TIME-SERIES PATTERN MATHCING USING DYNAMIC TIME WARPING ALGORITHM

CHINMAY BAKE

In [22]:
```
import numpy as np
from scipy.spatial.distance import euclidean
from fastdtw import fastdtw
import pandas as pd
import matplotlib.pyplot as plt
from python_speech_features import mfcc
```

INTRODUCTION & ORIGNAL RESEARCH BACKGROUND

One of most straightforward ways to match time series patterns is by using standard clustering algorithms on a flattened time series. The problem with them though; they are invariant to time lags and essentially ignore the time dimension associated with each data point.

Dynamic Time Warping is a technique to measure similarity between two temporal sequences that do not align exactly in time, speed, or length.

We would be using the exact same approach, although the background to this research project is a little different. This orginal proof-of-concept was designed specifically for audio temporal/time-series signals (it was my engineering capstone project!) and has its main application in Speech Recognition. Thus, the Mel Frequency Cepstrum part, although important from a Speech Recognition perspective, is actually optional for simply comparing two time-series signals.

MOTIVATION FOR TEMPORAL PATTERN MATCHING

Let us assume that we have time-series historical sales data for a diverse range of products. One of the most important things while building time-series predictive models is to generalise their behaviour. For eg: If I would like to build a function to transform a time series into a stationary time series, then I should be sure of what type transformation is required for the given task. Bottomline is, similar time series patterns could be clustered and could be used to train models representing that cluster. This could help in achieveing very accurate predictive models.

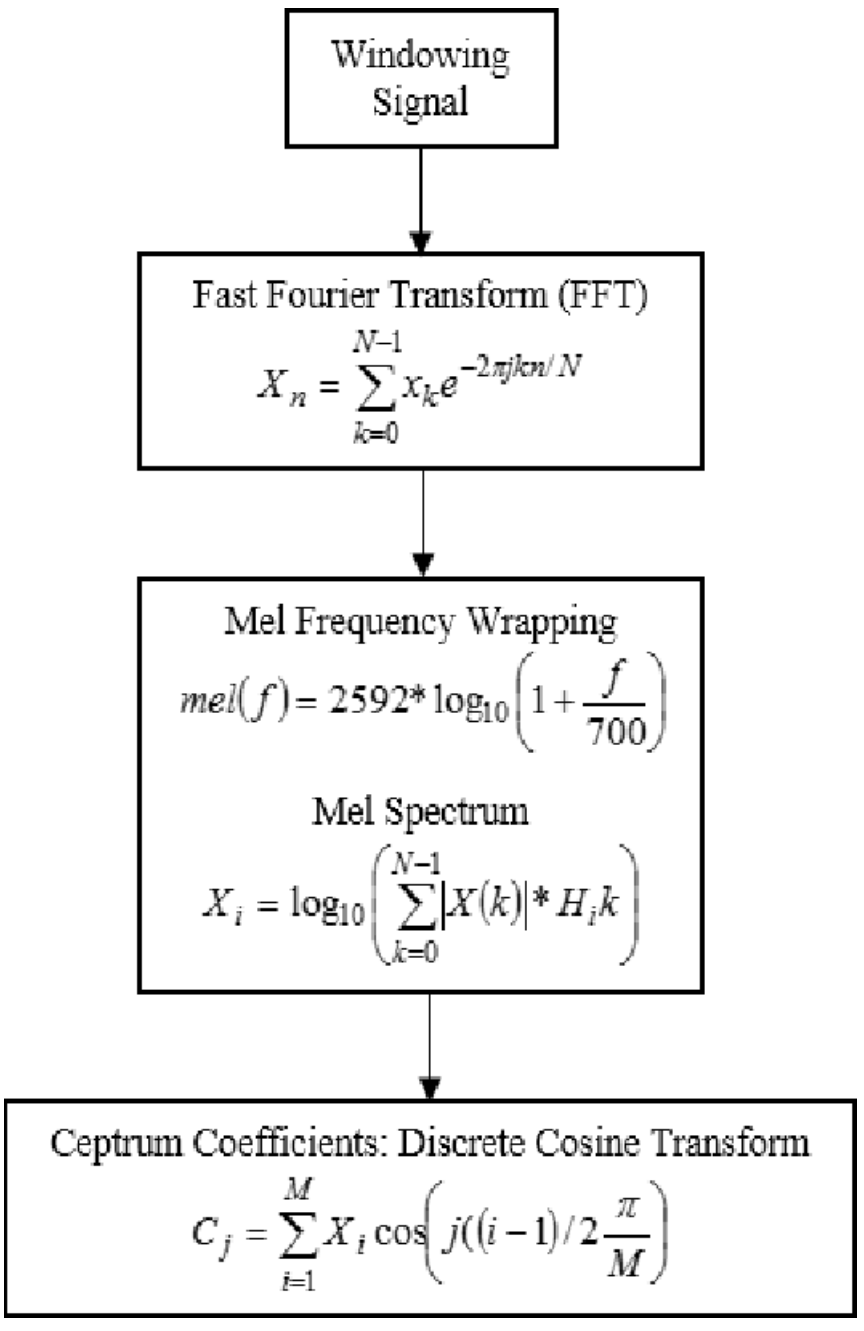
In [1]:
```
def simulate(amplitude_1,frequency_1,amplitude_2,frequency_2,samples):

    x = np.arange(samples)

    signal_1 = amplitude_1 * np.sin(2 * np.pi * frequency_1 * x/44100)
    signal_2 = amplitude_2 * np.sin(2 * np.pi * frequency_2 * x/44100)

    return (signal_1, signal_2)
```

