Михаил Крестьянинов @krestjaninoff
Пользователь
127,2 карма 0,0 рейтинг

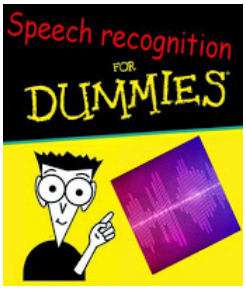
Профиль | 13 Публикации | 102 Комментарии | 238 Избранное | 18 Подписчики

13 июня 2014 в 10:13

Разработка → Распознавание речи для чайников

tutorial

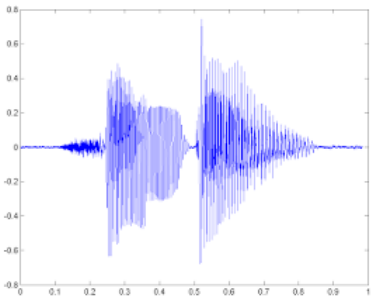
Алгоритмы*, Программирование*



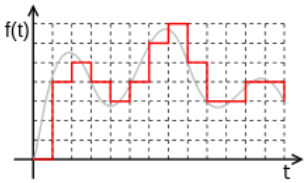
В этой статье я хочу рассмотреть основы такой интереснейшей области разработки ПО как Распознавание Речи. Экспертом в данной теме я, естественно, не являюсь, поэтому мой рассказ будет изобиловать неточностями, ошибками и разочарованиями. Тем не менее, главной целью моего «труда», как можно понять из названия, является не профессиональный разбор проблемы, а описание базовых понятий, проблем и их решений. В общем, прошу всех заинтересовавшихся пожаловать под кат!

Пролог

Начнём с того, что наша речь — это последовательность звуков. Звук в свою очередь — это суперпозиция (наложение) звуковых колебаний (волн) различных частот. Волна же, как нам известно из физики, характеризуются двумя атрибутами — амплитудой и частотой.



Для того, что бы сохранить звуковой сигнал на цифровом носителе, его необходимо разбить на множество промежутков и взять некоторое «усредненное» значение на каждом из них.



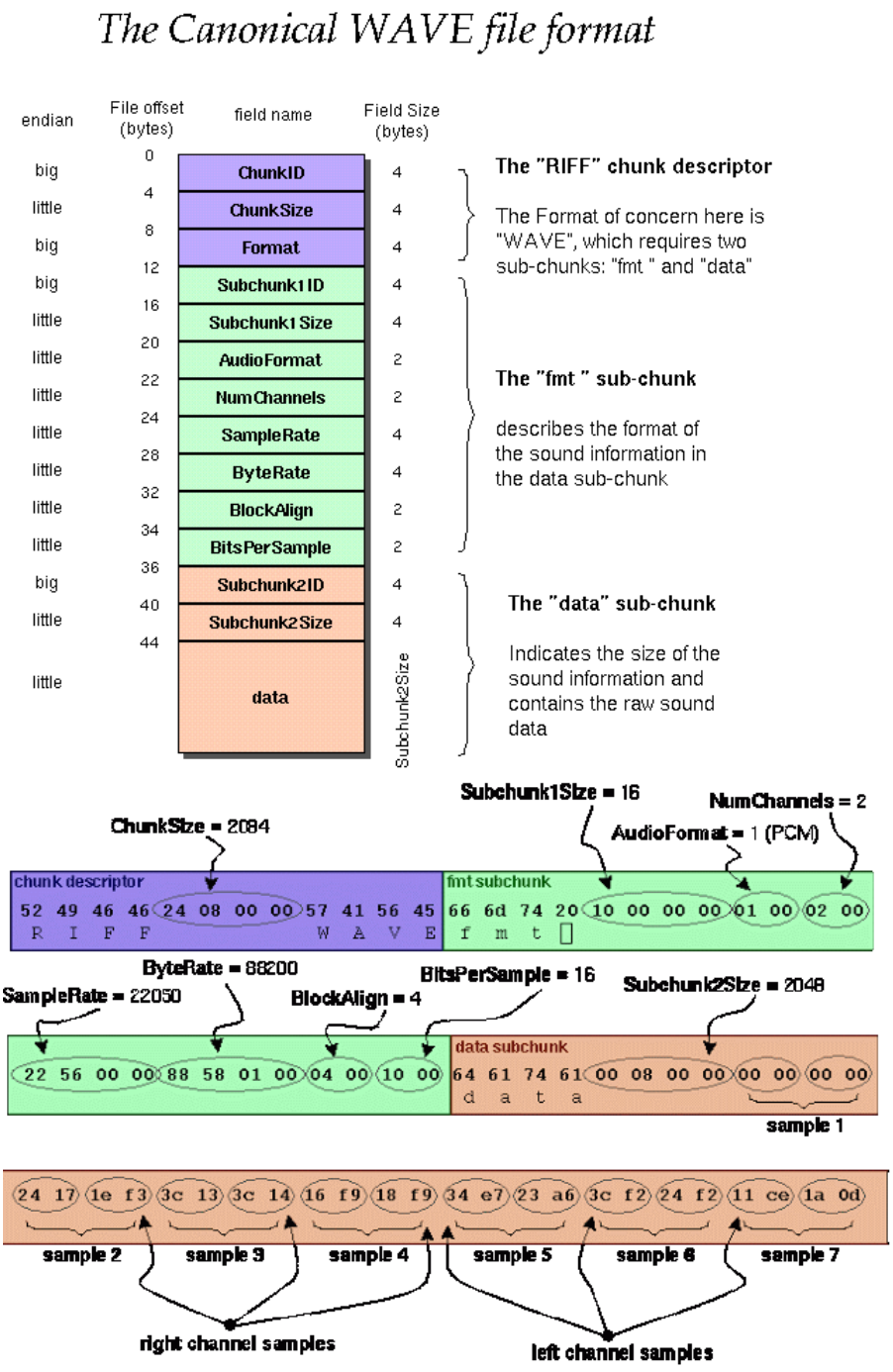
Таким вот образом механические колебания превращаются в набор чисел, пригодный для обработки на современных ЭВМ.

