной квалификационной работы спроектирован и программно реализован прототип системы идентификации человека по голосу. В качестве признаков речевого сигнала были взяты мел-кепстральные коэффициенты, а в качестве классификатора был выбран метод опорных векторов.

Созданная система предназначена для идентификации человека по голосу. Тестирование системы подтвердило её работоспособность и возможность использования для решения поставленной задачи.

В дальнейшем на основе созданного прототипа можно сделать полноценную систему распознавания в виде приложения на более популярных языках программирования, к примеру, Java или C#.

Список литературы

1. Иванов И.И. Анализ метода мел-частотных кепстральных коэффициентов применительно к процедуре голосовой аутентификации// Актуальные проблемы гуманитарных и естественных наук, № 10-1 / 2015

2. X. Huang, A. Acero, H. Hon. Spoken language processing: a guide to theory, algorithm, and system development. — Prentice Hall PTR, 2001. — P. 936.

3. Хроматиди, А.Ф. Исследование психофизиологического состояния человека на основе эмоциональных признаков речи: дис. …канд. тех. наук / А.Ф. Хроматиди. Таганрог, 2005.

4. Сидоров , К.В.,Филатова , Н.Н. (2012) Анализ признаков эмоционально-окрашенной речи. Вестник ТвГТУ, 180 (Вып. 20). стр. 26-32

5.Arya S., Mount D.M., Netanyahu N.S., Silverman R., Wu A. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions // Journal of the ACM. 1998. V. 45, N 6. P. 891–923

6. Садыхов Р.Х., Ракуш В.В. Модели гауссовых смесей для верификации диктора по произвольной речи // Доклады БГУИР. Минск, 2003. № 4. С. 95–103.

7. Reynolds D.A. An Overview of Automatic Speaker Recognition Technology // The International Conference on Acoustics, Speech, and Signal Processing ICASSP 02. 2002. P. 4072–4075.

8. Platt J.C. Fast Training of Support Vector Machines using Sequential Minimal Optimization // Advances in Kernel Methods / Ed. by B. Scholkopf, C.C. Burges, A.J. Smola. MIT Press, 1999. P. 185–208.

9. CJC Burges. A Tutorial on Support Vector Machines for Pattern Recognition http://www.music.mcgill.ca/˜rfergu/adamTex/references/Burges98.pdf

