UNIVERSITÄT BASEL

Computer Science / 15731-01 / 2020

Multimedia Retrieval

Chapter 5: Audio Retrieval

Dr. Roger Weber, roger.weber@ubs.com

5.1 Introduction

- There are two definitions for sound: the first one is based on physics and describes vibrations that propagate in the form of audible pressure waves through a medium (gas, liquid, solid). The second is based on the perception through the hearing mechanism, that is, as a sensation.
- **Physics of Sound**: soundwaves are generated by a source, for instance vibrations of a speaker, and traverse a media as wave with a specific wavelength λ (or frequency f), pressure p (amplitude or intensity, measured in decibel), speed v, and direction $\vec{x}$. Note that sounds only travel if a medium exist but not in vacuum. The particles of the medium locally vibrate but do not travel with the wave.
  - The human ear perceives frequencies between 20Hz and 20kHz, corresponding to sound waves of length 17m and 17mm in air at standard conditions, respectively. The relationship between wavelength and frequency is given by the speed of the wave: $\lambda \cdot f = v$.
  - The speed of sound waves depend on the medium: in air under standard conditions, sound travels with $v = 331 + 0.6 \cdot T$ m/s with T the temperature in Celsius. In water, sound travels much faster at speeds of about $v = 1482$ m/s. In solids, speeds are even higher ranging from $v = 4000$ m/s in wood up to $v = 12{,}000$ m/s in diamonds.
  - Sound travels in concentric waves that can be reflected, refracted (when passing from one medium to another), and attenuated (gradual loss of intensity as the wave travels). With the physic properties, it is possible to locate the source of the sound (or most recent reflection point).
  - Sound pressure is the difference between the local pressure in the medium and the pressure of the wave. It is often expressed as decibel: $L_p = 20 \cdot \log_{10}(p/p_{ref})$ with p the sound pressure and p_{ref} the reference pressure (20 μPa in air). The factor 20 (and not 10) is because we compare squares of pressures; with the logarithm, this adds and extra factor of 2. The logarithmic scale is necessary due to the wide dynamic range of perception. 0 dB is the auditory threshold and sounds above 120 dB may cause permanent hearing loss.

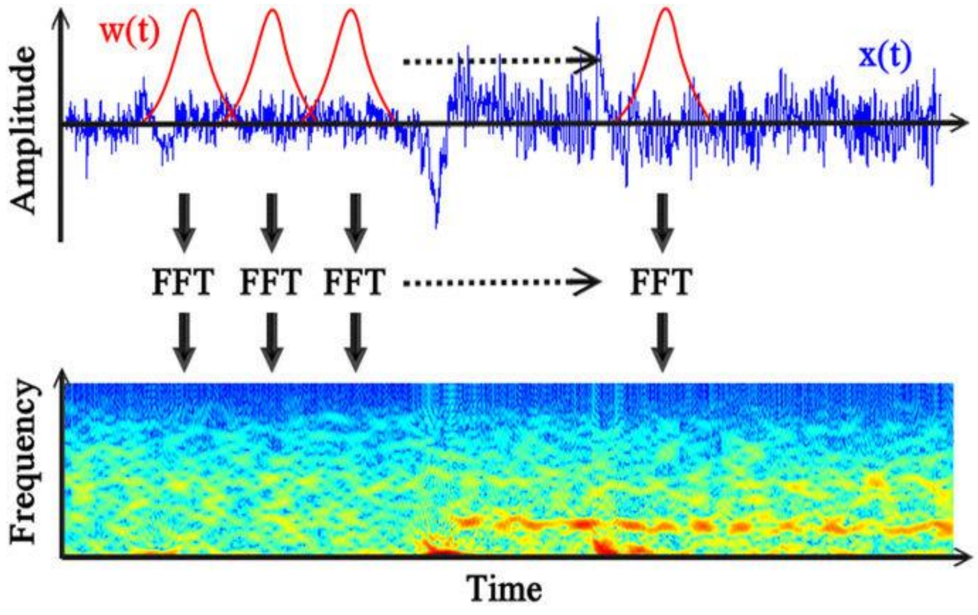
- **Perception of Sound**: historically, the term sound referred exclusively to the auditory perception (“that which is heard”). Nowadays, the term is used both for the physical effect as well as the sensation of that effect. The perception is bound to a range of frequencies. The human ear can perceive frequencies between 20Hz to 20kHz. A cat perceives frequencies between 500Hz and 79kHz. The higher range is useful to detect high frequency mice communication (at 40kHz). Bats have a range from 1kHz up to 200kHz and use the ultrasonic sounds for echolocation of prey. The elements of sound perception are:
  - **Pitch:** is the perceived (primary) frequency of sound. It is a perceptual property that allows us to judge music as “higher” or “lower”. Pitch requires a sufficiently stable and clear frequency to distinguish it from noise. It is closely related to frequency but not identical.
  - **Duration:** is the perceived time window of a sound, from the moment it is first noticed until it diminished. This is related to the physical duration of the wave signal, but compensates breaks of the signal. For instance, a broken radio signal can still be perceived as a continuous message.
  - **Loudness:** is the perceived level (“loud”, “soft”) of a signal. The auditory system stimulates over short time periods (~200ms): a very short sound is thus perceived softer than a longer sound with the same physical properties. Loudness perception varies with the mix of frequencies.
  - **Timbre:** is the perceived spectrum of frequencies over time. Sound sources (like guitar, rock falling, wind) have very characteristic timbres that are useful to distinguish them from each other. Timbre is a characteristic description of how sound changes over time (like a fingerprint).
  - **Sonic Texture:** describes the interaction of different sound sources like in an orchestra or when sitting in a train. The texture of a quiet market place is very distinct from the one of busy party.
  - **Spatial Location:** denotes the cognitive placement of the sound in the environment (not necessarily the true source) including the direction and distance. The combination of spatial location and timbre enables the focused attention to a single source (e.g., partner at a party).

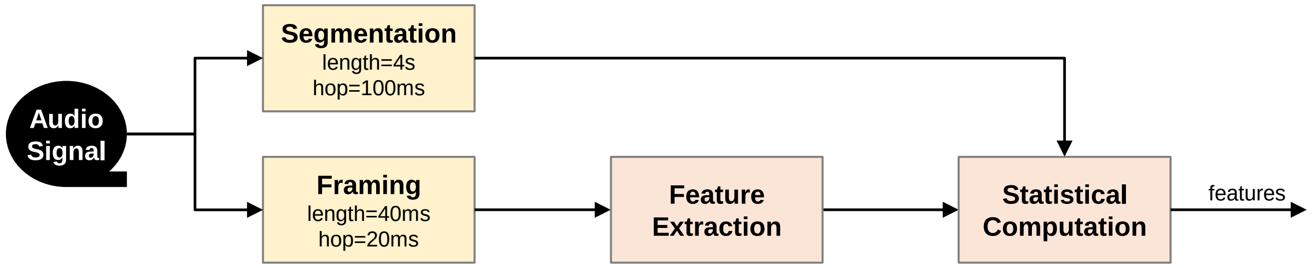
- Audio signals are expressed as an amplitude signal over time. To capture the continuous signal and create a discrete digital representation, the signal is sampled with a fixed frequency f_s. The Nyquist–Shannon sampling theorem states that the sampling rate limits the highest frequency f_{max} that can be resolved to half of the sampling rate ($f_{max} = f_s/2$). As the human perception ranges between 20Hz and 20kHz, sampling rates of CDs was defined at 44.1kHz and the one for DVD at 48kHz.
- To model human perception, it is necessary to transform the raw amplitude signal into a frequency space. Unlike with images, we cannot apply a Fourier transformation across the entire signal as this would average frequencies across the entire time scale and does not allow for an analysis of frequency changes over time. Instead, the Short-Term Fourier Transform (STFT) applies a window function and computes a local Fourier transformation around a time point and a given window size. In the discrete form the STFT of the raw amplitude signal $x(t)$ is given as:

$$X(t,\omega) = \sum_{n=-\infty}^{\infty} x(n) \cdot w(n-t) \cdot e^{-i\omega n}$$

With a window size of N samples, the discrete frequency ω ranges between 0 and $f_{max} = f_s/2$ at steps of f_s/N Hz. The absolute values of the complex value $X(t,\omega)$ denote the magnitude of the frequency ω at time point t

  – The picture on the right depicts the STFT with the red windowing function $w(t)$ as it is applied over time. The spectrogram is then the squared magnitudes $|X(t,\omega)|^2$ over time. One can use different windowing functions.

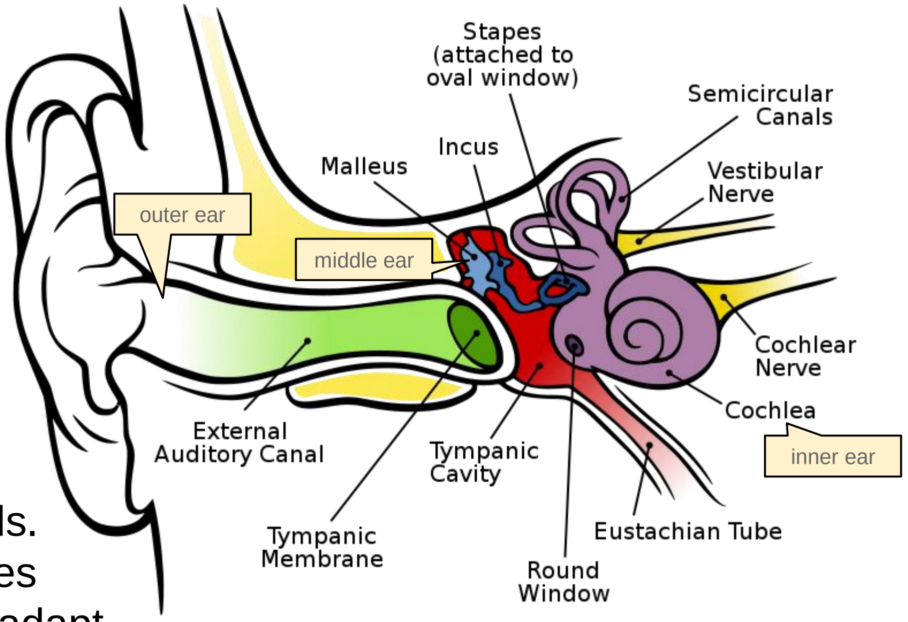




- Feature design requires further segmentation of the signal to capture statistics for changes over small time chunks (compare with timbre). In the picture above, the audios signal is first split into frames on which also the STFT is applied. The frames overlap with each other to avoid boundary effects. For each frame, we obtain a single feature vector. The second split of the audio signal creates overlapping segments encompassing several subsequent frames. The segment features are then a statistical summary over the features of its frames. The segments are the smallest unit for retrieval, and a single audio file is described by hundreds or thousands of segments.
  - Frame size: let the sampling frequency be $f_s = 48$ kHz. With a frame size of 40ms, the number of samples is $N = 1920$. Hence, the frequency resolution of STFT is $\frac{f_s}{N} = 20.83$ Hz. This is hardly sufficient to distinguish two musical pitches at the middle octave, but not for the first and second octave (each octave doubles the frequency). To improve frequency resolution, we could increase the window size (reducing sampling rate would result to audible artefacts). But then, we loose precision along the time axis as a broader range blurs the spectrum. In short, STFT requires us to compromise either on frequency resolution or time resolution. Alternative approaches with wavelets have solved this issue and provide both good time and frequency resolution.
  - Segment size: depends on the task at hand. For timbre detection (guitar, rock falling, wind) a shorter segment can be used. For spoken text, alternative segmentation approaches can be used. The 4s in the picture is a good starting point for generic audio analysis.