Отсюда следует, что задача распознавания речи сводится к «сопоставлению» множества численных значений (цифрового сигнала) и слов из некоторого словаря (русского языка, например).

Давайте разберемся, как, собственно, это самое «сопоставление» может быть реализовано.

Входные данные

Допустим у нас есть некоторый файл/поток с аудиоданными. Прежде всего нам нужно понять, как он устроен и как его прочесть. Давайте рассмотрим самый простой вариант — WAV файл.



Формат подразумевает наличие в файле двух блоков. Первый блок — это заголовка с информацией об аудиопотоке: битрейте, частоте, количестве каналов, длине файла и т.д. Второй блок состоит из «сырых» данных — того самого цифрового сигнала, набора значений амплитуд.

Логика чтения данных в этом случае довольно проста. Считываем заголовок, проверяем некоторые ограничения (отсутствие сжатия, например), сохраняем данные в специально выделенный массив.

Пример.

Распознавание

Чисто теоретически, теперь мы можем сравнить (поэлементно) имеющийся у нас образец с каким-нибудь другим, текст которого нам уже известен. То есть попробовать «распознать» речь… Но лучше этого не делать :)

Наш подход должен быть устойчив (ну хотя бы чуть-чуть) к изменению тембра голоса (человека, произносящего слово), громкости и скорости произношения. Поэлементным сравнением двух аудиосигналов этого, естественно, добиться нельзя.

Поэтому мы пойдем несколько иным путём.

Фреймы

Первым делом разобьём наши данные по небольшим временным промежуткам — фреймам. Причём фреймы должны идти не строго друг за другом, а “внахлёст”. Т.е. конец одного фрейма должен пересекаться с началом другого.

Фреймы являются более подходящей единицей анализа данных, чем конкретные значения сигнала, так как анализировать волны намного удобней на некотором промежутке, чем в конкретных точках. Расположение же фреймов “внахлёст” позволяет сгладить результаты анализа фреймов, превращая идею фреймов в некоторое “окно”, движущееся вдоль исходной функции (значений сигнала).

Опытным путём установлено, что оптимальная длина фрейма должна соответствовать промежутку в 10мс, «нахлёст» — 50%. С учётом того, что средняя длина слова (по крайней мере в моих экспериментах) составляет 500мс — такой шаг даст нам примерно 500 / (10 * 0.5) = 100 фреймов на слово.

Разбиение слов

Первой задачей, которую приходится решать при распознавании речи, является разбиение этой самой речи на отдельные слова. Для простоты предположим, что в нашем случае речь содержит в себе некоторые паузы (промежутки тишины), которые можно считать “разделителями” слов.

В таком случае нам нужно найти некоторое значение, порог — значения выше которого являются словом, ниже — тишиной. Вариантов тут может быть несколько:

- задать константой (сработает, если исходный сигнал всегда генерируется при одних и тех же условиях, одним и тем же способом);
- кластеризовать значения сигнала, явно выделив множество значений соответствующих тишине (сработает только если тишина занимает значительную часть исходного сигнала);
- проанализировать энтропию;

Как вы уже догадались, речь сейчас пойдёт о последнем пункте :) Начнём с того, что энтропия — это мера беспорядка, “мера неопределённости какого-либо опыта” (с). В нашем случае энтропия означает то, как сильно “колеблется” наш сигнал в рамках заданного фрейма.