Приложение А – код прототипа системы

```
library(e1071)
library(tuneR)

fileToVector=function(fileName){
  soundFile <- readWave(fileName)
  mel <- melfcc(WaveMC(data = soundFile@left));
  melArray <- as.matrix(mel)
  melAverage <- rowMeans(melArray)
  return (melAverage)
}

alehin1 <- fileToVector("res/alehin/alehin_01.wav")
alehin2 <- fileToVector("res/alehin/alehin_02.wav")
alehin3 <- fileToVector("res/alehin/alehin_03.wav")
alehin4 <- fileToVector("res/alehin/alehin_04.wav")
alehin5 <- fileToVector("res/alehin/alehin_05.wav")
alehin6 <- fileToVector("res/alehin/alehin_06.wav")
alehin7 <- fileToVector("res/alehin/alehin_07.wav")
alehin8 <- fileToVector("res/alehin/alehin_08.wav")
alehin9 <- fileToVector("res/alehin/alehin_09.wav")
alehin10 <- fileToVector("res/alehin/alehin_10.wav")
alehin11 <- fileToVector("res/alehin/alehin_11.wav")
alehin12 <- fileToVector("res/alehin/alehin_12.wav")
alehin13 <- fileToVector("res/alehin/alehin_13.wav")
alehin14 <- fileToVector("res/alehin/alehin_14.wav")
alehin15 <- fileToVector("res/alehin/alehin_15.wav")
alehin16 <- fileToVector("res/alehin/alehin_16.wav")
alehin17 <- fileToVector("res/alehin/alehin_17.wav")
```

```
alehin18 <- fileToVector("res/alehin/alehin_18.wav")
alehin19 <- fileToVector("res/alehin/alehin_19.wav")
alehin20 <- fileToVector("res/alehin/alehin_20.wav")
alehin21 <- fileToVector("res/alehin/alehin_21.wav")
alehin22 <- fileToVector("res/alehin/alehin_22.wav")
alehin23 <- fileToVector("res/alehin/alehin_23.wav")
alehin24 <- fileToVector("res/alehin/alehin_24.wav")
alehin25 <- fileToVector("res/alehin/alehin_25.wav")
alehin26 <- fileToVector("res/alehin/alehin_26.wav")
alehin27 <- fileToVector("res/alehin/alehin_27.wav")
alehin28 <- fileToVector("res/alehin/alehin_28.wav")
alehin29 <- fileToVector("res/alehin/alehin_29.wav")
alehin30 <- fileToVector("res/alehin/alehin_30.wav")

bashkov1 <- fileToVector("res/bashkov/bashkov_01.wav")
bashkov2 <- fileToVector("res/bashkov/bashkov_02.wav")
bashkov3 <- fileToVector("res/bashkov/bashkov_03.wav")
bashkov4 <- fileToVector("res/bashkov/bashkov_04.wav")
bashkov5 <- fileToVector("res/bashkov/bashkov_05.wav")
bashkov6 <- fileToVector("res/bashkov/bashkov_06.wav")
bashkov7 <- fileToVector("res/bashkov/bashkov_07.wav")
bashkov8 <- fileToVector("res/bashkov/bashkov_08.wav")
bashkov9 <- fileToVector("res/bashkov/bashkov_09.wav")
bashkov10 <- fileToVector("res/bashkov/bashkov_10.wav")
bashkov11 <- fileToVector("res/bashkov/bashkov_11.wav")
bashkov12 <- fileToVector("res/bashkov/bashkov_12.wav")
bashkov13 <- fileToVector("res/bashkov/bashkov_13.wav")
bashkov14 <- fileToVector("res/bashkov/bashkov_14.wav")
bashkov15 <- fileToVector("res/bashkov/bashkov_15.wav")
bashkov16 <- fileToVector("res/bashkov/bashkov_16.wav")
```

```
bashkov17 <- fileToVector("res/bashkov/bashkov_17.wav")
bashkov18 <- fileToVector("res/bashkov/bashkov_18.wav")
bashkov19 <- fileToVector("res/bashkov/bashkov_19.wav")
bashkov20 <- fileToVector("res/bashkov/bashkov_20.wav")
bashkov21 <- fileToVector("res/bashkov/bashkov_21.wav")
bashkov22 <- fileToVector("res/bashkov/bashkov_22.wav")
bashkov23 <- fileToVector("res/bashkov/bashkov_23.wav")
bashkov24 <- fileToVector("res/bashkov/bashkov_24.wav")
bashkov25 <- fileToVector("res/bashkov/bashkov_25.wav")
bashkov26 <- fileToVector("res/bashkov/bashkov_26.wav")
bashkov27 <- fileToVector("res/bashkov/bashkov_27.wav")
bashkov28 <- fileToVector("res/bashkov/bashkov_28.wav")
bashkov29 <- fileToVector("res/bashkov/bashkov_29.wav")