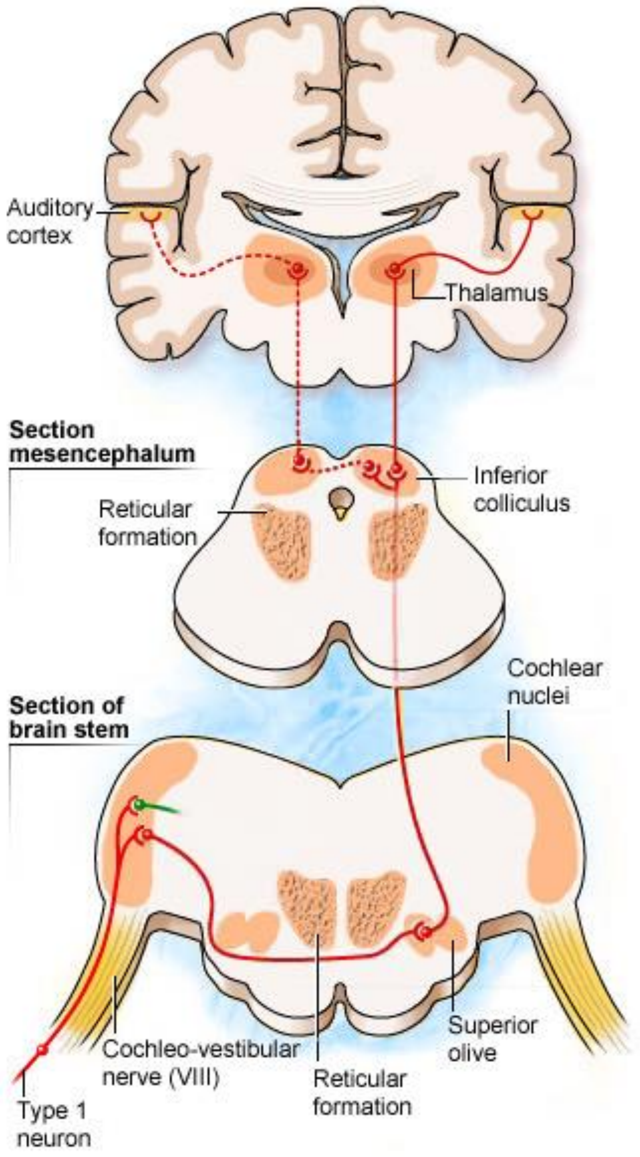
5.2 Auditory Perception and Processing

- The ear translates incoming pressure changes into electro-chemical impulses
  - The **outer ear** is the visible part of the organ. Sound waves are reflected and attenuated, and additional information is gained to help the brain identify the spatial location. The sound waves enter the auditory canal which amplifies sounds around 3kHz up to 100 times. This is an important range for voice recognition (e.g., to distinguish ‘s’ from ‘f’). Sound travels through the ear canal and hits the eardrum (tympanic membrane).
  - Waves from the eardrum travel through the **middle ear** (also filled with air) and a series of very small bones: hammer (malleus), anvil (incus), and stirrup (stapes). These bones act as a lever and amplify the signal at the oval window (vestibular window). Amplification is necessary as the cochlea is filled with liquid. A reflex in the middle ear prevents damage from very loud sounds.
  - The **inner ear** consists of the cochlea and the vestibular system. The latter is responsible for balance and motion detection and works similar to the cochlea. Along the cochlea runs the organ of Corti (spiral corti) with the hair cells. The outer hair cells amplify the signal and improve frequency selectivity. The inner hair cells are mechanical gates that close very rapidly under pressure (gate open means “no sound”). The base of the cochlea (closest to middle ear) captures high frequency sounds while the top captures low frequency sounds. The non-linear amplification of quiet sounds enlarges the range of sound detection. Chemical processes adapt to a constant signal focusing attention to changes.



Chittka L, Brockmann - Perception Space—The Final Frontier, A PLoS Biology Vol. 3, No. 4, e137 doi:10.1371/journal.pbio.0030137

- The electro-chemical impulses created by the inner hair cells releases neurotransmitters at the base of the cell that are captured by nerve fibers. There are 30’000 auditory fibers in each of the two cochlear nerves. Each fiber represents a particular frequency at a particular loudness level. Similarly, the vestibular nerve transmits balance and motion information. There are two pathways to the brain: the primary auditory pathway (discussed below) and the reticular pathway. The latter combines all sensory information in the brain to decide which sensory event requires highest priority by the brain. The primary path is as follows:
  - The **cochlear nuclear complex** is the first “processing unit” decoding frequency, intensity, and duration.
  - The **superior colliculus** (mesencephalum) infers spectral cues from frequency bands for sound location.
  - The **medical geniculate body** (thalamus) integrates auditory data to prepare for a motor response (e.g., vocal response)
  - Finally, the **auditory cortex** performs the basic and higher functions of hearing. Neurons are organized along frequencies. Frequency maps help to identify the source of the sound (e.g., wind). Further, it performs sound links to eliminate distortions due to reflection of waves. The auditory complex is essential to process temporal sequences of sound which are elementary for speech recognition and temporally complex sounds such as music.



5.2.1 Generic Acoustical Features

- The first set of features describe audio files from an acoustical perspective along the domains
  - Time Domain – considering the raw signal in the time space (amplitude signal)
  - Frequency Domain – transforming raw signal with STFT and analyzing frequencies and their energies at the given time point (see window technique)
  - Perceptual Domain – modelling the perceptual interpretation of the human ear
- **Feature in the Time domain (frame)**: we consider the amplitude signal in the time domain using a single frame F_i (see segmentation). For instance, with $f_s = 48$ kHz and a frame size of 40ms, the number of samples is $N = 1920$, and the hop distance between subsequent frame is 20ms.
  - **Short-Time Energy (STE)**: measures the raw energy as a sum of squares, normalized by the frame length. With audio signals, power is usually measured as decibel (which is one-tenth of a bel, a unit introduced by the first telephony system). An increase of 10 dB denotes a power change of a factor of 10. The metric is logarithmic: $L_P = 10\log_{10}(P/P_0)$. With that, STE for an amplitude signal $x(t)$ within a frame F_i (hence: $1 \le t \le N$) is defined as:

$$E_{STE}(i) = 10\log_{10}\left(\frac{1}{N}\sum_{t=1}^{N} x(t)^2\right)$$

  - **Zero-Crossing Rate (ZCR)**: counts, how often the sign of the amplitude signal over the duration of the frame F_i (e.g., from positive to negative values) changes:

$$ZCR(i) = \frac{1}{2N}\sum_{t=2}^{N}|sgn(x(t)) - sgn(x(t-1))|$$

- **Entropy of Energy (EoE)**: measures abrupt changes in the energy of the audio signal within a frame F_i. To this end, the frame is divided into L sub-frames of equal length spanning the entire frame. For each sub-frame S_l, the energy is measured and normalized by the total energy of the frame to obtain a sequence of “probabilities” that sum up to 1. The entropy of these “probabilities” is the Entropy of Energy. Choose L and N_{sub} such that $N = L \cdot N_{sub}$:

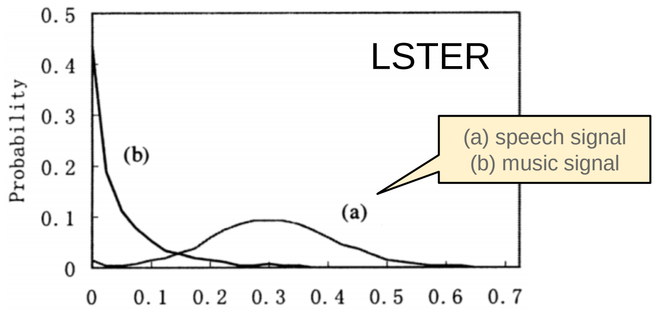
$$H_{EoE}(i) = -\sum_{l=1}^{L} e(i,l) \cdot \log_2 e(i,l) \qquad e(i,l) = \frac{\sum_{t=l\cdot N_{sub}}^{(l+1)\cdot N_{sub}-1} x(t)^2}{\sum_{t=1}^{N} x(t)^2}$$

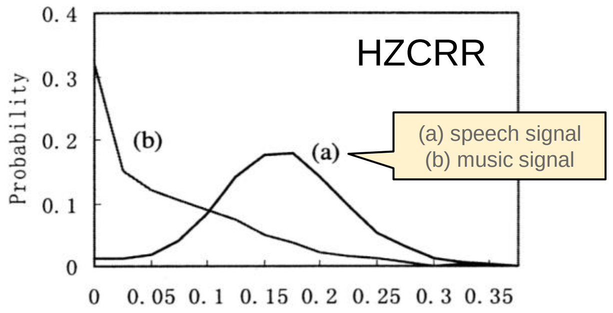
- **Feature in the Time domain (segment):** The following features summarize statistics across a segment S_j with M frames. Consider, for instance, a segment length of 4s, a frame size of 40ms and a frame hop distance of 20ms, then the number of frames is $M = 199$ (or $M = 200$ depending on how to treat the last frame that partially is in the segment and partially outside the segment).
  - **Low Short-Time Energy Ratio (LSTER)**: denotes the percentage of frames in the segment whose STE is below a third of the average STE across the segment S_j. Speech signals have a higher variation due to pauses between syllables.

$$r_{LSTER}(j) = \frac{1}{M}\sum_{i=1}^{M} \begin{cases} 1 & E_{STE}(i) < \frac{\mu_{STE}(j)}{3} \\ 0 & \text{otherwise} \end{cases} \qquad \mu_{STE}(j) = \frac{1}{M}\sum_{i=1}^{M} E_{STE}(i)$$

- **High Zero-Crossing Rate Ratio (HZCRR)**: speech signals have much more zero-crossings than a typical music signal, and the variations is much higher (due to breaks between syllables). The HZCRR over a segment S_j is defined as:

$$r_{HZCRR}(j) = \frac{1}{M}\sum_{i=1}^{M}\begin{cases}1 & ZCR(i) \geq \frac{3\mu_{ZCR}(j)}{2} \\ 0 & \text{otherwise}\end{cases} \qquad \mu_{ZCR}(j) = \frac{1}{M}\sum_{i=1}^{M} ZCR(i)$$





Lu, Zang, Content Analysis for Audio Classification Segmentation, 2002

- **Moments over STE and ZCR:** compute moments over STE and ZCR values across the segment S_j. This describes the distribution of values within the segment. The following formulas describe STE moments; ZCR moments are obtained similarly. Note that these are biased versions of the moments (which are close to unbiased moments if $M > 100$):

$$\mu_{STE}(j) = \frac{1}{M}\sum_{i=1}^{M}|E_{STE}(i)| \qquad v_{STE}(j) = \frac{1}{M}\sum_{i=1}^{M}(|E_{STE}(i)| - \mu_{STE}(j))^2$$

$$s_{STE}(j) = \frac{1}{M}\sum_{i=1}^{M}\left(\frac{|E_{STE}(i)| - \mu_{STE}(j)}{\sqrt{v_{STE}(j)}}\right)^3 \qquad k_{STE}(j) = \frac{1}{M}\sum_{i=1}^{M}\left(\frac{|E_{STE}(i)| - \mu_{STE}(j)}{\sqrt{v_{STE}(j)}}\right)^4$$

- **Histograms**: partition the value space of a feature and compute how often values fall into a partition across the frames of segment S_j. The normalized numbers yield a histogram over the feature values. This method is seldom used as it produces to large features than moments.

- **Feature in the Frequency Domain (frame)**: we consider the Fourier transformed signal in the frequency domain using a single frame F_i (see segmentation). For instance, with $f_s = 48$ kHz and a frame size of 40ms, the number of samples is $N = 1920$, and the hop distance between subsequent frame is 20ms. The Fourier transformed values $X(i,\omega)$ denotes the frequency spectrum of frame F_i with $0 \leq \omega \leq f_s/2$ and with steps $\Delta\omega = f_s/N = 25$ Hz. Also note that in the discrete notation of the Fourier transformed, i.e., $X(i,k)$ with $0 \leq k < N/2$, only the first half of the values are needed as the second half is symmetrical (as we had real values only in the time domain). In the following, we often use the discrete form $X(i,k) = X(i,\omega(k))$ with $\omega(k) = k \cdot f_s/N$.
  - **Spectral Centroid (SC)**: denotes the gravity center of the spectrum, i.e., the weighted average frequency in the spectrum of the frame F_i with the magnitude as weights, i.e., the magnitude is the absolute values of the (complex) $X(i,k)$. For convenience, let $K = N/2 - 1$. Hence:

$$SC(i) = \frac{\sum_{k=0}^{K} \omega(k) \cdot |X(i,k)|}{\sum_{k=0}^{K} |X(i,k)|}$$

    The centroid describes the “sharpness” of the signal in the frame. High values correspond to signals skewed at higher frequencies.
  - **Spectral Roll-off** (ω_r): denotes the frequency ω_r such that the sum of magnitudes with frequencies smaller than ω_r is $C = 85\%$ of the total sum of magnitudes. Hence, we look for a value $0 \leq r \leq K$ as follows (other values for $0 \leq C < 1$ are possible)

$$\omega_r = \omega(r) \quad \text{with } r \text{ smallest value that fulfills: } \sum_{k=0}^{r} |X(i,k)| \leq \mathrm{C} \cdot \sum_{k=0}^{K} |X(i,k)|$$

    Related to the spectral centroid, it measures how skewed the spectrum is towards higher frequencies which are dominant in speech.

