Speech Recognition using Dynamic Time Warping

Spoken and Written Language Processing

Data Science and Engineering
Polytechnic University of Catalonia

Claudia Len Manero - 53869554
Alina Castell Blasco - 49188689S

May 2024

1 Introduction

The goal of this assignment is to perform digit recognition using Dynamic Time Warping. To do so we will develop different tasks:

- WER function modification to evaluate the SER, which measures the performance of the DTW algorithm for the speaker identification.
- Evaluation of the WER of different variants of the cepstral coefficients to see how it affects the WER.
- Analysis of the influence of the cepstral normalization steps on the recognition accuracy.
- Extend the mfcc parameters to reduce the WER.
- Complete the DTW algorithm to compute the alignment with backtracking.
- And finally, study variations of the DTW and compare them to the baseline.

2 Speaker Error Rate

The task on this section was to modify the WER function to evaluate the speaker error rate (SER) and calculating the probability of detecting the correct speaker of the free-spoken-digit-dataset using the DTW distance to the rest of the recordings.

The probability of making an error in recognising a speaker from recordings of the same speaker is 4.4%, whereas the probability of making that error with recordings from other speakers is of 100.0%.

Having said that, here is the code used for calculating those probabilities:

```
# Speaker Error Rate (Accuracy)
# Modified the 'text' for 'speaker'
def ser(test_dataset, ref_dataset=None, same_spk=False):
    # Compute mfcc
    test_mfcc = get_mfcc(test_dataset)
    if ref_dataset is None:
        ref_dataset = test_dataset
        ref_mfcc = test_mfcc
    else:
        ref_mfcc = get_mfcc(ref_dataset)
    err = 0
    for i, test in enumerate(test_dataset):
        mincost = np.inf
        minref = None
        for j, ref in enumerate(ref_dataset):
            if not same_spk and test['speaker'] == ref['speaker']:
                # Do not compare with refrence recordings of the same speaker
                continue
            if test['wav'] != ref['wav']:
                distance = dtw(test_mfcc[i], ref_mfcc[j])
                if distance < mincost:
                    mincost = distance
                    minref = ref
        if test['speaker'] != minref['speaker']:
            err += 1
    ser = 100*err/len(test_dataset)
    return ser
```

3 Compare the WER with respect to the number of cepstral coefficients

In this section we have modified the number of cepstral coefficients to see how this number affects the WER of the speech recogniser. Our goal is to find the number that minimizes the WER, that is, it obtains the smallest WER. The following table shows the results:

	WER for different Cepstral Coefficients														
	5	6	7	8	**9**	10	11	12	13	14	15	16	17	18	19
=	16.2	12.5	8.8	8.1	**7.5**	6.2	5.6	6.9	7.5	6.9	6.2	6.2	6.2	6.9	7.5
≠	27.5	27.6	27.8	26.3	**24.3**	26.0	27.5	28.9	29.4	30.7	31.4	34.0	34.4	35.5	37.7

Table 1: WER for different Cepstral Coefficients and same (=) or other (≠) speaker recordings.

First, by analysing the WER values obtained on recordings from the same speaker, we can observe that as the number of cepstral coefficients increases from 5 to 11, the WER decreases significantly from 16.2% to 7.5%. This reduction in WER suggests that increasing the number of cepstral coefficients improves recognition accuracy by capturing more detailed information about the audio signal. In particular, between 9 and 11 cepstral coefficients, there is a notable improvement in recognition accuracy, with the WER dropping from 7.5% to 5.6%. However, after 11 cepstral coefficients, the WER fluctuates slightly, ranging from 6.2% to 7.5%. This suggests that adding more cepstral coefficients beyond a certain point does not significantly improve recognition accuracy and may even lead to overfitting or increased computational complexity without substantial benefits in accuracy.

Secondly, we can observe that the WER of the recordings from other speakers yield much higher values, due to the fact that different speakers have unique speech characteristics, such as pitch, accent, and pronunciation. This variability worsens the performance of the recognizer. As for the fluctuations of the recognition accuracy, a similar behaviour to the same speaker recordings can be observed. Increasing the performance up to a point where the cepstral coefficient is 9, and decreasing when this value starts to increase from 11.

