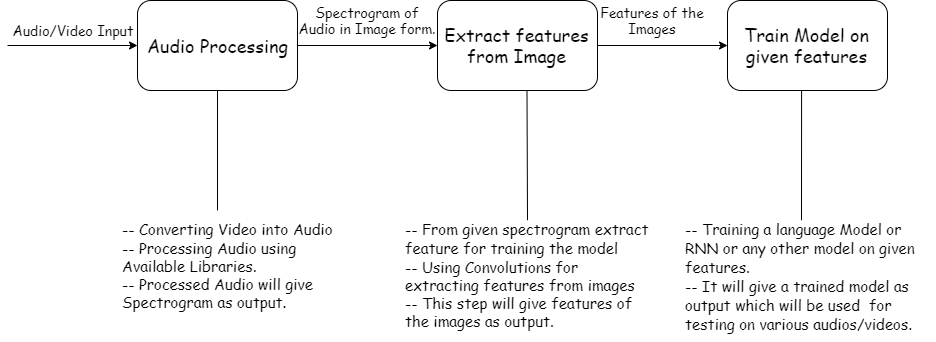
Subtitle Generator Prototype



Abstract:

This software takes the video or audio as input and generates subtitles or transcript respectively for the same. First the video/ audio file is processed using audio processing tools. The output of this function is spectrogram in form of Image. Second the processed audio is used to extract features using convolutions on the given input image.

This step will give features of the respective images. Now these extracted images will be used to train a model such as language model or RNN or any other combination of model for end-to-end speech to text recognition. This step will give trained model as an output which will be used for further testing on various other audios and videos.

After testing the model the given model will be used to give the transcript / subtitles for the given audio which will be synchronized with the given audio and converted into .txt for transcript and .srt file for subtitle.

Dataset:

We used [Tatoeba](https://tatoeba.org/eng/) dataset for prototype model building.

Tatoeba is a large database of sentences, translations, and spoken audio for use in language learning.

This dataset contains 1,265,664 sentences in English with labels of average length 3 seconds recording.

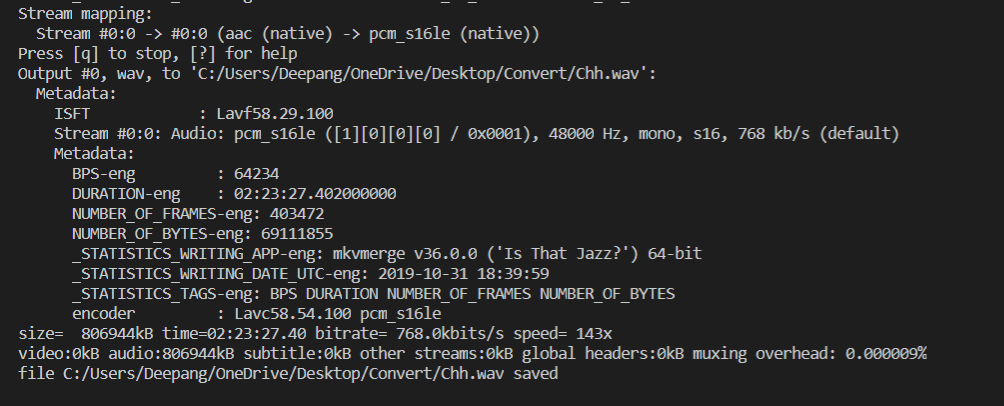
The total size of this dataset is 3.8 GB.

Field Structure for labels:

* Sentence id [tab] Lang [tab] Text

Video to Audio Processing:

For video to audio conversion we used [FFMPEG](https://www.ffmpeg.org/) library with python sub process command to achieve good quality audio file in less amount of time from videos. The output format is .wav file.



Now for the given video file of size 1.08 GB and length of 150 minutes, it takes FFMPEG about 1 min to generate its wav file which is of size 800 MB. This audio will then be used for audio processing.

Audio Processing:

The audio signal is a three-dimensional signal in which three axes represent time, amplitude and frequency.

We performed audio processing on the Tatoeba dataset using the [librosa](https://librosa.github.io/librosa/) python library which gives extensive features for working with audio.

Wave Plot of an Audio: [Time-domain]

Waveplots let us know the loudness of the audio at a given time.