bashkov30 <- fileToVector("res/bashkov/bashkov_30.wav")

brilev1 <- fileToVector("res/brilev/brilev_01.wav")
brilev2 <- fileToVector("res/brilev/brilev_02.wav")
brilev3 <- fileToVector("res/brilev/brilev_03.wav")
brilev4 <- fileToVector("res/brilev/brilev_04.wav")
brilev5 <- fileToVector("res/brilev/brilev_05.wav")
brilev6 <- fileToVector("res/brilev/brilev_06.wav")
brilev7 <- fileToVector("res/brilev/brilev_07.wav")
brilev8 <- fileToVector("res/brilev/brilev_08.wav")
brilev9 <- fileToVector("res/brilev/brilev_09.wav")
brilev10 <- fileToVector("res/brilev/brilev_10.wav")
brilev11 <- fileToVector("res/brilev/brilev_11.wav")
brilev12 <- fileToVector("res/brilev/brilev_12.wav")
brilev13 <- fileToVector("res/brilev/brilev_13.wav")
brilev14 <- fileToVector("res/brilev/brilev_14.wav")
brilev15 <- fileToVector("res/brilev/brilev_15.wav")
```

```
brilev16 <- fileToVector("res/brilev/brilev_16.wav")
brilev17 <- fileToVector("res/brilev/brilev_17.wav")
brilev18 <- fileToVector("res/brilev/brilev_18.wav")
brilev19 <- fileToVector("res/brilev/brilev_19.wav")
brilev20 <- fileToVector("res/brilev/brilev_20.wav")
brilev21 <- fileToVector("res/brilev/brilev_21.wav")
brilev22 <- fileToVector("res/brilev/brilev_22.wav")
brilev23 <- fileToVector("res/brilev/brilev_23.wav")
brilev24 <- fileToVector("res/brilev/brilev_24.wav")
brilev25 <- fileToVector("res/brilev/brilev_25.wav")
brilev26 <- fileToVector("res/brilev/brilev_26.wav")
brilev27 <- fileToVector("res/brilev/brilev_27.wav")
brilev28 <- fileToVector("res/brilev/brilev_28.wav")
brilev29 <- fileToVector("res/brilev/brilev_29.wav")
brilev30 <- fileToVector("res/brilev/brilev_30.wav")

brileva1 <- fileToVector("res/brileva/brileva_01.wav")
brileva2 <- fileToVector("res/brileva/brileva_02.wav")
brileva3 <- fileToVector("res/brileva/brileva_03.wav")
brileva4 <- fileToVector("res/brileva/brileva_04.wav")
brileva5 <- fileToVector("res/brileva/brileva_05.wav")
brileva6 <- fileToVector("res/brileva/brileva_06.wav")
brileva7 <- fileToVector("res/brileva/brileva_07.wav")
brileva8 <- fileToVector("res/brileva/brileva_08.wav")
brileva9 <- fileToVector("res/brileva/brileva_09.wav")
brileva10 <- fileToVector("res/brileva/brileva_10.wav")
brileva11 <- fileToVector("res/brileva/brileva_11.wav")
brileva12 <- fileToVector("res/brileva/brileva_12.wav")
brileva13 <- fileToVector("res/brileva/brileva_13.wav")
brileva14 <- fileToVector("res/brileva/brileva_14.wav")
```

```
brileva15 <- fileToVector("res/brileva/brileva_15.wav")
brileva16 <- fileToVector("res/brileva/brileva_16.wav")
brileva17 <- fileToVector("res/brileva/brileva_17.wav")
brileva18 <- fileToVector("res/brileva/brileva_18.wav")
brileva19 <- fileToVector("res/brileva/brileva_19.wav")
brileva20 <- fileToVector("res/brileva/brileva_20.wav")
brileva21 <- fileToVector("res/brileva/brileva_21.wav")
brileva22 <- fileToVector("res/brileva/brileva_22.wav")
brileva23 <- fileToVector("res/brileva/brileva_23.wav")
brileva24 <- fileToVector("res/brileva/brileva_24.wav")
brileva25 <- fileToVector("res/brileva/brileva_25.wav")
brileva26 <- fileToVector("res/brileva/brileva_26.wav")
brileva27 <- fileToVector("res/brileva/brileva_27.wav")
brileva28 <- fileToVector("res/brileva/brileva_28.wav")
brileva29 <- fileToVector("res/brileva/brileva_29.wav")
brileva30 <- fileToVector("res/brileva/brileva_30.wav")

chercas1 <- fileToVector("res/chercas/chercas_01.wav")
chercas2 <- fileToVector("res/chercas/chercas_02.wav")
