به نام خدا

طراحی سیستم ساده تشخیص صدا از موزیک

عنوان درس: پردازش سیگنال های دیجیتال

استاد درس : دکتر عابدین واحدیان مظلوم

نام دانشجو: مهدیه علیزاده

زمستان ۱۴۰۲

چکیده

در این پروژه قصد داریم یک سیستم تشخیص صدا از موزیک طراحی کنیم، با این کاربرد که در استدیو ها و ایستگاهای رادیویی در بسیاری از اوقات با تبلیغات و موزیک های ناخواسته مواجه میشویم در این پروژه قصد داریم به وسیله تکنیک های پردازش سیگنال گفتار را ازموزیک جدا سازی کنیم به طوری که وقتی تشخیص داده شود سیگنال اصلی گفتار است موزیک حذف شود و اگر سیگنال اصلی موزیک بود صدا قطع شود، که این کار را با استفاده از ویژگی های متمایز کننده سیگنال صدا و موزیک انجام دادیم.

فهرست مطالب

مقدمه

این پروژه با هدف قطع صدای موزیک در کانال ورودی و کاهش دامنه موزیک با شناسایی سیگنال صحبت انجام شده است. برای این منظور، از ویژگی PSD به منظور تشخیص و تفکیک سیگنال صحبت و موزیک استفاده شده است.

در این پروژه، ابتدا نمونه‌هایی از سیگنال صحبت و موزیک به طول چند دقیقه تهیه شده و با استفاده از فیلترینگ مناسب و ویژگی‌های مناسب، سیگنال‌های تحت بررسی تشخیص داده شده‌اند. با استفاده از تعدادی داده آموزشی از سیگنال صحبت و موزیک، آستانه‌ای برای تشخیص این دو نوع سیگنال بدست آمده است که در مرحله بعدی برای داده‌های تست استفاده می‌شود.

سپس، یک برنامه نوشته شده است که هر یک از نمونه‌های موجود برای تست سیگنال ورودی را تحلیل کرده و با شناسایی سیگنال صحبت، دامنه موزیک را کاهش داده و با شناسایی موزیک، صدا را قطع می‌کند. این برنامه قادر است آغاز و پایان صحبت و همچنین موزیک را در سیگنال ورودی شناسایی کند.

فصل دوم: روش انجام پروژه

۱-۲- مقدمه

در ابتدا یک سری داده به صورت تفکیک شده از صداهای افراد مختلف جمع آوری کردیم، یک سری موسیقی انواع مختلف پاپ، راک، جاز و.. نیز جمع آوری کردیم به صورت جدا، دومین کاری که انجام دادیم بدست آوردن یک feature مناسب برای تفکیک صدای انسان و موسیقی، ویژگی های مختلفی را امتحان کردیم در نهایت ویژگی که ما را به نتیجه رساند در این پروژه frequency_filter هست که به صورت یک کلاس در پروژه تعریف شده که از یک کلاسی به نام windowing ارث بری میکند. عملکرد کلاس windowing به طور کلی : دریافت یک صدا و برش آن به تعداد ثانیه های طول آن فایل، برای مثال اگر موجی به طول ۳۰ ثانیه دریافت کند خروجی در نهایت ۳۰ سیگنال یک ثانیه ایی خواهد بود. در نهایت میدانیم برای نشان دادن تفاوت بین دو سیگنال نیاز به یکسری پارامتر داریم، که این پارامتر اصلی همان threshold هست حالا با استفاده از داده ها و این ترشولد قرار است سیگنال های مختلف صدا و موسیقی را از هم تفکیک کنیم. در قسمت train دقیقا همین بیان شده به این صورت که میخواهد یک ترشولد از داده های اسپیچ و موزیک بدست آورد. برای انجام تست پروژه هم باید یک سری دیتا تست آماده کنیم، چون دیتاهای موجود در اینترنت به صورت پخش دو کانال جدا نبودند مجبور شدم کدی پیاده سازی کنیم که دو فایل موسیقی و اسپیچ را به صورت جدا دریافت کند .بعد از اینکه داده تست به صورت ترکیبی دو کاناله از صوت و موسیقی تولید شد به صورتی که در یک سری جاها همپوشانی داشته باشند، قرار هست قسمت هایی که صدای موسیقی روی اسپیچ به طور ناخواسته افتاده کم کم صدای خواننده محو شود و صدای موزیک افزایش یابد. و برعکس . تابع main ابتدا صدای تست را به صورت left و right میشکند تا کانال های مختلف صدا را جدا کند و بعد از آن عملیات predict را انجام دهد با استفاده از توابع مختلف که به تفکیک بررسی کردیم.

