Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between

Apr 21, 2016

Speech processing plays an important role in any speech system whether its Automatic Speech Recognition (ASR) or speaker recognition or something else. Mel-Frequency Cepstral Coefficients (MFCCs) were very popular features for a long time; but more recently, filter banks are becoming increasingly popular. In this post, I will discuss filter banks and MFCCs and why are filter banks becoming increasingly popular.

Computing filter banks and MFCCs involve somewhat the same procedure, where in both cases filter banks are computed and with a few more extra steps MFCCs can be obtained. In a nutshell, a signal goes through a pre-emphasis filter; then gets sliced into (overlapping) frames and a window function is applied to each frame; afterwards, we do a Fourier transform on each frame (or more specifically a Short-Time Fourier Transform) and calculate the power spectrum; and subsequently compute the filter banks. To obtain MFCCs, a Discrete Cosine Transform (DCT) is applied to the filter banks retaining a number of the resulting coefficients while the rest are discarded. A final step in both cases, is mean normalization.

Setup

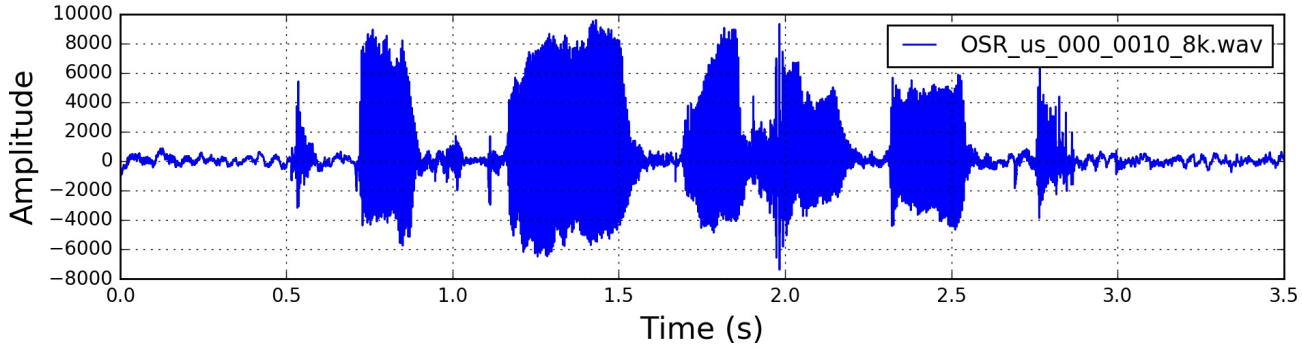
For this post, I used a 16-bit PCM wav file from here, called "OSR_us_000_0010_8k.wav", which has a sampling frequency of 8000 Hz. The wav file is a clean speech signal comprising a single voice uttering some sentences with some pauses in-between. For simplicity, I used the first 3.5 seconds of the signal which corresponds roughly to the first sentence in the wav file.

I'll be using Python 2.7.x, NumPy and SciPy. Some of the code used in this post is based on code available in this repository.

```
import numpy
import scipy.io.wavfile
from scipy.fftpack import dct
```

```
sample_rate, signal = scipy.io.wavfile.read('OSR_us_000_0010_8k.wav')
signal = signal[0:int(3.5 * sample_rate)]  # Keep the first 3.5 seconds
```

The raw signal has the following form in the time domain:



Signal in the Time Domain

Pre-Emphasis

The first step is to apply a pre-emphasis filter on the signal to amplify the high frequencies. A pre-emphasis filter is useful in several ways: (1) balance the frequency spectrum since high frequencies usually have smaller magnitudes compared to lower frequencies, (2) avoid numerical problems during the Fourier transform operation and (3) may also improve the Signal-to-Noise Ratio (SNR).

The pre-emphasis filter can be applied to a signal x using the first order filter in the following equation:

