



Олег @yourich

Пользователь

14,2

карма

0,0

рейтинг



Профиль

1

Публикации

2

Комментарии

17

Избранное

0

Подписчики

31 августа 2012 в 20:20

Разработка → Простой алгоритм распознавания речи по короткому словарю на основе MFCC из песочницы

Программирование*, C++*

Приветствую всех читателей habrahabr!

В последнее время наблюдается значительный рост интереса к технологиям, связанным с распознаванием речи. Можно назвать несколько причин этого роста, в частности, значительное количество возможностей и обучающего материала. На хабрахабре пользователем @domage был опубликован целый цикл статей по основам технологий распознавания речи. Также стоит отметить статью Мел-кепстральные коэффициенты (MFCC) и распознавание речи и выполненную на её основе работу по идентификации человека по голосу: Кто там? — Идентификация человека по голосу.

В данной работе предлагается простой алгоритм (и его реализация на C++) системы распознавания речи по короткому словарю, основанный на анализе статистического распределения мел-кепстральных коэффициентов (Mel-frequency cepstrum coefficients, MFCC).

Постановка задачи

Существуют множество методов распознавания речи, в подавляющем большинстве случаев они основаны на методах статистического анализа и теории вероятностей (Hidden Markov Model, Gaussian Mixture Model и т.п.). Как известно, компания google предоставляет бесплатный сервис по распознаванию коротких речевых сообщений. На основе этого сервиса было даже предложено распознавание речи при помощи микроконтроллера: Распознавание речи на STM32F4-Discovery . Однако, возникает вопрос: есть ли возможность сделать свою систему распознавания речи, пусть даже на довольно ограниченном по размеру словаре, без использования «внешних» сервисов, при этом чтобы она работала быстро и с приемлемым качеством?

Основная идея

Итак, для распознавания речи будем использовать MFCC. Чтобы не вдаваться в подробности скажу, что относиться к ним стоит лишь как к некоторому фильтру, на входе которого — фонограмма, на выходе — набор векторов (коэффициенты), который мы и будем распознавать как некоторое слово или набор слов. Справедливости ради стоит отметить, что существуют множество других акустических признаков, используемых для распознавания речи: Perceptual linear predictive (PLP), Linear prediction cepstral coefficient (LPCC), Linear frequency cepstral coefficients (LFCC).

Основная идея заключается в использовании [линейного дискриминантного анализа](#) для идентификации слова. Однако, он применим лишь для векторов одинаковой размерности. Т.к. слова могут быть различной длины, возникает вопрос: каким образом преобразовать последовательность произвольного числа MFCC-векторов в вектор фиксированной размерности?

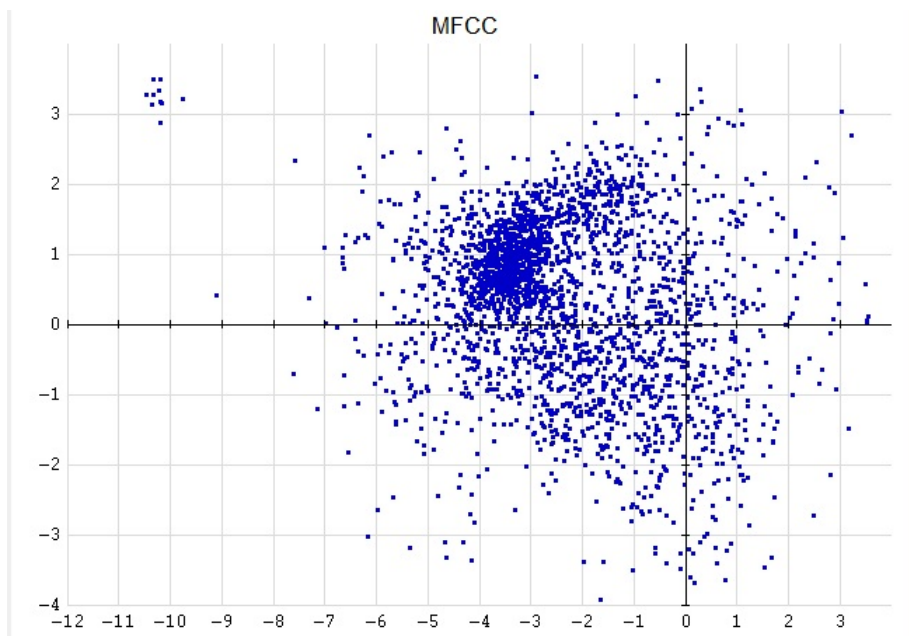
Можно поступить следующим образом: находить места «сгущения» распределения этих векторов и в качестве результирующего вектора брать конкатенацию векторов, являющихся центрами «сгущений». Такой конкатенированный вектор будем называть супервектором средних, а сами центры — средними значениями. При этом в качестве «отправной точки» будем использовать супервектор средних, полученный на всех MFCC-векторах всей базы обучения. Преобразовав таким образом последовательность MFCC-векторов в один супервектор средних фиксированной размерности, мы можем применять различные методы классификации.

