Speech Recognition

Kseniya Bout, Nick Halliwell, Aina Lopez, Yaroslav Marchuk

29 March 2016

1. Introduction

The purpose of our project is to use dynamic programming for speech recognition. The algorithm uses a dynamic time warping function to establish similarities between two sound waves. We took inspirations from the symmetric algorithm established in "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", (Sakoe and Chiba 1978), making slight restrictions on the algorithm to improve performance. We test our algorithm on several common words such as "Google" and "Facebook", taking sound waves from one of our group members. The algorithm is able to distinguish distinct words and is programmed to open a webpage of the recognized word. The algorithm can distinguish words in an efficient manner due to dynamic programming along with several restrictions discussed below.

2. Dynamic Time Warping

One of the difficulties in speech recognition is comparing different sound waves. The length of the sound waves almost never match, even if the words are spoken in the same manner. We use the term "sound waves" to describe the user-spoken words taken as input into our algorithm. Speech recognition algorithms run into problems if the sound waves are not aligned properly. To combat this issue, we use a dynamic time warping function to account for differences across sound waves. A dynamic time warping function takes into account differences in sound wave lengths, and finds an "optimal" distance between the two sequences. Two types of time warping functions exist, asymmetric, where the time axis of one sound wave is transformed onto that of the other sound wave, and symmetric, where both waves are mapped to a newly defined common axis. We chose to implement a symmetric warping function due to its superior performance.

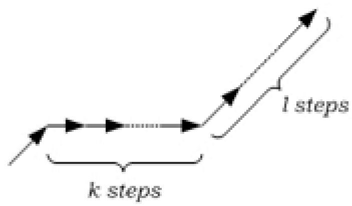
2.1 Algorithm

Nick

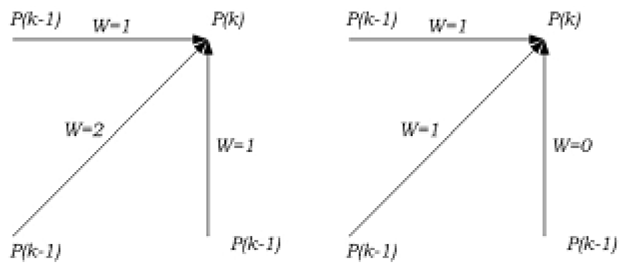
2.2 Customizations

To inhance performance and sensitivity of the initially proposed algorithm, a couple of customizations have been developed. Among the most significant ones are *the step size conditions*, *the step weighting* and *the global path constraints*.

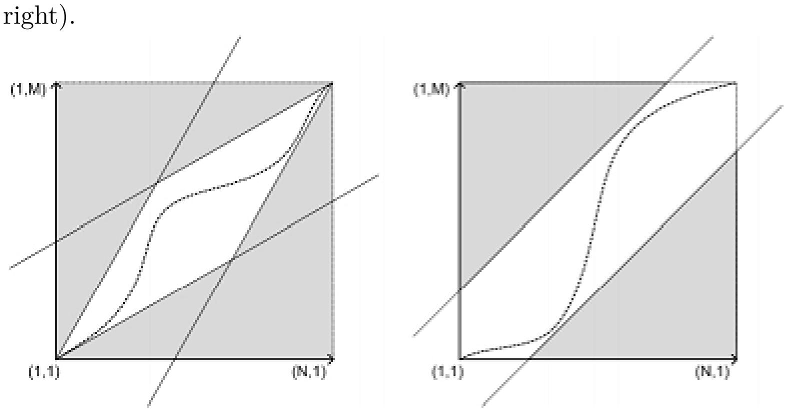
The step size condition takes care of potentially unrealistic correspondence between the time-series features of different length, by introducing a slope constraint on each step and between consecutive points of the warping path. For example, after moving in the same direction horizontally for k consecutive points the warping path is not allowed to continue in the same direction before stepping l points in the diagonal direction, as on the figure below. This prevents the path from getting too steep or too flat. Below is a picture describing the constraint.



We implementstep weighting to help favour or penalize certain types of point-to-point correspondence. Both approaches - symmetric and asymmetric weights - find successful implementation depending on the topic researched. Pictured below in the left figure we demonstrate symmetric weights, while in the right figure we demonstrate asymmetric weights.