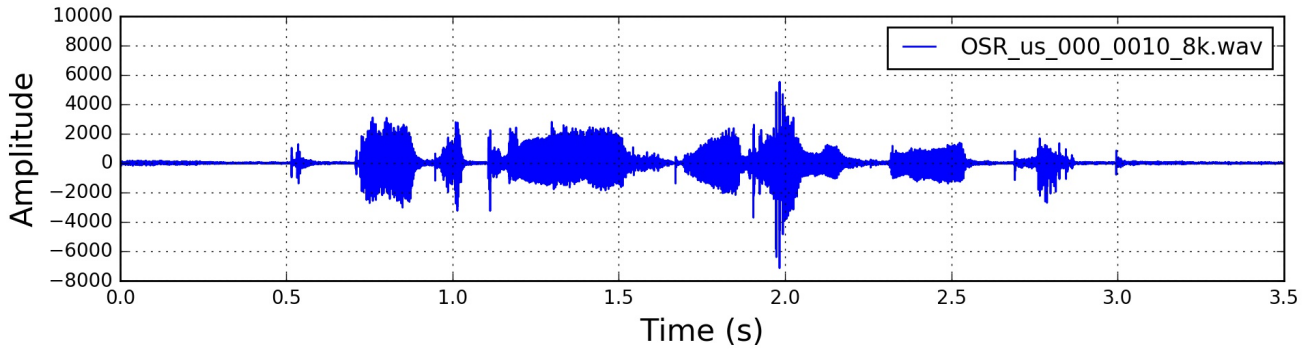
$$y(t) = x(t) - \alpha x(t-1)$$

which can be easily implemented using the following line, where typical values for the filter coefficient (α) are 0.95 or 0.97, `pre_emphasis = 0.97`:

```
emphasized_signal = numpy.append(signal[0], signal[1:] - pre_emphasis *
```

Pre-emphasis has a modest effect in modern systems, mainly because most of the motivations for the pre-emphasis filter can be achieved using mean normalization (discussed later in this post) except for avoiding the Fourier transform numerical issues which should not be a problem in modern FFT implementations.

The signal after pre-emphasis has the following form in the time domain:



Signal in the Time Domain after Pre-Emphasis

Framing

After pre-emphasis, we need to split the signal into short-time frames. The rationale behind this step is that frequencies in a signal change over time, so in most cases it doesn't make sense to do the Fourier transform across the entire signal in that we would lose the frequency contours of the signal over time. To avoid that, we can safely assume that frequencies in a signal are stationary over a very short period of time. Therefore, by doing a Fourier transform over this short-time frame, we can obtain a good approximation of the frequency contours of the signal by concatenating adjacent frames.

Typical frame sizes in speech processing range from 20 ms to 40 ms with 50% (+/-10%) overlap between consecutive frames. Popular settings are 25 ms for the frame size, `frame_size = 0.025` and a 10 ms stride (15 ms overlap), `frame_stride = 0.01`.

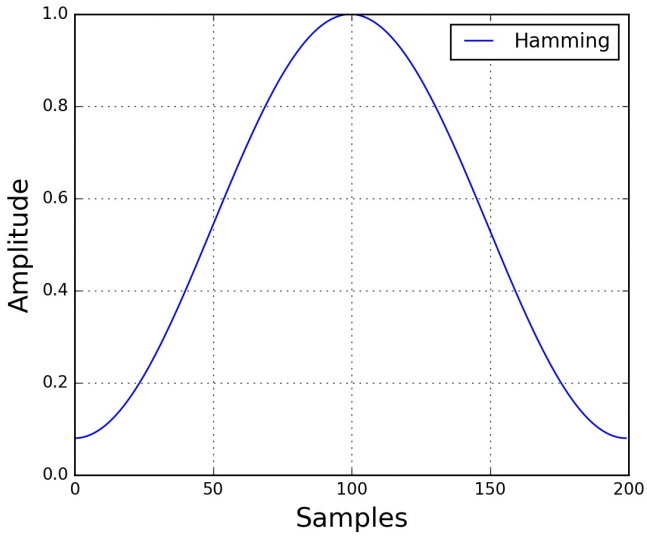
```
frame_length, frame_step = frame_size * sample_rate, frame_stride * sam
signal_length = len(emphasized_signal)
frame_length = int(round(frame_length))
frame_step = int(round(frame_step))
num_frames = int(numpy.ceil(float(numpy.abs(signal_length - frame_lengt

pad_signal_length = num_frames * frame_step + frame_length
z = numpy.zeros((pad_signal_length - signal_length))
pad_signal = numpy.append(emphasized_signal, z) # Pad Signal to make su

indices = numpy.tile(numpy.arange(0, frame_length), (num_frames, 1)) + 
frames = pad_signal[indices.astype(numpy.int32, copy=False)]
```

Window

After slicing the signal into frames, we apply a window function such as the Hamming window to each frame. A Hamming window has the following form:

$$w[n] = 0.54 - 0.46cos(\frac{2\pi n}{N-1})$$

where, $0 \leq n \leq N-1$, N is the window length. Plotting the previous equation yields the following plot:



Hamming Window

There are several reasons why we need to apply a window function to the frames, notably to counteract the assumption made by the FFT that the data is infinite and to reduce spectral leakage.

```
frames *= numpy.hamming(frame_length)
# frames *= 0.54 - 0.46 * numpy.cos((2 * numpy.pi * n) / (frame_length
```