۲-۲- جزییات الگوریتم

کلاس windowing :

ابتدا wave و sample rate را دریافت کرده و صدا را شکسته به اندازه تعداد ثانیه ها

```
# Windowing is a class to analyse signal like stft with 'window size = 1 second'
# this class helps us to analysing every window seprately
class Windowing():
  def __init__(self, wave, sample_rate):
    self.wave = wave
    self.sample_rate = sample_rate
    self.windows = None
    self.windowing()
  def windowing(self):
    time_len = int(len(self.wave) / self.sample_rate)
    window_len = int(len(self.wave) / time_len)
    self.window = np.zeros((time_len, window_len))
    freqs, psds = [], []
    for i in range(time_len):
      self.window[i] = self.wave[i*window_len:(i+1)*window_len]
    return self.window
```

کلاس frequency :

این کلاس از کلاس windowing ارث بری میکند در نتیجه هرکجا این کلاس را صدا زدیم wave را که دریافت میکند صدا را به تعداد ثانیه ها میشکند. در تک تک پنجره های یک ثانیه ایی روی آن پنجره fft زدیم، و ضرایبی از fft که کوچکتر از cutoffهست را در یکی و ضرایبی که از fft بزرگ تر هست را در یکی دیگر میریزیم و در اصل همان عملیات filtering را انجام دادیم.

مقدار cut off را هم ۲۰۰۰ در نظر گرفتیم با توجه به اینکه صدای انسان بین ۵۰۰ تا ۲۰۰۰ هست بهترین کات اف همان ۲۰۰۰ بوده.

```python
# filter the frequencies and get the ratio
class frequency_filter(Windowing):
  def detect(self, cut_off=2000, coe=3):

    ratio = []

    for w in self.window:
      fr, fq = self.fft_feature(w)

      fr = np.abs(fr)

      high_fr = [fr[index] for index, f in enumerate(fq) if f < cut_off]
      low_fr  = [fr[index] for index, f in enumerate(fq) if f > cut_off]
      self.make_lists_equal_size(high_fr, low_fr)

      high_fr = np.array(high_fr)
      low_fr  = np.array(low_fr )


      high_fr = linear_normalize(high_fr, c=2, d=1000)
      low_fr  = linear_normalize(low_fr , c=2, d=1000) ** coe

      high_fr = np.array(high_fr)
      low_fr  = np.array(low_fr )

      ratio.append(np.mean(low_fr/high_fr))

    return np.array(ratio)
```

سپس عملیات نرمالسازی را انجام دادیم بین ۲ تا ۱۰۰۰ و سپس ضرایب low را به توان ۳ رساندیم (چون به صورت تجربی فهمیدم اگر ضرایب low را به توان یک عدد ثابت خوبی برسانم میتوانم بین high , low تمایز خوبی قائل شوم) به صورت تجربی، و ضرایب low پنجره را تقسیم به high پنجره کردیم و میانگین گرفتیم، در نتیجه برای هر پنجره یک ratio داریم.

Train() : ابتدا دو تا فایل speech و music که از قبل ایجاد کردیم را میگردد و سپس هرآنچه speech وجود دارد تمام ratio ها بدست می آورد و آن ها در X ذخیره میکند و به همه آن ها ضریب ۱ میدهد. برای مثال در یک فایل صدای انسان به طول ۳۰ ثانیه ما برای هر ۳۰ تا پنجره ratio بدست اوردیم و لیبل همه آن ها ۱ خواهد بود.

مجدد برای فایل دوم تکرار میشود تا تمام فایل های صدا ، برای فایل ها ی موسیقی هم همین تکرار میشود و به همه آن ها در نهایت لیبل ۰ میدهیم.

```python
def train(threshold=70000, coe=3):
  # X and y are train data
  X = []
  y = []

  voices = '/content/drive/MyDrive/Dataset/voiceAndspeech/speech/'
  for index, file in enumerate(os.listdir(voices)):
    # if index == 3: break
    if os.path.isfile(os.path.join(voices, file)):
      print(index, end=',')
      sr, wv = wavfile.read(voices+file)
      en = frequency_filter(wave=wv, sample_rate=sr).detect()
      X.append(en)
      y.append(np.ones(en.shape[0],))
      print(en)

  print('----------------------------------------------------------------------------------------------------')
  music = '/content/drive/MyDrive/Dataset/voiceAndspeech/music/'
  for index, file in enumerate(os.listdir(music)):
    # if index == 3: break
    if os.path.isfile(os.path.join(music, file)):
      print(index, end=',')
      sr, wv = wavfile.read(music+file)
      en = frequency_filter(wave=wv, sample_rate=sr).detect()
      X.append(en)
      y.append(np.zeros(en.shape[0],))
      print(en)

  print('')
```

