feature_extractor

August 7, 2017

1 Feature Extractor

This is a helper module which defines some functions to extract features from a given audio file.
Each function has a little desctiption above it and also is described in its own docstring.
Module docstring and imports:

```
In [ ]: """
        This module is made to extract features from a given audio file.
        You can also extract them manually using helper functions in here.
        """

        import numpy as np
```

1.1 Extract frames

To extract features from an audio file the first step is to extract frames.

This function gets a signal (array of integers that show intensity at that sample), and returns a list of frames with given step size and frame length. note that the output frames will overlap and the amount of samples which they overlap is `frame_length - step`.

This function is not exported because it will be used in the next function.

```
In [ ]: def _get_frames(signal, step, frame_length):
            """
            This function is used to divide a given signal into
            frames with given length and given step size.
            This also appends a zero frame to the end of frames.

            Parameters
            ----------
            signal : array_like
                this is the signal that will be divided
                into multiple frames.
            step : int
                this specifies the step size i.e. the number of
                samples between start points of two
                consecutive frames.
            frame_length : int
                length of each frame i.e. the number of
```

```
        samples in each frame.

    Returns
    -------
    list
        list of frames.
    """
    max_start = int((len(signal) - frame_length) / step) + 1
    window = [signal[i * step:i * step + frame_length]
              for i in range(max_start)]
    last = signal[max_start * step:]
    window.append(np.pad(
        last,
        (0, frame_length - len(last)),
        'constant'
    ))
    window.append(np.zeros(frame_length))

    return window
```

Next functions is used to get frames from a given file. It uses the previously defined function and gets input in seconds rather than number of samples (which depends on sampling rate).

This function is exported.

```
In [ ]: def get_frames_from_file(filename,
                                 step=0.01,
                                 frame_length=0.025):
    """
    This function extracts frames from a given file.
    Uses scipy to read wav file.
    This function also adds a zero frame to the
    end of file and applys hamming window to all frames.

    Parameters
    ----------
    filename : string
        name of the audio file.
    step : float
        step size in seconds i.e. the difference between
        start points of two consecutive frames
        in seconds (default: 10 ms).
    frame_length : float
        length of each frame in seconds (default: 25 ms).

    Returns
    -------
    list
        list of frames.
```

```
        int
            sampling rate of the input file.
        """
        import scipy.io.wavfile
        # read file
        [fs, y] = scipy.io.wavfile.read(filename)
        # divide the signal to frames and apply
        # the hamming window function on them
        frames = []
        for f in _get_frames(y, int(fs * step), int(fs * frame_length)):
            frames.append(f * np.hamming(len(f)))
        return frames, fs
```

Now we define some helper functions to convert Mels to Hertz and vice versa.
To convert from frequency to Mel:

$$M(f) = 1125 \ln\left(1 + \frac{f}{700}\right)$$

And to convert from Mel to frequency:

$$M^{-1}(m) = 700 \times \left(e^{\frac{m}{1125}} - 1\right)$$

These functions are not exported because they will be used in the next function

In [ ]:
```
def _hz2mel(hz):
    """
    This function converts frequency to Mel scale.
    Supports numpy arrays.

    Parameters
    ----------
    hz : {float, array_like}
        input(s) in hertz.

    Returns
    -------
    {float, array_like}
        converted value of input.
        If input was array will return an array of same size.
    """
    return 1125 * np.log(1 + (hz / 700.))

def _mel2hz(mel):
    """
    This function converts Mel scale to frequency.
    Supports numpy arrays.

    Parameters
```

```
    ----------
    hz : {float, array_like}
        input(s) in Mel scale.

    Returns
    -------
    {float, array_like}
        converted value of input.
        If input was array will return an array of same size.
    """
    return 700 * (np.e ** (mel / 1125.) - 1)
```

This function is used to calculate Mel filter banks. To compute the filter bank: 1. Pick some low and high frequencies (we choose 300Hz and 8KHz). 2. Calculate Mel points (we choose to calculate 26 of them). 3. Convert Mels back to Hertz. 4. Round the resulting points (called Mel pints). 5. Create the filterbanks.The first filterbank will start at the first point, reach its peak at the second point, then return to zero at the 3rd point. The second filterbank will start at the 2nd point, reach its max at the 3rd, then be zero at the 4th etc. A formula for calculating these is as follows:

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

where $ M $ is the number of filters we want, and $ f $ is the list of $ M + 2 $ Mel-spaced frequencies.

```
In [ ]: def get_mel_filterbanks(n=26,
                                nfft=512,
                                samplerate=16000,
                                low_frequency=300,
                                high_frequency=8000): # step 1
    """
    Calculates Mel filter banks.

    Parameters
    ----------
    n : int
        number of filterbanks returned.
    nfft : int
        lenghth of fft output.
    samplerate : int
        sampling rate of the audio file
        (default 16 KHz).
    low_frequency : int
        starting frequency for filterbanks
        (default 300).
    high_frequency : int
```

```
        high frequency for filter banks.
        This can't be more than samplerates / 2.

    Returns
    -------
    list
        list of Mel filter banks.
        length of this list is n (the first parameter).
    """
    if high_frequency > samplerate // 2:
        high_frequency = samplerate // 2

    mel = np.linspace(
        _hz2mel(low_frequency),
        _hz2mel(high_frequency),
        n + 2
    ) # step 2
    hertz = _mel2hz(mel) # step 3
    fftbin = np.floor((nfft + 1) * hertz / samplerate)
                .astype(np.int64) # step 4

    fbank = np.zeros([n, (nfft // 2) + 1]) # step 5
    for j in range(n):
        for i in range(fftbin[j], fftbin[j+1]):
            fbank[j, i] = (i - fftbin[j]) /
                          (fftbin[j + 1] - fftbin[j])
        for i in range(fftbin[j + 1], fftbin[j + 2]):
            fbank[j, i] = (fftbin[j + 2] - i) /
                          (fftbin[j + 2] - fftbin[j + 1])

    return fbank
```