- **Band-Level Energy (BLE)**: refers to the sum of energy within a specified frequency range. The range is captured through a weighting function $w(k)$ in the Fourier domain with $0 \leq k \leq K$. The feature value is measured in decibel to match hearing perception:

$$BLE(i) = 10 \log_{10}\left(\sum_{k=0}^{K} |X(i,k)|^2 \cdot w(k)\right)$$

- **Spectral Flux (SF)**: describe the squared differences of normalized magnitudes from the previous frame. It provides information of the local spectral rate of change. A high value indicates a sudden change of magnitudes and thus a significant change of perception (only for $i > 1$):

$$SF(i) = \sum_{k=0}^{K}\left(\frac{|X(i,k)|}{\sum_{k=0}^{K}|X(i,k)|} - \frac{|X(i-1,k)|}{\sum_{k=0}^{K}|X(i-1,k)|}\right)^2$$

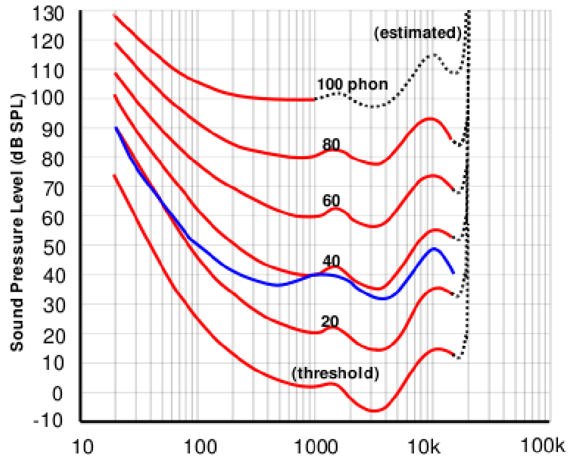
- **Spectral Bandwidth (SB)**: denotes the normalized magnitude weighted deviation from the spectral centroid. It describes the expected distance of frequencies from the spectral centroid:

$$SB(i) = \sqrt{\frac{\sum_{k=0}^{K}|X(i,k)| \cdot \left(\omega(k) - SC(i)\right)^2}{\sum_{k=0}^{K}|X(i,k)|}}$$

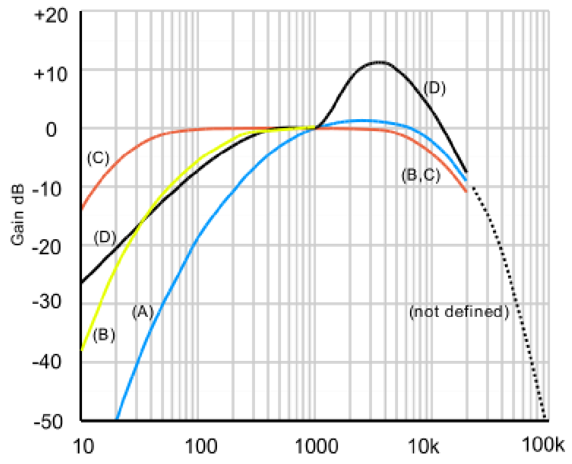
• **Feature in the Frequency Domain (segment)**: to summarize a segment, we can use again moments and histograms over the frame values for the various features above.

- **Feature in the Perceptual Domain (frame)**: the human ear and the interpretation of sound wave differs significantly from the raw physical measures. For instance, loudness is a measure of the energy in the sound wave. The human perception, however, amplifies frequencies differently, especially the ones between 2 and 5 kHz which are important for speech recognition. The following measures take perception into account.
  - **Loudness**: perception of the sound pressure level depends on the frequency as shown on the figure on the upper right side. Each red curve denotes how much energy is required such that an average listener perceives the pure tone as equally loud. As discussed before, the energy drops significantly between 2 and 5 kHz due to amplifications in the ear. To model this perception the international standard IEC 61672:2003 defined different weighting function as shown by the figure on the lower right side. The A-weighting curve is the most frequently used despite that it is only "valid" for low-level sounds. In addition, the human auditory system averages loudness over a 600-1000ms interval. The loudness at the F_i is hence the average over the previous 1000ms of the signal and not just the values in the frame. Let O be the number of frames over the last 1000ms. For instance, with a hop size of 20ms, $O = 50$. Loudness is measure in decibel, again, to match perception of increased loudness:

$$L(i) = \frac{10}{O} \sum_{o=0}^{O-1} \log_{10} \left(\frac{1}{K} \sum_{k=1}^{K} A(k) \cdot |X(i-o,k)|^2 \right)$$



Equal-loudness contours (red) (from ISO 226:2003 revision)
Original ISO standard shown (blue) for 40-phons



A-weighting (blue), B (yellow), C (red), and D-weighting (blk)

– **Mel Frequency Cepstral Coefficients (MFCC)**: represents the spectrum of the power spectrum over Mel frequency bands. The Mel frequency bands approximate the human auditory system. The method works in 4 steps:

1. Fourier Transform: compute the Fourier transform over the frame F_i. Here, we do not use a windowing function as with the STFT. Let N be the number of samples in the frame F_i and f_s be the sampling rate (e.g., $N = 1920, f_s = 48$ kHz)

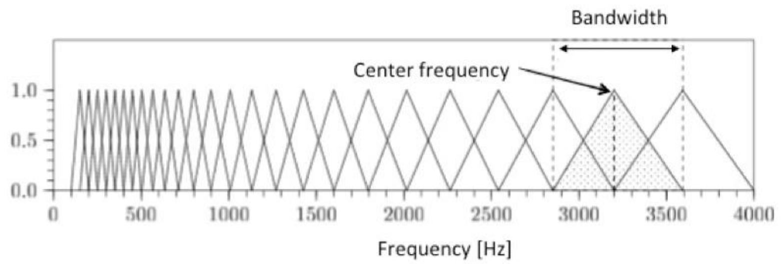
$$X(t,k) = \frac{1}{N}\sum_{j=0}^{N-1} x(j) \cdot e^{-i2\pi\frac{jk}{N}} \qquad \omega(k) = k \cdot \frac{f_s}{N}$$

2. Mel-Frequency Spectrum: the spectrum is computed over Mel frequency bands. Let B be the number of bands, and let f_{lower} and f_{upper} denote the lower and upper range of frequencies. Typically, we have $B = 26$, $f_{lower} = 300$ Hz, and $f_{upper} = 8000$ Hz. First, we create the bands. The conversion from frequencies to mels and vice versa is as follows:

$$freq(m) = 700 \cdot \left(e^{\frac{m}{1125}} - 1\right) \qquad mel(f) = 1125 \cdot \ln\left(1 + \frac{f}{700}\right)$$

The bands are triangle shaped windowing functions in the frequency space. Three frequencies define the start point, the middle point, and the end point. Two bands overlap with each other: the start point of a band is given by the middle point of the previous band. The frequencies are computed in the Mel space to match human perception. Given B bands, we need $B + 2$ frequencies given by ($0 \le b \le B + 1$)

$$f_c(b) = freq\left(mel(f_{lower}) + b \cdot \frac{mel(f_{upper}) - mel(f_{lower})}{B+1}\right)$$



With the frequencies $f_c(b)$, we can define now the windowing function $d(b,k)$ over the Fourier coefficients $X(t,k)$ for a given time point t. The shape has a triangle form:

$$d(b,k) = \begin{cases} 0 & \text{if } \omega(k) < f_c(b-1) \\ \dfrac{\omega(k) - f_c(b-1)}{f_c(b) - f_c(b-1)} & \text{if } f_c(b-1) \le \omega(k) \le f_c(b) \\ \dfrac{\omega(k) - f_c(b+1)}{f_c(b) - f_c(b+1)} & \text{if } f_c(b) \le \omega(k) \le f_c(b+1) \\ 0 & \text{if } w(k) \ge f_c(b+1) \end{cases}$$

This finally allows us to compute the Mel-frequency spectrum with a simple sum over the magnitude values of the Fourier coefficients weighted by each of the B bands. This leads to B values $M(t,b)$ for $1 \le b \le B$:

$$M(t,b) = \sum_{k=0}^{N/2-1} d(b,k) \cdot |X(t,k)|$$

3. Cepstral Coefficients: the cepstrum can be interpreted as a spectrum of a spectrum. The newer variant of MFCC computes the coefficients of a discrete cosine transformation and uses the first half of the coefficients. If we started with $B = 26$, we now obtain 13 cepstral values $c(t,b)$ with $1 \le b \le B/2$:

$$c(t,b) = \sum_{j=1}^{B} M(t,j) \cdot \cos\left(\frac{b(2j-1)\pi}{2B}\right) \qquad \text{with } 1 \le b \le B/2$$

4. Derivatives: the actual MFCC features are a combination of the cepstral values $c(t, b)$ and the first and second order derivatives. The derivatives describe the dynamic nature of spoken text. With 13 cepstral coefficients, we obtain 39 feature values:

$$\Delta c(t,b) = c(t+1,b) - c(t-1,b)$$
$$\Delta\Delta c(t,b) = \Delta c(t+1,b) - \Delta c(t-1,b)$$

$$MFCC(t) = [c(t,1), \dots, c(t,B/2), \Delta c(t,1), \dots, \Delta c(t,B/2), \Delta\Delta c(t,1), \dots, \Delta\Delta c(t,B/2)]$$

MFCC are the standard features for speech recognition. The feature values are used either in Hidden Markov Models or neural network to learn phonemes. A typical approach is to use the cepstral coefficients of a large spoken text body, to cluster the values into l clusters with a k-means clustering approach (see next chapter), and to use the clusters to quantize the vector and to create l states. The machine learning method then derives a mapping form a series of state transitions to a phonem. The phonem stream is then further processed to create words.

- It is also possible to search directly on the phonem stream. The query words, with the help of a dictionary, are mapped to phonems, and search is over phonems as terms. The advantage is that we do not have to train the system to recognize (countless) names. If further allows for fuzzy retrieval and is helpful if some of the phonems are not correctly recognized. On the other side, we do not have a transcript for the presentation of the answers.

- **Feature in the Perceptual Domain (segment)**: we can compute moments or histograms of the perceptual features across frames in a segment as before. The standard deviation of the 2nd MFCC coefficient $c(t, 2)$, for instance, is very discriminative to distinguish speech from music.

5.2.2 Music Features (Pitch Contour)

- Chroma based features closely relate to the twelve different pitch classes from music {C, C♯, D, D♯, E ,F, F♯, G, G♯, A, A♯, B}. Each pitch class, e.g., C, stands for all possible pitches at all octaves. All pitches relate to each other by octave. If two pitches of the same class lie an octave apart, their frequency has the ratio of 1:2 (or 2:1), i.e., with each higher octave the frequency doubles. Another important concept of music theory are the partials, overtone, fundamental, and harmonics
  - Each pitched instrument produces a combination of sine waves, the so-called **partials**. The combination with its own frequencies and changes of amplitude over time define the characteristic timbre of the instrument. The human auditory system is extremely advanced to recognize timbres and to distinguish instruments (but also voices) from many audio sources.
  - The **fundamental** is the partial with the lowest frequency corresponding to the perceived pitch. **Harmonics** are a set of frequencies that are positive integer multiples of the fundamental frequency. Although an instrument may have harmonic and inharmonic partials, the design of an instrument is often such that all partials come close to harmonic frequencies.
  - **Overtone** refers to all partials excluding the fundamental. The relative strength of the overtones define the characteristic timbre of an instrument as it changes over time.

  The pitch standard **A440** (also known as A4 of Stuttgart pitch) defines the A of the middle C at $f_{A4} = 440$ Hz and serves as a tuning standard for musical instruments. If we number the pitch classes (also called semitones) with $n = 0$ (C), …, $n = 11$ (B), we can express the frequency of the semitones in the octave o with $-1 \le o \le 9$ as follows (MIDI number would be $12(o+1)$+n; A4 has number 69):

$$f_{A440}(o,n) = f_{A4} \cdot 2^{\frac{12o+n-57}{12}} = 440 \cdot 2^{\frac{12o+n-57}{12}}$$

- **Table of note frequencies** (standard piano key frequencies)

Octave → **Note↓**	$o=-1$	$o=0$	$o=1$	$o=2$	$o=3$	$o=4$	$o=5$	$o=6$	$o=7$	$o=8$	$o=9$
C $(n=0)$	8.176	16.352	32.703	65.406	130.81	261.63	523.25	1046.5	2093.0	4186.0	8372.0
C♯ / D♭ $(n=1)$	8.662	17.324	34.648	69.296	138.59	277.18	554.37	1108.7	2217.5	4434.9	8869.8
D $(n=2)$	9.177	18.354	36.708	73.416	146.83	293.66	587.33	1174.7	2349.3	4698.6	9397.3
E♭ / D♯ $(n=3)$	9.723	19.445	38.891	77.782	155.56	311.13	622.25	1244.5	2489.0	4978.0	9956.1
E $(n=4)$	10.301	20.602	41.203	82.407	164.81	329.63	659.26	1318.5	2637.0	5274.0	10548.1
F $(n=5)$	10.914	21.827	43.654	87.307	174.61	349.23	698.46	1396.9	2793.8	5587.7	11175.3
F♯ / G♭ $(n=6)$	11.563	23.125	46.249	92.499	185.00	369.99	739.99	1480.0	2960.0	5919.9	11839.8
G $(n=7)$	12.250	24.500	48.999	97.999	196.00	392.00	783.99	1568.0	3136.0	6271.9	12543.9
A♭ / G♯ $(n=8)$	12.979	25.957	51.913	103.83	207.65	415.30	830.61	1661.2	3322.4	6644.9	
A $(n=9)$	13.750	27.500	55.000	110.00	220.00	440.00	880.00	1760.0	3520.0	7040.0	
B♭ / A♯ $(n=10)$	14.568	29.135	58.270	116.54	233.08	466.16	932.33	1864.7	3729.3	7458.6	
B $(n=11)$	15.434	30.868	61.735	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1	

Source: https://en.wikipedia.org/wiki/Scientific_pitch_notation

- Extracting pitch information from audio files requires the extraction of the fundamentals. A first, simple approach, is to map all frequencies from the STFT to a chroma value corresponding to the pitch class numbering as introduced above. We use again the A440 standard with $f_{ref} = 440$. Let $\omega(k) = k \cdot f_s / N$ be the frequency mapping of the k-th Fourier coefficient with sampling rate f_s and with N samples. Then, the chroma value (pitch class $p(k)$ and octave $o(k)$), are given as:

$$p(k) = \left\lfloor 9.5 + 12\log_2\left(\frac{k \cdot f_s}{N \cdot f_{ref}}\right)\right\rfloor \bmod 12 \qquad o(k) = \left\lfloor \frac{1}{12}\left(9.5 + 12\log_2\left(\frac{k \cdot f_s}{N \cdot f_{ref}}\right)\right)\right\rfloor$$

  – We can obtain a chroma related histogram by summing over the power spectrum using above mappings to obtain the pitch class and octave. A histogram vector for frame F_i is then:

$$h_{chroma}(i,o,p) = \frac{1}{\sum_{k=0}^{K}|X(i,k)|^2} \cdot \sum_{k=0}^{K} \begin{cases} |X(i,k)|^2 & \text{if } o = o(k) \wedge p = p(k) \\ 0 & \text{otherwise} \end{cases}$$

  – However, this does not allow to obtain the main pitch contour (or pitches if polyphonic) but simply provides a mapping to chroma values. We can estimate the fundamental f_0 in a time window if we search for the frequency which maximizes the sum of magnitudes over all its harmonics, i.e.:

$$f_0 = \frac{f_s}{N} \cdot \operatorname*{argmax}_{0 \le k \le K}\left(\sum_{m=1}^{M} g(k,m) \cdot |X(i,km)|\right) \qquad g(k,m) = \frac{\omega(k) + 27}{\omega(km) + 320} = \frac{k\frac{f_s}{N} + 27}{km\frac{f_s}{N} + 320}$$

  $g(k,m)$ is an empirically obtained function to weight the contributions of the different harmonics. The number M is the number of considered harmonics and depends on the maximum frequency available in the spectrum.