Fourier-Transform and Power Spectrum

We can now do an N-point FFT on each frame to calculate the frequency spectrum, which is also called Short-Time Fourier-Transform (STFT), where N is typically 256 or 512, `NFFT = 512`; and then compute the power spectrum (periodogram) using the following equation:

$$P = \frac{|FFT(x_i)|^2}{N}$$

where, x_i is the i^{th} frame of signal x. This could be implemented with the following lines:

```
mag_frames = numpy.absolute(numpy.fft.rfft(frames, NFFT))  # Magnitude
pow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2))  # Power Spectrum
```

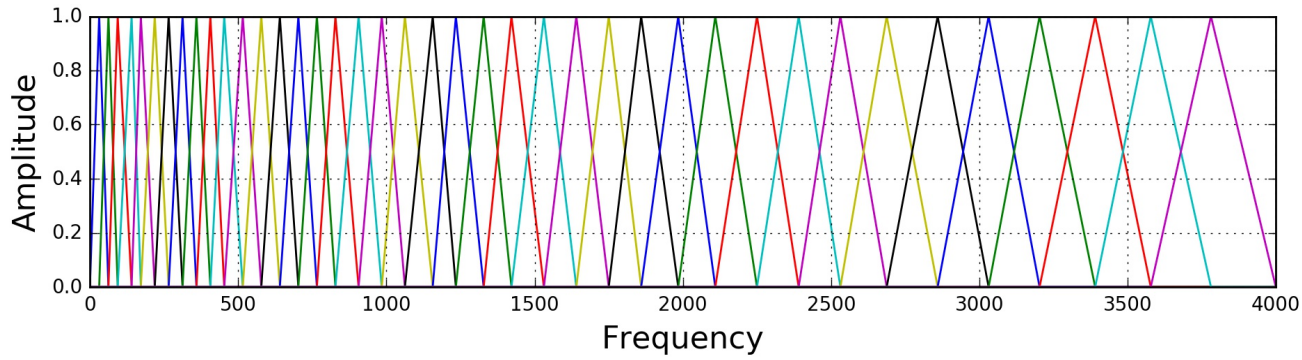
Filter Banks

The final step to computing filter banks is applying triangular filters, typically 40 filters, `nfilt = 40` on a Mel-scale to the power spectrum to extract frequency bands. The Mel-scale aims to mimic the non-linear human ear perception of sound, by being more discriminative at lower frequencies and less discriminative at higher frequencies. We can convert between Hertz (f) and Mel (m) using the following equations:

$$m = 2595 \log_{10}(1 + \frac{f}{700})$$

$$f = 700(10^{m/2595} - 1)$$

Each filter in the filter bank is triangular having a response of 1 at the center frequency and decrease linearly towards 0 till it reaches the center frequencies of the two adjacent filters where the response is 0, as shown in this figure:



Filter bank on a Mel-Scale

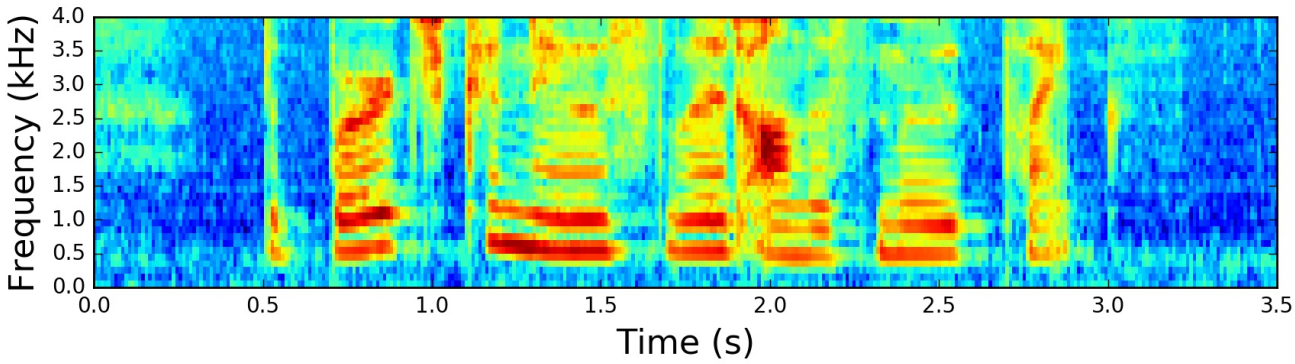
This can be modeled by the following equation (taken from here):