Для того, что бы подсчитать энтропию конкретного фрейма выполним следующие действия:

- предположим, что наш сигнал пронормирован и все его значения лежат в диапазоне [-1;1];
- построим гистограмму (плотность распределения) значений сигнала фрейма:

рассчитаем энтропию, как $E = \sum_{i=0}^{N-1} P[i] * \log_2(P[i])$;

Пример.

И так, мы получили значение энтропии. Но это всего лишь ещё одна характеристика фрейма, и для того, что бы отделить звук от тишины, нам по прежнему нужно её с чем-то сравнивать. В некоторых статьях рекомендуют брать порог энтропии равным среднему между её максимальным и минимальным значениями (среди всех фреймов). Однако, в моём случае такой подход не дал сколь либо хороших результатов.
К счастью, энтропия (в отличие от того же среднего квадрата значений) — величина относительно самостоятельная. Что позволило мне подобрать значение её порога в виде константы (0.1).

Тем не менее проблемы на этом не заканчиваются :(Энтропия может проседать по середине слова (на гласных), а может внезапно вскакивать из-за небольшого шума. Для того, что бы бороться с первой проблемой, приходится вводить понятие “минимально расстояния между словами” и “склеивать” близ лежачие наборы фреймов, разделённые из-за проседания. Вторая проблема решается использованием “минимальной длины слова” и отсечением всех кандидатов, не прошедших отбор (и не использованных в первом пункте).

Если же речь в принципе не является “членораздельной”, можно попробовать разбить исходный набор фреймов на определённым образом подготовленные подпоследовательности, каждая из которых будет подвергнута процедуре распознавания. Но это уже совсем другая история :)

MFCC

И так, мы у нас есть набор фреймов, соответствующих определённому слову. Мы можем пойти по пути наименьшего сопротивления и в качестве численной характеристики фрейма использовать средний квадрат всех его значений (Root Mean Square). Однако, такая метрика несёт в себе крайне мало пригодной для дальнейшего анализа информации.

Вот тут в игру и вступают Мел-частотные кепстральные коэффициенты (Mel-frequency cepstral coefficients). Согласно Википедии (которая, как известно, не врёт) MFCC — это своеобразное представление энергии спектра сигнала. Плюсы его использования заключаются в следующем:

- Используется спектр сигнала (то есть разложение по базису ортогональных [ко]синусоидальных функций), что позволяет учитывать волновую “природу” сигнала при дальнейшем анализе;
- Спектр проецируется на специальную mel-шкалу, позволяя выделить наиболее значимые для восприятия человеком частоты;
- Количество вычисляемых коэффициентов может быть ограничено любым значением (например, 12), что позволяет “сжать” фрейм и, как следствие, количество обрабатываемой информации;

Давайте рассмотрим процесс вычисления MFCC коэффициентов для некоторого фрейма.

Представим наш фрейм в виде вектора $x[k], 0 \leq k < N$, где N — размер фрейма.

Разложение в ряд Фурье

Первым делом рассчитываем спектр сигнала с помощью дискретного преобразования Фурье (желательно его “быстрой” FFT реализацией).

$$X[k] = \sum_{n=0}^{N-1} x[n] * e^{-2*\pi*i*k*n/N}, 0 \leq k < N$$

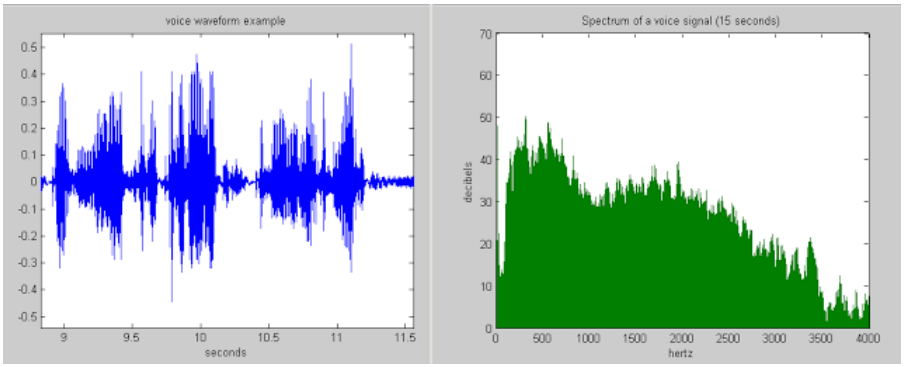
Так же к полученным значениям рекомендуется применить оконную функцию Хэмминга, что бы “сгладить” значения на границах фреймов.