- With the fundamental f_0, we obtain the pitch class $p(f_0)$ and the octave $o(f_0)$ of the time window. To extract several fundamentals from the frame, we repeat the following steps:
    1. Compute the magnitude spectrum $\left|X^{(0)}(i,k)\right|$
    2. Iterate $t = 0, 1, \ldots$ as long as $\sum_{k=0}^{K}\left|X^{(t)}(i,k)\right| > \epsilon$
        - Compute f_0 on the magnitude spectrum $\left|X^{(t)}(i,k)\right|$
        - Adjust the magnitude spectrum, i.e., subtract the magnitudes of the harmonics of the computed fundamental f_0 to obtain $\left|X^{(t+1)}(i,k)\right|$

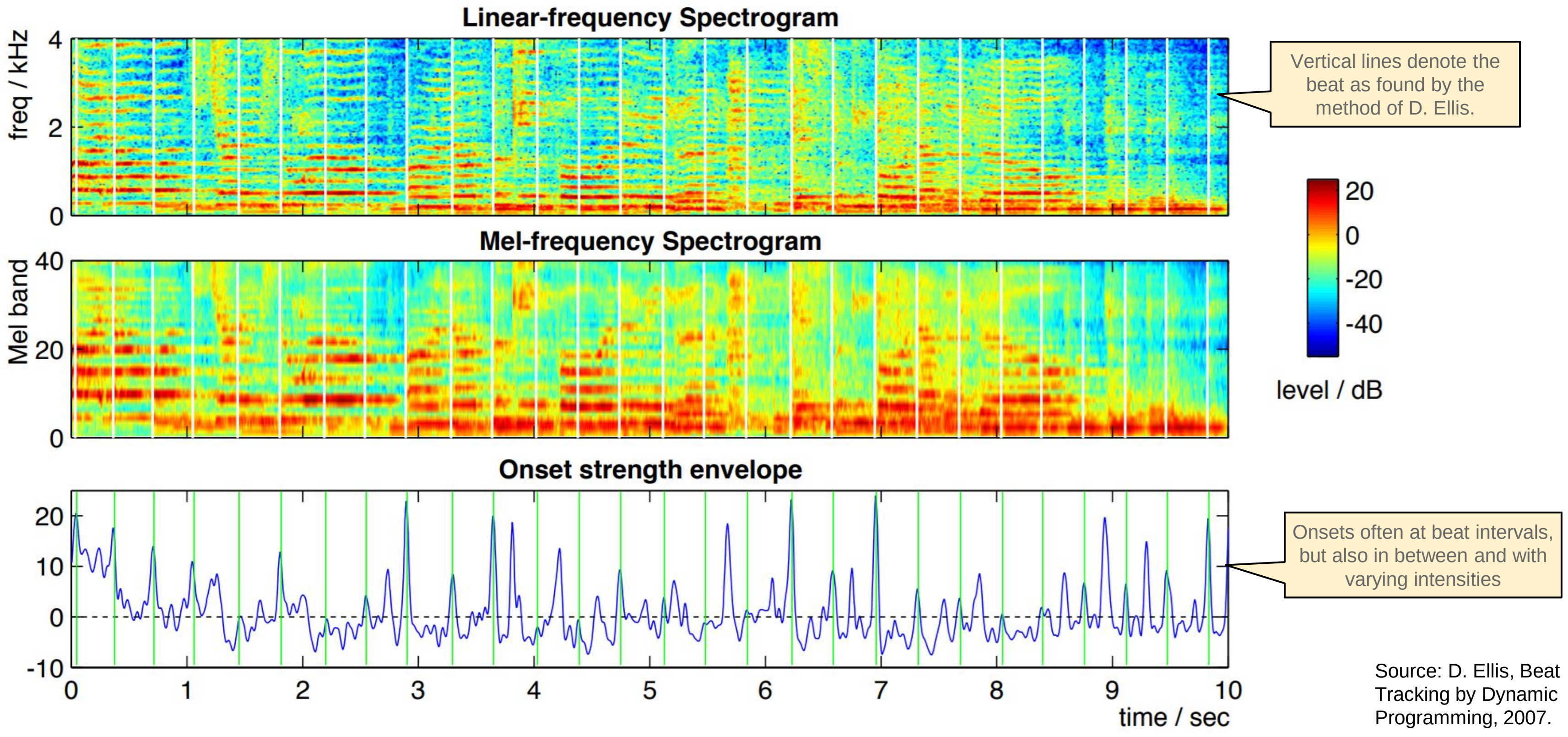
- Alternatively, we can compute the fundamental frequency f_0 in the time domain. To this end, we compute the autocorrelation of the audio signal at different time shifts Δt. Let N be the size of a frame and f_s be the sampling rate. To limit the search, we enforce the condition $1/f_{min} \geq \Delta\text{t} \geq 1/f_{max}$ for a minimum and maximum frequency range for the fundamental: $f_{min} \leq f_0 \leq f_{max}$. Furthermore, time shifts are integer multiples of the sampling period: $\Delta t = m/f_s$ with $f_s/f_{min} \geq m \geq f_s/f_{max}$. The autocorrelation for the frame F_i and the lag m is then defined as follows:

$$R(i,m) = \sum_{t=m}^{N} x(i,t) \cdot x(i,t-m)$$

To obtain the fundamental, we search for the lag m_0 that maximizes the autocorrelation and compute the frequency from this lag:

$$f_0 = \frac{f_s}{m_0} \qquad\qquad m_0 = \underset{f_s/f_{min} \geq m \geq f_s/f_{max}}{\operatorname{argmax}} R(i,m)$$

- Another music related feature is the tempo or beats per minute (bpm) of a play. In classical music, the tempo is often defined with ranges like Largo (40-60 bpm), Larghetto (60-66 bpm), Adagio (66-76 bpm), Andante (76-108 bpm), Moderato (108-120 bpm), Allegro (120-168 bpm), Presto (168-200 bpm), and Prestissimo (200+ bpm) and can vary over the play. Pop music has often a constant beat over the course of the song and bpms vary between 60 and 160, with 120 bpm being the most frequent choice for tempo.
  – Beat tracking is the search for regular onsets of energy at the beat intervals. With 100 bpm, we should observe an increase of energy at intervals of around 10ms (depending on the accuracy of the musician) indicating the beats. But it is not that straightforward as the example below shows:



Source: D. Ellis, Beat Tracking by Dynamic Programming, 2007.

- Onset envelope calculation: the onset is defined as the (positive) slope on the energy over the spectrum at a given point in time. Using STFT and mel bands, we obtain the mel spectrum $|X_{mel}(i,b)|$ as a weighted function from the frequency spectrum. The weighting is such that the areas underneath the triangular mel bands become equal. This firstly improves resolution of the lower frequencies and emphasizes them over the higher frequencies (basically a weighting by the inverse of the band width). The onset is then similar to the spectral flux, but we only consider positive slopes (hence onsets) and convert to decibels. Let B be the number of mel bans and F_i be the current frame, then the onset $o(i)$ is as follows:

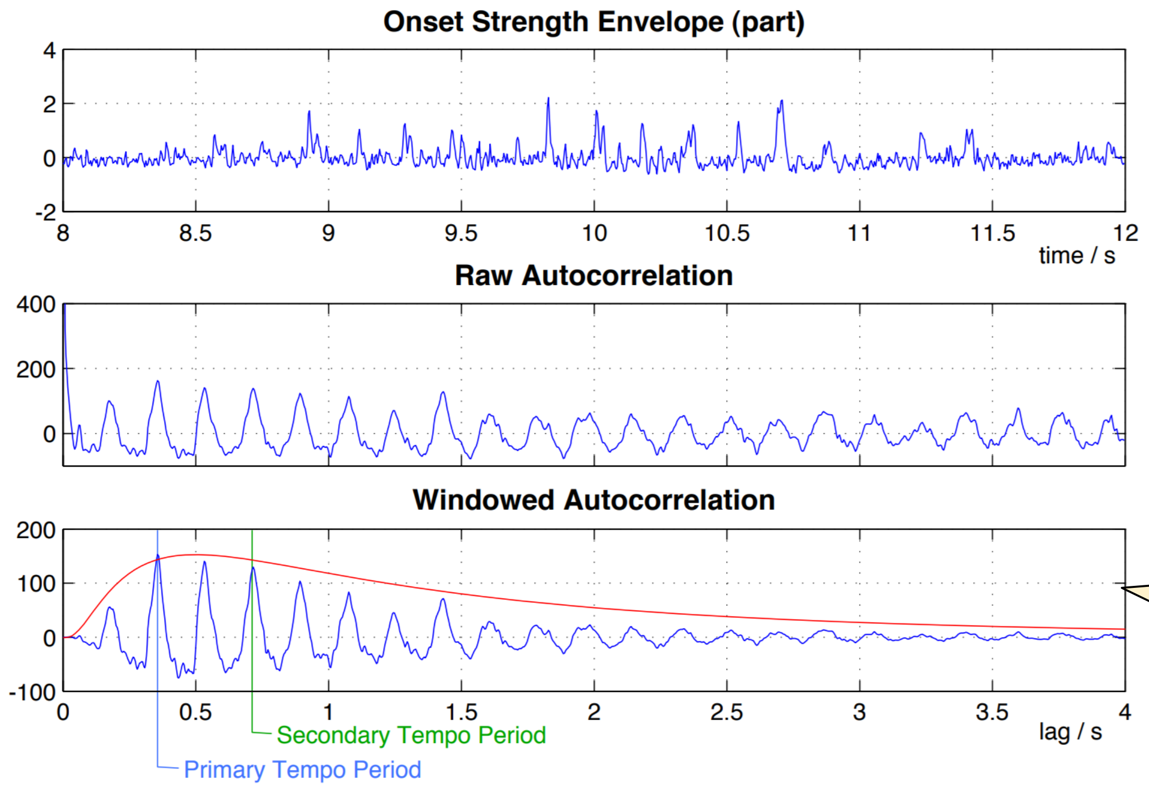
$$o(i) = \sum_{b=1}^{B} \max\left(0, \frac{\log_{10}|X_{mel}(i,b)|}{\log_{10}|X_{mel}(i-1,b)|} - 1\right)$$

- We can estimate the global tempo through autocorrelation over the onset $o(i)$ using a window function $w(i)$ (for instance using a Hann function). In other words, the we are looking for a time shift Δt such that peaks in the onset function coincide. The time shift that maximizes autocorrelation corresponds to the global tempo. We can compute the tempo per frame to obtain a tempogram, i.e., autocorrelations for the frame F_i, the lag l and the window function $w()$:

$$a(i,l) = \sum_{j=1}^{W} w(j) \cdot o(i) \cdot o(i+j)$$

  The tempo is given by the lag l_0 with the highest autocorrelation and we can convert to beats per minutes with $\Delta t = l_0/f_s$ and hence the tempo is $\frac{60}{\Delta t} = 60\frac{f_s}{l_0}$ bpm. Often, we find other peaks at $\{0.33l_0,\ 0.5l_0,\ 2l_0,\ 3l_0\}$ which mark secondary tempos if their autocorrelation is large enough. In addition, we can favor beats around, for instance, 120bpms if we know the genre (e.g., pop).

Example for tempo estimation within a time frame:



Weighting function prefers beats around 120bpm which is a lag of 0.5 seconds. Here the primary tempo is about 168, and the secondary at 84.