chercas3 <- fileToVector("res/chercas/chercas_03.wav")
chercas4 <- fileToVector("res/chercas/chercas_04.wav")
chercas5 <- fileToVector("res/chercas/chercas_05.wav")
chercas6 <- fileToVector("res/chercas/chercas_06.wav")
chercas7 <- fileToVector("res/chercas/chercas_07.wav")
chercas8 <- fileToVector("res/chercas/chercas_08.wav")
chercas9 <- fileToVector("res/chercas/chercas_09.wav")
chercas10 <- fileToVector("res/chercas/chercas_10.wav")
chercas11 <- fileToVector("res/chercas/chercas_11.wav")
chercas12 <- fileToVector("res/chercas/chercas_12.wav")
chercas13 <- fileToVector("res/chercas/chercas_13.wav")
```

```
chercas14 <- fileToVector("res/chercas/chercas_14.wav")
chercas15 <- fileToVector("res/chercas/chercas_15.wav")
chercas16 <- fileToVector("res/chercas/chercas_16.wav")
chercas17 <- fileToVector("res/chercas/chercas_17.wav")
chercas18 <- fileToVector("res/chercas/chercas_18.wav")
chercas19 <- fileToVector("res/chercas/chercas_19.wav")
chercas20 <- fileToVector("res/chercas/chercas_20.wav")
chercas21 <- fileToVector("res/chercas/chercas_21.wav")
chercas22 <- fileToVector("res/chercas/chercas_22.wav")
chercas23 <- fileToVector("res/chercas/chercas_23.wav")
chercas24 <- fileToVector("res/chercas/chercas_24.wav")
chercas25 <- fileToVector("res/chercas/chercas_25.wav")
chercas26 <- fileToVector("res/chercas/chercas_26.wav")
chercas27 <- fileToVector("res/chercas/chercas_27.wav")
chercas28 <- fileToVector("res/chercas/chercas_28.wav")
chercas29 <- fileToVector("res/chercas/chercas_29.wav")
chercas30 <- fileToVector("res/chercas/chercas_30.wav")

panova1 <- fileToVector("res/panova/panova_01.wav")
panova2 <- fileToVector("res/panova/panova_02.wav")
panova3 <- fileToVector("res/panova/panova_03.wav")
panova4 <- fileToVector("res/panova/panova_04.wav")
panova5 <- fileToVector("res/panova/panova_05.wav")
panova6 <- fileToVector("res/panova/panova_06.wav")
panova7 <- fileToVector("res/panova/panova_07.wav")
panova8 <- fileToVector("res/panova/panova_08.wav")
panova9 <- fileToVector("res/panova/panova_09.wav")
panova10 <- fileToVector("res/panova/panova_10.wav")
panova11 <- fileToVector("res/panova/panova_11.wav")
panova12 <- fileToVector("res/panova/panova_12.wav")
```

```
panova13 <- fileToVector("res/panova/panova_13.wav")
panova14 <- fileToVector("res/panova/panova_14.wav")
panova15 <- fileToVector("res/panova/panova_15.wav")
panova16 <- fileToVector("res/panova/panova_16.wav")
panova17 <- fileToVector("res/panova/panova_17.wav")
panova18 <- fileToVector("res/panova/panova_18.wav")
panova19 <- fileToVector("res/panova/panova_19.wav")
panova20 <- fileToVector("res/panova/panova_20.wav")
panova21 <- fileToVector("res/panova/panova_21.wav")
panova22 <- fileToVector("res/panova/panova_22.wav")
panova23 <- fileToVector("res/panova/panova_23.wav")
panova24 <- fileToVector("res/panova/panova_24.wav")
panova25 <- fileToVector("res/panova/panova_25.wav")
panova26 <- fileToVector("res/panova/panova_26.wav")
panova27 <- fileToVector("res/panova/panova_27.wav")
panova28 <- fileToVector("res/panova/panova_28.wav")
panova29 <- fileToVector("res/panova/panova_29.wav")
panova30 <- fileToVector("res/panova/panova_30.wav")

shadrina1 <- fileToVector("res/shadrina/shadrina_01.wav")
shadrina2 <- fileToVector("res/shadrina/shadrina_02.wav")
shadrina3 <- fileToVector("res/shadrina/shadrina_03.wav")
shadrina4 <- fileToVector("res/shadrina/shadrina_04.wav")
shadrina5 <- fileToVector("res/shadrina/shadrina_05.wav")
shadrina6 <- fileToVector("res/shadrina/shadrina_06.wav")
