Documentation
Audio Steganography Techniques

Sneha Mohanty (120799)

MEL-FREQUENCY CEPSTRAL COEFFICIENTS :

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum").

librosa.feature.mfcc(y=None, sr=22050, S=None, nmfcc=20, dcttype=2, norm='ortho', lifter=0, **kwargs):

Mel-frequency cepstral coefficients (MFCCs)

Parameters:

y : np.ndarray [shape=(n,)] or None

audio time series

sr : number > 0 [scalar]

sampling rate of `y`

S : np.ndarray [shape=(d, t)] or None

log-power Mel spectrogram

n_mfcc: int > 0 [scalar]

number of MFCCs to return

dct_type : {1, 2, 3}

Discrete cosine transform (DCT) type. By default, DCT type-2 is used.

norm : None or 'ortho'

If `dct_type` is *2 or 3*, setting `norm='ortho'` uses an ortho-normal DCT basis.

Normalization is not supported for `dct_type=1`.

lifter : number >= 0

If `lifter>0`, apply *liftering* (cepstral filtering) to the MFCCs:

```
M[n, :] <- M[n, :] * (1 + sin(pi * (n + 1) / lifter)) * lifter / 2
```

Setting `lifter >= 2 * n_mfcc` emphasizes the higher-order coefficients. As `lifter` increases, the coefficient weighting becomes approximately linear.

kwargs : additional keyword arguments

Arguments to `melspectrogram`, if operating on time series input

Returns:

M : np.ndarray [shape=(n_mfcc, t)]

MFCC sequence

Figure 1: librosa feature mfcc [1]

PREPROCESSING DATA:

The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

sklearn.preprocessing.scale(X, *, axis=0, withmean=True, withstd=True, copy=True):

Standardize a dataset along any axis. Center to the mean and component wise scale to unit variance.

Parameters:	**X : *{array-like, sparse matrix} of shape (n_samples, n_features)*** The data to center and scale. **axis : *int, default=0*** axis used to compute the means and standard deviations along. If 0, independently standardize each feature, otherwise (if 1) standardize each sample. **with_mean : *bool, default=True*** If True, center the data before scaling. **with_std : *bool, default=True*** If True, scale the data to unit variance (or equivalently, unit standard deviation). **copy : *bool, default=True*** set to False to perform inplace row normalization and avoid a copy (if the input is already a numpy array or a scipy.sparse CSC matrix and if axis is 1).
Returns:	**X_tr : *{ndarray, sparse matrix} of shape (n_samples, n_features)*** The transformed data.

Figure 2: sklearn preprocessing scale method [2]

Process of acquiring MFCC from a spectrogram :

1. The process of acquiring MFCCs from a spectrogram is illustrated in figure 3 below, where there is a triangular filterbank placed at linear steps on the mel-frequency scale.

2. Figure 4 shows the spectrogram of a speech segment.

3. When each window of that spectrogram is multiplied with the triangular filterbank, we obtain the mel-weighted spectrum, illustrated in the figure 5 .Here we see that the gross-shape of the spectrogram is retained, but the fine-structure has been smoothed out. In essence, this process thus removes the details related to the harmonic structure. Since the identity of phonemes such as vowels is determined based on macro-shapes in the spectrum, the MFCCs thus preserve that type of information and remove "unrelated" information such as the pitch.

4. The figure 6 illustrates the outcome once the mel-weighted spectrogram is multiplied with a DCT to obtain the final MFCCs. Where the mel-weighted spectrogram does retain the original shape of the spectrum, the MFCCs do not offer such easy interpretations. It is an abstract domain, which contains information about the spectral envelope of the speech signal.