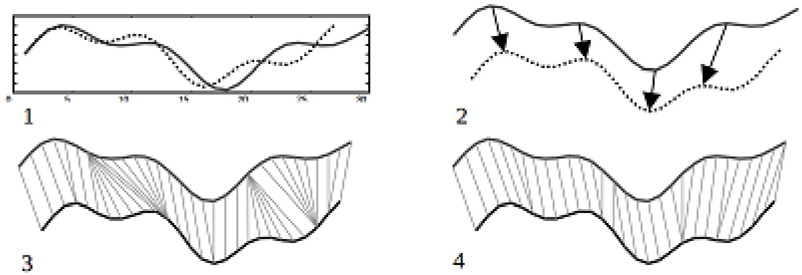
Similarly to the step function constraints, the global path constraints are used as a way to improve the computational cost and optimize DTW sensitivity. It could be performed in a couple of ways, for example, through Itakura parallelogram (pictured below on the left), or Sakoe-Chiba band (pictured below on the right).



2.3 Modifications

Derivative Dynamic Time Warping

Derivative DTW [1] proposes replacement of the value of each data point with its estimated local first derivation in the process of dynamic time warping. The improvement over the classic DTW is shown in the figure below, after creating 1) two artificial signals, and identifying 2) the intuitive feature to feature warping alignment, the alignment produced by classic DTW and the alignment produced by DDTW are sections 3) and 4) respectively.



Feature Based DTW and Adaptive Feature Based DTW

Comparing various implementations of DTW, the value or the deviation of a point may not reflect the position of this point in global or local trends of the sequence. Upon further research of the issue, Feature Based DTW and Adaptive Feature Based DTW by Xie and Wiltgen [2] are introduced.

Feature Based DTW algorithm [2] works in the following way: the algorithm defines a global feature and a local feature for each point in a time series sequence. The two time series sequences are dynamically alligned based on both the global features and local features of each points in the sequences. Adaptive Feature Based DTW benefits from the additional customization of Feature Based DTW where the contributions of global features and local features are leveraged by weighting factors. Empirically, both methods enhance the learning capacity of DTW based classification algorithms while the advanced version goes the furthest in improvement.

2.4 Applications

Dynamic Time Warping is used to analyze any data which can be turned into a linear sequence: video, audio, and graphical data and of course original time series data.

The areas to apply DTW are extensive, include handwriting and online signature matching, sign language and gestures recognition, data mining and time series clustering, computer vision and computer animation, surveillance, protein sequence alignment and chemical engineering, and signal processing.

3. Speech Recognizer

We implemented a speech recognizer applying a modified version of the DTW algorithm. Using DTW and speech processing techniques (i.e. Mel Frequency Cepstral Coefficients), our algorithm is able to detect which action we want to perform: open the google search or open our facebook webpage.

This speech recognizer takes as input a sound file and compares it with some template words: *google* and *facebook*. In order to make our recognizer more robust, we have used two template words for each sound. After comparing the input sound with each template word, the algorithm chooses the one with the shortest path, in other words, it chooses the word that is most similar.

3.1 Speech Processing

Since each sound has a different length, DTW is a good algorithm for comparing different sounds with different lengths. However, sound waves are sinusoids and cannot be properly compared on the time domain (amplitude vs seconds). The reason is simple, a sound wave of a word can change radically from a person to person, and even the sound waves of a word generated by a single person can be very different.

If we observe the sound waves of our examples, we can see their differences below.