To summarize, in both types of recordings we can see that the lowest WER values are achieved when the Cepstral coefficients are in the range from 9 to 11; however, the lowest of the both recordings does not pair. Therefore, we decided to compute a mean of both scores for these three coefficients, the results obtained were [9, 10, 11] = [15.9, 16.1, 16.55]. Thus, based on these results, the best cepstral coefficient value is 9 (in bold), which achieves the lowest WER for other speaker recordings of 54.3% and the second lowest for same speaker of 7.5%. The cepstral coefficient value of 9 will be used from now on to execute the rest of the sections.

4 Analyse the influence of each of the cepstral normalisation steps with and without liftering

In order to perform the analysis presented in this section, we will compute the Word Error Rate (WER) with all the combinations possible that include mean normalisation step, variance normalisation step and liftering. The values are 0 mean, 1 variance and 22 liftering; this last one is used to give more weight to the central frequencies.

WER with and without normalisation and liftering				
Mean	Variance	Liftering	Same Speaker	Other speakers
False	False	0	6.9%	54.6%
False	False	22	2.5%	41.4%
False	**True**	**0**	**7.5%**	**24.3%**
False	True	22	7.5%	38.2%
True	False	0	9.4%	39.8%
True	False	22	5.6%	28.2%
True	**True**	**0**	**7.5%**	**24.3%**
True	True	22	7.5%	38.2%

Table 2: All possible combinations of normalisation and liftering of 22.

On the one hand, the WER values obtained for the same speaker recognition seem to be, overall, lower when no variance (CMVN) is applied. Covering each case in particular we observe that when both mean (CMS) and variance normalization are turned off (False, False) or only mean normalization is turned on (True, False), we observe a WER of 6.9% without liftering, 2.5% with liftering, 9.4% without liftering and 5.6% with liftering, respectively. This means that when neither of the normalisation steps is applied, the liftering has a huge positive effect when appliede, it obtains a better WER. When only variance normalization is turned on (False, True) or both mean and variance normalization are turned on (True, True), we see a consistent WER of 7.5% regardless of liftering. The best result obtainted for the same speaker recognition WER is 2.5% achieved by using only liftering with no CMS or CMVN.

On the other hand, for other speakers, the WER is significantly higher when neither mean nor variance is applied regardless of the liftering, yielding results 54.6% and 41.4%. for different combinations of normalization and liftering. In the case when only mean normalization is turned on (True, False) a better WER value (28.2%) is obtained if the liftering is True as well. Whereas if only variance normalization is turned on (False, True) the better WER value (24.3%) is obtained without liftering. When both mean and variance normalization are turned on the better WER value is 24.3%, which equals the best WER value of other speakers recognition overall.

To conclude, the best combination of WER values (mean of same speaker and other speakers) is obtained when both variance normalisation and no liftering are applied, regardless of the mean normalisation step. The best WER values for same speaker and other speaker are 7.5% and 24.3%, respectively, (highlighted in bold).

5 Extend the MFCC parameters with the first-order derivatives

In this section, we will see how the WER values are affected by an addition of the first derivative to all the previous configurations. WER values for same speaker tend to be lower because the system thinks the numbers said in the audios match when in reality they don't, the only match is on the voice of the speaker. The first derivative allows the recognition system to be more robust to certain WER from *same speaker* audios; that is, when the WER value is low because the system considers that the audio is describing the target number correctly, when in fact the number does not match but the speaker does.

The configurations use 9 cepstral coefficients. See the results on the next table:

WER with first derivative				
Mean	Variance	Liftering	Same Speaker	Other speakers
False	False	0	7.5%	54.6%
False	False	22	3.1%	41.8%
False	**True**	**0**	**7.5%**	**24.0%**
False	**True**	**22**	**6.9%**	**37.8%**
True	False	0	9.4%	39.6%
True	False	22	6.2%	28.0%
True	**True**	**0**	**7.5%**	**24.0%**
True	**True**	**22**	**6.9%**	**37.8%**

Table 3: All possible combinations of normalisation and liftering of 22, and the first derivative.