Очевиден принципиальный недостаток такого подхода: не учитывается динамика распределения MFCC-признаков по времени, следовательно, система априори не способна различать, к примеру, слова «главрыба» и «абырвалг», т.к. общее распределение MFCC-векторов таких слов будет примерно одинаковым (соответственно, центры «сгущений» будут совпадать).

Описание алгоритма

В качестве базы обучения будем использовать множество файлов, каждый из которых представляет собой набор MFCC-векторов, полученных из фонограммы с записью того или иного слова. При этом файлы с записью одного и того же слова должны быть объединены в одну группу.

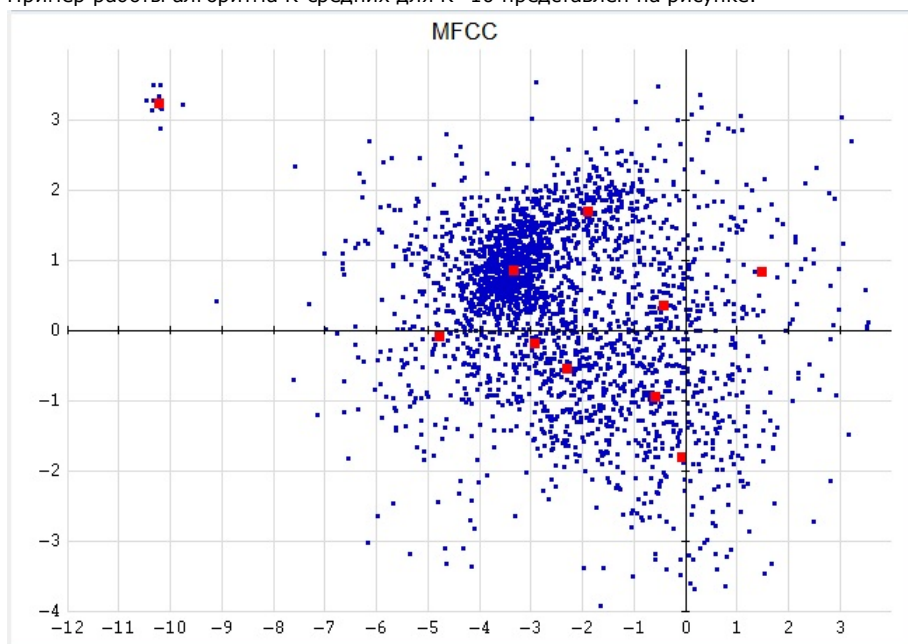
Вот как выглядит распределение первых двух компонент MFCC-векторов всей базы обучения:



Алгоритм состоит из следующих этапов:

1. Находим супервектор средних для всей базы обучения при помощи алгоритма К-средних.

Пример работы алгоритма К-средних для K=10 представлен на рисунке:



где большие красные квадраты и есть искомые средние значения.