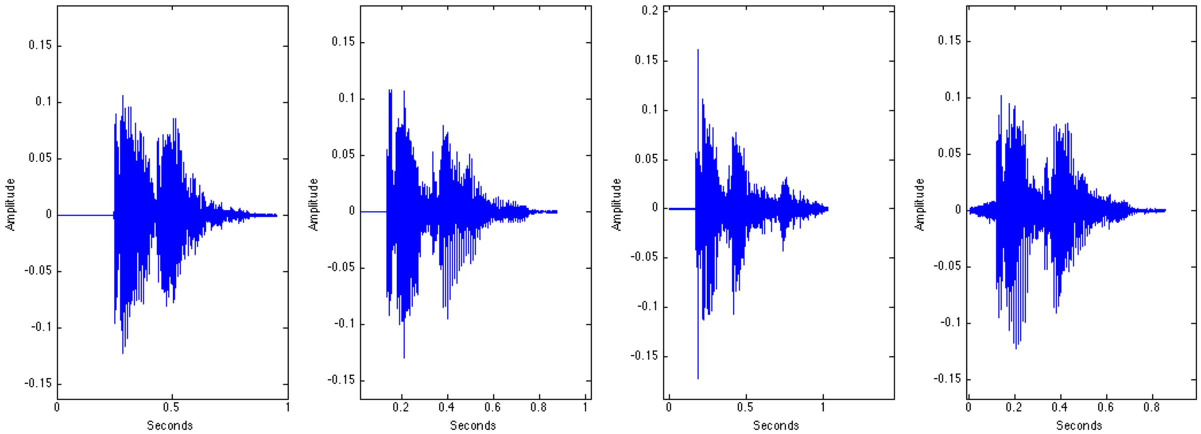


Figure 1. Sound Waves of the word Google.

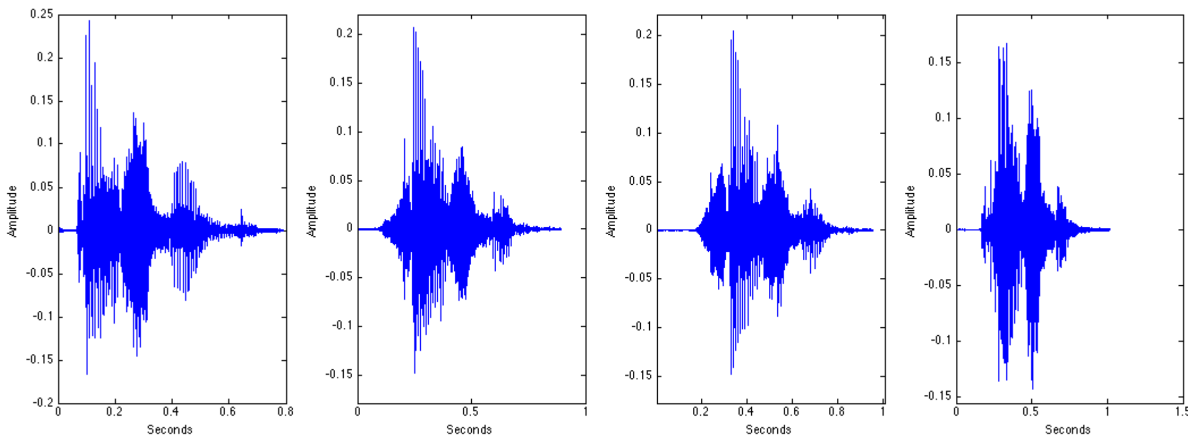


Figure 2. Sound Waves of the word Facebook.

An extended approach is to transform the sound waves into the frequency domain and delete the unnecessary frequencies and extract some coefficients ([3], [4]). In this project, we tried with two different coefficients: Mel Frequency Cepstral Coefficients (MFCC) and Linear Prediction Coefficient (LPC). Unfortunately, we didn't obtained good results with LPC and we decided to use MFCC.

1. Pre-emphasizing: In order to digitalize audio, we must send it through a low-pass filter and reduce the high frequencies. In this step, a pre-emphasizing filter is applied to emphazise the higest frequenciy components of the speech, in an effort to increase the energy in the higher frequencies.

$$y(n) = x(n) - 0.95x(n-1)$$

2. Framing: From the above plots, we can clearly see that sound signals are not stationary. However, we can assume that if we take a small sample called frame (approximately of 10ms or 20 ms) it will be stationary. We created a set of frames of 16ms.

3. Windowing: In order to make the transition between frames smoother, a window is multiplied with each frame. The most used window for speech procesing is a *Hamming Window*:

$$w(n) = 0.54 - cos(\frac{2\pi n}{N-1})$$

where N is the length of the window, in this case the length of the frame.