shadrina7 <- fileToVector("res/shadrina/shadrina_07.wav")
shadrina8 <- fileToVector("res/shadrina/shadrina_08.wav")
shadrina9 <- fileToVector("res/shadrina/shadrina_09.wav")
shadrina10 <- fileToVector("res/shadrina/shadrina_10.wav")
shadrina11 <- fileToVector("res/shadrina/shadrina_11.wav")
```

```
shadrina12 <- fileToVector("res/shadrina/shadrina_12.wav")
shadrina13 <- fileToVector("res/shadrina/shadrina_13.wav")
shadrina14 <- fileToVector("res/shadrina/shadrina_14.wav")
shadrina15 <- fileToVector("res/shadrina/shadrina_15.wav")
shadrina16 <- fileToVector("res/shadrina/shadrina_16.wav")
shadrina17 <- fileToVector("res/shadrina/shadrina_17.wav")
shadrina18 <- fileToVector("res/shadrina/shadrina_18.wav")
shadrina19 <- fileToVector("res/shadrina/shadrina_19.wav")
shadrina20 <- fileToVector("res/shadrina/shadrina_20.wav")
shadrina21 <- fileToVector("res/shadrina/shadrina_21.wav")
shadrina22 <- fileToVector("res/shadrina/shadrina_22.wav")
shadrina23 <- fileToVector("res/shadrina/shadrina_23.wav")
shadrina24 <- fileToVector("res/shadrina/shadrina_24.wav")
shadrina25 <- fileToVector("res/shadrina/shadrina_25.wav")
shadrina26 <- fileToVector("res/shadrina/shadrina_26.wav")
shadrina27 <- fileToVector("res/shadrina/shadrina_27.wav")
shadrina28 <- fileToVector("res/shadrina/shadrina_28.wav")
shadrina29 <- fileToVector("res/shadrina/shadrina_29.wav")
shadrina30 <- fileToVector("res/shadrina/shadrina_30.wav")

yarkeev1 <- fileToVector("res/yarkeev/yarkeev_01.wav")
yarkeev2 <- fileToVector("res/yarkeev/yarkeev_02.wav")
yarkeev3 <- fileToVector("res/yarkeev/yarkeev_03.wav")
yarkeev4 <- fileToVector("res/yarkeev/yarkeev_04.wav")
yarkeev5 <- fileToVector("res/yarkeev/yarkeev_05.wav")
yarkeev6 <- fileToVector("res/yarkeev/yarkeev_06.wav")
yarkeev7 <- fileToVector("res/yarkeev/yarkeev_07.wav")
yarkeev8 <- fileToVector("res/yarkeev/yarkeev_08.wav")
yarkeev9 <- fileToVector("res/yarkeev/yarkeev_09.wav")
yarkeev10 <- fileToVector("res/yarkeev/yarkeev_10.wav")
```

```
yarkeev11 <- fileToVector("res/yarkeev/yarkeev_11.wav")
yarkeev12 <- fileToVector("res/yarkeev/yarkeev_12.wav")
yarkeev13 <- fileToVector("res/yarkeev/yarkeev_13.wav")
yarkeev14 <- fileToVector("res/yarkeev/yarkeev_14.wav")
yarkeev15 <- fileToVector("res/yarkeev/yarkeev_15.wav")
yarkeev16 <- fileToVector("res/yarkeev/yarkeev_16.wav")
yarkeev17 <- fileToVector("res/yarkeev/yarkeev_17.wav")
yarkeev18 <- fileToVector("res/yarkeev/yarkeev_18.wav")
yarkeev19 <- fileToVector("res/yarkeev/yarkeev_19.wav")
yarkeev20 <- fileToVector("res/yarkeev/yarkeev_20.wav")
yarkeev21 <- fileToVector("res/yarkeev/yarkeev_21.wav")
yarkeev22 <- fileToVector("res/yarkeev/yarkeev_22.wav")
yarkeev23 <- fileToVector("res/yarkeev/yarkeev_23.wav")
yarkeev24 <- fileToVector("res/yarkeev/yarkeev_24.wav")
yarkeev25 <- fileToVector("res/yarkeev/yarkeev_25.wav")
yarkeev26 <- fileToVector("res/yarkeev/yarkeev_26.wav")

test1 <- fileToVector("res/test/alehin_31.wav")
test2 <- fileToVector("res/test/bashkov_31.wav")
test3 <- fileToVector("res/test/brilev_31.wav")
test4 <- fileToVector("res/test/yarkeev_27.wav")
test5 <- fileToVector("res/test/alehin_34.wav")
test6 <- fileToVector("res/test/bashkov_35.wav")
test7 <- fileToVector("res/test/chercas_35.wav")
test8 <- fileToVector("res/test/yarkeev_35.wav")
test9 <- fileToVector("res/test/alehin_35.wav")
test10 <- fileToVector("res/test/brilev_34.wav")
test11 <- fileToVector("res/test/brileva_34.wav")
test12 <- fileToVector("res/test/chercas_35.wav")
```