یک الگوریتم ساده SGD نوشتم با سه حلقه for ساده در x ها میگردیم به طور تجربی عدد مناسبی که بتواند تمایز قائل شود بین ۵۰۰۰۰۰ تا ۸۰۰۰۰۰ هست چون نمیدانیم ترشولد مناسب کدام است استپ ۱۰۰۰ تایی گرفتیم تا امتیاز بدست آورد، برای مثال اگر یک x بدهیم به این تابع آیا میتواند بفهمد لیبل آن چند است یا نه، اگر توانست مشخص کند که کوچکتر از threshold است و موزیک هست و یا اینکه بزرگتر از threshold است و صدای انسان است و درست پیش بینی کردی آنگاه یک امتیاز مثبت میدهد به آن پارامتر، آنگاه اگر امتیاز پارامتر بالاترین امتیاز بود تو بهترین پارامتر هستی.

و درنهایت عدد ۵۰۲۰۰۰ با ۸۰ درصد موفقیت بدست آمده در نتیجه ترشولد نهایی برا این داده ها بدست آمده.

با مقایسه با این ترشولد متوجه میشود که اگر ratio کوچکتر از ۵۰۲۰۰۰ باشد موزیک تشخیص داده میشود اگر بزرگتر از آن باشد اسپیچ است با احتمال ۸۰ درصد.

```
# SGD algorithm to find the best threshold for classifying
total = 0
score = 0
best_score = 0
best_param = 0
# the upper and bottom limit of the threshold
bounds = [th for th in range(500000, 800000, 1000)]
for threshold in bounds:
    for i in range(len(X)):
        for j in range(len(X[i])):
            if (X[i][j] < threshold and y[i][j] == 0) or (X[i][j] > threshold and y[i][j] == 1):
                score += 1
            total += 1
    if score / total > best_score:
        best_param = threshold
        best_score = score / total
print(best_param)
print(best_score)
return best_param

BEST_THRESHOLD=train()
```

برای ساخت داده تست:

یک نمونه اسپیچ و یک نمونه موزیک را به دلخواه جدا میکنیم، ابتدا تمام صفر ها حذف میکنیم و مثلا یک میکنیم، چون میخواهیم صفر برای ما یک معنی خاصی داشته باشد. به هر دو فایل به یکی از ابتدا صفر اضافه میکنیم و به دیگری به انتهاش صفر اضافه میکنیم یک مقدار شیفت میدهیم هر دو فایل را ، سپس با creat_sterio_track() از دو فایل یک فایل به صورت استریو میسازیم و سپس به وسیله np.column_stack() آن ها را ادغام میکنیم.

```
# MIXER
def create_stereo_track(l, r, sr, output_path=None):

    min_length = min(len(l), len(r))
    l = l[:min_length]
    r = r[:min_length]

    stereo_audio = np.column_stack((l, r))
    if not output_path == None:
        wavfile.write(output_path, sr, stereo_audio)
    return stereo_audio
```

```python
def add_zero_to_first(input_file, output_file):
    # Load the original WAV file
    original_fs, original_audio = wavfile.read(input_file)
    original_audio = disable_zero(original_audio)

    # Calculate the number of zero samples to prepend (specified duration in seconds)
    duration = 10
    silence_samples = int(duration * original_fs)

    # Generate the zero samples for mono audio
    zero_samples = np.zeros(silence_samples, dtype=original_audio.dtype)

    # Concatenate the zero samples with the original audio
    modified_audio = np.concatenate((zero_samples, original_audio))

    # Write the modified audio to a new WAV file
    wavfile.write(output_file, original_fs, modified_audio)
```

تابع main () :