Overall, we observe a tendency to improve the WER values when applying the first derivative. Some configurations have had considerable improvement and some less, however, considering how short the audios analysed are, and the similar tone all different speakers show when pronouncing the same one-word sentence, the improvements are great.

For the WER values of the same speaker, the only improvement is seen on the configurations when CMVN and lifter are true (True, 22), regardless of the CMN; the WER values decreases from a 7.5% without first derivative to a 6.9% with derivative. For the WER values of other speakers the configurations that most improve are these ones as well, from a value of 38.2% without derivative to a 37.8% with derivative. Both are highlighted in bold.

Still, the best overall values obtained considering both types of recordings are the same previous configurations (CMN, CMVN, lift) = (False/True, True, 0), which show a slight improvement on the other speaker audios as well. Both are highlighted in bold. This tells us that the applications of mean and variance normalisation steps tends to yield better results.

The code modifications done for implementing the first derivative are on the *get_mfcc* function, the rest of the functions were slightly modified to be able to compute all configurations at once. The code snippet is shown here:

```
# Applying first derivative
def get_mfcc2(dataset,n_mfcc=9,lifter=0, cms=True, cmvn=True,**kwargs):
    mfccs = []
    for sample in dataset:
        sfr, y = scipy.io.wavfile.read(sample['wav'])
        y = y/32768
        S = mfsc(y, sfr, **kwargs)
        # Compute the mel spectrogram
        M = mfsc2mfcc(S, n_mfcc=n_mfcc, lifter = lifter, cms = cms, cmvn = cmvn)
        # Move the temporal dimension to the first index
        M = M.T
        try:
            derivative1 = librosa.feature.delta(M)
        except Exception as e:
            print(f"Error computing first-order delta coefficients: {e}")
            continue
        Mderivada1 = np.hstack((M, derivative1))
        mfccs.append(Mderivada1.astype(np.float32))
    return mfccs
```

```
20
21 # Word Error Rate (Accuracy)
22 def wer(test_dataset,n_mfcc=9,lifter=0, cms=True, cmvn=True, ref_dataset=None,
       same_spk=False):
23     # Compute mfcc
24     test_mfcc = get_mfcc2(test_dataset, n_mfcc=n_mfcc, lifter = lifter, cms = cms,
       cmvn = cmvn)
25     if ref_dataset is None:
26         ref_dataset = test_dataset
27         ref_mfcc = test_mfcc
28     else:
29         ref_mfcc = get_mfcc2(ref_dataset, n_mfcc=n_mfcc, lifter = lifter, cms = cms,
       cmvn = cmvn)
30     (...)
31     return wer
32 print('COMPUTING FIRST DERIVATIVE WITH ALL CONFIGURATIONS ')
33 print(f'CMS - CMVN - LIFTER')
34
35 # Define a function to analyze the influence of cepstral normalization and liftering
       on recognition accuracy
36 def analyze_combis(normalize_mean, normalize_variance, apply_lifter):
37     # Define parameters for MFCC computation
38     mfcc_params = {
39         'lifter': 22 if apply_lifter else 0,  # Whether to apply liftering,
40         'cms': normalize_mean,  # Whether to apply cepstral mean subtraction
41         'cmvn': normalize_variance  # Whether to apply cepstral mean and variance
       normalization
42     }
43     # Compute WER
44     wer_value_same = wer(free10x4x4, ref_dataset=free10x4x4, same_spk=True, n_mfcc=9,
       lifter=mfcc_params['lifter'],
45                          cms=mfcc_params['cms'], cmvn=mfcc_params['cmvn'])
46     wer_value_other = wer(commands10x100, ref_dataset=commands10x10, n_mfcc=9, lifter=
       mfcc_params['lifter'],
47                          cms=mfcc_params['cms'], cmvn=mfcc_params['cmvn'])
48     return wer_value_same, wer_value_other
49
50 combinations = combinations = [
51     [False, False, False],
52     [False, False, True],
53     [False, True, False],
54     [False, True, True],
55     [True, False, False],
56     [True, False, True],
57     [True, True, False],
58     [True, True, True]]
59 for i in range(len(combinations)):
60     res1, res2 = analyze_combis(
61         normalize_mean=combinations[i][0], normalize_variance=combinations[i][1],
       apply_lifter=combinations[i][2])
62     print(f'{combinations[i][0]} - {combinations[i][1]} - {combinations[i][2]}: {res1
       }\% & {res2}\%')
```