trainMatrix <- rbind(alehin1[5:495], alehin2[5:495], alehin3[5:495], alehin4[5:495], alehin5[5:495],

alehin6[5:495], alehin7[5:495], alehin8[5:495], alehin9[5:495], alehin10[5:495], alehin11[5:495], alehin12[5:495],

alehin13[5:495], alehin14[5:495], alehin15[5:495], alehin16[5:495], alehin17[5:495], alehin18[5:495],

alehin19[5:495], alehin20[5:495], alehin21[5:495], alehin22[5:495], alehin23[5:495], alehin24[5:495],

alehin25[5:495], alehin26[5:495], alehin27[5:495], alehin28[5:495], alehin29[5:495], alehin30[5:495],

bashkov1[5:495], bashkov2[5:495], bashkov3[5:495], bashkov4[5:495], bashkov5[5:495], bashkov6[5:495],

bashkov7[5:495], bashkov8[5:495], bashkov9[5:495], bashkov10[5:495], bashkov11[5:495], bashkov12[5:495],

bashkov13[5:495], bashkov14[5:495], bashkov15[5:495], bashkov16[5:495], bashkov17[5:495], bashkov18[5:495],

bashkov19[5:495], bashkov20[5:495], bashkov21[5:495], bashkov22[5:495], bashkov23[5:495], bashkov24[5:495],

bashkov25[5:495], bashkov26[5:495],bashkov27[5:495], bashkov28[5:495], bashkov29[5:495], bashkov30[5:495],

brilev1[5:495], brilev2[5:495], brilev3[5:495], brilev4[5:495], brilev5[5:495], brilev6[5:495], brilev7[5:495],

brilev8[5:495], brilev9[5:495], brilev10[5:495], brilev11[5:495], brilev12[5:495], brilev13[5:495], brilev14[5:495],

brilev15[5:495], brilev16[5:495], brilev17[5:495], brilev18[5:495], brilev19[5:495], brilev20[5:495],

brilev21[5:495], brilev22[5:495], brilev23[5:495], brilev24[5:495], brilev25[5:495], brilev26[5:495],

brilev27[5:495], brilev28[5:495], brilev29[5:495], brilev30[5:495],

brileva1[5:495], brileva2[5:495], brileva3[5:495], brileva4[5:495], brileva5[5:495],