با استفاده از تابع split_and_save_channels دو تا کانال رو از input_file میگیریم و آن ها را در کانال های left و right ذخیره میکنیم، و در کانال اول چک میکند اولین کانال صفر است یا خیر، اگر با صفر شروع شده بود یعنی باند left ده ثانیه صفر خورده قبلش، اگر left ده ثانیه شیفت خورده بود حالا صفر های آن را حذف میکنیم تا آن ها را به estimation پاس بدهیم و تشخیص دهد که موزیک است یا اسپیچ.

```python
def delete_zeros(signal1, signal2):
    # signal one : zero at start
    # signal two : zero at end
    first_nonzero_index = np.argmax(signal1 != 0)
    trimmed_signal1 = signal1[first_nonzero_index:]
    last_nonzero_index = len(signal2) - np.argmax(signal2[::-1] != 0) - 1
    trimmed_signal2 = signal2[:last_nonzero_index + 1]
    return trimmed_signal1, trimmed_signal2
```

```python
# SPLITER
def split_and_save_channels(input_file, output_left=None, output_right=None):
    # Load the stereo audio file
    sr, y = wavfile.read(input_file)
    y = y.T
    # Extract left and right channels
    left_channel = y[0]
    right_channel = y[1]
    if not (output_left == None or output_right == None):
      # Save left and right channels as separate audio files
      wavfile.write(output_left, sr, left_channel)
      wavfile.write(output_right, sr, right_channel)
    return left_channel, right_channel, sr
def disable_zero(signal):
    for index, val in enumerate(signal):
        if val == 0: signal[index] = 1
    return signal
```

سپس تابع start_end_time دنبال صفر ها میگردد که زمان پایان و شروع هر دو کانال را متوجه شود.

باید ببیند چند ثانیه صفر خورده مثلا به اندازه ده ثانیه صفر خورده پس ابتدای شروع سیگنال میشود ۱۰، و سیگنال دوم از اول سیگنال شروع شده و انتهای آن منهای ده زمان پایان دوم بوده.

Trim_last() اگر left ابتدا شروع شده باشد، وقتی right وارد ماجرا میشود به صورت smooth باید حذف شود.

فایل خروجی را درنهایت میسازد، تمایز بین باند های left , right قائل میشود، باندی که اول شروع شده بود را حذف میکند در واقع.

تابع estimation ، سیگنال به ویندو ها شکسته میشود، مثلا اگر ۴۰ ثانیه باشد به اندازه ۴۰ تا ratio داریم، حالا باید بین ratio ها و threshold مقایسه کنیم، چون با احتمال ۸۰ درصد به ما احتمال موفقیت میدهد، یک مقایسه انجام میدهیم اگر تعداد یک ها بیشتر بود سیگنال ما speech خواهد بود ، اگر تعداد صفر ها بیشتر باشد پس موزیک خواهد بود.

در نهایت نتایج درون right_res و left_res ریخته میشود و این ها باهم ترکیب میشوند و فایل result را به ما میدهد.

```python
# estimate the signal is either the voice or music
def estimation(signal, sr):
  estimate_windows = frequency_filter(wave=signal, sample_rate=sr).detect()
  y_pred = np.array([-1 for i in range(len(estimate_windows))])

  for index, es in enumerate(estimate_windows):y_pred[index] = 1 if es > BEST_THRESHOLD else 0

  if np.sum(y_pred == 1) > np.sum(y_pred == 0):
    return 'voice'
  return 'music'

# trim the one that shows first when the second one get entered
def trim_last(pos0, pos1):
  for index in range(len(pos0)):
    if not pos1[index] == 0:
      pos0[index] = 0
  return pos0, pos1

# find the start and the end time of every band
def start_end_time(pos0, pos1, sr):
  first_nonzero_index = np.argmax(pos1 != 0)
  pos1_time = [first_nonzero_index / sr, len(pos1) / sr]

  last_nonzero_index = len(pos0) - np.argmax(pos0[::-1] != 0) - 1
  pos0_time = [0, last_nonzero_index / sr]

  return pos0_time, pos1_time
```

```python
def main():
  # get left, right and sample rate of two band audio
  left, right, sr = split_and_save_channels('/content/drive/MyDrive/Dataset/voiceAndspeech/test/zz.wav')


  if left[0] == 0:# if left channel is entered after the right
    left_tr, right_tr = delete_zeros(left, right) # delete zero from the start and end of two channel
    right_time, left_time = start_end_time(right, left, sr) # find what time every band started and ended
    right_res, left_res = trim_last(right, left) # trim the first one when the second one entered
  else:# if right channel is entered after the left
    right_tr, left_tr = delete_zeros(right, left)
    left_time, right_time = start_end_time(left, right, sr)
    left_res, right_res = trim_last(left, right)

  print(estimation(left_tr, sr), end=': ') # estimate what is the left channel
  print(left_time)
  print(estimation(right_tr, sr), end=': ') # estimate what is the right channel
  print(right_time)

  create_stereo_track(left_res, right_res, sr, 'res.wav') # output

main()
```