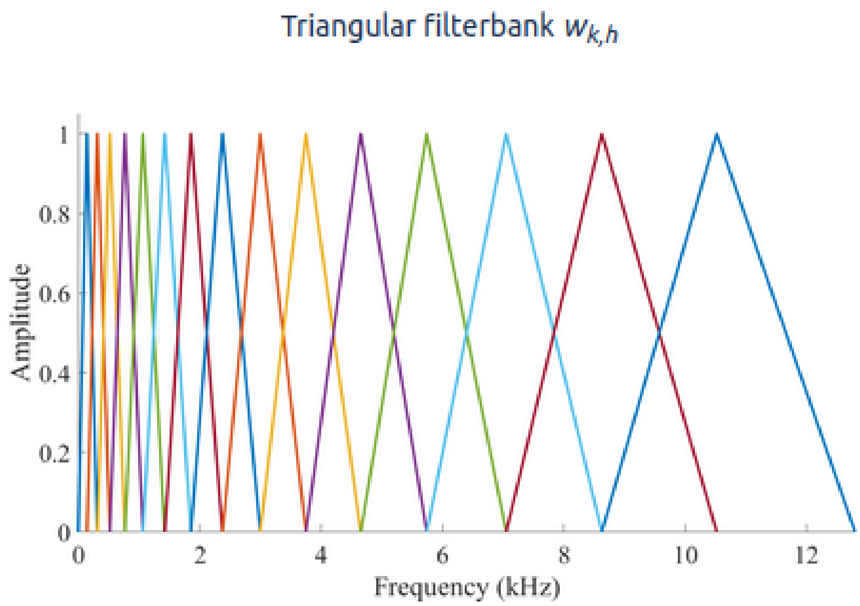


Figure 3: Triangular filterbank

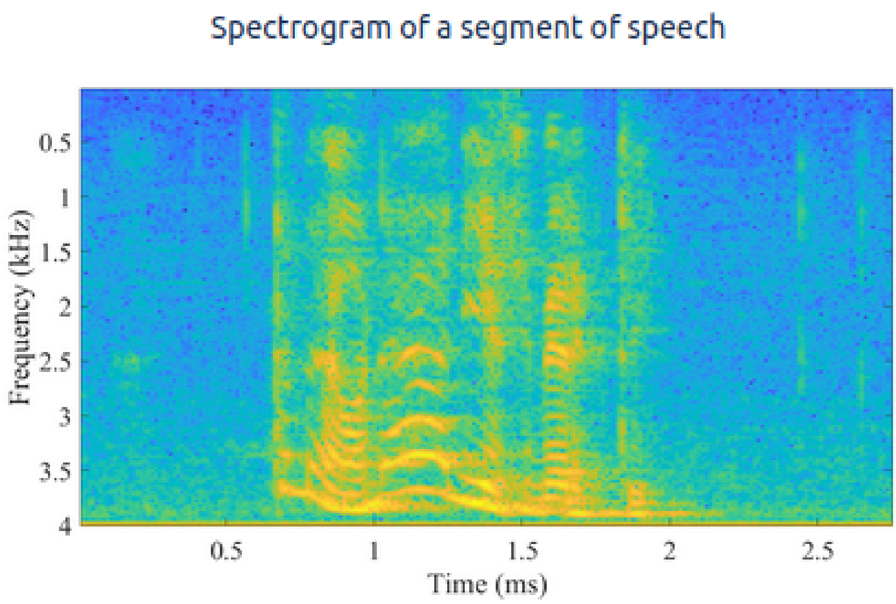


Figure 4: Spectrum of speech segment

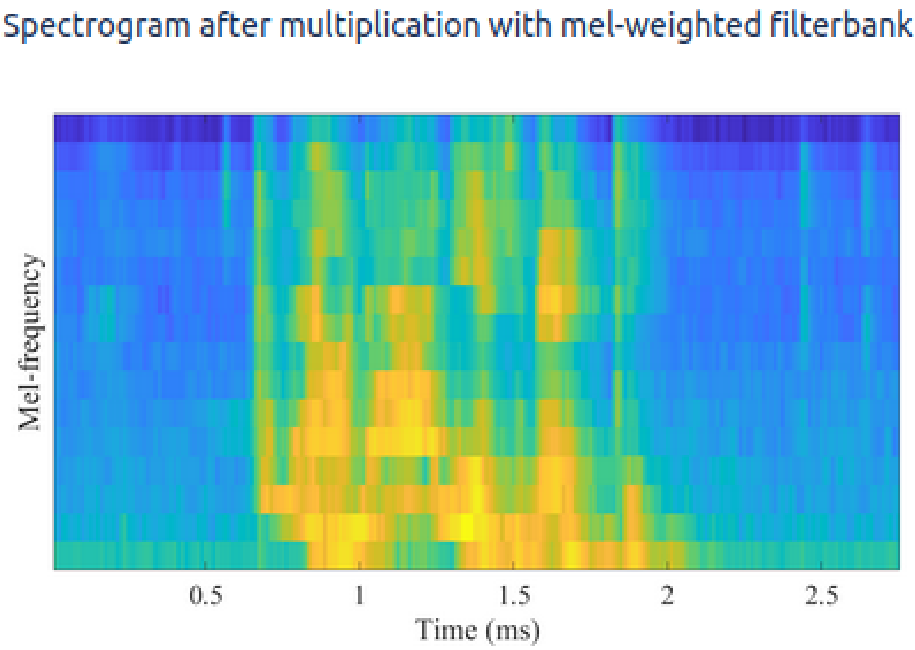