$$H[k] = 0.54 - 0.46 * \cos(2 * \pi * k/(N-1))$$

То есть результатом будет вектор следующего вида:

$$X[k] = X[k] * H[k], 0 \leq k < N$$

Важно понимать, что после этого преобразования по оси X мы имеем частоту (hz) сигнала, а по оси Y — магнитуду (как способ уйти от комплексных значений):



Расчёт mel-фильтров

Начнём с того, что такое mel. Опять же согласно Википедии, mel — это “психофизическая единица высоты звука”, основанная на субъективном восприятии среднестатистическими людьми. Зависит в первую очередь от частоты звука (а так же от громкости и тембра). Другими словами, эта величина, показывающая, на сколько звук определённой частоты “значим” для нас.

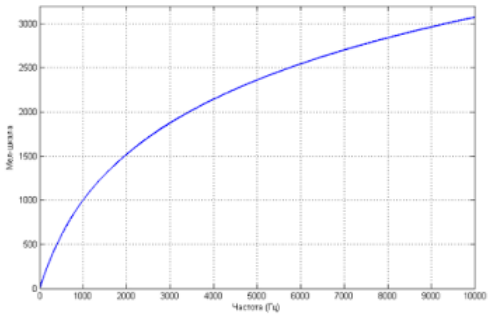
Преобразовать частоту в мел можно по следующей формуле (запомним её как «формула-1»):

$$M = 1127 * \log(1 + F/700)$$

Обратное преобразование выглядит так (запомним её как «формула-2»):

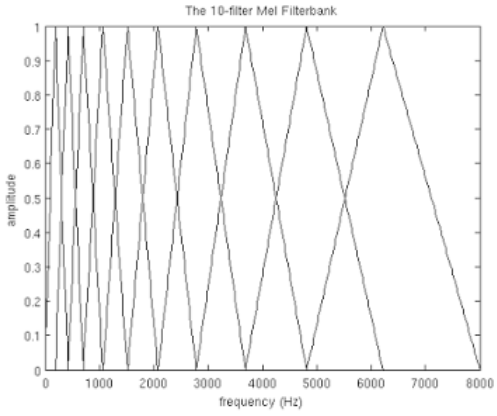
$$F = 700 * (e^{M/1127} - 1)$$

График зависимости mel / частота:



Но вернёмся к нашей задаче. Допустим у нас есть фрейм размером 256 элементов. Мы знаем (из данных об аудиоформате), что частота звука в данной фрейме 16000hz. Предположим, что человеческая речь лежит в диапазоне от [300; 8000]hz. Количество искомых мел-коэффициентов положим M = 10 (рекомендуемое значение).

Для того, что бы разложить полученный выше спектр по mel-шкале, нам потребуется создать “гребёнку” фильтров. По сути, каждый mel-фильтр это треугольная оконная функция, которая позволяет просуммировать количество энергии на определённом диапазоне частот и тем самым получить mel-коэффициент. Зная количество мел-коэффициентов и анализируемый диапазон частот мы можем построить набор таких вот фильтров:



Обратите внимание, что чем больше порядковый номер мел-коэффициента, тем шире основание фильтра. Это связано с тем, что разбиение интересующего нас диапазона частот на обрабатываемые фильтрами диапазоны происходит на шкале мелов.

Но мы опять отвлеклись. И так для нашего случая диапазон интересующих нас частот равен [300, 8000]. Согласно формуле-1 в на мел-шкале этот диапазон превращается в [401.25; 2834.99].

Далее, для того, что бы построить 10 треугольных фильтров нам потребуется 12 опорных точек:

> m[i] = [401.25, 622.50, 843.75, 1065.00, 1286.25, 1507.50, 1728.74, 1949.99, 2171.24, 2392.49, 2613.74, 2834.99]

Обратите внимание, что на мел-шкале точки расположены равномерно. Переведём шкалу обратно в герцы с помощью формулы-2:

> h[i] = [300, 517.33, 781.90, 1103.97, 1496.04, 1973.32, 2554.33, 3261.62, 4122.63, 5170.76, 6446.70, 8000]

Как видите теперь шкала стала постепенно растягиваться, выравнивая тем самым динамику роста "значимости" на низких и высоких частотах.