brileva6[5:495], brileva7[5:495], brileva8[5:495], brileva9[5:495], brileva10[5:495], brileva11[5:495], brileva12[5:495],

brileva13[5:495], brileva14[5:495], brileva15[5:495], brileva16[5:495], brileva17[5:495], brileva18[5:495], brileva19[5:495],

brileva20[5:495], brileva21[5:495], brileva22[5:495], brileva23[5:495], brileva24[5:495], brileva25[5:495], brileva26[5:495],

brileva27[5:495], brileva28[5:495], brileva29[5:495], brileva30[5:495],

chercas1[5:495], chercas2[5:495], chercas3[5:495], chercas4[5:495], chercas5[5:495],

chercas6[5:495], chercas7[5:495], chercas8[5:495], chercas9[5:495], chercas10[5:495], chercas11[5:495], chercas12[5:495],

chercas13[5:495], chercas14[5:495], chercas15[5:495], chercas16[5:495], chercas17[5:495], chercas18[5:495], chercas19[5:495],

chercas20[5:495], chercas21[5:495], chercas22[5:495], chercas23[5:495], chercas24[5:495], chercas25[5:495], chercas26[5:495],

chercas27[5:495], chercas28[5:495], chercas29[5:495], chercas30[5:495],

panova1[5:495], panova2[5:495], panova3[5:495], panova4[5:495], panova5[5:495],

panova6[5:495], panova7[5:495], panova8[5:495], panova9[5:495], panova10[5:495], panova11[5:495], panova12[5:495],

panova13[5:495], panova14[5:495], panova15[5:495], panova16[5:495], panova17[5:495], panova18[5:495], panova19[5:495],

panova20[5:495], panova21[5:495], panova22[5:495], panova23[5:495], panova24[5:495], panova25[5:495], panova26[5:495],

panova27[5:495], panova28[5:495], panova29[5:495], panova30[5:495],

shadrina1[5:495], shadrina2[5:495], shadrina3[5:495], shadrina4[5:495], shadrina5[5:495],

shadrina6[5:495], shadrina7[5:495], shadrina8[5:495], shadrina9[5:495], shadrina10[5:495], shadrina11[5:495], shadrina12[5:495],

shadrina13[5:495], shadrina14[5:495], shadrina15[5:495], shadrina16[5:495], shadrina17[5:495], shadrina18[5:495], shadrina19[5:495],

shadrina20[5:495], shadrina21[5:495], shadrina22[5:495], shadrina23[5:495], shadrina24[5:495], shadrina25[5:495], shadrina26[5:495],

shadrina27[5:495], shadrina28[5:495], shadrina29[5:495], shadrina30[5:495],

yarkeev1[5:495], yarkeev2[5:495], yarkeev3[5:495], yarkeev4[5:495], yarkeev5[5:495],

yarkeev6[5:495], yarkeev7[5:495], yarkeev8[5:495], yarkeev9[5:495], yarkeev10[5:495], yarkeev11[5:495], yarkeev12[5:495],

yarkeev13[5:495], yarkeev14[5:495], yarkeev15[5:495], yarkeev16[5:495], yarkeev17[5:495], yarkeev18[5:495], yarkeev19[5:495],

yarkeev20[5:495], yarkeev21[5:495], yarkeev22[5:495], yarkeev23[5:495], yarkeev24[5:495], yarkeev25[5:495], yarkeev26[5:495]

)

testMatrix <- rbind(test1[5:495], test2[5:495], test3[5:495], test4[5:495], test5[5:495], test6[5:495],

test7[5:495], test8[5:495], test9[5:495], test10[5:495], test11[5:495], test12[5:495])

classes <- c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

```
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
        5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
        8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8)
classes <- factor(classes, levels = c(1, 2, 3, 4, 5, 6, 7, 8))
levels(classes) <- c("alehin", "bashkov", "brilev", "brileva", "chercas", "panova", "shadrina", "yarkeev")


model <- svm(trainMatrix, classes, type = "C-classification")
new <- predict(model, testMatrix)
show(new)
```