Memoria Práctica 2

Detección de voz

Roger Reguan

Miquel Torrecilla Mercado

Processament d’àudio i veu

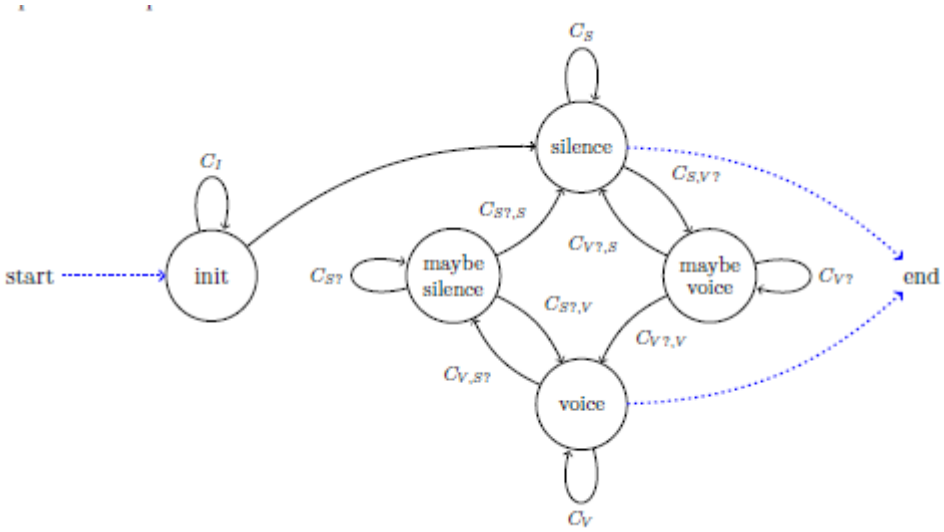
12 / 3 / 2020

Introducción

EL objetivo de esta práctica es crear un sistema que nos permita detectar, dada una señal de audio, segmentos de voz y de silencio mediante el uso de un autómata de estados finitos (FSA).

EL autómata tendrá 5 estados y será óptimo para la detección de la voz en señales con un ruido moderado. Usaremos una comparación de la potencia de la señal analizada mediante el uso de unos umbrales preestablecidos. Además a partir de nuestro autómata podremos saber la eficacia de nuestro sistema con el % de aciertos que tendremos en cada segmento tanto en voz como en silencio. Esto nos permitirá optimizarlo de una manera más ágil y eficaz durante el desarrollo de la práctica.

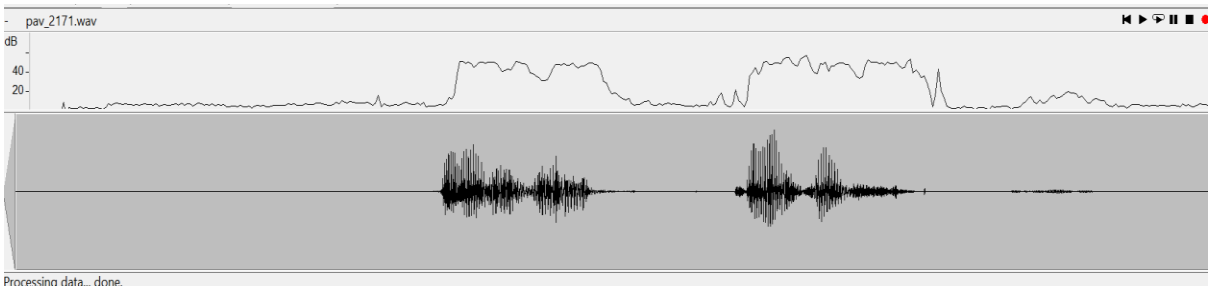
Los 5 estados son los siguientes:



Como ya hemos comentado tenemos cinco estados: init, silence, voice, maybe voice y maybe silence. Los estados principales son los de init, silence y voz que hacen referencia a la trama inicial (ini) en la que establecemos todos los valores iniciales y hacemos el cálculo de la potencia media y los umbrales decisión para determinar en qué situaciones cambiar de estado. Los otros dos estados hacen referencia a un periodo previo de transición desde silence a voz y viceversa y nos ayudarán a evitar errores que puedan venir producidos por el aumento del ruido de fondo o por un descenso brusco e indeseado en la potencia a lo largo de una trama de voz.

Tarea 1

La tarea 1 de la práctica nos pide que decidamos los valores de potencia en los que pondremos los umbrales. Usamos un análisis a simple vista de la señal usando el wavesurfer dónde decidimos usar los valores de 30 dB para el silencio.