Figure 5: Mel-weighted spectrum

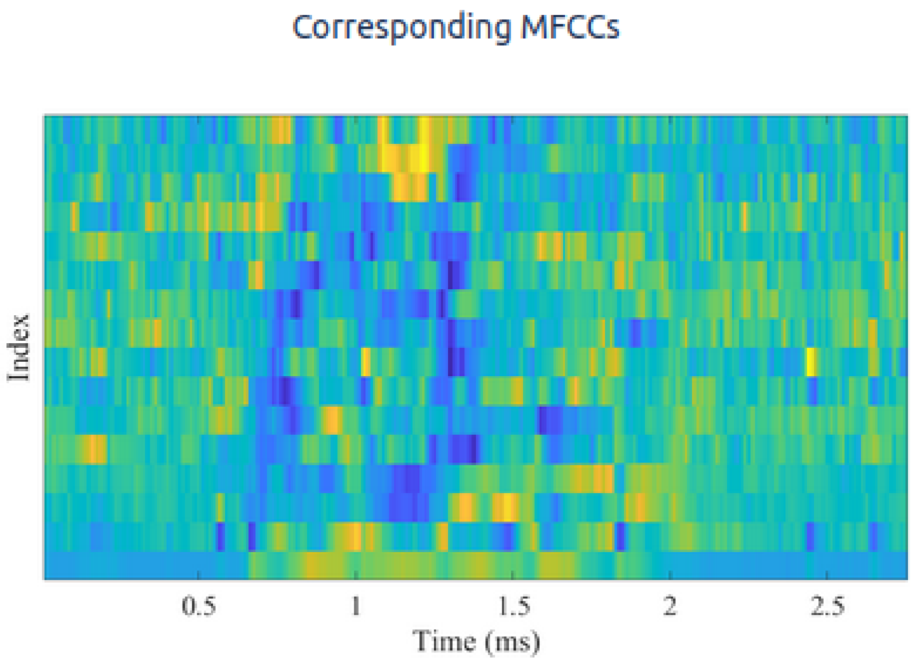
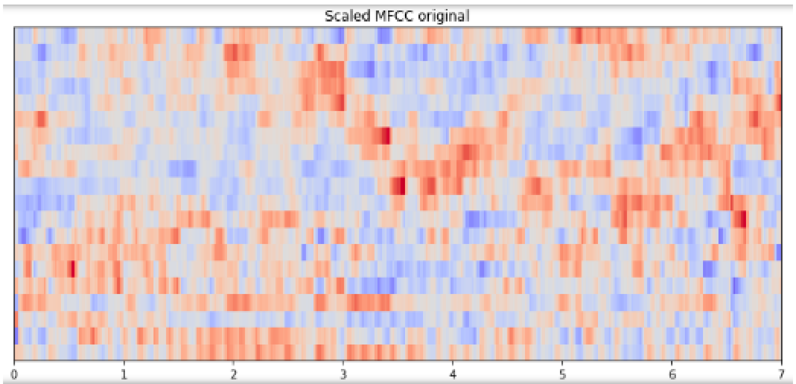
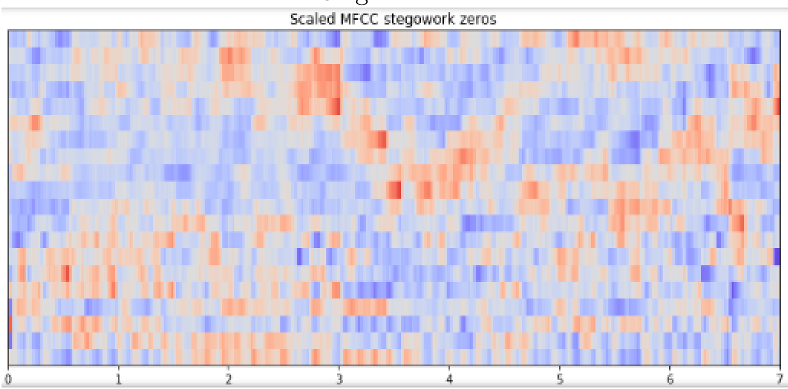