پیوست

در پایان یک سری ویژگی های مختلف از سیگنال ها را بدست آوردم، برای تشخیص بهتر میتوانیم از ترکیبی از ویژگی های زیر استفاده کنیم، توسط یک کلسیفایر خوب، اما بدلیل اینکه نتیجه فعلی نتیجه خوبی بود و تاکیید روی صرفا یک فیچر متمایز کننده بود از توابع زیر دیگر استفاده نکردم، و میتوانیم با گسترش این پروژه بعدا پروژه قوی تری برای این کار داشته باشیم.

```python
# high pass and low pass filters which return the data with respect to their frequencies
def apply_high_pass_filter(data, sample_rate, cutoff_frequency):
    num_taps = 15
    high_pass_filter = firwin(num_taps, cutoff_frequency, pass_zero=False, fs=sample_rate, window='hamming')

    filtered_data = lfilter(high_pass_filter, 1.0, data)

    return filtered_data

def apply_low_pass_filter(data, sample_rate, cutoff_frequency):
    num_taps = 15
    low_pass_filter = firwin(num_taps, cutoff_frequency, fs=sample_rate, window='hamming')

    filtered_data = lfilter(low_pass_filter, 1.0, data)

    return filtered_data
```

```python
# calculate the PSD
class PSD(Windowing):
  def detect(self):
    freqs, psds = [], []
    for i in range(len(self.window)):
      frequencies, psd = welch(self.window[i], fs=self.sample_rate, nperseg=1024)
      freqs.append(frequencies)
      psds.append(psd)
    return freqs, psds

# filter the data with respcet to their frequency and get their PSD ratio
class PSD_ratio(PSD):
  def detect_(self):
    self.window = np.array([apply_high_pass_filter(wave, self.sample_rate, 3000) for wave in self.window])
    _, high_psd = self.detect()
    self.window = np.array([apply_low_pass_filter (wave, self.sample_rate, 3000) for wave in self.window])
    _, low_psd  = self.detect()
    return np.min(np.array(low_psd)/np.array(high_psd), axis=1)

# filter the data with respcet to their frequency and get their ratio
class energy_ratio(Windowing):
 def detect(self):
    self.window = np.array([apply_high_pass_filter(wave, self.sample_rate, 3000) for wave in self.window])
    high_psd = np.array(self.window ** 2).copy()
    self.window = np.array([apply_low_pass_filter (wave, self.sample_rate, 3000) for wave in self.window])
    low_psd = np.array(self.window ** 2).copy()
    return np.min(np.array(low_psd)/np.array(high_psd), axis=1)
```