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

```
low_freq_mel = 0
high_freq_mel = (2595 * numpy.log10(1 + (sample_rate / 2) / 700))  # Co
```

```
mel_points = numpy.linspace(low_freq_mel, high_freq_mel, nfilt + 2)  # 
hz_points = (700 * (10**(mel_points / 2595) - 1))  # Convert Mel to Hz
bin = numpy.floor((NFFT + 1) * hz_points / sample_rate)

fbank = numpy.zeros((nfilt, int(numpy.floor(NFFT / 2 + 1))))
for m in range(1, nfilt + 1):
    f_m_minus = int(bin[m - 1])   # left
    f_m = int(bin[m])             # center
    f_m_plus = int(bin[m + 1])    # right

    for k in range(f_m_minus, f_m):
        fbank[m - 1, k] = (k - bin[m - 1]) / (bin[m] - bin[m - 1])
    for k in range(f_m, f_m_plus):
        fbank[m - 1, k] = (bin[m + 1] - k) / (bin[m + 1] - bin[m])
filter_banks = numpy.dot(pow_frames, fbank.T)
filter_banks = numpy.where(filter_banks == 0, numpy.finfo(float).eps, f
filter_banks = 20 * numpy.log10(filter_banks)  # dB
```

After applying the filter bank to the power spectrum (periodogram) of the signal, we obtain the following spectrogram:



Spectrogram of the Signal

If the Mel-scaled filter banks were the desired features then we can skip to mean normalization.

Mel-frequency Cepstral Coefficients (MFCCs)

It turns out that filter bank coefficients computed in the previous step are highly correlated, which could be problematic in some machine learning algorithms. Therefore, we can apply Discrete Cosine Transform (DCT) to decorrelate the filter bank coefficients and yield a compressed representation of the filter banks. Typically, for Automatic Speech Recognition (ASR), the resulting cepstral coefficients 2-13 are retained and the rest are discarded; `num_ceps = 12`. The reasons for discarding the other coefficients is that they represent

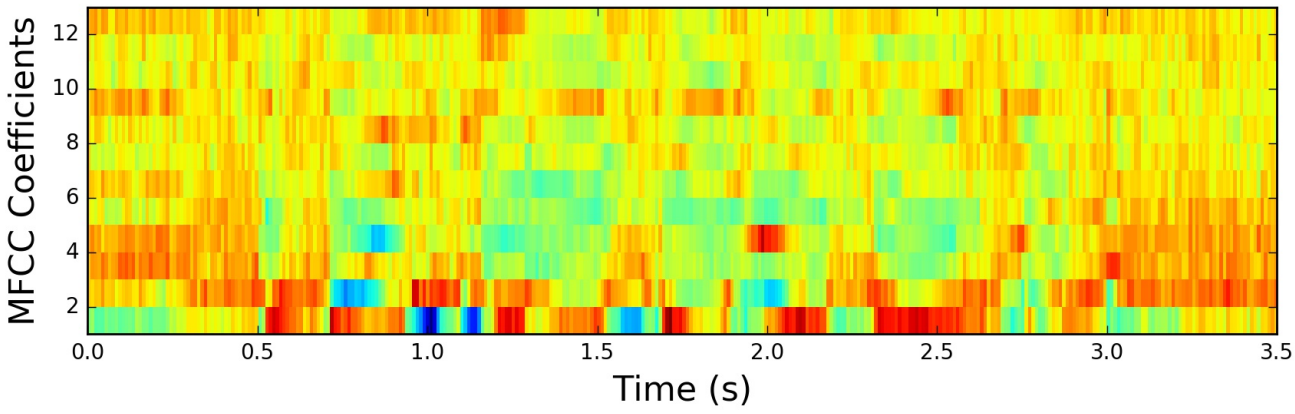
fast changes in the filter bank coefficients and these fine details don't contribute to Automatic Speech Recognition (ASR).

```
mfcc = dct(filter_banks, type=2, axis=1, norm='ortho')[:, 1 : (num_ceps
```

One may apply sinusoidal liftering[1] to the MFCCs to de-emphasize higher MFCCs which has been claimed to improve speech recognition in noisy signals.

```
(nframes, ncoeff) = mfcc.shape
n = numpy.arange(ncoeff)
lift = 1 + (cep_lifter / 2) * numpy.sin(numpy.pi * n / cep_lifter)
mfcc *= lift  #*
```

The resulting MFCCs:



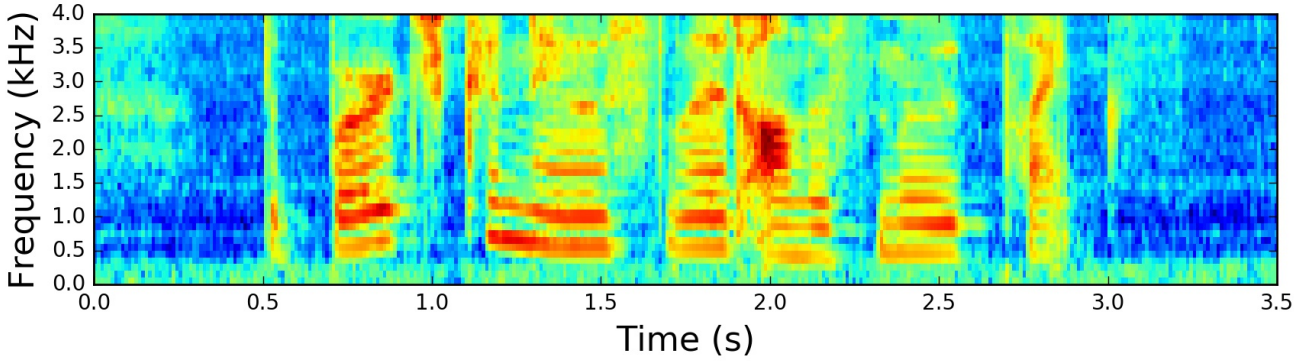
MFCCs

Mean Normalization

As previously mentioned, to balance the spectrum and improve the Signal-to-Noise (SNR), we can simply subtract the mean of each coefficient from all frames.

```
filter_banks -= (numpy.mean(filter_banks, axis=0) + 1e-8)
```

The mean-normalized filter banks:

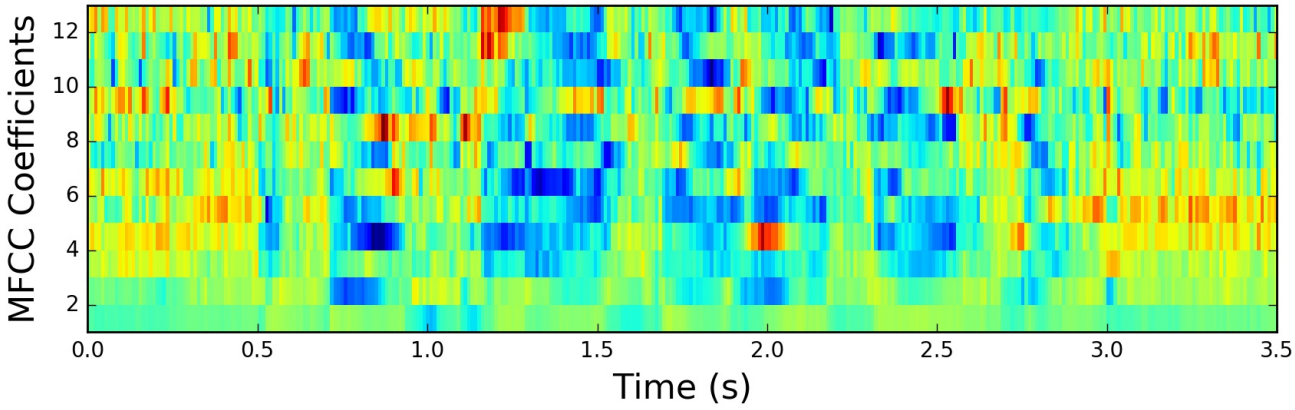


Normalized Filter Banks

and similarly for MFCCs:

```
mfcc -= (numpy.mean(mfcc, axis=0) + 1e-8)
```

The mean-normalized MFCCs:



Normalized MFCCs

Filter Banks vs MFCCs

To this point, the steps to compute filter banks and MFCCs were discussed in terms of their motivations and implementations. It is interesting to note that all steps needed to compute filter banks were motivated by the nature of the speech signal and the human perception of such signals. On the contrary, the extra steps needed to compute MFCCs were motivated by the limitation of some machine learning algorithms. The Discrete Cosine Transform (DCT) was needed to decorrelate filter bank coefficients, a process also referred to as whitening. In particular, MFCCs were very popular when Gaussian Mixture Models - Hidden Markov Models (GMMs-HMMs) were very popular and together, MFCCs and GMMs-HMMs co-evolved to be the standard way of doing Automatic Speech Recognition (ASR)[2]. With the advent of Deep Learning in speech systems, one might question if MFCCs are still the right choice given that deep neural networks are less susceptible to highly correlated input and

therefore the Discrete Cosine Transform (DCT) is no longer a necessary step. It is beneficial to note that Discrete Cosine Transform (DCT) is a linear transformation, and therefore undesirable as it discards some information in speech signals which are highly non-linear.

It is sensible to question if the Fourier Transform is a necessary operation. Given that the Fourier Transform itself is also a linear operation, it might be beneficial to ignore it and attempt to learn directly from the signal in the time domain. Indeed, some recent work has already attempted this and positive results were reported. However, the Fourier transform operation is a difficult operation to learn and may arguably increase the amount of data and model complexity needed to achieve the same performance. Moreover, in doing Short-Time Fourier Transform (STFT), we've assumed the signal to be stationary within this short time and therefore the linearity of the Fourier transform would not pose a critical problem.