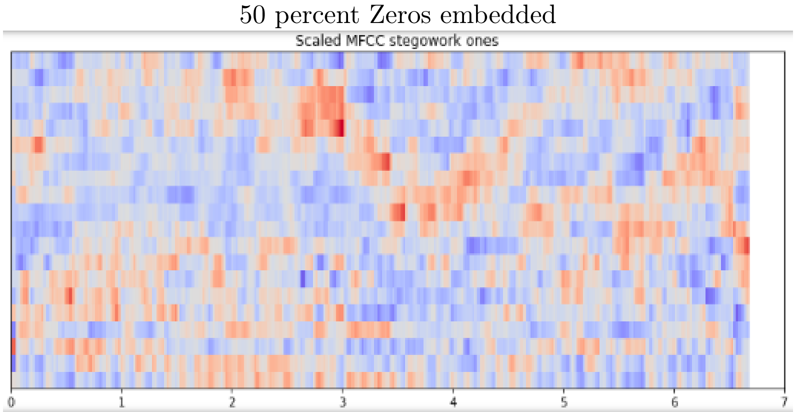
Figure 6: Final MFCCs



Original work



50 percent Zeros embedded



Ones embedded stegowork

Figure 7: Scealed MFCCs from Dataset

`librosa.zero_crossings`(*y, threshold=1e-10, ref_magnitude=None, pad=True, zero_pos=True, axis=- 1*) [source]

Find the zero-crossings of a signal `y`: indices `i` such that `sign(y[i]) != sign(y[j])`.

If `y` is multi-dimensional, then zero-crossings are computed along the specified `axis`.

Parameters:

y : np.ndarray

The input array

threshold : float > 0 or None

If specified, values where `-threshold <= y <= threshold` are clipped to 0.

ref_magnitude : float > 0 or callable

If numeric, the threshold is scaled relative to `ref_magnitude`.

If callable, the threshold is scaled relative to `ref_magnitude(np.abs(y))`.

pad : boolean

If `True`, then `y[0]` is considered a valid zero-crossing.

zero_pos : boolean

If `True` then the value 0 is interpreted as having positive sign.

If `False`, then 0, -1, and +1 all have distinct signs.

axis : int

Axis along which to compute zero-crossings.

Returns:

zero_crossings : np.ndarray [shape=y.shape, dtype=boolean]

Indicator array of zero-crossings in `y` along the selected axis.

Figure 8: librosa zero crossings [3]

librosa.feature.spectral_centroid(*y=None, sr=22050, S=None, n_fft=2048, hop_length=512, freq=None, win_length=None, window='hann', center=True, pad_mode='reflect'*) [source]

Compute the spectral centroid.

Each frame of a magnitude spectrogram is normalized and treated as a distribution over frequency bins, from which the mean (centroid) is extracted per frame.

More precisely, the centroid at frame `t` is defined as [1]:

```
centroid[t] = sum_k S[k, t] * freq[k] / (sum_j S[j, t])
```

where `S` is a magnitude spectrogram, and `freq` is the array of frequencies (e.g., FFT frequencies in Hz) of the rows of `S`.

[1] Klapuri, A., & Davy, M. (Eds.). (2007). Signal processing methods for music transcription, chapter 5. Springer Science & Business Media.

Parameters:

y : np.ndarray [shape=(n,)] or None

audio time series

sr : number > 0 [scalar]

audio sampling rate of `y`

S : np.ndarray [shape=(d, t)] or None

(optional) spectrogram magnitude

n_fft : int > 0 [scalar]

FFT window size

hop_length : int > 0 [scalar]

hop length for STFT. See `librosa.stft` for details.

freq : None or np.ndarray [shape=(d,) or shape=(d, t)]

Center frequencies for spectrogram bins. If *None*, then FFT bin center frequencies are used.