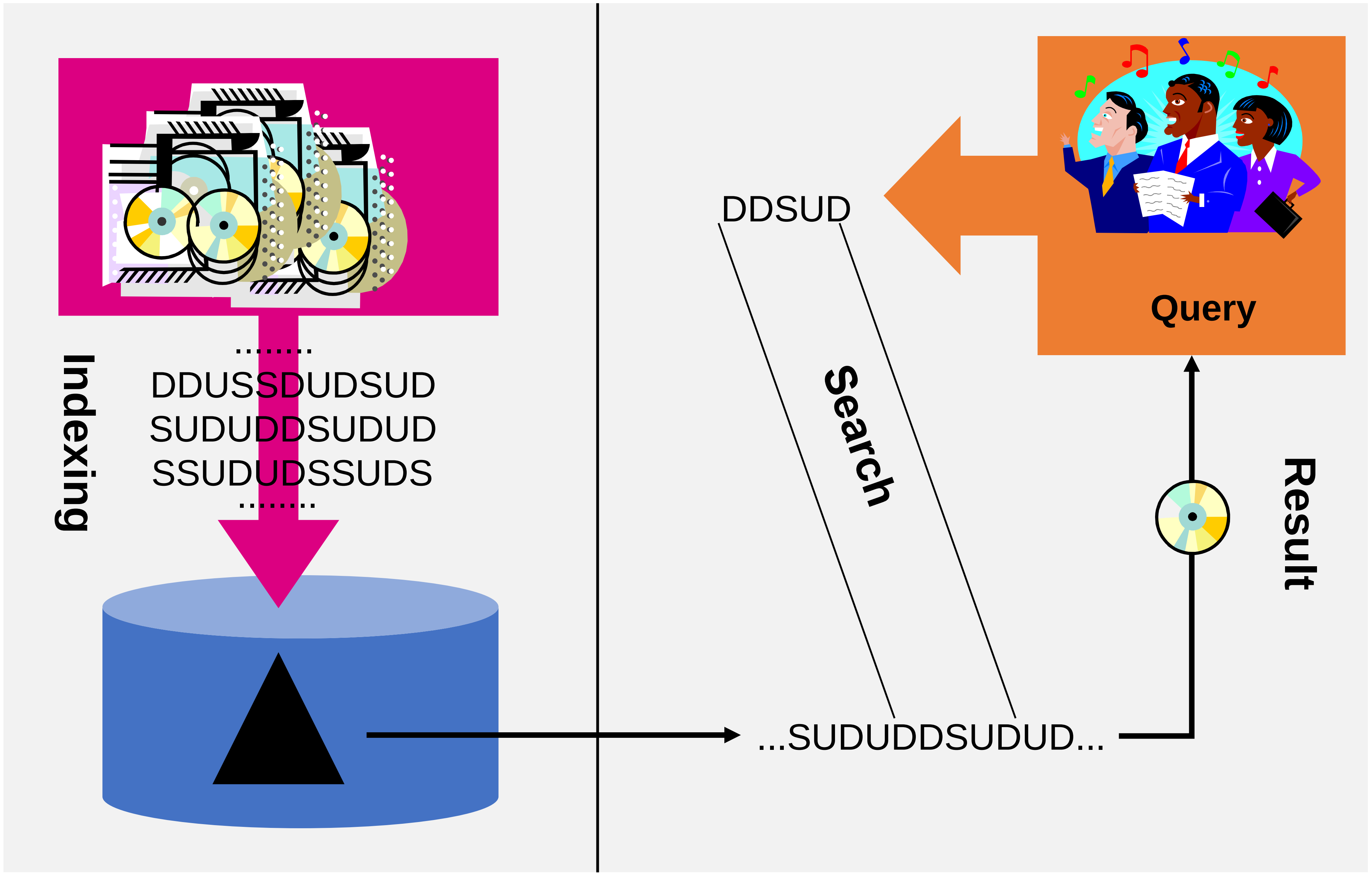
– Beat tracking is then the identification of the time points $\{t_i\}$ at which the onsets occur (as a human listener would tap to the music) and such that time intervals match the tempo (with some small deviations). These time points optimize the following objective function with $F(t_i - t_{i-1}, l_0)$ being a penalty function for deviations from ideal tempo and α a weighting to balance onset values and penalty values:

$$C(\{t_i\}) = \max_{\{t_i\}} \left(\sum_{i=1}^{T} o(f_s \cdot t_i) + \alpha \sum_{i=2}^{T} F(f_s \cdot (t_i - t_{i-1}), l_0) \right) \qquad F(\Delta l, l_0) = -\left(\log \frac{\Delta l}{l_0}\right)^2$$

5.2.3 Search for Tunes (Search by Humming)

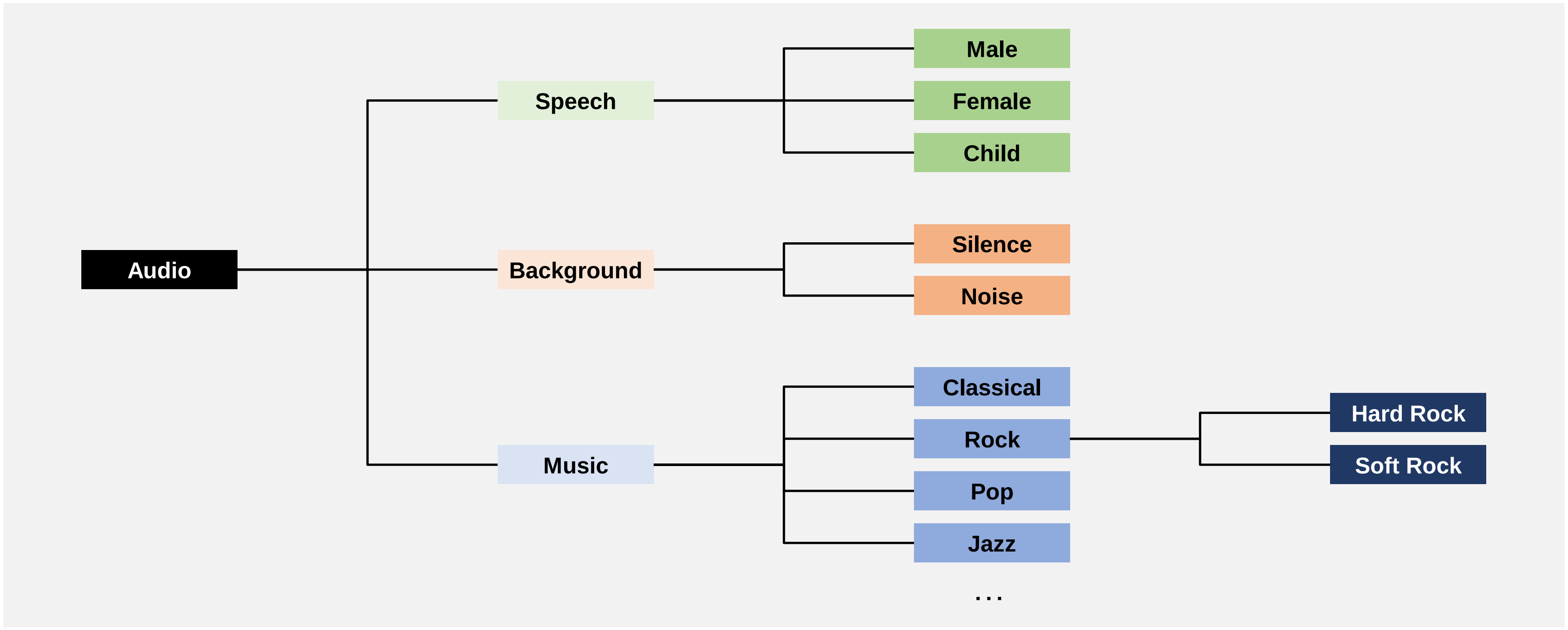
- With music, the tune is an important piece of information. Acoustical features like beat, tempo, or pitches are not sufficient for music related search. A tune, played a different pitch levels still appears similar. A tune at a slower tempo still appears similar. Hence, we need a better way to describe a tune and to find variations of it:
  - musipedia.org is a website offering different type of tune related searches including contour search and search by humming. The idea of contour search is to describe the relative changes of the tune. For each new pitch, we note:
    - **D** (down) if the preceding pitch was higher (tune goes down)
    - **U** (up) if the preceding pitch was lower (tune goes down)
    - **S** (same) / **R** (repeat) if the preceding pith is the same (tune stays flat)

    This transforms the stream of pitches to a stream with three terms (D, U, S). In this simple case, the duration of a pitch and pauses between pitches are ignored.
  - To search for music, one can hum the tune and the recording interface translates the humming into a sequence of terms following the above notation. The search becomes a simple string search in a database of songs.
  - There are many variations for contour search, i.e., taking duration or the step size between notes into account. Again, this translate into a contour but with additional terms. On the other side, as duration is not normalized, users may have more difficulties to hum the correct melody. The same with pitch differences: not everyone is pitch perfect but often we can remember the contour. Such interfaces are often for the more professional users.


DDSUD
Query
Search
Result
Indexing
....
DDUSSDUDSUD
SUDUDDSUDUD
SSUDUDSSUDS
....
...SUDUDDSUDUD...

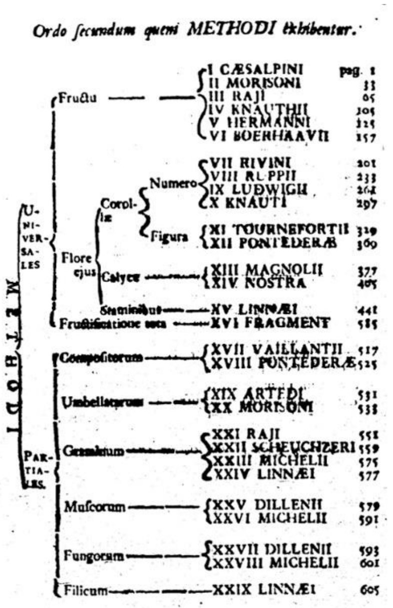
5.3 Audio Classification with Decision Trees

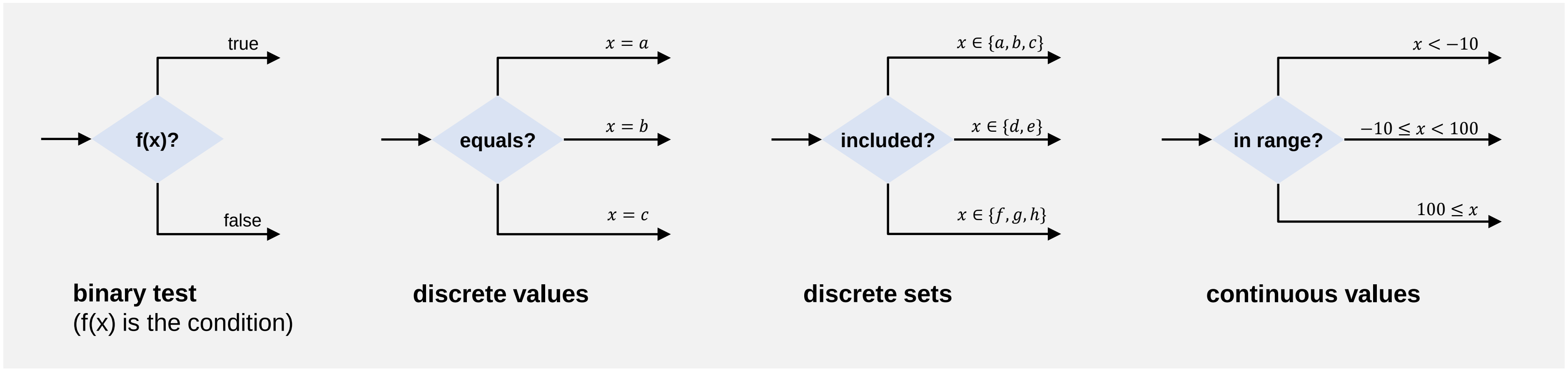
- Classification is a key concept to obtain higher-level features. The usual approach is to extract low-level features from the signal, normalize and transform the features, and deduce a mapping to pre-defined categories. Let us consider an audio database with a simple classification as follows:



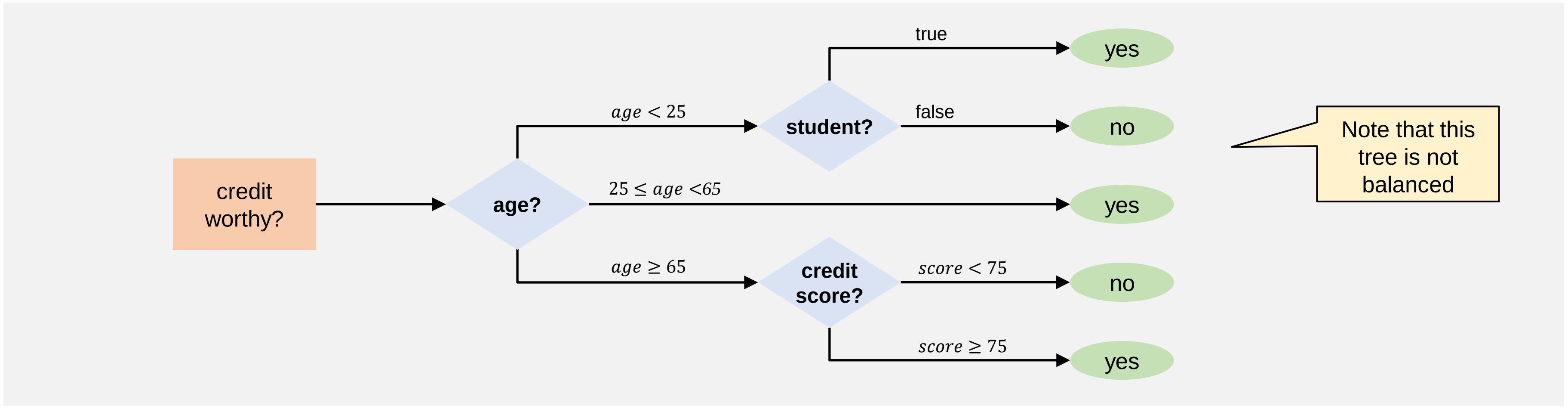
- Decision tree learning is a simple but effective classification approach. We start with a data set that has discrete and continuous features and given labels (targets for objects), and then create the “optimal” decision hierarchy to map the features with a series of tests to their labels. The resulting classification tree is easy understandable by humans and machines and can create efficient rules for classification, i.e., predicting the class with a minimal number of tests.