Conclusion

In this post, we've explored the procedure to compute Mel-scaled filter banks and Mel-Frequency Cepstrum Coefficients (MFCCs). The motivations and implementation of each step in the procedure were discussed. We've also argued the reasons behind the increasing popularity of filter banks compared to MFCCs.

tl;dr: Use Mel-scaled filter banks if the machine learning algorithm is not susceptible to highly correlated input. Use MFCCs if the machine learning algorithm is susceptible to correlated input.

---

1. Liftering is filtering in the cepstral domain. Note the abuse of notation in *spec*tral and *ceps*tral with *fil*tering and *lif*tering respectively. ↩
2. An excellent discussion on this topic is in this thesis. ↩

hamming windows instead of others? I mean, there is a big reason?. Thanks in advance.

3 ▲ ▼ • Reply • Share ›

Tomas Garcia ➚ Andrea Argudo • a year ago • edited

hamming window has been historically the standard for speech processing. It provides a good balance between frequency resolution (i.e. the separability of closely grouped peaks in the frequency domain) and dynamic range (i.e. the size of the main lobe compared to side lobes) that works well for speech processing applications. square-root hann is also common for applications where the signal is being reconstructed as it is more forgiving when applying aggressive frequency domain processing.

▲ ▼ • Reply • Share ›

Haytham Fayek Mod ➚ Andrea Argudo • a year ago

Hi Andrea, Thanks! I don't have a solid answer to this question. I don't think there is a big reason to use the hamming window particularly, but for example, as opposed to the Hann window, the Hamming window's settings cancel the largest side lobe which results in lower error. Whether this has an effect on the end result in unclear to me.

▲ ▼ • Reply • Share ›

Asheesh Sharma • 9 months ago

Hi, Haytham,

Thank you for sharing your knowledge.
Given that my interest in ASR is relatively new, MFCCs had me confused, which often raised the doubts you mentioned in the Filter Banks Vs MFCCs discussion.

As you have mentioned, Fast Fourier transforms are linear operations. Other than that, one should also consider the obvious computational overhead it brings in with itself (for a serious application).

Unfortunately, I couldn't find any research papers which show the positive results of "directly learning from the input signal". I might also have misunderstood what you meant there (i.e signal in time -> MFCCs). Can you point me in the right direction?

Currently, I am using filter-banks as a result of the dot product b/w the signal frames and filter banks directly, followed by DCT. I have yet to compare the results, but do you think that the approach is flawed?

with regards,
Asheesh

2 ▲ ▼ • Reply • Share ›

pvardanis • a month ago

Hi, Haytham Fayer, that was a great read indeed! I have a few questions though:

Question 1: Do you think calculating other features like spectral centroid, spectral spread etc. (that are normally calculated on the pure spectrogram) on the mel spectrogram would be better? Could this provide better results for classification?

Question 2: Also, by reading librosa.stft examples they compute the power spectrum by just multiplying to the power of 2, and you additionally divide by number of fft points (which is more correct in my opinion). Does it really make a difference?

Question 3: Why calculating the mel spectrogram directly from the spectrogram is just an approximation of the mel spectrogram and not exactly the same (when you calculate it directly from audio)?

Question 4: Why we don't care for the mfcc 1?

Reply • Share ›

PMacedoFilho Filho • 2 months ago

Many thanks Haytham. Excellent explanation on a theme not so easy. I made a comparison with the results with the results obtained using the python_speech_features module and the MFCC coefficients are very different. You can tell the reason. thank you !!!

Reply • Share ›

Rob Webster • 3 months ago

Hi Haytham Fayek, fantastic article. This helped me out so much with my dissertation, I was wondering if it would be okay to have you in my "acknowledgements" section?

Reply • Share ›

Haytham Fayek Mod → Rob Webster • 3 months ago

Of course, Thank you! I am glad it helped.

Reply • Share ›

Robin Algayres • 6 months ago

Hi Fayek, great article !
I don't understand why DCT would discard information. DCT is invertible so it should not lose anything from the signal by applying it to the melfilters. Same question for the FFT. Thank you!

Reply • Share ›

Haytham Fayek Mod → Robin Algayres • 3 months ago

Thanks, Robin! In the FFT, we discard the phase information, while in the DCT, we only keep the first n coefficients and discard the remaining ones; so in both cases, we're throwing away information.

Reply • Share ›

Julio • 6 months ago

how do you plot mfcc and pow_frame?