Otherwise, it can be a single array of `d` center frequencies, or a matrix of center frequencies as constructed by `librosa.reassigned_spectrogram`

win_length : int <= n_fft [scalar]

Each frame of audio is windowed by *window()*. The window will be of length `win_length` and then padded with zeros to match `n_fft`.

If unspecified, defaults to `win_length = n_fft`.

window : string, tuple, number, function, or np.ndarray [shape=(n_fft,)]

- a window specification (string, tuple, or number); see `scipy.signal.get_window`
- a window function, such as `scipy.signal.windows.hann`
- a vector or array of length `n_fft`

center : boolean

- If *True*, the signal `y` is padded so that frame *t* is centered at `y[t * hop_length]`.
- If *False*, then frame `t` begins at `y[t * hop_length]`

pad_mode : string

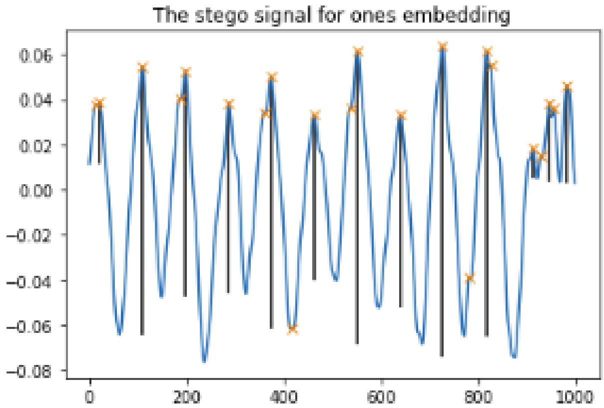
If `center=True`, the padding mode to use at the edges of the signal. By default, STFT uses reflection padding.

Returns:

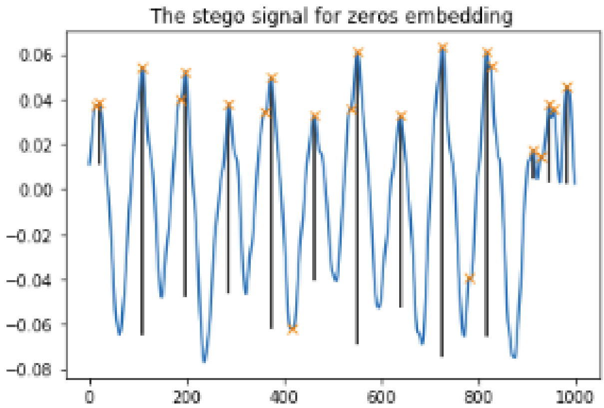
centroid : np.ndarray [shape=(1, t)]

centroid frequencies

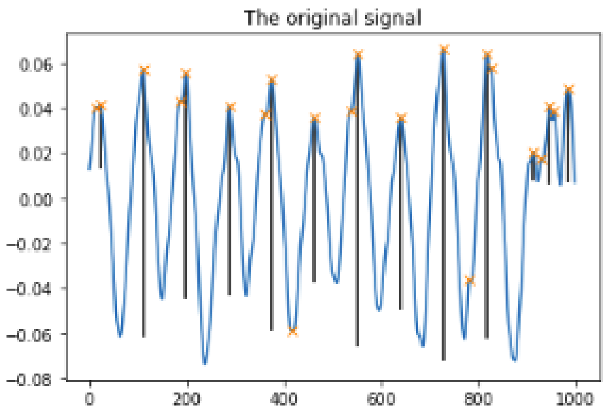
Figure 9: librosa feature spectral centroid [4]



[14 21 110 186 197 287 361 374 417 462 537 551 640 727 782 817 826 913
 928 946 955 983]



[14 21 110 186 197 287 361 374 417 462 537 551 640 727 782 817 826 913
 928 946 955 983]



[15 22 111 187 198 288 362 375 418 463 538 552 641 728 783 818 827 914
 929 947 956 984]

Figure 10: Peaks and Prominences from Dataset