Теперь нам нужно наложить полученную шкалу на спектр нашего фрейма. Как мы помним, по оси X у нас находится частота. Длина спектра 256 — элементов, при этом в него умещается 16000hz. Решив нехитрую пропорцию можно получить следующую формулу:

> f(i) = floor((frameSize+1) * h(i) / sampleRate)

что в нашем случае эквивалентно

> f(i) = 4, 8, 12, 17, 23, 31, 40, 52, 66, 82, 103, 128

Вот и всё! Зная опорные точки на оси X нашего спектра, легко построить необходимые нам фильтры по следующей формуле:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \le k \le f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \le k \le f(m+1) \\ 0 & k > f(m+1) \end{cases}$$

Применение фильтров, логарифмирование энергии спектра

Применение фильтра заключается в попарном перемножении его значений со значениями спектра. Результатом этой операции является mel-коэффициент. Поскольку фильтров у нас M, коэффициентов будет столько же.

$$S[m] = \log(\sum_{k=0}^{N-1} |X[k]|^2 * H_m[k]), 0 \le m < M$$

Однако, нам нужно применить mel-фильтры не к значениям спектра, а к его энергии. После чего прологарифмировать полученные результаты. Считается, что таким образом понижается чувствительность коэффициентов к шумам.

Косинусное преобразование

Дискретное косинусное преобразование (DCT) используется для того, что бы получить те самые "кепстральные" коэффициенты. Смысл его в том, что бы "сжать" полученные результаты, повысив значимость первых коэффициентов и уменьшив значимость последних.

В данном случае используется DCTII без каких-либо домножений на $\sqrt{(2/N)}$ (scale factor).

$$C[l] = \sum_{m=0}^{M-1} S[m] * \cos(\pi * l * (m + \frac{1}{2})/M)), 0 \leq l < M$$

Теперь для каждого фрейма мы имеем набор из M mfcc-коэффициентов, которые могут быть использованы для дальнейшего анализа.

Примеры код для вышележащих методов можно найти тут.

Алгоритм распознавания

Вот тут, дорогой читатель, тебя и ждёт главное разочарование. В интернетах мне довелось увидеть множество высокоинтеллектуальных (и не очень) споров о том, какой же способ распознавания лучше. Кто-то ратует за Скрытые Марковские Модели, кто-то — за нейронные сети, чьи-то мысли в принципе невозможно понять :)

В любом случае немало предпочтений отдаётся именно CMM, и именно их реализацию я собираюсь добавить в свой код… в будущем :)

На данный момент, предлагаю остановится на гораздо менее эффективном, но в разы более простом способе.

И так, вспомним, что наша задача заключается в распознавании слова из некоторого словаря. Для простоты, будем распознавать называния первых десять цифр: "один", "два", "три", "четыре", "пять", "шесть", "семь", "восемь", "девять", "десять".

Теперь возьмем в руки айфон/андроид и пройдёмся по L коллегам с просьбой продиктовать эти слова под запись. Далее поставим в соответствие (в какой-нибудь локальной БД или простом файле) каждому слову L наборов mfcc-коэффициентов соответствующих записей.

Это соответствие мы назовём "Модель", а сам процесс — Machine Learning! На самом деле простое добавление новых образцов в базу имеет крайне слабую связь с машинным обучением… Но уж больно термин модный :)

Теперь наша задача сводится к подбору наиболее "близкой" модели для некоторого набора mfcc-коэффициентов (распознаваемого слова). На первый взгляд задачу можно решить довольно просто:

- для каждой модели находим среднее (евклидово) расстояние между идентифицируемым mfcc-вектором и векторами модели;
- выбираем в качестве верной ту модель, среднее расстояние до которой будет наименьшим;

Однако, одно и тоже слово может произносится как Андреем Малаховым, так и каким-нибудь его эстонским коллегой. Другими словами размер mfcc-вектора для одного и того же слова может быть разный.

К счастью, задача сравнения последовательностей разной длины уже решена в виде Dynamic Time Warping алгоритма. Этот алгоритм динамическо программирования прекрасно расписан как в буржуйской Wiki, так и на православном Хабре.

Единственное изменение, которое в него стоит внести — это способ нахождения дистанции. Мы должны помнить, что mfcc-вектор модели — на самом деле последовательность mfcc-"подвекторов" размерности M, полученных из фреймов. Так вот, DTW алгоритм должен находить дистанцию между последовательностями эти самых "подвекторов" размерности M. То есть в качестве значений матрицы расстояний должны использовать расстояния (евклидовы) между mfcc-"подвекторами" фреймов.

Пример.