2. Для каждого файла базы находим собственные средние значения по формуле:

$$M_k = a * M_{k0} + (1 - a) * M_k', \quad k = 1:K$$

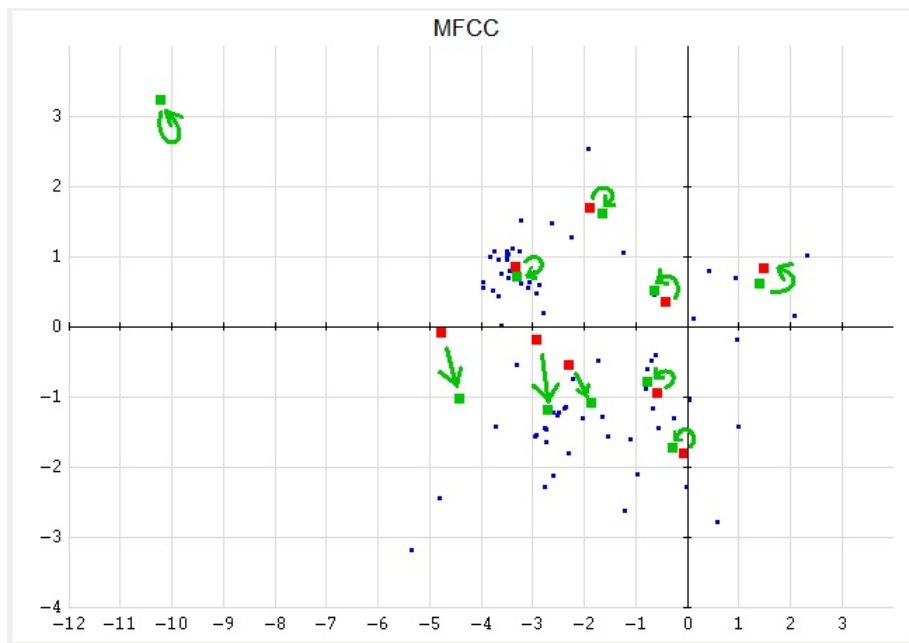
где M_{k0} — среднее значение, найденное в п.1,

M_k' — среднее значение, полученное в результате применения одной итерации алгоритма К-средних для MFCC-векторов файла с использованием в качестве начального значения M_{k0} ,

$a = R / (R + N_k)$, где R — коэффициент «чувствительности», N_k — число MFCC-векторов, соответствующие среднему значению M_k' .

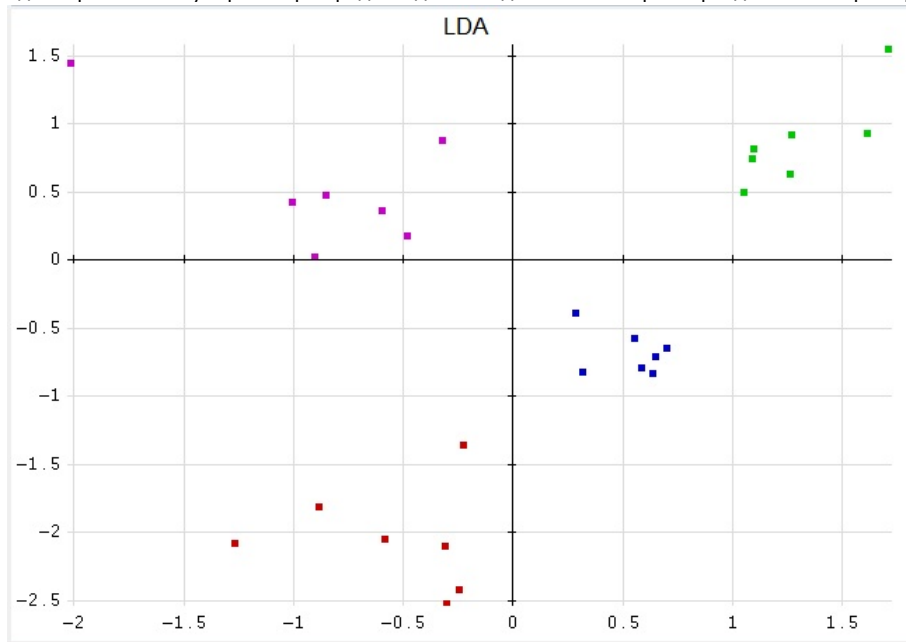
Найденные таким образом средние значения будем называть адаптированными средними значениями.