***librosa.display.waveplot***

Plot the amplitude envelope of a waveform.

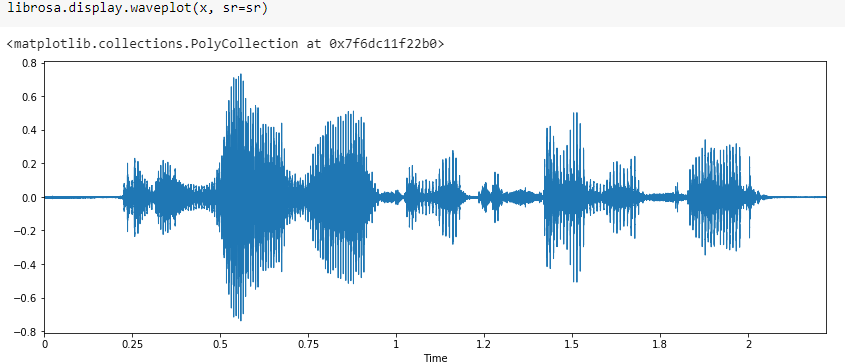
Parameters:

Y :

Audio time series (mono or stereo)

sr :

Sampling rate of y



But this is just a two dimensional representation.

Time-domain analysis completely ignores the frequency component whereas frequency domain analysis pays no attention to the time component.

We can get the time-dependent frequencies with the help of a spectrogram.

Another mathematical representation of sound is the Fourier Transform.

Fourier Transform is a function that gets a signal in the time domain as input, and outputs its decomposition into frequencies.

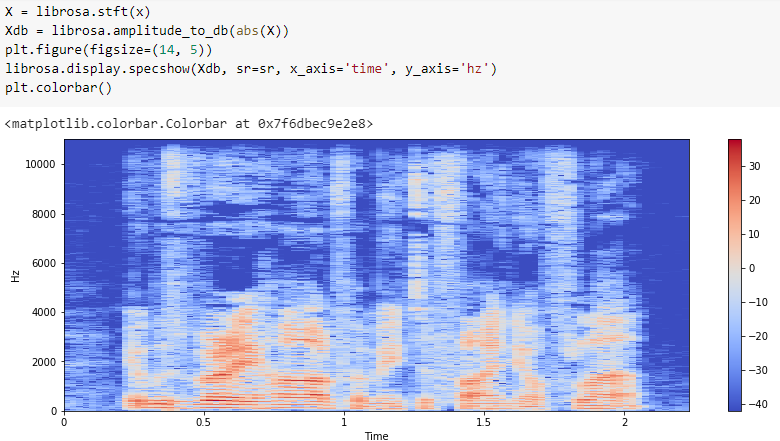
This technique is used in construction of Spectrograms.

Spectrogram of an Audio:

It’s a 2D plot between time and frequency where each point in the plot represents the amplitude of a particular frequency at a particular time in terms of intensity of color.

Basically spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time.

The spectrogram contains both, time and frequency related features that are used for classifying the audio signals.



The above figure shows how we can plot a spectrogram using the librosa library.

.*stft* converts data into short term Fourier transform. STFT converts signal such that we can know the amplitude of given frequency at a given time. Using STFT we can determine the amplitude of various frequencies playing at a given time of an audio signal. *.specshow* is used to display spectrogram.

We plotted different spectrogram to analyze which spectrogram can be used for feature extraction.

1. Log-Spectrogram –

It is a spectrogram which is having log scale of frequency as its y-axis [log scale of amplitudes.] and time as its x-axis.

The logarithmic spectrum, is a much more accessible representation.

It is not only more visual, but importantly, the logarithm approximates roughly the sensitivity of the ear, such that logarithmic spectra can be used to assess auditory importance of spectral features.

The logarithmic spectrum visualizes spectral content such that the magnitude of values is approximately uniform throughout the spectrum.



The above figure shows the log scaled spectrogram of an audio signal.

The only exception is zeros and other very small values in the magnitude spectrum, which give negative infinities or arbitrarily large negative values in the log spectrum.