Эксперименты

У меня не было возможности проверить работу данного подхода на большой "обучающей" выборке. Результаты же тестов на выборке из 3х экземпляров для каждого слова в несинтетических условиях показали мягко говоря нелучший результат — 65% верных распознаваний.

Тем не менее моей задачей было создание максимального простого приложения для распознавания речи. Так сказать "proof of concept" :)

Реализация

Внимательный читатель заметил, что статья содержит множество ссылок на GitHub-проект. Тут стоит отметить, что это мой первый проект на C++ со времён университета. Так же это моя первая попытка рассчитать что-то сложнее среднего арифметического со времён того же университета… Другими словами it comes with absolutely no warranty (c) :)

dct, dtw, fourier, mfcc, speech recognition

+51 61,1k 469

Михаил Крестьянинов @krestjaninoff карма 127,2 рейтинг 0,0

Похожие публикации

+29 Из пыльного архива в Интернет: как ABBYY Recognition Server оцифровывает библиотеки
11,1k 7 14

+16 C#, Разговоры с компом или System.Speech
25,6k 57 10

+47 ABBYY Recognition Server на службе ботаников Её Величества
7,5k 8 9

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 (20)

mbait 13 июня 2014 в 11:31 +2

Есть какие-то веские причины, по которым вы пишите свой проект, а не участвуете в уже существующих? Например, есть CMUSphinx и Kaldi.

krestjaninoff 14 июня 2014 в 00:49 +2

Да. Мне было просто интересно попробовать разобраться во всём самому!)

Fil 13 июня 2014 в 11:46 +7

Плюс за труд. Очень одобряю такие статьи и уважаю тех, кто их пишет. Но есть замечания, привел некоторые ниже. Некоторые придирочные, но главное замечание в том, что автору нужно находиться хотя бы на голову выше своей статьи. Есть некоторые моменты, в которых вы описываете как это делать, но не говорите зачем (на глубоком уровне). И хромает русский язык.

> Поэтому мой рассказ будет изобиловать неточностями, ошибками и разочарованиями

Честно говоря, немного расстраивает такой подход. Вроде бы здорово — человек старался, написал статью с отличной и интересной темой. Но почему бы не потратить время на повышение ее качества? Конечно, лучше заранее предупредить, чем потом слышать упреки. Но таким образом, вы сильно уменьшаете доверие к своей статье и желание ее читать. Какой же выход из этого? Просто сделать так, чтобы не было неточностей и ошибок, хотя бы свести их вероятность к минимуму. Если не уверены в каком-то предложении, изучите подробнее соответствующий материал. Да, это время, но оно будет работать на вас. И статья принесет больше пользы. По крайней мере, гораздо сильнее отразится на вашем рейтинге в положительную сторону.

> Волна же, как нам известно из физики, характеризуются двумя атрибутами — амплитудой и частотой

И начальной фазой.

> Для того, что бы сохранить звуковой сигнал на цифровом носителе, его необходимо разбить на множество промежутков и взять некоторое «усредненное» значение на каждом из них

Это подойдет для самого простого случая оцифровки. Но может сложиться впечатление, что АЦП работает так просто.

> Фреймы являются более подходящей единицей анализа данных, чем конкретные значения сигнала, так как анализировать волны намного удобней на некотором промежутке, чем в конкретных точках.

Скорее не то, что бы удобнее, а возможность получить спектр сигнала, ограниченного фреймом. Спектр конкретной точки не очень полезен.

> Мы знаем (из данных об аудиоформате), что частота звука в данной фрейме 16000hz

Имелась в виду частота дискретизации?

> построим гистограмму (плотность распределения) значений сигнала фрейма

Гистограмма — верно, но плотность распределения — понятие, обычно относящееся к непрерывным сигналам. Эти понятия часто путают.

> Так же к полученным значениям рекомендуется применить оконную функцию Хэмминга, что бы “сгладить” значения на границах фреймов.

главная цель — уменьшить утечку спектра

MichaelBorisov 13 июня 2014 в 14:11 +2

> Спектр конкретной точки не очень полезен

Спектр конкретной точки очень даже полезен, но вычислить его невозможно. Спектр нестационарного сигнала, каким является речь, вообще невозможно вычислить. Его можно только приблизительно оценивать. И тут у нас есть выбор — вычислить преобразование Фурье длинного куска сигнала, что даст нам усредненный спектр этого куска и тем самым ухудшит разрешение по времени; или вычислить БПФ короткого куска, но это ухудшит разрешение по частоте. Таким образом, разрешение по времени и разрешение по частоте — это взаимно конкурирующие показатели, улучшение одного из них приводит к ухудшению другого. См. принцип неопределенности Гейзенберга. Поэтому приходится искать какие-то компромиссы.