- The concept of classification trees is quite old. An early example is the classification scheme of Carl Linnaeus (1735) for plants (see right hand figure) and animals. Each node represents a test and each branch to the right denotes a possible outcome of the test. Leaf nodes, finally, contain the class labels. The tree does not have to be balanced and different numbers of tests may be required to reach a leaf node.
- A node in a classification tree usually tests for a single feature only. If the feature is discrete (a set of values), a node partitions the values into distinct sets (or just individual values) each with a separate branch out. The test in the node checks which partition includes the feature value. If the feature is continuous, the branches are given by distinct ranges in the feature domain. Features can be multi-dimensional but it is more common to treat each dimension as an individual ("orthogonal") feature achieved through dimensionality reduction. A special case is the binary test node which yields "true" if a condition on the feature is met and otherwise no. In many cases, nodes branch always into exactly two children (binary decision trees) but actually any number of branches is possible. Examples:





- The leaf nodes denote the labels (or targets) associated with the objects. The series of test should deterministically lead to a leaf node and thus the label. Example:



- In order to create a decision tree, the machine learning approach must identify a set of tests against the features of the training data sets that lead to the observed labels with a minimal number of steps. Once the tree is learned, we can follow the decision hierarchy for a new data instance until a node is reached. The label in the node is our prediction for that new data instance.
  – Note: the condition “minimal number of steps” leads to the most simple tree that maps features to labels following Occam’s razor (i.e., prefer simple solutions over complex ones)

- To construct decision trees, we will use a fundamental concept from information theory: **information gain**. In a nutshell, the information gain is the reduction of entropy given the observation that a random variable has a given value. With this in mind, we build test nodes in the decision tree such that they maximize the information gain, i.e., choose a feature and conditions on it that reduces the uncertainty of the outcome (here: label) as much as possible.
  - Let $\mathbb{T}$ be the training set of the form $(\boldsymbol{x}, y) = (x_1, x_2, x_3, \ldots, x_M, y)$ where x_j is the j-th feature value with values from $\mathbb{V}_j$ and y the target label. The expected information gain is then a function of entropy H. Let $\mathbb{T}_{j,v} = \{\boldsymbol{x} \in \mathbb{T} \mid x_j = v\}$ be the subset of $\mathbb{T}$ such that all elements have $x_j = v$:

$$IG(\mathbb{T}, x_j) = H(\mathbb{T}) - \sum_{v \in \mathbb{V}_j} \frac{|\mathbb{T}_{j,v}|}{|\mathbb{T}|} H(\mathbb{T}_{j,v})$$

    Entropy H is defined on the probabilities of the target labels y_i. $P(y_i)$ denotes the probability that a randomly selected item from $\mathbb{T}$ has the label $y = y_i$. We can estimate these probabilities through simple counting of labels in the training set.

$$H(\mathbb{T}) = -\sum_i P(y_i) \cdot \log_2(P(y_i))$$

$$H(\mathbb{T}_{j,v}) = -\sum_i P(y_i | x_j = v) \cdot \log_2\left(P(y_i | x_j = v)\right)$$

Entropy is usually based on $\log_2$ but for the purposes here, the basis of the logarithm is irrelevant

    In summary, the idea of information gain is to measure whether the entropy (uncertainty about the distribution of the target labels) would decrease if we split the data set along the feature x_j

– Example: consider the table on the right hand side. There are four features x_j in the first columns and a target y (“can we play tennis?”) in the last column. Let us compute the information gain if we choose $j = Windy$. The entropy $H(\mathbb{T})$ is obtained as:

Outlook	Temp.	Humidity	Windy	Play
Sunny	Hot	High	FALSE	No
Sunny	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Rainy	Mild	High	FALSE	Yes
Rainy	Cool	Normal	FALSE	Yes
Rainy	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Sunny	Mild	High	FALSE	No
Sunny	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	FALSE	Yes
Sunny	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rainy	Mild	High	TRUE	No

$$H(\mathbb{T}) = -\sum_{y\in\{Yes,No\}} P(y)\cdot\log_2(P(y)) = -\frac{9}{14}\cdot\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\cdot\log_2\left(\frac{5}{14}\right) = 0.9403$$

14 entries with 9 ‘Yes’ and 5 ‘No’

The entropy given the observation of $x_j = v$ for $j = Windy$ with $\mathbb{V}_{Windy} = \{TRUE, FALSE\}$ is:

6 TRUE entries with 3 ‘Yes’ and 3 ‘No’

$$H(\mathbb{T}_{j,TRUE}) = -\sum_{y\in\{Yes,No\}} P(y|x_j = TRUE)\cdot\log_2\left(P(y|x_j = TRUE)\right) = -\frac{3}{6}\cdot\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\cdot\log_2\left(\frac{3}{6}\right) = 1$$

$$H(\mathbb{T}_{j,FALSE}) = -\sum_{y\in\{Yes,No\}} P(y|x_j = FALSE)\cdot\log_2\left(P(y|x_j = FALSE)\right) = -\frac{6}{8}\cdot\log_2\left(\frac{6}{8}\right) - \frac{2}{8}\cdot\log_2\left(\frac{2}{8}\right) = 0.8113$$

8 FALSE entries with 6 ‘Yes’ and 2 ‘No’

Leading to an information gain $IG(T, x_j)$ of:

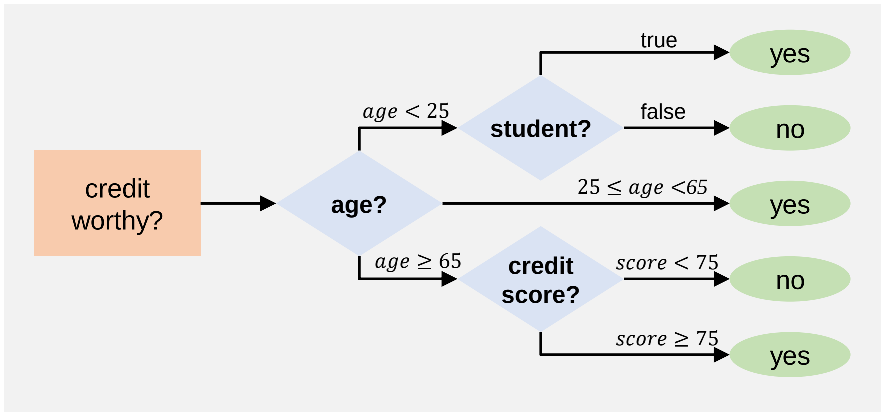
$$IG(T, x_j) = H(\mathbb{T}) - \sum_{v\in\{TRUE,FALSE\}} \frac{|\mathbb{T}_{j,v}|}{|\mathbb{T}|} H(\mathbb{T}_{j,v}) = 0.9403 - \frac{6}{14}\cdot 1 - \frac{8}{14}\cdot 0.8113 = 0.0481$$

14 entries with 6 ‘TRUE’ and 8 ‘FALSE’

- A high-level pseudo code for constructing a decision tree is given as follows

```
Function DecisionTree(Features, Targets)
    TrainingSet, ValidationSet, Attributes ← CleanseData(Features, Targets)
    Root ← BuildTree(TrainingSet Attributes)
    Rules ← PruneTree(Root, ValidationSet)
    Return Rules
```

  – We can re-write a decision tree as a set of rules where each rule denotes a path from the root to a leaf with all tests along the path and the label of the leaf. Depending on the programming context, the algorithm can produce native implementations with the high computational efficiency (while traversing the decision tree is sub-optimal). For instance:



Note that 'true' and 'false' are labels and not Boolean values in the rules

```
Rule Set:
   yes ← (age<25) AND (student=='true')
   no  ← (age<25) AND (student=='false')
   yes ← (25<=age) AND (age<65)
   no  ← (age>=65) AND (score<75)
   yes ← (age>=65) AND (score>=75)
```

rules compiler

```
public boolean isCreditWorthy(Customer c) {
    if c.getAge()<25  && c.isStudent()              return true;
    if c.getAge()<25  && !c.isStudent()             return false;
    if 25<=c.getAge() && c.getAge()<65              return true;
    if c.getAge()>=65 && c.getCreditScore()<75      return false;
    if c.getAge()>=65 && c.getCreditScore()>=75     return true;
    return false; // default: false
}
```

Further optimizations of code generation possible

- **Cleanse data:** our running example from the previous page had unique rows, i.e., there are no two entries with the exact same feature values. In practice, however, there will be several observations with the same feature values, and more importantly, they may have conflicting labels. In addition, some feature values may be missing, or labels are not given. Not all features might be useful for classification. E.g., having a column “Date” in our running example would not help us to identify good rules for classification. Further transformations on the data are possible depending on the domain. This can include outlier and noise elimination, or dimensionality reduction:

```
Function CleanseData(Features, Targets)
    Features, Targets ← eliminate entries with missing Targets (=NULL) and outliers
    Features ← predict missing Features (=NULL) with domain knowledge
    Features ← transform and normalize Features with domain knowledge
    Attributes ← select set of useful Features with domain knowledge

    collapse entries that share the same Features
    assign the most frequent label from Targets to the collapsed entry
    keep Counts (=number of entries) for correct entropy calculations later on
    Data ← combine Features, Targets, and Counts into a structure

    TrainingSet, ValidationSet ← Split Data into distinct sets with given Ratio (e.g., 70:30)
    Return TrainingSet, ValidationSet, Attributes
```

  – Note: most of the data cleansing and feature selection is domain dependent. Although there are generic approaches for data preparation such as dimensionality reduction, clustering and outlier elimination, most of the manual work goes into a good feature design with the goal to have as few features as possible with minimal correlation between each other and the ability to separate target values.

Additional information — not part of the exams

- **Build tree:** Tree construction is recursive. At each iteration, a new node is inserted with a test on a selected attribute. The algorithm is called for each branch until the subset is empty or has one label

```
Function BuildTree(Data, Attributes)
    N ← new Node and associate most common label in Targets with node N
    If all Targets have same label Then Return N
    If Attributes is empty OR Data too small Then Return N
    A, Tests, Fitness ← SelectBestAttribute(Data, Attributes)
    If Fitness below Threshold Then Return N
    ForEach T in Tests Do
        B ← add new branch to node N for test T
        P ← get partition of Data which fulfills test T
        If P is empty Then add new (empty) node below branch B with same label as node N
        Else C ← BuildTree(P, Attributes – {A}); add node C below branch B
    End
    Return N

Function SelectBestAttribute(Data, Attributes)
    ForEach A in Attributes
        Tests[A], Partitions ← split feature values for attribute A and determine partitions
        Fitness[A] ← determine a fitness/score for attribute A (e.g., information gain)
    End
    Abest ← find A with Fitness[A]==max(Fitness)
    Return Abest, Tests[Abest], Fitness[Abest]
```

The typical approach is to use an attribute only once on each decision path in the tree. Hence, tree height is limited by the number of selected attributes.

  – We can observe that attributes are only used once during classification. We may relax this condition for continuous features to enable finer interval splits later in the tree.
  – The scoring function determines how well an attributes helps us to decide quickly along the paths in the decision tree. In ID3 this is the information gain as introduced before; extensions such as C4.5 balance this with the ability of the attribute to generalize.
  – We will discuss further details when we review concrete implementations like ID3 and C4.5.