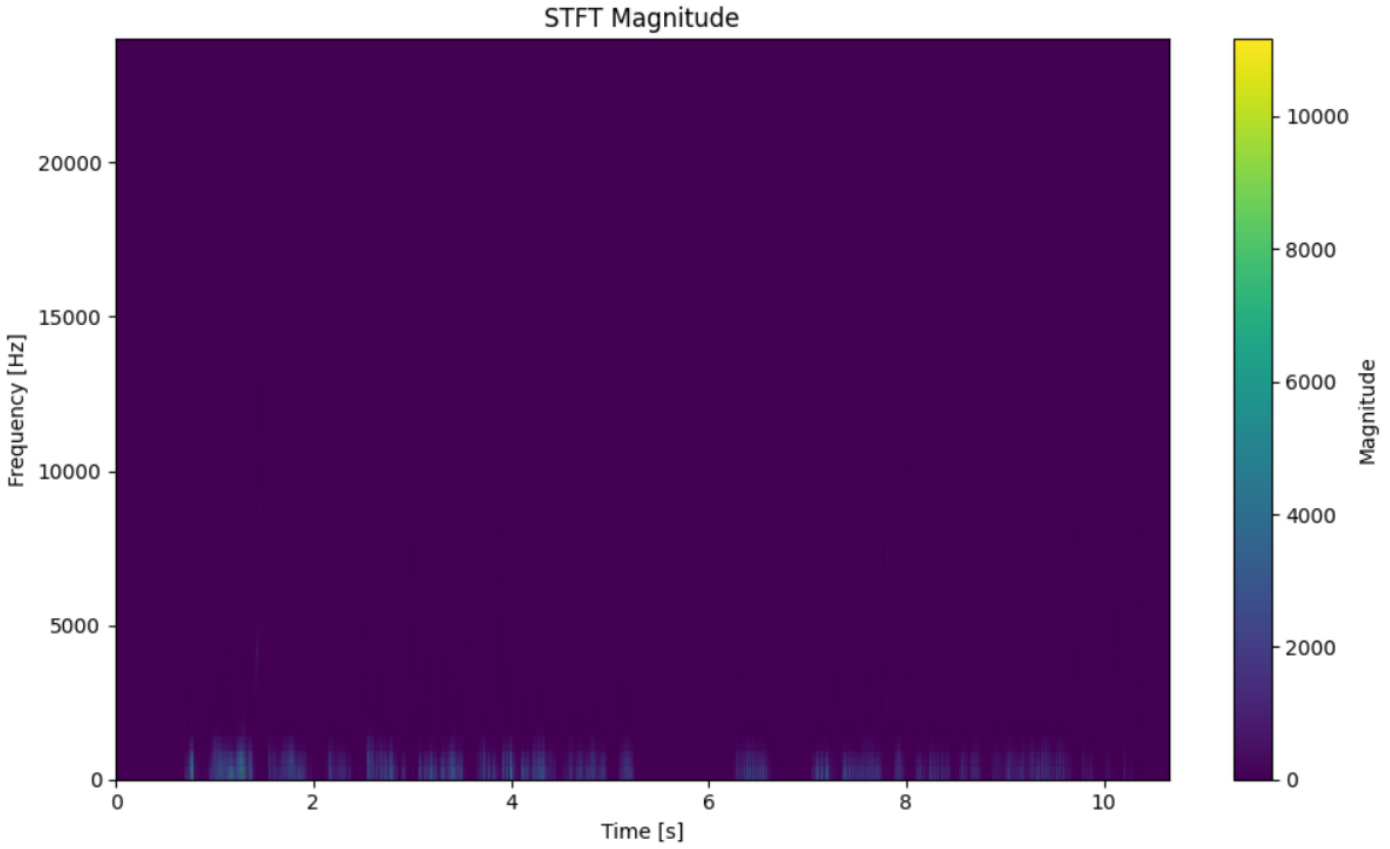
با استفاده از STFT روی فایل های مختلف صوت و موسیقی و کد زیر به تفاوت فاحش آن ها نیز رسیدم.

```python
def stftt(wave, sample_rate):
  nperseg = 100  # Length of each segment
  noverlap = 50  # Overlap between segments
  window = 'hann'  # Windowing function

  # Compute STFT
  f, t, Zxx = stft(wave, fs=sample_rate, nperseg=nperseg, noverlap=noverlap, window=window)
  plt.figure(figsize=(10, 6))
  plt.pcolormesh(t, f, np.abs(Zxx), shading='gouraud')
  plt.title('STFT Magnitude')
  plt.xlabel('Time [s]')
  plt.ylabel('Frequency [Hz]')
  plt.colorbar(label='Magnitude')
  plt.tight_layout()
  plt.show()
```

```
print("all voices STFT")
voices = '/content/drive/MyDrive/Dataset/voiceAndspeech/speech/'
for file in os.listdir(voices):
  if os.path.isfile(os.path.join(voices, file)):
    print(file)
    sr, wv = wavfile.read(voices+file)
    en = stftt(wave=wv, sample_rate=sr)
print("all musics STFT")
print('-----------------------------------------------------------------------------------------------')
musics = '/content/drive/MyDrive/Dataset/voiceAndspeech/music/'
for file in os.listdir(musics):
  if os.path.isfile(os.path.join(musics, file)):
    print(file)
    sr, wv = wavfile.read(musics+file)
    en = stftt(wave=wv, sample_rate=sr)
```

نمونه سیگنال اسپیچ :



نمونه سیگنال موسیقی:

```
all musics STFT
--------------------------------------------------------------------------------
audio_2024-02-06_11-51-16.wav
```