Кроме преобразования Фурье, существуют и другие способы оценки спектра, иногда позволяющие получить лучшие результаты. Есть методы, основанные на параметрических моделях (метод Юла-Уолкера, метод Берга, ковариационный, модифицированный ковариационный метод), вейвлет-преобразование Морлета, банки специально подогнанных фильтров и др. Еще есть очень многообещающий новый метод: Sparse Time-Frequency Representations.

Fil 13 июня 2014 в 15:54 0

Почему невозможно вычислить? На всякий случай уточню, что под спектром я понимаю Фурье-образ сигнала. Принцип неопределенности просто ограничивает частотно-временную локализацию, поэтому для конечной выборки мы не можем точно указать значение частоты, а лишь параметры локализации — ее среднее и дисперсию. Но в нашем случае и не нужно этого делать. Точка в дискретном случае — это импульс умноженный на константу, поэтому ПФ точки: $F\{a*\delta[n-b]\} = a*e^{-i\omega b}$. Представляет ли это выражение само по себе (как спектр) большой интерес? Но оно может быть полезным, если мы рассматриваем композицию импульсов, тогда их спектр может быть получен с помощью вышеуказанной формулы.

MichaelBorisov 13 июня 2014 в 16:41 0

Преобразование Фурье используется в процессе распознавания речи не как самоцель, а как метод оценки спектра сигнала. Поэтому ПФ длиной в 1 отсчет ничего не дает в плане оценки спектра, но тем не менее сам по себе мгновенный спектр — он объективно существует, хоть и неизвестен, и именно на его оценку направлены методы время-частотного анализа. И они далеко не всегда основаны на преобразовании Фурье.

MichaelBorisov 13 июня 2014 в 16:55 0

В качестве примера для иллюстрации, рассмотрим сигнал {x = sin(2*pi*f1*t) при t<0, x=sin(2*pi*f2*t) при t>=0. Это нестационарный сигнал, его спектр меняется скачкообразно в точке t=0. Нам известен истинный спектр этого сигнала — он представляет собой δ(f-f1) при t<0 и δ(f-f2) при t>=0. Однако попытки «вычислить» этот спектр, имея доступ только к значениям функции x(t) не позволят найти истинный спектр точно. Всегда будут как погрешности при определении частот f1 и f2, возможна некоторая «утечка спектра», а в окрестности t=0 эта утечка достигнет максимума.

Fil 13 июня 2014 в 17:01 +1

Да, я понял вас. Просто мы говорили о разных вещах. Конечно, спектральная и мгновенная частота не одно и то же.

krestjaninoff 14 июня 2014 в 01:01 +1

Спасибо за комментарий. Да я действительно не являюсь профессионалом в данной области, и да — я абсолютно согласен с тем, что автор должен быть «выше» своей статьи перед тем, как начать её писать. Единственная причина, по которой я, не будучи специалистом, решился на описание данного вопроса — отсутствие во интернете (по крайней мере во обозримой мною его части) информации для «чайников» во этом вопросе.

Если вы поделитесь со мной ссылкой, где подобный материал «для начинающих» освещен лучше — я немедленно удалю эту статью :)

mbait 13 июня 2014 в 11:50 +1

Что касается алгоритма распознавания — DTW совершенно не годится для дикторо-независимого распознавания, инфомации в MFCC-векторе просто недостаточно. Обычно, этот алгоритм используется для распознавания голосовых меток для одного и того же диктора. Например, голосовой набор адресной книги в телефоне. Но это только в случаях, где вычислительная мощность ограничена. Например, в Nokia Series 40. В Android для адресной книни используется GMM + грамматика, старая версия движка от Nuance. Эта же компания работает над распознаванием для Siri.

Ну и конечно же, я задам уже традиционный вопрос для этой рубрики: зачем, не являесь экспертом в этой области, вы пытаетесь что-то объяснить? Я никоим образом не связан с Яндекс, но примерно так я бы хотел видеть статьи «для начинающих».

mbait 13 июня 2014 в 12:05 0

Ещё фраза

> Кто-то ратует за Скрытые Марковские Модели, кто-то — за нейронные сети...

не совсем корректна. Традиционно фонемы моделировались смесью гауссианов. Там действительно основной алгоритм — Витерби для цепей Маркова. В последнее время популярность приобрело глубокое обучение. Но в любом случае — всё это происходит на этапе распознавания фонем. На этапе моделирования языка для свободной речи обычно используют N-граммные языковые модели, где используются всё те же скрытые марковские модели.