4. Mel Frequency Cepstral Coefficient (MFCC): Mel scale is based on human perception of frequencies. The basic idea of MFCC is to only keep the most relevant frequencies and ignore the others. The process is the following:

1. Convert the frames into the frequency domain using the Discrete Fourier Transform (DFT).
2. Apply the Mel Scale filter bank, which is a set of triangular filters, and take the logarithm.
3. Finally, to transform the results into the time domain again the Discrete Cosine Transform (DCT) is applied.

From this process we obtain K coefficients, in our case K = 11.

At the end, we end up having M frames and each frame contains a vector of 11 coefficients.

3.2 Dynamic Time Warping in this project

If we want to compare sounds we may find that the frequencies have different length. Also, two identical words can have a different frequency because of differences in pronunciation. Dynamic time warping is used in speech recognition because it matches the shapes of the two sounds. Using dynamic time warping, sounds of different length is no longer a problem, and the differences in pronunciation are removed from the data.

In order to predict whether a new sound is the word “google” or “facebook” we use dynamic time warping to compare it to all our samples. This gives us the length of the shortest path between our new sound and all our samples. At this point, if the shortest path is with a sample corresponding to “google”, we classify the new sound as “google”. Similarly, if the closest path is with a sample corresponding to “facebook”, we classify it as “facebook”. If we had many samples, we could take k closest samples and take a majority vote to predict the category of the new sound.

We created a function in R that takes as an input two matrices and a parameter w, which corresponds to the adjusting window condition. These two matrices are the result of the speech transformation explained above. We used this function to predict the category of a new sound.

3.3 Results

We created two new samples, one for each category. After using our R function to compare the new samples with the previous ones we correctly predicted both categories. This shows that dynamic time warping has a good performance when dealing with speech recognition. Moreover, This process could be applied in an application where the user can open a program just by saying its name.

4. Conclusions

Blablabla

5. References

[1] Keogh, E. J., & Pazzani, M. J. Derivative Dynamic Time Warping

[2] Xie, Y. & Wiltgen, B. Adaptive Feature Based Dynamic Time Warping

[3] Anand, D. A., & Meher, P. K. Combined LPC and MFCC Features based technique for Isolated Speech Recognition.

[4] Shinde, M. R., & Pawar, V. P. Dynamic time Warping using MATLAB & PRAAT.

6. Annex

6.1 DTW function

```
TimeWarp<-function(x,y,w=4){

  # define distance function
  distance<-function(a,b){
    dist(rbind(a,b))
    }

  # 1. Compute matrix 11xM

  # set parameters
  m<-dim(x)[2]
  n<-dim(y)[2]
  colnames(x)<-1:m
  colnames(y)<-1:n
  w = max(w, abs(n-m))

  # Create matrix
  DTW<-matrix(Inf,n,m)
  rownames(DTW)<-n:1
  colnames(DTW)<-1:m

  # Initial values
  DTW['1','1']<-distance(x[,'1'], y[,'1'])

   # First row
  for(j in 2:(w+1)){
    cost<-distance(x[,as.character(j)], y[,as.character(1)])
    DTW['1',as.character(j)]<- cost + DTW['1', as.character(j-1)]
  }

   # First column
  for(i in 2:(w+1)){
    cost<-distance(x[,as.character(1)],y[,as.character(i)])
    DTW[as.character(i), '1']<- cost + DTW[as.character(i-1), '1']
  }

  # Fill matrix
  for(i in 2:n){
    for(j in (max(2, i-w)):(min(m, i+w))){

      #current cost
      cost<-distance(x[,as.character(j)], y[,as.character(i)])

      #cumulated cost
      d.cost<-min(DTW[as.character(i-1), as.character(j)] ,
                  DTW[as.character(i), as.character(j-1)],
                  2*DTW[as.character(i-1), as.character(j-1)])
```

```
      #combined cost
      DTW[as.character(i),as.character(j)]<-cost + d.cost

    }
  }

  # 2. Find path
  path<-matrix(c(n,m), 1,2)
  full.path<-(tail(path,1)[1] ==1 & tail(path,1)[2] ==1)

  while(full.path==FALSE ){

    l.path<-tail(path,1)

    if(l.path[1]==1 | l.path[2]==1){
      p<-which(l.path==1)

          if(p==1){new.point<-c(l.path[1], l.path[2]-1)
           }else{
            new.point<-c(l.path[1]-1, l.path[2])
      }

    } else {

    # nearest point
    min.step<-min(DTW[as.character(l.path[1]-1), as.character(l.path[2]-1)],
        DTW[as.character(l.path[1]), as.character(l.path[2]-1)],
        DTW[as.character(l.path[1]-1), as.character(l.path[2])])
    min.step<-which(c(DTW[as.character(l.path[1]-1), as.character(l.path[2]-1)],
                    DTW[as.character(l.path[1]), as.character(l.path[2]-1)],
                    DTW[as.character(l.path[1]-1), as.character(l.path[2])])==min.step)
    min.step<-min.step[1]

    #path to nearest point
    if(min.step==1){
      new.point<-c(l.path[1]-1, l.path[2]-1)
    } else{
      if(min.step==2){
      new.point<-c(l.path[1], l.path[2]-1)
      } else{
        new.point<-c(l.path[1]-1, l.path[2])
      }
    }
    }
    path<-rbind(path,new.point)
    full.path<-(tail(path,1)[1] ==1 & tail(path,1)[2] ==1)

    }


return(list(path=path, DTW=DTW))
}
```