Now, we take a glimpse at the steps to compute MFCCs:

1. Frame the signal into short frames (already done).
2. For each frame calculate the periodogram estimate of the power spectrum.

Suppose that we show our original signal of frame $ i $ with notation $ s_i(n) $ (n being the number of sample). When we calculate the complex DFT we get $ S_i(k) $.

$$S_i(k) = \sum_{n=1}^{N} s_i(n) h(n) e^{\frac{-j2\pi kn}{N}}$$

where $ h(n) $ is an $ N $ sample long analysis window (e.g. hamming window), and $ K $ is the length of the DFT. The periodogram-based power spectral estimate for the speech frame $ s_i(n) $ is given by:

$$P_i(k) = \frac{1}{N} |S_i(k)|^2$$

This is called the Periodogram estimate of the power spectrum. We take the absolute value of the complex fourier transform, and square the result. We would generally perform a 512 point FFT and keep only the first 257 coefficents. 3. Apply the Mel filterbank to the power spectra, sum the energy in each filter. 4. Take the logarithm of all filterbank energies. 5. Take the DCT of the log filterbank energies. 6. Keep DCT coefficients 2-13, discard the rest. 7. Calculate frame energy. To do that, simply sum the power spectrum (because power spectrum is actually energy of the signal in different frequencies).

```
In [ ]: def mfcc(signal,
                 nfft=512,
                 samplerate=16000,
                 nfilt=26,
                 mel_low=300,
                 mel_high=8000):
            """
            This function extracts Mel frequency cepstral
            coefficients from the given signal.
            This signal must be a list of extracted frames.

            Parameters
            ----------
            signal : array_like
                array of frames (each frame is an array itself).
            nfft : int
                number of fft output (default 512).
            samplerate : int
                sampling rate of input signal (default 16000).
            nfilt : int
                number of filters in Mel filter banks
                (default 26).
            mel_low : int
                starting frequency of Mel filter banks
                (default 300 Hz).
            mel_high : int
                high frequency of Mel filter banks (default 8 KHz).

            Returns
            -------
            array
                returns an array of extracted features.
                Length of array is the same as length of
                input (number of frames).
                Each feature vector consists of 12 MFCCs
                and energy of that frame. So each feature
                vector is of length 13.
            """
            from scipy.fftpack import dct
            # already multiplied signal by hamming window
```

```
    # so all we need to to do is to call fft
    magspec = np.absolute(np.fft.rfft(signal, nfft))
    powspec = (1. / len(signal)) * np.square(magspec) # step 2

    fbank = get_mel_filterbanks(nfilt,
                                nfft,
                                samplerate,
                                mel_low,
                                mel_high)

    ft = np.dot(powspec, fbank.T) # step 3
    # if ft is zero, we get problems with log
    ft = np.where(ft == 0, np.finfo(float).eps, ft)

    ft = np.log(ft) # step 4

    ft = dct(ft, norm='ortho')[:,:12] # steps 5 and 6

    energy = np.sum(powspec, 1) # step 7
    # if energy is zero, we get problems with log
    energy = np.where(energy == 0,np.finfo(float).eps,energy)

    ft = [np.append(ft[i], energy[i]) for i in range(len(energy))]

    return np.array(ft)
```

For simplicity, we define yet another function that gets a filename and extracts features and returns them. This is used in the end, when we want to test our ASR.

```
In [1]: def extract(filename,
                    frame_step=0.01,
                    frame_length=0.025,
                    nfft=512,
                    nfilt=26,
                    mel_low=300,
                    mel_high=8000):
    """
    This function extracts Mel frequency
    cepstral coefficients from the given signal.
    This signal must be a list of extracted frames.

    Parameters
    ----------
    filename : string
        name of the file.
    frame_step : float
        step size in seconds i.e. the difference between
        start points of two consecutive frames in seconds
```

```
    (default: 10 ms).
frame_length : float
    length of each frame in seconds (default: 25 ms).
nfft : int
    number of fft output (default 512).
nfilt : int
    number of filters in Mel filter banks (default 26).
mel_low : int
    starting frequency of Mel filter banks
    (default 300 Hz).
mel_high : int
    high frequency of Mel filter banks (default 8 KHz).

Returns
-------
array
    returns an array of extracted features.
    Length of array is the the number of
    frame in input file.
    Each feature vector consists of 12 MFCCs
    and energy of that frame. So each feature
    vector is of length 13.
"""
signal = get_frames_from_file(filename,
                              frame_step,
                              frame_length)
return mfcc(signal, nfft, nfilt, mel_low, mel_high)
```