6 Complete the DTW algorithm to compute the alignment (backtracking)

In this last mandatory section we implement a backtracking algorithm to obtain the alignment of the DTW. To do so we used the _traceback(D) function defined in the commented code of the dtw function:

```
    def _traceback(D):
        """
        Backtracks through the accumulated cost matrix to find the optimal path.

        Args:
            D (numpy.ndarray): The accumulated cost matrix.

        Returns:
            path (numpy.ndarray): The optimal alignment path as a single array
    containing pairs of indices.
        """
        i, j = np.array(D.shape) - 2
        path = [(i, j)]
        while (i, j) != (0, 0):
            tb = np.argmin((D[i, j], D[i, j + 1], D[i + 1, j]))
            if tb == 0:
                i -= 1
                j -= 1
            elif tb == 1:
                i -= 1
            else:
                j -= 1
            path.append((i, j))
        path.reverse()
        return path
```

With this function the output of the dtw is now the distance and the alignment.

We also modified the WER function to plot the alignment of the sequences when the digit is the same and when it is different to analyze the differences.

```
    if test['text'] != minref['text']:
            err += 1
            if plot_d < 3:
                print("Plot different digit")
                plt.plot(alignment)
                plt.show()
                plot_d += 1
        else:
            if plot_s < 3:
                print("Plot same digit")
                plt.plot(alignment)
                plt.show()
                plot_s += 1
```

To analyze the plots we have to understand that each line represents a sequence and each axis represent the values of one of the sequences (x for the blue and y for the orange), meaning when one of the lines keeps the same value while the other advances it means the first one is waiting for the second one to align.

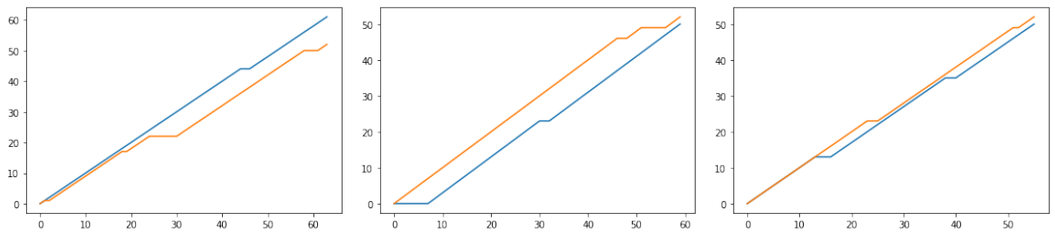

Figure 1: Alignment for same digit and speaker cases

As we can see for the cases when the digit and speaker are the same in both sequences, alignment is fairly close and sequences defer very little from each other. That is because we are using the same digit as reference and test and the same speaker, meaning we should be encountering less differences.

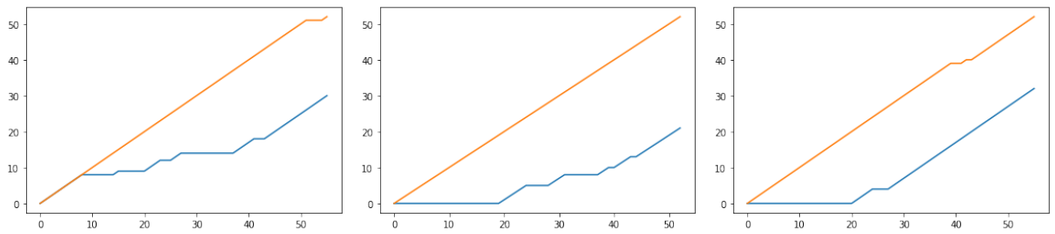

Figure 2: Alignment for different digit and same speaker cases

In the case of having different digits there is a notable difference and alignment tends to be worse, making one of the sequences wait for the other most of the algorithm, that maybe be caused by different silences in the digits making one wait for the other.