1. Mel-Spectrogram –

Mel Scale -

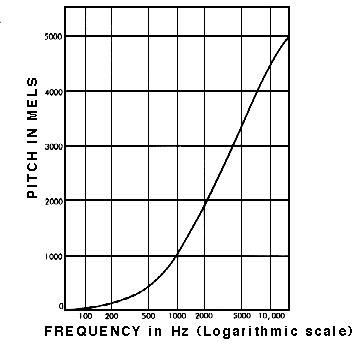
It is based on the Mel Scale, mathematically speaking, is the result of some non-linear transformation of the frequency scale.

Mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another.

The reference point between this scale and normal frequency measurement is defined by equating a 1000 Hz tone, 40 dB above the listener's threshold, with a pitch of 1000 mels.

Above about 500 Hz, increasingly large intervals are judged by listeners to produce equal pitch increments.

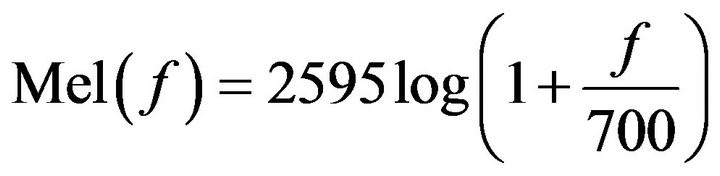
Which basically means that it divides any range above 1 kHz into equal bins which are audible to the human ears.



As a result, four octaves on the hertz scale above 500 Hz are judged to comprise about two octaves on the Mel scale.

Divided not by distance on the frequency dimension, but distance as it is heard by the human ear.

A frequency measured in Hertz (f) can be converted to the Mel scale using the following formula:



Mel Spectrogram, is, rather surprisingly, a Spectrogram with the Mel Scale as its y axis.

***librosa.filters.mel***

Create a Filterbank matrix to combine FFT bins into Mel-frequency bins.

To directly plot the Mel Scaled Frequency on any domain we use the direct function provided by librosa.



One way of evaluating periodic structures in a signal on different scales is to use the Fourier transform.

Specifically, we can take the discrete Fourier transform (DFT) or the discrete cosine transform (DCT) of the log-spectrum, to obtain a representation known as the cepstrum.

Cepstrum –

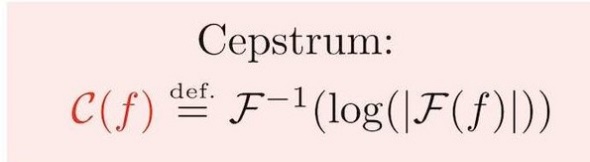
The cepstrum can be seen as information about the rate of change in the different spectrum bands.

Power spectra of windows of speech signals contain information about the most important features of speech signals like the identity of vowels.

Cepstrum pitch determination is particularly effective because the effects of the vocal excitation (pitch) and vocal tract (formants) are additive in the logarithm of the power spectrum and thus clearly separate.

A second useful piece of information in the cepstrum is the harmonic structure of the log-spectrum.

The resulting spectrum is neither in the frequency domain nor in the time domain and hence B.P. Bogert decided to call it the quefrency domain.



Mel-Frequency Cepstral Coefficients (MFCCs) –

To further improve on the cepstral representation, we can include more information about auditory perception into the model.

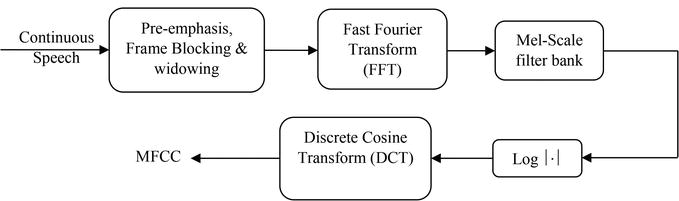
Specifically, by introducing information about human perception, we focus the model on that part of the information which human listeners would find important.

The mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10–20) which concisely describe the overall shape of a spectral envelope.

The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum.

MFCC computation is a replication of the human hearing system intending to artificially implement the ear’s working principle with the assumption that the human ear is a reliable speaker recognizer. Because of this reason MFCC are used as features for speech recognition.

MFCCs are commonly derived as follows:



Take the Fourier transform of (a windowed excerpt of) a signal.

Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows.

Take the logs of the powers at each of the mel frequencies.

Take the discrete cosine transform of the list of mel log powers, as if it were a signal.