Una vez implementado el código y analizado los resultados vimos que no era una buena aproximación. El nivel de potencia medio del ruido puede variar considerablemente de un audio a otro, por lo que debíamos caracterizar un nivel de ruido distinto para cada uno. Por tanto, decidimos usar la función compute_features() para calcular la potencia media del primer frame y asumimos que todos los frames empezaban en silencio, así que usamos ese valor como el umbral del silencio. Observamos una mejora considerable de los resultados, pasando de un 70% de acierto a casi un 90%. Pero no estábamos contentos con dichos resultados así que utilizamos un último método.

El método definitivo que utilizamos fue el de calcular la potencia media de la señal de los primeros 10 frames luego hacer la media geométrica de estos y establecer ese valor como el umbral del silencio.

El código utilizado es el siguiente:

```
static float power_array[10];
Features f = compute_features(x, vad_data->frame_length);
vad_data->last_feature = f.p; /* save feature, in case you want to show */

switch (vad_data->state) {
case ST_INIT:
  power_array[vad_data->fr_cont] = vad_data->last_feature;
  vad_data->fr_cont++;
  if(vad_data->fr_cont == 10){
    vad_data->state = ST_SILENCE;
    for(unsigned int i=0; i<10; i++){
      vad_data->power += pow(10, power_array[i]/10);
    }
    vad_data->power = 10*log10(vad_data->power/10);
    vad_data->k0 = vad_data->power;
    vad_data->k1 = vad_data->k0 + vad_data->p_alpha1;

    vad_data->fr_cont = 0;
    vad_data->power =0;
  }
  break;
```

Nota: Esta parte del código está dentro de la función vad() en el fichero vad.c.

Tarea 2

En esta tarea se nos pide que escribamos el fichero meson.build para poder compilar el programa de una forma rápida con un solo comando.

```
1  project ('Práctica 2 de PAV - detección de actividad vocal','c')
2
3  exe = 'vad'
4  src = ['src/main_vad.c', 'src/vad.c', 'src/pav_analysis.c']
5  lib = ['-lm', '-lsndfile']
6
7  executable(exe, sources: src, link_args: lib)
8
```

Además se nos pide ejecutar vad sin modificar nada. Vemos que los resultados son alrededor del 50% y esto se debe a que ahora está puesto de forma aleatoria. Es igual de probable de que ponga una trama en Silence o en Voice: P(S) = P(V) = 0,5.

Ejercicio 1

Analicemos ahora el código implementado en los archivos main_vad.c y vad.c, los cuáles nos van a permitir realizar la detección de voz y de silencio dados los distintos archivos de audio.

```
 switch (vad_data->state) {

  case ST_INIT:
        //mostrado en la tarea 1.
        break;

  case ST_SILENCE:
        if (f.p > vad_data->k1)
              vad_data->state = ST_MAYBE_VOICE;
        break;

  case ST_VOICE:
        if (f.p < vad_data->k0)
              vad_data->state = ST_MAYBE_SILENCE;
        break;

  case ST_MAYBE_VOICE:
        vad_data->fr_cont++;
        if (vad_data->fr_cont==vad_data->fr_threshold_voice){
              vad_data->state = ST_VOICE;
              vad_data->fr_cont =0;
        }else if(f.p > vad_data->k1)
              vad_data->state = ST_MAYBE_VOICE;
        else{
              vad_data->state = ST_MAYBE_SILENCE;
              vad_data->fr_cont = 0;
        }
        break;

  case ST_MAYBE_SILENCE:
        vad_data->fr_cont++;
        if (vad_data->fr_cont==vad_data->fr_threshold_silence){
              vad_data->state = ST_SILENCE;
              vad_data->fr_cont =0;
        }else if(f.p < vad_data->k1)
              vad_data->state = ST_MAYBE_SILENCE;
        else{
              vad_data->state = ST_MAYBE_VOICE;
              vad_data->fr_cont = 0;
        }
        break;

  case ST_UNDEF:
        break;
  }

  if (vad_data->state == ST_SILENCE || vad_data->state == ST_VOICE){
        //sumar aqui y solo tener un contador.
        return vad_data->state;
  }
  else
        return ST_UNDEF;
}

void vad_show_state(const VAD_DATA *vad_data, FILE *out) {
        fprintf(out, "%d\t%f\n", vad_data->state, vad_data->last_feature);
}
```

Como podemos observar, y para nuestro algoritmo, hemos decidido “extremar” los umbrales en los cuales vamos a cambiar al estado *maybe_voice* o *maybe_silence*. Por ejemplo, estando en *silence*, solo pasaremos al *maybe_voice* cuando superemos el umbral k1. Con esto evitaremos cambiar de estado innecesariamente cuando el nivel de potencia sea mayor al de k0 pero no alcanze nunca al k1, donde seguramente solo habrá un aumento del ruido de fondo y no habrá voz.