Reply • Share ›

Ego ren → Julio • 6 months ago

Hello, I use these following code to get plot pic.

```
plt.subplot(312)
filter_banks -= (numpy.mean(filter_banks,axis=0) + 1e-8)
plt.imshow(filter_banks.T, cmap=plt.cm.jet, aspect='auto')
plt.xticks(np.arange(0, (filter banks.T).shape[1],
```

```
int((filter_banks.T).shape[1] / 4)),
['0s', '0.5s', '1s', '1.5s','2.5s','3s','3.5'])
ax = plt.gca()
ax.invert_yaxis()
plt.title('the spectrum image')
```

2 • Reply • Share ›

Vaibhav Gupta • 7 months ago

can someone help me with plotted spectogram

• Reply • Share ›

Alex Beloi • 8 months ago

Nice write-up!

I found a small typo: "Fourier transform across the entire signal in that we would loose the frequency contours", I believe you mean "lose" instead of "loose".

• Reply • Share ›

Haytham Fayek Mod → Alex Beloi • 3 months ago

Thanks Alex! Fixed.

• Reply • Share ›

Amogh Hassija • 8 months ago

Hi, great article. Could you explain liftering a bit more please, and what should be the value of 'cep_lifter'?

• Reply • Share ›

Shreyas Gupta • 8 months ago • edited

There is a small mistake in the frames calculation. with np.ceil, we will loose one frame in some cases. This is loss of some signal data. I don't know whether it matters but we can overcome that with We can overcome this with padding the audio signal with zeroes and then calculating number of frames by

int(np.floor((signal_length - frame_length) / frame_step) + 1)

Really useful article though! cheers!

EDIT : I think both approaches are the same :)

• Reply • Share ›

Lucrece Summer → Shreyas Gupta • 5 months ago

I agree with your catch for frame calculation. By "both approaches are the same" do you mean the final result is the same?

• Reply • Share ›

Guessous Mohamed Amine • 10 months ago

Thank you for this excellent article!! Can you please refer us to tutorial on DSP. It's been a while since I've touch into this!!

• Reply • Share ›

ash03yo • a year ago • edited

I think cep_lifter its not defined...
so deafult is 22 right?

Reply • Share ›

Haytham Fayek Mod ash03yo • 3 months ago

Right.

Reply • Share ›

Matthew Canova • a year ago

Hi Haytham Fayek, great article and thank you for sharing! I noticed that when you posted the Mel-scale filter bank modeling equations, you have a small typo in the bottom right it should be k > f(m+1). Thank you again!

Reply • Share ›

Haytham Fayek Mod Matthew Canova • 3 months ago

Nice catch. Thanks, Matthew! Fixed.

Reply • Share ›

Guido Gallopyn • a year ago

Very nice post

but one comment

'The Discrete Cosine Transform (DCT) was needed to decorrelate filter bank coefficients, a process also referred to as whitening"

I think the correct reason for the DCT is the following. The speech signal is a pulse train (glottal signal for voiced speech) or random noise (unvoiced speech), filtered by a time varying filter (vocal tract) that changes with the phones spoken. In the frequency domain that is Glottal * H(phone) and the glottal signal is a frequency comb (pitch frequency and harmonics). In a spectrogram this is visible as the repetitive frequency lines during voiced speech. For recognition purposes only H(phone) is needed and no the glottal signal. The glottal signal is removed by applying a lowpass filter (smoothing) on the frequency response (after application of the Mel filter banks), which is accomplished in the cepstral domain by taking a DCT and by only keeping the lowest cepstral coefficients.
Removing glottal signal from the speech signal removes variability that is mostly redundant for recognition, but not entirely (especially in tonal languages) and neural nets may be able to use it at the expense of more variability.
Decorrelation of features with GMM was typically accomplished with an LDA, or later bottleneck features

Reply • Share ›

richardyo888 • a year ago

i have a question...
how do you applied a Pre-Emphasis filter, if you are not in the frecuency domain?... i mean, i dont get it, you just sampled your audio file, but is in the time domain rigt?
i already do this part on python and yes it works, just i want to know how?

Reply • Share ›

Haytham Fayek Mod → richardyo888 • a year ago

The pre-emphasis filter is just a filter. Filtering can be done in either the time or frequency domains.

Reply • Share ›

yash gaurkar • a year ago

Thank you very much for this article. This method is really helpful for feature extraction.

Reply • Share ›

João Miguel Silva • a year ago

Hi Haytham Fayek

indices = numpy.tile(numpy.arange(0, frame_length), (num_frames, 1)) + numpy.tile(numpy.arange(0, num_frames * frame_step, frame_step), (frame_length, 1)).T

Why do you use .T at the end of this line of code?

Reply • Share ›

Haytham Fayek Mod → João Miguel Silva • a year ago

.T transposes the output of the tile function to fit the desired shape.