The MFCCs are the amplitudes of the resulting spectrum.

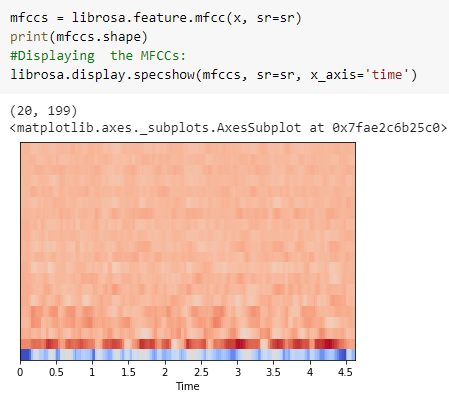
* ***librosa.feature.mfcc*** :

Mel-frequency cepstral coefficients (MFCCs)

y : audio time series

sr: sampling rate of y

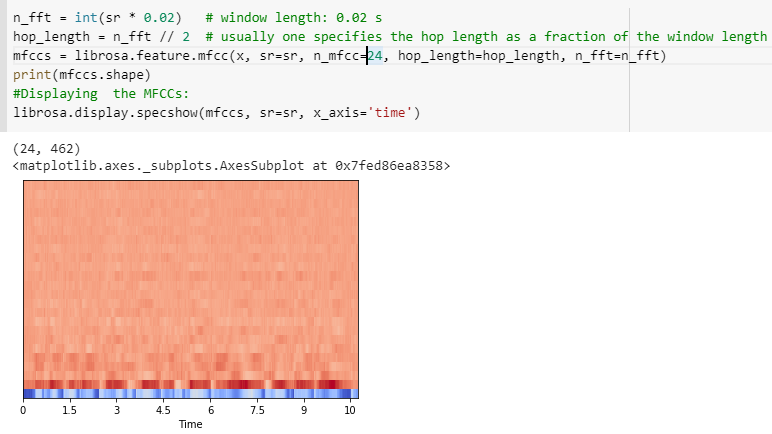
n\_mfcc: number of MFCCs to return



The first value represents the number of mfccs calculated and another value represents a number of frames available.

Here there are 20 MFCC features for each audio sample with Sample rate 22050Hz Frequency and of average length 3 sec.

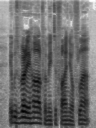
We can tweak number of frames by changing the hop length and changing the window length respectively.



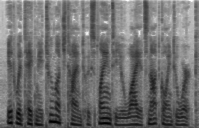
Hop Length means how many samples are to be skipped to start the new window of audio.

Every window is over lapping such that covering enough information for our feature extraction from the particular audio clip.

Some saved Images of audios from Tatoeba dataset:



“It was very hard for me to find your flat.”



“It would take me too much time to explain to you why it's not going to work.”

We used Google Colab as our working environment for audio processing which supports high end calculations using the support of GPU.

Now that we have features extracted from the audio sample, we select a model for training.

Model Selection

Libraries Used:

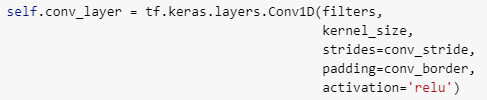
1. [TensorFlow](https://www.tensorflow.org/): The core open source library which helps develop and train ML models.
2. [Keras](https://keras.io/): It is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) [neural-network](https://en.wikipedia.org/wiki/Artificial_neural_network) library written in [Python](https://en.wikipedia.org/wiki/Python_(programming_language)). It is capable of running on top of [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow" \o "TensorFlow), designed to enable fast experimentation with deep neural networks.

MODEL 1:

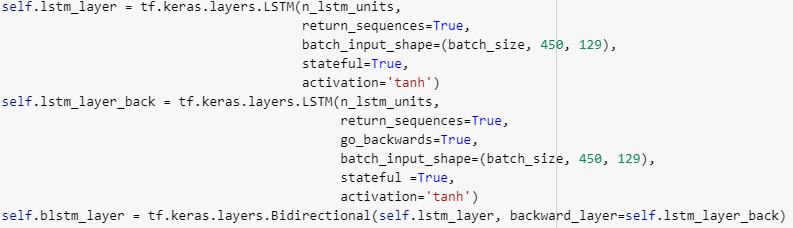
Model Info:

This model consists of a 1D convolutional layer followed by a bidirectional LSTM followed by a fully connected layer applied at each time step.