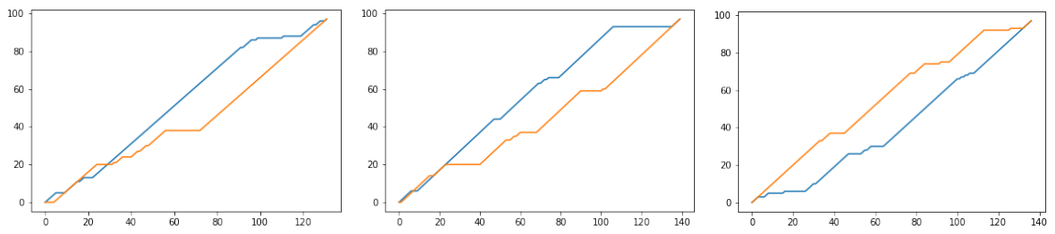

Figure 3: Alignment for same digit and different speaker cases

Analyzing the cases where the speaker was different but the digit was the same we can see the results are worse than the ones seen when the speaker was the same. This may be caused by how each speaker pronounces and makes pauses when talking.

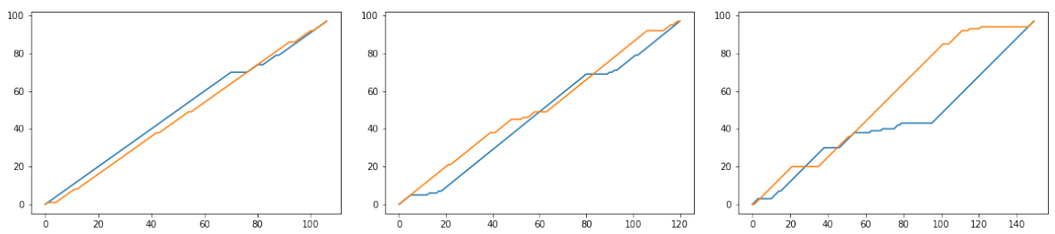

Figure 4: Alignment for different digit and speaker cases

Finally we looked at the cases where both the speaker and digit where different and we were surprised by the results. As they don't differ as much as expected from the ones where the speaker was different but the digit was the same. There is even one result which presents a better alignment than the ones given by the previous cases.

With this analysis we can conclude that what affects more is the speaker election, as when we compare samples of the same speaker there is a clear difference between using the same digit as reference and test and not doing so, which make the recognition task easier.

7 Comparative study of different variants of the DTW algorithm

Following the last section we tried penalizing bad alignments to try to improve results. To do so we introduced a penalty to the dtw function:

```
def dtw_penalty(x, y, metric='sqeuclidean'):
    r, c = len(x), len(y)
    penalty = 20
    D = np.zeros((r + 1, c + 1))
    D[0, 1:] = np.inf
    D[1:, 0] = np.inf
    # Initialize the matrix with dist(x[i], y[j])
    D[1:, 1:] = scipy.spatial.distance.cdist(x, y, metric)
    for i in range(r):
        for j in range(c):
            min_prev = min(D[i, j], D[i+1, j] + penalty , D[i, j+1] + penalty)
            D[i+1, j+1] += min_prev
    if len(x) == 1:
        path = zeros(len(y)), range(len(y))
    elif len(y) == 1:
        path = range(len(x)), zeros(len(x))
    else:
        path = _traceback(D)
    return D[-1, -1], path
```

This gives a higher dtw distance, but produces better alignments. To be able to compare it to the previous version we also plotted the alignment to check results.