MichaelBorisov 13 июня 2014 в 14:20 +1

Спасибо за статью. У вас очень хорошо получилось подать материал «для чайников», вы быстро и понятно переходите от основ цифровой обработки сигналов к собственно распознаванию речи. И пусть действительно в статье имеются неточности, она представляет большой интерес для желающих понять подходы к распознаванию речи. Думаю, из вас вышел бы хороший преподаватель.

Теперь о неточностях. Выше уже указали на некоторые из них. Я отмечу следующее:

> Для того, что бы сохранить звуковой сигнал на цифровом носителе, его необходимо разбить на множество промежутков и взять некоторое «усредненное» значение на каждом из них.

Это в корне неверно. Сигнал ни в коем случае нельзя усреднять перед дискретизацией во времени. Это приведет к искажениям частотной характеристики, так как эквивалентно применению фильтра типа «скользящее среднее». На самом деле, для дискретизации сигнала необходимо взять отсчеты (мгновенные значения сигнала) через равные промежутки времени. Вся информация об уровне сигнала между взятыми отсчетами теряется. Но у нас есть теорема Котельникова, которая определенных условиях позволяет полностью восстановить сигнал, так что такое выбрасывание информации не страшно.

krestjaninoff 14 июня 2014 в 01:04 0

Спасибо, учту!

Trotil 13 июня 2014 в 16:08 0

Вот интересно, каждый язык характеризуется своим набором произносимых звуков человеком. Звуки эти разные: например, в русском языке есть звуки, которых нет в английском (и наоборот). Можно ли проранжировать языки по трудности их распознавания? Где окажется русский язык? Где окажется английский язык?

sergey_dobrodey 13 июня 2014 в 16:36 0

я думаю очень много зависит от самого говорящего.

zz_wolf 13 июня 2014 в 18:08 0

Спасибо за статью, написано хорошо и понятно. Но хотелось бы посмотреть результаты на правильном бенчмарке — насколько ваш препроцессинг (выглядит он убедительно) хорош по сравнению со state-of-the-art подходами.

Yogami 13 июня 2014 в 23:45 0

Если проговаривать в распознавалку от Google четко и ясно — он понимает большинство слов. Но если натравить его на обычную речь по радио или телевидении — он ничего не может распознать. Какие-то программы могут справиться с такой задачей?

mbait 14 июня 2014 в 01:01 0

Нужна тонкая настройка модели и распознавателя. Основы распознавания речи несколько более сложны, чем описал тут автор. В первом посте я привёл ссылки на два наиболее активно развиваемых проекта. У обоих есть хорошая документация и список рассылки, куда можно задавать вопросы. В CMUSphinx можете задавать их по-русски.

aledovskiy 14 июня 2014 в 03:37 0

Спасибо за статью. Хочу добавить зачем вообще нужны MFCC.
Как многие знают, алгоритм распознавания — это черный ящик. Что-то на входе, что-то на выходе. На вход идет вектор свойств (feature vector), а на выходе мы получаем решение алгоритма. И в задачах распознавания сложность состоит не самом алгоритме (как было сказано, алгоритмов много и они хорошо изучены), а именно в формировании вектора свойств. Вектор свойств должен быть некоторой фиксированной длины, поэтому напрямую отсчеты сигнала отправить нельзя. К тому же понятно, что по отсчетам напрямую распознать речь очень сложно — варьируется амплитуда, длительность звуков и т.д. Можно было бы отправить в алгоритм спектр. Но тут тоже возникают сложности — частот очень много, по ним сложно отличать звуки, просто каша получается. Кстати, насколько я знаю, в настоящий момент в распознавании речи распространены методы, основанные на вейвлетах, которые характеризуют сигнал и по частоте и по времени.
В общем, теперь вы можете понять, почему были разработаны кепстральные коэффициенты, как некоторый вектор, способный качественно охарактеризовать речь. Кстати, MFCC это лишь один из многих других вариантов feature vector.

KvanTTT 16 июня 2014 в 12:23 0

> Теперь возьмем в руки айфон/андроид и пройдёмся по L коллегам с просьбой продиктовать эти слова под запись. Далее поставим в соответствие (в какой-нибудь локальной БД или простом файле) каждому слову L наборов mfcc-коэффициентов соответствующих записей.

А можно ли вместо того, чтобы записывать голос коллег, синтезировать слова с помощью специальных программ- или сервисов- синтезаторов? Ведь получается слишком много слов. А вообще, скорее всего, синтезировать нужно не все слова, а отдельные сочетания букв.