1. 1D Convolutional layer: This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. This layer is used to extract the features from spectrogram and provide them further to lstm.



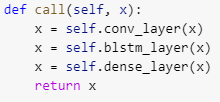
1. Bidirectional LSTM: This layer is the combination of forward and backward LSTM which is responsible for actual training and prediction. Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feed forward neural networks, LSTM has feedback connections.



1. Dense Layer: This is the simple dense neural network layer which combines the output of LSTM and processes it. Dense implements the operation: output = activation (dot (input, kernel) + bias) where activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer.



Finally, we combine all the three layers and the model is ready:

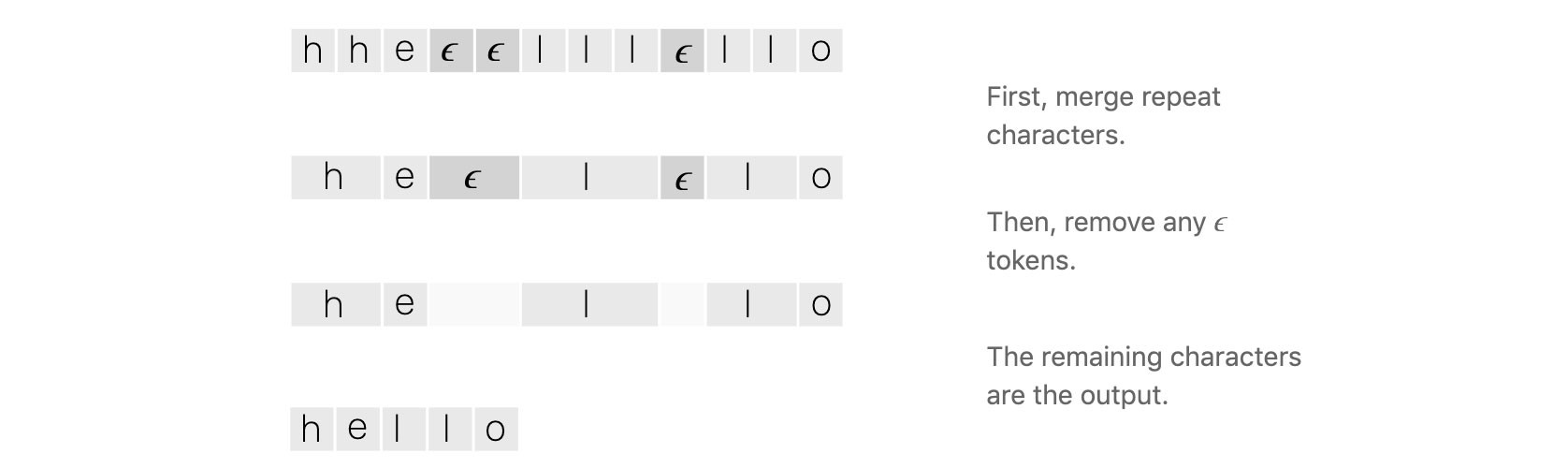


OPTIMIZER: Adam

Adaptive Moment Estimation (Adam) is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients, similar to momentum.

CTC [Connectionist Temporal Classification]:

The CTC Compression Rule:

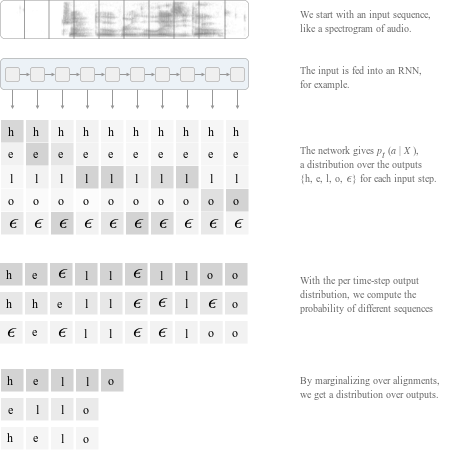


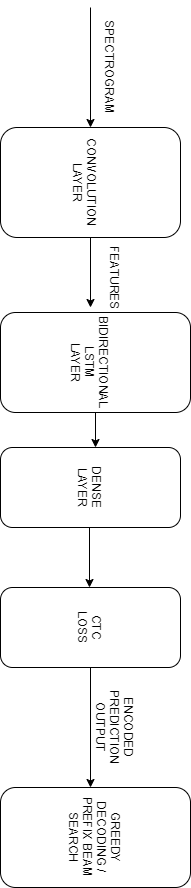
The alignment-free concept in CTC actually pushes the complexity from mapping to searching.