Пример адаптированных средних значений для файла представлен на рисунке:



3. Имея теперь вместо исходных фонограмм адаптированные супервектора средних, проводим LDA для N классов (каждый класс соответствует одному слову).

В результате мы должны получить матрицу, состоящую из векторов нового базиса, при проекции на который исходные адаптированные супервектора средних должны достаточно хорошо разделяться. Пример для N=4:



4. Проецируем все адаптированные супервектора средних на новый базис и находим средние значения и СКО (среднее квадратичное отклонение) проекций для каждого класса.
5. Для определения принадлежности тестовой фонограммы тому или иному классу (т.е. распознавания), выполняем для неё пп. 2 и 4, далее находим расстояния полученной проекции до средних значений всех классов (можно дополнительно нормировать их на соответствующее СКО). Минимальное расстояние и будет соответствовать классу, к которому принадлежит тестовая фонограмма.

Реализация

Полную реализацию описанного алгоритма вместе с исходными кодами и базой для тестирования можно взять [здесь](#).

Создание собственной системы распознавания слов состоит из следующих этапов:

1. Запись фонограмм для обучения и тестирования

Для записи можно воспользоваться любой программой, умеющей записывать звук и сохранять его в формате WAV. Я рекомендую использовать бесплатную программу [Audacity](#).

Разработанная система не умеет выделять речевые сегменты, поэтому при записи нужно стараться, чтобы в фонограмме присутствовала только речь. Чем качественнее используется микрофон, тем качественнее получается система. Записывать необходимо в моно-режиме с частотой дискретизации 16000.

2. Построение MFCC-векторов

Для построения MFCC-векторов можно использовать бесплатную библиотеку [SPro 5.0](#). Я взял на себя ответственность,

немного перебрал эту библиотеку, исправил парочку ошибок и сделал сборку программы sfbcep.exe под windows (см. папку ../spro-5.0). 32-разрядная версия этой программы лежит в папке ../tools. Для построения MFCC-векторов я использовал следующие параметры:

```
sfbcep.exe --format=wave --sample-rate=16000 --mel --freq-min=0 --freq-max=8000 --fft-length=256 --length=16.0 --shift=10.0 --m-seps=13 [входной WAVE-файл] [выходной файл с MFCC-векторами]
```

3. Обучение и тестирование системы

Для обучения и тестирования системы я написал программу wrsystem на языке C++. Полный исходный код находится в папке ../wrsystem. 32-разрядную версию этой программы можно взять в папке ../tools.

Релизация алгоритма LDA была позаимствована из библиотеки [ALGLIB](#).

Программа wrsystem имеет два режима работы: обучение (в случае наличия параметра --learn) и тестирование. Эта программа принимает на вход три основных параметра:

- Путь к файлу с описанием базы обучения (тестирования) (параметр --base). Пример файла с описанием базы лежит в папке ../base, также описание формата можно посмотреть, запустив программу с параметром --help.
- Путь к бинарному файлу, хранящему результат обучения системы (параметр --system). В режиме обучения этот файл создается, в режиме тестирования — считывается.
- Путь к файлу, в который записываются результаты тестирования системы на указанной базе: матрица перепутывания и значение WER (Word Error Rate) (параметр --test_results).