Additional information — not part of the exams

- **Prune tree:** decision trees tend to overfit to the training set due to their recursive creation of nodes until no further attribute split is possible. As a consequence, generalization to new data sets may be poor. As we discussed earlier, a validation step allows a machine learning approach to compromise training errors with the ability to generalize. To do so, the pruning step eliminates tests that are not significantly improving performance against the validation set (remember Occam's razor). Pruning can also be done during building time: in `BuildTree()`, if the data set is too small or if the split along an attribute is not significant enough (fitness too small), the algorithm stops the recursion. We illustrate a few pruning techniques:
  – Elimination of branches: we assess the performance against the validation set, for instance, using accuracy (percentage of correct predictions). Then, we visit decision nodes and replace the subtree underneath them with leaf nodes if that improves overall accuracy

```
Function PruneTree(Root, ValidationSet)
    Repeat
        Accuracy ← get total accuracy for ValidationSet
        ForEach N underneath Root
            If N is leaf Then Accuracy[N]=Accuracy
            Else
                replace subtree at node N with leaf (keep label of N = most common target)
                Accuracy[N] ← get total accuracy for ValidationSet
                insert original N into the tree again
            End
        End
        N ← find node N with AccuracyNode[N]==max(AccuracyNode)
        If AccuracyNode[N]>Accuracy Then replace subtree at node N with leaf
    Until AccuracyNode[N]<=Accuracy

    Return (Rules ← create rule set given the tree underneath node Root)
```

This pseudo-code is obviously not optimized for speed but rather shows the steps that are necessary for pruning

Additional information — not part of the exams

- Pruning rules: Each rule contributes to the overall accuracy for the data items that pass through it. Initially, rules are not sorted because they are mutually exclusive (i.e., each data item can fulfill exactly one rule). The ‘pruning rules’ approach considers each condition in the rules and eliminates them if that improves overall accuracy. As a side effect, rules are no longer distinct and need to be sorted by their contribution to the overall accuracy.

This pseudo-code is obviously not optimized for speed but rather shows the steps that are necessary for pruning

```
Function PruneTree(Root, ValidationSet)
    Rules ← create rule set given the tree underneath node Root
    Repeat
        Accuracy ← sort Rules by accuracy; get total accuracy for ValidationSet
        ForEach R in Rules
            ForEach condition C in R
                remove condition C in R
                AccuracyRule[R][C] ← get total accuracy for ValidationSet
                insert condition C into R again
            End
        END
        R,C ← find rule R and condition C with AccuracyRule[R][C]==max(AccuracyRule)
        If AccuracyRule[R][C]>Accuracy Then remove condition C in R
    Until AccuracyRule[R][C]<=Accuracy

    Return (Rules ← sort Rules by accuracy)
```

Additional information — not part of the exams

- Implementations (selected examples):
  - The **ID3 algorithm** was invented by Ross Quinlan in 1986. It only worked for attributes with discrete values and used the information gain to select attributes. For each attribute x_j and each value in $\mathbb{V}_j$, the training set $\mathbb{T}$ is split into subsets $\mathbb{T}_{j,v}$ with $v \in \mathbb{V}_j$. The information gain is:

$$IG(\mathbb{T}, x_j) = H(\mathbb{T}) - \sum_{v \in \mathbb{V}_j} \frac{|\mathbb{T}_{j,v}|}{|\mathbb{T}|} H(\mathbb{T}_{j,v})$$

To compute the entropy $H(\mathbb{T})$ over the K labels y_k, we simply count the frequencies $f_k(\mathbb{T})$ of y_k in the set $\mathbb{T}$. Similarly, for the subsets $\mathbb{T}_{j,v}$, the frequencies are given by $f_k(\mathbb{T}_{j,v})$. This leads to:

$$IG(\mathbb{T}, x_j) = -\sum_{k=1}^{K} \frac{f_k(\mathbb{T})}{|\mathbb{T}|} \cdot \log_2\left(\frac{f_k(\mathbb{T})}{|\mathbb{T}|}\right) + \sum_{v \in \mathbb{V}_j} \frac{|\mathbb{T}_{j,v}|}{|\mathbb{T}|} \sum_{k=1}^{K} \frac{f_k(\mathbb{T}_{j,v})}{|\mathbb{T}_{j,v}|} \cdot \log_2\left(\frac{f_k(\mathbb{T}_{j,v})}{|\mathbb{T}_{j,v}|}\right)$$

The best attribute x_{j^*} maximizes the information gain, hence:

$$j^* = \underset{j}{\operatorname{argmax}}\, IG(\mathbb{T}, x_j) = \underset{j}{\operatorname{argmax}} \sum_{k=1}^{K} \sum_{v \in \mathbb{V}_j} f_k(\mathbb{T}_{j,v}) \cdot \log_2\left(\frac{f_k(\mathbb{T}_{j,v})}{|\mathbb{T}_{j,v}|}\right)$$

Since we are looking for the maximum value, the base of the logarithm is irrelevant.

If $\mathbb{T}_{j,v}$ is empty, the summand evaluates to $0 \cdot \log_2 \frac{0}{0} = 0$, i.e., empty partitions are simply ignored. Similarly, if $f_k(\mathbb{T}_{j,v}) = 0$, the summand evaluates to $0 \cdot \log_2 \frac{0}{|\mathbb{T}_{j,v}|} = 0$.

  - Decision nodes only exists for discrete attributes. Partitioning is straightforward: for each possible value of the attribute, its partition contains all training items that have that value. Should a partition be empty (e.g., at that level of the tree no item has the value), prediction assume the most common label of the node.

– Ross Quinlan refined the ID3 algorithm and published the **C4.5 algorithm** in 1993. It got quite popular after ranking #1 in a Springer LNCS publication "10 top data mining algorithms (2008)". C4.5 addresses many of the disadvantages of the original ID3 algorithm:

  - The information gain measure favors attributes with many values increasing the risk of overfitting the training data. Quinlan improved selection of attributes with the so-called split information. It is given as the entropy with respect to the attribute values rather than with respect to the target values as usually in this section. For each attribute x_j with discrete values $v \in \mathbb{V}_j$, the training set $\mathbb{T}$ is split into subsets $\mathbb{T}_{j,v}$. The split information $SI(\mathbb{T}, x_j)$ is:

$$SI(\mathbb{T}, x_j) = -\sum_{v \in \mathbb{V}_j} \frac{|\mathbb{T}_{j,v}|}{|\mathbb{T}|} \cdot \log_2 \frac{|\mathbb{T}_{j,v}|}{|\mathbb{T}|}$$

    The gain ratio is then the ratio between information gain and split information:

$$GR(\mathbb{T}, x_j) = \frac{IG(\mathbb{T}, x_j)}{SI(\mathbb{T}, x_j)}$$

    A practical issue, however, occurs if one $\mathbb{T}_{j,v}$ is almost as big a $\mathbb{T}$. This leads to a split information that is close to zero and hence a very large gain ratio. Clearly, such attributes are not interesting for decision nodes as most entries lie in the same branch. To counter this anomaly, the attribute selection process works as follows:

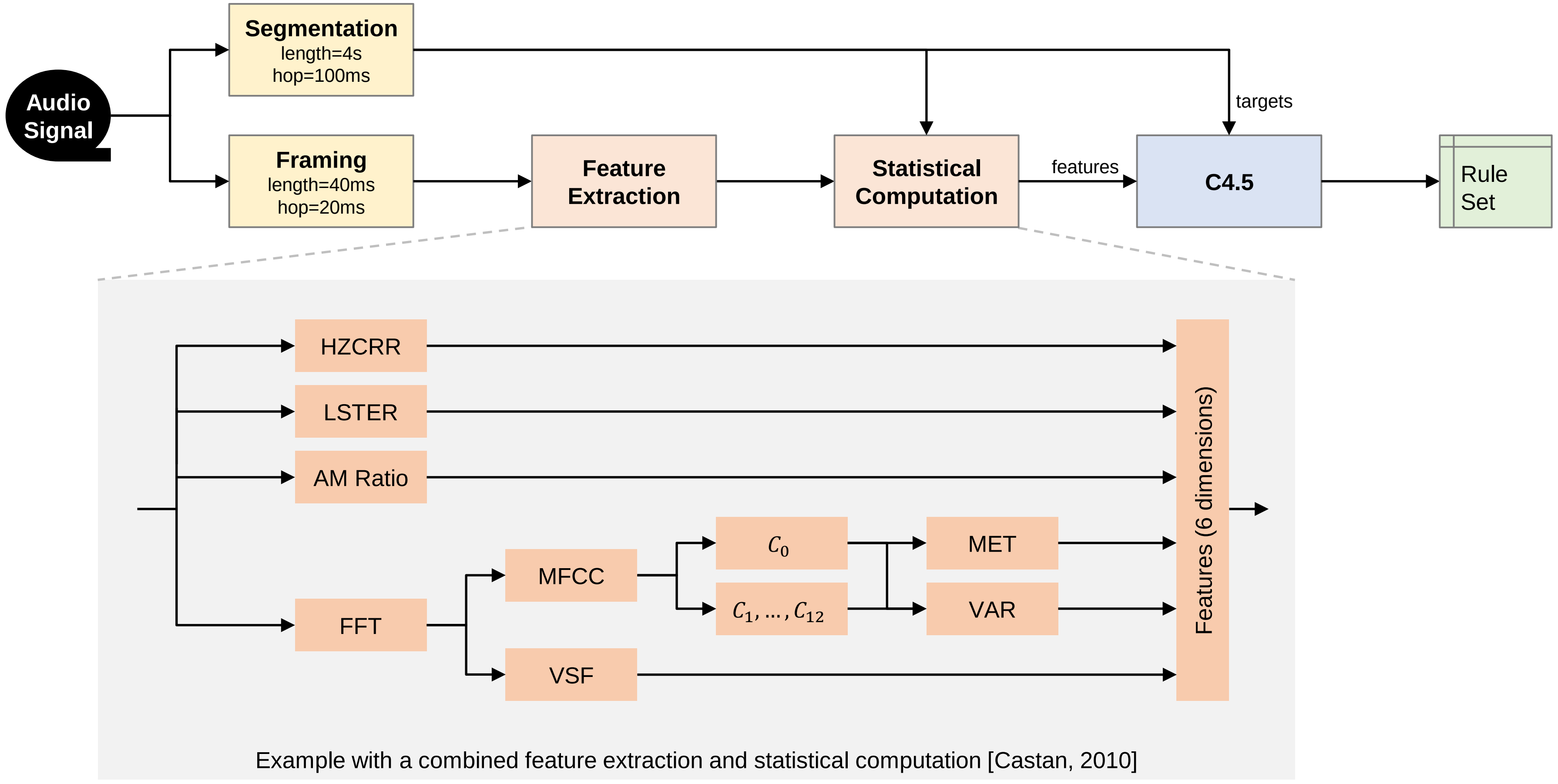
    - compute $IG(\mathbb{T}, x_j)$ for all x_j
    - select a threshold $IG_{Threshold}$, for example:
      - $IG_{Threshold} = \text{avg}\left(IG(\mathbb{T}, x_j)\right)$ (mean information gain)
      - $IG_{Threshold} = P_{50}\left(IG(\mathbb{T}, x_j)\right)$ (median information gain, 50-percentile)
    - $j^* = \underset{j;\, IG(\mathbb{T},x_j) > IG_{Threshold}}{\text{argmax}} \left(\frac{IG(\mathbb{T},x_j)}{SI(\mathbb{T},x_j)}\right)$

- To allow for continuous values in decision trees, C4.5 maps a continuous attribute x_j to a Boolean attribute with a simple threshold value τ. If $x_j < \tau$, then the value is ‘true’, otherwise it is ‘false’. So how can we select the best threshold value? Consider the example

Temperature	40	48	60	72	80	90
Play	**No**	**No**	**Yes**	**Yes**	**Yes**	**No**

  Obviously, we want to select a threshold value that maximizes the information gain. By sorting the training set according to the attribute values, we only need to identify adjacent data items with different targets, and create threshold candidates in between their values. For the example above, the first candidate is between 48 (No) and 60 (Yes), so we select $(48 + 60)/2 = 54$. The second candidate is between 80 (Yes) and 90 (No), so it is $(80 + 90)/2 = 85$. This produces two decision criteria $Temperature_{<54}$ and $Temperature_{<85}$ for which we compute the information gain and select the better one.
  There are extensions that map a continuous attribute to several intervals instead of just two as proposed by the C4.5 algorithm. In such cases, a balance is required to avoid overfitting to the intervals of the training set (e.g., using the validation set and an alternate scoring).
- There are several strategies to address missing values: the most simple one is to dismiss the object further down the branch if a test cannot be performed. This, however, disables also predictions for new data sets with missing values. A better strategy is to assign the most common value for the attribute at the current node (either for all training items or for only those that share the same target) and continue with this new value. A more complex approach is to compute distributions across all values, and use fractions for each value when following the branch. For instance, if there are two values, and if 40% of the data items have value 1 and 60% have value 0, a data item missing this attribute will be split and used in both branches but only counting for a fraction (0.4 in the branch for 1, 0.6 in the branch for 0).