CTC Loss:

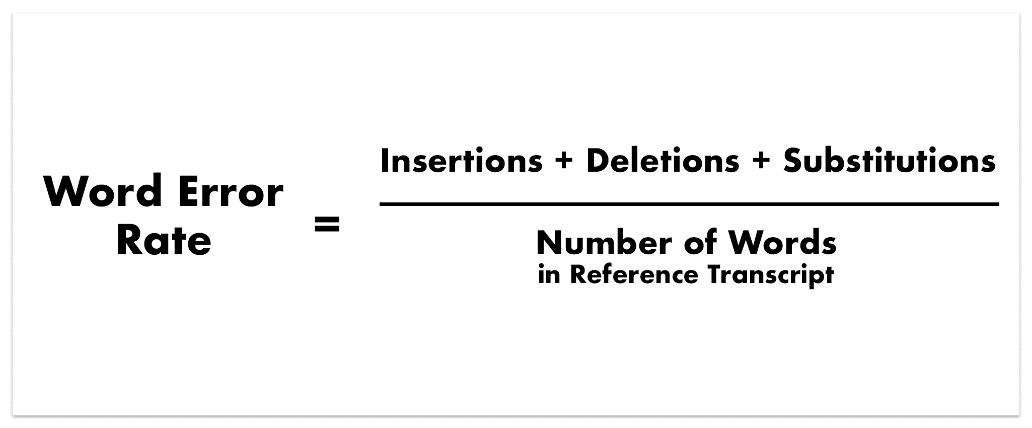
We need to sum over all paths that generate the same word sequence. For example, to find the probability for the word “hello”, we sum over all the corresponding paths like “heεllεlloo”, “hhellεεlεo”, “εeεllεεloo” etc.

To find the most likely word sequence, we search for different path combinations. We need to sum over all paths that generate the same word sequence.





Error Rate**:**

****

The WER is derived from the [Levenshtein distance](https://en.wikipedia.org/wiki/Levenshtein_distance), working at the word level instead of the [phoneme](https://en.wikipedia.org/wiki/Phoneme) level.

The WER is a valuable tool for comparing different systems as well as for evaluating improvements within one system.

This kind of measurement, however, provides no details on the nature of translation errors and further work is therefore required to identify the main source of error and to focus any research effort.

Issues of WER:

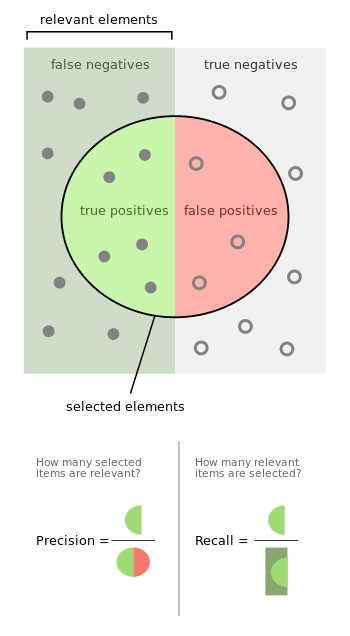
The fundamental problem with WER is that every word is worth the same number of points.

Along with ignoring the importance of words, WER is also a brutally harsh judge: it gives no partial credit. Even if a miss-transcribed word is just one character off, WER treats it the same as a complete, nonsensical whiff.

Another issue with WER is its total disregard for speaker labels and punctuation.

These may or may not be important, depending on your use-case—but it is obviously a major simplification.

F1 Score:



Precision: basically tells us that out of the results classified as positive by our model, how many were actually positive.

Recall: tells us how many true positives (points labelled as positive) were recalled or found by our model.

F1-score is a metric which takes into account both, precision and recall as we can’t always evaluate both and then take the higher one for our model.

It is the harmonic mean of precision and recall.