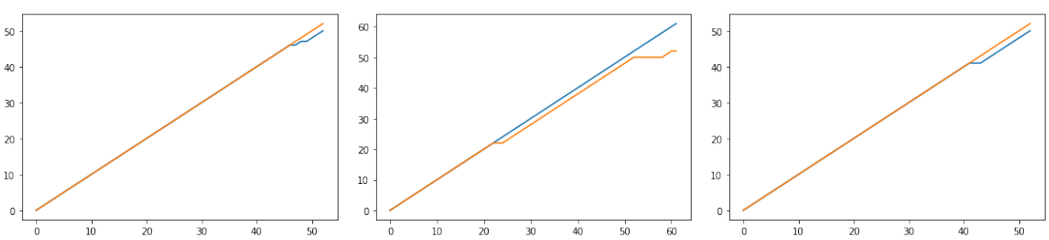

Figure 5: Alignment for same digit and speaker cases

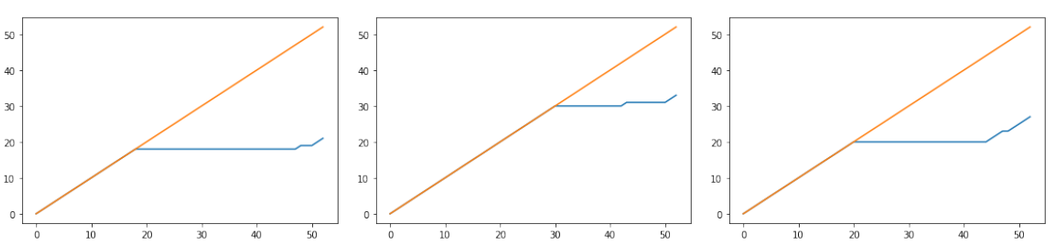

Figure 6: Alignment for same digit and speaker cases

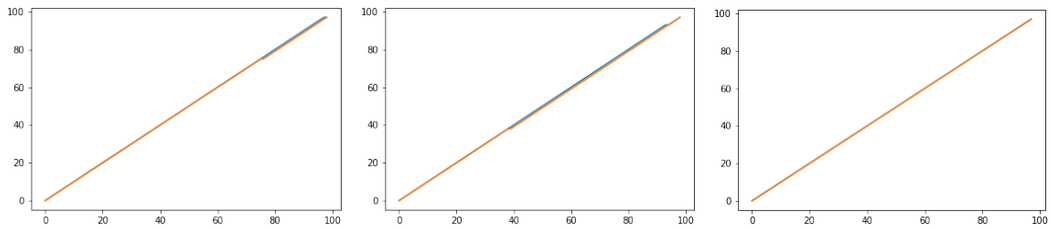

Figure 7: Alignment for same digit and speaker cases

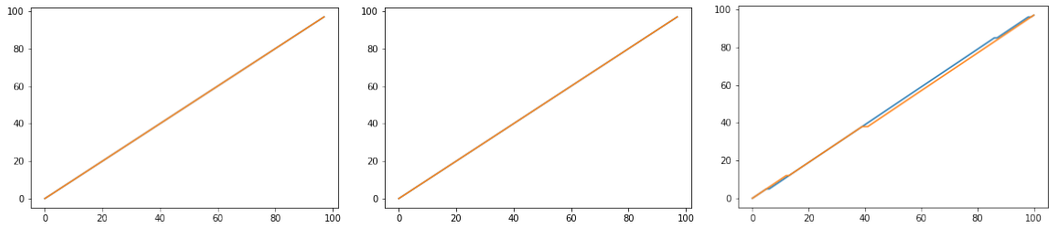

Figure 8: Alignment for same digit and speaker cases

With the new dtw we can see how alignment has improved notably for all cases, specially for the ones where we have different speakers, giving us an almost perfect alignment in all cases. Although alignment now is better, this probably has bad effects in the WER final value as we can see from the metrics, which were initially 6.9% for same speakers and 28.9% for different speakers and now is:

```
1    WER including reference recordings from the same speaker: 11.9%
2    WER including reference recordings from other speakers: 46.3%
```

This clearly tells us that although the alignment produced is better, the error is higher and it doesn't benefit the task, rather worsen it.