1 Reply • Share ›

Siba Zazo • a year ago

hi ,very useful article
i was wondering if you have any information about MFSC
in particular about applying mel filterbank to the magnitude spectrum if we don't want to truncate the size of the magnitude spectrum to be equal to the half of FFT_size+1
i have mel-filterbank matrix with shape=(40,257)
and magnitude-spectrum with shape=(400,584)

Reply • Share ›

M Zohaib • a year ago

Hello, Haytham Fayek, What information we are getting from spectrogram, fliter bank and cepstrum plot for MFCCs, how we visualize the graphs you have obtained above, i am working in this area, pls guide me.

Reply • Share ›

Pravar Kapoor • a year ago

Hey, the article is very helpful. I have a question, the output of MFCC produces hundreds of feature vectors spanning across 12 columns, depicting that we had 12 coefficients ? Can you please explain why are we getting hundreds of tuples/rows? Should'nt we get only one row?

Reply • Share ›

Leonardo • a year ago

Great Article!!
I still have a question, why are you using :
signal length - frame length)) / frame step

to get the number of frames instead of just

signal_length/frame_length???

Reply • Share ›

yash gaurkar ➚ Leonardo • a year ago

for ex:
if i want to calculate number of frames for a signal with (signal length = 100, frame length = 50, frame step = 25):
Since this is a small example we can look at the range in which different frame lies i.e., the frames will be:
frame 1 : from 0 to 49 (indexing from 0)
frame 2 : from 25 to 74
frame 3: from 50 to 99
so we will have 3 frames
so the formula will be [((signal length - frame length)/frame step)) + 1] = [((100-50)/25) + 1] = 3
but the formula which you are suggesting there you are not taking in account the first frame. which is from 0:50.

Reply • Share ›

Haytham Fayek Mod ➚ Leonardo • a year ago

To account for the overlap between frames.

Reply • Share ›

Sam Lin • a year ago

Awesome article!!
I am fresh to speech and it helps me a lot.
I have some questions,
Is filter bank also called "mel-spectrogram" in some paper?
also why does mel-spectrogram often applied logarithm afterwards for analyzing ASR, for example the process applied in this paper https://arxiv.org/abs/1701.....

Thank you so much!!!

Reply • Share ›

Ravi Teja • 2 years ago • edited

Hi Haytham Fayek, great article. I learnt a lot from this. Can you explain why a linear transformation like fft/dct discards some information in speech ? , fft/dct can recover the signal completely without any loss in information when inverse functions are applied right ?

Reply • Share ›

Haytham Fayek Mod ➚ Ravi Teja • a year ago

Thanks! Note that we only used a number of coefficients from the DCT operation and so we can't recover the signal exactly from those coefficients.

Reply • Share ›

chandra sutrisno • 2 years ago

Hi,

Thank you for this useful tutorial. Currently, I am trying to build voice/speaker recognition so this is really helpful.

I use your method to extract the features from my data set. I have 10 sample voice and each sample yields different shape of array? Is this real? How come this can be happens? And then how to feed it into machine learning model since it a 2D array? Should I use any NN model?

Please advise

• Reply • Share ›

Haytham Fayek Mod → chandra sutrisno • a year ago

Yes, since your voice samples are of different lengths, the number of features would vary according to the lengths. You'll want to divide those into equal number of frames and then align the transcription to the frames.

• Reply • Share ›

שחף נחמיאס → chandra sutrisno • 2 years ago

Would like to hear the answer to that questions as well.
Great tutorial!

• Reply • Share ›

CHIA-CHEN Wei • 2 years ago

Dear Haytham Fayek
How to implement ISTFT from the result "numpy.fft.rfft(frames, NFFT)" to get the original signal "emphasized_signal"?
Thank you.

• Reply • Share ›

Haytham Fayek Mod → CHIA-CHEN Wei • 2 years ago • edited

Check this out: https://docs.scipy.org/doc/....

• Reply • Share ›

CHIA-CHEN Wei → Haytham Fayek • 2 years ago

Thank you very much

• Reply • Share ›

Saul Rodriguez • 2 years ago

Hi Haytham, thanks so much for your post. I am having some issues replicating your plots. Do you have the complete python code that you used for this post? Thank you.

• Reply • Share ›

Haytham Fayek Mod → Saul Rodriguez • 2 years ago

I am glad you found it useful. The complete code is just a concatenation of the above.

1 • Reply • Share ›

dev g • 2 years ago

do you have any idea, how I can convert this code into MATLAB code??

• Reply • Share ›

Haytham Fayek Mod dev g • 2 years ago

It's pretty straightforward but careful MATLAB's 1-based indexing.

• Reply • Share ›

Chen Tiancai • 2 years ago

how to convert the mfcc to spectrum envelop? Is there any python-based tool to do this? thank you !

• Reply • Share ›