Una vez en *maybe_voice* nuestro autómata, conforme vayamos avanzando de muestras, irá acumulando en un contador el tiempo que llevamos en *maybe_voice*, siempre y cuando no bajemos de k0. Al llegar a un determinado número de muestras en *maybe_voice* (y por lo tanto, estando seguramente en lo cierto) pasaremos al estado voice considerando voice todo este rato. En caso de que estando en *maybe_voice* bajemos de k0, volveremos al estado *silence*, considerando que siempre hemos estado detectando silencio.

Para el caso contrario, es decir, estando en *voice* pasar a *maybe_silence*, el algoritmo es análogo. La única diferencia viene dada en los tiempo exigidos en cada maybe para asegurar el cambio de silencio a voz o viceversa.

Nuestro algoritmo pinta bien. En el siguiente ejercicio jugaremos con los parámetros de threshold y los niveles de potencia ki, y modificandolos en función de los resultados y de nuestra intuición, lograremos establecer aquellos valores que maximicen el resultado de nuestro algoritmo, el F-Score.

Observemos ahora el código implementado en main_vad.c:

```
if (verbose & DEBUG_VAD) vad_show_state(vad_data, stdout);

    if (state != last_state) {
        if(last_state == ST_VOICE || last_state == ST_SILENCE){
            if (t != last_t)
                fprintf(vadfile, "%.5f\t%.5f\t%s\n", last_t * frame_duration, t * frame_duration, state2str(last_state));
            last_t = t;
        }
        last_state = state;
        if(state == ST_VOICE || state == ST_SILENCE){
            if (t != last_t)
                fprintf(vadfile, "%.5f\t%.5f\t%s\n", last_t * frame_duration, t * frame_duration, state2str(last_state));
            last_t = t;
        }
    }
```

En nuestro main, principalmente haremos el print del resultado. Nuestro algoritmo va a presentar sólo S o V y nunca un estado maybe. Para ello, como observamos en el código, nos fijamos en el estado actual y en el anterior para la toma de la decisión.

Ejercicio 2

Después de variar los diferentes parámetros e ir analizando, a su vez, como variaba el resultado de nuestro algoritmo, los valores finales son los siguientes:

vad_data->power = 0; //(=k0)

vad_data->p_alpha1 = 6. //És el alpha para k1 → k1 = k0 + alpha

vad_data->fr_threshold_silence = 14; //Threshold del límite de veces de silence.

vad_data->fr_threshold_voice = 7; //Threshold del límite de veces de voice.

Veamos los resultados:

```
*************** Summary ***************
Recall V:375.27/384.10 97.70%   Precision V:375.27/419.64 89.43%   F-score V (2)  : 95.93%
Recall S:228.21/272.58 83.72%   Precision S:228.21/237.04 96.28%   F-score S (1/2): 93.47%
===> TOTAL: 94.691%
```

Nuestro algoritmo logra un F-Score Total de prácticamente el 94,7%. Además observemos que és un poquito más eficaz en detectar tramas de voz (casi el 96%) que de silencio, cosa que nos asegura no perder información, es decir, creemos que es menor grave el error de considerar voz alguna trama de silencio que al revés.

Ejercicios de ampliación

En este apartado, veremos podemos lograr que el programa, en el caso de que se indique un fichero .wav de salida (opción --output-wav), escriba en él la señal de la entrada, sustituyendo por cero los valores de los intervalos identificados como silencio.

El código queda implementado de la siguiente manera, dentro del fichero main_vad.c:

```
if (sndfile_out != 0 && (state == ST_SILENCE || state == ST_UNDEF)) {
        /* TODO: go back and write zeros in silence segments */
        sf_seek(sndfile_out, -n_read, SEEK_CUR);
        sf_write_float(sndfile_out,buffer_zeros, n_read);
}
```

Al realizar este apartado, reafirmamos nuestra creencia de que es mejor priorizar un F-Score de V alto antes que el de S (sabiendo que cuanto más grandes ambos mejor, evidentemente) pues de lo contrario pondríamos, con mayor error, tramas de voz a 0, y nuestro objetivo es preservar ante todo la señal de voz.