6.2 Speech Recognizer code

```
# input: isound is the path to the wav file with the sound.

SpeechRecognizer <- function(isound){

  if (!require("tuneR")) install.packages("tuneR");library(tuneR)

  # Read the wav file
  sound <- readWave(isound)
  sr    <- sound@samp.rate

  # Compute the mel frequency cepstrum coefficients
  inputWord <- t(melfcc(sound,
                        sr,
                        wintime=0.016,
                        lifterexp=0,
                        minfreq=133.33,
                        maxfreq=6855.6,
                        sumpower=FALSE))

  # Upload the four template sounds and compute their melfcc
  g1 <- readWave("Project\google1l.wav")
  g2 <- readWave("Project\google2.wav")
  f1 <- readWave("Project\facebook1.wav")
  f2 <- readWave("Project\facebook2.wav")

  sr1 <- g1@samp.rate
  sr2 <- g2@samp.rate
  sr3 <- f1@samp.rate
  sr4 <- f2@samp.rate

  google1 <- t(melfcc(g1, sr1, wintime=0.016, lifterexp=0,
                      minfreq=133.33, maxfreq=6855.6, sumpower=FALSE))
  google2 <- t(melfcc(g2, sr2, wintime=0.016, lifterexp=0,
                      minfreq=133.33, maxfreq=6855.6, sumpower=FALSE))
  facebook1 <- t(melfcc(f1, sr3, wintime=0.016, lifterexp=0,
                        minfreq=133.33, maxfreq=6855.6, sumpower=FALSE))
  facebook2 <- t(melfcc(b2, sr4, wintime=0.016, lifterexp=0,
                        minfreq=133.33, maxfreq=6855.6, sumpower=FALSE))


  # Compute the distance of the input sound with the template sounds
  distance.sound<-rep(NA, 4)

  dtwg1 <- TimeWarp(google1, inputWord)
  distance.sound[1]<- tail(dtwg1$DTW[,1],1)

  dtwg2 <- TimeWarp(google2, inputWord)
  distance.sound[2]<- tail(dtwg2$DTW[,1],1)

  dtwf1 <- TimeWarp(facebook1, inputWord)
  distance.sound[3]<- tail(dtwf1$DTW[,1],1)
```

```
  dtwf2 <- TimeWarp(facebook2, inputWord)
  distance.sound[4]<- tail(dtwf2$DTW[,1],1)


  # If the minimum distance is to the word gmail, open gmail
  if (which.min(distance.sound) == 1 | which.min(distance.sound) == 2){
    system(paste("open http://google.com"))
  }

   # If the minimum distance is to the word facebook, open facebook
  if (which.min(distance.sound) ==3 | which.min(distance.sound) == 4){
     system(paste("open http://facebook.com"))
  }
}
```