Speaker Identification Project

---

1. DTW Plain Implementation

```
public static double Compute(Sequence s, Sequence t)
{
   int n = s.Frames.Length;
   int m = t.Frames.Length;
   var dtw = new double[n + 1, m + 1];
   const double INF = 1e12;

   // init EL TABLE
   for (int i = 0; i <= n; i++)
      for (int j = 0; j <= m; j++)
         dtw[i, j] = INF;
   dtw[0, 0] = 0;

   // fill EL TABLEEEE
   for (int i = 1; i <= n; i++)
   {
      for (int j = 1; j <= m; j++)
      {
         double cost = FrameDistance(s.Frames[i - 1].Features,t.Frames[j - 1].Features);

         double best = dtw[i - 1, j];

         if (dtw[i, j - 1] < best)
```

```
            best = dtw[i, j - 1];
        if (dtw[i - 1, j - 1] < best)
            best = dtw[i - 1, j - 1];



        dtw[i, j] = cost + best;
      }
    }

    return dtw[n, m];
}
```

Description:

- **Purpose: compute Dynamic Time Warping distance between two sequences without pruning.**
- **Input: two MFCC sequences (Sequence s, Sequence t).**
- **Output: cumulative distance (double).**

---

2. DTW with Pruning Implementation

```
public static double ComputePruned(Sequence s, Sequence t, int W)
{
    int n = s.Frames.Length;
    int m = t.Frames.Length;
    var prev = new double[m + 1];
    var cur = new double[m + 1];
    const double INF = 1e12;
```

```
// init first row
for (int j = 0; j <= m; j++) prev[j] = INF;
prev[0] = 0;

for (int i = 1; i <= n; i++)
{
  for (int j = 0; j <= m; j++) cur[j] = INF;

  int j0;
  if (i - W < 1)
  { j0 = 1; }
  else
  { j0 = i - W;}

  /////////
  int j1;
  if (i + W > m)
  { j1 = m; }
  else
  { j1 = i + W;}

  for (int j = j0; j <= j1; j++)
  {
    double cost = FrameDistance(s.Frames[i - 1].Features,t.Frames[j - 1].Features);
```

```
        double best = prev[j];
        if (prev[j - 1] < best) best = prev[j - 1];
        if (cur[j - 1] < best) best = cur[j - 1];

        cur[j] = cost + best;
      }

      // swapwapwapwapwapwapwap rows
      var tmp = prev; prev = cur; cur = tmp;
    }

    return prev[m];
}
```

Description:

- **Pruning by window W: limits matching to a diagonal band of width W.**
- **Early abandon: stops when cumulative cost exceeds threshold.**

---

3. Enrollment & Identification Code

Database :

```
public class Template
{
    public string Name { get; set; }
    public Sequence Seq { get; set; }
    public Template(string name, Sequence seq)
    {
```

```
      Name = name; Seq = seq;
    }
  }

 // list of all enrolled templates
 List<Template> templates = new List<Template>();
```

Buttons :

```
private void button1_Click(object sender, EventArgs e)
{
   if (seq == null || templates.Count == 0)
   {
      MessageBox.Show("Nothing to match.");
      return;
   }

   double bestDist = double.MaxValue;
   string bestName = null;

   // loop through all templates
   foreach (var tpl in templates)
   {


      var sw = Stopwatch.StartNew();
```

```
        // plain DTW:
        double d = DTW.Compute(tpl.Seq, seq);
        sw.Stop();
        // or use pruning: DTW.ComputePruned(tpl.Seq, currentSeq, W);
        Console.WriteLine($"Compute plain: {sw.Elapsed.TotalSeconds:F4}s");
        if (d < bestDist)
        {
          bestDist = d;
          bestName = tpl.Name;
        }
      }

      MessageBox.Show($"Identified as: {bestName}\nDistance = {bestDist:F2}");

    }

    private void button2_Click(object sender, EventArgs e)
    {
      if (seq == null || templates.Count == 0)
      {
        MessageBox.Show("Nothing to match.");
        return;
      }

      int W = (int)numericUpDown2.Value;
```

```
    double bestDist = double.MaxValue;
    string bestName = null;

    foreach (var tpl in templates)
    {
        var sw = Stopwatch.StartNew();

        double d = DTW.ComputePruned(tpl.Seq, seq, W);
        sw.Stop();

        Console.WriteLine($"Compute plain in Pruned: {sw.Elapsed.TotalSeconds:F4}s");
        if (d < bestDist)
        {// for best so far 34an tfdl tkarn be a3la rkm
            bestDist = d;
            bestName = tpl.Name;
        }
    }

    MessageBox.Show(
        $"Identified as: {bestName}\n" +
        $"Pruned distance (W={W}): {bestDist:F2}"
    );
    //UpdateUI();
 }
private void button3_Click(object sender, EventArgs e)
{
```

```
  if (seq == null || templates.Count == 0)
  {
     MessageBox.Show("Nothing to match.");
     return;
  }

  int W = (int)numericUpDown2.Value;
 //  int t = (int)numericUpDown3.Value;
 // mmokn n3ml a7sn mn el code hna rkm kber masln

  double bestDist = double.MaxValue;
  string bestName = null;

  foreach (var tpl in templates)
  {
     var sw = Stopwatch.StartNew();
     double d = DTW.ComputePruned(tpl.Seq, seq, W,bestDist);
     sw.Stop();
     Console.WriteLine($"Compute plain in threshold:
{sw.Elapsed.TotalSeconds:F4}s");

     if (d < bestDist)
     {
        bestDist = d;
        bestName = tpl.Name;
     }
```

```
    }

    MessageBox.Show(
        $"Identified as: {bestName}\n" +
        $"Pruned distance (W={W}): {bestDist:F2}"
    );
}
```

Description:

Enrollment phase

1. **Record or load an audio sample.**
2. **Extract its MFCC feature sequence (seq).**
3. **Add a new Template(name, seq) to the templates list.**

Identification phase

1. **If seq is null or templates is empty, show "Nothing to match."**
2. **Otherwise, for each enrolled template:**
    - **Compute the DTW distance between its Seq and the current seq.**
    - **Keep track of the smallest distance and its template name.**
3. **Display the best-matching name and distance.**

---

4. Time & Space Complexity Analysis

Method	Time Complexity	Space Complexity
DTW Plain (Compute)	O(N×M)	O(N×M)
DTW Pruned (ComputePruned window W)	O(N×W)	O(W)

Notes:

- **N = length of reference sequence, M = length of test sequence.**

- **W = pruning window width.**
- **Early-abandon reduces average time when sequences diverge.**

---

5. Performance Comparison

- **Pruned DTW achieves ~3× speedup over plain DTW with no loss in accuracy.**
- **Early-abandon further improves speed (~1.5×) when thresholds applied.**

End of Documentation