Результаты экспериментов

В качестве эксперимента я создал систему, которая умеет распознавать 14 слов, записанных моим голосом. Для обучения системы я записал каждое слово 4-5 раз, а для тестирования — 7 раз. Итого база обучения содержит 63 файла, а база тестирования — 98. Использовались следующие параметры при обучении:

- Количество средних значений: 10
- Коэффициент «чувствительности» при адаптации: 20
- Размерность проекции: 20
- Использование нормализации на СКО: отсутствует

Результат тестирования на базе обучения показал уровень ошибки распознавания слов (WER) 1,6%, а на базе тестирования 5,1%.

На что стоит обратить внимание

Хотелось бы сказать несколько замечаний. Во-первых, для того, чтобы любая система (включая описанную здесь) могла качественно распознавать речь любого человека, необходимо иметь огромную базу обучения с записью всех слов, произнесенных разными людьми в разном эмоциональном состоянии с использованием различных записывающих устройств (телефон, микрофон, подслушивающее устройство и т.п.). Т.е. система, которую вы обучите, используя только свой голос и только вашу домашнюю гарнитуру, наверняка не будет работать для ваших знакомых и даже для вас, если вы будете использовать какой-нибудь другой микрофон. Во-вторых, описанная система имеет сильно ограниченный потенциал в силу своей тривиальности. Не смотря на то, что она работает, данный подход был предложен только в качестве эксперимента и не подходит для промышленного использования без каких-либо доработок.

На этом всё, спасибо за внимание!

🔗 программирование, C++, распознавание речи

↑ +40 ↓

👁 42,1k ★ 247



Олег @yourich

карма рейтинг
14,2 0,0

Похожие публикации

+20 Asterisk + UniMRCP + VoiceNavigator. Синтез и распознавание речи в Asterisk. Часть 2

👁 3,1k ★ 72 💬 8

+24 Asterisk + UniMRCP + VoiceNavigator. Синтез и распознавание речи в Asterisk. Часть 1

👁 13,6k ★ 114 💬 22

+1 Распознавание речи на собственном сайте: тестовый стенд распознавания Speereo

👁 6,7k ★ 10 💬 13

Реклама помогает поддерживать и развивать наши сервисы

Подробнее

Спецпроект

Самое читаемое

Разработка

Сейчас Сутки Неделя Месяц

+11 Бравый справочник css-свойств для новичка
👁 6,6k ★ 105 💬 11

+47 Утки, Таиланд и T-SQL... или что может подстерегать программистов при работе с SQL Server?
👁 5,9k ★ 104 💬 22

+26 СМИ узнали о новом способе перехвата паролей и PIN-кода с мобильных телефонов
👁 14,6k ★ 48 💬 34

+15 Кто ты по профессии: Разница между «Programmer», «Software Engineer» и «Computer Scientist»
👁 12,9k ★ 70 💬 23


+19 Глубокое обучение для новичков: распознаем изображения с помощью сверточных сетей
👁 5k ★ 152 💬 7

Комментарии (3)

 **Akson87** 31 августа 2012 в 21:36 #


0 ↑ ↓

Спасибо за статью! Интересно!

 **BorisPlus** 31 августа 2012 в 22:00 #

0 ↑ ↓

Спасибо. Хорошо, что есть исходники. А в sPro какие были недочеты?

 **yourich** 2 сентября 2012 в 14:05 # ↵ ↑

0 ↑ ↓

В SPro не совсем правильно считывались wave-файлы (там считалось, что chunk «data» следует сразу за chunkом «fmt», что не всегда верно) и не считывался какой-то параметр (точно не помню какой).

Только зарегистрированные пользователи могут оставлять комментарии. Войдите, пожалуйста.

Интересные публикации



- Создание движка для блога с помощью Phoenix и Elixir / Часть 3. Добавление ролей 💬 2
- Пепелац с гравипапой — как я искала (и даже нашла) CRM для медицинского центра 💬 0
- Перевод отрывков из книги Роберта Хайнлайна «Заберите себе правительство» — часть 18 💬 11
- Глубокое обучение для новичков: распознаем изображения с помощью сверточных сетей 💬 7
- Утки, Таиланд и T-SQL... или что может подстерегать программистов при работе с SQL Server? 💬 22