For prototyping this proof-of-concept, we use simulated signals instead of real ones. The simulation includes signals with varying amplitudes and frequencies. The first step in feature engineering is to compute MFCC of the data. MFCC takes into account human perception for sensitivity at appropriate frequencies by converting the conventional frequency to Mel Scale, and are thus suitable for speech recognition tasks quite well. Below are the steps on how to compute MFCC -



Source:Semanticscholar.org

In [79]:
```
def similarity(series1,series2):

    mfcc1 = mfcc(series1)
    mfcc2 = mfcc(series2)

    distance, path = fastdtw(mfcc1, mfcc2, dist=euclidean)

    return distance

def similar_series(z):

    closest = []

    for i in range(10,101,10):
        value = z[i-10:i].sort_values(by = 0, ascending = True)[1:2].index[0]
        closest.append(value)

    for k in range(0,10,1):

        if closest[k] < 10:
            print(closest[k])
        else:
            print(str(closest[k])[0] + '-->' + str(closest[k])[1])
```

DYNAMIC TIME WARPING

We utilise this method for matching the feature engineered mfcc data points. it is widely used in comparing and evaluating temporal data. It works by first evaluating the least cost distance to reach every square in the grid and then tracing back the path which corresponds to the overall least cost distance.

Below is the algorithm in a nutshell:

$$DTW(x, y) = \min_{\pi} \sqrt{\sum_{(i,j) \in \pi} d(x_i, y_j)^2}$$

where $\pi = [\pi_0, \ldots, \pi_K]$ is a path that satisfies the following properties:

- it is a list of index pairs $\pi_k = (i_k, j_k)$ with $0 \leq i_k < n$ and $0 \leq j_k < m$
- $\pi_0 = (0, 0)$ and $\pi_K = (n-1, m-1)$
- for all $k > 0$, $\pi_k = (i_k, j_k)$ is related to $\pi_{k-1} = (i_{k-1}, j_{k-1})$ as follows:
  - $i_{k-1} \leq i_k \leq i_{k-1} + 1$
  - $j_{k-1} \leq j_k \leq j_{k-1} + 1$

Source: tslearn documentation

PROOF OF CONCEPT - DYNAMIC TIME WARPING IN CONJUCTION WITH MFCC

- *Compute MFCC of the series which has to compared along with the reference series*
- *Evaluate the distance between them using Dynamic Time Warping*
- *Loop over the array of time-series data points to be compared*
- *Record the distances*
- *Minimum recorded distances for respective series have most likelihood of being similar*

In [81]:
```
def check_series():

    no_of_series = 10
    freq = []
    amplitude = []
    sim = []

    for i in range(0,no_of_series,1):
        freq.append(np.random.randint(5,10))
        amplitude.append(np.random.randint(20,80))

    for i in range(0,no_of_series,1):
        for j in range(0,no_of_series,1):
            x = simulate(freq[i],amplitude[i],freq[j],amplitude[j],100)
            sim.append(similarity(x[0],x[1]))

    return similar_series(pd.DataFrame(sim))

check_series()
```

```
1
1-->3
2-->6
3-->1
4-->3
5-->6
6-->5
7-->9
8-->6
9-->7
```

REFERENCES

- Baum, L. E., Petrie, T., Soules, G. & Weiss, N. (1970), ‘A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains’, Ann.Math. Sastist.
- Blum, T. L., Keislar, D. F., Wheaton, J. A. & Wold, E. H. (1999), Method and article of manufacture for content-based analysis, storage, retrieval, and segmentation of audio information, U.S Patent 5, 918, 223
- Foote, J. T. (1997), Content-based retrieval of music and audio, in ‘SPIE’, pp. 138 – 147
- Logan, B. T. & Chu, S. (2000), Music summarization using key phrases, in ‘Proceedings IEEE Internal Conference on Acoustics, Speech, and Signal Processing