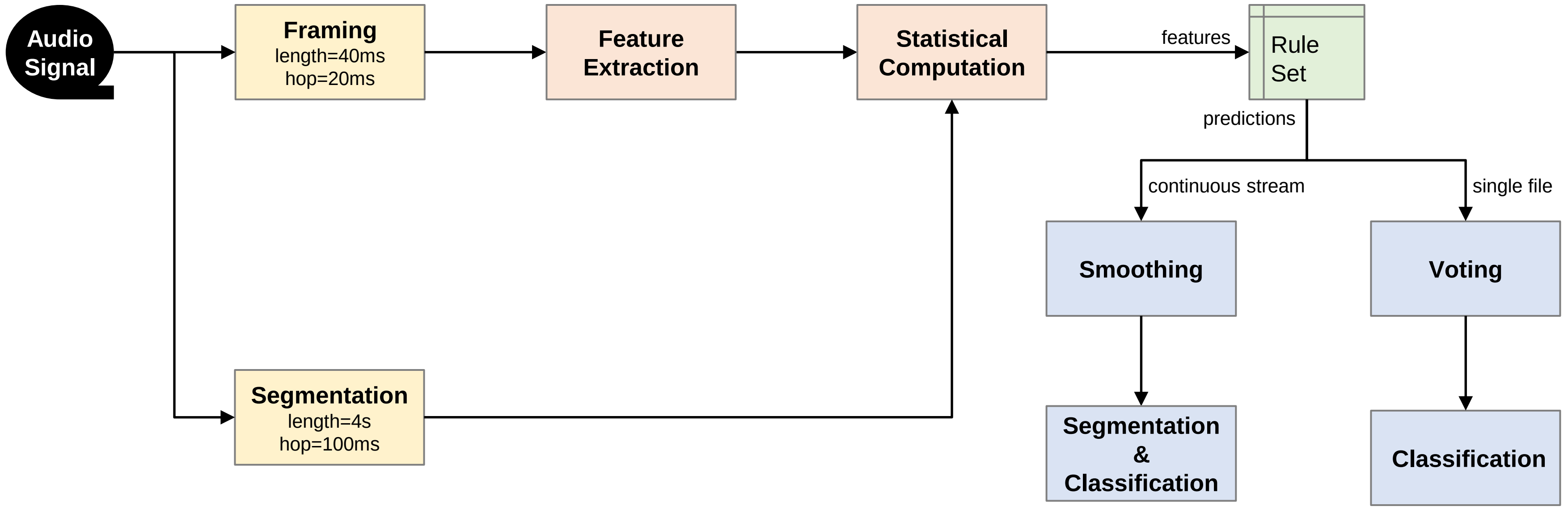
- **Example: audio classification**
  - Decision trees are very simple and produce efficient classifiers that are more than sufficient for many tasks. An example is discussed here: classify audio signals into speech and music.
  - In the learning phase, we need to pre-process the audio signal, extract features, gather statistical information about features and their mapping to output classes (music, speech), and select the best features for classification. In this example, we use C4.5 to select features and derive rules.



Example with a combined feature extraction and statistical computation [Castan, 2010]

- Framing and Segmentation: the audio signal is processed in overlapping frames and segments. Each frame and segment has the same length, and the hop distance specifies when the subsequent frame/segment starts. Typically, features are extracted per frame, and statistical measures are applied for the segment over its frame.
- Castan (2010) focused on a small number of characteristic features:
  - **HZCRR**: The Zero-Crossing Rates (ZCR) measures how often the amplitude of the signal passes the 0-value within a frame. The High Zero-Crossing Rate Ratio (HZCRR) measures, per segment, the ratio (percentage) of ZCR values of frames in the segment that are 1.5 times higher than the average ZCR value of frames in the segment.
  - **LSTER**: The Short Time Energy (STE) is simple the sum of squared amplitude of the signal within the frame (a measure of energy in the frame). The Low Short Time Energy Ratio measures, per segment, the ratio (percentage) of STE values of frames in the segment that are smaller than 50% of the average STE value of frames in the segment.
  - **AMR**: The Amplitude Modulation Ratio (AMR) measures the low-pass energy of a frame, i.e., the sum of squared amplitude after applying a low-pass filter with cut-off at 25Hz. It then measure the ratio of highest energy over lowest energy over all frames in the segment. Speech has a much higher ratio than music due to gaps between vowels and consonants.
  - **VSF**: The Spectral Flux (SF) is the Euclidean distance between subsequent frames over their fourier transformed signals (spectrum magnitudes). The Variation of Spectral Flux (VSF) measures the variance over the frames in the segment.
  - **MET & VAR**: For each frame, we extract 13 Mel-Frequency Cepstrum Coefficients (MFCC) denoted as $C_0, \ldots, C_{12}$. The Minimum-Energy Tracking (MET) measure how long C_0 is above a threshold. Pauses in speech will result in short lengths. VAR sums the variance of all MFCC over the frames in the segment. Small VAR values indicate music.

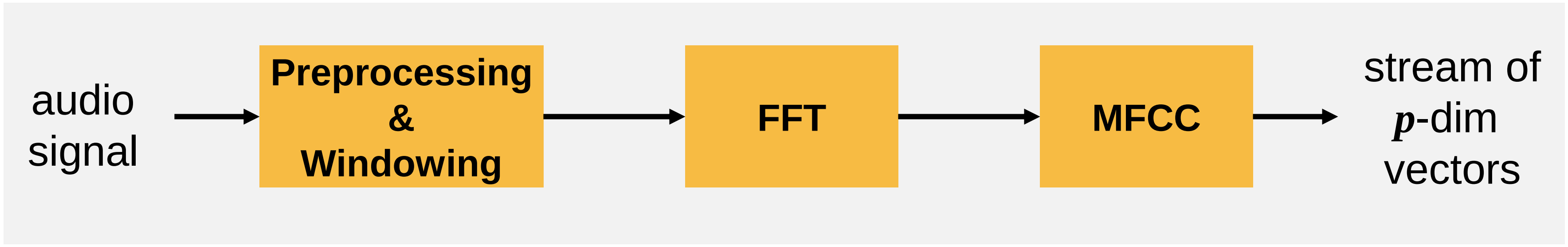
– In the prediction phase, we need to perform the same pre-processing, windowing, feature extraction, and statistical computations as in the learning phase. In addition, we want to smooth the results over the entire duration of the song (voting based approach) or to segment a continuous audio signal (e.g., radio broadcast) to detect changes from speech to music.



- Smoothing uses weighted sums over past predictions with exponentially smaller weighs to avoid fast alteration between targets. If enough support for a change is present, segmentation closes the current segment (not to be confused with the segments used for feature extraction) and labels it with the last class label. Then it marks the start of a new segment.
- Voting is rather simple: the single file is classified either by the label most frequently predicted for its segments, or classification returns probabilities for labels based on their frequencies in the predictions over all segments of the single file.

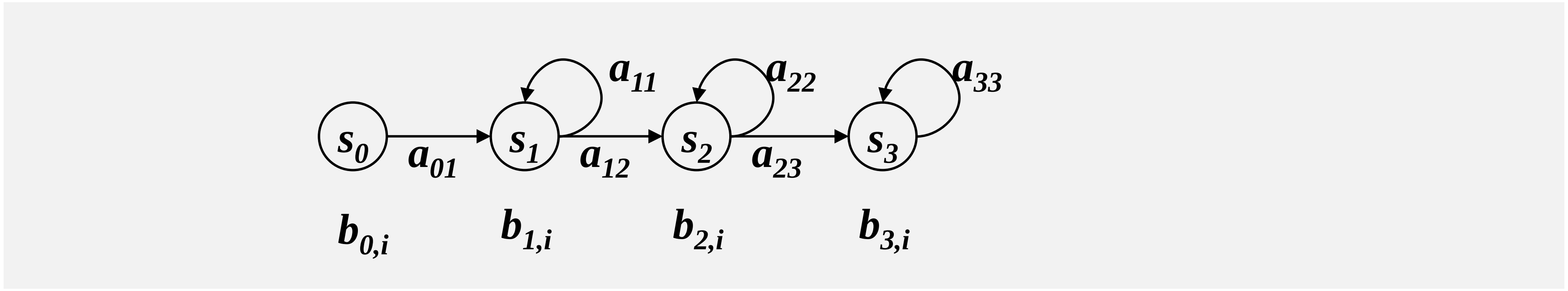
5.4 Spoken Text

- We are not going too deep into speech recognition. We rather give a short overview of the techniques and discuss the retrieval aspects.
  - In a first step, the audio signal needs to be pre-processed to reduce noise. For instance, the pitch level of a speaker, the tempo, or the loudness should not influence the result at all. The usual method is to use the MFCC approach discussed earlier.

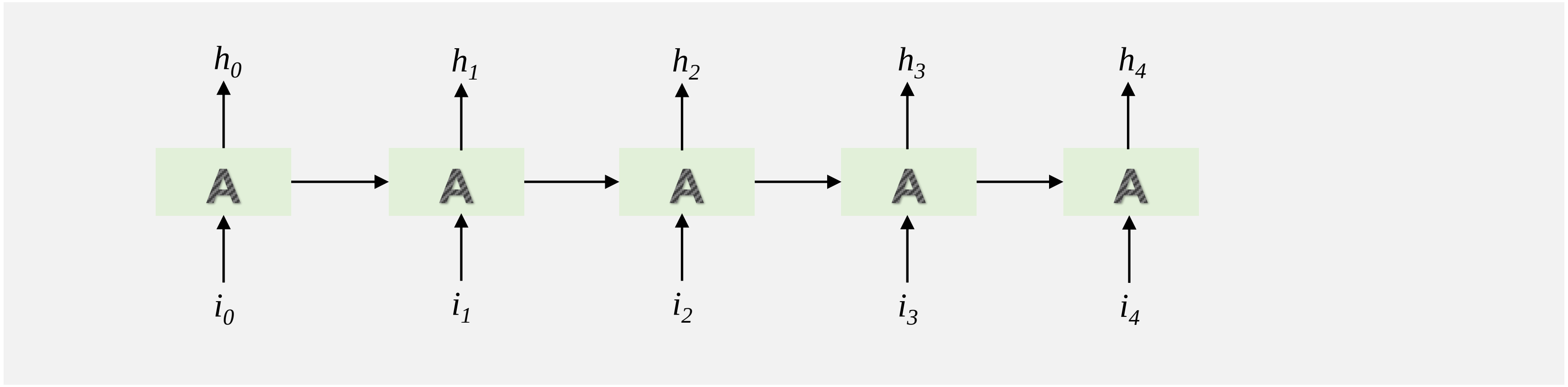


  - The result of the MFCC analysis is a stream of p-dimensional vectors. These vectors build the basis to learn phonemes from which the words and texts are derived. There are two approaches to learn phonemes:
    a) Model the phonemes with a Hidden Markov Model (HMM) based on quantized vector data and model the temporal transitions. We can use a k-means algorithm to divide the p-dimensional vectors in to a set of k states
    b) Model a neural network and learn phonemes (typically requires so-called recurrent networks which can keep a current state and feed that into the next iteration of the network run)

- The Hidden Markov Model (HMM) builds a network of states (quantized MFCC data) with probabilities of transitioning from one state to the other. Each phoneme is represented by its own HMM and a softmax across the phonetic units determines the recognized phoneme at a point in time. Building HMMs requires more "human intervention" or experience but lead to very efficient recognizers



- In contrast, a recurrent neural network does require less "domain" knowledge. The term recurrent means that the network has a notion for the current state which is fed back into the network with each time step. You can consider recurrent network like a cascade of (the same) networks where some output of a network instance becomes the input of the next instance:



- Once we recognized phonemes, there are two paths to continue:
  - recognition of words based on the stream of phonemes
  - retrieval in the phoneme stream

- **Recognition of words** is similar to phonemes (HMM, neuronal networks). However, only “known” words can be recognized, and it is often difficult to distinguish two subsequent words in the streams (no breaks between two words in spoken language). At the end, we get a text stream and can use any of the text retrieval methods to search through spoken text.

- **Retrieval in the stream of phonemes**
  – Schäuble used N-grams to describe the phoneme stream (rather than words), and mapped queries (keywords) into the phoneme space (spoke queries go through the same recognition approach; written text is mapped to phonemes based on dictionaries). From his perspective, retrieval in the phoneme stream is better suited than full word recognition:
    - Current word recognizer only detect a limited number of known words. This is not sufficient for good retrieval quality. Word recognition is more expensive than just phoneme recognition.
    - The German language contains composite words and a high degree of strong flexion. The number of potential words to recognize almost explodes. In English, the approach is much better suited.
    - (new) Names and brands are difficult to learn

  – N-gram retrieval allows for partial matches where a word recognizer struggles to obtain the correct word. If enough sequences of phonemes (N-grams) match, the spoken text still can be found. Names and brands are not a challenge any more.

5.5 Literature and Links

- P. Mermelstein (1976), "Distance measures for speech recognition, psychological and instrumental," in Pattern Recognition and Artificial Intelligence, C. H. Chen, Ed., pp. 374–388. Academic, New York.

- Frameworks and Libraries
  - **Librosa** (http://librosa.github.io/librosa/) is a Python library for advances audi and music analysis. It provides base algorithms to create music retrieval systems.

- Interesting courses at other universities
  - Music Information Retrieval, Vienna University of Technology, Austria, http://www.ifs.tuwien.ac.at/mir/
  - Music Information Retrieval, New York University, USA, http://www.nyu.edu/classes/bello/MIR.html
  - Music Signal Processing, Columbia University, USA https://www.ee.columbia.edu/~dpwe/e4896/index.html